XMLmind Assembly Processor Manual

Hussein Shafie

XMLmind Assembly Processor Manual

Hussein Shafie

Publication date September 15, 2023

Abstract

Explains how to install, use and embed XMLmind Assembly Processor.

Table of Contents

List of Figures

List of Examples

Chapter 1. What is XMLmind Assembly Processor?

XMLmind Assembly Processor is a Java™ software component and a command-line utility (called **assembly**) which processes a DocBook v5.1 `assembly` and all the referenced topics in order to create the equivalent "flat", monolithic, document (e.g. a DocBook v5.1 `book`). This equivalent document is called the *realized* document.

The realized document is then transformed to other formats (PDF, HTML, etc), normally, as if it were created by hand, using the DocBook XSL stylesheets.

As of v0.9.3, XMLmind Assembly Processor includes XInclude 1.1 and DocBook Transclusion processors, which allows to create modular DocBook v5.1 documents without facing limitations.

XMLmind Assembly Processor is free, open source, software, which like the DocBook XSL stylesheets, is licensed under the terms of the MIT License.

Chapter 2. Installing XMLmind Assembly Processor

2.1. System requirements

Make sure that you have a Java™ 8+ runtime installed on your machine. To check this, please open a command window and type "**java -version**" followed by Enter. You should get something looking like this:

```
C:\> java -version
openjdk version "20.0.2" 2023-07-18
OpenJDK Runtime Environment (build 20.0.2+9-78)
OpenJDK 64-Bit Server VM (build 20.0.2+9-78, mixed mode)
```

2.2. Installation

Simply unzip `assembly-`*`X_Y_Z`*`.zip` in any directory.

After that, you can run command-line utility **assembly** by simply executing *`assembly_install_dir`*`/bin/assembly.bat` (*`assembly_install_dir`*`/bin/assembly` on the Mac and on Linux).

```
C:\> mkdir XMLmind
C:\> cd XMLmind
C:\XMLmind> unzip assembly-1_3_3.zip
C:\XMLmind> dir assembly-1_3_3
... <DIR> bin
... <DIR> doc
... <DIR> docsrc
... <DIR> legal
...
C:\XMLmind> assembly-1_3_3\bin\assembly.bat
assembly: ERROR: too few command-line arguments
Usage: assembly [option]* in_assembly_file out_docbook_file|-
...
```

See also

- Section 2.1, “System requirements”

2.3. Contents of the installation directory

`bin/`, `bin/assembly`, `bin/assembly.bat`

Contains the **assembly** command-line utility. Use shell script `assembly` on Linux and on the Mac. Use `assembly.bat` on Windows.

`doc/`, `doc/index.html`

Contains the documentation of XMLmind Assembly Processor.

`docsrc/`, `docsrc/manual.xml`

Contains the DocBook v5.1 source of the documentation of XMLmind Assembly Processor. File `docsrc/manual.xml` contains an assembly. You may want to use this sample DocBook v5.1 assembly to experiment with the **assembly** command-line utility.

`legal/`, `legal.txt`

Contains legal information about XMLmind Assembly Processor and about third-party components used in XMLmind Assembly Processor.

`lib/*.jar`

All the Java™ class libraries needed to run XMLmind Assembly Processor:

- `assembly.jar` contain the code of XMLmind Assembly Processor.
- `xmlresolver.jar` contains XMLResolver, an enhanced XML resolver with XML Catalog support

  This component is needed only if your assembly loads XML documents having a DTD (e.g. transform DITA topics to DocBook topics).

`src/`, `src/build.xml`

Contains the Java™ source code of XMLmind Assembly Processor. `src/build.xml` is an ant build file which allows to rebuild `lib/assembly.jar`.

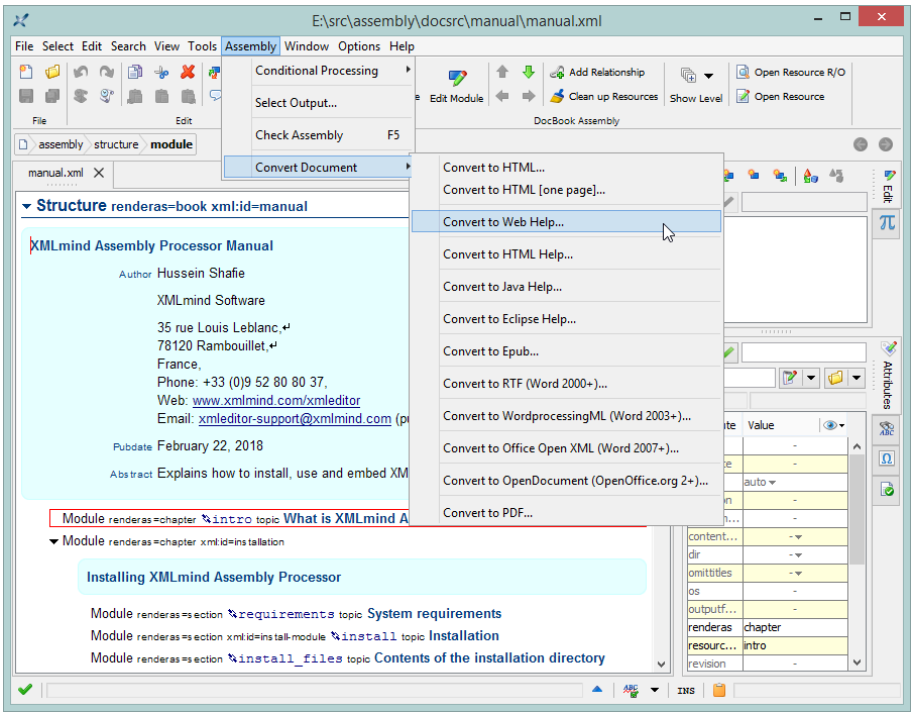
Chapter 3. Getting started

What if you just want to quickly experiment with DocBook assemblies and topics?

The simplest is to download and install XMLmind DocBook Editor (or XMLmind XML Editor) Personal Edition v7.3+ from http://www.xmlmind.com/xmleditor/download.shtml.

You can then open this document —"*XMLmind Assembly Processor Manual*", an assembly found in *`assembly_install_dir`*`/docsrc/manual.xml`— in XMLmind DocBook Editor (or XMLmind XML Editor) and use menu Assembly → Convert Document to convert it to any format you want.

Figure 3.1. This assembly `manual.xml` opened in XMLmind XML Editor



In order to explain how XMLmind Assembly Processor is used, we'll convert this document —"*XMLmind Assembly Processor Manual*", an assembly found in *`assembly_install_dir`*`/docsrc/manual.xml`— first to multi-page HTML and then to PDF.

You'll find in *`assembly_install_dir`*`/docsrc/convert_manual.bat` (Windows) and in *`assembly_install_dir`*`/docsrc/convert_manual.sh` (Linux, Mac) a copy of the commands used in this chapter.

Assembly `manual.xml` contains a `structure` specifying a book. The content of the chapters and sections of this book is obtained from topics such as `intro.xml`, `requirements.xml`, `install.xml`, etc.

```
<structure renderas="book" xml:id="manual">
  <info>
    <title>XMLmind Assembly Processor Manual</title>
    <author>
      <personname>...</personname>
      ...
```

```
  </info>
  <module renderas="chapter" resourceref="intro"/>
  <module renderas="chapter" xml:id="installation">
    <info>
      <title>Installing XMLmind Assembly Processor</title>
    </info>
    <module renderas="section" resourceref="requirements"/>
    <module renderas="section" resourceref="install"/>
    ...
</structure>
```

Example 3.1. Converting `manual.xml` to multi-page HTML

First, generate a “flat”, monolithic, book out of assembly `manual.xml`. This equivalent “flat” document is called the *realized* document.

```
C:\XMLmind\assembly-1_3_3\docsrc> ..\..\bin\assembly1¬
  -v2 -format web3¬
  manual.xml out\manual_realized_web.xml4
```

1. The assembly processor command-line utility is called **assembly** and is found in *`assembly_install_dir`*`/bin/`.
2. Option `-v` turns on the verbosity of **assembly**.
3. Option `-format web` instructs **assembly** to target the "`web`" output format. Because the structure found in assembly `manual.xml` ends with:

   ```
      ...
    <module renderas="index">
      <filterout outputformat="web"/>
      <info><title>Index</title></info>
    </module>
  </structure>
   ```

   this is used to exclude the index from the generated HTML pages.
4. The realized book is created in `out/manual_realized_web.xml`. You may want to open this file in a text or XML editor to see by yourself that it looks very much like a hand-written DocBook book.

Finally, convert `out/manual_realized_web.xml` to multi-page HTML using the DocBook XSL stylesheets.

```
C:\XMLmind\assembly-1_3_3\docsrc> java -cp C:\...\saxon.jar¬
  com.icl.saxon.StyleSheet¬
  -o ..\..\doc\manual\index.html out\manual_realized_web.xml1¬
  C:\...\docbook-xsl-ns-1.79.1\html\chunk.xsl2¬
  base.dir=../../doc/manual/3¬
  chunk.section.depth=0¬
  section.autolabel=1¬
  section.label.includes.component.label=1¬
  use.id.as.filename=1
```

1. `out/manual_realized_web.xml` is the file to be transformed to multi-page HTML.
2. `html/chunk.xsl` is the XSLT stylesheet used to generate multi-page HTML.
3. *`base.dir`*, *`chunk.section.depth`*, etc, are all parameters passed to the XSLT stylesheet. The HTML files are created in `../../doc/manual/` as specified by *`base.dir`*.

What if an assembly contains several structure?

Assembly `manual.xml` contains only a single structure. However, an assembly may contain several structure. By default, **assembly** processes first found structure. Option `-`

`struct` *`structure_ID`* allows to specify the `xml:id` of the structure to be processed. In the case of the above example, we could have invoked "`assembly -struct manual`" because the structure found in assembly `manual.xml` starts with:

```
<structure renderas="book" xml:id="manual">
  ...
```

Example 3.2. Converting `manual.xml` to PDF

First, generate a “flat”, monolithic, book out of assembly `manual.xml`. The realized book is created in `out/manual_realized.xml`.

```
C:\XMLmind\assembly-1_3_3\docsrc> ..\..\bin\assembly -v¬
  manual.xml out\manual_realized.xml
```

Second, transform `out/manual_realized.xml` to XSL-FO (a standard page description format) using XSLT stylesheet `fo/docbook.xsl`.

```
C:\XMLmind\assembly-1_3_3\docsrc> java -cp C:\...\saxon.jar¬
  com.icl.saxon.StyleSheet¬
  -o out\manual_realized.fo out\manual_realized.xml¬
  C:\...\docbook-xsl-ns-1.79.1\fo\docbook.xsl¬
  paper.type=A4¬
  section.autolabel=1¬
  section.label.includes.component.label=1¬
  variablelist.as.blocks=1¬
  ulink.show=0¬
  shade.verbatim=1
```

Finally, convert the XSL-FO file to PDF using Apache FOP. The PDF file is created in `../../doc/manual/manual.pdf`.

```
C:\XMLmind\assembly-1_3_3\docsrc> C:\...\fop-2.7\fop¬
  -fo out\manual_realized.fo¬
  -pdf ..\..\doc\manual\manual.pdf
```

Chapter 4. Command-line options

The **assembly** command-line utility (found in *assembly_install_dir*/bin/) is auto-documented. Suffice to execute:

```
C:\assembly\bin> assembly
```

in order to display the following help.

Usage

```
assembly option* in_assembly_file out_docbook_file|-
```

Processes a structure found in assembly file *in_assembly_file* and creates realized DocBook document in *out_docbook_file*.

If *out_docbook_file* is specified as "-", then *in_assembly_file* is simply checked for errors.

Options are:

`-struct` *structure_id*

Specifies the `xml:id` of the structure to be processed.

By default, it's the first found structure.

`-format` *output_format*

Specifies the target output format.

By default, it's the default format of the processed structure if any, and the "*implicit format*" otherwise. The "implicit format" matches `output`, `filterin`, `filterout` elements without any `outputformat` attribute.

Multiple output formats separated by "`;`" may be specified. For example, "`EPUB;expert`" means output format is "`EPUB`" *or* "`expert`".

`-check`

Check realized document for cross-reference errors, missing image resources, etc. This option reports warnings, not errors. Thus this option does not prevent the realized document from being saved to disk.

Note

If your document requires conditional processing (that is, *profiling*), then the check step may report false errors. These false errors are caused by the fact that the conditional processing step has not been applied to the realized document prior to the check step.

Example: two of the chapters referenced by assembly `book.xml` have `xml:id="install"`. First chapter has also `os="windows"`. Second chapter has also `os="mac"`.

If you run `assembly -check book.xml -`, you'll get a duplicate ID warning caused by `xml:id="install"`.

On the other hand, if you run `assembly -check -profile os windows book.xml -`, you'll not have this duplicate ID error. Why that? Because by applying profile `os="windows"`, second chapter (having `os="mac"`) is excluded from the realized document prior to checking it.

`-profile` *`attribute_name attribute_value`*

Specifies a profiling attribute. One or more `-profile` options allows to specify the profile applied to the realized document prior to checking it. Specifying one or more `-profile` options is not useful unless you also use the `-check` option.

`-v`, `-vv`, `-vvv`

Turn verbosity on. More Vs means more verbose.

`-version`

Print version number and exit.

Using XML catalogs to resolve entities and URIs

An assembly may reference *virtual topics*. Let's call virtual topics, topics which are not expressed in DocBook v5.1 and which need to be converted on the fly to DocBook v5.1 before being added to the realized document. This facility is implemented by the means of the `transform` element and the `grammar` or `transform` attributes of the `output` element.

These virtual topics often start with a `DOCTYPE`. DITA example:

```
<!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN"
"http://docs.oasis-open.org/dita/v1.2/os/topic.dtd">
<topic id="topic">
  <title>A DITA Topic</title>
  ...
```

Unless you specify an XML catalog pointing to a local copy of `topic.dtd`, **assembly** will download this DTD from `http://docs.oasis-open.org/dita/v1.2/os/topic.dtd`, which may take a while or even fail.

XML catalogs are specified by the means of environment variable `XML_CATALOG_FILES`[1]. This variable contains a semicolon-delimited list of catalog files or URIs. Windows example:

```
C:\>set XML_CATALOG_FILES=..\..\ditac\schema\catalog.xml
```

[1] Or equivalent Java™ system property xml.catalog.files.

Chapter 5. Limitations and implementation specificities

5.1. Limitations

- Attribute `type` of element `structure` is ignored.
- Element `relationships` is supported. However attribute `type` of elements `relationships` and `relationship` is still ignored.

Related information

- Section 5.2, “Implementation specificities”

5.2. Implementation specificities

Elements `filterin` and `filterout` versus profiling using the DocBook XSL stylesheets

- The use of the `outputformat` attribute and/out `filterin` and `filterout` elements in an `assembly` and the use of conditional processing (profiling) by the means of the DocBook XSL stylesheets is orthogonal.

  It's certainly possible to have an assembly making use of the `outputformat` attribute and containing `filterin` and `filterout` elements and also to pass the equivalent of XSLT stylesheet parameters `profile.`*`attr_name=attr_value`* (that is, a profile) to XMLmind Assembly Processor. However, in this case, *XMLmind Assembly Processor will not apply this profile to the assembly itself prior to processing the assembly*.

  The `-profile` command-line option allows to apply a profile to the *realized document* (and not to the assembly itself) before checking the realized document for cross-reference errors. This option is not useful unless the `-check` option is also used.

Generating links using element `relationships`

- The `instance` child elements of a `relationships` elements are copied to all the `relationship` child elements of a `relationships` elements. Example:

```
<relationships type="seealso sequence">
  <instance linkend="tut1"/>

  <relationship>
    <association></association>
    <instance linkend="tut2"/>
  </relationship>

  <relationship type="path">
    <association>Quick start</association>
    <instance linkend="tut3"/>
  </relationship>
</relationships>
```

  is equivalent to:

```
<relationships>
  <relationship type="seealso">
    <association></association>
```

```
    <instance linkend="tut2"/>
    <instance linkend="tut1"/>
  </relationship>

  <relationship type="path">
    <association>Quick start</association>
    <instance linkend="tut3"/>
    <instance linkend="tut1"/>
  </relationship>
</relationships>
```

- For now, attribute `type` of elements `relationships` and `relationship` does not contribute to the realized document. However notice in above example how first token (`seealso` in above example) found in `relationships/@type` is “inherited” by the `relationship` child elements not having this attribute (first `relationship` child in above example). Also note that tokens other than the first one (`sequence` in above example) are not “inherited”.

- Element `relationships` may have just `instance` child elements and no `relationship` child elements. Example:

```
<relationships type="seealso">
  <instance linkend="tut1"/>
  <instance linkend="tut2"/>
</relationships>
```

  is equivalent to:

```
<relationships>
  <relationship type="seealso">
    <association></association>
    <instance linkend="tut1"/>
    <instance linkend="tut2"/>
  </relationship>
</relationships>
```

- The `linking` attribute of element `instance` is supported with the following values: `sourceonly`, `targetonly`, `normal`, `none`, with the same semantics as the corresponding `linking` attribute of DITA.

- The links are generated at the end of the realized modules in the form of an `itemizedlist` having attribute `remap="relationships"` and starting with a `title` child element. The text of this `title` comes from `relationship`/`association`.

- An empty `association` may be used to specify the default title for a group of links. This default title is "`Related information`" (translated to the language —`xml:lang`— of the realized module).

- No links are generated at the end of realized modules having attribute `contentonly="true"`.

- The `linkend` attribute of element `instance` may contain the `xml:id` of a resource or the `xml:id` of a module. Example:

```
<relationships>
  <instance linkend="tut1"/>
  <instance linkend="tut2-module"/>
</relationships>
```

  is equivalent to:

```
<relationships>
```

```
  <instance linkend="tut1-module"/>
  <instance linkend="tut2-module"/>
</relationships>
```

for an `assembly` containing:

```
<resource xml:id="tut1" href="tut1.xml"/>
<resource xml:id="tut2" href="tut2.xml"/>
<resource xml:id="tut3" href="tut3.xml"/>
...

<module xml:id="tut1-module" resourceref="tut1">
<module xml:id="tut2-module">
  <info>
    <title>Going further with FooBar</title>
  </info>
  <module resourceref="tut2" contentonly="true"/>
  <module resourceref="tut3" contentonly="true"/>
</module>
```

Adding/replacing common attributes to the realized document

- All common attributes (`annotations`, `dir`, `remap`, `revisionflag`, `role`, `version`, `xml:base`, `xml:id`, `xml:lang`, `xreflabel`) found on a `module` or `structure` are copied to the corresponding realized element, *possibly replacing the same common attributes already set on the resource*. Example:

  ```
  <module renderas="section" xml:id="sect01" resourceref="t1">
  ```

  where resource `t1` contains:

  ```
  <topic version="5.1" xml:id="t1 ...
  ```

  gives realized section:

  ```
  <section version="5.1" xml:base="... t1.xml" xml:id="sect01"> ...
  ```

Adding/replacing metadata to the realized document

- If a `module` or `structure` has both `info` and `merge` child elements then both child elements contribute to the metadata of the corresponding realized element.
- Elements `info` and `merge` may used to replace, but also to *add*, child elements to the `info` of the corresponding realized element.
- Element `merge` may have both a `resourceref` attribute and child elements.

Controlling chunking

- Module or output attribute `chunk=false` may be used to add a `<?dbhtml stop-chunking>` processing-instruction to the corresponding realized element. The other `chunk` attribute values `true` and `auto` are not supported.
- Output attribute `file=`*`dir/basename`* may be used to add a `<?dbhtml dir="`*`dir`*`" filename="`*`basename`*`">` processing-instruction to the corresponding realized element.

Miscellaneous

- The `href` attribute of a `resource` element may have a fragment. However a fragment is supported only if the resource is a native DocBook v5+ document which does not need to be transformed.

- XMLmind Assembly Processor uses a built-in XInclude 1.1 processor rather than the XInclude 1.0 implementation provided by the XML parser (that is, Xerces). Note that for now, this built-in XInclude 1.1 processor only supports the XPointer element() scheme.

Related information

- Section 5.1, “Limitations”

5.2.1. About filtering

- Element `filterin` is best explained by an example:

```
<module resourceref="MyTopic"/>
<filterin os="mac" userlevel="beginner;intermediate"/>
```

The above example means: *exclude* from the contents of realized topic `MyTopic` all the elements having a `os` attribute not containing `mac` and also exclude all the elements having a `userlevel` attribute not containing `beginner` or `intermediate`.

For those who know the DocBook XSL stylesheets, this is equivalent to passing parameters `profile.os=mac` and `profile.userlevel=beginner;intermediate` to the profiling stylesheets.

For example, if resource `MyTopic` points to a topic containing:

```
<para xml:id="p1" os="windows">Paragraph #1.</para>
<para xml:id="p2" os="mac;linux">Paragraph #2.</para>
<para xml:id="p3" userlevel="advanced;expert">Paragraph #3.</para>
<para xml:id="p4" userlevel="intermediate;advanced">Paragraph #4.</para>
```

then paragraph `p1` and `p3` are excluded from the realized document while paragraphs `p2` and `p4` are not.

- Element `filterout` is best explained by an example:

```
<module resourceref="MyTopic"/>
<filterout os="mac" userlevel="beginner;intermediate"/>
```

The above example means: exclude from the contents of realized topic `MyTopic` all the elements having a `os` attribute containing `mac` and no other value and also exclude all the elements having a `userlevel` attribute containing `beginner` and/or `intermediate` and no other value.

For example, if resource `MyTopic` points to a topic containing:

```
<para xml:id="p1" os="mac">Paragraph #1.</para>
<para xml:id="p2" os="mac;linux">Paragraph #2.</para>
<para xml:id="p3" userlevel="beginner">Paragraph #3.</para>
<para xml:id="p4" userlevel="intermediate;advanced">Paragraph #4.</para>
```

then paragraph `p1` and `p3` are excluded from the realized document while paragraphs `p2` and `p4` are not.

- It's possible to have both `filterin` and `filterout` elements for the same effectivity attribute. Example:

```
<module resourceref="MyTopic"/>
<filterin os="windows;mac"/>
<filterout os="linux"/>
```

For example, if resource `MyTopic` points to a topic containing:

```
<para xml:id="p1" os="linux">Paragraph #1.</para>
<para xml:id="p2" os="mac;linux">Paragraph #2.</para>
<para xml:id="p3" os="android">Paragraph #3.</para>
<para xml:id="p4" os="android;windows">Paragraph #4.</para>
<para xml:id="p5" os="android;linux">Paragraph #5.</para>
```

then paragraph `p1`, `p3` and `p5` are excluded from the realized document while paragraphs `p2` and `p4` are not.

- Just like `output` elements, `filterin` and `filterout` elements are considered in order and relevant filters are combined. A `filterin` or `filterout` element is relevant if it does not have an `outputformat` attribute or if its `outputformat` attribute matches the output format passed as a parameter to the assembly processor.

- The sequence of `filterin` and `filterout` elements "inherited" from ancestor `structure` and `module`s and/or directly added to a `module` is combined to form a single `filterin` element and a single `filterout` element. The resulting single `filterin` element and single `filterout` element are applied to the `module`. Example:

```
<filterin os="windows"/>
<filterout userlevel="beginner;intermediate"/>
<filterin os="mac;linux"/>
<filterout os="linux"/>
<filterin userlevel="intermediate;advanced"/>
```

The above sequence is equivalent to:

```
<filterin os="windows;mac" userlevel="intermediate;advanced"/>
<filterout os="linux" userlevel="beginner"/>
```

Chapter 6. Embedding XMLmind Assembly Processor in a Java™ application

1. Add *assembly_install_dir*/lib/assembly.jar to your CLASSPATH.

   Optionally, if your assembly loads XML documents having a DTD (e.g. transform DITA topics to DocBook topics), also add*assembly_install_dir*/lib/xmlresolver.jar to your CLASSPATH. File xmlresolver.jar contains https://xmlresolver.org/ an enhanced XML resolver with XML Catalog support.

2. Create an instance of Processor:

   ```
   Processor processor = new Processor();
   ```

   ### Important

   Do not share this instance between different threads, as this class is *not* thread-safe.

   If you don't want the error, warning and progress messages to be displayed on System.err and to System.out, implement interface Console and pass an instance of your implementation to the constructor.

3. Parameterize the processor by invoking either configure(String[]) or individual configuration methods such as setProcessedStructId(String), setOutputFormat(String), etc. Example:

   ```
   int l = -1;
   try {
       l = processor.configure(args);
   } catch (IllegalArgumentException e) {
       // FATAL ERROR. DO SOMETHING HERE.
   }

   // PARSE THE REMAINING ARGUMENTS, IF ANY,
   // STARTING AT INDEX l.
   ```

4. Finally invoke method process(URL, File). Pass this method the input assembly URL and the output realized document save file.

   ```
   try {
       if (!processor.process(inURL, outFile)) {
           // FATAL ERROR. DO SOMETHING HERE.
           // ERRORS HAVE BEEN DISPLAYED ON THE Console.
       }
   } catch (IOException e)
       // FATAL ERROR. DO SOMETHING HERE.
   }
   ```

Appendix A. History of changes

v1.3.3 (September 15, 2023)

- Upgraded XMLResolver to version 5.2.1.

v1.3.2 (July 13, 2023)

- Upgraded XMLResolver to version 5.2.0.
- Modified the language fixup of the XInclude 1.1 implementation in order to support the `lang` attribute in lieu of or in addition to the `xml:lang` attribute. This allows to use this XInclude 1.1 implementation to process DocBook 4 or XHTML documents. (Note that this XInclude 1.1 implementation already supported the `id` attribute in lieu of the `xml:id` attribute.)

v1.3.1 (April 20, 2023)

XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported on Java™ 20 platforms.

v1.3 (March 8, 2023)

- **Bug fix:** made the language fixup of the XInclude 1.1 implementation more conforming to the specification.
- Upgraded XMLResolver to version 5.1.1.

v1.2 (December 5, 2022)

- Replaced the Apache Commons Resolver (`lib/resolver.jar`) by the XMLResolver (`lib/xmlresolver.jar`).
- XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported on Java™ 19 platforms.

v1.1.1 (April 12, 2022)

XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported on Java™ 18 platforms.

v1.1 (November 15, 2021)

- Some internal changes were needed to make XMLmind Assembly Processor compatible with XMLmind XML Editor v10+.
- XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported
  - on Java™ 17 platforms;
  - on macOS Monterey (version 12.0), including on Macs having an Apple M1 (ARM-based) processor;
  - on Windows 11.

v1.0.12 (May 14, 2021)

XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported on Java™ 16 platforms.

v1.0.11 (November 30, 2020)

XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported on Java™ 15 platforms.

v1.0.10 (July 23, 2020)

- **Bug fix:** XMLmind Assembly Processor reported "`missing child element <info>`" warning for assembly elements like `<module renderas="index"/>`. (`<module renderas="index"/>` is realized as a DocBook `index` element and an `index` element is not required to start with an `info` element or to have a title.)
- XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported on Java™ 14 platforms.

v1.0.9 (February 26, 2020)

The `itemizedlist` which is automatically generated to represent relationships between topics has now attribute `spacing="compact"`.

v1.0.8 (January 16, 2020)

XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported on Java™ 13 platforms.

v1.0.7_01 (May 17, 2019)

XMLmind Assembly Processor now requires a Java 8+ runtime in order to compile and run.

v1.0.7 (March 25, 2019)

XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported on Java™ 12 platforms.

v1.0.6 (November 27, 2018)

XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported on Java™ 11 platforms.

v1.0.5 (August 21, 2018)

Minor internal changes needed to make XMLmind Assembly Processor compatible with XMLmind XML Editor v8.2.

v1.0.4 (May 22, 2018)

XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported on Java™ 10 platforms.

v1.0.3 (February 24, 2018)

Minor internal changes needed to make XMLmind Assembly Processor compatible with XMLmind XML Editor v8.

v1.0.2_01 (December 11, 2017)

XMLmind Assembly Processor, which passed all non-regression tests, is now officially supported on Java™ 9 platforms.

v1.0.2 (June 09, 2017)

Changed "Licensor" from "Pixware SARL" to "XMLmind Software" in all licenses.

v1.0.1 (March 06, 2017)

Bug fixes:

- Info elements copied from the assembly to the realized document were not given proper `xml:base` and `xml:lang` attributes.

  In practice, due to this bug, it was not possible to successfully convert to other formats an assembly where some `info` elements contained `imagedata`.

v1.0.0 (December 22, 2016)

XMLmind Assembly Processor v1.0 is the first version to fully support processing DocBook 5.1 assemblies. This includes XInclude 1.1, DocBook Transclusion and assembly features such as `relationships` and `transforms`.

Enhancements:

- Element `relationships` is now processed. The `linking` attribute of element `instance` is supported with the following values: `sourceonly`, `targetonly`, `normal`, `none`, with the same semantics as the corresponding `linking` attribute of DITA. Attribute `type` of elements `relationships` and `relationship` is still ignored.

Bug fixes:

- The `revhistory` child of the `structure` element was ignored.

Incompatibilities:

- Attribute `omittitles` now discards `title`, `titleabbrev`, `subtitle` from the included resource. Previously, setting this attribute to `true` discarded all the metadata just like `contentonly="true"`.

- In `structure` and in `module`, `info` and `merge` are used in the same way. These elements may be used to add or replace metadata in the realized `structure` or `module`.

- The implementation of `filterin` and `filterout` is completely different from what it was in previous versions. More information in Section 5.2.1, “About filtering”.

v0.9.4 (February 16, 2016)

- XMLmind Assembly Processor can now check the realized document for cross-reference errors, missing image resources, etc. This is done by passing new `-check` option to the **assembly** command-line.

  Note that if your document requires conditional processing (that is, *profiling*), then this check step may report false errors. These false errors are caused by the fact that the conditional processing step has not been applied to the realized document prior to the check step.

New option `-profile` allows to specify a profiling attribute. Therefore passing one or more `-profile` options to the **assembly** command-line allows to apply a profile to the realized document prior to checking it.

Example: two of the chapters referenced by assembly `book.xml` have `xml:id="install"`. First chapter has also `os="windows"`. Second chapter has also `os="mac"`.

If you run `assembly -check book.xml -`, you'll get a duplicate ID warning caused by `xml:id="install"`.

On the other hand, if you run `assembly -check -profile os windows book.xml -`, you'll not have this duplicate ID error. Why that? Because by applying profile `os="windows"`, second chapter (having `os="mac"`) is excluded from the realized document prior to checking it.

- DocBook 5 Transclusion Processor: removed `targetptr` from the IDREF-list.

v0.9.3 (February 03, 2016)

- Now uses a built-in XInclude 1.1 processor rather than the XInclude 1.0 implementation provided by the XML parser (that is, Xerces). Note that for now, this built-in XInclude 1.1 processor only supports the XPointer element() scheme.
- Now uses a built-in DocBook 5 Transclusion Processor to process `trans:idfixup`, `trans:suffix` and `trans:linkscope` attributes possibly set on `xi:include` elements.

v0.9.2 (January 22, 2016)

Bug fix: in some cases, a processing-instruction was moved (in the realized document) to the beginning of the element containing it and this, no matter its actual location within this parent element.

v0.9.1 (September 8, 2015)

Minor internal changes.

v0.9.0 (June 24, 2015)

First release.

Index