DocBook Assemblies and Topics for the Impatient

Hussein Shafie, XMLmind Software

December 08, 2017

Table of Contents

1. Why create DocBook assemblies and topics? 1
    1.1. Why prefer DocBook over DITA? 1
2. What is a `topic`? 2
3. What is an `assembly`? 2
    3.1. Assembly resources 4
4. Specifying the name of a realized element 4
5. Adding metadata to realized elements 6
    5.1. Replacing all the titles of the pulled topic 8
6. Aggregating contents pulled from topics 9
7. Filtering contents “pulled” from topics 10
8. Different realized documents depending on selected output format 12
9. Transforming contents “pulled” from topics 14
10. Creating navigation links between related topics 15
11. Processing DocBook assemblies 17

1. Why create DocBook assemblies and topics?

DocBook 5.1 introduces two new elements: `assembly` and `topic`. These elements allows to author large, modular, documents. Such documents are generally created and maintained over years by a team of technical writers rather than by a single person. Such documents generally share a large number of topics. That is, in such documents, most topics may be seen as *reusable parts*.

For example, let's suppose a automobile manufacturer created two assemblies. The first one is a repair manual for automobile model A comprising 753 topics and the second one is a repair manual for automobile model B comprising 825 topics. These two repair manuals are expected to have a large number of topics in common, for example a topic explaining how to replace the battery of the car.

1.1. Why prefer DocBook over DITA?

DocBook 5.1 introduces two new elements: `assembly` and `topic`. An DocBook `assembly` serves the same purpose as a DITA `map` and a DocBook `topic` serves the same purpose as a DITA `topic`. DITA 1.3 is a proven technology which exists since several years. So why bother learning DocBook 5.1 assemblies and topics?

In a nutshell, the answer is: simplicity. In our opinion, DITA 1.3 complexity costs do not allow small or even mid-size companies to use this documentation technology. This is not the case of DocBook 5.1.

- DocBook assemblies are quick and simple to learn. Suffice to read this tutorial to learn *everything* about assemblies.
- DocBook already has all the elements you'll ever need when writing technical documentation (e.g. `videoobject`, `audioobject`, `calloutlist`), while DITA 1.3 often requires you to create specialized vocabularies.
- DocBook comes with decent, flexible, well-documented, XSL stylesheets.

2. What is a `topic`?

A `topic` element is a semantically neutral container having a content model similar the one of the `chapter` element.

A `topic` element is always stored in its own file.

In terms of content, a `topic` element is expected to deal with a single topic and to be very loosely coupled, if not at all, to other `topic` elements. For example, avoid creating cross-references (`xref`, `link`) between topic A and topic B.

In this tutorial, we'll use `assembly` elements which reference `topic` elements exclusively. In fact, any DocBook container (`chapter`, `section`, `appendix`, etc) may act as a *topic* —some contents dealing about a single topic stored in its one file— and as such, be referenced in assemblies.

3. What is an `assembly`?

A DocBook 5.1 `assembly` specifies how to create a plain, “normal”, DocBook document (e.g. a `book`) out of contents “pulled” from topic files.

The “normal” DocBook document created by the means of an assembly is called *the realized document*.

The main elements of an assembly are `structure` and `module`:

- A `structure` element specifies the contents of the realized document. It contains a number of possibly nested `module` elements.
- A `module` element specifies which contents is to be “pulled” from the topic file and how this contents is then copied into the realized document.

Example 1. A structure containing a sequence of modules

File `assembly1.xml`:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>
    <resource href="topic1.xml" xml:id="topic1"/>
    <resource href="topic2.xml" xml:id="topic2"/>
    <resource href="topic3.xml" xml:id="topic3"/>
  </resources>

  <structure>
    <module resourceref="topic1"/>
    <module resourceref="topic2"/>
    <module resourceref="topic3"/>
  </structure>
</assembly>
```

`Topic1`, `topic2`, `topic3` are topics found in files all similar to `topic1.xml`:

```
<topic version="5.1" xml:id="t1" xmlns="http://docbook.org/ns/docbook">
  <info>
    <title>Title of topic #1</title>
  </info>

  <para>Paragraph <phrase userlevel="advanced">(that is, element <tag
  class="element">para</tag>)</phrase> contained in topic #1.</para>
</topic>
```

The corresponding realized document is found in `realized1.xml`:

```
<UNKNOWN xmlns="http://docbook.org/ns/docbook" version="5.1">
  <topic version="5.1" xml:base="file:/home/john/samples/topic1.xml"
         xml:id="t1">
    <info>
      <title>Title of topic #1</title>
    </info>
```

```
    <para>Paragraph <phrase userlevel="advanced">(that is,
    element <tag class="element">para</tag>)</phrase> contained
    in topic #1.</para>
  </topic>

  <topic version="5.1" xml:base="file:/home/john/samples/topic2.xml"
         xml:id="t2">
    <info>
      <title>Title of topic #2</title>
    </info>

    <para>Paragraph <phrase userlevel="advanced">(that is,
    element <tag class="element">para</tag>)</phrase> contained
    in topic #2.</para>
  </topic>

  <topic version="5.1" xml:base="file:/home/john/samples/topic3.xml"
         xml:id="t3">
    <info>
      <title>Title of topic #3</title>
    </info>

    <para>Paragraph <phrase userlevel="advanced">(that is,
    element <tag class="element">para</tag>)</phrase> contained
    in topic #3.</para>
  </topic>
</UNKNOWN>
```

A `structure` may be seen as a top-level `module`. Just like a `module`, a `structure` may be used to pull contents from a topic file and copy it into the realized document.

Example 2. A structure containing nested modules

File `assembly2.xml`:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>
    <resource href="topic1.xml" xml:id="topic1"/>
    <resource href="topic2.xml" xml:id="topic2"/>
    <resource href="topic3.xml" xml:id="topic3"/>
  </resources>

  <structure resourceref="topic1">
    <module resourceref="topic2">
      <module resourceref="topic3"/>
    </module>
  </structure>
</assembly>
```

The corresponding realized document is found in `realized2.xml`:

```
<topic xmlns="http://docbook.org/ns/docbook" version="5.1"
       xml:base="file:/home/john/samples/topic1.xml" xml:id="t1">
  <info>
    <title>Title of topic #1</title>
  </info>

  <para>Paragraph <phrase userlevel="advanced">(that is,
  element <tag class="element">para</tag>)</phrase> contained
  in topic #1.</para>

  <topic version="5.1" xml:base="file:/home/john/samples/topic2.xml"
         xml:id="t2">
    <info>
      <title>Title of topic #2</title>
    </info>

    <para>Paragraph <phrase userlevel="advanced">(that is, element
```

```
    <tag class="element">para</tag>)</phrase> contained in topic #2.</para>

    <topic version="5.1" xml:base="file:/home/john/samples/topic3.xml"
           xml:id="t3">
       <info>
         <title>Title of topic #3</title>
       </info>

       <para>Paragraph <phrase userlevel="advanced">(that is,
       element <tag class="element">para</tag>)</phrase> contained
       in topic #3.</para>
     </topic>
  </topic>
</topic>
```

3.1. Assembly resources

Notice in the above examples that a `module` or `structure` never directly references a topic file. That is, what follows is invalid:

```
<module href="topic1.xml"/>
```

Instead, a `module` or `structure` may have a `resourceref` attribute containing the ID of a `resource` declared in a `resources` section of the `assembly`:

```
  <resources>
    <resource href="topic1.xml" xml:id="topic1"/>
    ...
  </resources>

...

<module resourceref="topic1"/>
```

This approach has a number of advantages, one of them being the ability to include predefined libraries of resources in assemblies. Example:

```
  <xi:include href="my_resources.xml" xpointer="group1"/>

...

<module resourceref="topic1"/>
```

where `my_resources.xml` contains:

```
<assembly version="5.1"
          xmlns="http://docbook.org/ns/docbook">
  ...
  <resources xml:id="group1">
    <resource href="topic1.xml" xml:id="topic1"/>
    <resource href="topic2.xml" xml:id="topic2"/>
    <resource href="topic3.xml" xml:id="topic3"/>
  </resources>
  ...
</assembly>
```

4. Specifying the name of a realized element

First realized document, `realized1.xml`, is invalid as its root element is `UNKNOWN`. Second realized document, `realized2.xml`, is also invalid as DocBook 5.1 topics may not nest.

In fact, these problems happened because the two corresponding assemblies, `assembly1.xml` and `assembly2.xml`, do not make use of attribute `renderas`[1].

[1]The value of attribute `renderas` is the `QName` of a DocBook 5.1 element. Example 1: "`http://docbook.org/ns/docbook`" is the default namespace: `renderas="book"`. Example 2: prefix "`db`" is bound to namespace "`http://docbook.org/ns/docbook`": `renderas="db:chapter"`.

- If a module pulls the content of a topic, then attribute `renderas` may be used to change the name of the pulled element.
- If a module does not pull the content of a topic, then attribute `renderas` *must be used* because it specifies the name of the realized element.

Example 3. Using attribute `renderas` to change the name of the pulled element

File `assembly2a.xml`:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>...</resources>

  <structure renderas="chapter" resourceref="topic1">
    <module renderas="section" resourceref="topic2">
      <module renderas="section" resourceref="topic3"/>
    </module>
  </structure>
</assembly>
```

The corresponding realized document is found in `realized2a.xml`:

```
<chapter xmlns="http://docbook.org/ns/docbook" version="5.1"
  xml:base="file:/home/john/samples/topic1.xml" xml:id="t1">
  <info>
    <title>Title of topic #1</title>
  </info>

  <para>Paragraph...in topic #1.</para>

  <section version="5.1" xml:base="file:/home/john/samples/topic2.xml"
           xml:id="t2">
    <info>
      <title>Title of topic #2</title>
    </info>

    <para>Paragraph...in topic #2.</para>

    <section version="5.1" xml:base="file:/home/john/samples/topic3.xml"
             xml:id="t3">
      <info>
        <title>Title of topic #3</title>
      </info>

      <para>Paragraph...in topic #3.</para>
    </section>
  </section>
</chapter>
```

Example 4. Using attribute `renderas` to create realized elements

File `assembly1a.xml`:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>...</resources>

  <structure renderas="book">
    <module renderas="chapter" resourceref="topic1"/>
    <module renderas="chapter" resourceref="topic2"/
    <module renderas="appendix" resourceref="topic3"/>
    <module renderas="index"/>
  </structure>
</assembly>
```

Notice how an empty `index` element can be added at the end of the realized book.

The corresponding realized document is found in `realized1a.xml`:

```
<book xmlns="http://docbook.org/ns/docbook" version="5.1">
```

```
  <info><title>???</title></info>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic1.xml"
           xml:id="t1">
    <info>
      <title>Title of topic #1</title>
    </info>

    <para>Paragraph...in topic #1.</para>
  </chapter>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic2.xml"
           xml:id="t2">
    <info>
      <title>Title of topic #2</title>
    </info>

    <para>Paragraph...in topic #2.</para>
  </chapter>

  <appendix version="5.1" xml:base="file:/home/john/samples/topic3.xml"
            xml:id="t3">
    <info>
      <title>Title of topic #3</title>
    </info>

    <para>Paragraph...in topic #3.</para>
  </appendix>

  <index>
    <info><title>???</title></info>
  </index>
</book>
```

5. Adding metadata to realized elements

The realized document, `realized1a.xml`, created out of assembly `assembly1a.xml`, lacks a title, author, date, etc, to be usable.

```
<book xmlns="http://docbook.org/ns/docbook" version="5.1">
  <info><title>???</title></info>
  ...
  <index>
    <info><title>???</title></info>
  </index>
</book>
```

The `info` child element of a `module` or `structure` may be used to add *metadata* to the corresponding realized element. By metadata, we mean elements `title`, `titleabbrev`, `subtitle` and `info` (with or without title elements).

Example 5. Adding metadata to a realized element

File `assembly1b.xml`:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>...</resources>

  <structure renderas="book">
    <info>
      <title>Title of the book</title>

      <author>
        <personname>John Doe</personname>
      </author>
    </info>

    <module renderas="chapter" resourceref="topic1"/>
    <module renderas="chapter" resourceref="topic2"/>
```

```
    <module renderas="appendix" resourceref="topic3"/>

    <module renderas="index">
      <info>
        <title>Index</title>
      </info>
    </module>
  </structure>
</assembly>
```

The corresponding realized document is found in `realized1b.xml`:

```
<book xmlns="http://docbook.org/ns/docbook" version="5.1">
  <info>
    <title>Title of the book</title>

    <author>
      <personname>John Doe</personname>
    </author>
  </info>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic1.xml"
           xml:id="t1">...</chapter>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic2.xml"
           xml:id="t2">...</chapter>

  <appendix version="5.1" xml:base="file:/home/john/samples/topic3.xml"
            xml:id="t3">...</appendix>

  <index>
    <info>
      <title>Index</title>
    </info>
  </index>
</book>
```

However, the `info` child element of a `module` or `structure` may *not* be used to add metadata to the realized element when this `module` or `structure` pulls some contents from a topic. In this case, unless you use element `info` (or element `merge`), it's the metadata of the pulled topic which are copied to the realized document.

Example 6. Adding/replacing metadata to/in the pulled topic

File `assembly2b.xml`:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>...</resources>

  <structure renderas="chapter" resourceref="topic1" xml:id="chapterA">
    <info>
      <title>Title of the chapter</title>

      <author>
        <personname>John Doe</personname>
      </author>
    </info>

    <module renderas="section" resourceref="topic2">
      <module renderas="section" resourceref="topic3"/>
    </module>
  </structure>
</assembly>
```

The corresponding realized document is found in `realized2b.xml`:

```
<chapter xmlns="http://docbook.org/ns/docbook" version="5.1"
         xml:base="file:/home/john/samples/topic1.xml"
         xml:id="chapterA">
  <info>
```

```
    <title>Title of the chapter</title>
    <author>
      <personname>John Doe</personname>
    </author>
  </info>

  <para>Paragraph...in topic #1.</para>

  <section version="5.1" xml:base="file:/home/john/samples/topic2.xml"
           xml:id="t2">
    <info>
      <title>Title of topic #2</title>
    </info>

    <para>Paragraph...in topic #2.</para>

    <section version="5.1" xml:base="file:/home/john/samples/topic3.xml"
             xml:id="t3">...</section>
  </section>
</chapter>
```

Adding/replacing *attributes* to/in a realized element

Notice in the above example, how attribute `xml:id="chapterA"` specified on the `structure` has been used to replace the ID of pulled topic `topic1`. Without this `xml:id="chapterA"` attribute, the ID of the realized chapter would have been "`t1`".

This happened because all common attributes[2] found on a `module` or `structure` are copied to the corresponding realized element.

5.1. Replacing all the titles of the pulled topic

In some cases, you'll want to replace all the titles of the pulled topic whatever are these titles (`title`, `titleabbrev`, `subtitle`).

This can be implemented by using attribute `omittitles=true`. When a `module` element has attribute `omittitles=true`, then the titles of the topic are omitted in the realized document. Of course, this implies that you'll have to add an `info` (or `merge`) child element containing a `title` to `module` elements having attribute `omittitles=true`.

Example 7. Replacing *all* the titles of the pulled topic

File `assembly1c.xml`:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>...</resources>

  <structure renderas="book">
    <info>
      <title>Title of the book</title>
    </info>

    <module renderas="chapter" resourceref="topic1"/>
    <module renderas="chapter" resourceref="topic2"/>

    <module omittitles="true" renderas="appendix" resourceref="topic3">
      <info>
        <title>Title of the appendix</title>
        <titleabbrev>Appendix</titleabbrev>
      </info>
    </module>

    <module renderas="index">...</module>
  </structure>
</assembly>
```

[2] More precisely, attributes `annotations`, `dir`, `remap`, `revisionflag`, `role`, `version`, `xml:base`, `xml:id`, `xml:lang`, `xreflabel`.

The corresponding realized document is found in `realized1c.xml`:

```
<book xmlns="http://docbook.org/ns/docbook" version="5.1">
  <info>
    <title>Title of the book</title>
  </info>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic1.xml"
           xml:id="t1">...</chapter>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic2.xml"
           xml:id="t2">...</chapter>

  <appendix version="5.1" xml:base="file:/home/john/samples/topic3.xml"
            xml:id="t3">
    <info>
      <title>Title of the appendix</title>
      <titleabbrev>Appendix</titleabbrev>
    </info>

    <para>Paragraph...in topic #3.</para>
  </appendix>

  <index>...</index>
</book>
```

Related information

- Section 6, “Aggregating contents pulled from topics”

6. Aggregating contents pulled from topics

A DocBook 5.1 `topic` may contain any kind of reusable content, even content much more fine-grained than some prose dealing about a single subject. For example, you may find useful creating topics containing just a single table. In such case, you'll want to create a realized element (e.g. a `section`) containing contents pulled from several of such “micro topics”.

This can be implemented by using attribute `contentonly=true`. When a `module` element has attribute `contentonly=true`, then the contents being pulled is just the “body” of the topic. That is, the root element “envelope” of the topic is omitted in the realized document. The metadata (title elements, `info` element) of the topic are also omitted in the realized document.

Example 8. Using attribute `contentonly=true` to aggregate pulled contents

File `assembly1d.xml`: the “bodies” of topics `topic1` and `topic2` are added to realized chapter "`Title of first chapter`".

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>...</resources>

  <structure renderas="book">
    <info>
      <title>Title of the book</title>
    </info>

    <module renderas="chapter" xml:id="chapter1">
      <info>
        <title>Title of first chapter</title>
      </info>

      <module contentonly="true" resourceref="topic1"/>
      <module contentonly="true" resourceref="topic2"/>
    </module>

    <module renderas="appendix" resourceref="topic3"/>

    <module renderas="index">
```

```
      <info>
        <title>Index</title>
      </info>
    </module>
  </structure>
</assembly>
```

The corresponding realized document is found in `realized1d.xml`:

```
<book xmlns="http://docbook.org/ns/docbook" version="5.1">
  <info>
    <title>Title of the book</title>
  </info>

  <chapter xml:id="chapter1">
    <info>
      <title>Title of first chapter</title>
    </info>

    <para xml:base="file:/home/john/samples/topic1.xml">
    Paragraph...in topic #1.</para>

    <para xml:base="file:/home/john/samples/topic2.xml">
    Paragraph...in topic #2.</para>
  </chapter>

  <appendix version="5.1" xml:base="file:/home/john/samples/topic3.xml"
            xml:id="t3">
    <info>
      <title>Title of topic #3</title>
    </info>

    <para>Paragraph...in topic #3.</para>
  </appendix>

  <index>
    <info>
      <title>Index</title>
    </info>
  </index>
</book>
```

Related information

- Section 5.1, “Replacing all the titles of the pulled topic”

7. Filtering contents “pulled” from topics

We have already learned that attributes `omittitles=true` and `contentonly=true` provide the author with a simple way to modify the contents “pulled” from topics. In addition to these attributes, elements `filterin` and `filterout`, which are child elements of `module` and `structure`, also allow to modify the contents pulled from topics.

Elements `filterin` and `filterout` generally have an `outputformat` attribute which specifies the class of output formats (`web`, `print`, etc)[3] to which these filters apply.

Example 9. Filtering topic contents when the "web" output format has been selected.

File `assembly1e.xml`:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>...</resources>

  <structure renderas="book">
    <filterout outputformat="web" userlevel="advanced;expert"/>❶
```

[3]Class of output formats (`web`, `print`, etc) rather than an actual file format (PDF, EPUB, RTF, Eclipse Help, etc).

```
    <info>
      <title>Title of the book</title>
      <author>
        <personname>John Doe</personname>
      </author>
    </info>

    <module renderas="chapter" resourceref="topic1"/>

    <module renderas="chapter" resourceref="topic2"/>

    <module renderas="appendix" resourceref="topic3">
      <filterin outputformat="web" userlevel="beginner;intermediate;advanced"/>❷
    </module>

    <module renderas="index">
      <filterout outputformat="web"/>❸

      <info>
        <title>Index</title>
      </info>
    </module>
  </structure>
</assembly>
```

❶ In the above example, effectivity attribute `userlevel` has 4 possible value: `beginner`, `intermediate`, `advanced` and `expert`.

This `filterout` element means: when selected output format is `web`, discard all elements pulled from topics having a `userlevel` attribute containing `advanced` and/or `expert` and no other value.

We could have specified: `<filterin outputformat="web userlevel="beginner;intermediate"/>` which means: when selected output format is `web`, discard all elements pulled from topics having a `userlevel` attribute not containing `beginner` or `intermediate`.

❷ First `filterout` element, being a child of a `structure`, applies to all pulled topics. Similarly, `filterin` and `filterout` elements contained in a `module` applies to the topics pulled from this `module` and all its `module` descendants.

This `filterin` element is used to override what is specified by the first `filterout` element. Elements pulled from topic `topic3` having a `userlevel` attribute containing `expert` and no other value are still excluded, but those having a `userlevel` attribute containing `beginner`, `intermediate` or `advanced` are included.

❸ This `filterout` element having no effectivity attribute other than `outputformat` is used to unconditionally exclude the index from the realized document when selected output format is `web`.

The corresponding realized document is found in `realized1e.xml`[4]:

```
<book xmlns="http://docbook.org/ns/docbook" version="5.1">
  <info>...</info>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic1.xml"
           xml:id="t1">
    <info>
      <title>Title of topic #1</title>
    </info>

    <para>Paragraph contained in topic #1.</para>
  </chapter>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic2.xml"
           xml:id="t2">
    <info>
```

[4]Generated by executing the following command-line:

```
assembly -format web assembly1e.xml realized1e.xml
```

where command **assembly** is XMLmind Assembly Processor.

```
      <title>Title of topic #2</title>
    </info>

    <para>Paragraph contained in topic #2.</para>
  </chapter>

  <appendix version="5.1" xml:base="file:/home/john/samples/topic3.xml"
            xml:id="t3">
    <info>
      <title>Title of topic #3</title>
    </info>

    <para>Paragraph <phrase userlevel="advanced">(that is,
    element <tag class="element">para</tag>)</phrase> contained
    in topic #3.</para>
  </appendix>
</book>
```

Note that using `filterin` and `filterout` elements in your DocBook assemblies should not prevent you from also using the "normal" conditional processing (also called *profiling*) supported by the DocBook XSL stylesheets. Elements `filterin` and `filterout` exist mainly to modify the contents of the realized document *depending on the selected output format*. Conditional processing generally uses more elaborate conditions depending on the computer architecture, operating system, security level, etc, to which the generated document applies.

Related information

- Section 8, "Different realized documents depending on selected output format"

8. Different realized documents depending on selected output format

In the previous lesson, we have learned to filter the contents "pulled" from topics using elements `filterin` and `filterout`. However the `filterin` and `filterout` elements specified in `assembly1e.xml` all have attribute `outputformat=web`. This implies that the filters are not applied to pulled contents unless output format `web` has been selected.

A `structure` element can indeed specify several realized documents depending on selected output format. So the question is: how to select an output format?

- The selected output format is passed as a parameter to the assembly processor[5].
- When this is not the case, then the selected output format is the value of attribute `defaultformat` of element `structure`.
- When this `defaultformat` attribute is missing, then the selected output format is the "*implicit format*". The "implicit format" matches `output`, `filterin`, `filterout` elements without any `outputformat` attribute.

In this lesson, we'll learn yet another method to modify the contents "pulled" from topics: element `output`, which like `filterin` and `filterout`, is a child element of `module` and `structure`.

Elements `filterin`, `filterout` and `output` are considered in order by the assembly processor and *relevant* filters are combined. A `filterin`, `filterout` or `output` element is relevant if it does not have an `outputformat` attribute or if its `outputformat` attribute matches selected output format.

Example 10. A structure specifying two different realized documents

File `assembly1f.xml`:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>...</resources>

  <structure defaultformat="print"❶ renderas="book">
    <info>
```

[5]Using means which are specific to this assembly processor.

```
      <title>Title of the book</title>

      <author>
        <personname>John Doe</personname>
      </author>
    </info>

    <module renderas="chapter" resourceref="topic1">
      <output chunk="false"❷ file="introduction.html"❸ outputformat="web"/>
    </module>

    <module renderas="chapter" resourceref="topic2"/>

    <module renderas="appendix" resourceref="topic3">
      <output transform="para2simpara"/>❹
    </module>

    <module renderas="index">
      <output outputformat="web" suppress="true"/>❺

      <info>
        <title>Index</title>
      </info>
    </module>
  </structure>

  <transforms>...</transforms>
</assembly>
```

❶ By default, the selected output format is `print` and not `web`.

❷ When selected output format is `web` —that is, when generating HTML pages— do not split the first chapter into several pages (chunks); instead create a single HTML page containing the first chapter in its entirety.

❸ When selected output format is `web`, the single HTML page containing first chapter is to be created in a file called `introduction.html`.

❹ This output directive having no `outputformat` attribute is applied whatever the output format being selected.

❺ When selected output format is `web`, the index is excluded from the realized document. This output directive has the same effect as `<filterout outputformat="web"/>`.

The corresponding realized document is found in `realized1f.xml`[6]:

```
<book xmlns="http://docbook.org/ns/docbook" version="5.1">
  <info>...</info>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic1.xml"
           xml:id="t1">
    <info>
      <title>Title of topic #1</title>
    </info>

    <para>Paragraph...in topic #1.</para>
  </chapter>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic2.xml"
           xml:id="t2">
    <info>
      <title>Title of topic #2</title>
    </info>

    <para>Paragraph...in topic #2.</para>
```

[6]Generated by executing the following command-line:

```
assembly assembly1f.xml realized1f.xml
```

or:

```
assembly -format print assembly1f.xml realized1f.xml
```

where command **assembly** is XMLmind Assembly Processor.

```
  </chapter>

  <appendix version="5.1" xml:base="file:/home/john/samples/topic3.xml"
            xml:id="t3">
    <info>
      <title>Title of topic #3</title>
    </info>

    <db:simpara xmlns:db="http://docbook.org/ns/docbook">
    Paragraph...in topic #3.</db:simpara>
  </appendix>

  <index>
    <info>
      <title>Index</title>
    </info>
  </index>
</book>
```

When output format `web` has been selected, the realized document is found in `realized1f-web.xml`[7]:

```
<book xmlns="http://docbook.org/ns/docbook" version="5.1">
  <info>...</info>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic1.xml"
           xml:id="t1">
    <?dbhtml filename="introduction.html"?>
    <?dbhtml stop-chunking?>
    <info>
      <title>Title of topic #1</title>
    </info>

    <para>Paragraph...in topic #1.</para>
  </chapter>

  <chapter version="5.1" xml:base="file:/home/john/samples/topic2.xml"
           xml:id="t2">
    <info>
      <title>Title of topic #2</title>
    </info>

    <para>Paragraph...in topic #2.</para>
  </chapter>

  <appendix version="5.1" xml:base="file:/home/john/samples/topic3.xml"
            xml:id="t3">
    <info>
      <title>Title of topic #3</title>
    </info>

    <db:simpara xmlns:db="http://docbook.org/ns/docbook">
    Paragraph...in topic #3.</db:simpara>
  </appendix>
</book>
```

Related information

- Section 7, “Filtering contents “pulled” from topics”

9. Transforming contents “pulled” from topics

The ultimate way to transform contents “pulled” from topics is to apply an *XSLT stylesheet* to the topic prior to copying its content to the realized document.

[7] Generated by executing the following command-line:

```
assembly -format web assembly1f.xml realized1f-web.xml
```

where command **assembly** is XMLmind Assembly Processor.

Transforming a topic is specified by the means of attribute `grammar` of element `resource` or by attributes `transform` or `grammar` of element `output`. In both cases, the transformation must have been declared by the means of a `transform` element found in the `transforms` section of the `assembly`.

Example 11. Converting `para` elements to `simpara` elements in the contents pulled from a topic

In `assembly1f.xml`, we have specified:

```
    <module renderas="appendix" resourceref="topic3">
      <output transform="para2simpara"/>
    </module>
```

where `transform` element is declared as follows in the `transforms` section of the `assembly`:

```
  <transforms>
    <transform href="simpara.xsl" name="para2simpara"/>
  </transforms>
```

File `simpara.xsl` contains a simple XSLT 1.0 stylesheet which transform all DocBook `para` elements to `simpara` elements:

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

<xsl:output method="xml" encoding="UTF-8" />

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

...

</xsl:stylesheet>
```

This transformation applied to topic `topic3` results in (excerpts from `realized1f.xml`).

```
  <appendix version="5.1" xml:base="file:/home/john/samples/topic3.xml"
            xml:id="t3">
    <info>
      <title>Title of topic #3</title>
    </info>

    <db:simpara xmlns:db="http://docbook.org/ns/docbook">
    Paragraph...in topic #3.</db:simpara>
  </appendix>
```

10. Creating navigation links between related topics

In order to make topics as reusable as possible in different contexts, it is recommended to avoid creating cross-references (`xref`, `link`) between the topics. Instead, it's possible to specify navigation links in the `assembly` by the means of element `relationship`. The `relationship` elements are grouped in the `relationships` sections of the `assembly`.

Example 12. Creating "See also" links between two topics

File `assembly1g.xml`:

```
<assembly version="5.1" xmlns="http://docbook.org/ns/docbook">
  <resources>...</resources>

  <structure renderas="book">
```

```
    <info>...</info>

    <module renderas="chapter" resourceref="topic1"/>
    <module renderas="chapter" resourceref="topic2"/>
    <module renderas="appendix" resourceref="topic3"/>

    <module renderas="index">...</module>
  </structure>

  <relationships>
    <relationship>
      <association>See also</association>

      <instance linkend="topic1"/>
      <instance linkend="topic2"/>
    </relationship>
  </relationships>
</assembly>
```

The corresponding realized document is found in `realized1g.xml`:

```
<book version="5.1" xmlns="http://docbook.org/ns/docbook">
  <info>...</info>

  <chapter version="5.1"
           xml:base="file:/home/john/samples/topic1.xml"
           xml:id="t1">
    <info>
      <title>Title of topic #1</title>
    </info>

    <para>Paragraph...in topic #1.</para>

    <itemizedlist>
      <title>See also</title>
      <listitem>
        <para><xref linkend="t2"/></para>
      </listitem>
    </itemizedlist>
  </chapter>

  <chapter version="5.1"
           xml:base="file:/home/john/samples/topic2.xml"
           xml:id="t2">
    <info>
      <title>Title of topic #2</title>
    </info>

    <para>Paragraph...in topic #2.</para>

    <itemizedlist>
      <title>See also</title>
      <listitem>
        <para><xref linkend="t1"/></para>
      </listitem>
    </itemizedlist>
  </chapter>

  <appendix version="5.1"
            xml:base="file:/home/john/samples/topic3.xml"
            xml:id="t3">
    <info>
      <title>Title of topic #3</title>
    </info>

    <para>Paragraph...in topic #3.</para>
  </appendix>

  <index>...</index>
</book>
```

11. Processing DocBook assemblies

Free, open source, XMLmind Assembly Processor and free, open source, DocBook XSL stylesheets are all the tools you need to convert your DocBook 5.1 assemblies to a number of formats (PDF, HTML, EPUB, etc).

However, if you prefer an integrated tool which, in addition to processing DocBook 5.1 assemblies, also allows to quickly and easily create assemblies and topics, then you'll want to give XMLmind DocBook Editor (or XMLmind XML Editor) Personal Edition v7.3+ a try. More information in "*XMLmind Assembly Processor Manual - What if you just want to quickly experiment with DocBook assemblies and topics?*".

Figure 1. An assembly opened in XMLmind XML Editor