The DoCookBook

Recipes for DocBook Developers

Thomas Schraitle

The DoCookBook: Recipes for DocBook Developers

Thomas Schraitle
Proofreading, suggestions, and ideas: Antje Faber

0.8.0

Publication date 26 Sep 2018, 20:46:48+02:00

Licensed under Creative Commons

Table of Contents

Welcome!

Mission

> *The DBCookBook Project [https://doccookbook.sf.net]* aims to create an open source book about DocBook [http://www.docbook.org/] and the DocBook XSL stylesheets [http://docbook.sf.net/release/xsl/current/doc/] written as cookbook and released under Creative Commons license.

According to its homepage [http://www.docbook.org/whatis], "DocBook is a schema (available in several languages including RELAX NG, SGML and XML DTDs, and W3C XML Schema) maintained by the DocBook Technical Committee of OASIS. It is particularly well suited to books and papers about computer hardware and software (though it is by no means limited to these applications)."

Although there are helpful and valuable documentation#be it as a printed book[1] or online[2]#I always felt there is a gap. This gap could be bridged with a guide that only concentrates on problems and their solution in a concise manner explained by step-by-step procedures. Until now nobody wrote a book that combines possible problems, pitfalls, "how to do this", and the benefits in a cookbook style. My book is intented to be such a book. However, it *does not* seek to replace any other information, be it printed or online. It aims for *enhancing* the world of DocBook with a different view.

The topics in this book are mostly based on my own experiences with DocBook and the DocBook XSL stylesheets. However, it also includes questions and answers from other sources: from the DocBook mailinglists, other sites that inspired me, or questions from collegues. It is the nature of such topics that there might be some overlap with other sources. When other authors inspired me, I have included their names. If you think your name is missing, please let me know.

As I have learned (and still do) a lot from the masters of DocBook, it is time to give something back. What you are reading is the first draft of the fruits of these efforts. Currently, this book is available under a Creative Commons license (see Licensed under Creative Commons [ii] if you want to know more). I hope you enjoy reading this book as much as I enjoy writing it.

[1] Like the reference *DocBook 5: The Definitive Guide [http://www.docbook.org/tdg51/en/html/docbook.html]* from Norman Walsh or Bob Stayton's *DocBook XSL [http://www.sagehill.net/docbookxsl/]*

[2] Dave Pawson maintains a DocBook FAQ [http://www.dpawson.co.uk/] and the previous mentioned books can be found online too.

1. Audience

This book covers topics which range from very easy to pretty demanding. The more challenging a problems is the more you need experience from different domains (CSS, FO, XSLT, etc.) As such, this book does not and cannot attempt to teach you DocBook or any such related technology (with the exception of Chapter 1, *Knowing DocBook's Structure* [1]). If you want to learn DocBook or need some help with XML or XSLT, search elsewhere. There are plenty of information available either online or printed.

This book is aimed at DocBook users who write *and* seek answers for their customization problems. However, with some try-and-error, it can also be useful for ordinary writers who want to try something out. For readers who want further information, links are included in the *See Also* section at the end of each topic.

2. DocBook 4 or 5?

Although a lot of these topics can be used for both versions with minimal changes, this book focuses on DocBook 5. Why? There are several reasons:

- DocBook 4 is in maintenance mode and DocBook 5 is the future.
- I wanted to show exhaustive examples for DocBook 5. With this book, I hope, users can learn how to adopt and use it.
- It is easy to transform DocBook 4 into version 5 as shown in Section 3.3, “Converting DocBook from Version 4 to Version 5” [63].
- It simplifies writing *a lot* if you can concentrate on only one version and not on two. It is also easier to read for you.

The differences are documented in the *DocBook5: The Definitive Guide* book, section *What’s New in DocBook V5.0? [http://www.docbook.org/tdg5/en/html/ch01.html#introduction-whats-new]*.

Apart from the DocBook examples, the transformation examples are currently based on the DocBook XSL 1.0 stylesheets. They are stable, well-documented and can be used with any XSLT 1.0 processor.

On the other side, there is always the tension between implementing new concepts and retaining backward compatibility. For this reason, the XSLT 1.0 stylesheets are rewritten for XSLT 2.0. This task is not finished yet. As soon as they are production-stable, they will be covered in this book too.

3. Further Readings

By no means is this book complete. As this book is based on technologies like CSS, XML, (X)HTML, and XSLT, it is always a good idea to seek other information. I highly recommend to read the books by Norman Walsh and Bob Stayton as mentioned before.

4. Typographical Conventions

The following typographical conventions are used in this book:

- *Italic* is used for definition of new terms or emphasized text.
- `Monospace text` is used for code listings, XML structures, URLs, filenames, namespaces, or links.
- **Monospace bold text** for commands or emphasized text in listings.
- *`CAPITAL MONOSPACE`* is used for placeholders in listings.

5. Feedback

The project is hosted on GitHub [https://github.com]® under the name tomschr/dbcookbook [https://github.com/tomschr/dbcookbook]. Or write me directly to <docbook-xml@online.ms>. Feel free to send me praise, constructive criticism, suggestions, ideas, improvements, or anything else.

6. Contribute to the Book

The source code of this book is published and maintained as a Git [https://git-scm.com] repository. Feel free to clone it, make improvements and mail me your changes to <docbook-xml@online.ms> or send me pull requests..

Getting a copy takes just a few seconds. If you have Git already installed on your system, run the following command:

```
git clone git@github.com:tomschr/dbcookbook.git
```

Git is a free, distributed version control system and available for Mac, Linux, and Windows. Download it from https://git-scm.com/downloads.

If you do not want to change the source code, but want to contribute, open a ticket. Describe your bugs, grammar errors, typos, or other inconveniences you have found and add them to the list at https://github.com/tomschr/dbcookbook/issues/new.

7. Contributors

Thanks to all who contribute to this book (sorted alphabetically):

- Antje Faber
- Stefan Hinz
- Peter Schmelzer

8. Licensed under Creative Commons

9. Acknowledgments

Such a large project is usually not a one man show. Many people contributed to this book:

- Antje Faber for proofreading some of my texts and the original idea.
- Stefan Hinz for providing me with valuable feedback and improvements.
- Lars Vogel for his suggestions and improvements.
- Norman Walsh for DocBook, the DocBook stylesheets, and all the nice tools that makes live with DocBook so much easier.
- Matthias Weckbecker for helping me with some wording and ideas.

Chapter 1. Knowing DocBook’s Structure

Abstract

DocBook has a wide variety of elements covering different semantics: from structural elements like `book`, `chapter`, `appendix`, to block elements like `orderedlist`, `table`, `figure`, to inline elements like `filename`, `quote`, or `link`. All of these elements are well covered in *The Definitive Guide [http://www.docbook.org/tdg51/en/html/docbook.html]*. This chapter shows more the unknown parts and how to avoid pitfalls and improve consistency throughout your documentation.

1.1. Introduction

To successfully use DocBook you basically need to know how to “transform” your ideas into DocBook’s structure. Usually, the DocBook elements are mostly self-explanatory: if you want to write a book, use `book`, for a chapter use `chapter` and so on. All you need is covered in the *Definitive Guide [http://www.docbook.org/tdg51/en/html/docbook.html]* with examples and the reference of all elements.

However, a reference usually do not cover the finer details. For example, should I use `sect1` or `section`? How do I modularize my documentation? What is the recommendation to use multiple indices?

1.2. Selecting the Right Top Level Element

Problem

You want to know what root elements are possible and what else is needed to create a DocBook V5 document.

Solution

A document type definition (DTD) allows *every* definied element to become a root element. A DTD cannot create any constraints to limit this set. On the other hand, RELAX NG -- the official schema language for DocBook 5 and up -- can impose such constraints.

To create a valid DocBook V5 document you need:

- A valid DocBook V5 element to begin your document.
- An attribute `version` containing the used DocBook version.
- The DocBook 5 namespace, possibly other namespaces too.

To assemble all the above hints, a valid DocBook 5 document could look like this:

Example 1.1. A Valid DocBook 5 Book

```
<book version="5.0" xmlns="http://docbook.org/ns/docbook">
  <title>The Example Book</title>
  <chapter>
     <title>An Example Chapter</title>
     <para>This is a paragraph.</para>
  </chapter>
</book>
```

Discussion

There a many more possible root elements, not only `book`. Using the list from the last section, a valid DocBook V5 element to start with can be:

```
acknowledgements    dedication          reference           sect3
appendix            glossary            refsect1            sect4
article             index               refsect2            sect5
bibliography        para                refsect3            section
book                part                refsection          set
chapter             preface             sect1               setindex
colophon            refentry            sect2               toc
```

For the next DocBook release, V5.1, the technical committee plans to include all elements that can contain an info wrapper.

Regarding namespaces, DocBook 5 uses only one namespace (which is http://docbook.org/ns/docbook). However, sometimes it is needed to include other in order to adhere to the XML specification.

A namespace declaration consists of an (optional) namespace prefix and a namespace URI. The prefix can be selected freely, however, it is recommended to use the prefixes in Table 1.1, “Common Prefixes and Their Namespaces” [3]. Although they are not a “standard” in its strict sense, but over time they have been extensively used.

Table 1.1. Common Prefixes and Their Namespaces

Prefix	Namespace
d[a], db	http://docbook.org/ns/docbook
fo	http://www.w3.org/1999/XSL/Format

Prefix	Namespace
h	http://www.w3.org/1999/xhtml
mml	http://www.w3.org/1998/Math/MathML
rng	http://relaxng.org/ns/structure/1.0
svg	http://www.w3.org/2000/svg
xi	http://www.w3.org/2001/XInclude
xlink	http://www.w3.org/1999/xlink
xsd	http://www.w3.org/2001/XMLSchema
xsl	http://www.w3.org/1999/XSL/Transform

[a] In this book the DocBook prefix is omitted and the standard namespace is used.

Usually it is a good idea to insert all the namespaces used in the document in the root element. For example, Example 1.2, “Start Tag with Several Namespace Declarations” [4], shows three namespace declarations: the DocBook namespace, using no prefix; the XInclude namespace using the xi prefix, and the XLink namespace using the xlink prefix.

Example 1.2. Start Tag with Several Namespace Declarations

```
<book version="5.0"
   xmlns="http://docbook.org/ns/docbook"
   xmlns:xi="http://www.w3.org/2001/XInclude"
   xmlns:xlink="http://www.w3.org/1999/xlink">
```

1.3. Modularize Your Document with XIncludes

Problem

You need a method to split your document into several “modules” and put it together afterwards.

Solution

Use *XInclude*. It is a W3C specification and defines the elements `xi:include` and `xi:fallback`. They are not DocBook elements (as they are definied by the W3C and not OASIS), however they have been integrated in version 5.x[1]. Note, XIncludes work in DocBook regardless which version (4.x or 5.x) you use.

If you want to use XIncludes, you need these things:

- A XML parser who supports XIncludes.
- The XInclude namespace `http://www.w3.org/2001/XInclude`, usually bound to the xi prefix.
- The element `xi:include`. In general, it can be placed everywhere where you can place DocBook elements. It is a placeholder for the real content and works as a reference.
- The attribute `href` inside the `<xi:include>` start tag. It is an URI which refers to your included file.
- The file which is referenced by the `href` attribute. The content of the file will replace the `xi:include` element. It contains usually a DocBook element with its content.

The following example shows a book which points to a substructure assuming chapters:

Example 1.3. A Book with XIncluded Chapters

```
<book version="5.0"
  xmlns="http://docbook.org/ns/docbook"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <title>...</title>
  <xi:include href="intro.xml"/>
  <xi:include href="conceptual-overview.xml"/>
</book>
```

The above book contains an introduction (file `intro.xml`) and a conceptual overview (file `conceptual-overview.xml`). Both are referenced by the XInclude´s `href` attribute.

Before you transform your document, you need to resolve your XIncludes first, either by your XML parser or “manually” by a XSLT transformation. The following procedure shows a typical workflow:

Procedure 1.1. Typical Workflow with XIncludes

1. Write your document structure, usually it will be a book or an article. Do not forget to include the XInclude namespace `http://www.w3.org/2001/XInclude` into the root element, commonly bound to the prefix xi.
2. Add `xi:include` elements for those content you want to maintain in a separate file. Typically, this can be an appendix, chapter, preface, glossary, or any other DocBook element.
3. Resolve your XIncludes. Use a XML parser who supports XIncludes, for example, **xmllint** from the libxml2 library [http://xmlsoft.org]. This XML parser brings the `--xinclude` option to resolve all your XInclude elements in one step:

   ```
   xmllint --xinclude --output big.xml book.xml
   ```

   The above command resolves all XIncludes and saves the result in the file `big.xml`. Note, this does not perform any validation! It just replaces `xi:include` with the content of the referenced file. After the XInclude elements are resolved, the file looks now like this:

[1] To use XInclude with DocBook 5.x, use the `docbookxi.rnc` RELAX NG schema.

```
<book version="5.0"
  xmlns="http://docbook.org/ns/docbook"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <title>...</title>
  <chapter>
    <title>Introduction</title>
    <para>...</para>
  </chapter>
  <chapter>
   <title>Conceptual Overview</title>
   <para>...</para>
  </chapter>
</book>
```

4. Validate the result (in our example, it is `big.xml`) with your DocBook schema.

5. Transform the result file with your stylesheets into your target format.

Discussion

The previous procedure showed a book with xincluded chapters. It is possible to even go deeper and include also section into a chapter. Actually, there is no limit. You should only be aware that you do not create circular references (file A includes file B and B includes A).

As XIncludes are very common nowadays, resolving `xi:include` and transforming into the output format can be done in one step. This is the case for most tools:

xmllint from the libxslt library
: Use the `--xinclude` option as shown:

```
xsltproc --xinclude STYLESHEET XMLFILE
```

Saxon 6
: Unfortunately, Saxon 6 needs some more configuration. Most Linux distribution have already a **saxon6** command. However, it can be difficult to customize it correctly, so this is the line you need:

```
java -Dorg.apache.xerces.xni.parser.XMLParserConfiguration=org.apache.xerces.parsers.XIncludd
   JARPATH/saxon6.jar:JARPATH/xml-commons-apis.jar:JARPATH/jaxp_parser_impl.jar:JARPATH/xml-
   com.icl.saxon.StyleSheet \
   -x org.apache.xml.resolver.tools.ResolvingXMLReader \
   -y org.apache.xml.resolver.tools.ResolvingXMLReader \
   -r org.apache.xml.resolver.tools.CatalogResolver\
   ARGS
```

  The line contains different properties:

  - The org.apache.xerces.xni.parser.XMLParserConfiguration property sets the XInclude processor which is done by Xerces in this case.

  - The *JARPATH* is the path to your JAR files. In most FHS conformant Linux distributions, nowadays this is usually `/usr/share/java`.

  - Additionally, with the `xml-commons-resolver.jar` file, Saxon 6 is able to resolve catalogs. To tell Saxon you need to set the `-r`, `-x`, and `-y` options the a URI resolver class, and the specified Sax parser for source file and stylesheet.

  - The *ARGS* are the specific arguments for Saxon and contains source document and stylesheet. To list all available options, use `-h`.

Saxon 9
: Version 9 contains the `-xi` option to resolve XIncludes (assuming you have a script **saxon9** which does all the heavy Java lifting):

```
saxon9 -xi -xsl:STYLESHEET -s:XMLFILE
```

The last section showed a general method to work with XIncludes. In most cases this is enough. However, XIncludes offers more benefits which are discovered in the following subsections.

Fallbacks

If the referenced file in the `xi:include` element is not available, the XInclude step will fail. How can you avoid that? The XInclude specification defines also the `xi:fallback` element. This element can be used to add code when a referenced resource could not be retrieved:

Example 1.4. Fallback Possibility with `xi:fallback`

```
<xi:include href="revhistory.xml">
  <xi:fallback>
    <para>The revision history could not be retrieved.</para>
  </xi:fallback>
</xi:include>
```

The previous code does the following: When the `xi:include` element is being processed, the XML parser tries to include the file `revhistory.xml`. If the file can not be retrieved, the XML parser will consider the `xi:fallback` element and include the contents. In the above case it includes a `para` element showing the failed attempt.

This method is useful when you want to process files which cannot be expected to always be there. For example, the previous revision history needs to be generated first. However, it is not always sure that the revision history can be generated from an possible offline version control system.

Including Text

The previous examples dealt with included resources in XML only. If you need to include text, this can also be done with XInclude.

Most common usecase is including source code which is maintained separately. The following example points to a C source code which needs to be included as text:

Example 1.5. Included Text in a Programlisting

```
<programlisting language="c"><xi:include
   parse="text"
   href="parser.c"/></programlisting>
```

The important line is `parse="text"`. This advises the XInclude processor to handle the referenced file as text and not as XML. The default value for `parse` is `xml`.

It is recommended to remove any whitespaces inside `programlisting` as shown above to avoid spurious indendation or linebreaks.

More explanations can be found in Section 1.8, “Incorporating External Files in Code Listings” [16].

See Also

- Section 1.8, “Incorporating External Files in Code Listings” [16]
- XML Inclusions (XInclude) Version 1.0 [http://www.w3.org/TR/xinclude/]
- Using XInclude [http://www.sagehill.net/docbookxsl/ModularDoc.html#UsingXinclude]

1.4. Mastering Multiple Indices

Problem

You need in your document two separate indices, a general one and another one for persons.

Solution

Use the `type` attribute both on `indexterm` and `index`. The content of the `type` attribute can be any value. Usually it is a descriptive name (in our case, “persons” would be fine).

For example, the following paragraph contains two indexterms. However, Guido is marked up as a “person's index”:

```
<para> The Python<indexterm>
   <primary>Python</primary></indexterm> programming
   language was designed by
   Guido van Rossum.<indexterm type="persons">
   <primary>van Rossum, Guido</primary></indexterm>
</para>
```

Use two `index` element at the end of your document to distinguish between the two:

```
<index/> 1

<index type="persons"> 2
  <title>Persons</title>
</index>
```

1 Generates the “normal” index as it has no `type` attribute.
2 Collects only those index entries which contains a `type` attribute with the value `persons` (as with Guido in the above example).

Discussion

Some documents contain two indices, a general and one for persons. This is quite common thesedays as it simplifies where to find your search term. You can add as many indices as you need, but more than two are normally uncommon.

By default, multiple indices are turned on by the DocBook stylesheets. So no special customization are needed. However, if you want to collect all index terms in one index, set the parameter *`index.on.type [http://docbook.sf.net/release/xsl/current/doc/html/index.on.type.html]`* to `0` (zero). In that case, the `type` attribute has no effect.

See Also

- Section 3.11, “Adding Index Entries (Semi-)Automatically” [96]

1.5. Distinguish Between Section and SectX Elements

Problem

You need to know what the differences and advantages are of using a `section` over a `sectX` element.

Solution

The differences between `section` against `sectX` are depicted in Table 1.2, “Comparison Between `section` and `sectX` Elements” [9]:

Table 1.2. Comparison Between `section` and `sectX` Elements

Issue	`section`	`sectX`
Nesting Level?	Undefinied[a]	Up to five levels (`sect5`)
Readability?	Worse	Better
Easy Relocation?	Yes	No, needs to be renamed
Renaming after relocation?	No	Yes

[a] See discussion.

Discussion

Apart from technical issues, using `section` or `sect1` is in most cases a matter of taste. However, knowing some of the differences in Table 1.2 [9] can be helpful to decide which is for your document the better element.

Nesting Level If you have documents which goes further than level five, use the `section` element. This can be indefinitely nested. On the other side, if you want to restrict the level up to five, better use the `sectX` elements. You can, however, use DocBook's RELAX NG schema to customize it further (see next section).

Readability This issue depends on how you edit your DocBook source code. If you edit it manually in your editor (in other words: without the aid of some kind of “WYSYWYG”), you can see and understand the level straight from the tag name when using `sectX`. Especially if you have lots of nested section this makes it a lot easier than `section`. However, modern XML editors can show the XPath where you are, so it may be negligible.

Relocation If you have to change the structure a lot, using `section` elements make it a lot easier. When you use a `sect2` element and this should become a `sect1` element you have to rename it.

See Also

http://www.docbook.org/tdg51/en/html/ch05.html#ex.limitsdepth

1.6. Using Lists

Problem

You want to know what list types are available in DocBook.

Solution

DocBook provides three basic lists which are used most often:

- `itemizedlist`, for unordered lists (like `ul` in HTML)
- `orderedlist` for numbered lists (like `ol` in HTML)
- `variablelist`, for lists containing terms and their definition (like `dl` in HTML)

Additionally, for specific purposes, DocBook provides special lists (not explained in this topic):

- `bibliolist` is a wrapper for bibliographic content. A `bibliolist` is usually inserted in section-like elements where a `bibliography` is not allowed.
- `calloutlist`, a usually numbered list which points to a line in a listing
- `glosslist` is a wrapper for glossary content. A `glosslist` is usually inserted in section-like elements where a `glossary` is not allowed.
- `segmentedlist` is a list which can be used which have a one-to-one dependence with its title. A `segmentedlist` can be formatted in a number of ways (tabular or as a list block).
- `simplelist` is an unordered list for single words or small phrases.

The previous lists are used to maintain semantic distinction.

Discussion

The list types `orderedlist` and `itemizedlist` are structural identical (except for its initial element) as you will see in the following subsections.

The `variablelist` is slightly different and contains the `term` element for its term.

Features of `itemizedlist`

An `itemizedlist` is a unordered list and can be written as shown in the following example:

Example 1.6. Unordered List with Three Entries

```
<itemizedlist>
  <listitem>
    <para>The first entry</para>
  </listitem>
  <listitem>
    <para>The second entry</para>
  </listitem>
  <listitem>
    <para>The third entry</para>
  </listitem>
</itemizedlist>
```

The `itemizedlist` list in DocBook uses by default a disc symbol (• solid circle). If you nest an `itemizedlist`, it will first start the disc (• solid circle) and uses a circle (# open circle) and finally a square (# solid square). If you nest your list deeper, the sequence is repeated. The hierarchy looks like this:

Level	Shows as
1.	• • •
2.	# # #
3.	■ ■ ■

The default bullet symbol can be overwritten with the `mark` attribute:

```
<itemizedlist mark="circle">
```

It is even possible to deviate from its default symbol in a `listitem` by using the `overwrite` attribute:

```
<itemizedlist>
  <listitem>
    <para>The first entry</para>
  </listitem>
  <listitem overwrite="square">
    <para>The second entry</para>
  </listitem>
  <listitem>
    <para>The third entry</para>
  </listitem>
</itemizedlist>
```

Features of `orderedlist`

An `orderedlist` can be written as shown in the following example:

Example 1.7. Numbered List with Three Entries

```
<orderedlist>
  <listitem>
    <para>The first entry</para>
  </listitem>
  <listitem>
    <para>The second entry</para>
  </listitem>
  <listitem>
    <para>The third entry</para>
  </listitem>
</orderedlist>
```

The `orderedlist` in DocBook numbers as arabic numbers by default. If you nest an `orderedlist`, it will be automatically numbered as shown in the following sequence:

Level	Shows as
1.	1, 2, 3
2.	a, b, c
3.	i, ii, iii
4.	A, B, C
5.	I, II, III

The numbering style in an `orderedlist` can be changed with the `numberation` attribute. Allowed values are arabic, loweralpha, lowerroman, upperalpha, upperroman, or arabicindic.

Features of `variablelist`

An `variablelist`, despite its confusing name, holds all sort of terms and their definition. It is not limited to variables only. The following example shows how to use a `variablelist`.

Example 1.8. Definition List with Two Entries

```
<variablelist>
  <varlistentry>
    <term>Hamburg</term>
    <listitem>
       <para>Town in the northern part of Germany</para>
    </listitem>
  </varlistentry>
  <varlistentry>
    <term>Nueremberg</term>
    <listitem>
       <para>Town in the south-eastern part of Germany</para>
    </listitem>
  </varlistentry>
</variablelist>
```

A `varlistentry` can contain more than one term. How the terms are displayed depends usually on the output formats. Each `term` can be printed on a separate line or on the same line separated by commas.

See Also

External Links

- http://www.docbook.org/tdg5/en/html/bibliolist.html
- http://www.docbook.org/tdg5/en/html/calloutlist.html
- http://www.docbook.org/tdg5/en/html/glosslist.html
- http://www.docbook.org/tdg5/en/html/itemizedlist.html
- http://www.docbook.org/tdg5/en/html/orderedlist.html
- http://www.docbook.org/tdg5/en/html/segmentedlist.html
- http://www.docbook.org/tdg5/en/html/simplelist.html
- http://www.sagehill.net/docbookxsl/Lists.html

1.7. Incorporating Code Listings

Problem

You need to add an element which is used for code listings.

Solution

Use the `programlisting` element. For example, the following markup shows some Python code:

```
<programlisting>#!/usr/bin/python

def sayhello(name):
  """Say hello to someone"""
  return "Hello %s" % name

sayhello("Tux")</programlisting>
```

If you want to benefit from syntax highlighting later, add the `language` attribute:

```
<programlisting language="python">...</programlisting>
```

Discussion

In most cases, `programlisting` is probably enough for all your needs. However, the DocBook stylesheets and the DocBook extension supports some convenient features you should know:

- Annotated lines for short comments
- Callouts, for more exhaustive explanations
- Line numbering, if refering to a number is enough
- External code files, if you want to incorporate files from outside your DocBook document

The above features are explained in the following subsections.

Using Line Annotatations

Annotated lines are short comments created by `lineannotation`:

```
<programlisting>#!/usr/bin/python <lineannotation>She-bang line</lineannotation>

def sayhello(name):
  """Say hello to someone""" <lineannotation>Python's doc comment</lineannotation>
  return "Hello %s" % name

sayhello("Tux")
<programlisting>
```

Line annotations are printed in italic by default:

```
#!/usr/bin/python Shebang line

def sayhello(name):
  """Say hello to someone""" Python's doc comment
  return "Hello %s" % name

sayhello("Tux")
```

They are only useful, if you have short comments. If you need further explanations, callouts are better suited for this task.

Problems can arise when a reader wants to copy and paste text containing line annotations. Such text usually creates syntax problems. In this case, use a comment symbol for the appropriate programming language. For Python, it is the hash sign which can be combined in a `programlisting` context as follows:

```
<programlisting> ...
# <lineannotation>Function to output a warm welcome message</lineannotation>
def sayhello(name):
  ...
</programlisting>
```

With this method, an annotated line does not create any syntax errors.

Using Callouts

toms, 2012-09-02: Refer to its own topic

Callouts are little marks in your code which refers to a more exhaustive explanation, usually *after* the code. Such an explanation is marked up as a `calloutlist`:

```
<programlisting language="python"
>#!/usr/bin/python <co xml:id="co.shebang"/>

def sayhello(name):
  """Say hello to someone""" <co xml:id="co.doccomment"/>
  return "Hello %s" % name

sayhello("Tux")</programlisting>
<calloutlist>
   <callout arearefs="co.shebang">
     <para>A Shebang line. Unix-like operating systems can parse
           this line, execute the corresponding interpreter and pass
           the script as first argument.</para>
   </callout>
   <callout arearefs="co.doccomment">
     <para>A Python doc comment. Usually, written as triple quotes
           or as triple apostrophs marks the doc comment for a function
           or class.</para>
   </callout>
</calloutlist>
```

The result will look like this:

```
#!/usr/bin/python 1

def sayhello(name):
  """Say hello to someone""" 2
  return "Hello %s" % name

sayhello("Tux")
```

1 A Shebang line. Unix-like operating systems can parse this line, execute the corresponding interpreter and pass the script as first argument.

2 A Python doc comment. Usually, written as triple quotes or as triple apostrophs marks the doc comment for a function or class.

Callouts are a good way to give your reader more explanations. Find more information in *TODO: Add xref*.

Setting Line Numbering

Numbering program listings can be done with the attribute `linenumbering`:

```
<programlisting linenumbering="numbered">...
```

The result can look like this:

```
 1 | #!/usr/bin/python
 2 |
 3 | def sayhello(name):
 4 | """Say hello to someone"""
 5 | return "Hello %s" % name
 6 |
 7 | sayhello("Tux")
```

To influence the numbering, DocBook and the DocBook stylesheets has several ways:

`startinglinenumber`
: This attribute in `<programlisting>` specifies the initial line number.

`continuation`
: This attribute in `<programlisting>` determines whether line numbering continues from the previous element or restarts. Possible values are `continues` or `restarts` (default).

`<?dbhtml linenumbering.everyNth="N"?>`, `<?dbfo linenumbering.everyNth="N"?>`
: This processing instruction (PI) specifies interval for line numbers. By default, every 5th line displays its number.

`<?dbhtml linenumbering.separator="text"?>`, `<?dbfo linenumbering.separator="text"?>`
: This PI specifies separator text for line numbers. By default it is a single space. This text is printed after the line number but before the line from the program listing.

`<?dbhtml linenumbering.width="width"?>`, `<?dbfo linenumbering.width="width"?>`
: This PI specifies width for line numbers. By default, this value is set to three which gives you a reserved space of numbers up to 999.

`linenumbering.extension [http://docbook.sourceforge.net/release/xsl/current/doc/html/linenumbering.extension.html]`
: Parameter which enables the line numbering extension

To use the processing instructions, add them in `programlisting` without any whitespace:

```
<programlisting><?dbhtml
   linenumbering.everyNth="2"
   linenumbering.width="2" ?>#!/usr/bin/python
...
</programlisting>
```

This will create the following output:

```
  | #!/usr/bin/python
 2|
  | def sayhello(name):
 4| """Say hello to someone"""
  | return "Hello %s" % name
 6|
  | sayhello("Tux")
```

Incorporating External Files

To incorporate external files is explained in a separate topic. Refer to Section 1.8, “Incorporating External Files in Code Listings” [16].

See Also

- http://www.sagehill.net/docbookxsl/ProgramListings.html
- Section 1.8, “Incorporating External Files in Code Listings” [16]

1.8. Incorporating External Files in Code Listings

Problem

You have external files (codes, example files, and others) that you want to incorporate into a `programlisting` element, but you want to keep your files separate from your documentation files.

Solution

Use one of the following methods to incorporate external files into your `programlisting` element:

- `textobject`/`textdata`
- `xi:include` (XIncludes)

You can mix both, but for consistency reasons, it is better to stick to one method.

Incorporating External Files with `textdata`

Usually, `textdata` is included inside a `textobject` which is then included inside a `programlisting` element.

To refer to a file `foo.py` with `textdata`, use the code in Example 1.9, "External File Markuped up with `textdata`" [16].

Example 1.9. External File Markuped up with `textdata`

```
<programlisting><textobject>
   <textdata fileref="foo.py"/>
</textobject></programlisting>
```

Incorporating External Files with XIncludes

The same code in Example 1.9, "External File Markuped up with `textdata`" [16] can be expressed with XInclude:

Example 1.10. External File Markuped up with `xi:include`

```
<programlisting><xi:include
  href="foo.py" parse="text"
/><programlisting>
```

Do not forget to declare the namespace URI http://www.w3.org/2001/XInclude. This can be done either on the `xi:include` element itself, or in the root element of your documentation. Preferably, it is easier to declare it once inside the root element.

Discussion

Keeping your external files separate from your documentation can have several reasons. Probably you need to change the files very often or you do not have control over these files. In such cases, it is better to leave them as separate files and don't include them directly into your XML code. Otherwise it would be tedious to include them manually whenever your code changes.

To avoid such tedious tasks, the `textdata` and `xi:include` elements were invented. The element `textdata` was introduced in DocBook 4.2, released in 2002. At that time the XInclude specification was not written yet, the first public release was published in 2006. Too late for DocBook to incorporate it into its schema.

Nowadays, in the light of XInclude, the element `textdata` can be considered as obsolete—especially for DocBook 5 documents. Therefor, for new documents, prefer the XInclude element(s) as this has more advantages (see Table 1.3, "Comparision Between `textdata` and `xi:include`" [17]).

Table 1.3. Comparision Between `textdata` and `xi:include`

Topic	`textdata`	`xi:include`
Needs DocBook extension?	Yes	No
Can be used elsewhere in the document?	No	Yes
Can include text?	Yes	Yes
Can include XML markup?	No	Yes
Can use fallback mechanism?	No	Yes
Included in DocBook 4?	Yes	No[a]
Included in DocBook 5?	Yes	Yes[b]

[a] Not included in the DTD, but can be used anyway.

[b] Only available in the separate files `docbookxi.rn{c,g}`

If you still want (or have) to use `textdata` you need to transform your DocBook documents with Saxon or Xalan. Currently, the extension functions are only available for these XSLT processor, not xsltproc. Make sure the DocBook extension JAR file(s) are included into your `CLASSPATH` variable. Furthermore, set the parameters *`use.extensions`* and *`textinsert.extension`* to the value of `1` to enable this functionality.

See Also

- http://www.sagehill.net/docbookxsl/ProgramListings.html
- http://www.docbook.org/tdg51/en/html/textdata.html

1.9. Inserting Remarks

Problem

You want to insert comments or remarks in your document to enable or disable them in the output format.

Solution

For this purpose, DocBook provides a `remark [http://www.docbook.org/tdg51/en/html/tag.html]` element. The following example shows how to use it:

Example 1.11. A Remark Inside a Paragraph

```
<para>
This is a small text.<remark>Add more content</remark>
</para>
```

Discussion

To add comments in your document, there are two methods:

- Use XML comments
- Use DocBook's `remark` element

XML comments can be placed almost everywhere in your XML document. It is a very common method to give yourself or others some hints. This can be done with XML comments like in this example:

```
<para>
This is a text <!-- Ohh, we should add more -->
</para>
```

However, XML comments has one drawback: They are suppressed during transformation. As such, they are unavailable in your output format and can not be displayed anymore. Usually, you *do not* want them to appear in your output.

On the contrary, by using the `remark` element, comments can be shown or suppressed whenever you want them. For example, if you write your document and put some remarks, you can show them to your contributors for review. After the document is ready to be published, leave the remark as they are but suppress the remark transformation.

To display or suppress your remarks, use the parameter *`show.comments`*. This parameter can appear either on the command line of your XSLT processor or in a customization layer. Any non-zero value shows your remarks whereas a value of zero suppress them.

When should you use XML comments and when remarks? The following table gives you a quick overview.

Table 1.4. Comparison of XML Comments versus Remarks

Topic	XML Comments	Remarks
When to use it?	Comments which are only read by you	Comments which can and should be read by others
Where to place it?	Almost everywhere[a]	Restricted by the DocBook schema[b]
Can be displayed and suppressed?	No, always suppressed	Yes, by using the parameter *`show.comments`*

[a] An example where you *cannot* insert your XML comment is inside an attribute.

[b] See http://www.docbook.org/tdg51/en/html/tag.html for more details.

1.10. Knowing the Difference of Cross References and Links

Problem

You want to set a “link” but you do not know which DocBook element to use.

Solution

DocBook distinguishes between two link types:

- **Internal Links or Cross References** used primarily with `xref`
- **External Links** used primarily with `link`

Discussion

There are several difference between `xref` and `link`. An `xref` is used for cross referencing another part of the document. In its simplest form, you need only an ID value of the object where you want to link to:

Example 1.12. A Cross Reference

```
<chapter xml:id="tea">
  <title>All About Tea</title>
  <section xml:id="about-tea">
    <title>About Tea</title>
    <para>This section gives you an introduction about tea.</para>
  </section>
  <section xml:id="prepar">
    <title>Preparing Tea</title>
    <para>Read an introduction in <xref linkend="about-tea"/>.</para>
  </section>
</chapter>
```

The DocBook XSL stylesheets take care of resolving the `xref`. By default, it replaces the `xref` element with a number and the title of the reference. It could look like this:

```
Read an introduction in Section 1.1, “About Tea”.
```

The text `Section 1.1, “About Tea”` is a “hot link” and points to the respective section. Using `xref` for cross references has several advantages:

Advantages of Cross References Using `xref`

- Whenever a text of a title changes, all `xref`s pointing to it will automatically change too
- During validation, the `linkend` attribute is checked, if this ID exists
- During resolution, additonal words like “Section”, “Chapter”, or “Figure” are automatically inserted according to the used document language
- You do not have to care about numbering
- For further customization, use the `xrefstyle` attribute

In contrast, an `link` is used in most cases for *external* links.

Example 1.13. A Hyperlink

```
<section xml:id="about-tea">
```

```
  <title>About Tea</title>
  <para>See <link xlink:href="http://www.example.org/tea"/> for further information.</para>
</section>
```

Using an `link` has the following advantages:

Advantages of External Links Using `link`

- Usually no automatic resolution performed (only if you use `xlink:href`)
- No validation necessary between source and target
- Link content can be manually created
- Additional inline elements allowed

The differences between `xref` and `link` are collected in Table 1.5, “Differences Between `xref` and `link`” [20].

Table 1.5. Differences Between `xref` and `link`

Topic	`xref`	`link`
TDG[a] Definition	A cross reference to another part of the document	A hypertext link
TDG Link	http://www.docbook.org/tdg51/en/html/xref.html	http://www.docbook.org/tdg51/en/html/link.html
Content Model	empty [http://www.docbook.org/tdg51/en/html/xref.html]	text and inline elements [http://www.docbook.org/tdg51/en/html/link.html]
Resolution	Yes, automatically during transformation	No, link content must be definied manually
ID/IDREF Checking	Yes	No[b]
Restrictions	Only cross referencing to elements which contains a `title`	No restrictions

[a] The Definitive Guide, see http://www.docbook.org/tdg51/en/html/docbook.html

[b] Checking can be performed, when `linkend` attribute is used.

See Also

- Section 1.11, “Inserting Cross References” [21]
- Section 1.12, “Choosing Between Link Methods” [25]
- http://www.docbook.org/tdg51/en/html/link.html
- http://www.docbook.org/tdg51/en/html/xref.html

1.11. Inserting Cross References

Problem

You want to create a cross reference from one part of your document into another.

Solution

Use the `xref` element for this purpose. A cross reference is a one-to-one relationship from your `xref` to your target. To create such a relationship you need two things:

1. The ID of your target. This is the value of the `xml:id` attribute. Make sure, it points to the structure itself, not the title.

2. The empty `xref` element and a `linkend` attribute which holds the ID of your target.

For example, you want to reference to the introduction chapter in your book. The chapter looks like this:

```
<chapter xml:id="intro">
  <title>Introduction</title>
  ...
</chapter>
```

As you can see, the chapter contains a title but most importantly, it also has a `xml:id` attribute. This is our “anchor” or ID.

If you want to create a cross reference to your introduction chapter, use the `xref` element as follows:

```
<para>Get basic information in <xref linkend="intro"/>.</para>
```

The previous `xref` is rendered as follows:

```
Get basic information in Chapter 1. “Introduction”.
```

Discussion

Cross referencing is important in your documents as it makes your text accessible for your reader. The previous section showed the simplest form of a cross reference. However, there is more to find out about `xref`. The following subsections gives you some useful tips and tricks:

Reference to Elements with Titles Only

Basically, DocBook allows you to insert a `xml:id` attribute on *every* element if you wish. However, that does not mean, you should insert one. As you could see from the previous section, when the `xref` is resolved, the DocBook stylesheets uses the `title` element to create the reference to its target. If you refer to a element where no title is allowed (for example, a `para`), the cross reference cannot resolved correctly. Depending on the version of the DocBook stylesheets, you get either an error or the `xref` uses the title of its nearest ancestor (for example, the parent's section where the above child para is included).

Fine-tuning Cross References

In cases where it is not enough to just refer to a target, use the `xrefstyle` attribute to fine-tune the appearance of the resolved cross reference. The `xrefstyle` attribute expects one of the following values:

`template: TEMPLATE_KEYWORDS`
: After the `template:` text, use the % placeholders to define where title, number, etc. should appear.

`select: SELECT_KEYWORDS`
: After the `select:`, use specific keywords which maps to title, number, etc.

ANY_NAME
: If the `xrefstyle` attribute does not start with neither `select` nor `template`, it is interpreted as a name which can be found in a language file. This solution is not covered in this topic.

Using `template`

The `template` method contains an immutable string including placeholders. These placeholders are replaced with its real values during transformation. For example, if you want a different text to this section than the default, you could use the following code:

```
Section <xref linkend="dbc.markup.xref" xrefstyle="template:%t"/> helps you with the xref eleme
```

During transformation, the `%t` placeholder is replaced by the target's title:

```
Section Inserting Cross References [21] helps you with the xref element.
```

As you could see, any other string not starting with % is untouched and literally copied. Table 1.6, “Available Placeholders for `template`” [22] gives you an overview about all available placeholders.

Table 1.6. Available Placeholders for `template`

Placeholder	Description
%d	Direction (“above” or “below”)
%n	Number
%o	Document name (only for `olink`s)
%p	Page number, if applicable
%s	Subtitle, if applicable
%t	Title

Using `select`

The `select` method expects several self-explanatory keywords which are also replaced during transformation. The difference is, you cannot use free text as in `template` and as such gives you less freedom. For instance, the first example in Using `template` [22] can also be expressed by using `select`:

```
Section <xref linkend="dbc.markup.xref" xrefstyle="select:title"/> helps you with the xref elem
```

Table 1.7. Available Keywords for `select`

Keyword	Example	Description
Labels		
`label`	`Chapter 5`	label and number
`labelname`	`Chapter`	Name of the referencing element
`labelnumber`	`5`	Only the number
Titels		
`title`	`Inserting Cross References`	Title without quotes
`quotedtitle`	`“Inserting Cross References”`	Title with quotes
Pagenumber		

Keyword	Example	Description
page	(page 100)	Page number in brackets
Page	“Page 100”	Uppercase “page”
pageabbrev	(p. 100)	Abbreviated page
pagenumber	100	Only the page number
nopage	n/a	Suppressed page number

The order of the keywords does not matter; a `select:title nopage` is the same than `select:nopage title`. The order is taken from the language files in `common/`. For example, the following English language file defines the order of a chapter title (number followed by the title):

```
<l:context name="title-numbered">
  <l:template name="article/appendix" text="%n. %t"/>
</l:context>
```

Using `xreflabel` Only in Rare Cases

In rare cases, you have to reference to an element which does not allow a title. For example, if you want to reference to a paragraph. In such a case, use the `xreflabel`.

For example, if you reference to this paragraph with an ID of `_para1`, your `xref` will be resolved as follows:

```
<xref linkend="_para1"/>: the section called “Using xreflabel Only in Rare Cases” [23]
```

As you can see, it prints the title of its ancestor element. However, if you add a `xreflabel` to this paragraph, you will end up with this:

```
<para xml:id="_para2" xreflabel="A better test"> ... <para
...
<xref linkend="_para2"/>: A better test [23]
```

Remember, this should not be your default markup when cross referencing. Always try to refer to a element which contains a title as noted in the section called “Reference to Elements with Titles Only” [21].

Watch For Language Pitfalls

When using `xref`s, it can be sometimes difficult to write a “fluent” sentence. For example, you could be inclined to start a sentence like this:

```
The following chapter <xref linkend="dbc.markup"/> explains...
```

However, the output is undesireable:

```
The following chapter Chapter 1, Knowing DocBook’s Structure [1] explains...
```

Removing the word “chapter” makes it less ugly, but the problem still persists: how to create fluent, easy to read sentences where cross references are nicely resolved?

One solution is to rewrite the sentence and start with the `xref`:

```
<xref linkend="dbc.markup"/> explains ...
```

If you do not like this style, you can switch to passive voice as another solution:

```
The topic ABC is explained in <xref linkend="dbc.markup"/>.
```

Sometimes passive voice is not appropriate. In cases where you do not want to change the tense or grammar, use the `template` or `select` methods. Applied to the first example, you can rewrite the `xref` with `xrefstyle`:

```
The following <xref linkend="dbc.markup" xrefstyle="chapter %t"/>, explains...
```

See Also

- http://www.docbook.org/tdg5/en/html/xref.html
- Section 1.10, “Knowing the Difference of Cross References and Links” [19]
- Section 1.12, “Choosing Between Link Methods” [25]
- Section 2.15, “Append Text or Graphics in Cross-References” [57]

1.12. Choosing Between Link Methods

Problem

You need some ideas how to markup links.

Solution

Use the `link`[2] element. Usually, external links are used in two ways:

- **Without Text** The link element is completely empty:

```
<link xlink:href="https://github.com/tomschr/dbcookbook"/>
```

- **With Text** The link element contains some text or even other elements:

```
<link xlink:href="https://github.com/tomschr/dbcookbook">The DoCookBook</link>
```

Don't forget to add the XLink namespace declaration into the root element of your document, for example:

```
<book xmlns:xlink="http://www.w3.org/1999/xlink" ...>
```

Discussion

The `link` element has several features for creating internal or external links. It is allowed to use text content or leave a `link` completely empty. Regardless of the text content, you have to choose between the attributes `xlink:href` or `linkend`.

Knowing the Properties of the `xlink:href` and `linkend` Attributes

Using `linkend` leads always to an internal link and is semantically equivalent to `xref`. You cannot create external links with the `linkend` attribute.

On the other hand, the XLink attribute `xlink:href` can be used for both types: internal or external links. An example is shown in Table 1.8, “Internal and External Links Examples” [25].

Table 1.8. Internal and External Links Examples With `linkend` vs. `xlink:href`

	`xlink:href`	`linkend`
Datatype	xsd:anyURI	xsd:IDREF
Internal	`#cha.intro`	`cha.intro`
External	`http://www.docbook.org`	#

Note, validation is only performed when using `linkend`, not with `xlink:href`! The reason for this is the different datatypes. The XML parser can only validate a xsd:ID and xsd:IDREF connection.

Choosing the Needed Combination

With text content, you have several combinations which is shown in Table 1.9, “Different Possibilities for Links” [25]:

Table 1.9. Different Possibilities for Links

		Content	
		With Content	**Empty**
Linking Attributes	`xlink:href`	Content is a link to the specified URI	The specified URI is used as the content

[2] In DocBook 4 use the `ulink` element

		Content	
		With Content	**Empty**
	`linkend`	Content is an internal link to the element identified	Semantically equivalent to an `xref` (internal link)

The following list shows some examples:

Links With Content Using `xlink:href`
: Use this code to create an external link with a text different than the URL:

```
<link xlink:href="https://github.com/tomschr/dbcookbook">My Project</link>
```

Links With Content Using `linkend`
: Use this code to create an internal link (cross reference). The “hot text” is created from the content of `link`:

```
<link linkend="cha.intro">Introduction</link>
```

Empty Links Using `xlink:href`
: Use this code to create an external link where the “hot text” is taken from the URL:

```
<link xlink:href="http://www.example.org/Intro"/>
```

  If you want to create an internal link without ID validation check, use this code:

```
<link xlink:href="#cha.intro"/>
```

  This is equivalent to the following `xref` notation:

```
<xref linkend="cha.intro"/>
```

  Note the absense of the # character. Regardless if it is an empty `link` with `xlink:href` or a `xref` with `linkend`, the ID has to exist somewhere in the document. Otherwise, the element cannot be resolved during processing and and the link will be broken.

Empty Links Using `linkend`
: This notation is semantically equivalent to an `xref`. See also Section 1.11, “Inserting Cross References” [21]:

```
<link linkend="cha.intro"/>
```

  The only difference of the previous example is the ID `cha.intro` must be unique and is checked if it is available in your document. If not, it is an validation error.

In most cases, it is enough to just use an empty link element. As such, avoid the following notation when the URL in `xlink:href` is the same than the text:

```
<link xlink:href="https://github.com/tomschr/dbcookbook">https://github.com/tomschr/dbcookbook<
```

This is mostly unnecessary as it is overly verbose, difficult to read, and does not add any value. Correct such links and remove the text:

```
<link xlink:href="https://github.com/tomschr/dbcookbook"/>
```

Shortening Links

DocBook 5 also adds new concepts. One of this new concept is that *every* element can become a link. In other words, you do *not* have to use a `link` element anymore. For example, if you want to insert a book title with an additional link, in DocBook 4 it has to be written like this:

```
<citetitle><ulink url="https://github.com/tomschr/dbcookbook">The DoCookBook</ulink></citetitle
```

or this, depending on what you prefer:

```
<ulink url="https://github.com/tomschr/dbcookbook"><citetitle>The DoCookBook</citetitle></ulink
```

With DocBook 5 this can be shortend:

```
<citetitle xlink:href="https://github.com/tomschr/dbcookbook">The DoCookBook</citetitle>
```

Apart from the previous described link concept, DocBook 5 allows to control the link behavior. For example, to open a new window when clicking a link, use the `xlink:show` attribute:

```
<link xlink:show="new" xlink:href="https://github.com/tomschr/dbcookbook"/>
```

The attribute value `show` opens a “new window, frame, pane, or other relevant presentation context” (see Common Linking Attributes [http://www.docbook.org/tdg51/en/html/ref-elements.html#common.linking.attributes]). Of course, this is only useful for online formats.

See Also

- Section 1.11, “Inserting Cross References” [21]
- Section 1.10, “Knowing the Difference of Cross References and Links” [19]
- http://www.docbook.org/tdg5/en/html/link.html

1.13. Inserting Inline Quotes

Problem

You want to use inline quotes in a consistent and language independant way.

Solution

Use the `quote` tag:

```
<para>This is an <quote>English quote</quote>.</para>
```

Discussion

When people need quotes, they usually write "this". However, from a typographical point of view, this is wrong. There are special start and end quotation characters. The more elegant way is to use “this”.

However, the open and close characters are language dependent. Each language has different quotation rules. German readers prefer „this“, French « this », and the Italians use «this».

Using quote markers directly has one disadvantages: if your text contains different languages you need to know how quotes are displayed in the respective language. Also it is easier and more consistent to leave this typographic detail to the DocBook stylesheets.

One nice thing with inserting `quote` is it can be adapted to a different language by using the `xml:lang` attribute:

```
<para>This is a <quote xml:lang="de">German quote</quote>.</para>
```

1.14. Markup References to Man Pages

Problem

You search for an easy way to markup inline references to man pages.

Solution

Use `citerefentry [http://www.docbook.org/tdg51/en/html/citerefentry.html]`.

Example 1.14. Markup Code for Referencing Man Pages

```
<citerefentry>
  <refentrytitle>xmllint</refentrytitle>
  <manvolnum>1</manvolnum>
</citerefentry>
```

It is rendered like this:

Example 1.15. Rendered Result

xmllint(1)

Discussion

Some discussion about the problem and solution.

See Also

Include URL or bibliographic references.

Chapter 2. Common Customizations

Abstract

Some typical problems are independant from their target formats. This chapter gives you some information what falls into this category.

2.1. Introduction

A lot of customizations can be shared between different output formats. As such, it is probably a good idea to think about how to structure and store your customizations before starting. The following directories are a good start, especially if you want to support different output formats:

```
mycustomizations/
  +-- common/
  +-- fo/
  +-- html/
  +-- futher formats...
```

The `common` directory contains all the stylesheets which are not specific to any formats. On the contrary, `fo`, `html`, etc. contain the respective customizations for their target formats.

Furthermore, it is recommended to use the same file name than the original from the DocBook XSL stylesheets. This makes it easier to see, what changes can be expected within your files.

2.2. Finding Professional Fonts

Problem

You need to find professional fonts, preferably released under an open source license which can be used for print or Web formats.

Solution

There are some very good fonts which can be used freely. Most fonts are even released under the Open Font License [http://scripts.sil.org/OFL] or other free or open licenses. The following list is a small overview which is by no means complete (use also your favorite search engine):

http://scripts.sil.org
: SIL is a nonprofit organization for sustainable language development. One part is to “support the use of non-Roman and complex scripts in language development.” SIL creates the Charis SIL, Doulos SIL, Gentium, Andika, and other typefaces. All fonts are released under the Open Font License (OFL). This license was specially created for fonts and was initiated by SIL *FIXME: Check!*

http://www.google.com/webfonts
: Google Web Fonts is a huge collection of open source fonts. They can be freely incorporated into your Web sites using CSS or can be downloaded and embedded in PDF.

http://www.tug.org/tetex/
: The TeTeX distribution and CTAN contains also some high quality fonts, however, more aimed at the print business and not for the Web.

http://www.linuxlibertine.org
: The Linux Libertine fonts are designed to be an alternative to the serif Times fonts. The Linux Libertine and the Linux Biolinum fonts are a collection of serif and sans serif fonts with the usual Latin, Greek, Cyrillic, and Hebrew glyphs. Additionally, the fonts contain ligatures, true small caps, old style numbers, proportional or monospaced numbers, true superscript and subscript, and much more. A monospace variant is currently designed.

https://dejavu-fonts.github.io/
: The DejaVu font family are derived from the Bitstream Vera fonts. Due to license issues, the font has to be renamed. The fonts contain now lots of glyphs and supports Latin, Greek, Cyrillic, Armenian, Georgian, and other languages.

http://www.stixfonts.org
: The STIX fonts (Scientific and Technical Information Exchange) is a set of comprehensive fonts for scientific and engineering manuscripts. They can be used both for print and online publishing.

Discussion

Before you use a font, answer the following questions:

- **Supported Languages and Amount of Glyphs** Does your font support your language? Although modern fonts use Unicode thesedays, nevertheless your language might not be supported. Especially if you need “uncommon” languages or rarely used glyphs.

- **Missing Styles** Some fonts are published in only one style, usually *regular*. Such fonts makes it more difficult to universally adopt them in different scenarios where more styles (bold, italic, or a combination) are needed.

- **Font Formats** Modern fonts are usually Unicode fonts published as OTF or TTF format. Older fonts are mostly PostScript Type 1 fonts which have a limited amount of glphys (theoretically 256). PostScript fonts cannot be used on the Web.

- **Font Combinations** For a non-designer, it is generally easier to use a complete font family conisting of sans serif, serif and monospaced typefaces. The reason for this is these typefaces are made to fit together. It is much more difficult to select fonts which work aesthetically as it needs experiences.

- **Quality** This is a broad topic and it depends heavily where you want to use the font. Is it primarily used for Web content or only for print? Or both? Some fonts looks only good on a Web page, other work only for printed documents. Sometimes the *hinting* (adjustment of a font to a certain resolution) may or may not be implemented.

2.3. Writing Customization Layers

Problem

You want to write a customization layer for the DocBook XSL stylesheets, but you do not know how to do it.

Solution

A DocBook XSL customization layer comprise the following components:

- A XSLT stylesheet skeletton with the start tag `<xsl:stylesheet>` and the end tag `</xsl:stylesheet>`
- At least the namespaces for XSLT, DocBook 5, and your output format (for example, FO or XHTML).
- A `<xsl:import/>` element to incoporate the base DocBook XSLT stylesheet.
- Your customizations (parameters, variables, templates, or other imports).

How this can look is shown in the following listing.

Example 2.1. General Customization Layer

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://docbook.org/ns/docbook">

  <xsl:import href="https://cdn.docbook.org/release/xsl/current/FORMAT/docbook.xsl"/>

  <!-- Your customizations go here -->

</xsl:stylesheet>
```

Discussion

Why not modify the original stylesheets? There are some profound reasons which speaks against such modifications:

- Whenever you update your stylesheet, all your modifications are lost.
- You cannot separate between different customizations, for example, in one book you need an index but in the other it is unwanted.
- If you modify the original stylesheets directly you can not go back to the former output.
- It is hard to distinguish between your customizations and the original one.

Refrain from editing the original stylesheets! Always write a customization layer.

Apart from the above, obvious reasons, the following list gives you some recommendations:

- Insert common parameters or templates into a stylesheet which is included in your different customizations.
- Structure your output formats in different directories as shown in Section 2.1, “Introduction” [32].
- Name your main stylesheet `docbook.xsl` or `chunk.xsl` (HTML only).
- If your customization grows, outsource your customization into several stylesheets and include them into your “main” stylesheet.
- Name your files like the DocBook XSL stylesheets. For example, if you customize `filename` (an inline element) put it into the file `inline.xsl`.

An example how this can look like, is shown in Example 2.2, “Structure ” [36]:

Example 2.2. Structure

```
mycustomizations/
  +-- common/
      +-- common.xsl 1
  +-- fo/
      +-- docbook.xsl 2
      +-- inline.xsl  3
  +-- html/
      +-- docbook.xsl 4
      +-- chunk.xsl   5
      +-- inline.xsl  6
```

1 Common customizations across different formats
2 Main stylesheet for FO output; includes `inline.xsl`
3 Contains all modifications for inline elements regarding the FO output.
4 Main stylesheet for single HTML output; includes `inline.xsl`
5 Main stylesheet for chunked HTML output; imports `docbook.xsl`
6 Contains all modifications for inline elements regarding the HTML output.

To use one of the customization layer with, for example, **xsltproc**, use:

```
xsltproc mycustomizations/html/docbook.xsl mydocbook.xml
```

See Also

http://www.sagehill.net/docbookxsl/CustomMethods.html#CustomizationLayer

2.4. Inserting Date and Time

Problem

You want to insert the current date, time, or both in your output format (for example, to show the date of creation).

Solution

Use the `<?dbtimestamp?>` processing instruction (PI). For example, integrate it into the `pubdate` to show the current publication date on the titlepage:

```
<info>
  <!-- ... -->
  <pubdate><?dbtimestamp?></pubdate>
</info>
```

Discussion

The above PI includes the date in its *localized* form, or in other words: the output depends on the current language. If you do not have set any language, the default is English (`en_US`) which is `month/day/year`. If you want to change the default format, use one of the two options:

- Change the default format
- Add pseudo-attributes

Changing the Default Format

Each language contains in its language files*add xref* the context `datetime`. The following code shows it for the English language file:

```
<l:context name="datetime">
  <l:template name="format" text="m/d/Y"/>
</l:context>
```

This is only useful, if you want to completely change the appearance of a date in your document.

Adding Pseudo-Attributes

The simpler method is to change the PI directly. Especially if you do not want to customize the date for the complete document or you want it change individually. Basically, add a *pseudo-attribute*[1] into the `<?dbtimestamp?>` PI. For example, if you want to insert only the year, add the `format` pseudo-attribute:

```
<?dbtimestamp format="Y"?>
```

More format-letters can be found in the reference page (see link at the *See Also* section).

See Also

- http://www.sagehill.net/docbookxsl/Datetime.html
- http://docbook.sourceforge.net/release/xsl/current/doc/pi/dbtimestamp.html, the reference page for the PI

[1] A *pseudo-attribute* looks like an XML attribute due to its similar syntax. However, it is not. A processing instruction can only contain text and no attributes. For more information about PIs, refer to the XML specification at http://www.w3.org/TR/REC-xml/#sec-pi.

2.5. Accessing Title Contents

Problem

You need to retrieve a title (or subtitle), but your current node is not a `title`.

Solution

The DocBook XSL stylesheets offers the `title.markup` mode for this purpose. Usually, insert the following code in the appropriate place:

```
<xsl:apply-templates select="." mode="title.markup"/>
```

The `xsl:apply-templates` recursively finds the correct `title` node, regardless where you are. It also looks into an `info` element, if necessary.

Discussion

DocBook contains three elements which are used for title markup:

`title [http://www.docbook.org/tdg51/en/html/title.html]`
: The main title. According to the TDG “it identifies the titles of documents and parts of documents, and is the required caption on formal objects.”

`subtitle [http://www.docbook.org/tdg51/en/html/subtitle.html]`
: A optional subtitle is an alternative or explanatory title in addition to its main title. Usually a subtitle is printed below the main title on the same page.

`titleabbrev [http://www.docbook.org/tdg51/en/html/titleabbrev.html]`
: The optional, abbreviated title is used in case the main title is overly verbose. Usually this element is not printed with the main title on the same page. However, it is used in the table of content, for example.

All of the above title elements contains a special mode to get the content of this node:

Table 2.1. Modes for Title Elements

Element	Mode
`title`	`title.markup`
`subtitle`	`subtitle.markup`
`titleabbrev`	`titleabbrev.markup`

To further elaborate why you should use one of the previous modes, let's assume you have this chapter title:

```
<chapter>
   <title>Programming in Python</title>
   <!-- further substructure pruned -->
</chapter>
```

Let's further assume, you have a template where you need the title from the above chapter. One simple, yet unfavorable, solution could be coded like this:

```
<xsl:template name="...">
   <xsl:value-of select="ancestor::d:title"/>
</xsl:template>
```

This has several disadvantages:

- The XPath does not consider a title inside an `info` element.
- The `xsl:value-of` returns the *string value*. In most cases this is correct. However, if you have a `quote` inside `title`, the `quote` is not processed and you will not get any quotation marks.

Replacing `xsl:value-of` through the `xsl:apply-templates` as shown in the solution section, ensures correct processing of child elements inside `title`.

The same is true for `subtitle` and `titleabbrev`. Using the `subtitle.markup` or `titleabbrev.markup` mode ensure you get always the correct content.

To see what combination results in what string, find an overview in Table 2.2, “Different Combinations and Their Results on `*.markup` Modes” [39].

Table 2.2. Different Combinations and Their Results on `*.markup` Modes

Code	Results
`<title>Programming in Python</title>`	• `title.markup`: “Programming in Python” • `subtitle.markup`: “” (empty string) • `titleabbrev.markup`: “Programming in Python”
`<title>Programming in Python</title>` `<subtitle>With an IDE</subtitle>`	• `title.markup`: “Programming in Python” • `subtitle.markup`: “With an IDE” • `titleabbrev.markup`: “Programming in Python”
`<title>Programming in Python</title>` `<titleabbrev>Python</titleabbrev>`	• `title.markup`: “Programming in Python” • `subtitle.markup`: “” (empty string) • `titleabbrev.markup`: “Python”

`<title>Programming in Python</title>`
`<subtitle>With an IDE</subtitle>`
`<titleabbrev>Python</titleabbrev>`
	• `title.markup`: “Programming in Python”
	• `subtitle.markup`: “With an IDE”
	• `titleabbrev.markup`: “Python”

See Also

- http://www.sagehill.net/docbookxsl/ReplaceTemplate.html#UtilityTemplates

2.6. Getting the Documentation Title

Problem

You need to get the documentation title as a string.

Solution

Use the named template `get.doc.title` (from `common/utilities.xsl`):

```
<xsl:template match="...">
   <xsl:variable name="doctitle">
     <xsl:call-template name="get.doc.title"/>
   </xsl:variable>
   <!-- ... use the varialbe $doctitle ...-->
</xsl:template>
```

Discussion

The documentation title is the outmost title of your text. This can be a title from a book, glossary, set, article, or other, depending on what is your root element.

The `get.doc.title` template returns a string. In most cases this is probably enough, for example, if you want to place them as an attribute value where markup is not allowed.

If you need the full markup, use the methods described in Section 2.5, “Accessing Title Contents” [38].

See Also

- Section 2.5, “Accessing Title Contents” [38]
- Section 4.7, “Numbering Part, Chapter, Appendix, and other Titles” [133]

2.7. Extracting Information from Your Own Processing Instructions

Problem

You need an own processing instruction (not one from the DocBook XSL stylesheets) to fine-tune your transformation.

Solution

The DocBook XSL stylesheet offers the `pi-attribute` template for this purpose. It expects a processing instruction node and a "pseudo-attribute" name. For example, the following processing instruction is given:

```
<?toms-html background-color="blue"?>
```

Call `pi-attribute` to extract the content of the pseudo-attribute `name`:

```
<xsl:call-template name="pi-attribute">
  <xsl:with-param name="pis" select="processing-instruction('toms-html')"/>
  <xsl:with-param name="attribute" select="'background-color'"/>
</xsl:call-template>
```

This gives you the result `blue`.

Discussion

Processing instructions (PIs) are usually needed for fine-tune your output formats. For example, to add page break information, background color, or other stylistic hints. PIs should *not* be used to insert data which should normally reside inside a DocBook element! As a general rule of thumb: Use PIs if you want provide layout specific information for your output format if there is no other method.

The following example use the PI `<?toms-html?>` which is included inside an `para` element to insert background color in HTML:[2]

1. Invent a target name for your processing instruction. This name should be unique and must not conflict with existing PIs (like `<?dbhtml?>`, `<?dbfo?>`, etc.) For example, you could use your abbreviated name and the output format to distinguish it from other processing instructions.

2. Insert your PI into a para which indicates a different background color:

   ```
   <para><?toms-html background-color="blue"?>
     ...
   </para>
   ```

3. Create a customization layer as shown in Section 2.3, "Writing Customization Layers" [35].

4. Customize the `para` template and add the following code to your customization layer. The customized template call `pi-attribute` with the relevant parameters. If no PI was set, apply the original template:

   ```
   <xsl:template match="d:para">
     <xsl:variable name="bg">
       <xsl:call-template name="pi-attribute">
         <xsl:with-param name="pis" select="processing-instruction('toms-html')"/>
         <xsl:with-param name="attribute" select="'background-color'"/>
       </xsl:call-template>
     </xsl:variable>
     <xsl:choose>
       <xsl:when test="$bg != ''">
         <div style="background-color:{$bg}">
   ```

[2]Usually, such a need would be solved through CSS and not PIs. However, this example shows just the principle so it is a perfect use case.

```
          <xsl:apply-imports/>
        </div>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-imports/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

1. Fill the variable `bg` with the result of `pi-attribute`. Note, the `select` only matches for PIs *inside* a `para`.
2. Decide what to do when background color is found.
3. Wrap our paragraph around `div` with the corresponding `background-color` inside a `style` attribute.
4. Delegate the default handling to the original `para` template.

5. Build your document with your customization layer.

The previous example recognizes a PI *inside* a `para` only. If you want to recognize also PIs *before*, use the following XPath in the `select` attribute of `pi-attribute`:

```
( processing-instruction('toms-html') |
  preceding-sibling::processing-instruction('toms-html')[1]
)[last()]
```

If two PIs are available (one inside, another outside), the one which is located inside takes precedence.

See Also

- http://docbook.sourceforge.net/release/xsl/current/doc/lib/pi-attribute.html

2.8. Retrieving XPath

Problem

You need the path to your current node—the XPath—to output it as debugging information.

Solution

The DocBook XSL stylesheet offers the `xpath.location` template for this purpose. To output the current XPath, use the following procedure:

1. Import the `xpath.location` template from the DocBook XSL stylesheets in your customization layer:

```
<xsl:import href="http://docbook.sourceforge.net/release/xsl/current/lib/lib.xsl"/>
```

2. Add the following code in your template:

```
<xsl:message>
  <xsl:text>Current XPath: </xsl:text>
  <xsl:call-template name="xpath.location"/>
</xsl:message>
```

If you need the XPath for a different node, extend the previous code:

```
<xsl:message>
  <xsl:text>Current XPath: </xsl:text>
  <xsl:call-template name="xpath.location">
    <xsl:with-param name="node" select="YOUR_XPath_Expression"/>
  </xsl:call-template>
</xsl:message>
```

Discussion

Currently, the template in `lib/lib.xsl` is limited: it walks from the current node up to the root node and outputs just the element name. No namespace prefixes, no predicates. If you have a DocBook document with different namespaces, the original version does not help.

In Example 2.3, “Namespace-aware Output of an XPath” [43], the revised template recognizes namespaces and counts elements on the sibling axis.

Example 2.3. Namespace-aware Output of an XPath

```
<xsl:stylesheet version="1.0"
  xmlns:n="urn:x-toms:ns:namespaces"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  exclude-result-prefixes="n">
  <namespaces xmlns="urn:x-toms:ns:namespaces">
    <ns prefix="d">http://docbook.org/ns/docbook</ns>
    <ns prefix="xi">http://www.w3.org/2001/XInclude</ns>
    <ns prefix="rdf">http://www.w3.org/1999/02/22-rdf-syntax-ns#</ns>
    <ns prefix="cc">http://creativecommons.org/ns#</ns>
    <ns prefix="svg">http://www.w3.org/2000/svg</ns>
  </namespaces>

  <xsl:variable name="nsnodes" select="document('')//n:namespaces/n:ns"/>

  <xsl:template name="xpath.location">
    <xsl:param name="node" select="."/>
    <xsl:param name="path" select="''"/>
    <xsl:param name="method">prefix</xsl:param>
```

```
    <xsl:variable name="next.path">
      <xsl:choose>
        <xsl:when test="$method = 'prefix' and $nsnodes[namespace-uri($node)]">
          <xsl:value-of select="concat($nsnodes[namespace-uri($node) = .]/@prefix, ':')"/>
        </xsl:when>
        <xsl:when test="$method = 'clark' and namespace-uri($node) != ''">
          <xsl:value-of select="concat('{', namespace-uri($node), '}')"/>
        </xsl:when>
      </xsl:choose>
      <xsl:value-of select="local-name($node)"/>
      <xsl:if test="generate-id($node) != generate-id(/*) and
                    count(../*[local-name()=local-name(current()) and
                               namespace-uri()=namespace-uri(current())]) > 1">
        <xsl:text>[</xsl:text>
        <xsl:value-of select="count(preceding-sibling::*
                          [local-name()=local-name(current()) and
                           namespace-uri()=namespace-uri(current())]) + 1"/>
      <xsl:text>]</xsl:text>
    </xsl:if>
      <xsl:if test="$path != ''">
        <xsl:text>/</xsl:text>
      </xsl:if>
      <xsl:value-of select="$path"/>
    </xsl:variable>

    <xsl:choose>
      <xsl:when test="$node/parent::*">
        <xsl:call-template name="xpath.location">
          <xsl:with-param name="node" select="$node/parent::*"/>
          <xsl:with-param name="path" select="$next.path"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="concat('/', $next.path)"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

1. Contains the list of namespaces and their prefixes
2. Create a node set of all namespace to prefix mappings
3. Defines the method how prefixes are displayed. You can choose between “prefix” and the Clark notation. The latter adds the namespace in curly brackets before the element name, but makes the whole XPath very verbose.
4. Choose between the prefixed or Clark notation
5. Checks, if we are not at the root node and there is more than one element with the same name and the same namespace. If this is true, we output a predicate where the current number is enclosed in squared brackets.

The template can output the XPath in two different notations:

Prefix (`method="prefix"`, default)
: Namespace nodes are abbreviated with a prefix. The prefix is definied in the `namespaces` element. A XPath to the first chapter inside a book looks like this:

```
/d:book/d:chapter[1]
```

Clark (`method="clark"`)
: Namespace nodes are printed with the full namespace URL. The namespace URL is definied in the `namespaces` element. A XPath to the first chapter inside a book looks like this:

```
/{http://docbook.org/ns/docbook}book/{http://docbook.org/ns/docbook}chapter[1]
```

Use Clark Notation For Debugging Purposes Only

The above notation is called the *Clark notation*, after James Clark, the editor of the XSLT 1.0 and XPath 1.0 specifications. However, this notation cannot be used in XSLT as it is not officially supported. Only use the prefixed version in this case. The Clark notation is just useful for debugging purposes when you want or need the full namespace URLs.

Be aware, the previous template is slower than the original version (it has to count a little bit more). For debugging purpose it is probably fine.

See Also

- http://docbook.sourceforge.net/release/xsl/current/doc/lib/xpath.location.html
- A recursive solution, considering also attribute, text, and processing instruction nodes: http://www.dpawson.co.uk/xsl/sect2/xpath.html#d11572e154

2.9. Extending Language Files with Your Own Text

Problem

You need a general solution to add localized text without hard-coding it into your stylesheet. Depending on the language, it should display the correct, translated text.

Solution

Use *language files* to add your text. Language files are the DocBook way to support several languages, all located under the `common` directory. Each file is in XML format and contains all your translated text as a key/value pair. For more complex settings and to group things, there are *contexts*. The key is never translated, it a constant and is only needed to find and retrieve the translated text.

To use a language file, proceed as follows:

Procedure 2.1. Adding Your Own Localized Text

1. Create a customization layer first as shown in Section 2.3, “Writing Customization Layers” [35].

2. Add the parameter *`local.l10n.xml`* in your customization layer and point it to your language file (in this case, it is named `myl10n.xml`, but you can use any name you like):

   ```
   <xsl:param name="local.l10n.xml" select="document('myl10n.xml')"/>
   ```

3. Open the `myl10n.xml` file and insert the following XML code as an example:

   ```
   <l:i18n xmlns:l="http://docbook.sourceforge.net/xmlns/l10n/1.0">
     <l:l10n language="en" english-language-name="English">
       <l:gentext key="Authors" text="Authors"/>
       <l:gentext key="lastbuilt" text="Last built: "/>
     </l:l10n>
     <l:l10n language="de" english-language-name="German">
       <l:gentext key="Authors" text="Authoren"/>
       <l:gentext key="lastbuilt" text="Zuletzt gebaut: "/>
     </l:l10n>
   </l:i18n>
   ```

   The above code shows two language, English and German (marked in the `language` attribute). Furthermore, it contains two text entries for each language (“Authors” and “lastbuilt”).

4. Call `gentext` in your template to retrieve the translated text:

   ```
   <xsl:call-template name="gentext">
     <xsl:with-param name="key" select="'Authors'"/>
   </xsl:call-template>
   ```

Discussion

The “gentext” method to insert language specific text is quite powerful, but not almighty. It helps you to keep translateable text in one place and avoids hard-coded locations in your stylesheets. If you need the translated text not for the default language (usually marked in the root element), but for a different language, use the `xsl:with-param` as shown:

```
<xsl:call-template name="gentext">
  <xsl:with-param name="key" select="'Authors'"/>
  <xsl:with-param name="lang">de</xsl:with-param>
</xsl:call-template>
```

See Also

http://www.sagehill.net/docbookxsl/Localizations.html

2.10. Extracting Language Information

Problem

You need the language of the current element or of your DocBook document.

Solution

Use the `l10n.language` template from `common/l10n.xsl`. It extracts the language attribute from the current context node or its ancestors:

```
<xsl:variable name="language">
   <xsl:call-template name="l10n.language"/>
</xsl:variable>
```

It returns a “normalized” string which is RFC1766 compliant.

If you need to change the current node, use the *`target`* parameter. For example, the following code extracts the language from the root element:

```
<xsl:variable name="lang">
   <xsl:call-template name="l10n.language">
      <xsl:with-param name="target" select="/*"/>
   </xsl:call-template>
</xsl:variable>
```

Discussion

Language information is stored in DocBook either in the `lang` (4.x) or `xml:lang` (>5.x) attributes. These attributes can be set on every DocBook element.

For example, if you have a chapter inside a book and want to extract the contents of the language attribute from this chapter, the following XPath expression gives you this information:

```
/book/chapter/@lang              DocBook 4.x
/db:book/db:chapter/@xml:lang    DocBook 5.x
```

Although the XPath expressions are simple, they have some drawbacks:

- If no language attribute is set, no content can be retrieved. The used language is undefined.
- There is no default language when a language is not set.
- Language information can be written inconsistently, for example, `en` or `EN`.
- The above expressions do not take into account the current context node. For example, the language in a `foreignphrase` element is usually different than a chapter or other parts of the document.
- There are no checks for RFC compliance.

All the previous problems are considered by the `l10n.language` template. It searches for the nearest anchestor who contains a language attribute and returns its content. For example, consider the following structure:

```
book xml:lang="en"
  chapter
    section
      para
        foreignphrase xml:lang="de"
```

If your current context is on the `foreignphrase` element, `l10n.language` will return `de` as language. However, if your context is on the `para` element, you will get `en`. The `para`, `section`, and `chapter` elements are children

of book and therefore in the scope of the language attribute which gives you en. In the case no language is given, the template returns a default language from the *l10n.gentext.default.language* parameter (usually English).

Sometimes, returning only the language is not enough. The following table gives you some additional functions from common/l10n.xsl which can be useful for your code (Question marks(?) denote an optional parameter):

Table 2.3. Extracting Language Information

Template	Description
`attrnode = language.attribute(node="."?)`	Useful for HTML. Returns an attribute node `lang` with the extracted language from the current node or one of its ancestors
`attrnode = xml.language.attribute(node="."?)`	Useful for XHTML. Returns an attribute node `xml:lang` with the extracted language from the current node or one of its ancestors
`string = l10n.language.name(lang?)`	Returns the English language name

See Also

- http://www.ietf.org/rfc/rfc1766.txt
- http://docbook.sourceforge.net/release/xsl/current/doc/html/l10n.gentext.default.language.html
- http://docbook.sourceforge.net/release/xsl/current/doc/html/l10n.lang.value.rfc.compliant.html

2.11. Extracting and Formatting Person and Author Information

Problem

You need to format an author, a group of authors, or any other name of persons.

Solution

The DocBook stylesheets provide the template `person.name` for a single name or `person.name.list` for a group of names. For example, consider the following `info` element:

```
<info>
  <author>
    <personname>
      <firstname>Tux</firstname>
      <surname>Penguin</surname>
    </personname>
  </author>
</info>
```

To retrieve the author name, use the following code in your template:

```
<xsl:call-template name="person.name">
  <xsl:with-param name="node" select="$theauthor"/>
</xsl:call-template>
```

whereas the variable `theauthor` points to the author node. The previous template returns the expected string:

```
Tux Penguin
```

For a group of names it is similar, just replace `person.name` with `person.name.list`.

Discussion

At a first glance, names seems pretty easy to format. However, it is a little bit more complicated. For example, the title of a person, its lineage, or middle names can makes it sometimes harder to extract all the information. The following author contains a title and a lineage:

```
<author>
  <personname>
    <honorific>Dr.</honorific>
    <firstname>Tux</firstname>
    <othername>Tuxy</othername>
    <surname>Penguin</surname>
    <lineage>Jr.</lineage>
  </personname>
</author>
```

Depending on what you need, the DocBook stylesheets can format the name with different

Table 2.4. Overview of Available Templates

Template	Result
`person.name`	`Dr. Tux Tuxy Penguin, Jr`
`person.name.last-first`	`Penguin, Tux`
`person.name.family-given`	`Penguin Tux [FAMILY Given]`

Keep in mind, the current implementation only takes into account the first occurance of `honorific` or `lineage`.

2.12. Splitting Header Into Label and Title With Modes

Problem

Your header contains a consecutive number and a title. As you want to style them differently, you need to split them in this two pieces.

Solution

The DocBook stylesheets have two modes which are “responsible” for generating the labels and titles: the modes `label.markup` and `title.markup`.

To “split” the header, you need to introduce an inline element for styling. For HTML this is `span` with `class` attribute, for XSL-FO it is `fo:inline`. The principle is the same, but this topic shows only HTML for better readability.

Customize the `label.markup` and `title.markup` templates for the specific DocBook element. To simplify the maintenance and customization, use the XSLT `<xsl:apply-imports/>` tag. This allows you to call the original template without repeating it and to introduce your changes. Save the result from `<xsl:apply-imports/>` in a variable and and output it only, if the variable is not empty. The following template shows this approach for a chapter label:

```
<xsl:template match="d:chapter" mode="label.markup">
  <xsl:variable name="number">
    <xsl:apply-imports/>
  </xsl:variable>

  <xsl:if test="string($number) != ''">
    <span class="number"><xsl:value-of select="$number"/></span>
  </xsl:if>
</xsl:template>
```

For a chapter title the template is even simpler:

```
<xsl:template match="d:chapter" mode="title.markup">
  <span class="title">
    <xsl:apply-imports/>
  </span>
</xsl:template>
```

Add the same changes for all elements that you need a different styling. After you have included this changes in your customization layer, a chapter title in HTML is shown as follows:

```
<h2 class="title"><a id="id275730"></a>Chapter
  <span class="number">1</span>. <span class="title">...</span>
</h2>
```

To style the number in bold and the title in a normal weight, use the following CSS rule:

```
h2.title > .number {
  font-weight: bold;
}
h2.title > .title {
  font-weight: normal;
}
```

Of course, you can apply all CSS rules you want.

Discussion

Although the customization has some benefits (simple, easy to maintain, support the language files), you should be aware of some issues.

The `label.markup` and `title.markup` modes are very basic modes; as such, they are used in other templates as well, not only for titles. This can lead to side effects, which you may or may not want. For example, the modes are used also for the table of content and for resolving cross references.

To deal with this situation is two-fold:

- Leave it as it is and customize it at the CSS level.
- Only

Protecting Cross References

To prevent the splitting in cross references, customize the `insert.title.markup` and `insert.label.markup`modes as follows:

```
<xsl:template match="*" mode="insert.label.markup">
  <xsl:param name="purpose"/>
  <xsl:param name="xrefstyle"/>
  <xsl:param name="label"/>

  <xsl:value-of select="$label"/>
</xsl:template>

<xsl:template match="d:chapter" mode="insert.title.markup">
  <xsl:param name="purpose"/>
  <xsl:param name="xrefstyle"/>
  <xsl:param name="title"/>

  <xsl:choose>
    <xsl:when test="$purpose = 'xref'">
      <em><xsl:copy-of select="$title"/></em>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$title"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

The difference between the original and the above customization is the `xsl:copy-of` tag. In the above customization it is replaced by `xsl:value-of`. This small change ensure only the string value is copied, not any elements.

Protecting the Table of Contents

To avoid the same structure in table of contents, customize the `toc.line` template as follows:

```
<xsl:template name="toc.line">
  <xsl:param name="toc-context" select="."/>
  <xsl:param name="depth" select="1"/>
  <xsl:param name="depth.from.context" select="8"/>
  <xsl:variable name="title">
    <xsl:apply-templates select="." mode="titleabbrev.markup"/>
  </xsl:variable>

  <span class="{local-name(.)}">
    <xsl:if test="$autotoc.label.in.hyperlink = 0">
      <xsl:variable name="label">
        <xsl:apply-templates select="." mode="label.markup"/>
      </xsl:variable>
      <xsl:value-of select="$label"/>
      <xsl:if test="$label != ''">
        <xsl:value-of select="$autotoc.label.separator"/>
      </xsl:if>
```

```
    </xsl:if>

    <a>
      <xsl:attribute name="href">
        <xsl:call-template name="href.target">
          <xsl:with-param name="context" select="$toc-context"/>
          <xsl:with-param name="toc-context" select="$toc-context"/>
        </xsl:call-template>
      </xsl:attribute>
      <!-- * if $autotoc.label.in.hyperlink is non-zero, then output the label -->
      <!-- * as part of the hyperlinked title -->
      <xsl:if test="not($autotoc.label.in.hyperlink = 0)">
        <xsl:variable name="label">
          <xsl:apply-templates select="." mode="label.markup"/>
        </xsl:variable>
        <xsl:value-of select="$label"/>
        <xsl:if test="$label != ''">
          <xsl:value-of select="$autotoc.label.separator"/>
        </xsl:if>
      </xsl:if>
      <xsl:value-of select="$title"/>
    </a>
  </span>
</xsl:template>
```

1 Saves the output from `<xsl:apply-templates/>` in the variable `title` to use it later
2 Use `<xsl:value-of/>` to get the string from the `label` variable
3 Insert the string content of variable `title`

See Also

.

2.13. Numbering Figures and the Like Consistently Through your Document

Problem

You want to enumerate your figures, tables, programlisting or examples consistently through your document.

Solution

Use a customization layer and insert the following template in your customization layer:

Example 2.4. Template to Number Figures Consistently

```
<xsl:template match="d:figure" mode="label.markup"> 1
    <xsl:choose>
      <xsl:when test="@label"> 2
        <xsl:value-of select="@label"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:number format="1" from="d:book|d:article" level="any"/> 3
      </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

1 Matches for `figure` elements in `label.markup` mode (a special mode for getting or calculating numbers).
2 If the figure contains a `label` attribute, it will be used. This gives you the possibility to overwrite the displayed label.
3 Calculates the number of the current `figure` element, starting on a book or article level.

Discussion

The template shown in Example 2.4 [55] matches only `figure`s. It can be extended to match also `procedure`s, `tables`, `examples` etc. If you do not need the support for the `label` attribute, remove the `xsl:choose` and its children, but leave the `xsl:number` element.

If you need a different numbering schema, modify the `format` attribute in the `xsl:number` start-tag. For example, if you write `[a]` you will get consecutive numerals in brackets like `[a]`, `[b]`, `[c]` etc.

2.14. Converting Lowercase to Uppercase or Vice Versa

Problem

You need to convert a text string from lowercase or uppercase, respectively, taking care of the current locale.

Solution

Use the utility template `string.lower` or `string.upper` from the DocBook stylesheets:

```
<xsl:call-template name="string.upper">
   <xsl:with-param name="string">This is your string</xsl:with-param>
</xsl:call-template>
```

Or:

```
<xsl:call-template name="string.lower">
   <xsl:with-param name="string">This is your string</xsl:with-param>
</xsl:call-template>
```

Discussion

To transform a string into uppercase letters, XPath offers the `translate` function. The `translate` function expects three parameters: the string you want to change, a string of lowercase letters, and a string of uppercase letters. Your code looks like this:

```
<xsl:value-of select="translate($string, 'abcdefghijklmnopqrstuvwxyz', 'ABCDEFGHIJKLMNOPQRSTUVW
```

When called by the XSLT processor, the content of the `string` variable is transformed into uppercase. This works pretty well if your language is English or your string does not contain any accented characters. However, if your string contains, for example, the German lowercase letter “ö”, `translate` does not transform it into the uppercase letter “Ö”.

This solves the above templates `string.lower` and `string.upper`. They use the language files of the DocBook stylesheets and get the lowercase and uppercase letters from a so called “gentext templates”. The uppercase.alpha and lowercase.alpha entries are used in the current locale.

However, this transformation is not always perfect. For example, it is recommend to transform the German sharp “ß” (lowercase) into “SS” (uppercase).[3] This makes the `translate` function unusable for such corner cases. Only XPath 2.0 support local-sensitive mappings with `fn:upper-case [http://www.w3.org/TR/xpath-functions/#func-upper-case]`.

See Also

- `common/common.xsl`
- XPath 1.0 `translate(string, fromString, toString) [http://www.w3.org/TR/xpath/#function-translate]` function

[3]Since 2008, Unicode contains the capital sharp ß (see http://en.wikipedia.org/wiki/Capital_%E1%BA%9E which is located at position U+1E9E. Most fonts contain this glyphs now, but not all. You should carefully investigate your fonts before you change anything. For a German keyboard layout under X11, press **Caps Lock**+**ß** to display the capital sharp ß which is printed as “#”.

2.15. Append Text or Graphics in Cross-References

Problem

You have a cross-reference (usually tagged with `xref`) and you want to append text or graphics after the reference is resolved.

Solution

Use the `xref-to-prefix` or `xref-to-suffix` mode. These modes are called before or after the `xref` is being resolved. As such it makes it pretty easy to add text or graphics.

For example, the following example appends the text "[x]" to each `xref`, regardless where it points to:

Example 2.5. Appending Simple Text to Each Element

```
<xsl:template match="*" mode="xref-to-suffix">
   <xsl:text>[X]</xsl:text>
</xsl:template>
```

If you need the text only for specific elements (for example, only for chapters) use the `match` attribute:

Example 2.6. Appending Simple Text for Each Cross-Reference Pointing to a Chapter

```
<xsl:template match="d:chapter" mode="xref-to-suffix">
   <xsl:text>[X]</xsl:text>
</xsl:template>
```

Discussion

If you have to insert text before or after your reference only, you can use one of the templates described above. However, if you need to display a graphic, you need to insert FO or HTML specific code. For HTML, this could look like this:

Example 2.7. Adding Graphic for HTML

```
<xsl:template match="d:chapter" mode="xref-to-suffix">
   <img src="chapter.png" alt="Chapter"/>
</xsl:template>
```

If you need to add a graphic for FO, use the following template:

Example 2.8. Adding Graphic for FO

```
<xsl:template match="d:chapter" mode="xref-to-suffix">
   <xsl:variable name="srcfile">
     <xsl:call-template name="fo-external-image">
       <xsl:with-param name="filename" select="'chapter.png'"/>
     </xsl:call-template>
   </xsl:variable>

   <fo:external-graphic src="{$srcfile}" height="1em"/>
</xsl:template>
```

The same principle applies if you want to add something *before* your cross-reference. In this case, replace the `xref-to-suffix` mode with `xref-to-prefix`.

Chapter 3. Manipulating DocBook Document Structure

Abstract

DocBook files can be created manually or by tools. Sometimes tools create a structure which is inconvenient for some reasons or you have legacy documents which contain the “wrong” structure. This chapter shows how to restructure your DocBook document in the way you want.

3.1. Introduction

The methods described in this chapter are mostly XSLT solutions where you can find more or less in any XSLT book. Actually, some ideas origin from Sal Mangano´s excellent *XSLT Cookbook*. However, the solutions in this chapter are targeted exclusively on DocBook.

Most of these solutions are based on an *identity transformation* stylesheet which is shown in Example 3.1, “`copy.xsl`” [60].

Example 3.1. `copy.xsl`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="node() | @*">
    <xsl:copy>
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

This simple stylesheet copies everything from the input document to the output document without any modifications. Although it seems quite useless at a first glance, it reveals its full power when combining it with customizations.

3.2. Pretty-Printing DocBook Documents

Problem

Your file, be it autogenerated or somehow “mangled” is poorly indented and you want to get rid of this.

Solution

There are different solutions to this problem:

- a “pretty-print” stylesheet
- the XML parser **xmllint**
- the **xmlformat** command

XML Parser xmllint

The XML parser **xmllint** offers the `--format` option to turn on indentation for each element.

```
xmllint --format XMLFILE
```

The Pretty-Print Stylesheet

The simplest stylesheet for indentation is shown in Example 3.2, “`pretty.xsl`” [61]. It relies on the `copy.xsl` stylesheet.

Example 3.2. `pretty.xsl`

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

  <xsl:import href="copy.xsl"/>
  <xsl:output indent="yes"/>

  <xsl:strip-space elements="*"/>
  <xsl:preserve-space elements="d:screen d:literallayout
    d:programlisting d:address"/>
</xsl:stylesheet>
```

Command xmlformat

The **xmlformat** tool is a Perl script which is available from the Web site http://www.kitebird.com/software/xmlformat/.

The tool distinguishes between block elements, inline elements, and verbatim elements (similar to DocBook). The difference between the types is the whitespace normalization.

Block elements typically begin with a new line and children are indented. Spacing before and after can be controlled too.

Inline elements occur in block elements. Normalization and line-wrapping occurs in regard to the enclosing block element.

Verbatim elements are not formatted at all. That means, the content of the input element is the same as the content of the output element, including whitespaces.

Discussion

The **xmllint** command with its `--format` option is the easiest candidate but lacks customization. This is useful if you do not have any other tools at hand and you prefer a quick and rough reformatting.

The `pretty.xsl` stylesheet is a pure XSLT solution. As such, it works on every platform which supports an XSLT processor. It is adaptable to your needs, but mixed content (like in `para`) is problematic.

The most adaptable method is **xmlformat**.

3.3. Converting DocBook from Version 4 to Version 5

Problem

You have a DocBook document in version 4.x, but you need 5.x.

Solution

Generally, the difference between version 4 and version 5 is minimal. Refer to the The Definitive Guide [http://www.docbook.org/tdg51/en/html/ch01.html#d6e550] for detailed information what has been added, removed, or renamed.

One major change is that all DocBook 5.x elements are in the namespace `http://docbook.org/ns/docbook`. All these changes are taken into account by the `db4-upgrade.xsl`, see the URL https://github.com/docbook/docbook/blob/master/relaxng/tools/db4-upgrade.xsl. You just need to apply this stylesheet to your source DocBook document, for example:

```
$ xsltproc --output doc5.xml db4-upgrade.xsl doc.xml
```

After the migration, the file `doc5.xml` contains your DocBook5 source.

Discussion

One disadvantage is that entities are not preserved. This is not a stylesheet issue but a XML issue. The entities are already resolved when the XSLT processor gets its hand on the source tree. The stylesheet never sees the entities.

In cases you need to leave your entities untouched, refer to Section 3.18, “Preserving Entities” [114].

See Also

- Section 3.18, “Preserving Entities” [114]
- http://docbook.org/docs/howto/#convert4to5

3.4. Converting DocBook from Version 5 to Version 4

Problem

You have a DocBook document in version 5.x, but you need 4.x.

Solution

Generally, the difference between version 4 and version 5 are minimal. Refer to the The Definitive Guide [http://www.docbook.org/tdg51/en/html/ch01.html#d6e550] for detailed information what has been added, removed, or renamed.

In case you have or get DocBook 5 and need the former version, the following stylesheet which supports the core transformation might help:

Example 3.3. `db5to4-core.xsl`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:d="http://docbook.org/ns/docbook"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xi="http://www.w3.org/2001/XInclude"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns:exsl="http://exslt.org/common"
  exclude-result-prefixes="d xi xlink exsl html">

  <xsl:import href="copy.xsl"/>

  <xsl:output method="xml" indent="yes"
    doctype-public="-//OASIS//DTD DocBook XML V4.5//EN"
    doctype-system="http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd"/>
  <xsl:strip-space elements="*"/>
  <xsl:preserve-space
    elements="d:screen d:programlisting d:literallayout xi:*"/>
  <xsl:variable name="inlines">abbrev accel acronym alt anchor
    annotation application author biblioref citation citebiblioid
    citerefentry citetitle classname code command computeroutput
    constant coref database date editor email emphasis envar errorcode
    errorname errortext errortype exceptionname filename firstterm
    footnote footnoteref foreignphrase function glossterm guibutton
    guiicon guilabel guimenu guimenuitem guisubmenu hardware indexterm
    initializer inlineequation inlinemediaobject interfacename jobtitle
    keycap keycode keycombo keysym link literal markup menuchoice
    methodname modifier mousebutton nonterminal olink ooclass
    ooexception oointerface option optional org orgname package
    parameter person personname phrase productname productnumber prompt
    property quote remark replaceable returnvalue shortcut subscript
    superscript symbol systemitem tag termdef token trademark type uri
    userinput varname wordasword xref</xsl:variable>

  <!-- Overwrite standard template and create elements without
       a namespace node
  -->
  <xsl:template match="d:*">
    <xsl:element name="{local-name()}">
      <xsl:apply-templates select="@*|node()"/>
    </xsl:element>
```

```
</xsl:template>

<xsl:template match="@xml:id|@xml:lang">
  <xsl:attribute name="{local-name()}">
    <xsl:apply-templates/>
  </xsl:attribute>
</xsl:template>

<!-- Suppress the following attributes: -->
<xsl:template match="@annotations|@version"/>
<xsl:template match="@xlink:*"/>

<xsl:template match="@xlink:href">
  <xsl:choose>
    <xsl:when test="contains($inlines, local-name(..))">
      <ulink url="{.}" remap="{local-name(..)}">
        <xsl:value-of select=".."/>
      </ulink>
    </xsl:when>
    <xsl:otherwise>
      <xsl:message>@xlink:href could not be processed!
parent element: <xsl:value-of select="local-name(..)"/>
      </xsl:message>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="d:*[@xlink:href]">
  <xsl:choose>
    <xsl:when test="contains($inlines, local-name())">
      <ulink url="{@xlink:href}" remap="{local-name(.)}">
        <xsl:element name="{local-name()}">
          <xsl:apply-templates
            select="@*[local-name() != 'href' and
                       namespace-uri() != 'http://www.w3.org/1999/xlink']
                    |node()"/>
        </xsl:element>
      </ulink>
    </xsl:when>
    <xsl:otherwise>
      <xsl:element name="{local-name()}">
        <xsl:apply-templates
          select="@*[local-name() != 'href' and
                     namespace-uri() != 'http://www.w3.org/1999/xlink']
                  |node()"/>
      </xsl:element>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="d:link/@xlink:href">
  <xsl:attribute name="url">
    <xsl:value-of select="."/>
  </xsl:attribute>
</xsl:template>

<xsl:template match="d:link[@xlink:href]">
  <ulink>
    <xsl:apply-templates select="@*|node()"/>
  </ulink>
</xsl:template>
```

```
  <xsl:template match="d:link[@linkend]">
    <link>
      <xsl:apply-templates select="@*|node()"/>
    </link>
  </xsl:template>

  <!-- Renamed DocBook elements -->
  <xsl:template match="d:personblurb">
    <authorblurb>
      <xsl:apply-templates select="@*|node()"/>
    </authorblurb>
  </xsl:template>
  <xsl:template match="d:tag">
    <sgmltag>
      <xsl:apply-templates select="@*|node()"/>
    </sgmltag>
  </xsl:template>

  <!-- New DocBook v5.1 and HTML elements, no mapping available -->
  <xsl:template match="d:acknowledgements|d:annotation|d:arc
                       |d:cover
                       |d:definitions
                       |d:extendedlink
                       |d:givenname
                       |d:locator
                       |d:org|d:tocdiv
                       |html:*">
    <xsl:message>Don't know how to transfer "<xsl:value-of
      select="local-name()"/>" element into DocBook 4</xsl:message>
  </xsl:template>

  <xsl:template match="d:orgname">
     <othername>
       <xsl:apply-templates select="@*|node()"/>
     </othername>
   </xsl:template>

</xsl:stylesheet>
```

Use your favorite XSLT processor to transform your documents.

Discussion

The stylesheet from Example 3.3, "`db5to4-core.xsl`" [64] imports the templates from `copy.xsl` using an *identity transformation*. That means, whatever is *not* specified gets copied. In most cases that is what you want—a DocBook 5 `section` element will be transformed into an equally named DocBook 4 `section` element without the namespace.

Where it gets difficult are the new elements which are introduced in DocBook 5. Whenever the stylesheet encounters those it will print a warning. These elements are not copied to the output stream. If you have one of those elements you need to customize the stylesheet yourself.

Another issue is the almost ubiquitary `info` element which can appear in structual and block elements. The above stylesheet does it wrong and copies any `info` element straight into the output stream. However, DocBook 4 has different element names for meta information in DocBook 5. If you have `info` elements in your document they have to be renamed, depending on the parent element:

- **Meta Information Inside Structural Elements** An `info` element inside a `sect1` appends the suffix `info` to the name of its parent element and is renamed therefore as `sect1info`. This rule is applied for structural elements like `book`, `chapter`, and others.

- **Meta Information Inside Block Elements** An `info` element inside an `example` is renamed as `blockinfo`. This rule is applied for block elements like `equation`, `figure`, and others.

Apart from the renaming, the order of the renamed `info` element is crucial. Consider the following DocBook 5 structure:

```
section
  title
  info
```

This structure has to be renamed and reorganized as follows:

```
section
  sectioninfo
  title
```

As you can see, the `title` element appears now *after* the renamed `info`. All these issues are solved with the following additional stylesheet:

Example 3.4. `db5to4-info.xsl`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://docbook.org/ns/docbook"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:exsl="http://exslt.org/common"
  exclude-result-prefixes="d xlink exsl">

  <!-- Structural elements using info -->
  <xsl:template match="d:appendix[d:info]
                      |d:article[d:info]
                      |d:bibliography[d:info]
                      |d:book[d:info]
                      |d:chapter[d:info]
                      |d:colophon[d:info]
                      |d:equation[d:info]
                      |d:glossary[d:info]
                      |d:index[d:info]
                      |d:legalnotice[d:info]
                      |d:part[d:info]
                      |d:partintro[d:info]
                      |d:preface[d:info]
                      |d:reference[d:info]
                      |d:refsect1[d:info]
                      |d:refsect2[d:info]
                      |d:refsect3[d:info]
                      |d:refsection[d:info]
                      |d:refsynopsisdiv[d:info]
                      |d:sect1[d:info]
                      |d:sect2[d:info]
                      |d:sect3[d:info]
                      |d:sect4[d:info]
                      |d:sect5[d:info]
                      |d:section[d:info]
                      |d:set[d:info]
                      |d:setindex[d:info]">
    <!-- Change order of info and title  -->
    <xsl:element name="{local-name()}">
      <xsl:apply-templates select="@*"/>
      <xsl:apply-templates select="d:title/preceding-sibling::processing-instruction()
                                  |d:title/preceding-sibling::comment()"/>
      <xsl:apply-templates select="d:info"/>
      <xsl:apply-templates select="d:title"/>
```

```
    <!-- Process the rest -->
    <xsl:apply-templates select="d:info/following-sibling::node()"/>
  </xsl:element>
</xsl:template>

<!-- Block elements using info -->
<xsl:template match="d:bibliolist[d:info]
                     |d:blockquote[d:info]
                     |d:equation[d:info]
                     |d:example[d:info]
                     |d:figure[d:info]
                     |d:glosslist[d:info]
                     |d:informalequation[d:info]
                     |d:informalexample[d:info]
                     |d:informalfigure[d:info]
                     |d:informaltable[d:info]
                     |d:itemizedlist[d:info]
                     |d:legalnotice[d:info]
                     |d:msgset[d:info]
                     |d:orderedlist[d:info]
                     |d:procedure[d:info]
                     |d:qandadiv[d:info]
                     |d:qandaentry[d:info]
                     |d:qandaset[d:info]
                     |d:table[d:info]
                     |d:task[d:info]
                     |d:taskprerequisites[d:info]
                     |d:taskrelated[d:info]
                     |d:tasksummary[d:info]
                     |d:variablelist[d:info]">
  <xsl:element name="{local-name()}">
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates select="d:title/preceding-sibling::processing-instruction()
                                 |d:title/preceding-sibling::comment()"/>
    <xsl:apply-templates select="d:info">
      <xsl:with-param name="infoname">block</xsl:with-param>
    </xsl:apply-templates>
    <xsl:apply-templates select="d:title|
                                 d:title/following-sibling::processing-instruction()[1]
                                 |d:title/following-sibling::comment()[1]"/>

    <!-- Process the rest -->
    <xsl:apply-templates select="d:info/following-sibling::node()"/>
  </xsl:element>
</xsl:template>

<!-- Suppress other info elements who has no direct mapping -->
<xsl:template match="d:*[d:info]"/>

<xsl:template match="d:info">
  <xsl:param name="infoname" select="local-name(..)"/>
  <xsl:variable name="rtf-node">
    <xsl:element name="{$infoname}info">
    <xsl:apply-templates select="@*|node()"/>
  </xsl:element>
  </xsl:variable>
  <xsl:choose>
    <xsl:when test="count(exsl:node-set($rtf-node)/*/*) > 0">
      <xsl:copy-of select="$rtf-node"/>
    </xsl:when>
    <xsl:otherwise><!-- Don't copy, it's empty --></xsl:otherwise>
```

```
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

To combine both, use the following stylesheet:

Example 3.5. `db5to4-withinfo.xsl`

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

  <xsl:import href="db5to4-core.xsl"/>
  <xsl:import href="db5to4-info.xsl"/>

  <xsl:output method="xml" indent="yes"
    doctype-public="-//OASIS//DTD DocBook XML V4.5//EN"
    doctype-system="http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd"/>
</xsl:stylesheet>
```

The above stylesheets were separated between a core functionality (`db5to4-core.xsl`) with additional `info` element handling (`db5to4-info.xsl`). In most cases you will use the stylesheet `db5to4-withinfo.xsl`, but if you want to implement a different `info` handling you can. In Example 3.5, “`db5to4-withinfo.xsl`” [69] just replace the line with importing `db5to4-info.xsl` with your own implementation.

3.5. Splitting DocBook Documents

Problem

You have a big DocBook document, like a book, and you want to split each `chapter`, `appendix` etc. into a separate file.

Solution

There are three solutions to this problems:

- Apply a XSLT stylesheet which uses the extension element `exsl:document`
- Run the **dbautosplit** Perl script
- Use the assembly stylesheet `topic-maker-chunk.xsl` (see Section 3.14, “Splitting DocBook 5 Documents Into Topics” [107] for more information).

The following subsections describe each method.

Splitting with XSLT

The stylesheet in Example 3.6, “`dbsplit.xsl`” [70] uses the extension element to create separate documents from the main output. It uses code from `html/chunk-code.xsl`.

Example 3.6. `dbsplit.xsl`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet
[
 <!ENTITY db "https://cdn.docbook.org/release/xsl/current">
]>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://docbook.org/ns/docbook"
  xmlns:exsl="http://exslt.org/common"
  xmlns:xi="http://www.w3.org/2001/XInclude"
  extension-element-prefixes="exsl">

  <xsl:import href="chunker.xsl"/>
  <xsl:import href="copy.xsl"/>
  <xsl:output indent="yes"/>


  <xsl:param name="base.dir" select="'out/'"/>
  <xsl:param name="use.id.as.filename" select="0"/>
  <xsl:param name="rootid"/>

  <xsl:param name="dbsplit.root.filename">Index</xsl:param>
  <xsl:param name="dbsplit.ext">.xml</xsl:param>
  <xsl:param name="dbsplit.chunk.depth" select="2"/>

  <xsl:template name="object.id">
    <xsl:param name="object" select="."/>
    <xsl:choose>
      <xsl:when test="$object/@id">
        <xsl:value-of select="$object/@id"/>
      </xsl:when>
      <xsl:when test="$object/@xml:id">
        <xsl:value-of select="$object/@xml:id"/>
      </xsl:when>
```

```
    <xsl:otherwise>
      <xsl:value-of select="generate-id($object)"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="*" mode="recursive-chunk-filename">
  <xsl:param name="recursive" select="false()"/>
  <xsl:variable name="filename">
    <xsl:choose>
      <!-- if this is the root element, use the dbsplit.root.filename -->
      <xsl:when test="not(parent::*) and $dbsplit.root.filename != ''">
        <xsl:value-of select="$dbsplit.root.filename"/>
        <xsl:value-of select="$dbsplit.ext"/>
      </xsl:when>
      <!-- Special case -->
      <xsl:when
        test="self::d:legalnotice and not($generate.legalnotice.link = 0)">
        <xsl:choose>
          <xsl:when
            test="(@id or @xml:id) and not($use.id.as.filename = 0)">
            <!-- * if this legalnotice has an ID, then go ahead and use -->
            <!-- * just the value of that ID as the basename for the file -->
            <!-- * (that is, without prepending an "ln-" too it) -->
            <xsl:value-of select="(@id|@xml:id)[1]"/>
            <xsl:value-of select="$dbsplit.ext"/>
          </xsl:when>
          <xsl:otherwise>
            <!-- * otherwise, if this legalnotice does not have an ID, -->
            <!-- * then we generate an ID... -->
            <xsl:variable name="id">
              <xsl:call-template name="object.id"/>
            </xsl:variable>
            <!-- * ...and then we take that generated ID, prepend an -->
            <!-- * "ln-" to it, and use that as the basename for the file -->
            <xsl:value-of select="concat('ln-',$id,$dbsplit.ext)"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <!-- if there's no dbhtml filename, and if we're to use IDs as -->
      <!-- filenames, then use the ID to generate the filename. -->
      <xsl:when test="(@id or @xml:id) and $use.id.as.filename != 0">
        <xsl:value-of select="(@id|@xml:id)[1]"/>
        <xsl:value-of select="$dbsplit.ext"/>
      </xsl:when>
      <xsl:otherwise/>
    </xsl:choose>
  </xsl:variable>

  <xsl:choose>
    <xsl:when test="not($recursive) and $filename != ''">
      <!-- if this chunk has an explicit name, use it -->
      <xsl:value-of select="$filename"/>
    </xsl:when>

    <xsl:when test="self::d:set">
      <xsl:value-of select="$dbsplit.root.filename"/>
      <xsl:if test="not($recursive)">
        <xsl:value-of select="$dbsplit.ext"/>
      </xsl:if>
    </xsl:when>
```

```
<xsl:when test="self::d:book">
  <xsl:text>bk</xsl:text>
  <xsl:number level="any" format="01"/>
  <xsl:if test="not($recursive)">
    <xsl:value-of select="$dbsplit.ext"/>
  </xsl:if>
</xsl:when>

<xsl:when test="self::d:article">
  <xsl:if test="/d:set">
    <!-- in a set, make sure we inherit the right book info... -->
    <xsl:apply-templates mode="recursive-chunk-filename"
      select="parent::*">
      <xsl:with-param name="recursive" select="true()"/>
    </xsl:apply-templates>
  </xsl:if>

  <xsl:text>ar</xsl:text>
  <xsl:number level="any" format="01" from="d:book"/>
  <xsl:if test="not($recursive)">
    <xsl:value-of select="$dbsplit.ext"/>
  </xsl:if>
</xsl:when>

<xsl:when test="self::d:preface">
  <xsl:if test="/d:set">
    <!-- in a set, make sure we inherit the right book info... -->
    <xsl:apply-templates mode="recursive-chunk-filename"
      select="parent::*">
      <xsl:with-param name="recursive" select="true()"/>
    </xsl:apply-templates>
  </xsl:if>

  <xsl:text>pr</xsl:text>
  <xsl:number level="any" format="01" from="d:book"/>
  <xsl:if test="not($recursive)">
    <xsl:value-of select="$dbsplit.ext"/>
  </xsl:if>
</xsl:when>

<xsl:when test="self::d:chapter">
  <xsl:if test="/d:set">
    <!-- in a set, make sure we inherit the right book info... -->
    <xsl:apply-templates mode="recursive-chunk-filename"
      select="parent::*">
      <xsl:with-param name="recursive" select="true()"/>
    </xsl:apply-templates>
  </xsl:if>

  <xsl:text>ch</xsl:text>
  <xsl:number level="any" format="01" from="d:book"/>
  <xsl:if test="not($recursive)">
    <xsl:value-of select="$dbsplit.ext"/>
  </xsl:if>
</xsl:when>

<xsl:when test="self::d:appendix">
  <xsl:if test="/d:set">
    <!-- in a set, make sure we inherit the right book info... -->
    <xsl:apply-templates mode="recursive-chunk-filename"
```

```
      select="parent::*">
      <xsl:with-param name="recursive" select="true()"/>
    </xsl:apply-templates>
  </xsl:if>

  <xsl:text>ap</xsl:text>
  <xsl:number level="any" format="a" from="d:book"/>
  <xsl:if test="not($recursive)">
    <xsl:value-of select="$dbsplit.ext"/>
  </xsl:if>
</xsl:when>

<xsl:when test="self::d:part">
  <xsl:choose>
    <xsl:when test="/d:set">
      <!-- in a set, make sure we inherit the right book info... -->
      <xsl:apply-templates mode="recursive-chunk-filename"
        select="parent::*">
        <xsl:with-param name="recursive" select="true()"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise> </xsl:otherwise>
  </xsl:choose>

  <xsl:text>pt</xsl:text>
  <xsl:number level="any" format="01" from="d:book"/>
  <xsl:if test="not($recursive)">
    <xsl:value-of select="$dbsplit.ext"/>
  </xsl:if>
</xsl:when>

<xsl:when test="self::d:reference">
  <xsl:choose>
    <xsl:when test="/d:set">
      <!-- in a set, make sure we inherit the right book info... -->
      <xsl:apply-templates mode="recursive-chunk-filename"
        select="parent::*">
        <xsl:with-param name="recursive" select="true()"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise> </xsl:otherwise>
  </xsl:choose>

  <xsl:text>rn</xsl:text>
  <xsl:number level="any" format="01" from="d:book"/>
  <xsl:if test="not($recursive)">
    <xsl:value-of select="$dbsplit.ext"/>
  </xsl:if>
</xsl:when>

<xsl:when test="self::d:refentry">
  <xsl:choose>
    <xsl:when test="parent::d:reference">
      <xsl:apply-templates mode="recursive-chunk-filename"
        select="parent::*">
        <xsl:with-param name="recursive" select="true()"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
      <xsl:if test="/d:set">
        <!-- in a set, make sure we inherit the right book info... -->
```

```
        <xsl:apply-templates mode="recursive-chunk-filename"
          select="parent::*">
          <xsl:with-param name="recursive" select="true()"/>
        </xsl:apply-templates>
      </xsl:if>
    </xsl:otherwise>
  </xsl:choose>

  <xsl:text>re</xsl:text>
  <xsl:number level="any" format="01" from="d:book"/>
  <xsl:if test="not($recursive)">
    <xsl:value-of select="$dbsplit.ext"/>
  </xsl:if>
</xsl:when>

<xsl:when test="self::d:colophon">
  <xsl:choose>
    <xsl:when test="/d:set">
      <!-- in a set, make sure we inherit the right book info... -->
      <xsl:apply-templates mode="recursive-chunk-filename"
        select="parent::*">
        <xsl:with-param name="recursive" select="true()"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise> </xsl:otherwise>
  </xsl:choose>

  <xsl:text>co</xsl:text>
  <xsl:number level="any" format="01" from="d:book"/>
  <xsl:if test="not($recursive)">
    <xsl:value-of select="$dbsplit.ext"/>
  </xsl:if>
</xsl:when>

<xsl:when test="self::d:sect1 or self::d:sect2 or self::d:sect3 or
                self::d:sect4 or self::d:sect5 or self::d:section">
  <xsl:apply-templates mode="recursive-chunk-filename"
    select="parent::*">
    <xsl:with-param name="recursive" select="true()"/>
  </xsl:apply-templates>
  <xsl:text>s</xsl:text>
  <xsl:number format="01"/>
  <xsl:if test="not($recursive)">
    <xsl:value-of select="$dbsplit.ext"/>
  </xsl:if>
</xsl:when>

<xsl:when test="self::d:bibliography">
  <xsl:choose>
    <xsl:when test="/d:set">
      <!-- in a set, make sure we inherit the right book info... -->
      <xsl:apply-templates mode="recursive-chunk-filename"
        select="parent::*">
        <xsl:with-param name="recursive" select="true()"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise> </xsl:otherwise>
  </xsl:choose>

  <xsl:text>bi</xsl:text>
  <xsl:number level="any" format="01" from="d:book"/>
```

```
      <xsl:if test="not($recursive)">
        <xsl:value-of select="$dbsplit.ext"/>
      </xsl:if>
    </xsl:when>

    <xsl:when test="self::d:glossary">
      <xsl:choose>
        <xsl:when test="/d:set">
          <!-- in a set, make sure we inherit the right book info... -->
          <xsl:apply-templates mode="recursive-chunk-filename"
            select="parent::*">
            <xsl:with-param name="recursive" select="true()"/>
          </xsl:apply-templates>
        </xsl:when>
        <xsl:otherwise> </xsl:otherwise>
      </xsl:choose>

      <xsl:text>go</xsl:text>
      <xsl:number level="any" format="01" from="d:book"/>
      <xsl:if test="not($recursive)">
        <xsl:value-of select="$dbsplit.ext"/>
      </xsl:if>
    </xsl:when>

    <xsl:when test="self::d:index">
      <xsl:choose>
        <xsl:when test="/d:set">
          <!-- in a set, make sure we inherit the right book info... -->
          <xsl:apply-templates mode="recursive-chunk-filename"
            select="parent::*">
            <xsl:with-param name="recursive" select="true()"/>
          </xsl:apply-templates>
        </xsl:when>
        <xsl:otherwise> </xsl:otherwise>
      </xsl:choose>

      <xsl:text>ix</xsl:text>
      <xsl:number level="any" format="01" from="d:book"/>
      <xsl:if test="not($recursive)">
        <xsl:value-of select="$dbsplit.ext"/>
      </xsl:if>
    </xsl:when>

    <xsl:when test="self::d:setindex">
      <xsl:text>si</xsl:text>
      <xsl:number level="any" format="01" from="d:set"/>
      <xsl:if test="not($recursive)">
        <xsl:value-of select="$dbsplit.ext"/>
      </xsl:if>
    </xsl:when>

    <xsl:otherwise>
      <xsl:text>chunk-filename-error-</xsl:text>
      <xsl:value-of select="name(.)"/>
      <xsl:number level="any" format="01" from="d:set"/>
      <xsl:if test="not($recursive)">
        <xsl:value-of select="$dbsplit.ext"/>
      </xsl:if>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

```
  <xsl:template name="generate-filename">
    <xsl:apply-templates select="." mode="recursive-chunk-filename"/>
  </xsl:template>

  <xsl:template name="generate-content">
    <xsl:variable name="filename">
      <xsl:call-template name="make-relative-filename">
        <xsl:with-param name="base.dir" select="$base.dir"/>
        <xsl:with-param name="base.name">
           <xsl:call-template name="generate-filename"/>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:variable>
    <xsl:variable name="depth" select="count(ancestor::*)"/>

    <!--<xsl:message>generate-content:
      name: <xsl:value-of select="name()"/>
      filename: <xsl:value-of select="$filename"/>
      depth: <xsl:value-of select="$depth"/>
      test: <xsl:value-of select="$depth >= $dbsplit.chunk.depth"/>
    </xsl:message>-->

    <xsl:choose>
      <xsl:when test="$depth &lt;= $dbsplit.chunk.depth">
        <xi:include href="{$filename}"/>
        <xsl:call-template name="write.xml.chunk">
          <xsl:with-param name="filename" select="$filename"/>
          <xsl:with-param name="content">
            <xsl:copy-of select="preceding-sibling::processing-instruction()|
                                 preceding-sibling::comment()"/>
            <xsl:copy>
              <xsl:apply-templates/>
            </xsl:copy>
          </xsl:with-param>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:copy>
          <xsl:apply-templates/>
        </xsl:copy>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template
    match="d:acknowledgements|d:appendix|d:article|
           d:bibliography|
           d:chapter|d:colophon|d:dedication|d:glossary|
           d:part|d:preface|d:reference|d:topic|
           d:sect1|d:section[not(parent::d:section)]">
    <xsl:call-template name="generate-content"/>
  </xsl:template>

</xsl:stylesheet>
```

The `chunker.xsl` file is more or less a stripped down version of the `html/chunker.xsl` file from the DocBook XSL stylesheets. It is used to create processor independent writing of our XML chunks. The file was integrated into `dbsplit.xsl` to make it more independent from the DocBook XSL stylesheets.

Example 3.7. `chunker.xsl`

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:saxon="http://icl.com/saxon"
  xmlns:lxslt="http://xml.apache.org/xslt"
  xmlns:redirect="http://xml.apache.org/xalan/redirect"
  xmlns:exsl="http://exslt.org/common"
  xmlns="http://www.w3.org/1999/xhtml" version="1.0"
  exclude-result-prefixes="saxon lxslt redirect exsl"
  extension-element-prefixes="saxon redirect lxslt exsl">

  <xsl:param name="chunker.output.method" select="'xml'"/>
  <xsl:param name="chunker.output.encoding" select="'UTF-8'"/>
  <xsl:param name="chunker.output.indent" select="'no'"/>
  <xsl:param name="chunker.output.omit-xml-declaration" select="'no'"/>
  <xsl:param name="chunker.output.standalone" select="'no'"/>
  <xsl:param name="chunker.output.doctype-public" select="''"/>
  <xsl:param name="chunker.output.doctype-system" select="''"/>
  <xsl:param name="chunker.output.media-type" select="''"/>
  <xsl:param name="chunker.output.cdata-section-elements" select="''"/>
  <xsl:param name="chunker.output.quiet" select="0"/>

  <xsl:param name="saxon.character.representation"
    select="'entity;decimal'"/>

  <xsl:template name="make-relative-filename">
    <xsl:param name="base.dir" select="'./'"/>
    <xsl:param name="base.name" select="''"/>

    <xsl:choose>
      <!-- put Saxon first to work around a bug in libxslt -->
      <xsl:when test="element-available('saxon:output')">
        <!-- Saxon doesn't make the chunks relative -->
        <xsl:value-of select="concat($base.dir,$base.name)"/>
      </xsl:when>
      <xsl:when test="element-available('exsl:document')">
        <!-- EXSL document does make the chunks relative, I think -->
        <xsl:choose>
          <xsl:when test="count(parent::*) = 0">
            <xsl:value-of select="concat($base.dir,$base.name)"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="$base.name"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:when test="element-available('redirect:write')">
        <!-- Xalan doesn't make the chunks relative -->
        <xsl:value-of select="concat($base.dir,$base.name)"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:message terminate="yes">
          <xsl:text>Don't know how to chunk with </xsl:text>
          <xsl:value-of select="system-property('xsl:vendor')"/>
        </xsl:message>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template name="write.chunk">
    <xsl:param name="filename" select="''"/>
```

```
<xsl:param name="quiet" select="$chunker.output.quiet"/>
<xsl:param name="suppress-context-node-name" select="0"/>
<xsl:param name="message-prolog"/>
<xsl:param name="message-epilog"/>

<xsl:param name="method" select="$chunker.output.method"/>
<xsl:param name="encoding" select="$chunker.output.encoding"/>
<xsl:param name="indent" select="$chunker.output.indent"/>
<xsl:param name="omit-xml-declaration"
  select="$chunker.output.omit-xml-declaration"/>
<xsl:param name="standalone" select="$chunker.output.standalone"/>
<xsl:param name="doctype-public"
  select="$chunker.output.doctype-public"/>
<xsl:param name="doctype-system"
  select="$chunker.output.doctype-system"/>
<xsl:param name="media-type" select="$chunker.output.media-type"/>
<xsl:param name="cdata-section-elements"
  select="$chunker.output.cdata-section-elements"/>

<xsl:param name="content"/>

<xsl:if test="$quiet = 0">
  <xsl:message>
    <xsl:if test="not($message-prolog = '')">
      <xsl:value-of select="$message-prolog"/>
    </xsl:if>
    <xsl:text>Writing </xsl:text>
    <xsl:value-of select="$filename"/>
    <xsl:if test="name(.) != '' and $suppress-context-node-name = 0">
      <xsl:text> for </xsl:text>
      <xsl:value-of select="name(.)"/>
      <xsl:if test="@id or @xml:id">
        <xsl:text>(</xsl:text>
        <xsl:value-of select="(@id|@xml:id)[1]"/>
        <xsl:text>)</xsl:text>
      </xsl:if>
    </xsl:if>
    <xsl:if test="not($message-epilog = '')">
      <xsl:value-of select="$message-epilog"/>
    </xsl:if>
  </xsl:message>
</xsl:if>

<xsl:choose>
  <xsl:when test="element-available('exsl:document')">
    <xsl:choose>
      <!-- Handle the permutations ... -->
      <xsl:when test="$media-type != ''">
        <xsl:choose>
          <xsl:when
            test="$doctype-public != '' and $doctype-system != ''">
            <exsl:document href="{$filename}" method="{$method}"
              encoding="{$encoding}" indent="{$indent}"
              omit-xml-declaration="{$omit-xml-declaration}"
              cdata-section-elements="{$cdata-section-elements}"
              media-type="{$media-type}"
              doctype-public="{$doctype-public}"
              doctype-system="{$doctype-system}"
              standalone="{$standalone}">
              <xsl:copy-of select="$content"/>
            </exsl:document>
```

```
      </xsl:when>
      <xsl:when
        test="$doctype-public != '' and $doctype-system = ''">
        <exsl:document href="{$filename}" method="{$method}"
          encoding="{$encoding}" indent="{$indent}"
          omit-xml-declaration="{$omit-xml-declaration}"
          cdata-section-elements="{$cdata-section-elements}"
          media-type="{$media-type}"
          doctype-public="{$doctype-public}"
          standalone="{$standalone}">
          <xsl:copy-of select="$content"/>
        </exsl:document>
      </xsl:when>
      <xsl:when
        test="$doctype-public = '' and $doctype-system != ''">
        <exsl:document href="{$filename}" method="{$method}"
          encoding="{$encoding}" indent="{$indent}"
          omit-xml-declaration="{$omit-xml-declaration}"
          cdata-section-elements="{$cdata-section-elements}"
          media-type="{$media-type}"
          doctype-system="{$doctype-system}"
          standalone="{$standalone}">
          <xsl:copy-of select="$content"/>
        </exsl:document>
      </xsl:when>
      <xsl:otherwise>
        <!-- $doctype-public = '' and $doctype-system = ''"> -->
        <exsl:document href="{$filename}" method="{$method}"
          encoding="{$encoding}" indent="{$indent}"
          omit-xml-declaration="{$omit-xml-declaration}"
          cdata-section-elements="{$cdata-section-elements}"
          media-type="{$media-type}" standalone="{$standalone}">
          <xsl:copy-of select="$content"/>
        </exsl:document>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:when>
  <xsl:otherwise>
    <xsl:choose>
      <xsl:when
        test="$doctype-public != '' and $doctype-system != ''">
        <exsl:document href="{$filename}" method="{$method}"
          encoding="{$encoding}" indent="{$indent}"
          omit-xml-declaration="{$omit-xml-declaration}"
          cdata-section-elements="{$cdata-section-elements}"
          doctype-public="{$doctype-public}"
          doctype-system="{$doctype-system}"
          standalone="{$standalone}">
          <xsl:copy-of select="$content"/>
        </exsl:document>
      </xsl:when>
      <xsl:when
        test="$doctype-public != '' and $doctype-system = ''">
        <exsl:document href="{$filename}" method="{$method}"
          encoding="{$encoding}" indent="{$indent}"
          omit-xml-declaration="{$omit-xml-declaration}"
          cdata-section-elements="{$cdata-section-elements}"
          doctype-public="{$doctype-public}"
          standalone="{$standalone}">
          <xsl:copy-of select="$content"/>
        </exsl:document>
```

```
          </xsl:when>
          <xsl:when
            test="$doctype-public = '' and $doctype-system != ''">
            <exsl:document href="{$filename}" method="{$method}"
              encoding="{$encoding}" indent="{$indent}"
              omit-xml-declaration="{$omit-xml-declaration}"
              cdata-section-elements="{$cdata-section-elements}"
              doctype-system="{$doctype-system}"
              standalone="{$standalone}">
              <xsl:copy-of select="$content"/>
            </exsl:document>
          </xsl:when>
          <xsl:otherwise>
            <!-- $doctype-public = '' and $doctype-system = ''"> -->
            <exsl:document href="{$filename}" method="{$method}"
              encoding="{$encoding}" indent="{$indent}"
              omit-xml-declaration="{$omit-xml-declaration}"
              cdata-section-elements="{$cdata-section-elements}"
              standalone="{$standalone}">
              <xsl:copy-of select="$content"/>
            </exsl:document>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:when>

  <xsl:when test="element-available('saxon:output')">
    <xsl:choose>
      <!-- Handle the permutations ... -->
      <xsl:when test="$media-type != ''">
        <xsl:choose>
          <xsl:when
            test="$doctype-public != '' and $doctype-system != ''">
            <saxon:output
              saxon:character-representation="{$saxon.character.representation}"
              href="{$filename}" method="{$method}"
              encoding="{$encoding}" indent="{$indent}"
              omit-xml-declaration="{$omit-xml-declaration}"
              cdata-section-elements="{$cdata-section-elements}"
              media-type="{$media-type}"
              doctype-public="{$doctype-public}"
              doctype-system="{$doctype-system}"
              standalone="{$standalone}">
              <xsl:copy-of select="$content"/>
            </saxon:output>
          </xsl:when>
          <xsl:when
            test="$doctype-public != '' and $doctype-system = ''">
            <saxon:output
              saxon:character-representation="{$saxon.character.representation}"
              href="{$filename}" method="{$method}"
              encoding="{$encoding}" indent="{$indent}"
              omit-xml-declaration="{$omit-xml-declaration}"
              cdata-section-elements="{$cdata-section-elements}"
              media-type="{$media-type}"
              doctype-public="{$doctype-public}"
              standalone="{$standalone}">
              <xsl:copy-of select="$content"/>
            </saxon:output>
          </xsl:when>
```

```
        <xsl:when
          test="$doctype-public = '' and $doctype-system != ''">
          <saxon:output
            saxon:character-representation="{$saxon.character.representation}"
            href="{$filename}" method="{$method}"
            encoding="{$encoding}" indent="{$indent}"
            omit-xml-declaration="{$omit-xml-declaration}"
            cdata-section-elements="{$cdata-section-elements}"
            media-type="{$media-type}"
            doctype-system="{$doctype-system}"
            standalone="{$standalone}">
            <xsl:copy-of select="$content"/>
          </saxon:output>
        </xsl:when>
        <xsl:otherwise>
          <!-- $doctype-public = '' and $doctype-system = ''"> -->
          <saxon:output
            saxon:character-representation="{$saxon.character.representation}"
            href="{$filename}" method="{$method}"
            encoding="{$encoding}" indent="{$indent}"
            omit-xml-declaration="{$omit-xml-declaration}"
            cdata-section-elements="{$cdata-section-elements}"
            media-type="{$media-type}" standalone="{$standalone}">
            <xsl:copy-of select="$content"/>
          </saxon:output>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when
          test="$doctype-public != '' and $doctype-system != ''">
          <saxon:output
            saxon:character-representation="{$saxon.character.representation}"
            href="{$filename}" method="{$method}"
            encoding="{$encoding}" indent="{$indent}"
            omit-xml-declaration="{$omit-xml-declaration}"
            cdata-section-elements="{$cdata-section-elements}"
            doctype-public="{$doctype-public}"
            doctype-system="{$doctype-system}"
            standalone="{$standalone}">
            <xsl:copy-of select="$content"/>
          </saxon:output>
        </xsl:when>
        <xsl:when
          test="$doctype-public != '' and $doctype-system = ''">
          <saxon:output
            saxon:character-representation="{$saxon.character.representation}"
            href="{$filename}" method="{$method}"
            encoding="{$encoding}" indent="{$indent}"
            omit-xml-declaration="{$omit-xml-declaration}"
            cdata-section-elements="{$cdata-section-elements}"
            doctype-public="{$doctype-public}"
            standalone="{$standalone}">
            <xsl:copy-of select="$content"/>
          </saxon:output>
        </xsl:when>
        <xsl:when
          test="$doctype-public = '' and $doctype-system != ''">
          <saxon:output
            saxon:character-representation="{$saxon.character.representation}"
```

```
                href="{$filename}" method="{$method}"
                encoding="{$encoding}" indent="{$indent}"
                omit-xml-declaration="{$omit-xml-declaration}"
                cdata-section-elements="{$cdata-section-elements}"
                doctype-system="{$doctype-system}"
                standalone="{$standalone}">
                <xsl:copy-of select="$content"/>
              </saxon:output>
            </xsl:when>
            <xsl:otherwise>
              <!-- $doctype-public = '' and $doctype-system = ''"> -->
              <saxon:output
                saxon:character-representation="{$saxon.character.representation}"
                href="{$filename}" method="{$method}"
                encoding="{$encoding}" indent="{$indent}"
                omit-xml-declaration="{$omit-xml-declaration}"
                cdata-section-elements="{$cdata-section-elements}"
                standalone="{$standalone}">
                <xsl:copy-of select="$content"/>
              </saxon:output>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>

    <xsl:when test="element-available('redirect:write')">
      <!-- Xalan uses redirect -->
      <redirect:write file="{$filename}">
        <xsl:copy-of select="$content"/>
      </redirect:write>
    </xsl:when>

    <xsl:otherwise>
      <!-- it doesn't matter since we won't be making chunks... -->
      <xsl:message terminate="yes">
        <xsl:text>Can't make chunks with </xsl:text>
        <xsl:value-of select="system-property('xsl:vendor')"/>
        <xsl:text>'s processor.</xsl:text>
      </xsl:message>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="write.xml.chunk">
  <xsl:param name="filename" select="''"/>
  <xsl:param name="quiet" select="$chunker.output.quiet"/>
  <xsl:param name="suppress-context-node-name" select="0"/>
  <xsl:param name="message-prolog"/>
  <xsl:param name="message-epilog"/>
  <xsl:param name="method" select="'xml'"/>
  <xsl:param name="encoding" select="$chunker.output.encoding"/>
  <xsl:param name="media-type" select="$chunker.output.media-type"/>
  <xsl:param name="content"/>
  <xsl:call-template name="write.chunk">
    <xsl:with-param name="filename" select="$filename"/>
    <xsl:with-param name="quiet" select="$quiet"/>
    <xsl:with-param name="suppress-context-node-name"
      select="$suppress-context-node-name"/>
    <xsl:with-param name="message-prolog" select="$message-prolog"/>
    <xsl:with-param name="message-epilog" select="$message-epilog"/>
```

```
      <xsl:with-param name="method" select="$method"/>
      <xsl:with-param name="encoding" select="$encoding"/>
      <xsl:with-param name="indent" select="'yes'"/>
      <xsl:with-param name="omit-xml-declaration" select="'no'"/>
      <xsl:with-param name="standalone" select="'no'"/>
      <xsl:with-param name="doctype-public"/>
      <xsl:with-param name="doctype-system"/>
      <xsl:with-param name="media-type" select="$media-type"/>
      <xsl:with-param name="cdata-section-elements"/>
      <xsl:with-param name="content" select="$content"/>
    </xsl:call-template>
  </xsl:template>
</xsl:stylesheet>
```

It can be customized with several parameters:

base.dir
: Determines the output directory

dbsplit.chunk.depth
: Controls the depth where to split

dbsplit.ext
: Defines the file extension for each filename written.

dbsplit.root.filename
: Identifies the name of the root filename when splitted

use.id.as.filename
: Uses ID values as filenames

rootid
: Specify the root element to split

Splitting with the dbautosplit

The **dbautosplit** command is a Perl [http://www.perl.org] script which is only be available in openSUSE Linux [http://www.opensuse.org] distribution. As such, you can search for the command **dbautosplit** [http://software.opensuse.org/package/dbsplit-tools?search_term=dbsplit-tools] or download it from http://rpmfind.net/linux/rpm2html/search.php?query=dbautosplit. Additionaly it needs the XML-DOM and XML-RegExp packages. Download and install these package from CPAN [http://www.cpan.org], the Comprehensive Perl Archive Network, before you proceed.

For example, if you have a book with a preface and five chapters, use the following command to split it:

```
dbautosplit --level 1 MyBook.xml
```

It will generate this output:

```
Creating file : out/preface-01/index.xml
Creating file : out/chapter-01/index.xml
Creating file : out/chapter-02/index.xml
Creating file : out/chapter-03/index.xml
Creating file : out/chapter-04/index.xml
Creating file : out/chapter-05/index.xml
Creating file : out/index.xml
```

By default, **dbautosplit** creates an `out/` directory. Each preface and chapter is stored in a separate subdirectory, named `preface` or `chapter` and with a consecutive number. The book itself is saved with `xi:include` elements in `out/index.xml`.

You can change the output behavior by using a template system similar to `printf` function (see manpage). If you want to name the chapters by their `xml:id` attribute, use the following commandline:

```
dbautosplit -l 1 -o "%attr(xml:id)%txt(.xml)" MyBook.xml
```

A possible output could look like this:

```
Creating file : out/preface.xml
Creating file : out/markup.xml
Creating file : out/common.xml
Creating file : out/structure.xml
Creating file : out/fo.xml
Creating file : out/html.xml
Creating file : out/index.xml
```

Discussion

The shown methods have both advantages and drawbacks.

Table 3.1. Comparison of Splitting Method

	splitdb/chunker.xsl	**dbautosplit**	**topic-maker-chunk.xsl**
Change Splitting Level?	yes	yes	yes
Special Support for Assemblies?	no	no	yes

See Also

- Section 1.3, “Modularize Your Document with XIncludes” [5]
- Section 3.14, “Splitting DocBook 5 Documents Into Topics” [107]
- Section 3.15, “Assembling Topics” [109]

3.6. Extracting One Element from DocBook Document

Problem

You have a big DocBook document and you need to extract one structural element like a `chapter`, `appendix` etc. to edit or process it separately from the main document.

Solution

To make the solution work, the structural element needs an ID attribute. If this is available, use the following stylesheet:

Example 3.8. Extracting Stylesheet `rootid.xsl`

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

  <xsl:key name="id" match="*" use="@id|@xml:id"/>
  <!-- Contains the ID attribute of the extracted element: -->
  <xsl:param name="rootid"/>
  <!-- Controls some log messages: 0=off, 1=on -->
  <xsl:param name="rootid.debug" select="0"/>

  <xsl:template match="/">
    <xsl:choose>
      <xsl:when test="$rootid !=''">
        <xsl:if test="count(key('id',$rootid)) = 0">
          <xsl:message terminate="yes">
            <xsl:text>ID '</xsl:text>
            <xsl:value-of select="$rootid"/>
            <xsl:text>' not found in document.</xsl:text>
          </xsl:message>
        </xsl:if>
        <xsl:call-template name="rootid.debug.message"/>
        <xsl:call-template name="rootid.process"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:call-template name="normal.process"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template name="rootid.debug.message">
    <xsl:if test="$rootid.debug != 0">
      <xsl:message>
        <xsl:text>Using ID </xsl:text>
        <xsl:value-of select="concat('&quot;', $rootid, '&quot;')"/>
      </xsl:message>
    </xsl:if>
  </xsl:template>

  <xsl:template name="rootid.process">
    <xsl:apply-templates select="key('id',$rootid)" mode="process.root"/>
  </xsl:template>

  <xsl:template name="normal.process">
    <xsl:apply-templates/>
```

```
  </xsl:template>

  <xsl:template match="node() | @*" mode="process.root">
    <xsl:copy>
      <xsl:apply-templates select="@* | node()" mode="process.root"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Pass the *`rootid`* parameter to your XSLT processor with the corresponding ID, for example:

```
xsltproc --stringparam rootid intro rootid.xsl XML_FILE
```

The result contains only the element with the corresponding ID value and everything inside it.

Discussion

This solution cuts off the element with the corresponding ID and copies the element itself and its children to the output stream. The copying is done in the `process.root` mode. The stylesheet does not apply any further processing. This can be a disadvantage, for example, a `xref` pointing outside of the respective element. If the resulting file contains such a cross reference, it will not be valid anymore.

It is possible to convert such cross references into a “resolved form” by using the following code:

Example 3.9. `rootid-resolve-xrefs.xsl`

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

  <xsl:import href="rootid.xsl"/>

  <xsl:template match="d:xref" mode="process.root">
    <xsl:variable name="xhref" select="@xlink:href"/>
    <!-- is the @xlink:href a local idref link? -->
    <xsl:variable name="xlink.idref">
      <xsl:choose>
        <xsl:when test="starts-with($xhref,'#')">
          <xsl:value-of select="substring($xhref, 2)"/>
        </xsl:when>
        <xsl:when test="contains($xhref, '://')">
         <xsl:message>
            <xsl:text>ERROR: Don't know what do do with @xlink:href: </xsl:text>
            <xsl:value-of select="$xhref"/></xsl:message>
        </xsl:when>
        <xsl:otherwise/>
      </xsl:choose>
    </xsl:variable>
    <xsl:variable name="xlink.targets" select="key('id',$xlink.idref)"/>
    <xsl:variable name="linkend.targets" select="key('id',@linkend)"/>
    <xsl:variable name="target" select="($xlink.targets | $linkend.targets)[1]"/>
    <xsl:variable name="refelem" select="local-name($target)"/>

    <xsl:variable name="this.div"
      select="ancestor-or-self::d:*[@xml:id = $rootid][1]"/>
    <xsl:variable name="target.div"
      select="$target/ancestor-or-self::d:*[@xml:id = $rootid][1]"/>

    <xsl:choose>
```

```
      <xsl:when test="generate-id($this.div) = generate-id($target.div)">
        <xsl:copy-of select="."/>
      </xsl:when>
      <xsl:otherwise>
        <phrase xmlns="http://docbook.org/ns/docbook" remap="xref">
          <xsl:choose>
            <xsl:when test="@linkend">
              <xsl:attribute name="role">
                <xsl:value-of select="@linkend"/>
              </xsl:attribute>
            </xsl:when>
            <xsl:when test="$xlink.idref != ''">
              <xsl:attribute name="role">
                <xsl:value-of select="$xlink.idref"/>
              </xsl:attribute>
            </xsl:when>
            <xsl:otherwise>
              <xsl:attribute name="role">
                <xsl:value-of select="$xhref"/>
              </xsl:attribute>
            </xsl:otherwise>
          </xsl:choose>
          <xsl:apply-templates
            select="@*[local-name() != 'linkend' and
                       local-name() != 'href']"
            mode="process.root"/>
          <xsl:apply-templates
            select="($target/ancestor-or-self::d:*[d:title])[last()]/d:title/node()"
            mode="process.root"/>
        </phrase>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

The stylesheet in Example 3.9, “`rootid-resolve-xrefs.xsl`” [86] imports the `rootid.xsl` and inherits all templates. To implement a different behaviour we need to add a new template matching for the `xref` element in mode `process.root`.

The template contains mostly code from the DocBook XSL stylesheets with some minor changes. The general behaviour is described in the following sequence:

1. Make sure, everything is in place and a `xlink:href` attribute does not contain a `://` string. If this is the case emit an error message.

2. Populate the variables `xlink.targets` and `linkend.targets` with the target node. For `xlink.targets` use the XLink attribute `xlink:href`, for the variable `linkend.targets` use the `linkend` attribute. As only one of these attributes can be available, but not both, the variables are filled with zero nodes or more.

3. Create the set union of the variables `xlink.targets` and `linkend.targets` and select only one node.

4. Now it gets interesting: our context node is in `xref`. We need to know the node where the value of the `xml:id` attribute equals our *`rootid`* parameter. We climb up tree with the `ancestor-or-self` axis specifier and select every DocBook element. With the predicate [@xml:id = $rootid] the node set is filtered and only those element(s) are preserved where this expression is true. Only one node from the node set is selected.

   This is done also for the target node and the result is saved in the variable `target.div`

5. The two node from the previous operation are compared through the `generate-id` function. That leaves two options:

   - **Both nodes are equal** The `xref` points somewhere inside the tree under the *`rootid`* element. That means, we can copy the `xref` element.

- **Both nodes are not equal** The `xref` points outside of the *`rootid`* element. That means, you need to “resolve” the `xref` element to prevent validation errors.

6. If the `xref` needs to be revamped, we use the `phrase` element, copy all attributes (except `linkend` and `xlink:href`), and copy anything inside the title of the target node. As the target node could not be a title itself, we use again the `ancestor-or-self` axis to climb up the tree and select the first emerging title.

TODO: Add graphic to illustrate the method

See Also

Section 3.5, “Splitting DocBook Documents” [70] uses a different approach without needing an ID attribute.

3.7. Transforming sectX Elements into section Elements

Problem

You need to transform every sectX element into a section element.

Solution

This problem is solved through the following XSLT stylesheet:

Example 3.10. Transforms every sectX Element into a section Element

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

  <xsl:import href="copy.xsl"/>

  <xsl:template match="d:sect1|d:sect2|d:sect3|d:sect4|d:sect5">
    <xsl:element name="section" namespace="http://docbook.org/ns/docbook">
      <xsl:apply-templates select="node()"/>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Discussion

The stylesheet from Example 3.10, “Transforms every sectX Element into a section Element” [89] is very easy: it imports the standard rules to copy every node from copy.xsl and create special rules for all sectX elements. As every sectX element creates the same structure, we can collect it in one template rule.

3.8. Transforming section Elements into sectX Elements

Problem

You need to transform every sectX element into a section element.

Solution

This problem is solved through the following XSLT stylesheet:

Example 3.11. Transforms every sectX Element into a section Element

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

  <xsl:import href="copy.xsl"/>

  <xsl:template match="d:section">
    <xsl:variable name="level" select="count(ancestor::*)"/>
    <xsl:element name="sect{$level}" namespace="http://docbook.org/ns/docbook">
      <xsl:copy-of select="@*"/>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

Discussion

The above stylesheet calculates the section level with the ancestor axis, because the amount of elements is directly correlated with the level of the corresponding section. With an *attribute value template [http://www.w3.org/TR/xslt#attribute-value-templates]* it is inserted in the name attribute.

There is one caveat: The stylesheet does not check if the limit is reached. Currently (with version 5.1), DocBook supports levels up to 5. If you nest your section elements too deep, you can end up with, let's say, sect8 which is not allowed in DocBook. To avoid making mistakes, it is better to check the level:

Example 3.12. Error Checking of Section Levels

```
<xsl:template match="d:section">
  <xsl:variable name="level" select="count(ancestor::*)"/>
  <xsl:choose>
    <xsl:when test="$level &lt;= 5">
      <xsl:element name="sect{$level}"
          namespace="http://docbook.org/ns/docbook">
        <xsl:copy-of select="@*"/>
        <xsl:apply-templates/>
      </xsl:element>
    </xsl:when>
    <xsl:otherwise>
      <xsl:message>ERROR: section <xsl:value-of
          select="normalize-space(d:title)"/> to deep</xsl:message>
      <!-- What to do if the section is too deeply nested? -->
    </xsl:otherwise>
```

```
  </xsl:choose>
</xsl:template>
```

Amend the stylesheet when the section is too deeply nested (see above comment). You could avoid the section level (which is a bad idea) but it's better to rework the source document.

If you want, you can stop the transformation if the section level is too high. Change the `xsl:message` as follows:

```
<xsl:message terminate="yes">...
```

3.9. Transforming bridgehead Elements into section Elements

Problem

You have a DocBook document which contains several bridgehead elements. The bridgehead has to be transformed into the correct section structure.

Solution

A bridgehead element is a “free-floating heading”. In most cases it is a bad idea as it is difficult to handle in XSLT. To create the correct section hierarchy, a stylesheet need to collect all nodes between a bridgehead element and the next one. The following stylesheet uses a set difference method:

Example 3.13. Transforms every bridgehead Element into a section Element

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

  <xsl:import href="copy.xsl"/>

  <xsl:output indent="yes"/>
  <xsl:strip-space elements="*"/>
  <xsl:preserve-space elements="d:screen d:programlisting d:literallayout"/>

  <xsl:template match="d:section[d:bridgehead]">
    <!-- All nodes inside our section: -->
    <xsl:variable name="node1" select="node()"/>
    <!-- All nodes -->
    <xsl:variable name="node2"
      select="d:bridgehead[1]|
              d:bridgehead[1]/following-sibling::node()"/>

    <!-- Copy our section with all attributes, apply the set difference
      and investigate the first bridgehead
    -->
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:apply-templates select="$node1[count(.|$node2) != count($node2)]"/>
      <xsl:apply-templates select="d:bridgehead[1]"/>
    </xsl:copy>
    <xsl:text>&#10;</xsl:text>
  </xsl:template>

  <xsl:template match="d:bridgehead">
    <!-- All nodes who follow bridgeheads: -->
    <xsl:variable name="node1"
      select="following-sibling::node()"/>
    <!-- All nodes who follow the next bridgehead including the next
         bridgehead.
         The next bridgehead element is included as we don't want it
         in the set difference:
    -->
    <xsl:variable name="node2"
      select="following-sibling::d:bridgehead[1]|
```

```
            following-sibling::d:bridgehead[1]/following-sibling::node()"/>

    <!-- Create the section element with all attributes and apply
      standard rules for the diff set -->
    <xsl:element name="section">
      <xsl:copy-of select="@*"/>
      <xsl:text>&#10;  </xsl:text>
      <xsl:element name="title">
        <xsl:apply-templates select="node()"/>
      </xsl:element>
      <xsl:apply-templates select="$node1[count(.|$node2) != count($node2)]"/>
    </xsl:element>

    <!-- Process the next bridgehead -->
    <xsl:apply-templates select="following-sibling::d:bridgehead[1]"/>
  </xsl:template>

</xsl:stylesheet>
```

Discussion

The solution is unfortunately not very trivial in XSLT 1.0. The template rule `d:section[d:bridgehead]` matches only `section`s which contain one or more `bridgehead` elements. The template rule performs the following steps:

1. Collect all nodes inside a `section` and save it in variable `node1`.
2. Collect all nodes who follows the next `bridgehead` including the next `bridgehead` itself and save it in variable `node2`.
3. Creates a `section` element and copy all attributes from the `bridgehead` element.
4. Creates a `title` element and apply the content from the `bridgehead` element. This copies the content from `bridgehead`.
5. Calculates the set difference between `node1` and `node2`. This weird expression is needed in XSLT 1.0 to create a node set which contains only those nodes up to the first `bridgehead`.
6. Handle the first `bridgehead` element which is covered by our `bridgehead` template rule.

The `bridgehead` template rule is responsible for transforming the current `bridgehead` element into a `section`. It is also responsible for the next `bridgehead`s. The rule performs the following steps:

1. Collect all nodes following of the current `bridgehead` element and save it in variable `node1`.
2. Collect all nodes following of the next `bridgehead` element including the next `bridgehead` element itself. Save the node set in variable `node2`.
3. Creates a `section` element and copy all attributes from the `bridgehead` element.
4. Creates a `title` element and apply the content from the `bridgehead` element. This copies the content from `bridgehead`.
5. Calculate the set difference between `node1` and `node2` and apply the correct template rule (usually they are just copied).
6. Close the `section` element and handle the next `bridgehead` element.

3.10. Moving Block Elements Outside of Paragraphs

Jeni Tennison
Dave Pawson

Problem

DocBook allows to insert block elements like `example`, `figure`, etc. to be inserted in `para`s. This is sometimes hard to process with XSLT and you want to move those block elements outside of your paragraph. After this modification, the paragraph contains only text and inline elements (like `simpara`).

Solution

The solution explained

Example 3.14. `move-blocks-outof-para.xsl`

```
<!DOCTYPE xsl:stylesheet
[
 <!ENTITY dbblocks "d:address|d:bibliolist|d:blockquote|d:bridgehead|d:calloutlist|d:caution|d:
 <!ENTITY dbselfblocks "self::d:address|self::d:bibliolist|self::d:blockquote|self::d:bridgehea
 <!ENTITY dbblocksinpara "d:para/d:address|d:para/d:bibliolist|d:para/d:blockquote|d:para/d:bri
]>
<xsl:stylesheet xmlns:d="http://docbook.org/ns/docbook"
  xmlns="http://docbook.org/ns/docbook"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <xsl:import href="copy.xsl"/>

  <xsl:strip-space elements="d:para"/>
  <xsl:preserve-space elements="d:screen d:programlisting d:literallayout"/>
  <xsl:output indent="yes"/>

  <xsl:template match="d:para">
    <xsl:apply-templates select="node()[1]"/>
  </xsl:template>

  <xsl:template match="&dbblocksinpara;">
    <xsl:copy-of select="."/>
    <xsl:text>&#10;</xsl:text>
    <xsl:apply-templates select="following-sibling::node()[1]"/>
  </xsl:template>

  <xsl:template match="d:para/*|d:para/text()">
    <xsl:element name="{local-name(..)}">
      <xsl:apply-templates select="." mode="copy"/>
    </xsl:element>
    <xsl:text>&#10;</xsl:text>
    <xsl:apply-templates
      select="following-sibling::*[&dbselfblocks;][1]"/>
  </xsl:template>

  <xsl:template match="d:para/node()" mode="copy">
    <xsl:copy-of select="."/>
    <xsl:if test="not(following-sibling::node()[1][&dbselfblocks;])">
      <xsl:apply-templates select="following-sibling::node()[1]"
        mode="copy"/>
```

```
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

Discussion

TBD

3.11. Adding Index Entries (Semi-)Automatically

Problem

You want to add index entries (using the `indexterm` element) automatically and consistently into your document.

Solution

To see how the automatic addition works, the following procedure demonstrate this for the element `envar`.

Procedure 3.1. Adding `indexterm` Elements to `envar`

1. Use the element `envar` in your document as usual. By default all `envar` elements get an `indexterm`. In cases you do not want this, add the attribute `condition` with its value `noindex` to suppress indexterm generation. This is done in the second `para` element:

   Example 3.15. `profile-envar.xml`

   ```
   <article  version="5.0"
     xmlns="http://docbook.org/ns/docbook">
     <title>Profiling Test</title>
     <para>Environment variable <envar>XML_CATALOG_FILES</envar></para>
     <para>Environment variable <envar condition="noindex">FOO</envar></para>
     <para>Environment variable <envar os="windows">Path</envar><envar
       os="linux">PATH</envar></para>
   </article>
   ```

2. Create a stylesheet `profile-tags.xsl` with the following content:

   Example 3.16. `profile-tag.xsl`

   ```
   <xsl:stylesheet version="1.0"
     xmlns:d="http://docbook.org/ns/docbook"
     xmlns="http://docbook.org/ns/docbook"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

   <xsl:param name="preferred">pref</xsl:param>

   <xsl:template name="check.index">
     <xsl:param name="node" select="."/>
     <xsl:param name="default" select="1"/>

       <xsl:choose>
         <xsl:when test="$node/@condition = 'noindex'">0</xsl:when>
         <xsl:when test="$node/@condition = 'index'">1</xsl:when>
         <xsl:otherwise><xsl:value-of select="$default"/></xsl:otherwise>
       </xsl:choose>
   </xsl:template>

   <xsl:template match="d:footnote|d:title|d:indexterm" mode="profile">
     <!-- Indexterms doesn't/shouldn't occur in the descendants of
          these elements so just copy it -->
     <xsl:copy-of select="."/>
   </xsl:template>

   <xsl:template match="d:envar" mode="profile">
       <xsl:variable name="do.index">
         <xsl:call-template name="check.index"/>
       </xsl:variable>
   ```

```
      <!-- Copy original element -->
      <xsl:copy-of select="."/>

      <xsl:if test="$do.index != 0">
        <indexterm>
          <primary><xsl:value-of select="."/></primary>
        </indexterm>
        <indexterm>
          <xsl:if test="contains(@conformance, $preferred)">
             <xsl:attribute name="significance">preferred</xsl:attribute>
          </xsl:if>
          <primary>environment variables</primary>
          <secondary><xsl:value-of select="."/></secondary>
        </indexterm>
      </xsl:if>
  </xsl:template>
  </xsl:stylesheet>
```

3. Create the stylesheet `add-indexterms.xsl`. This stylesheet is based on `profile.xsl` of the DocBook XSL stylesheets. It contains the special mode `profile` to process elements to observe profiling conditions.

Example 3.17. `add-indexterms.xsl`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet
[
  <!ENTITY db "https://cdn.docbook.org/release/xsl/current">
]>
<xsl:stylesheet version="1.0"
  xmlns:d="http://docbook.org/ns/docbook"
  xmlns="http://docbook.org/ns/docbook"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

<xsl:import href="&db;/profiling/profile.xsl"/>
<xsl:output indent="yes" method="xml"/>
<xsl:include href="profile-tags.xsl"/>

</xsl:stylesheet>
```

4. Transform your document:

```
xsltproc add-indexterms.xsl profile-envar.xml
```

After applying the stylesheet `add-indexterms.xsl` you will get the following output:

Example 3.18. Output of the Transformation

```
<article xmlns="http://docbook.org/ns/docbook" version="5.0">
  <title>Profiling Test</title>
  <para>Environment variable <envar>XML_CATALOG_FILES</envar><indexterm>
    <primary>XML_CATALOG_FILES</primary>
  </indexterm><indexterm>
    <primary>environment variables</primary>
    <secondary>XML_CATALOG_FILES</secondary>
    </indexterm></para>
  <para>Environment variable <envar condition="noindex">FOO</envar></para>
  <para>Environment variable <envar os="windows"
        >Path</envar><indexterm><primary>Path</primary></indexterm><indexterm><primary>en
        variables</primary><secondary>Path</secondary></indexterm><envar
      os="linux"
      >PATH</envar><indexterm>
        <primary>PATH</primary>
```

```
      </indexterm><indexterm>
        <primary>environment variables</primary>
        <secondary>PATH</secondary>
      </indexterm></para>
</article>
```

Discussion

Let´s go back at the beginning first. Assume you want to show an environment variable in the index. Usually you would mark up the text with the `envar` element and add an `indexterm` right after the first one. As is is useful to find the index term also under the primary term “environment variables”, you add an additional `indexterm`. This could look like this:

```
<para>Use the <envar>PATH</envar><indexterm>
        <primary>PATH</primary>
      </indexterm><indexterm>
        <primary>environment variables</primary>
        <secondary>PATH</secondary>
      </indexterm>
      to do ...
</para>
```

Although this is the usual method, it has some drawbacks:

- **It is hard to read** If you are get used to read the bare XML code, it is hard to read as the text is broken into pieces. The text is cluttered with `indexterm` elements all along.

- **It may be inconsistent** If you forgot the “s” in the primary index term it will lead to double entries (one singular and one plural form). This lead to inconsistencies. It can be painful if you have to go through the complete document just to fix the singular form into the plural form (or vice versa).

- **Whitespace could matter** The `indexterm` element(s) start *directly* after your term. If you or your editor introduces one or more whitespaces after your dedicated index term, in the worst case it could lead to a wrong page number in the index. This mainly affects the PDF rather than any online formats but could confuse your readers.

All of the above problems can be solved with the stylesheet from Example 3.16, “`profile-tag.xsl`” [96]. It exploits the DocBook XSL stylesheet´s profiling mechanism. Normally, profiling is a method to *remove* certain structures from a document rather than *add* something. In our case we use the special `profile` mode to customize the automatic index term addition.

With the above stylesheet, it is possible to influence how your index terms appear. This is done with the `condition`[1] attribute, demonstrated on our `envar` example:

`<envar>...</envar>`
: Adds the `indexterm`s directly after the `envar` element.

`<envar condition="index">...</envar>`
: Same as the previous entry.

`<envar condition="noindex">...</envar>`
: Suppresses any automatic generation of index entries.

`<envar condition="pref">...</envar>`
: Adds an *preferred* index entry. The keyword “pref” can be customized through the *`preferred`* parameter. If the keyword is added in the `condition` attribute, the following code is created:

```
<indexterm significance="preferred">...</indexterm>
```

This method can not solve all index problems. You should know some of its limits:

- **Document Type** Technical documents are more applicable than novels as the former contains usually a set of elements which are consistently used.

[1] Theoretically you could use any of the several common attributes available on every DocBook element. The `condition` was the one attribute that has fitted the best in the authors mind.

- **Consistent Elements** The document needs not only consistently use the same elements for the same structure, it has to use a specific element in the first place. For example, if you want to show your configuration files in your index, you need to mark it up with `filename`, otherwise this method has no chance.

- **Needed Elements** Similar to the previous point, you have to know which elements you need to show up in the index. You have to select from all possible inline elements only a handful which you consider important enough.

- **Only for Inline Elements** This method works only for inline elements well. DocBook´s inline elements occur usually inside a paragraph but can also show up in a title.

- **Hard-coded Primary Entry** It can be critized to add hard-coded text into the stylesheet `profile-tag.xsl` (here: “environment variables”). If you maintain different languages, you should replace it with a more general solution and move the language specific text into language files as described in Section 2.9, “Extending Language Files with Your Own Text” [46].

Although this method does not replace hand-written index entries, it can ease the pain. Especially for those entries which can be be inserted automatically it improves consistency. The method described above can also be implemented for other inline elements, like persons, functions, etc.

See Also

- Section 1.4, “Mastering Multiple Indices” [8]

3.12. Including Revision Information from Version Control Systems

Problem

You want to include revision information into your DocBook document from your version control systems, like Bazaar, Subversion, Mercurial, Git, and others.

Solution

Usually, the solution for each version control system is to output its log into its specific XML output and transform it with XSLT into a `revhistory` element.

Bazaar

Bazaar [http://bazaar.canonical.com] does not come with a predefined XML output. If you need this functionality, you have to install the xmloutput plugin first. This brings the `--xml` option to the **log** subcommand. Proceed as follows:

Procedure 3.2. Installing the Bazaar Plugin xmloutput

1. Download the tar archive from http://wiki.bazaar.canonical.com/XMLOutput.
2. Unpack the archive.
3. Change to the extension directory (usually something like `bzr-xmloutput-`*`VERSION`*) and run:

   ```
   python setup build_ext
   ```

4. Copy the result directory `build/lib/bzrlib/plugins/xmloutput/` to your plugin directory. For Linux it is something like `$HOME/.bazaar/plugins/`, for Windows copy it to `$APPDATA/bazaar/`*`VERSION`*`/plugins`
5. Check if the plugin is correctly detected:

   ```
   bzr plugins
   ```

   You should see the following line:

   ```
   xmloutput VERSION
       This plugin adds an option (--xml) to log and provides an xml version of
       some built-in commands.
   ```

After you have successully installed the plugin, create the log output in XML with the following command:

```
bzr log --xml REPO > bzr-log.xml
```

To convert the previous log file into DocBook's `revhistory`, use the following stylesheet:

Example 3.19. Stylesheet to Convert Bazaar's Log File

TBD

Git

Git [http://git-scm.com] does not have any options to turn the log into XML. However, with the `--pretty` option you can collect the information and wrap it into any XML elements you wish. The following command uses this option and **sed** to insert the `<logs>` start tag in the first line and `</logs>` end tag in the last line:

```
git log --date=iso --pretty=format:"<logentry revision='%h'>%n <author email='%ae'>%an</author>
  | sed -e '1i<logs>' -e '$a</logs>' > git-log.xml
```

As the element names can be freely chosen, we use the same names as in Example 3.20, “Mercurial Log File in XML” [101]. This allows us to use the stylesheet from Example 3.21, “Stylesheet `hg2revhistory.xsl` to Convert Mercurial XML Log Files into DocBook's `revhistory`” [101].

Mercurial

Mercurial [http://mercurial.selenic.com] has the `--style` option to output its log into XML:

```
hg log --style xml > hg-log.xml
```

The XML file of the Mercurial log looks like this:

Example 3.20. Mercurial Log File in XML

```
<log>
  <logentry revision="69"
    node="dfadd024594c4083362fe6919264362803dcd285">
    <tag>tip</tag>
    <author email="tux@example.de">Tux Penguin</author>
    <date>2011-05-22T01:56:21+02:00</date>
    <msg xml:space="preserve">Corrected xml:id attribute</msg>
    <paths>
      <path action="M">xml/structure/topic.extract-element.xml</path>
    </paths>
  </logentry>
  <logentry revision="68"
    node="833287df8943d0bab5ec65ec8aafe5bc42002289">
    <author email="wilber@example.de">Wilber Gimp</author>
    <date>2011-05-22T01:56:03+02:00</date>
    <msg xml:space="preserve">Changed chapter title</msg>
    <paths>
      <path action="M">xml/structure/topic.revision-list.xml</path>
    </paths>
  </logentry>
  <!-- Many more entries ... -->
</log
```

The following stylesheet converts the log file from Example 3.20, “Mercurial Log File in XML” [101] into DocBook's `revhistory`:

Example 3.21. Stylesheet `hg2revhistory.xsl` to Convert Mercurial XML Log Files into DocBook's `revhistory`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet
[
  <!ENTITY dbns "http://docbook.org/ns/docbook">
]>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" indent="yes"/>

  <xsl:param name="include.paths" select="1"/>
  <xsl:param name="include.copies" select="1"/>

  <xsl:template match="log">
    <revhistory xmlns="&dbns;">
      <xsl:apply-templates/>
    </revhistory>
  </xsl:template>
```

```
<xsl:template match="logentry">
  <revhistory xmlns="&dbns;">
    <xsl:apply-templates select="@revision|*"/>
  </revhistory>
</xsl:template>

<xsl:template match="logentry/@revision">
  <revnumber xmlns="&dbns;">
    <xsl:apply-templates/>
  </revnumber>
</xsl:template>

<xsl:template match="tag">
  <revremark xmlns="&dbns;">
    <xsl:apply-templates/>
  </revremark>
</xsl:template>

<xsl:template match="date">
  <date xmlns="&dbns;">
    <xsl:apply-templates/>
  </date>
</xsl:template>

<xsl:template match="author">
  <xsl:variable name="firstname" select="substring-before(., ' ')"/>
  <xsl:variable name="surname" select="substring-after(., ' ')"/>
  <author xmlns="&dbns;">
    <personname>
      <firstname><xsl:value-of select="$firstname"/></firstname>
      <!--<othername></othername>-->
      <surname><xsl:value-of select="$surname"/></surname>
    </personname>
    <email><xsl:value-of select="@email"/></email>
  </author>
</xsl:template>

<xsl:template match="msg">
  <revdescription xmlns="&dbns;">
    <para>
      <xsl:apply-templates/>
    </para>
  </revdescription>
</xsl:template>

<xsl:template match="paths">
  <xsl:if test="$include.paths != 0">
    <itemizedlist xmlns="&dbns;">
      <title>Paths</title>
      <xsl:apply-templates/>
    </itemizedlist>
  </xsl:if>
</xsl:template>

<xsl:template match="path">
  <listitem xmlns="&dbns;">
    <para>
      <xsl:apply-templates select="@action|text()"/>
    </para>
  </listitem>
</xsl:template>
```

```
  <xsl:template match="path/@action">
    <xsl:text>[</xsl:text>
    <xsl:value-of select="."/>
    <xsl:text>] </xsl:text>
  </xsl:template>
  <xsl:template match="path/text()">
    <filename xmlns="&dbns;">
      <xsl:value-of select="."/>
    </filename>
  </xsl:template>

  <xsl:template match="copyies">
    <xsl:if test="$include.paths != 0">
      <listitem xmlns="&dbns;">
        <title>Copies</title>
        <para>
          <xsl:apply-templates select="@*|text()"/>
        </para>
      </listitem>
    </xsl:if>
  </xsl:template>
  <xsl:template match="copy">
    <listitem xmlns="&dbns;">
      <para>
        <xsl:apply-templates select="@*|text()"/>
      </para>
    </listitem>
  </xsl:template>

  <xsl:template match="copy/@source">
    <filename xmlns="&dbns;">
      <xsl:value-of select="."/>
    </filename>
  </xsl:template>
  <xsl:template match="copy/text()">
    <xsl:text> -> </xsl:text>
    <filename xmlns="&dbns;">
      <xsl:value-of select="."/>
    </filename>
  </xsl:template>

</xsl:stylesheet>
```

Subversion

Subversion [http://subversion.tigris.org] is similar to Mercurial: It has an `--xml` option to turn the log output into XML:

```
svn log --xml > svn-log.xml
```

With the `-v` option (verbose) you get additional path information. As the output is very similar to Mercurial, you can apply the stylesheet shown in Example 3.21, “Stylesheet `hg2revhistory.xsl` to Convert Mercurial XML Log Files into DocBook's `revhistory`” [101] also for Subversion's log file.

As an alternative, the DocBook Subversion repository (see https://docbook.svn.sourceforge.net/svnroot/docbook/trunk) contains the XSLT stylesheet `releasetools/svnlog2docbook.xsl` with additional features.

Discussion

TBD

3.13. Creating an Acronym List

Contributed idea and stylesheet: Peter Lavin

Problem

You have a glossary with acronyms and you want to generate a sorted list of these acronyms automatically.

Solution

The following prerequisites are needed:

1. The element `glossary` and for each `glossterm` your acronym as shown in the following structure:

```
<glossary version="5.0"
  xmlns="http://docbook.org/ns/docbook">
  <title>Acronym Test File</title>
  <info>
    <author>
      <personname>
        <firstname>Thomas</firstname>
        <surname>Schraitle</surname>
      </personname>
    </author>
  </info>

  <glossentry xml:id="xml">
    <acronym>XML</acronym>
    <glossdef>
      <para>Lore ipsum</para>
    </glossdef>
  </glossentry>
  <glossentry xml:id="pdf">
    <acronym>PDF</acronym>
    <glossdef>
      <para>Lore ipsum</para>
    </glossdef>
  </glossentry>
  <glossentry xml:id="oasis">
    <acronym>OASIS</acronym>
    <glossdef>
      <para>Lore ipsum</para>
    </glossdef>
  </glossentry>
  <glossentry xml:id="dtd">
    <acronym role="foo" xml:id="dtd-acronym">DTD</acronym>
    <glossdef>
      <para>Lore ipsum</para>
    </glossdef>
  </glossentry>
</glossary>
```

2. The following stylesheet to apply it to your XML structure:

Example 3.22. `make_acronyms.xsl`

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://docbook.org/ns/docbook"
  xmlns="http://docbook.org/ns/docbook"
  exclude-result-prefixes="d">
```

```
  <xsl:output encoding="UTF-8" indent="yes" method="xml" omit-xml-declaration="yes"/>

  <xsl:param name="appendixid">acronyms</xsl:param>
  <xsl:param name="docbookversion">5.0</xsl:param>

  <xsl:template match="/">
    <!-- The comment below appears in the output file-->
<xsl:comment>
 This file is dynamically generated at build time from a glossary file.
 To change this file, make changes to the appropriate glossary.xml file.
 In order to generate the acronyms file, any glossary entries with an
 acronym tag must have an xml:id.
</xsl:comment>
    <appendix version="{$docbookversion}">
      <xsl:if test="$appendixid != ''">
        <xsl:attribute name="xml:id">
          <xsl:value-of select="$appendixid"/>
        </xsl:attribute>
      </xsl:if>
      <title>Acronyms</title>
      <para> This appendix displays acronyms sorted alphabetically and
        linked to their definition in the glossary. </para>
      <xsl:apply-templates select=".//d:glossary[1]"/>
    </appendix>
  </xsl:template>

  <xsl:template match="d:glossary">
    <itemizedlist remap="glossary">
      <xsl:apply-templates select="d:glossentry[d:acronym]">
        <xsl:sort select="d:acronym"/>
      </xsl:apply-templates>
    </itemizedlist>
  </xsl:template>

  <xsl:template match="d:glossentry[d:acronym]">
    <listitem>
      <xsl:apply-templates select="d:acronym"/>
    </listitem>
  </xsl:template>

  <xsl:template match="d:acronym">
    <xsl:variable name="id">
      <xsl:choose>
        <xsl:when test="(ancestor-or-self::d:*/@xml:id)[last()]">
          <xsl:value-of select="(ancestor-or-self::d:*/@xml:id)[last()]"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:message>No ID found!</xsl:message>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
      <para>
        <xsl:copy>
          <xsl:if test="$id != ''">
            <xsl:attribute name="linkend">
              <xsl:value-of select="$id"/>
            </xsl:attribute>
          </xsl:if>
          <xsl:attribute name="remap">acronym</xsl:attribute>
          <xsl:copy-of select="node()"/>
```

```
          </xsl:copy>
        </para>
    </xsl:template>

  </xsl:stylesheet>
```

The result will look like this:

```
<!--
 This file is dynamically generated at build time from a glossary file.
 To change this file, make changes to the appropriate glossary.xml file.
 In order to generate the acronyms file, any glossary entries with an
 acronym tag must have an xml:id.
-->
<appendix xmlns="http://docbook.org/ns/docbook" version="5.0" xml:id="acronyms">
  <title>Acronyms</title>
  <para> This appendix displays acronyms sorted alphabetically and
        linked to their definition in the glossary. </para>
  <itemizedlist remap="glossary">
    <listitem>
      <para>
        <acronym linkend="dtd-acronym" remap="acronym">DTD</acronym>
      </para>
    </listitem>
    <listitem>
      <para>
        <acronym linkend="oasis" remap="acronym">OASIS</acronym>
      </para>
    </listitem>
    <listitem>
      <para>
        <acronym linkend="pdf" remap="acronym">PDF</acronym>
      </para>
    </listitem>
    <listitem>
      <para>
        <acronym linkend="xml" remap="acronym">XML</acronym>
      </para>
    </listitem>
  </itemizedlist>
</appendix>
```

Discussion

TBD

3.14. Splitting DocBook 5 Documents Into Topics

Problem

You need to create topics of your existing DocBook 5 document to use it with the new `assembly` [http://www.docbook.org/tdg51/en/html/assembly.html] and `topic` [http://www.docbook.org/tdg51/en/html/topic.html] elements in DocBook 5.1.

Solution

Use the `assembly/topic-maker-chunk.xsl` stylesheet to create topics from your current DocBook 5 document. This breaks apart your existing document into modular files and creates an assembly file.

To disassemble an existing `book.xml` document, use the **xsltproc** command like this (the variable `DB` contains the path of your DocBook XSL directory):

```
xsltproc --xinclude \
    --stringparam assembly.filename  assembly.xml \
    --stringparam base.dir topics/ \
    $DB/assembly/topic-maker-chunk.xsl  \
    book.xml
```

The **xsltproc** command outputs the master assembly file `assembly.xml`. The document's content is break up into modular chunks, saved in the `topics/` directory. The master file contains a single `structure` [http://www.docbook.org/tdg51/en/html/structure.html] element.

Discussion

The `topic-maker-chunk.xsl` contains several parameters to influence the chunking process:

Table 3.2. Parameters to Influence Chunking Process Into Topics

Parameter	Description
`assembly.filename`	Name of the assembly file (default: `myassembly.xml`)
`base.dir`	Directory name where to store all generated topic files (default: `topics/`)
`chunk.first.sections`	Should the first top-level section be chunked? (default: `1` = yes, otherwise no)
`chunk.section.depth`	Depth to which sections should be chunked (default is `3`)
`manifest.in.base.dir`	Puts the assembly file into the directory set by *`base.dir`* instead of the current directory
`topic.elements`	Elements which are converted into topics (default is `preface`, `chapter`, `article`, and `section`)
`html.ext`	Extension of the topic files (default is `.xml`)
`root.id.suffix`	Suffix to add, when *`root.as.resourceref`* is set to `0` (default is `-info`)
`root.as.resourceref`	Should the root element also converted into a topic? (default is `1` = yes, otherwise no)
`use.id.as.filename`	Use `xml:id` value of chunk elements to create filename? (default: `1` = yes, otherwise no)

The `topic-maker-chunk.xsl` stylesheet reuses the same chunking algorithm than the XHTML stylesheets. That means, it breaks the document at the same boundaries and you can alter the chunking process with the same parameters than the XHTML stylesheets.

The following subsections shows some specific examples to influence the chunking process.

Influencing where to Store the Assembly File

By default, the `topic-maker-chunk.xsl` puts the assembly file in the current directory instead of the directory set by *`base.dir`*. If you want to have the assembly file and its topics together in the same directory, set the parameter *`manifest.in.base.dir`* to `1`.

Changing the Chunking Depth

By default, the parameter *`chunk.section.depth`* is set to `3` which means, the stylesheet puts each element into a separate file down to `sect2`. The higher the value, the more file it creates (and vice-versa).

If you need finer control of the chunking process, use the `dbhtml stop-chunking` processing-instruction as described in http://docbook.sourceforge.net/release/xsl/current/doc/pi/dbhtml_stop-chunking.html.

See Also

- http://docbook.sourceforge.net/release/xsl/current/doc/html/chunking.html
- http://docbook.sourceforge.net/release/xsl/current/doc/pi/dbhtml_stop-chunking.html

3.15. Assembling Topics

Problem

You want to convert an `assembly` structure and its associated modular files into a single DocBook 5 file.

Solution

You need the following prerequisites:

1. An *assembly file*. This starts with the `assembly [http://www.docbook.org/tdg51/en/html/assembly.html]` root element from DocBook 5.1. The assembly file contains references to all of your modules which composes your realized DocBook document. Or in other words: the assembly file maps the modules to structure.
2. The `assembly/assemble.xsl` stylesheet from the DocBook XSL distribution. Get the latest version as it is added in 2012.
3. Your module files based on DocBook 5.

Use **xsltproc** to create a single DocBook 5 (the variable `DB` contains the path of your DocBook XSL directory):

```
xsltproc \
    $DB/assembly/assemble.xsl \
    assembly.xml
```

Discussion

The `assemble.xsl` stylesheet contains several parameters to influence the assembling process:

Table 3.3. Parameters to Influence the Assembling Process

Parameter	Description
`root.default.renderas`	Sets the name of the root element of the assembled document, if not specified with `renderas` attribute
`structure.id`	Selects one `structure` element from several; similar to the *`rootid`* parameter for the XHTML stylesheets
`topic.default.renderas`	Sets the name of the output element of any topic elements if not specified with `renderas` attribute
`output.format`	Selects which of the possible output formats are being generated. The value of this parameter should match on a `format` attribute on `output` elements.
`output.type`	Selects which structure element to process; the value should match with the `type` attribute on a `structure` element

The following subsections shows some specific examples to influence the assembling process.

Selecting a Structure by ID

For example, your assembly file looks like this:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources> ... </resources>
  <structure xml:id="quickstart"> ... </structure>
  <structure xml:id="reference"> ... </structure>
</assembly>
```

As you can see, you have two `structure` elements, one for a quick start, another for a reference. To process only the reference, set the parameter *`structure.id`* to the `xml:id` of the reference as follows:

```
xsltproc \
    --stringparam structure.id reference \
    $DB/assembly/assemble.xsl \
    assembly.xml
```

If you omit the *structure.id* parameter, the `assemble.xsl` stlyesheet selects always the first `structure` element.

Selecting a Structure by Type

For example, your assembly file looks like this:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources> ... </resources>
  <structure type="help.system"> ... </structure>
  <structure type="startup"> ... </structure>
</assembly>
```

As you can see, there are two `structure` elements, one for a help system, another for a startup. To process only the startup, set the parameter *output.type* as follows:

```
xsltproc \
    --stringparam output.type startup \
    $DB/assembly/assemble.xsl \
    assembly.xml
```

See Also

- Section 3.14, “Splitting DocBook 5 Documents Into Topics” [107]
- http://www.docbook.org/tdg51/en/html/assembly.html

3.16. Creating an Assembly File Manually

Problem

You want create topic oriented documents with the `assembly [http://www.docbook.org/tdg51/en/html/assembly.html]` element manually.

Solution

In its simplest form, an assembly file contains the following elements:

`assembly`
: The root element of an assembly file. It is bound to the DocBook 5 namespace `http://docbook.org/ns/docbook`.

`resources`
: Wrapper element for a collection of topics. Each topic is identified by a `resource` element and its `xml:id` or `fileref` attributes:

  ```
  <resource xml:id="preface" fileref="preface.xml"/>
  ```

  An assembly file can contain more than one `resource` elements.

`structure`
: Wrapper element for a document to be assembled. Primarily it contains of `module`s which refers to a `resource` elements.

The following example assumes, the files are composed of `topic` root elements, except for the `glossary.xml` file (which is comprised by a `glossary` element).

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>
    <resource fileref="intro.xml" xml:id="intro"/>
    <resource fileref="kde.xml" xml:id="kde"/>
    <resource fileref="gnome.xml" xml:id="gnome"/>
    <resource fileref="glossary.xml" xml:id=""/>
  </resources>
  <structure xml:id="user-guide">
    <output renderas="book"/>
    <module resourceref="intro"/>
    <module resourceref="kde"/>
    <module resourceref="gnome"/>
    <module resourceref="glossary"/>
  </structure>
</assembly>
```

Discussion

TDB

See Also

- http://www.docbook.org/tdg51/en/html/ch06.html
- Section 3.14, “Splitting DocBook 5 Documents Into Topics” [107]
- Section 3.15, “Assembling Topics” [109]

3.17. Using Entities as Placeholders

Problem

You need text or a small XML structure that you want to use throughout your document.

Solution

Use custom entities. An entity is a placeholder for text or XML. You can define them inside a DOCTYPE declaration or in external files. The following procedure shows you how to create and use a entity:

1. Create a new `.ent` file, for example, `entities.ent`.
2. Add your entity definitions to the file.

   Each definition starts with `<!ENTITY`, the entity name, the content, and a final > character like this:

   ```
   <!ENTITY product "FooMatic">
   ```

   You can add as many entity definitions as you like. Keep in mind the following restrictions:

   - If your content contains characters like < or &, write them as `&lt;` and `&amp;`.
   - It is allowed to use other entities like this:

     ```
     <!ENTITY version "2.5">
     <!ENTITY product "FooMatic">
     <!ENTITY productver "&product; &version;">
     ```

     However, it is not allowed to create circular references like this:

     ```
     <!ENTITY a "&b;">
     <!ENTITY b "&a;">
     ```

   - If you use XML structures in your entity, the content of the entity must be well-formed. This is allowed:

     ```
     <!ENTITY x "<foo>x</foo>">
     ```

     However, this is not:

     ```
     <!ENTITY x-start "<foo>x">
     <!ENTITY x-end "</foo>">
     ```

3. In each file where you want to use your entity definitions, add the following header and replace *`ROOT`* with the name of the root element:

   ```
   <!DOCTYPE ROOT [
     <!ENTITY % ents SYSTEM "entities.ent">
     %ents;
   ]>
   ```

4. To use the definied entities, write `&product;` in a text, for example:

   ```
   <para>Use &product; to make your children happy.</para>
   ```

Discussion

Entities improve the consistency of your texts. If your product name, your version, or any other definition changes, it is very easy to change its definition too, without going through all of your texts.

Entities can contain text and XML. For example, if you have text where you need to add the same file name repeatedly, it would be a good idea to create a entity:

```
<!ENTITY configfile "<filename xmlns='http://docbook.org/ns/docbook'>foo.conf</filename>"
```

The only problem with XML structures in entities is, you need to add the DocBook namespace in each definition. By default, each XML element belongs to no namespace.

See Also

- Section 3.18, “Preserving Entities” [114]

3.18. Preserving Entities

Problem

You want to process your XML file, but you do not want to expand any of the defined entities.

Solution

Replace all your entities with a string that is easy to search for. For example, you can replace the entity `&product;` with the string `[[[product]]]` (assuming that the string `[[[product]]]` doesn't occur anywhere else in our document).

Before you can proceed, you need to prepare your system as described in Procedure 3.3, “Preparing the Workflow” [114].

Procedure 3.3. Preparing the Workflow

1. Make sure you have Python 3 installed on your system. Find installation instructions on the project's home page at https://www.python.org/. The script works with Python 3.4 and above.

   On Linux or macOS Python may already be installed. If that is not the case, Python can be installed using the system's package manager.

2. Download the Python script `ents2text.py`.

3. Save the script where it can be found by the system. Linux users can store the script in either `~/bin` or `/usr/bin`. In this procedure, we use `~/bin`. Make the script executable with:

   ```
   $ chmod +x ~/bin/ents2text.py
   ```

4. Create a link:

   ```
   $ ln -rs ~/bin/ents2text.py ~/bin/text2ents.py
   ```

   This is required for converting from entities to text and vice versa. The script uses the script's name to determine the conversion direction.

Once you have done all the preparations, proceed with Procedure 3.4, “Workflow for Protecting Custom Entities” [114].

Procedure 3.4. Workflow for Protecting Custom Entities

1. Convert all entities to text with the following command:

   ```
   $ ents2text.py XMLFILE1 XMLFILE2...
   ```

   The script converts all entities to their protected notation in place and creates backup files with the extension `.bak`.

2. Process your XML file with your XML tools.

3. Revert all the protected notation to their original entity notation:

   ```
   $ text2ents.py XMLFILE1 XMLFILE2...
   ```

Discussion

Not expanding entities looks like an abnormal use case as XML parsers are supposed to expand entities by default. However, expanding entities has one disadvantage: after the XML parser has resolved entities, the content is indistinguishable from other content. In other words, you cannot distinguish content that comes from an entity definition from existing content. You cannot “go back” and revert this process once all entities are expanded.

For this reason, preserving entities can be useful when you want to process XML, but keep the definied entities untouched. Procedure 3.4 [114] showed one solution. However, be aware of certain issues that might cause problems with XML files:

- The script does not know XML. As such, it reads the XML file line by line and replaces each line through regular expressions.
- The script can replace entities in situations where it is not desirable. For example, if you have an entity in an `href` of a `xi:include` element, such reference will not work anymore when resolving XIncludes.
- The script doesn't handle external entities. External entities refer to external storage objects, for example:

  ```
  <!DOCTYPE book [
    <!ENTITY intro SYSTEM "chap-intro.xml">
  ]>
  <book>
    <title>...</title>
    &intro;
  </book>
  ```

  Such external entities usually refer to whole XML structures like chapters, sections etc. Replacing such structures with ordinary text would lead to syntax errors in your XML files. However, such entities are not used very often and they should be replaced by XInludes (see Section 1.3, “Modularize Your Document with XIncludes” [5]).

If you prefer a solution that can handle XML, use the Python script `ents2pi.py`. This script can parse XML and replaces each entity with a processing instruction. For example, the entity `&product;` is converted to the PI `<?entity product>`. That makes it easier to process it with XSLT or any other XML-agnostic tool.

Table 3.4. Comparison Between `ent2text.py` and `ents2pi.py`

	`ent2text.py`	**`ents2pi.py`**
	Entity to Text	**Entity to PI**
XML-aware	No	Yes
Easy to process it further	More difficult	Easy
Preserving source code	Always	Usually
Dependencies	Only Python standard library	lxml[a]

[a] See https://pypi.org/project/lxml/

See Also

- Section 3.17, “Using Entities as Placeholders” [112]
- https://github.com/tomschr/dbcookbook/raw/develop/en/xml/structure/entities/ents2text.py
- https://github.com/tomschr/dbcookbook/raw/develop/en/xml/structure/entities/ents2pi.py

Chapter 4. Print Customizations

Abstract

The DocBook XSL stylesheets has got more than 200 parameters which can customize some aspects of the layout very easily (like the page dimensions). This chapter shows customizations which go beyond simple parameter change.

4.1. Introduction

Currently, there is only one stylesheet available for creating FO output: `fo/docbook.xsl`. To be exact, this creates XSL-FO 1.1 output which is compatible with decent FO formatters.

Furthermore, the DocBook FO stylesheets supports extensions which enables the formatter to implement specific features. These features go beyond the FO specifications like PDF bookmarks, indices, etc. Currently, the stylesheets support extensions from PTC's Arbortext, Antenna House's AXF, Apache's FOP, and RenderX's XEP.

4.2. Designing a Title Page

Problem

You want to design a title page of your book or article.

Solution

To design your title page there are two ways to do it, regardless if it is a book or an article:

1. **Indirectly** Write a title page template which contains the wanted elements in their respective order.

2. **Directly** Customize the specific named templates.

Both methods are shown below. Although the following descriptions focus on a book title page, the same procedure applies for an article title page as well.

Before we customize the stylesheets, we need to define, *what* we want to display on our title page(s). Depending on if it is a left or a right page, different elements needs to be shown. For this example, we use the following requisites:

Recto (Right) Page

This is the “main page” and the content appears in the following order:

- the book title, from `/book/title` or `/book/info/title`
- the book subtitle, from `/book/subtitle` or `/book/info/subtitle`
- the book's author, from `/book/info/author`
- the edition, from `/book/info/edition`

Verso (Left) Page

This usually holds the imprint and the content appears in the following order:

- the book's title and subtitle in a smaller font size
- the author, from `/book/info/author`
- the edition, from `/book/info/edition`
- some legal text (copyright), from `/book/info/legalnotice`
- the ISBN, from `/book/info/biblioid` with `class` and the value `isbn`

Using Title Page Templates

To create a title page using a title page template proceed as follows:

1. Prepare the title page template:

   a. Copy the file `fo/titlepage.templates.xml` from the DocBook XSL stylesheet distribution to a directory where all your FO customization is stored. Use the filename `booktitlepage.xml` so we know, it contains only a title page for a book.

   b. Open the file `booktitlepage.xml` and remove anything except the root and `t:titlepage` elements with the attribute `t:element="book"`. Your title page template should look like this:

```
<!DOCTYPE t:templates [
<!ENTITY hsize0 "10pt">
<!ENTITY hsize1 "12pt">
```

```
<!ENTITY hsize2 "14.4pt">
<!ENTITY hsize3 "17.28pt">
<!ENTITY hsize4 "20.736pt">
<!ENTITY hsize5 "24.8832pt">
<!ENTITY hsize0space "7.5pt"> <!-- 0.75 * hsize0 -->
<!ENTITY hsize1space "9pt"> <!-- 0.75 * hsize1 -->
<!ENTITY hsize2space "10.8pt"> <!-- 0.75 * hsize2 -->
<!ENTITY hsize3space "12.96pt"> <!-- 0.75 * hsize3 -->
<!ENTITY hsize4space "15.552pt"> <!-- 0.75 * hsize4 -->
<!ENTITY hsize5space "18.6624pt"> <!-- 0.75 * hsize5 -->
]>
<t:templates xmlns:t="http://nwalsh.com/docbook/xsl/template/1.0"
             xmlns:param="http://nwalsh.com/docbook/xsl/template/1.0/param"
             xmlns:fo="http://www.w3.org/1999/XSL/Format"
             xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <t:titlepage t:element="book" t:wrapper="fo:block">
     <!-- Content disabled for better legibility -->
   </t:titlepage>
</t:templates>
```

2. Customize the content of your titlepage and save your changes into the `booktitlepage.xml` file.

   a. Change the recto page and locate the `t:titlepage-content` element with its attribute `t:side="recto"`. As definied in Recto (Right) Page [119], we remove everything what is not needed. Additionally we have to output edition, which we get from an empty `<edition/>` tag. The content of the `t:titlepage-content` should look like this:

```
<t:titlepage-content t:side="recto">
  <title     t:named-template="division.title"
             param:node="ancestor-or-self::book[1]"
             text-align="center"
             font-size="&hsize5;"
             space-before="&hsize5space;"
             font-weight="bold"
             font-family="{$title.fontset}"/>
  <subtitle  text-align="center"
             font-size="&hsize4;"
             space-before="&hsize4space;"
             font-family="{$title.fontset}"/>
  <author    font-size="&hsize3;"
             space-before="&hsize2space;"
             keep-with-next.within-column="always"/>
  <edition   font-size="&hsize3;"/>
</t:titlepage-content>
```

   b. Change the verso page and locate the `t:titlepage-content` element with its attribute `t:side="verso"`. As definied in Verso (Left) Page [119], again, we remove everything what is not needed. The content of the `t:titlepage-content` should look like this:

```
<t:titlepage-content t:side="verso">
  <title     t:named-template="book.verso.title"
             font-size="&hsize2;"
             font-weight="bold"
             font-family="{$title.fontset}"/>
  <subtitle  t:named-template="book.verso.title"
             font-size="&hsize2;"
             font-weight="bold"
             font-family="{$title.fontset}"/>
  <author/>
  <edition/>
  <legalnotice/>
  <biblioid  t:predicate="[@class = 'isbn']"/>
```

```
</t:titlepage-content>
```

c. Leave the other elements (`t:titlepage-separator` and `t:titlepage-before`) as they are.

3. Use the `template/titlepage.xsl` stylesheet from the DocBook XSL distribution to transform your `book-titlepage.xml` title page definition to create the `booktitlepage.xsl` output:

```
xsltproc --output booktitlepage.xsl template/titlepage.xsl booktitlepage.xml
```

4. Insert the constructed `booktitlepage.xsl` into your customization layer `mybooktitlepage.xsl`:

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

  <xsl:import href="https://cdn.docbook.org/release/xsl/current/fo/docbook.xsl"/>
  <xsl:include href="booktitlepage.xsl"/>
  <!-- More customizations hidden -->
</xsl:stylesheet>
```

5. Build your book as usual with your customization layer.

Customizing Title Pages Directly

To create a title page directly using named templates proceed as follows:

1. Open the file `fo/titlepage.templates.xsl` of your DocBook XSL stylesheet distribution.

2. Search for the correct template name(s). A template name is composed of the following components, wheras *`ELEMENT`* and *`SIDE`* are placeholders:

   `ELEMENT``.titlepage.`*`SIDE`*

   As we want to change the title page of a book for the left (verso) and right (recto) pages, the correct template names are `book.titlepage.recto` and `book.titlepage.verso`.

3. Create a new file, for example `booktitlepage.xsl`. This will contain all our customizations of our book's title page.

4. Copy the `book.titlepage.recto` and `book.titlepage.verso` templates into your new file `booktitlepage.xsl`.

5. Customize the template `book.titlepage.recto` according to the definitions in Recto (Right) Page [119]:

   a. Leave the two `xsl:choose` conditions intact to select the book's title and subtitle.

   b. Insert after the last `</xsl:choose>` the following line to select the book's author from `d:info/d:author`:

   ```
   <xsl:apply-templates mode="book.titlepage.recto.auto.mode" select="d:info/d:author
   ```

   c. Insert the following line to select the book's edition from `d:info/d:edition`:

   ```
   <xsl:apply-templates mode="book.titlepage.recto.auto.mode" select="d:info/d:editio
   ```

   d. Remove the other `xsl:apply-templates` elements as we only want to display the title, subtitle, author, and the editor. The template should look like this:

   ```
   <xsl:template name="book.titlepage.recto">
     <xsl:choose>
       <xsl:when test="d:bookinfo/d:title">
         <xsl:apply-templates mode="book.titlepage.recto.auto.mode" select="d:bookinf
       </xsl:when>
   ```

```
    <xsl:when test="d:info/d:title">
      <xsl:apply-templates mode="book.titlepage.recto.auto.mode" select="d:info/d:title"/
    </xsl:when>
    <xsl:when test="d:title">
      <xsl:apply-templates mode="book.titlepage.recto.auto.mode" select="d:title"/>
    </xsl:when>
  </xsl:choose>

  <xsl:choose>
    <xsl:when test="d:bookinfo/d:subtitle">
      <xsl:apply-templates mode="book.titlepage.recto.auto.mode" select="d:bookinfo/d:sub
    </xsl:when>
    <xsl:when test="d:info/d:subtitle">
      <xsl:apply-templates mode="book.titlepage.recto.auto.mode" select="d:info/subtitle"
    </xsl:when>
    <xsl:when test="d:subtitle">
      <xsl:apply-templates mode="book.titlepage.recto.auto.mode" select="d:subtitle"/>
    </xsl:when>
  </xsl:choose>

  <xsl:apply-templates mode="book.titlepage.recto.auto.mode" select="d:info/d:author"/>
  <xsl:apply-templates mode="book.titlepage.recto.auto.mode" select="d:info/d:edition"/>
</xsl:template>
```

6. Customize the template `book.titlepage.verso` according to the definitions in Verso (Left) Page [119] (note the different mode):

   a. Leave the two `xsl:choose` conditions intact to select the book's title and subtitle.

   b. Insert after the last `</xsl:choose>` the following line to select the book's author from `d:info/d:author`:

   ```
   <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:info/d:author"/>
   ```

   c. Insert the following line to select the book's edition from `d:info/d:edition`:

   ```
   <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:info/d:edition"/>
   ```

   d. Insert the following line to select some legal text from `d:info/d:legalnotice`:

   ```
   <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:info/d:legalnotice"/
   ```

   e. Insert the following line to select the book's ISBN number from `d:info/d:biblioid`:

   ```
   <xsl:apply-templates mode="book.titlepage.verso.auto.mode"
       select="d:info/d:biblioid[@class='isbn']"/>
   ```

   f. Remove the other `xsl:apply-templates` elements as we only want display the title, subtitle, author, editor, legal text, and the ISBN number. The template should look like this:

   ```
   <xsl:template name="book.titlepage.verso">
     <xsl:choose>
       <xsl:when test="d:bookinfo/d:title">
         <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:bookinfo/d:ti
       </xsl:when>
       <xsl:when test="d:info/d:title">
         <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:info/d:title"/
       </xsl:when>
       <xsl:when test="d:title">
         <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:title"/>
       </xsl:when>
     </xsl:choose>
   ```

```
        <xsl:choose>
          <xsl:when test="d:bookinfo/d:subtitle">
            <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:bookinf
          </xsl:when>
          <xsl:when test="d:info/d:subtitle">
            <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:info/d:
          </xsl:when>
          <xsl:when test="d:subtitle">
            <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:subtitl
          </xsl:when>
        </xsl:choose>

        <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:info/d:auth
        <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:info/d:edit
        <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:info/d:lega
        <xsl:apply-templates mode="book.titlepage.verso.auto.mode" select="d:info/d:bibl
      </xsl:template>
```

7. Insert the constructed `booktitlepage.xsl` into your customization layer `mybooktitlepage.xsl`:

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

  <xsl:import href="https://cdn.docbook.org/release/xsl/current/fo/docbook.xsl"/>
  <xsl:include href="booktitlepage.xsl"/>
  <!-- More customizations hidden -->
</xsl:stylesheet>
```

8. Build your book as usual with your customization layer.

Discussion

There are several reasons to customize a title page:

- You want a different order of the default elements
- You want to show or hide elements
- You want to insert corporate logos, links, or other graphical illustrations
- You want to distinguish between a document in draft or in final state.

All these requirements can be done, either with the direct or indirect method. Apart from your own preferences, use the direct method if you have more complicated conditions which can not be expressed by simple title page definitions. Sometimes it is faster to just add the respective named template instead of going through the indirect method.

Regardless which method you prefer, it ends up with the same named templates. For a book title page, the named templates are called in the following order, starting with `book.titlepage`:

```
book.titlepage
  book.titlepage.before.recto
  book.titlepage.recto
  book.titlepage.before.verso
  book.titlepage.verso
  book.titlepage.separator
```

Usually, the `book.titlepage` template is only customized in rare cases. For example, if you want to revamp the previous schema completely and add additional pages like half titles [http://en.wikipedia.org/wiki/Half_title]. Each of the templates in `book.titlepage` are responsible for a different setup of a title page. We discussed the `book.titlepage.recto` and `book.titlepage.verso` templates already. The templates `book.ti-`

`tlepage.separator` and `book.titlepage.before.verso` contain page breaks, whereas `book.titlepage.before.recto` is empty.

Note, all of the above templates process title page elements in a special mode. As you have seen in the section called "Customizing Title Pages Directly" [121], elements for a recto page are processed in the `book.titlepage.recto.auto.mode` mode. The same principle applies for a verso page and its mode `book.titlepage.verso.auto.mode`. However, not all elements have a template with such modes as not all elements can (and should) appear on a title page. Look into the file `fo/titlepage.templates.xsl` to see which are definied. Assume title and subtitle have already templates for recto and verso pages as well as all the elements which are listed in the original `book.titlepage.verso` and `book.titlepage.recto`.

For example, the `edition` element has certainly no templates for a recto and verso book title page. If you do not have definied one, a fallback mechanism takes place. For this reason, if you call a element in `book.titlepage.verso` and `book.titlepage.recto`, you need also to define the respective templates:

```
<xsl:template match="d:edition" mode="book.titlepage.recto.auto.mode">
   <!-- Add your code for a recto page -->
</xsl:template>

<xsl:template match="d:edition" mode="book.titlepage.verso.auto.mode">
   <!-- Add your code for a verso page -->
</xsl:template>
```

Usually, you want to display the edition on a recto page different than on a verso page.

See Also

- http://www.sagehill.net/docbookxsl/HTMLTitlePage.html
- Section 4.3, "Styling Title Pages" [125]

4.3. Styling Title Pages

Problem

You do not want to overhaul the default title pages, but you want to change small things, like the font size, margins, or other stylistic parameters.

Solution

The difficulty is to find the correct template to customize. Here is a general procedure how to do this:

Procedure 4.1. Finding the Correct Template to Customize for a Title Page

1. Open the file `fo/titlepage.templates.xsl` from your DocBook XSL distribution.
2. Determine the following parameters and write it down or memorize it:
   - The title page which you want to customize (usually something like book, chapter, etc.), refered as *`DIVISION`*
   - The element on that title page, referred as *`ELEMENT`*
   - The side, be it recto (right) or verso (left), refered as *`SIDE`*.
3. Compose a template in your mind from the previous parameters and replace the placeholders:

   ```
   <xsl:template match="ELEMENT" mode="DIVISION.titlepage.SIDE.auto.mode">
   ```
4. Try to find the template from Step 3 [125] in `fo/titlepage.templates.xsl`.
5. If you have found the template depicted in Step 3 [125], copy it to your customization layer. If there is no such template, create a new one with the same signature.
6. Customize the template.
7. Build your document with your customization layer.

Discussion

To make the general explanations a bit more useful, here is a small example: you want to change the font size for the title of a book's recto title page. Applying Procedure 4.1, “Finding the Correct Template to Customize for a Title Page” [125] leads to the following template:

```
<xsl:template match="d:title" mode="book.titlepage.recto.auto.mode">
```

After you have copied it to your customization layer, you can change the `font-size` (marked bold):

```
<xsl:template match="d:title" mode="book.titlepage.recto.auto.mode">
  <fo:block xmlns:fo="http://www.w3.org/1999/XSL/Format"
     xsl:use-attribute-sets="book.titlepage.recto.style"
     text-align="center"
     font-size="40pt"
     space-before="18.6624pt"
     font-weight="bold"
     font-family="{$title.fontset}">
    <xsl:call-template name="division.title">
      <xsl:with-param name="node" select="ancestor-or-self::d:book[1]"/>
    </xsl:call-template>
  </fo:block>
</xsl:template>
```

In the previous example, the font size was changed from the original value of `24.8832pt` to `40pt`. The other objects are unchanged. Other attributes can be changed as needed.

See Also

Section 4.2, “Designing a Title Page” [119]

4.4. Influencing the Leading

Tony Graham

Problem

You want to change the leading between consecutive lines of text.

Solution

Leading is the distance from one baseline to the next, or in other words, the “interlinear space.” It is not the same as the line height. Unfortunately, the FO specification named the property for influencing the leading line-height which makes it confusing.

The DocBook stylesheets introduce the parameter *`line-height`* with the same name and functionality. It acts as a placeholder for the above line-height property. The following examples do all the same (except for the value `normal`). It is assumed, you have a base font size of 10pt; negative values in line-height are not allowed:

`normal`
: This is the default value. The real value is set by the formatter and depends on the font size, usually it is a value between 1.0 and 1.2. *Say more about the real value*

Length
: Sets the line height of the respective value:

```
<xsl:param name="line-height">12pt</xsl:param>
```

The leading is 2pt (12pt _ 10pt).

Number
: Sets the line height is the result of the value multiplied by the font size:

```
<xsl:param name="line-height">1.2</xsl:param>
```

Percentage
: Sets the line height is the result of the percentage value multiplied by the font size:

```
<xsl:param name="line-height">120%</xsl:param>
```

Discussion

Unfortunately, the situation with line height in FO is a bit more complicated as it first seems. Usually, this is what most people expects:

```
First line
 [Gap]
Second Line
 [Gap]
Third Line
 [Gap]
```

However, the line-height property is a “half-leading” value which are added before and after a line. This leads to the more realistic picture:

```
 [1/2 Gap]
First line
 [1/2 Gap]
 [1/2 Gap]
Second Line
 [1/2 Gap]
 [1/2 Gap]
Third Line
```

```
[1/2 Gap]
```

If the line height is constant in each line, the effect is the same (except at the top or bottom of a reference area). According to the FO specification of `line-height` the FO property is a *compound datatype [http://www.w3.org/TR/xsl11/#spacecond]* which has minimum, optimum, maximum, conditionally, and precedence. The minimum, optimum, and maximum constraints are length

That said, leading has a direct influence on how easy a text is read. If the value is too small, consecutive lines gets overlapped. If the value is too big, the cohersion of text gets lost. Most text requires positive leading. Usually 9pt/11pt, 10pt/12pt, 11pt/13pt, and 12pt/15pt are the most common (the two values depicting the font size and line height).

According to Robert Bringhurst in his book *The Elements of Typographic Style* he recommends more lead in the following situations (cited from his book from page 37):

- Dark faces need more lead than light ones
- Large-bodied faces need more lead than smaller-bodied ones
- Unserifed faces often need more lead (or a shorter line) than their serifed counterparts
- Text with thickened by superscripts, subscripts, mathematical expressions, or the frequent use of full capitals

See Also

- http://en.wikipedia.org/wiki/Leading, the Wikipedia article about leading
- http://www.w3.org/TR/xsl11/#line-height, the definition of line-height in the XSL-FO specification
- http://w3-org.9356.n7.nabble.com/Line-height-vs-line-spacing-td213282.html, Tony Graham about "Line height vs. line spacing"

4.5. Hyphenating URLs

Problem

You have URLs or paths which you want to hyphenate correctly. The URLs have to break on slashes or other characters only, but not between words.

Solution

The hyphenation of URLs in the DocBook stylesheets are controlled by two parameters: *ulink.hyphenate [http://docbook.sourceforge.net/release/xsl/current/doc/fo/ulink.hyphenate.html]* and *ulink.hyphenate.chars [http://docbook.sourceforge.net/release/xsl/current/doc/fo/ulink.hyphenate.chars.html]*. The first parameter, if not empty, turns on hyphenation. Specify a hyphenation character, usually either a Unicode soft hyphen (U+00AD) or a Unicode zero-width space (U+200B).

The second parameter, *ulink.hyphenate.chars*, let you define your allowable hyphenation points. The default value is a slash (/), but URLs can contain more characters where it is desirable to hyphenate. For this reason, the DocBook parameter reference recommends the following value:

```
<xsl:param name="ulink.hyphenate.chars">:/@&?.#</xsl:param
```

Discussion

The easiest way is to set the parameters *ulink.hyphenate.chars* and *ulink.hyphenate* to the values showed in the last section and be happy. However, for professional needs, this is not enough. The parameters and embedded algorithm do not take into account protocols, for example `http`. Protocols begin with the schema followed by `://` as in `http://`. In some situations (although rare), a hyphenation can occur between the double slashes or before the colon. Furthermore, according to the *Chicago Manual of Style*, it is desirable to distinguish characters *before* and *after* the hyphenation takes place.

All these requirements are implemented in the stylesheet showed in Example 4.1, "`hyphenate-url.xsl`" [129]. It cuts off the protocol with its `://` and iterates through each characters and checks, if a hyphenation point needs to be inserted before or after it.

Example 4.1. `hyphenate-url.xsl`

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

<xsl:template name="hyphenate-url">
  <xsl:param name="url" select="''"/>

  <!-- Remove the "schema://" prefix, so it disturbs not the
       algorithm in "hyphenate-url-string" -->
  <xsl:choose>
   <xsl:when test="$ulink.hyphenate = ''">
      <xsl:value-of select="$url"/>
   </xsl:when>
   <xsl:when test="contains($url, '://')">
      <xsl:value-of select="substring-before($url, '://')"/>
      <xsl:text>://</xsl:text>
      <xsl:copy-of select="$ulink.hyphenate"/>
      <xsl:call-template name="hyphenate-url-string">
        <xsl:with-param name="url" select="substring-after($url, '://')"/>
      </xsl:call-template>
   </xsl:when>
```

```
   <xsl:otherwise>
      <xsl:call-template name="hyphenate-url-string">
        <xsl:with-param name="url" select="normalize-space($url)"/>
      </xsl:call-template>
   </xsl:otherwise>
  </xsl:choose>
</xsl:template>


<xsl:template name="hyphenate-url-string">
  <xsl:param name="url" select="''"/>
  <xsl:variable name="char" select="substring($url, 1,1)"/>

  <xsl:choose>
   <xsl:when test="$url=''"/>
   <!-- Insert breakpoint _before_ the character -->
   <xsl:when test="contains($ulink.hyphenate.before.chars, $char)">
     <xsl:value-of select="concat($ulink.hyphenate, $char)"/>
     <xsl:call-template name="hyphenate-url-string">
       <xsl:with-param name="url" select="substring($url, 2)"/>
     </xsl:call-template>
   </xsl:when>
   <!-- Insert breakpoint _after_ the character -->
   <xsl:when test="contains($ulink.hyphenate.after.chars, $char)">
     <xsl:value-of select="concat($char, $ulink.hyphenate)"/>
     <xsl:call-template name="hyphenate-url-string">
       <xsl:with-param name="url" select="substring($url, 2)"/>
     </xsl:call-template>
   </xsl:when>
   <xsl:otherwise>
     <xsl:value-of select="$char"/>
     <xsl:call-template name="hyphenate-url-string">
       <xsl:with-param name="url" select="substring($url, 2)"/>
     </xsl:call-template>
   </xsl:otherwise>
  </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

Include the above stylesheet into your customization layer with additionally the following parameters:

```
<!-- Insert breakpoint /before/ the following characters: -->
<xsl:param name="ulink.hyphenate.before.chars"
  >.,%?&amp;#\-+{_</xsl:param>
<!-- Insert breakpoint /after/ the following characters: -->
<xsl:param name="ulink.hyphenate.after.chars"
  >/:@=};</xsl:param>
```

4.6. Creating Initials and Drop Caps

Problem

You need a “drop cap”, or Initial [http://en.wikipedia.org/wiki/Initial], which is a “a letter at the beginning of a work, a chapter, or a paragraph that is larger than the rest of the text.”

Solution

Typographically, you have different options to place initials:

- on the same baseline as the first line of text,
- as floats, embedded between two or more lines,
- besides the left margin of the text block without indentation.

Additionally you can design your initials as graphics, but we will come to this later.

Place Initials on the Same Baseline

This is the easiest method to implement. The stylesheet normalizes the text node of the first paragraph, extract the first character and wrap it between an `fo:inline` element with font size, family, and weight:

Example 4.2. `initials-baseline.xsl`

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

  <xsl:param name="drop.caps.size">20pt</xsl:param>
  <xsl:param name="drop.caps.font-family" select="$body.font.family"/>
  <xsl:param name="drop.caps.font-weight">bold</xsl:param>

  <xsl:template name="create.initial">
    <xsl:param name="initial" select="substring(normalize-space(.),1,1)"/>
    <fo:inline>
      <fo:inline font-size="{$drop.caps.size}"
        font-weight="{$drop.caps.font-weight}"
        font-family="{$drop.caps.font-family}">
        <xsl:copy-of select="$initial"/>
      </fo:inline>
      <xsl:value-of select="substring(normalize-space(.), 2)"/>
    </fo:inline>
  </xsl:template>
</xsl:stylesheet>
```

To use it, include it into your customization layer. It is usually called in text nodes like the first paragraph from a section which parent is an article:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet
[
 <!ENTITY db "https://cdn.docbook.org/release/xsl/current">
]>
<xsl:stylesheet version="1.0"
  xmlns:d="http://docbook.org/ns/docbook"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:import href="&db;/fo/docbook.xsl"/>
  <xsl:include href="initials-baseline.xsl"/>

  <xsl:template match="d:article/d:section/d:para[1]/text()">
    <xsl:call-template name="create.initial"/>
  </xsl:template>
</xsl:stylesheet>
```

It is easy to extend it to chapters, you just have to copy the template and correct the content in the `match` attribute to:

```
<xsl:template match="d:chapter/d:section/d:para[1]/text()">
```

Implement Initials as Floats

TBD

Place Initials in Left Margin

TBD

Discussion

The solution described in the section called “Place Initials on the Same Baseline” [131] is in most cases sufficient. However, if the first character is *not* a letter, any character is incorrectly made an initial.

See Also

Find information about the property `line-height` in Section 4.4, “Influencing the Leading” [127].

4.7. Numbering Part, Chapter, Appendix, and other Titles

Problem

You want a number in front of your structural elements like `appendix`, `chapter`, `section`, etc.

Solution

The DocBook XSL stylesheets provide already a decent numbering scheme for high level divisions like parts, appendices, chapters, and references. The most common use is numbering section. If you want to automatically number section titles, set the *`section.autolabel`* parameter either manually or through a customization layer, for example:

```
<xsl:param name="section.autolabel" select="1"/>
```

Discussion

The DocBook XSL stylesheets contain autolabel parameters for the common elements `appendix`, `chapter`, `part`, `preface`, `qandadiv`, `reference`, and `section`. These can be set individually to change the numbering of for the same elements. The default values are shown in Table 4.1, “Autolabel Parameters and their Default Values” [133].

Table 4.1. Autolabel Parameters and their Default Values

Parameter	Default	Example
`appendix.autolabel`	`A`	Appendix A. Python 3
`chapter.autolabel`	`1`	Chapter 1. A Tutorial Introduction
`part.autolabel`	`I`	Part I. The Python Language
`preface.autolabel`	`0`	Prefaces are usually not numbered
`qandadiv.autolabel`	`1`	3.1 Installation Questions
`reference.autolabel`	`I`	I. Python Reference
`section.autolabel`	`0`	Not numbered by default

The autolabel parameters can contain the following values:

Value `0`
: A value of zero disables the automatic numbering. For example, if you want to disable automatic numbering for appendix elements, use this:

```
<xsl:param name="appendix.autolabel" select="0"/>
```

Value `1`
: A value of one enables the automatic numbering and uses arabic numerals. For example, section numbering is disabled by default. Use the code in the Solution [133] section.

String
: A string enables a different automatic numbering schema, for example roman numerals. See the following table to get an overview.

Table 4.2. Possible Formats for Autolabel Parameters

Value	Alternative Value	Style	Example
`1`	`arabic`	Arabic numerals	1, 2, 3, 4, …
`I`	`upperroman`	Uppercase roman numerals	I, II, III, IV, …
`i`	`lowerroman`	Lowercase roman numerals	i, ii, iii, iv, …
`A`	`upperalpha`	Uppercase letters	A, B, C, D, …
`a`	`loweralpha`	Lowercase letters	a, b, c, d, …

Value	Alternative Value	Style	Example
١	arabicindic	Arabic-Indic numerals	# ,# ,# ,#, …

Mostly you will probably want sections to be numbered including its parent numbering. If you do not touch *chapter.autolabel*, use the parameter *section.label.includes.component.label* and set it to 1:

```
<xsl:param name="section.autolabel" select="1"/>
<xsl:param name="section.label.includes.component.label" select="1"/>
```

With the following parameters you can further customize your numbering:

component.label.includes.part.label
: Controls if appendix or chapter number labels are prefixed with their contained part label.

label.from.part
: Defines if the components inside a part is renumbered or not. A value of non-zero restarts the chapter number and counts again from one. Use this numbering if you want unambiguous numerals.

  The value zero (the default) restarts the component number throughout each book.

section.autolabel.max.depth
: Controls which sections get a number. By default, all sections are get numbered (default value is 8.) If you want to number only sections at level 1, set the parameter to the value 1.

Let us assume the following structure in a book:

```
Part: The Python Language
  Chapter: A Tutorial Introduction
     Section: Running Python
     Section: Variables

  Chapter: Lexical Conventions and Syntax
     Section: Line Structure and Indentation
     Section: Identifiers and Reserved Words

Part: The Python Library
  Chapter: Build-In Functions
     Section: Build-In Functions and Types
     Section: Build-In Exceptions
  Chapter: Python Runtime Services
     Section: atext
     Section: copy
```

Table 4.3. Parameter Combinations

Parameters	Result

No parameters set, using the default settings	I. The Python Language
	  1. A Tutorial Introduction
	  Running Python
	  Variables
	  2. Lexical Conventions and Syntax
	  Line Structure and Indentation
	  Identifiers and Reserved Words
	II. The Python Library
	  3. Build-In Functions
	  Build-In Functions and Types
	  Build-In Exceptions
	  4. Python Runtime Services
	  atext
	  copy

section.autolabel=1	I. The Python Language 1. A Tutorial Introduction

Parameters	Result
	1. Running Python 2. Variables 2. Lexical Conventions and Syntax 1. Line Structure and Indentation 2. Identifiers and Reserved Words II. The Python Library 3. Build-In Functions 1. Build-In Functions and Types 2. Build-In Exceptions 4. Python Runtime Services 1. atexit 2. copy
section.autolabel=1 section.label.includes.component.label=1	I. The Python Language 1. A Tutorial Introduction 1.1. Running Python 1.2. Variables 2. Lexical Conventions and Syntax 2.1. Line Structure and Indentation 2.2. Identifiers and Reserved Words II. The Python Library 3. Build-In Functions 3.1. Build-In Functions and Types 3.2. Build-In Exceptions 4. Python Runtime Services 4.1. atexit 4.2. copy
section.autolabel=1 section.label.includes.component.label=1 component.label.includes.part.label	I. The Python Language I.1. A Tutorial Introduction I.1.1. Running Python I.1.2. Variables I.2. Lexical Conventions and Syntax I.2.1. Line Structure and Indentation I.2.2. Identifiers and Reserved Words II. The Python Library II.3. Build-In Functions II.3.1. Build-In Functions and Types II.3.2. Build-In Exceptions II.4. Python Runtime Services II.4.1. atexit II.4.2. copy
section.autolabel=1 section.label.includes.component.label=1 component.label.includes.part.label label.from.part=1	I. The Python Language I.1. A Tutorial Introduction I.1.1. Running Python I.1.2. Variables I.2. Lexical Conventions and Syntax I.2.1. Line Structure and Indentation I.2.2. Identifiers and Reserved Words II. The Python Library II.1. Build-In Functions II.1.1. Build-In Functions and Types II.1.2. Build-In Exceptions II.2. Python Runtime Services II.2.1. atexit II.2.2. copy

However, when dealing with other numbering systems, the above parameters are not enough. To support, for example, Japanese numbering, you need to customize the named template `autolabel.format` from `common/labels.xsl`.

Example 4.3. Extending `autolabel.format`

```
<xsl:template name="autolabel.format">
  <xsl:param name="context" select="."/>
  <xsl:param name="format"/>

  <xsl:choose>
    <xsl:when test="string($format) != 0">
      <xsl:choose>
        <xsl:when test="string($format)='001'">
          <xsl:value-of select="$format"/>
        </xsl:when>
        <xsl:when test="$format='loweralpha' or $format='a'">
          <xsl:value-of select="'a'"/>
        </xsl:when>
        <xsl:when test="$format='lowerroman' or $format='i'">
          <xsl:value-of select="'i'"/>
        </xsl:when>
        <xsl:when test="$format='upperalpha' or $format='A'">
          <xsl:value-of select="'A'"/>
        </xsl:when>
        <xsl:when test="$format='upperroman' or $format='I'">
          <xsl:value-of select="'I'"/>
        </xsl:when>
        <xsl:when test="$format='arabicindic' or $format='#'">
          <xsl:value-of select="'#'"/>
        </xsl:when>
        <xsl:when test="$format='japanese' or $format='&#x4e00;'">
          <xsl:value-of select="'&#x4e00;'"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:message>
            <xsl:text>Unexpected </xsl:text>
            <xsl:value-of select="local-name(.)"/>
            <xsl:text>.autolabel value: </xsl:text>
            <xsl:value-of select="$format"/>
            <xsl:text>; using default.</xsl:text>
          </xsl:message>
          <xsl:call-template name="default.autolabel.format"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>
  </xsl:choose>
</xsl:template>
```

Unfortunately, the **xsltproc** processor does not support this currently (as in version 1.1.24). Only Saxon does it correct.

See Also

- http://www.sagehill.net/docbookxsl/SectionNumbering.html
- http://en.wikipedia.org/wiki/Category:Numerals

4.8. Inserting Graphics on Title Pages

Peter Schmelzer

Problem

You need to know how to insert a graphic to your titlepage.

Solution

To insert a graphic to one of your title pages, you need to create a template for titlepages as described in Section 4.2, “Designing a Title Page” [119].

To add your graphic, the XSL-FO specification has defined an `fo:block-container` to encapsulate an `fo:external-graphics` element. Additionally, there are many, many options to modify your graphic on the title pages like scaling or positioning. The following listing show how to use both elements.

```
<fo:block space-before="1em" space-after="1em">
    <fo:external-graphic src="{$img.src.path}/your-graphic.svg"/>
</fo:block>
```

Depending on the position within the customized title element, where you want to place your graphic, you must place the `fo:block-container` before or after the title block-container.

The following example places the graphic after the document title. The graphic is scaled by 50% and the aspect ratio is preserved.

```
<xsl:template match="d:title" mode="book.titlepage.recto.auto.mode">
    <fo:block xmlns:fo="http://www.w3.org/1999/XSL/Format"
        xsl:use-attribute-sets="book.titlepage.recto.style" text-align="center"
        font-size="32pt" space-before="18.6624pt" font-weight="bold">
        <xsl:call-template name="division.title">
            <xsl:with-param name="node" select="ancestor-or-self::d:book[1]"/>
        </xsl:call-template>
        <fo:block space-before="1em" space-after="1em">
            <fo:external-graphic src="{$img.src.path}/your-graphic.svg" content-height="5
                scaling="uniform"/>
         </fo:block>
    </fo:block>
</xsl:template>
```

Discussion

It's a good idea to use only supported high resolution bitmap graphics (300x300 dpi) or supported vector graphics for your title page. Otherwise you may notice that your graphic looks like small blocks.

Note

Your graphic is placed between the left and right margins of your title page. Depending on the graphic size, the graphic will be scaled up or down automatically.

See Also

- The W3C recommendation for fo:external-graphic http://www.w3.org/TR/xsl/#fo_external-graphic.
- How to Bring XSL-FO External Graphic to Fronthttps://stackoverflow.com/questions/45600101/how-to-bring-xsl-fo-external-graphic-to-front
- Supported graphic formats http://www.sagehill.net/docbookxsl/GraphicFormats.html

4.9. Inserting Text and Logos into Page Headers and Footers

Peter Schmelzer

Problem

You need to know how to insert text and a logo in your page header.

Solution

This example solution will place a logo on top right side of a single-sided document or on the left/right side on a double-sided document. Other placements are possible and will be shown in the *Discussion* section. The logo is at max 1 inch high to avoid conflicts with the body text area. Furthermore, we will place the document title on the opposite of the logo, centered to the logo high. The template for our solution is obtained from 'pagesetup.xsl' style sheet.

A logo on every page in your document will give your work a personal touch or a corporate identity. Furthermore it's a way to protect your hard work against simple duplication through a copier or a simple forwarding of parts of your document.

First of all you need to create a customization layer as described in the following steps. If you have a customization layer style sheet, you can skip the first step, otherwise follow the steps to create your customization layer:

1. Create a customization layer as shown in Section 2.3, “Writing Customization Layers” [35].

2. Include the style sheet from Example 4.4, “Placing a Logo in the Right Page Header (`graphic-pageheader.xsl`)” [138] into your customization layer:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <xsl:stylesheet version="1.0"
     xmlns:fo="http://www.w3.org/1999/XSL/Format"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:d="http://docbook.org/ns/docbook">
     <!-- ... -->
     <xsl:include href="graphic-pageheader.xsl"/>
     <!-- ... -->
   </xsl:stylesheet>
   ```

3. Due to a missing attribute-set for the header or footer content, we use the “header.table” Template from the *pagesetup.xsl* and replace all *display-align="before"* occurrences with **display-align="center"**. Add the modified Template to your customization layer.

   Because the “header.table” Template has more than 150 lines, we skip the printout here and focus on the logo and text placing.

   If you don't want to add only the first part of our solution and later the second one, you can include the hole style sheet in the next step.

4. Include the style sheet Example 4.4, “Placing a Logo in the Right Page Header (`graphic-pageheader.xsl`)” [138] into your customization layer and modify it to fit your needs.

   Subsequent, we will explain the options we will focus on:

   Example 4.4. Placing a Logo in the Right Page Header (`graphic-pageheader.xsl`)

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <xsl:stylesheet version="1.0" xmlns:fo="http://www.w3.org/1999/XSL/Format"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:d="http://docbook.org/ns/docbook">

       <!-- Path to your logo file -->
   ```

```
<xsl:param name="logo.src.path">logo/</xsl:param>
<!-- Name of your logo file -->
<xsl:param name="header.image.filename">Opensource.svg</xsl:param>
<!-- Header rule yes (1) / no (0) -->
<xsl:param name="header.rule" select="0"/>

<xsl:template name="header.table">
[...]
</xsl:template>

<xsl:template name="header.content">
    <xsl:param name="pageclass" select="''"/>
    <xsl:param name="sequence" select="''"/>
    <xsl:param name="position" select="''"/>
    <xsl:param name="gentext-key" select="''"/>

    <fo:block>
        <xsl:choose>
            <xsl:when test="$pageclass = 'titlepage'">
                <!-- No footer on titlepages -->
            </xsl:when>
            <xsl:otherwise>
                <!-- Not a titlepage -->
                <xsl:choose>
                    <xsl:when test="$double.sided = 0">
                        <!-- Start single-sided -->
                        <xsl:choose>
                            <xsl:when test="$position = 'left'">
                                <!-- Nothing or maybe a customer logo -->
                                <fo:block>
                                    <xsl:apply-templates select="."
                                    mode="titleabbrev.markup"/>
                                </fo:block>
                            </xsl:when>
                            <xsl:when test="$position = 'center'">
                                <!-- Nothing -->
                            </xsl:when>
                            <xsl:when test="$position = 'right'">
                                <fo:block>
                                    <fo:external-graphic
                                    content-height="10mm">
                                    <xsl:attribute name="src">
                                    <xsl:call-template
                                    name="fo-external-image">
                                    <xsl:with-param name="filename"
                                    select="concat($logo.src.path, $header.image.f
                                    </xsl:call-template>
                                    </xsl:attribute>
                                    </fo:external-graphic>
                                </fo:block>
                            </xsl:when>
                        </xsl:choose>
                    </xsl:when>
                    <!-- End single-sided -->
                    <xsl:otherwise>
                        <!-- Start double-sided -->
                        <xsl:choose>
                            <xsl:when test="$position = 'left'">
                                <xsl:choose>
                                    <xsl:when
                                    test="$sequence = 'even' or $sequence = 'blank
```

```
                                <fo:block>
                                <fo:external-graphic
                                content-height="10mm">
                                <xsl:attribute name="src">
                                <xsl:call-template
                                name="fo-external-image">
                                <xsl:with-param name="filename"
                                select="concat($logo.src.path, $header.image.filenam
                                />
                                </xsl:call-template>
                                </xsl:attribute>
                                </fo:external-graphic>
                                </fo:block>
                                </xsl:when>
                                <xsl:otherwise>
                                <!-- Start left (odd) -->
                                <fo:block>
                                <xsl:apply-templates select="."
                                mode="titleabbrev.markup"/>
                                </fo:block>
                                <!-- Nothing -->
                                </xsl:otherwise>
                            </xsl:choose>
                        </xsl:when>
                        <!-- End left (odd) -->
                        <xsl:when test="$position = 'center'">
                            <!-- Nothing -->
                        </xsl:when>
                        <!-- Start right (even) -->
                        <xsl:when test="$position = 'right'">
                            <xsl:choose>
                                <xsl:when
                                test="
                                        $sequence = 'even' or
                                        $sequence = 'blank'">
                                <fo:block>
                                <xsl:apply-templates select="."
                                mode="titleabbrev.markup"/>
                                </fo:block>
                                </xsl:when>
                                <xsl:otherwise>
                                <fo:block>
                                <fo:external-graphic
                                content-height="10mm">
                                <xsl:attribute name="src">
                                <xsl:call-template
                                name="fo-external-image">
                                <xsl:with-param name="filename"
                                select="concat($logo.src.path, $header.image.filenam
                                />
                                </xsl:call-template>
                                </xsl:attribute>
                                </fo:external-graphic>
                                </fo:block>
                                </xsl:otherwise>
                            </xsl:choose>
                        </xsl:when>
                    </xsl:choose>
                </xsl:otherwise>
                <!-- Double-sided -->
            </xsl:choose>
```

```
                </xsl:otherwise>
                <!--Not a title page-->
            </xsl:choose>
        </fo:block>
    </xsl:template>
</xsl:stylesheet>
```

1. The directory path to your logos.
2. This is the default value and will be used if no parameters are passed.
3. We assume that your logo should be placed in the header of all pages except title pages. You can switch single- or double-sided with this variable.
4. A `fo:block`-element creates a rectangle area on a page which can contain lists, tables or graphics. In this demo, the document title is visible on the left side of single-sided documents.
5. In this demo, the center position has no content, but you can add a page number for example.
6. This is the place for our logo as described at the top of our solution.
7. To avoid conflicts with the body text area, we limit its height to 10mm. When you specify only one parameter like `height` or `width`, your logo will be scaled with the correct aspect ratio.
9. The template which is called to import and place the logo.
8. The file name is passed on the `src` attribute.
10. Our file defined at the top of our customization layer.

5. Use your customization layer to transform your DocBook document into FO.

Discussion

The stylesheet `graphic-pageheader.xsl` creates a logo in the right edge of your page and uses the parameter *`logo.src.path`* to find the logo. By default, a logo is stored in `logo/`. If your logos are in a different location, change it with your XSLT processor. For example, the following line uses the logo(s) image source path `logo.src.path` parameter to find the logo(s). This parameter is used within the position test 10 [ ] to find the right logo.

```
<xsl:param name="logo.src.path">logo/</xsl:param>
```

See Also

- Full parameter description for `fo:external-graphic`http://www.w3.org/TR/xsl/#fo_external-graphic

---

Chapter 5. (X)HTML Customizations

Abstract

HTML is a very important output format for the DocBook XSL stylesheets as it is not only displayed on Web sites, but also used “inside” ebooks (EPUB). It comes in several variants.

5.1. Introduction

Currently, there are several HTML variants available as target formats. As such, “the” HTML does not exist, only a variety of HTML dialects:

Overview of Different HTML XSLT 1.0 Stylesheets

`eclipse/eclipse.xsl`, `eclipse/eclipse3.xsl`
: Contains stylesheet to document for the Eclipse help system [http://www.ibm.com/developerworks/opensource/library/os-echelp/]. The main stylesheets use code from the HTML 4.x stylesheets.

`html/chunk.xsl`, `html/docbook.xsl`
: Contains stylesheets for the “classical” HTML 4 [http://www.w3.org/TR/html4/].

`htmlhelp/htmlhelp.xsl`
: Contains stylesheets for HTMLHelp [http://en.wikipedia.org/wiki/Microsoft_Compiled_HTML_Help], the standard help system from Windows 3.0 through Windows XP. The main stylesheet `htmlhelp.xsl` uses code from the HTML 4.x stylesheets.

`webhelp/xsl/webhelp.xsl`
: Contains the result of the GSOC 2010 and creates a Web page with a sidebar, search form, and hierarchical navigation. The stylesheet incorporates the XHTML stylesheets.

`website/chunk-website.xsl`, `website/website.xsl`
: Contains stylesheets to create a “Web site”.

`xhtml/chunk.xsl`, `xhtml/docbook.xsl`
: Contains stylesheets for XHTML 1.0 [http://www.w3.org/TR/xhtml1/] (“transitional XHTML”). These stylesheets are generated through a XSLT transformation from the HTML core stylesheets.

`xhtml-1_1/chunk.xsl`, `xhtml-1_1/docbook.xsl`
: Contains stylesheets for XHTML 1.1 [http://www.w3.org/TR/xhtml11/]. Like the one for XHTML, they are generated through a XSLT transformation. In comparison, the XHTML 1.1 omits certain attributes which were allowed in XHTML or HTML. The missing part must be implemented by CSS. The XHTML 1.1 stylesheets are also used for EPUB creation.

`xhtml5/chunk.xsl`, `xhtml5/docbook.xsl`
: Contains stylesheets for HTML 5 [http://dev.w3.org/html5/spec/Overview.html]. These stylesheets customizes the XHTML stylesheets. As such it inherits all the features and parameters. The HTML 5 stylesheets are also used for EPUB3 creation.

5.2. Adding Authors to Table of Contents

Problem

You have a document which is written by different authors (like in proceedings) and want to include the author names in the table of contents.

Solution

You need to customize the `toc.line` template. Proceed as follows:

1. Make sure your chapters, appendices, sections, etc. contain at least one author, like in the following example:

```
<sect1>
  <title>How to Become a Linux Mascott</title>
  <info>
    <author>
      <personname>
        <firstname>Tux</firstname>
        <surname>Penguin</surname>
      </personname>
    </author>
  </info>
  <!-- ... -->
</sect1>
```

2. Create a customization layer first as shown in Section 2.3, “Writing Customization Layers” [35].

3. Add the following line to your customization layer:

```
<xsl:include href="autotoc.xsl"/>
```

4. Create a new file `autotoc.xsl` in the same directory, with the following contents:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml">

  <xsl:template name="toc.line">
    <xsl:param name="toc-context" select="."/>
    <xsl:param name="depth" select="1"/>
    <xsl:param name="depth.from.context" select="8"/>

    <xsl:variable name="author" select="*/author|*/authorgroup/author"/>

    <xsl:if test="$author">
      <span class="author">
        <xsl:call-template name="person.name.list">
          <xsl:with-param name="person.list" select="$author"/>
        </xsl:call-template>
        <xsl:text>: </xsl:text>
      </span>
    </xsl:if>
    <span class="{local-name(.)}">
      <xsl:if test="$autotoc.label.in.hyperlink = 0">
        <xsl:variable name="label">
          <xsl:apply-templates select="." mode="label.markup"/>
        </xsl:variable>
        <xsl:copy-of select="$label"/>
        <xsl:if test="$label != ''">
          <xsl:value-of select="$autotoc.label.separator"/>
```

```
          </xsl:if>
        </xsl:if>

        <a>
          <xsl:attribute name="href">
            <xsl:call-template name="href.target">
              <xsl:with-param name="context" select="$toc-context"/>
              <xsl:with-param name="toc-context" select="$toc-context"/>
            </xsl:call-template>
          </xsl:attribute>

          <xsl:if test="not($autotoc.label.in.hyperlink = 0)">
            <xsl:variable name="label">
              <xsl:apply-templates select="." mode="label.markup"/>
            </xsl:variable>
            <xsl:copy-of select="$label"/>
            <xsl:if test="$label != ''">
              <xsl:value-of select="$autotoc.label.separator"/>
            </xsl:if>
          </xsl:if>

          <xsl:apply-templates select="." mode="titleabbrev.markup"/>
        </a>
      </span>
    </xsl:template>
  </xsl:stylesheet>
```

1. Collects all author nodes regardless if they appear in `info` or `authorgroup`
2. Checks, if `author` nodes were found
3. Calls `person.name.list` and pass the nodes from 1 [ ]
4. Contains the original code from `autotoc.xsl` and creates the title entry for the table of contents

5. Rebuild your document with your customization layer.

When you process the above section with the customization layer, you will receive the following line:

```
Tux Penguin: How to Become a Linux Mascott
```

Or in HTML notation:

```
<span class="author">Tux Penguin: </span>
<span class="section">
  <a href="...">How to Become a Linux Mascott</a>
</span>
```

Depending on the parameter *`section.autolabel`* the title can be prefixed with a number.

Discussion

The previous stylesheet works also for one or more authors. It does not matter if the `author` elements appear as a direct child of `info` or `authorgroup`. The following code produces the same string:

```
<info>
  <author>
    <personname>
      <firstname>Tux</firstname>
      <surname>Penguin</surname>
    </personname>
  </author>
  <author>
    <personname>
      <firstname>Wilber</firstname>
```

```
<info>
  <authorgroup>
    <author>
      <personname>
        <firstname>Tux</firstname>
        <surname>Penguin</surname>
      </personname>
    </author>
    <author>
      <personname>
```

```
      <surname>Gimp</surname>
    </personname>
  </author>
</info>
```

```
      <firstname>Wilber</firstname>
      <surname>Gimp</surname>
    </personname>
  </author>
 </authorgroup>
</info>
```

Even if you mix `author` and `authorgroup` you will get the correct string output.

You can also use elements other than `author`. To detect `editor`, extend the variable `author` as follows:

```
<xsl:variable name="author"
      select="*/author|*/editor|*/authorgroup/author|*/authorgroup/editor"/>
```

See Also

- http://www.sagehill.net/docbookxsl/TOCcontrol.html

5.3. Creating Permalinks

Problem

You need for every appendix, chapter, section, etc. a small hint (often depicted as "¶") which points to the division itself. This "permalink" simplies the retrieval and indentification of such a division.

Solution

You need an ID attribute on your division (chapter, section, examples, etc.) to make it work. The `permalinks.xsl` stylesheet is a named template which expects an *`id`* parameter.

Example 5.1. Permalink Stylesheet (`permalinks.xsl`)

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

  <xsl:param name="generate.permalink" select="1"/>
  <xsl:param name="permalink.text">¶</xsl:param>

  <xsl:template name="permalink">
    <xsl:param name="node" select="."/>

    <xsl:if test="$generate.permalink != '0'">
      <span class="permalink">
        <a alt="Permalink" title="Permalink">
          <xsl:attribute name="href">
            <xsl:call-template name="href.target">
              <xsl:with-param name="object"  select="$node"/>
            </xsl:call-template>
          </xsl:attribute>
          <xsl:copy-of select="$permalink.text"/>
        </a>
      </span>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

1. Contains parameter *`generate.permalink`* to switch on or off permalink creation.
2. Contains parameter *`permalink.text`* which is inserted as the permalink text.
3. Creates a `span` element as a wrapper element which contains all information about the permalink.
4. Creates a HTML `a` element. The pointer in the `href` attribute is used from the *`id`* parameter.
5. Creates a `href` attribute with the correct link, regardless if it points to a chunked file or not.

This is of course not enough. Use the following steps to include it into your customization layers:

1. Create a customization layer as shown in Section 2.3, "Writing Customization Layers" [35].

2. Include the stylesheet from Example 5.1, "Permalink Stylesheet (`permalinks.xsl`)" [148] into your customization layer:

   ```
   <xsl:stylesheet version="1.0"
     xmlns="http://www.w3.org/1999/xhtml"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <!-- ... -->
     <xsl:include href="permalinks.xsl"/>
     <!-- Add here more xsl:include's of the following files -->
   </xsl:stylesheet>
   ```

3. Add `xsl:include` tags to your customization layer with points to one or all of the following files to include permalinks in their respective elements.

   - For appendices, chapters, prefaces, etc. use the `component.title` template (originates from `{xhtml,xhtml-1_1,html}/component.xsl`):

     ```
     <xsl:stylesheet version="1.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:d="http://docbook.org/ns/docbook"
       xmlns="http://www.w3.org/1999/xhtml"
       exclude-result-prefixes="d">

       <xsl:template name="component.title">
         <xsl:param name="node" select="."/>

         <xsl:variable name="level">
           <xsl:choose>
             <xsl:when test="ancestor::d:section">
               <xsl:value-of select="count(ancestor::d:section)+1"/>
             </xsl:when>
             <xsl:when test="ancestor::d:sect5">6</xsl:when>
             <xsl:when test="ancestor::d:sect4">5</xsl:when>
             <xsl:when test="ancestor::d:sect3">4</xsl:when>
             <xsl:when test="ancestor::d:sect2">3</xsl:when>
             <xsl:when test="ancestor::d:sect1">2</xsl:when>
             <xsl:otherwise>1</xsl:otherwise>
           </xsl:choose>
         </xsl:variable>
         <xsl:element name="h{$level+1}" namespace="http://www.w3.org/1999/xhtml">
           <xsl:attribute name="class">title</xsl:attribute>
           <xsl:if test="$generate.id.attributes = 0">
             <xsl:call-template name="anchor">
               <xsl:with-param name="node" select="$node"/>
               <xsl:with-param name="conditional" select="0"/>
             </xsl:call-template>
           </xsl:if>
           <xsl:apply-templates select="$node" mode="object.title.markup">
             <xsl:with-param name="allow-anchors" select="1"/>
           </xsl:apply-templates>
           <xsl:call-template name="permalink">
             <xsl:with-param name="node" select="$node"/>
           </xsl:call-template>
         </xsl:element>
       </xsl:template>
     </xsl:stylesheet>
     ```

   - For figures, examples, procedures, tables, etc. use the `formal.object.heading` template (originates from `{xhtml,xhtml-1_1,html}/formal.xsl`):

     ```
     <xsl:stylesheet version="1.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:d="http://docbook.org/ns/docbook"
       xmlns="http://www.w3.org/1999/xhtml"
       exclude-result-prefixes="d">

       <xsl:template name="formal.object.heading">
         <xsl:param name="object" select="."/>
         <xsl:param name="title">
           <xsl:apply-templates select="$object" mode="object.title.markup">
             <xsl:with-param name="allow-anchors" select="1"/>
           </xsl:apply-templates>
     ```

```
    </xsl:param>
    <p class="title">
      <b><xsl:copy-of select="$title"/></b>
      <xsl:call-template name="permalink">
        <xsl:with-param name="node" select="$object"/>
      </xsl:call-template>
    </p>
  </xsl:template>
</xsl:stylesheet>
```

- For parts, use the `division.title` template (originates from `{xhtml,xhtml-1_1,html}/division.xsl`):

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://docbook.org/ns/docbook"
  xmlns="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="d">

  <xsl:template name="division.title">
    <xsl:param name="node" select="."/>

    <h1>
      <xsl:attribute name="class">title</xsl:attribute>
      <xsl:call-template name="anchor">
        <xsl:with-param name="node" select="$node"/>
        <xsl:with-param name="conditional" select="0"/>
      </xsl:call-template>
      <xsl:apply-templates select="$node" mode="object.title.markup">
        <xsl:with-param name="allow-anchors" select="1"/>
      </xsl:apply-templates>
      <xsl:call-template name="permalink">
        <xsl:with-param name="node" select="$node"/>
      </xsl:call-template>
    </h1>
  </xsl:template>
</xsl:stylesheet>
```

- For glossary terms, use the following template

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://docbook.org/ns/docbook"
  xmlns="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="d">

  <xsl:template match="d:glossentry/d:glossterm">
    <xsl:variable name="id" select="(@id|../@id)[last()]"/>
    <xsl:apply-imports/>
    <xsl:if test="$generate.permalink != 0">
      <xsl:message>Generating permalink for glossterm <xsl:value-of
          select="concat('&quot;', normalize-space(.), '&quot;')"/></xsl:message>
      <xsl:call-template name="permalink">
        <xsl:with-param name="node" select=".."/>
      </xsl:call-template>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

- For sections, use `section.title` (originates from `{xhtml,xhtml-1_1,html}/sections.xsl`):

```
<xsl:stylesheet version="1.0"
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:d="http://docbook.org/ns/docbook"
    xmlns="http://www.w3.org/1999/xhtml" exclude-result-prefixes="d">

    <xsl:template name="section.title">
      <xsl:variable name="section"
        select="(ancestor::d:section |
                 ancestor::d:simplesect|
                 ancestor::d:sect1|
                 ancestor::d:sect2|
                 ancestor::d:sect3|
                 ancestor::d:sect4|
                 ancestor::d:sect5)[last()]"/>

      <xsl:variable name="renderas">
        <xsl:choose>
          <xsl:when test="$section/@renderas = 'sect1'">1</xsl:when>
          <xsl:when test="$section/@renderas = 'sect2'">2</xsl:when>
          <xsl:when test="$section/@renderas = 'sect3'">3</xsl:when>
          <xsl:when test="$section/@renderas = 'sect4'">4</xsl:when>
          <xsl:when test="$section/@renderas = 'sect5'">5</xsl:when>
          <xsl:otherwise><xsl:value-of select="''"/></xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <xsl:variable name="level">
        <xsl:choose>
          <xsl:when test="$renderas != ''">
            <xsl:value-of select="$renderas"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:call-template name="section.level">
              <xsl:with-param name="node" select="$section"/>
            </xsl:call-template>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:variable>

      <xsl:call-template name="section.heading">
        <xsl:with-param name="section" select="$section"/>
        <xsl:with-param name="level" select="$level"/>
        <xsl:with-param name="title">
          <xsl:apply-templates select="$section" mode="object.title.markup">
            <xsl:with-param name="allow-anchors" select="1"/>
          </xsl:apply-templates>
          <xsl:if test="$level = 1">
            <xsl:call-template name="permalink">
              <xsl:with-param name="node" select="$section"/>
            </xsl:call-template>
          </xsl:if>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:template>
  </xsl:stylesheet>
```

4. Use your customization layer to transform your DocBook document into (X)HTML.

Discussion

Permalinks are useful as it allows it easily to copy its URL. The DocBook XSL stylesheets allows the addition of permalinks in different locations, but the above customizations make the permalink more “attached” to its title.

All the above customization files create almost the same structure in regards to its permalink. For example, a chapter title looks like this in HTML:

```
<h2 class="title">The Chapter Title<span
      class="permalink"><a href="..." title="Permalink"
      alt="Permalink">¶</a></span>
</h1>
```

With CSS rules, you can style your permalink whatever you like.

5.4. Creating Simple Navigation in Chapters

Problem

You need a simple navigation for your single HTML file. The navigation should contain for each chapter a link to the next and previous chapters including an up link pointing to the enclosing part or book. This is depicted in the following graphic:

Solution

The following file defines the named template `generate.simple.navigation`:

Example 5.2. `simple-navigation.xsl`

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://docbook.org/ns/docbook"
  xmlns="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="d">

  <xsl:template name="generate.simple.navigation">
    <xsl:param name="node" select="."/>
    <xsl:variable name="prev" select="$node/preceding-sibling::d:chapter"/>
    <xsl:variable name="next" select="$node/following-sibling::d:chapter"/>
    <xsl:variable name="up"   select="$node/parent::d:*"/>

    <div class="chapter-navigation">
      <ul>
        <xsl:if test="count($next) >0">
          <li class="next">
            <xsl:call-template name="gentext.nav.next"/>
            <xsl:text>: </xsl:text>
            <a>
              <xsl:attribute name="href">
                <xsl:call-template name="href.target">
                  <xsl:with-param name="object" select="$next"/>
                </xsl:call-template>
              </xsl:attribute>
              <xsl:value-of select="$next/d:title"/>
            </a>
          </li>
        </xsl:if>
        <xsl:if test="count($prev) >0">
          <li class="prev">
            <xsl:call-template name="gentext.nav.prev"/>
            <xsl:text>: </xsl:text>
            <a>
              <xsl:attribute name="href">
                <xsl:call-template name="href.target">
                  <xsl:with-param name="object" select="$prev"/>
                </xsl:call-template>
              </xsl:attribute>
              <xsl:value-of select="$prev/d:title"/>
            </a>
          </li>
        </xsl:if>
        <xsl:if test="count($up) >0">
          <li class="up">
```

```
            <xsl:call-template name="gentext.nav.up"/>
            <xsl:text>: </xsl:text>
            <a>
              <xsl:attribute name="href">
                <xsl:call-template name="href.target">
                  <xsl:with-param name="object" select="$up"/>
                </xsl:call-template>
              </xsl:attribute>
              <xsl:value-of select="$up/d:title"/>
            </a>
          </li>
        </xsl:if>
      </ul>
    </div>
  </xsl:template>
</xsl:stylesheet>
```

This is not enough, of course. Use the following steps to include it into your customization layers:

1. Create a customization layer as shown in Section 2.3, “Writing Customization Layers” [35].

2. Include the stylesheet from Example 5.2, “`simple-navigation.xsl`” [153] into your customization layer:

   Example 5.3. `db-simple-navigation.xsl`

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE xsl:stylesheet [
     <!ENTITY db "https://cdn.docbook.org/release/xsl/current">
   ]>
   <xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:d="http://docbook.org/ns/docbook"
     xmlns="http://www.w3.org/1999/xhtml"
     exclude-result-prefixes="d">

     <xsl:import href="&db;/xhtml/docbook.xsl"/>
     <xsl:import href="simple-navigation.xsl"/>

     <xsl:param name="generate.simple.navigation" select="1"/>
     <xsl:param name="html.stylesheet">book.css</xsl:param>

     <xsl:template name="chapter.titlepage.before.recto">
       <xsl:if test="$generate.simple.navigation != 0">
         <xsl:call-template name="generate.simple.navigation"/>
       </xsl:if>
     </xsl:template>
   </xsl:stylesheet>
   ```

3. Use the file `db-simple-navigation.xsl` to transform your documents.

If you want to switch on or off the above behaviour, use the parameter *`generate.simple.navigation`*[1] and set it to `0`.

Discussion

The named template `generate.simple.navigation` works as follows:

1. Three variables are definied for the previous, next, and up links. All depends on the `node` parameter which is by default the context node. This is useful as you can change the behaviour very easily by just changing the context node when calling the template with `xsl:call-template`.

[1] It is not an error to have a parameter with the same name as a named template.

2. The `prev` and `next` variables are filled with the previous or next chapter elements along their sibling axis. In comparison, the `up` variable needs to find its enclosing element; it uses the `parent` axis for this purpose.

   If there are no previous or next chapters, the node set will be empty. For the `up` variable, this depends on where a chapter belongs; usually it is enclosed in a `book` or `part` element. As we need only the parent element As such this variable can not be empty (only if the chapter would be the root element.)

3. The simple navigation is implemented as a unordered list inside a `div` element. The enclosing `div` element is used to make styling with CSS easier.

4. With `xsl:if` we check the amount of nodes in our `prev` and `next` variables. Only if the variable contains more than zero nodes it will a listitem created.

5. If we create a listitem, we want to insert a text to help our readers and to distinguish the different directions. The text has to be independent from the used language; so hard-coding the text would not be sufficient. For this reason, the DocBook stylesheets contains named templates to generate localized text. The next, previous, and up link texts are created through the named templates `generate.nav.next`, `generate.nav.prev`, and `generate.nav.up`.

6. As the generated localized text contains only the text without interpunctations, we append ": ".

7. We create the link with the `a` link. The needed `href` attribute is created by calling `href.target` (origin `html.xsl`). This template is responsible for the correct linking value.

8. The content of the `a` element is inserted from the `title` element of the corresponding node. As we are only interested in the string value, we can use `xsl:value-of`.

The `generate.simple.navigation` template matches only chapters along the sibling axis. As such it can not find a glossary or appendix after a chapter. If you want to create links for *any* component elements, not only chapters, you need to change the variables `prev` and `next`. A first attempt would lead to this:

```
<xsl:variable name="prev" select="$node/preceding-sibling::d:*[1]"/>
<xsl:variable name="next" select="$node/following-sibling::d:*[1]"/>
```

This works for the `next` variables. However, our expression for the `prev` variable contains a bug. Consider the following structure:

```
book
  title
  info
  chapter
  ...
```

The `preceding-sibling` axis returns the `info` and `title` elements. This is not what you want as these are no component elements for our `prev` link. In that case we need an expression to filter out the unwanted elements. This is done with an predicate:

```
<xsl:variable name="prev" select="$node/preceding-sibling::d:*[not(self::d:title|self::d:
```

That expression first creates a node set with *all* preceding elements. In a second step they are check against the term `not(self::d:title|self::d:info)`; only those elements remain in the node set which are not `title` or `info` elements. In our above example, this leads to a node set with zero nodes and is exactly what we wanted to achieve.

With the previous change, we allow *any* structural element to be included in a next or previous link. However, we show our navigation links only in chapter titlepages at the moment (the named template `chapter.titlepage.before.recto`.) We need to extend the stylesheet `db-simple-navigation.xsl` to allow elements like `appendix`, `glossary`, etc. Refer to the content modell of DocBook´s `book [http://www.docbook.org/tdg51/en/html/book.html]` element for details. For an `appendix` this looks like this:

```
<xsl:template name="appendix.titlepage.before.recto">
  <xsl:if test="$generate.simple.navigation != 0">
    <xsl:call-template name="generate.simple.navigation"/>
  </xsl:if>
</xsl:template>
```

For the other elements this is exactly the same except the name. Just use the element name and append `.titlepage.before.recto`.

See Also

5.5. Implementing “Breadcrumbs”

Problem

You want to display a “path” to your current chapter, section etc. to improve fast jumping to parent structures.

Solution

Use breadcrumbs[2] to improve navigation. Use the following procedure to create breadcrumbs for your documents:

1. Create a customization layer as shown in Section 2.3, “Writing Customization Layers” [35].

2. Create a file `breadcrumbs.xsl` with the following content:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:d="http://docbook.org/ns/docbook"
  xmlns="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="d">

  <xsl:param name="breadcrumbs.separator" select="' > '"/>

  <xsl:template name="generate.breadcrumbs">
    <xsl:param name="current.node" select="."/>
    <div class="breadcrumbs">
      <xsl:for-each select="$current.node/ancestor::*">
        <span class="breadcrumb-link">
          <a>
            <xsl:attribute name="href">
              <xsl:call-template name="href.target">
                <xsl:with-param name="object" select="."/>
                <xsl:with-param name="context" select="$current.node"/>
              </xsl:call-template>
            </xsl:attribute>
            <xsl:apply-templates select="." mode="title.markup"/>
          </a>
        </span>
        <xsl:copy-of select="$breadcrumbs.separator"/>
      </xsl:for-each>
      <!-- Display the current node, but not as a link -->
      <span class="breadcrumb-node">
        <xsl:apply-templates select="$current.node" mode="title.markup"/>
      </span>
    </div>
  </xsl:template>
</xsl:stylesheet>
```

   1. Parameter to separate each component
   2. Iterates over the ancestor axis
   3. Creates the `href` attribute by calling `href.target` to create appropriate link references even for chunked output
   4. Inserts the processed title
   5. Inserts the breadcrumbs separator
   6. Inserts the processed title of the current node but not as a link

3. Include `breadcrumbs.xsl` into your customization layer from Step 1 [157]:

```
<xsl:include href="breadcrumbs.xsl"/>
```

[2] Breadcrumbs are a navigation aid and show the titles from the top page to the current page. The term origins from the trail of breadcrumbs left by Hänsel and Gretel in the popular fairytale written by the Brothers Grimm.

4. Add the following code into your customization layer:

```
<xsl:template name="user.header.content">
  <xsl:call-template name="generate.breadcrumb"/>
</xsl:template>
```

5. Build your document with your customization layer.

For example, consider the current topic. It is embedded into this nested structure:

```
book: The DoCookBook
  chapter: (X)HTML Customizations
     sect1: Implementing “Breadcrumbs”
```

The above `breadcrumbs.xsl` stylesheets creates this HTML code when the current node is this topic:

```
<div class="breadcrumbs">
  <span><a href="...X...">The DoCookBook</a></span>
  >
  <span><a href="...Y...">(X)HTML Customizations</a></span>
  >
  <span class="breadcrumb-node">Implementing “Breadcrumbs”</span>
</div>
```

Leading to the following appearance:

```
The DoCookBook > (X)HTML Customizations > Implementing “Breadcrumbs”
```

Whereas all `span`s contains links to their respective components except the last one.

Discussion

TBD

See Also

- http://www.sagehill.net/docbookxsl/HTMLHeaders.html#BreadCrumbs
- Section 5.4, “Creating Simple Navigation in Chapters” [153]

5.6. Moving the Table of Contents (TOC)

Problem

You have a book and want the table of contents appear after the main content.

Solution

The processing of a book is handled in the `xhtml/division.xsl` file. The template needs to be copied and inserted in your customization layer. To move your table of contents, do the following:

1. Create a customization layer as shown in Section 2.3, “Writing Customization Layers” [35].

2. Create a file `move-toc.xsl` with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet  version="1.0"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:d="http://docbook.org/ns/docbook"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  exclude-result-prefixes="d">

  <xsl:template match="d:book">
    <xsl:call-template name="id.warning"/>
    <div>
      <xsl:apply-templates select="." mode="common.html.attributes"/>
      <xsl:if test="$generate.id.attributes != 0">
        <xsl:attribute name="id">
          <xsl:call-template name="object.id"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:call-template name="book.titlepage"/>

      <xsl:apply-templates select="d:dedication"
        mode="dedication"/>
      <xsl:apply-templates select="d:acknowledgements"
        mode="acknowledgements"/>

      <xsl:variable name="toc.params">
        <xsl:call-template name="find.path.params">
          <xsl:with-param name="table"
            select="normalize-space($generate.toc)"/>
        </xsl:call-template>
      </xsl:variable>

      <xsl:apply-templates/>

      <xsl:call-template name="make.lots">
        <xsl:with-param name="toc.params" select="$toc.params"/>
        <xsl:with-param name="toc">
          <xsl:call-template name="division.toc">
            <xsl:with-param name="toc.title.p"
              select="contains($toc.params, 'title')"/>
          </xsl:call-template>
        </xsl:with-param>
      </xsl:call-template>
    </div>
  </xsl:template>

</xsl:stylesheet>
```

1. The `xsl:apply-templates` creates first all the content
2. Create the table of content

3. Include `move-toc.xsl` into your customization layer from Step 1 [159]:

```
<xsl:include href="move-toc.xsl"/>
```

4. Build your document with your customization layer.

Discussion

The same principle applies to other structural elements, although the handling is a bit different.

5.7. Implementing Syntax Highlighting with Google Code Prettify

Problem

You want to highlight your source codes in `programlisting` elements with Google Code Prettifiy [https://github.com/google/code-prettify].

Solution

Syntax highlighting makes source code easier to read as keywords, strings, etc. appear in different colors. Use the following procedure to implement syntax highlighting with Google Code Prettifiy:

1. Download the source code bundle from the project's home page at https://github.com/google/code-prettify.

2. Copy the `src` directory to your HTML output directory and rename it to `highlighter`.

3. Create a customization layer as explained in Section 2.3, “Writing Customization Layers” [35].

4. Set the *`html.stylesheet`* stylesheet parameter:

   ```
   <xsl:param name="html.stylesheet">highlighter/prettify.css</xsl:param>
   ```

   This value is needed to add the default style into the header of the HTML output.

5. Add the named template `user.footer.content` with the following content:

   ```
   <xsl:template name="user.footer.content">
      <xsl:param name="node" select="."/>
      <script src="highlighter/prettify.js"></script>
      <script>prettyPrint();</script>
   </xsl:template>
   ```

   This template is called before the end of `</body>` tag. Inside the template, load the `prettify.js` and call the function `prettyPrint()`

6. Add a template for `programlisting` which only matches, when you add a `language` attribute. Use the `class.value` mode as follows:

   ```
   <xsl:template match="d:programlisting[@language]" mode="class.value">
     <xsl:param name="class" select="local-name(.)"/>
     <xsl:choose>
       <xsl:when test="@linenumbering = 'numbered'">
          <xsl:value-of select="concat('prettyprint linenums ', $class)"/>
       </xsl:when>
       <xsl:otherwise>
          <xsl:value-of select="concat('prettyprint ', $class)"/>
       </xsl:otherwise>
     </xsl:choose>
   </xsl:template>
   ```

7. If you want to enable syntax highlighting, make sure you add a `language` attribute in your `programlisting` tag.

   If you want to enable line numbering, add the attribute `linenumbering` with the value `linenums`.

8. Rebuild your HTML with your customization layer.

Discussion

The current implementation uses a JavaScript solution to render the result in your browser. This has the advantage that you do not need any extensions in your customization layer during transformation. You only need to add the above additions to your customization layer and your source code to enable syntax highlighting.

However, if JavaScript is disabled it won't work anymore. If you rely on minimal HTML without any JavaScript code, this solution is not for you. Use the highlighting mechanism which uses a XSLT extension. *Add link to highlighting extension topic with JAR*

Using the Language Attribute

As you could see above, the code in Step 6 [161] uses the `language` attribute, but doesn't pass it to the HTML code. This is not needed as Google Code Prettify will guess.[3] If you do not want to rely on this algorithm, change the above code and insert the language explitly:

```
<xsl:template match="d:programlisting[@language]" mode="class.value">
  <xsl:param name="class" select="local-name(.)"/>
  <xsl:variable name="lang" select="concat('lang-', @language, ' ')"/>
  <xsl:choose>
    <xsl:when test="@linenumbering = 'numbered'">
      <xsl:value-of select="concat('prettyprint ', $lang, 'linenums ', $class)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="concat('prettyprint ', $lang, $class)"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Be aware that the above code does not make any checks, it just copies the value. If your `language` attribute contains an unsupported value, it may work not as expected.

Using a Different Style

If you do not like the default style from `highlighter/prettify.css`, use one of the CSS files from the `styles` directory. You can view an example in the Gallery [https://github.com/google/code-prettify/styles/index.html].

Copy your favorite style into the `highlighter` directory and change the *`html.stylesheet`* pointing to your CSS file.

See Also

Include URL or bibliographic references.

[3] "You don't need to specify the language since prettyprint() will guess." cited from https://github.com/google/code-prettify/blob/master/README.md#how-do-i-specify-the-language-of-my-code.

5.8. Controlling the Chunking Process

Problem

You want to split your result HTML into different files, all correctly linked.

Solution

The DocBook XSL stylesheets uses the term *chunking* for splitting up your result into different HTML files. A *chunk* is therefor a single HTML file. Use the `chunk.xsl` stylesheet, it is available for all HTML variants. Usually this is enough to chunk your result.

To influence the chunking process, use the following parameters:

`base.dir`
: Sets the output directory for all chunks. If not set, the output directory is system dependent. Usually it is the current directory from where you have executed your XSLT processor.

`chunk.section.depth`
: Sets the depth to which sections should be chunked. Default is 1.

`chunk.first.sections`
: Controls, if a first top-level `sect1` or `section` element is chunked. If non-zero, a separate file (“chunk”) is created, otherwise the section is included in the component. Default is 0 (= zero, no separate chunk is created).

`use.id.as.filename`
: Controls the filename of the chunked element. If non-zero, the filename is derived from the ID of the element. If zero, the filename is generated and numbered according to its position. Default is 0 (= zero, do not use IDs for file names).

Discussion

To better understand what the stylesheet creates, lets assume the following book structure:

Example 5.4. Book Structure With Components and Sections

```
book
  preface
  chapter
    sect1
    sect1
  appendix
    sect1
    sect1
```

The following subsections show how to influence the output.

Knowing the Default Behaviour

Using the `chunk.xsl` stylesheet with **xsltproc**, **saxon**, or any other XSLT processor leads to the following file names:

Example 5.5. Result of Chunking Without Parameters (Default)

```
# No parameters set, default behaviour
book         --> index.html
  preface    --> pr01.html
  chapter    --> ch01.html
    sect1
    sect1    --> ch01s02.html
  appendix   --> apa.html
```

```
    sect1
    sect1    --> apas02.html
```

As you can see, the file name consists of several components:

- **An abbreviation of the chunked element** Each chunked element is assigned an abbreviation, one or two characters long. The available abbreviations are shown in Table 5.1 [164].

Table 5.1. Abbreviations for Chunked Elements

Abbreviation	Element
ap	appendix
ar	article
bi	bibliography
bk	book
ch	chapter
co	colophon
go	glossary
ix	index
pr	preface
pt	part
re	refentry
rn	reference
s	section
se	set
si	setindex
to	topic

- **A consecutive number** Each chunked component gets a number. For example, the first chapter has `ch01`, the second chapter `ch02`, and so on.

- **Additional sub components** If components has subcomponents like sections, the subcomponent's abbreviation is included in the file name. As such, the second section in the first chapter gets the file name `ch01s02.html`.

Writing to a Directory

If you want to have your files in a specific directory, set the parameter *`base.dir`* to your preferred value, for example:

```
# base.dir=html/
book         --> html/index.html
  preface    --> html/pr01.html
  chapter    --> html/ch01.html
    sect1
    sect1    --> html/ch01s02.html
  appendix   --> html/apa.html
    sect1
    sect1    --> html/apas02.html
```

Chunking the First Section

Example 5.5 [163] showed, that the first section is *not* chunked. This is the default behavior. However, if you want the first section also to be written in a separate file, set the *`chunk.first.sections`* parameter to `1` to get the following result:

```
book         --> index.html
```

```
  preface    --> pr01.html
  chapter    --> ch01.html
    sect1    --> ch01s01.html
    sect1    --> ch01s02.html
  appendix   --> apa.html
    sect1    --> apas01.html
    sect1    --> apas02.html
```

Influencing the Chunking Depth

If we have very deeply nested structures with sections and subsections, we may want to chunk these as well. As an example, lets assume the following chapter with these subsections:

```
chapter
  sect1
    sect2
      sect3
        sect4
    sect2
  sect1
```

By default, only `sect1` elements are written to a file. Anything below a `sect1` like `sect2`, `sect3` etc. is written to the same file that contains the `sect1` element.

To control the chunking process for sections, use the parameter *`chunk.section.depth`*. By default, the parameter is set to `1` which is equivalent to chunk only level one sections. Setting *`chunk.section.depth`* to `2` has the following effect:

```
# chunk.section.depth=2, chunk.first.sections=1
chapter          --> ch01.html
  sect1          --> ch01s01.html
    sect2        --> ch01s01s01.html
      sect3
        sect4
    sect2        --> ch01s01s02.html
  sect1          --> ch01s02.html
```

As you can see, with the value of `2`, level two sections (`sect2`) are written to a separate file. A value of `3` has the following effect:

```
# chunk.section.depth=3, chunk.first.sections=1
chapter          --> ch01.html
  sect1          --> ch01s01.html
    sect2        --> ch01s01s01.html
      sect3      --> ch01s01s01s01.html
        sect4
    sect2        --> ch01s01s02.html
  sect1          --> ch01s02.html
```

In other words, parameter *`chunk.section.depth`* cuts at the respective section level.

Create Stable File Names through IDs

The previous sections used *predictable* file names, but *not stable* ones. If you add or remove a section or chapter, the numbering of the chapters and sections will change and as such the file names too. If you want to share a link, this naming scheme is not useful as it is not stable.

Stable file names are not affected when you restructure your document. If you add or remove a structural element, the file names will still be the same.

To create such stable file names, use the parameter *`use.id.as.filename`*. This creates a file name through the `xml:id` attribute of your component. However, you should keep in mind some issues when you use this naming scheme:

- **Validate your document before you transform it** Validating your document shows you any problems with IDs. For example, double IDs, missing IDs, and syntactically wrong IDs. This is very useful as the DocBook XSL stylesheets do not check for file names which occur twice. This could lead to a situation where one file name overwrites the other.

- **Set IDs to your components** You need to set IDs to your components, otherwise it will fallback to the default naming scheme.

- **Use "speaking" IDs** Some tools can generate IDs automatically which could lead to something like `y8w739zya`. Such IDs are nonsense and useless as you cannot memorize them and they do not give any hints. Avoid that and replace such IDs with some meaningful and easy to remember name. This will also benefit your file names.

- **Avoid unusual characters in your ID** Although it may be tempting to use umlauts, diacritica, or other Unicode characters, it is recommended to stay in the realm of the ASCII character set. Depending on the file system, the tools you use, or the operating system, Unicode characters could not be fully supported and as such could lead to wrong file names.

- **Structure your IDs consistently** It is easier to find a HTML file if it is named consistently. For example, if you have a chapter about introduction, you could set the ID to `intro`. Any sections inside this chapter would use it as a prefix and append their own. A section with describes an overview could have an ID named `intro.overview`. This helps you when you search a specific HTML file.

Lets amend Example 5.4, "Book Structure With Components and Sections" [163] with IDs:

Example 5.6. Book Structure with IDs

```
book xml:id="book"
  preface xml:id="preface"
  chapter xml:id="intro"
    sect1 xml:id="intro.concept"
    sect1 xml:id="intro.requirements"
  appendix xml:id="app.overview"
    sect1 xml:id="app.overview.method-a"
    sect1 xml:id="app.overview.method-b"
```

Transforming it through `chunk.xsl` leads to the following result:

```
# use.id.as.filename=1, chunk.first.sections=1
book         --> index.html
  preface    --> preface.html
  chapter    --> intro.html
    sect1    --> intro.concept.html
    sect1    --> intro.requirements.html
  appendix   --> app.overview.html
    sect1    --> app.overview.method-a.html
    sect1    --> app.overview.method-b.html
```

Maybe the file name of the book is a bit surprising. By default, its basename is `index`. If you want to change that too, set the parameter *`root.filename`* to your preferred value (without a file extension).

Of course, you can combine it with all the other parameters which are explained in this topic.

See Also

- http://www.sagehill.net/docbookxsl/ChunkingCustomization.html

5.9. Setting your own Files and Directories Names during Chunking

Problem

You need a method to use the chunking process, but you want to define your own file and directory names for the HTML output.

Solution

The DocBook XSL stylesheets recognize the specific processing instruction (PI) `<?dbhtml>` for HTML output to influence the chunking process. The PI knows the following pseudo-attributes:

`<?dbhtml dir="PATH">`
: Specifies a directory name in which to write files. It is possible to add a trailing slash or leave it out, the stylesheets know how to deal with both cases.

`<?dbhtml filename="FILENAME">`
: Specifies a file name for a chunk. The value must contain only the file name with an optional extension, but not any directories.

You can combine `filename` and `dir`. Insert this processing instruction into your component of your XML file.

Discussion

The `<?dbhtml>` processing instruction should be inserted after the start tag of the component. Consider the following structure with such PIs:

```
<book version="5.0" xml:id="book"
    xmlns="http://docbook.org/ns/docbook" xml:lang="en">
    <?dbhtml dir="book"?>
  <title>Chunking Test</title>
  <preface xml:id="preface">
    <?dbhtml dir="pre/"?>
    <!-- ... -->
  </preface>
  <chapter xml:id="intro">
    <?dbhtml dir="intro" filename="index.html"?>
    <!-- ... -->
  </chapter>
  <appendix xml:id="app.overview">
    <?dbhtml dir="app" filename="index.html"?>
    <!-- ... -->
  </appendix>
</book>
```

When you transform it to HTML with the `chunk.xsl` stylesheet, you will get the following directory structure:

```
book/
### app/
#   ### ...
#   ### index.html
### index.html
### intro/
#   ### ...
#   ### index.html
### pre/
    ### ...
    ### pr01.html
```

You can combine other chunking parameters as described in Section 5.8, “Controlling the Chunking Process” [163].

See Also

- http://www.sagehill.net/docbookxsl/Chunking.html
- http://docbook.sf.net/release/xsl/current/doc/pi/dbhtml_dir.html
- http://docbook.sf.net/release/xsl/current/doc/pi/dbhtml_filename.html

Anhang A. Namensnennung – Keine kommerzielle Nutzung – Weitergabe unter gleichen Bedingungen 3.0 Deutschland

Entnommen von http://creativecommons.org/licenses/by-nc-sa/3.0/de/.

> CREATIVE COMMONS IST KEINE RECHTSANWALTSKANZLEI UND LEISTET KEINE RECHTSBERATUNG. DIE BEREITSTELLUNG DIESER LIZENZ FÜHRT ZU KEINEM MANDATSVERHÄLTNIS. CREATIVE COMMONS STELLT DIESE INFORMATIONEN OHNE GEWÄHR ZUR VERFÜGUNG. CREATIVE COMMONS ÜBERNIMMT KEINE GEWÄHRLEISTUNG FÜR DIE GELIEFERTEN INFORMATIONEN UND SCHLIEßT DIE HAFTUNG FÜR SCHÄDEN AUS, DIE SICH AUS DEREN GEBRAUCH ERGEBEN.
>
> **Lizenz**
>
> DER GEGENSTAND DIESER LIZENZ (WIE UNTER „SCHUTZGEGENSTAND“ DEFINIERT) WIRD UNTER DEN BEDINGUNGEN DIESER CREATIVE COMMONS PUBLIC LICENSE („CCPL“, „LIZENZ“ ODER „LIZENZVERTRAG“) ZUR VERFÜGUNG GESTELLT. DER SCHUTZGEGENSTAND IST DURCH DAS URHEBERRECHT UND/ODER ANDERE GESETZE GESCHÜTZT. JEDE FORM DER NUTZUNG DES SCHUTZGEGENSTANDES, DIE NICHT AUFGRUND DIESER LIZENZ ODER DURCH GESETZE GESTATTET IST, IST UNZULÄSSIG.
>
> DURCH DIE AUSÜBUNG EINES DURCH DIESE LIZENZ GEWÄHRTEN RECHTS AN DEM SCHUTZGEGENSTAND ERKLÄREN SIE SICH MIT DEN LIZENZBEDINGUNGEN RECHTSVERBINDLICH EINVERSTANDEN. SOWEIT DIESE LIZENZ ALS LIZENZVERTRAG ANZUSEHEN IST, GEWÄHRT IHNEN DER LIZENZGEBER DIE IN DER LIZENZ GENANNTEN RECHTE UNENTGELTLICH UND IM AUSTAUSCH DAFÜR, DASS SIE DAS GEBUNDENSEIN AN DIE LIZENZBEDINGUNGEN AKZEPTIEREN.

1. Definitionen

1.a Der Begriff „**Abwandlung**“ im Sinne dieser Lizenz bezeichnet das Ergebnis jeglicher Art von Veränderung des Schutzgegenstandes, solange die eigenpersönlichen Züge des Schutzgegenstandes darin nicht verblassen und daran eigene Schutzrechte entstehen. Das kann insbesondere eine Bearbeitung, Umgestaltung, Änderung, Anpassung, Übersetzung oder Heranziehung des Schutzgegenstandes zur Vertonung von Laufbildern sein. Nicht als Abwandlung des Schutzgegenstandes gelten seine Aufnahme in eine Sammlung oder ein Sammelwerk und die freie Benutzung des Schutzgegenstandes.

1.b Der Begriff „**Sammelwerk**“ im Sinne dieser Lizenz meint eine Zusammenstellung von literarischen, künstlerischen oder wissenschaftlichen Inhalten, sofern diese Zusammenstellung aufgrund von Auswahl und Anordnung der darin enthaltenen selbständigen Elemente eine geistige Schöpfung darstellt, unabhängig davon, ob die Elemente systematisch oder methodisch angelegt und dadurch einzeln zugänglich sind oder nicht.

1.c „**Verbreiten**“ im Sinne dieser Lizenz bedeutet, den Schutzgegenstand oder Abwandlungen im Original oder in Form von Vervielfältigungsstücken, mithin in körperlich fixierter Form der Öffentlichkeit anzubieten oder in Verkehr zu bringen.

1.d Unter „**Lizenzelementen**“ werden im Sinne dieser Lizenz die folgenden übergeordneten Lizenzcharakteristika verstanden, die vom Lizenzgeber ausgewählt wurden und in der Bezeichnung der Lizenz zum Ausdruck kommen: „Namensnennung“, „Keine kommerzielle Nutzung“, „Weitergabe unter gleichen Bedingungen“.

1.e Der „**Lizenzgeber**“ im Sinne dieser Lizenz ist diejenige natürliche oder juristische Person oder Gruppe, die den Schutzgegenstand unter den Bedingungen dieser Lizenz anbietet und insoweit als Rechteinhaberin auftritt.

1.f„**Rechteinhaber**“ im Sinne dieser Lizenz ist der Urheber des Schutzgegenstandes oder jede andere natürliche oder juristische Person oder Gruppe von Personen, die am Schutzgegenstand ein Immaterialgüterrecht erlangt hat, welches die in Abschnitt 3 [170] genannten Handlungen erfasst und bei dem eine Einräumung von Nutzungsrechten oder eine Weiterübertragung an Dritte möglich ist.

1.gDer Begriff „**Schutzgegenstand**“ bezeichnet in dieser Lizenz den literarischen, künstlerischen oder wissenschaftlichen Inhalt, der unter den Bedingungen dieser Lizenz angeboten wird. Das kann insbesondere eine persönliche geistige Schöpfung jeglicher Art, ein Werk der kleinen Münze, ein nachgelassenes Werk oder auch ein Lichtbild oder anderes Objekt eines verwandten Schutzrechts sein, unabhängig von der Art seiner Fixierung und unabhängig davon, auf welche Weise jeweils eine Wahrnehmung erfolgen kann, gleichviel ob in analoger oder digitaler Form. Soweit Datenbanken oder Zusammenstellungen von Daten einen immaterialgüterrechtlichen Schutz eigener Art genießen, unterfallen auch sie dem Begriff "Schutzgegenstand" im Sinne dieser Lizenz.

1.hMit „**Sie**“ bzw. „**Ihnen**“ ist die natürliche oder juristische Person gemeint, die in dieser Lizenz im Abschnitt 3 [170] genannte Nutzungen des Schutzgegenstandes vornimmt und zuvor in Hinblick auf den Schutzgegenstand nicht gegen Bedingungen dieser Lizenz verstoßen oder aber die ausdrückliche Erlaubnis des Lizenzgebers erhalten hat, die durch diese Lizenz gewährten Nutzungsrechte trotz eines vorherigen Verstoßes auszuüben.

1.i.Unter „**Öffentlich Zeigen**“ im Sinne dieser Lizenz sind Veröffentlichungen und Präsentationen des Schutzgegenstandes zu verstehen, die für eine Mehrzahl von Mitgliedern der Öffentlichkeit bestimmt sind und in unkörperlicher Form mittels öffentlicher Wiedergabe in Form von Vortrag, Aufführung, Vorführung, Darbietung, Sendung, Weitersendung, zeit- und ortsunabhängiger Zugänglichmachung oder in körperlicher Form mittels Ausstellung erfolgen, unabhängig von bestimmten Veranstaltungen und unabhängig von den zum Einsatz kommenden Techniken und Verfahren, einschließlich drahtgebundener oder drahtloser Mittel und Einstellen in das Internet.

1.j.„**Vervielfältigen**“ im Sinne dieser Lizenz bedeutet, mittels beliebiger Verfahren Vervielfältigungsstücke des Schutzgegenstandes herzustellen, insbesondere durch Ton- oder Bildaufzeichnungen, und umfasst auch den Vorgang, erstmals körperliche Fixierungen des Schutzgegenstandes sowie Vervielfältigungsstücke dieser Fixierungen anzufertigen, sowie die Übertragung des Schutzgegenstandes auf einen Bild- oder Tonträger oder auf ein anderes elektronisches Medium, gleichviel ob in digitaler oder analoger Form.

2. **Schranken des Immaterialgüterrechts** Diese Lizenz ist in keiner Weise darauf gerichtet, Befugnisse zur Nutzung des Schutzgegenstandes zu vermindern, zu beschränken oder zu vereiteln, die Ihnen aufgrund der Schranken des Urheberrechts oder anderer Rechtsnormen bereits ohne Weiteres zustehen oder sich aus dem Fehlen eines immaterialgüterrechtlichen Schutzes ergeben.

3. **Einräumung von Nutzungsrechten** Unter den Bedingungen dieser Lizenz räumt Ihnen der Lizenzgeber – unbeschadet unverzichtbarer Rechte und vorbehaltlich des Abschnitts f [172] – das vergütungsfreie, räumlich und zeitlich (für die Dauer des Schutzrechts am Schutzgegenstand) unbeschränkte einfache Recht ein, den Schutzgegenstand auf die folgenden Arten und Weisen zu nutzen („unentgeltlich eingeräumtes einfaches Nutzungsrecht für jedermann“):

   a. Den Schutzgegenstand in beliebiger Form und Menge zu vervielfältigen, ihn in Sammelwerke zu integrieren und ihn als Teil solcher Sammelwerke zu vervielfältigen;

   b. Abwandlungen des Schutzgegenstandes anzufertigen, einschließlich Übersetzungen unter Nutzung jedweder Medien, sofern deutlich erkennbar gemacht wird, dass es sich um Abwandlungen handelt;

   c. den Schutzgegenstand, allein oder in Sammelwerke aufgenommen, öffentlich zu zeigen und zu verbreiten;

   d. Abwandlungen des Schutzgegenstandes zu veröffentlichen, öffentlich zu zeigen und zu verbreiten.

   Das vorgenannte Nutzungsrecht wird für alle bekannten sowie für alle noch nicht bekannten Nutzungsarten eingeräumt. Es beinhaltet auch das Recht, solche Änderungen am Schutzgegenstand vorzunehmen, die für bestimmte nach dieser Lizenz zulässige Nutzungen technisch erforderlich sind. Alle sonstigen Rechte, die über diesen Abschnitt hinaus nicht ausdrücklich durch den Lizenzgeber eingeräumt werden, bleiben diesem allein vorbehalten. Soweit Datenbanken oder Zusammenstellungen von Daten Schutzgegenstand dieser Lizenz oder Teil dessen sind und einen immaterialgüterrechtlichen Schutz eigener Art genießen, verzichtet der Lizenzgeber auf sämtliche aus diesem Schutz resultierenden Rechte.

4. **Bedingungen** Die Einräumung des Nutzungsrechts gemäß Abschnitt 3 [170] dieser Lizenz erfolgt ausdrücklich nur unter den folgenden Bedingungen:

4.a Sie dürfen den Schutzgegenstand ausschließlich unter den Bedingungen dieser Lizenz verbreiten oder öffentlich zeigen. Sie müssen dabei stets eine Kopie dieser Lizenz oder deren vollständige Internetadresse in Form des Uniform-Resource-Identifier (URI) beifügen. Sie dürfen keine Vertrags- oder Nutzungsbedingungen anbieten oder fordern, die die Bedingungen dieser Lizenz oder die durch diese Lizenz gewährten Rechte beschränken. Sie dürfen den Schutzgegenstand nicht unterlizenzieren. Bei jeder Kopie des Schutzgegenstandes, die Sie verbreiten oder öffentlich zeigen, müssen Sie alle Hinweise unverändert lassen, die auf diese Lizenz und den Haftungsausschluss hinweisen. Wenn Sie den Schutzgegenstand verbreiten oder öffentlich zeigen, dürfen Sie (in Bezug auf den Schutzgegenstand) keine technischen Maßnahmen ergreifen, die den Nutzer des Schutzgegenstandes in der Ausübung der ihm durch diese Lizenz gewährten Rechte behindern können. Dieser Abschnitt a [171] gilt auch für den Fall, dass der Schutzgegenstand einen Bestandteil eines Sammelwerkes bildet, was jedoch nicht bedeutet, dass das Sammelwerk insgesamt dieser Lizenz unterstellt werden muss. Sofern Sie ein Sammelwerk erstellen, müssen Sie auf die Mitteilung eines Lizenzgebers hin aus dem Sammelwerk die in Abschnitt d [171] aufgezählten Hinweise entfernen. Wenn Sie eine Abwandlung vornehmen, müssen Sie auf die Mitteilung eines Lizenzgebers hin von der Abwandlung die in Abschnitt d [171] aufgezählten Hinweise entfernen.

4.b Sie dürfen eine Abwandlung ausschließlich unter den Bedingungen:

i. dieser Lizenz,

ii. einer späteren Version dieser Lizenz mit denselben Lizenzelementen;

iii. einer rechtsordnungsspezifischen Creative-Commons-Lizenz mit denselben Lizenzelementen ab Version 3.0 aufwärts (z.B. Namensnennung – Keine kommerzielle Nutzung – Weitergabe unter gleichen Bedingungen 3.0 US) oder

iv. der Creative-Commons-Unported-Lizenz mit denselben Lizenzelementen ab Version 3.0 aufwärts

verbreiten oder öffentlich zeigen („Verwendbare Lizenz“).

Sie müssen stets eine Kopie der verwendbaren Lizenz oder deren vollständige Internetadresse in Form des Uniform-Resource-Identifier (URI) beifügen, wenn Sie die Abwandlung verbreiten oder öffentlich zeigen. Sie dürfen keine Vertrags- oder Nutzungsbedingungen anbieten oder fordern, die die Bedingungen der verwendbaren Lizenz oder die durch sie gewährten Rechte beschränken. Bei jeder Abwandlung, die Sie verbreiten oder öffentlich zeigen, müssen Sie alle Hinweise auf die verwendbare Lizenz und den Haftungsausschluss unverändert lassen. Wenn Sie die Abwandlung verbreiten oder öffentlich zeigen, dürfen Sie (in Bezug auf die Abwandlung) keine technischen Maßnahmen ergreifen, die den Nutzer der Abwandlung in der Ausübung der ihm durch die verwendbare Lizenz gewährten Rechte behindern können. Dieser Abschnitt b [171] gilt auch für den Fall, dass die Abwandlung einen Bestandteil eines Sammelwerkes bildet, was jedoch nicht bedeutet, dass das Sammelwerk insgesamt der verwendbaren Lizenz unterstellt werden muss.

4.c Die Rechteeinräumung gemäß Abschnitt 3 [176] gilt nur für Handlungen, die nicht vorrangig auf einen geschäftlichen Vorteil oder eine geldwerte Vergütung gerichtet sind („nicht-kommerzielle Nutzung“, „Non-commercial-Option“). Wird Ihnen in Zusammenhang mit dem Schutzgegenstand dieser Lizenz ein anderer Schutzgegenstand überlassen, ohne dass eine vertragliche Verpflichtung hierzu besteht (etwa im Wege von File-Sharing), so wird dies nicht als auf geschäftlichen Vorteil oder geldwerte Vergütung gerichtet angesehen, wenn in Verbindung mit dem Austausch der Schutzgegenstände tatsächlich keine Zahlung oder geldwerte Vergütung geleistet wird.

4.d Die Verbreitung und das öffentliche Zeigen des Schutzgegenstandes oder auf ihm aufbauender Abwandlungen oder ihn enthaltender Sammelwerke ist Ihnen nur unter der Bedingung gestattet, dass Sie, vorbehaltlich etwaiger Mitteilungen im Sinne von Abschnitt a [171], alle dazu gehörenden Rechtevermerke unberührt lassen. Sie sind verpflichtet, die Rechteinhaberschaft in einer der Nutzung entsprechenden, angemessenen Form anzuerkennen, indem Sie – soweit bekannt – Folgendes angeben:

i. Den Namen (oder das Pseudonym, falls ein solches verwendet wird) des Rechteinhabers und / oder, falls der Lizenzgeber im Rechtevermerk, in den Nutzungsbedingungen oder auf andere angemessene Weise eine Zuschreibung an Dritte vorgenommen hat (z.B. an eine Stiftung, ein Verlagshaus oder eine Zeitung) („Zuschreibungsempfänger“), Namen bzw. Bezeichnung dieses oder dieser Dritten;

ii. den Titel des Inhaltes;

iii. in einer praktikablen Form den Uniform-Resource-Identifier (URI, z.B. Internetadresse), den der Lizenzgeber zum Schutzgegenstand angegeben hat, es sei denn, dieser URI verweist nicht auf den Rechtevermerk oder die Lizenzinformationen zum Schutzgegenstand;

iv. und im Falle einer Abwandlung des Schutzgegenstandes in Übereinstimmung mit Abschnitt b [170] einen Hinweis darauf, dass es sich um eine Abwandlung handelt.

Die nach diesem Abschnitt d [171] erforderlichen Angaben können in jeder angemessenen Form gemacht werden; im Falle einer Abwandlung des Schutzgegenstandes oder eines Sammelwerkes müssen diese Angaben das Minimum darstellen und bei gemeinsamer Nennung mehrerer Rechteinhaber dergestalt erfolgen, dass sie zumindest ebenso hervorgehoben sind wie die Hinweise auf die übrigen Rechteinhaber. Die Angaben nach diesem Abschnitt dürfen Sie ausschließlich zur Angabe der Rechteinhaberschaft in der oben bezeichneten Weise verwenden. Durch die Ausübung Ihrer Rechte aus dieser Lizenz dürfen Sie ohne eine vorherige, separat und schriftlich vorliegende Zustimmung des Lizenzgebers und / oder des Zuschreibungsempfängers weder explizit noch implizit irgendeine Verbindung zum Lizenzgeber oder Zuschreibungsempfänger und ebenso wenig eine Unterstützung oder Billigung durch ihn andeuten.

4.e Die oben unter a [171] bis d [171] genannten Einschränkungen gelten nicht für solche Teile des Schutzgegenstandes, die allein deshalb unter den Schutzgegenstandsbegriff fallen, weil sie als Datenbanken oder Zusammenstellungen von Daten einen immaterialgüterrechtlichen Schutz eigener Art genießen.

4.f. Bezüglich Vergütung für die Nutzung des Schutzgegenstandes gilt Folgendes:

i. **Unverzichtbare gesetzliche Vergütungsansprüche:** Soweit unverzichtbare Vergütungsansprüche im Gegenzug für gesetzliche Lizenzen vorgesehen oder Pauschalabgabensysteme (zum Beispiel für Leermedien) vorhanden sind, behält sich der Lizenzgeber das ausschließliche Recht vor, die entsprechende Vergütung einzuziehen für jede Ausübung eines Rechts aus dieser Lizenz durch Sie.

ii. **Vergütung bei Zwangslizenzen:** Sofern Zwangslizenzen außerhalb dieser Lizenz vorgesehen sind und zustande kommen, behält sich der Lizenzgeber das ausschließliche Recht auf Einziehung der entsprechenden Vergütung für den Fall vor, dass Sie eine Nutzung des Schutzgegenstandes für andere als die in Abschnitt c [171] als nicht-kommerziell definierten Zwecke vornehmen, verzichtet für alle übrigen, lizenzgerechten Fälle von Nutzung jedoch auf jegliche Vergütung.

iii. **Vergütung in sonstigen Fällen:** Bezüglich lizenzgerechter Nutzung des Schutzgegenstandes durch Sie, die nicht unter die beiden vorherigen Abschnitte i [172] und ii [172] fällt, verzichtet der Lizenzgeber auf jegliche Vergütung, unabhängig davon, ob eine Einziehung der Vergütung durch ihn selbst oder nur durch eine Verwertungsgesellschaft möglich wäre. Der Lizenzgeber behält sich jedoch das ausschließliche Recht auf Einziehung der entsprechenden Vergütung (durch ihn selbst oder eine Verwertungsgesellschaft) für den Fall vor, dass Sie eine Nutzung des Schutzgegenstandes für andere als die in Abschnitt c [171] als nicht-kommerziell definierten Zwecke vornehmen.

4.g Persönlichkeitsrechte bleiben – soweit sie bestehen – von dieser Lizenz unberührt.

5. **Gewährleistung** SOFERN KEINE ANDERS LAUTENDE, SCHRIFTLICHE VEREINBARUNG ZWISCHEN DEM LIZENZGEBER UND IHNEN GESCHLOSSEN WURDE UND SOWEIT MÄNGEL NICHT ARGLISTIG VERSCHWIEGEN WURDEN, BIETET DER LIZENZGEBER DEN SCHUTZGEGENSTAND UND DIE EINRÄUMUNG VON RECHTEN UNTER AUSSCHLUSS JEGLICHER GEWÄHRLEISTUNG AN UND ÜBERNIMMT WEDER AUSDRÜCKLICH NOCH KONKLUDENT GARANTIEN IRGENDEINER ART. DIES UMFASST INSBESONDERE DAS FREISEIN VON SACH- UND RECHTSMÄNGELN, UNABHÄNGIG VON DEREN ERKENNBARKEIT FÜR DEN LIZENZGEBER, DIE VERKEHRSFÄHIGKEIT DES SCHUTZGEGENSTANDES, SEINE VERWENDBARKEIT FÜR EINEN BESTIMMTEN ZWECK SOWIE DIE KORREKTHEIT VON BESCHREIBUNGEN. DIESE GEWÄHRLEISTUNGSBESCHRÄNKUNG GILT NICHT, SOWEIT MÄNGEL ZU SCHÄDEN DER IN ABSCHNITT 6 [172] BEZEICHNETEN ART FÜHREN UND AUF SEITEN DES LIZENZGEBERS DAS JEWEILS GENANNTE VERSCHULDEN BZW. VERTRETENMÜSSEN EBENFALLS VORLIEGT.

6. **Haftungsbeschränkung** DER LIZENZGEBER HAFTET IHNEN GEGENÜBER IN BEZUG AUF SCHÄDEN AUS DER VERLETZUNG DES LEBENS, DES KÖRPERS ODER DER GESUNDHEIT NUR, SOFERN IHM WENIGSTENS FAHRLÄSSIGKEIT VORZUWERFEN IST, FÜR SONSTIGE SCHÄDEN NUR BEI GROBER FAHRLÄSSIGKEIT ODER VORSATZ, UND ÜBERNIMMT DARÜBER HINAUS KEINERLEI FREIWILLIGE HAFTUNG.

7. **Erlöschen**

7.a Diese Lizenz und die durch sie eingeräumten Nutzungsrechte erlöschen mit Wirkung für die Zukunft im Falle eines Verstoßes gegen die Lizenzbedingungen durch Sie, ohne dass es dazu der Kenntnis des Lizenzgebers vom Verstoß oder einer weiteren Handlung einer der Vertragsparteien bedarf. Mit natürlichen oder juristischen Personen, die Abwandlungen des Schutzgegenstandes oder diesen enthaltende Sammelwerke unter den Bedingungen dieser Lizenz von Ihnen erhalten haben, bestehen nachträglich entstandene Lizenzbeziehungen jedoch solange weiter, wie die genannten Personen sich ihrerseits an sämtliche Lizenzbedingungen halten. Darüber hinaus gelten die Ziffern 1 [169], 2 [170], 5 [172], 6 [172], 7 [173] und 8 [173] auch nach einem Erlöschen dieser Lizenz fort.

7.b Vorbehaltlich der oben genannten Bedingungen gilt diese Lizenz unbefristet bis der rechtliche Schutz für den Schutzgegenstand ausläuft. Davon abgesehen behält der Lizenzgeber das Recht, den Schutzgegenstand unter anderen Lizenzbedingungen anzubieten oder die eigene Weitergabe des Schutzgegenstandes jederzeit einzustellen, solange die Ausübung dieses Rechts nicht einer Kündigung oder einem Widerruf dieser Lizenz (oder irgendeiner Weiterlizenzierung, die auf Grundlage dieser Lizenz bereits erfolgt ist bzw. zukünftig noch erfolgen muss) dient und diese Lizenz unter Berücksichtigung der oben zum Erlöschen genannten Bedingungen vollumfänglich wirksam bleibt.

8. **Sonstige Bestimmungen**

8.a Jedes Mal wenn Sie den Schutzgegenstand für sich genommen oder als Teil eines Sammelwerkes verbreiten oder öffentlich zeigen, bietet der Lizenzgeber dem Empfänger eine Lizenz zu den gleichen Bedingungen und im gleichen Umfang an, wie Ihnen in Form dieser Lizenz.

8.b Jedes Mal wenn Sie eine Abwandlung des Schutzgegenstandes verbreiten oder öffentlich zeigen, bietet der Lizenzgeber dem Empfänger eine Lizenz am ursprünglichen Schutzgegenstand zu den gleichen Bedingungen und im gleichen Umfang an, wie Ihnen in Form dieser Lizenz.

8.c Sollte eine Bestimmung dieser Lizenz unwirksam sein, so bleibt davon die Wirksamkeit der Lizenz im Übrigen unberührt.

8.d Keine Bestimmung dieser Lizenz soll als abbedungen und kein Verstoß gegen sie als zulässig gelten, solange die von dem Verzicht oder von dem Verstoß betroffene Seite nicht schriftlich zugestimmt hat.

8.e Diese Lizenz (zusammen mit in ihr ausdrücklich vorgesehenen Erlaubnissen, Mitteilungen und Zustimmungen, soweit diese tatsächlich vorliegen) stellt die vollständige Vereinbarung zwischen dem Lizenzgeber und Ihnen in Bezug auf den Schutzgegenstand dar. Es bestehen keine Abreden, Vereinbarungen oder Erklärungen in Bezug auf den Schutzgegenstand, die in dieser Lizenz nicht genannt sind. Rechtsgeschäftliche Änderungen des Verhältnisses zwischen dem Lizenzgeber und Ihnen sind nur über Modifikationen dieser Lizenz möglich. Der Lizenzgeber ist an etwaige zusätzliche, einseitig durch Sie übermittelte Bestimmungen nicht gebunden. Diese Lizenz kann nur durch schriftliche Vereinbarung zwischen Ihnen und dem Lizenzgeber modifiziert werden. Derlei Modifikationen wirken ausschließlich zwischen dem Lizenzgeber und Ihnen und wirken sich nicht auf die Dritten gemäß Ziffern a [173] und b [173] angeboteten Lizenzen aus.

8.f Sofern zwischen Ihnen und dem Lizenzgeber keine anderweitige Vereinbarung getroffen wurde und soweit Wahlfreiheit besteht, findet auf diesen Lizenzvertrag das Recht der Bundesrepublik Deutschland Anwendung.

Appendix B. Attribution-NonCommercial-ShareAlike 3.0 Unported

Taken from http://creativecommons.org/licenses/by-nc-sa/3.0/de/deed.en.

> CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.
>
> ### License
>
> THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.
>
> BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

1.a **"Adaptation"** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

1.b **"Collection"** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section g [176] below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

1.c **"Distribute"** means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

1.d **"License Elements"** means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, Noncommercial, ShareAlike.

1.e **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

1.f **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram

the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

1.g. **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

1.h. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

1.i. **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

1.j. **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. **Fair Dealing Rights.** Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. **License Grant.** Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

   b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

   c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

   d. to Distribute and Publicly Perform Adaptations.

   The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section f [179], all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights described in Section e [177].

4. **Restrictions.** The license granted in Section 3 [176] above is expressly made subject to and limited by the following restrictions:

   4.a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer

of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section a [176] applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section d [177], as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section d [177], as requested.

4.b You may Distribute or Publicly Perform an Adaptation only under: (i) the terms of this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-NonCommercial-ShareAlike 3.0 US) ("Applicable License"). You must include a copy of, or the URI, for Applicable License with every copy of each Adaptation You Distribute or Publicly Perform. You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License. You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section b [177] applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

4.c You may not exercise any of the rights granted to You in Section 3 [176] above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in con-nection with the exchange of copyrighted works.

4.d If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section a [176], keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and, (iv) consistent with Section b [176], in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section d [177] may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

4.e For the avoidance of doubt:

4.e.N **Non-waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

4.e.W **Waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section c [177] and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

4.e. **Voluntary License Schemes.** The Licensor reserves the right to collect royalties, whether individually or,
   ii. in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section c [177].

4.f. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section b [176] of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section b [176] of this License (right to make Adaptations) but not otherwise.

5. **Representations, Warranties and Disclaimer** UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING AND TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THIS EXCLUSION MAY NOT APPLY TO YOU.

6. **Limitation on Liability.** EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. **Termination**

7.a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1 [175], 2 [176], 5 [178], 6 [178], 7 [178], and 8 [178] will survive any termination of this License.

7.b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. **Miscellaneous**

8.a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

8.b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

8.c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

8.d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

8.e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall

not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

8.f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark “Creative Commons” or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at http://creativecommons.org.

Colophon

toms, 2012-05-17: FIXME