

xtpxlib-xdoc

An xtpxlib component for generating documentation

0 Table of Contents

0 Documentation generation with xtpplib-xdoc	2
0.1 Installing xtpplib-xdoc	2
0.2 Running xtpplib-xdoc	2
1 Description	3
1.1 Overview	3
1.2 The xtpplib-xdoc main toolchain	3
1.3 Documentation website generation	4
2 xdoc extensions	5
2.1 xdoc extension elements	5
2.1.1 Extension element xdoc:dump-parameters	5
2.1.2 Extension element xdoc:transform	5
2.2 Built-in xdoc transformations	5
2.2.1 XProc 1.0 pipeline: code-docgen.xpl	5
2.2.2 XProc 1.0 pipeline: xml-description.xpl	6
2.3 Writing your own xdoc transformations	6
3 DocBook dialect	7
3.1 Supported root elements	7
3.2 Document information	7
3.3 Chapter/Section structure	7
3.4 Block constructions	7
3.5 Inline elements	9
3.6 Other constructs	11
3.7 Fixed-width column mechanism	11

0 Documentation generation with xtpplib-xdoc



Figure 0-1 - Xatapult Content Engineering

Component **xtpplib-xdoc** version {`$current-release-version`} - {`$current-release-date`}
 Xatapult Content Engineering - <http://www.xatapult.com> - +31 6 53260792
 Erik Siegel - erik@xatapult.com

xtpplib-xdoc is a component for generating documentation. It uses [DocBook 5.1](#) + additional markup as its base format and can (currently) generate PDF and XHTML as output. It can also generate component documentation (like the one you're looking at now), as a simple website and PDF. See "Description" on page 3 for an overview.

xtpplib-xdoc is part of the {`$library-name`} library. {`$library-name`} contains software for processing XML, using languages like XSLT and XProc. It consists of several separate components, named {`$library-name`}-*. Everything can be found on GitHub ([{`\$owner-company-github`}](#)). The core component of {`$library-name`} is {`$core-component-name`}. Most components will be accompanied by a GitHub pages documentation site ([TBD](#)).

0.1 Installing xtpplib-xdoc

To install **xtpplib-xdoc** you can do one of the following:

- Clone its GitHub repository ([{`\$git-uri`}](#)) to some appropriate location on disk. The `master` branch will always contain the latest stable version (currently {`$current-release-version`} {`$current-release-date`}).
- Or download the latest release from [{`\$owner-company-github`}/xtpplib-xdoc/releases](#) and unpack it somewhere.

xtpplib-xdoc is dependent on the following other {`$library-name`} components. **Important:** These must be installed in the *same base directory* as **xtpplib-xdoc**:

- [xtpplib-common](#)
- [xtpplib-container](#)

0.2 Running xtpplib-xdoc

xtpplib-xdoc consists of XProc (1.0) pipelines and XSLT (2.0 and 3.0) stylesheets. To run these you'll need:

- To run the XProc pipelines use [XML Calabash](#) Xproc 1.0 processor. This library uses several of its non-standard extensions, another XProc processor is therefore probably not usable.
- To run XSLT stylesheets the [Saxon](#) processor is preferred. In most cases the (open source) HE (Home Edition) version will be sufficient. If not a PE (Professional Edition) license is necessary.
- As an alternative run the software from within the [oXygen XML](#) IDE.

1 Description

1.1 Overview

xtpplib-xdoc's main points:

- **xtpplib-xdoc**'s starting point is documentation written in [DocBook 5.1](#) plus extensions. This source format is called *xdoc*.
- *xdoc* has two kinds of extensions:
 - Parameters, coming from a parameter file, that are expanded. This useful for, for instance, status information, dates/times, standard words and phrases, etc. [\[TBD LINK\]](#)
 - The so-called **[*** Referenced linkend id "chapter-xdoc-transforms" not found (phase: inline)]**. These transformations convert something into DocBook and insert the result back in the main document. This is used for, for instance code documentation generation. There are standard transformations but you can also write your own (in XSLT or XProc).
- An *xdoc* document can be converted into pure DocBook by an XProc pipeline [\[TBD LINK\]](#)
- Additional XProc pipelines [\[TBD LINK\]](#) convert this DocBook into PDF (through XSL-FO) or basic XHTML.
- This mechanism also serves as a basis for documentation generation of complete software components [\[TBD LINK\]](#). The result's are meant to be used as GitHub pages.
- Noteworthy additional functionality is an XProc module [\[TBD LINK\]](#) that converts (simple) Markdown into DocBook.

1.2 The xtpplib-xdoc main toolchain

The following figure illustrates **xtpplib-xdoc**'s main toolchain:

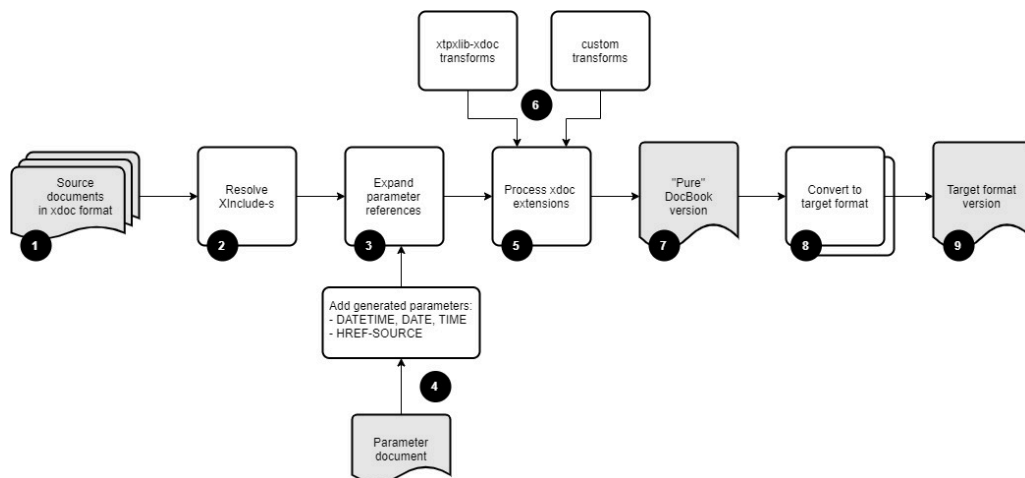


Figure 1-1 - **xtpplib-xdoc**'s main toolchain

1. The **xtpplib-xdoc** module uses a format called *xdoc* as its source format. The basis of *xdoc* is a limited version of [DocBook 5.1](#). On top of this *xdoc* adds several extensions for parameter handling, generating text, etc.
The [DocBook 5.1](#) standard is huge and therefore for practical reasons only partly implemented in this component. A description of what is supported can be found in [\[TBD\]](#).
2. The first processing step in the toolchain performs basic XInclude processing. This means that you can build your document from smaller parts, for instance one file per chapter.
Another application of the XInclude processing is to get the data in for the *xdoc* extensions processing in step 5.
3. The next step is to expand any parameter references in the source document. A parameter is a name/value pair. To expand its value in the document use either `${name}` or `{ $name }` (both mean the same). Parameters are expanded both in text and in attribute values.

4. Parameters come from two sources:

- An (optional) parameter file. This file must be in the parameters format as handled by the parameters module of `{core-component-name}`. For instance:

```
<parameters>
  <parameter name="my-parameter">
    <value>Some value...</value>
  </parameter>
</parameters>
```

This format has several additional features. See [\[TBD\]](#) for more information.

- The toolchain adds several parameter of its own:

Parameter	Description	Example value(s)
DATETIME	The date and time the toolchain executed in YYYY-MM-DD hh:mm:ss format.	2019-11-11 12:12:12
DATE	The date part of the DATETIME parameter.	2019-11-11
TIME	The time part of the DATETIME parameter.	12:12:12
HREF-SOURCE	The main source's filename.	C:/my/path/sourcedoc.xml /my/path/sourcedoc.xml

Table 1-1 - Parameters added by the **xtpxlib-xdoc** toolchain

5. Next the so-called xdoc extensions are processed. These extensions are specific elements in the xdoc (<http://www.xtpxlib.nl/ns/xdoc>) namespace. Currently there are two:

<xdoc:transform>

The **<xdoc:transform>** element triggers a transformation with this element (and its children) as input. Transformations can be either XProc (1.0) or XSLT (2.0 or 3.0). The result of such a transformation must be valid DocBook and is inserted in the document.

<xdoc:dump-parameters>

This extensions dumps all parameters in the document, either as a table or as an XML comment. Useful for debugging the parameter expansion. See [\[TBD\]](#)

6. The transformations triggered by **<xdoc:transform>** can come from two sources:

- Transformations that are built into the **xtpxlib-xdoc** module. These are generic transformations for, for instance, documenting XML structures or code. An overview of these can be found in [\[TBD\]](#).
- Your own transformations. Guidelines on how to write these can found in [\[TBD\]](#).

7. The result of the toolchain so-far is a document in "pure" [DocBook 5.1](#).

8. From this you can transform to some target format. The **xtpxlib-xdoc** component contains transformations to PDF and HTML (see [\[TBD\]](#)). These transformations are rather specific for the **xtpxlib-xdoc** component and might not be directly usable for other use-cases. You can of course copy-and-adapt these or use some other DocBook conversion transformation .

9. The result of all this is a document in the desired target format.

Information about the pipelines that implement this toolchain can be found in [\[TBD\]](#)

1.3 Documentation website generation

[\[TBD\]](#)

2 xdoc extensions

The **xtpxlib-xdoc** component uses (a subset of) [DocBook 5.1](#) as its main vocabulary. Within this there can be so-called *xdoc extensions*. These are specific elements in the `xdoc` (<http://www.xtpxlib.nl/ns/xdoc>) namespace that trigger special actions, like generating code documentation or describe some XML document.

2.1 xdoc extension elements

The `xdoc` format recognizes the following extension elements. The `xdoc:` namespace prefix used here is bound to the <http://www.xtpxlib.nl/ns/xdoc> namespace (`xmlns:xdoc="http://www.xtpxlib.nl/ns/xdoc"`).

2.1.1 Extension element `xdoc:dump-parameters`

This extension element shows the parameters that are currently in use. It is mainly use for debugging purposes.

```
<xdoc:dump-parameters type? = "comment" | "table" />
```

Attribute	#	Type	Description						
type	?	xs:string	Default: comment						
			Whether to dump the parameters as a comment or table.						
			<table><tr><th>Value</th><th>Description</th></tr><tr><td>comment</td><td>Dump the parameters as an XML comment</td></tr><tr><td>table</td><td>Dump the parameters as a table</td></tr></table>	Value	Description	comment	Dump the parameters as an XML comment	table	Dump the parameters as a table
			Value	Description					
comment	Dump the parameters as an XML comment								
table	Dump the parameters as a table								

2.1.2 Extension element `xdoc:transform`

This extension element runs an `xdoc` transformation (either XProc (1.0) or XSLT (2.0 or 3.0)). It is completely replaced by the outcome of the transformation.

There are several built-in transformations. You can also write your own transformations.

```
<xdoc:transform href = xs:anyURI
  (any)? >
  <!-- Usually an <xi:include> element to load the data for the transformation. -->
</xdoc:transform>
```

Attribute	#	Type	Description
href	1	xs:anyURI	Reference to the actual transformation. Relative names are resolved against the location of the source document. This file's extension determines whether an XProc 1.0 (extension: <code>.xpl</code>) or an XSLT (extension: <code>.xsl</code>) is done. A value that starts with <code>\$xdoc</code> is assumed to be an <code>xtpxlib-xdoc</code> built-in transformation (e.g. <code>href="\$xdoc/code-docgen.xpl"</code>). See also "Built-in xdoc transformations" on page 5.
(any)	?		Often transformations specify additional attributes on the <code><xdoc:transform></code> element to parametrize their functionality. Any additional attribute is allowed here.

2.2 Built-in xdoc transformations

2.2.1 XProc 1.0 pipeline: `code-docgen.xpl`

Takes some XML document (XSL, XSD, XProc, ordinary XML, etc.) and tries to generate some documentation out of it. For instance this section itself is generated by the `code-docgen.xpl` transformation on itself.

Typical usage (within an `xdoc` source document):

```
<xdoc:transform href="$xdoc/code-docgen.xpl filecomponents="..." header-level="..." >
  <xi:include href="path/to/document/to/generate/documentation/for"/>
</xdoc:transform>
```

- **@filecomponents:** Determines the display of the filename:
 - When lt 0, no filename is displayed
 - When eq 0, the full filename (with full path) is displayed
 - When gt 0, this number of filename components is displayed. So 1 means filename only, 2 means filename and direct foldername, etc.
- **@header-level:** Determines what kind of DocBook section is created:
 - When le 0, no separate section is created, all titles will be output as bridgehead elements.
 - Otherwise a title with this level is created (e.g. header-level="1" means a sect1 element).

When the format to document has means to add documentation of itself (like XProc (p:documentation) or XML Schema (xs:annotation)), this is used. When there is no such thing (like for XSLT and straight XML), comments starting with a tilde (~) are used.

The descriptions can contain simple Markdown.

The following formats are supported

- XML documents: only the header comment is used.
- XSLT (2.0 and 3.0) stylesheets: document all exported parameters, variables, functions and named templates. SOMething is supposed to be for export if its *not* in the no or local namespace.
- XProc (1.0) pipelines and libraries
- XML Schemas: use the main annotation and document the global elements

Port	Type	Primary?	Description
source	in	yes	The document to generate documentation for, wrapped in an xdoc:transform element.
result	out	yes	The resulting DocBook output.

2.2.2 XProc 1.0 pipeline: xml-description.xpl

Turns an XML element description into the appropriate DocBook. An annotated schema for the element's description markup can be found in xsd/element-description.xml.

Typical usage (within an xdoc source document):

```
<xdoc:transform href="$xdoc/xml-description.xpl">
  <xi:include href="path/to/xml/description.xml"/>
</xdoc:transform>
```

Port	Type	Primary?	Description
source	in	yes	The document containing the XML description, wrapped in an xdoc:transform element.
result	out	yes	The resulting DocBook output, containing the element's description.

2.3 Writing your own xdoc transformations

[TBD]

3 DocBook dialect

The xtpxlib-xdoc component uses [DocBook 5.1](#) as its source and target vocabulary. However, it does not implement the full standard (which is huge!) but only those elements/attributes that were deemed necessary. This document will explain what is in and what's not.

3.1 Supported root elements

Both the <book> and the <article> root element are supported. **[TBD rephrase]** A <book> root results in paged output (for, as the name implies, a book. The <article> root results in something more memo style (like this).

3.2 Document information

Document information: The only document information elements recognized are (any others are ignored):

```
<info>

  <title> ... main title ... </title>
  <subtitle> ... subtitle ...</subtitle>
  <pubdate> ... publication date ... </pubdate>
  <author>
    <personname> ... author name ...</personname>
  </author>

  <orgname> ... organization ... </orgname>

  <mediaobject role="top-logo">
    <!-- Use either role="top-logo" or no role attribute. -->
    <imageobject>
      <imagedata fileref="..." width="...(opt)" height="...(opt)"/>
    </imageobject>
  </mediaobject>

  <mediaobject role="center-page">
    <imageobject>
      <imagedata fileref="..." width="...(opt)" height="...(opt)"/>
    </imageobject>
  </mediaobject>

</info>
```

All elements are optional.

3.3 Chapter/Section structure

- For books, <preface>, <chapter>, <appendix> and <sect1> to <sect3> are recognized and handled.
- In articles only <sect1> to <sect3> are allowed.

3.4 Block constructions

the following block level constructions are recognized and handled:

- **Paragraphs:** Normal <para> elements recognize the following role attribute values (multiple, whitespace separated, values allowed):

@role value	Description
break, smallbreak	Inserts an empty line, either full or small height. The contents of the <para> element is ignored.
break-before break-after	Adds extra whitespace before or after the paragraph
header keep-with-next	Keeps this paragraph with the next one together on a page.

@role value	Description
keep-with-previous	Keeps this paragraph with the previous one together on a page.

Table 3-1

- **Lists:** Both `<itemizedlist>` and `<orderedlist>` are allowed.
- **Tables:** Both `<table>` and `<informaltable>` are allowed. An example of a formal table above. An informal table below.

Example of
an informal table

Add `role="nonumber"` to a table to stop it from getting a number:

Blurp	Blorb
Example	of
an	unnumbered table

Unnumbered table

Tables are notoriously difficult in that FOP cannot compute column widths automatically. To amend this (a little bit) add `colspec/@colwidth` information. There is also a mechanism for columns with code (set in a fixed-width font), see "Fixed-width column mechanism" on page 11.

- **Program listings:** For program listings use the `<programlisting>` element
The easiest way to handle this turned out to put longer program listings in external files and use an `<xi:include parse="text">` construction:

```
<programlisting><xi:include href="ref" parse="text" /></programlisting>
```

 Or use a `<![CDATA[` construction around the piece of code.
- **Figures:** Both `<figure>` and `<informalfigure>` are allowed. Width and height can be set on the image data.



Figure 3-1 - An example of a figure... (this in fixed width)

Add `role="nonumber"` to a figure to stop it from getting a number.

- **Bridgeheads:** The `<bridgehead>` element inserts a bridgehead paragraph (bold, underlined and with an empty line before):

This is a bridgehead...

- **Simple lists:** The `<simplelist>` element inserts a simple list:

An entry
Another entry...

- **Variable lists:** The `<variablelist>` element inserts a variable list list (also very useful for explaining terms, definitions, etc.):

The first entry

The explanation of the first entry!

The second entry

The explanation of the second entry!

- **Notes and warnings:** The `<note>` element inserts a note and `<warning>` a warning:

NOTE:

This is a note! Nulla ac ex urna. Ut auctor odio quis nulla porta bibendum. Proin hendrerit molestie velit sit amet tristique. Vivamus laoreet ligula leo, vitae placerat ipsum porta sed. Morbi blandit ex mauris, eu volutpat tortor mattis eu. Nam et molestie mi. Aliquam erat volutpat. Aenean a imperdiet lectus. Phasellus condimentum dignissim laoreet.

WARNING:

This is a warning! Nulla ac ex urna. Ut auctor odio quis nulla porta bibendum. Proin hendrerit molestie velit sit amet tristique. Vivamus laoreet ligula leo, vitae placerat ipsum porta sed. Morbi blandit ex mauris, eu volutpat tortor mattis eu. Nam et molestie mi. Aliquam erat volutpat. Aenean a imperdiet lectus. Phasellus condimentum dignissim laoreet.

If you add a `<title>` element, the standard title (NOTE/WARNING) will be replaced by its contents.

- **Sidebar:** The `<sidebar>` element inserts sidebar section:

Title of the sidebar

Contents of the sidebar. Nulla ac ex urna. Ut auctor odio quis nulla porta bibendum. Proin hendrerit molestie velit sit amet tristique. Vivamus laoreet ligula leo, vitae placerat ipsum porta sed. Morbi blandit ex mauris, eu volutpat tortor mattis eu. Nam et molestie mi. Aliquam erat volutpat. Aenean a imperdiet lectus. Phasellus condimentum dignissim laoreet.



- **Examples:** The `<example>` element inserts an example:


Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus elementum diam nec nunc elementum, eget dapibus dui malesuada. Aenean facilisis consequat odio, vitae euismod eros tempor nec. Vestibulum cursus tortor tortor, semper euismod sapien sagittis et.

Example 3-1 - Example of an example

Add `role="nonumber"` to an example to stop it from getting a number.

3.5 Inline elements

the following inline elements are recognized and handled:

- **`<emphasis>`:** Sets *emphasis*.
Use `role="bold"` or `role="underline"` to set a specific type of emphasis.
- **`<literal>` or `<code>`:** Creates a piece of literal, mono-spaced text.
- **`<link>`:** Outputs some link (e.g. a web address). This is either the contents of the element or, if not available, the contents of the `xlink:href` attribute.
Like [this](#) or like this <http://www.xatapult.nl>.
- **`<inlinemediaobject>`:** Inserts an inline image , like this.
- **`<citation>`:** Inserts a citation between square brackets like this: [CITATION].

- **<command>**: Use to indicate an executable program or a user provided command, like this: *git checkout origin*
- **<email>**: Use to indicate an email address, like this: *info@xatapult.com*
- **<filename>**: Use to indicate a filename, like this: *blabla.xml*
- **<replaceable>**: Use to indicate text to be replaced with user or context supplied values, like this: *add your own stuff here*
- **<keycap>**: Use to indicate a keyboard physical key, like this: Return
- **<superscript>**, **<subscript>**: For super- and subscripts, like this: XX^{super} YY_{sub}
- **<userinput>**: Use to indicate data entered by the user, like this: **data entered here**
- **<tag>**: Indicates an object from the XML vocabulary. The `class` attribute signifies what:

@class value	Result example(s)
attribute	@attribute @class
attvalue	"attribute value" "some value for an attribute"
emptytag	<element/> <docbook/>
endtag	</element> </docbook>
pi	<?processing-instruction x="y"?>
comment	<!-- Some comment line... -->
Anything else defaults to element	<element> <docbook>

Table 3-3

- **<xref>**: Inserts a cross-reference to the id referenced by @linkend
 - Use `role="page-number-only"` to get just a page number.
 - Use `role="simple"` to always get: page #
 - Use `role="text"` to only get the (unquoted) text only in cases where a "..." on page ... would normally appear.
 - Use `role="capitalize"` to force the reference string (for chapters/appendices/pages/figures/tables/...) to start with an upper-case character (so you can be sure a sentence that starts with an `<xref>` always starts with a capital).

Otherwise it depends on what is pointed to:

Target	Result/Examples
To anything that holds an <code>xreflabel</code> attribute	"paragraph with xreflabel attribute" on page 7
To a chapter or appendix	chapter # or appendix #
To a section	"Document information" on page 7
To a table (with a number), like this one	table 3-4
To a figure (with a number)	figure 3-1
To an example (with a number)	example 3-1

Target	Result/Examples
To anything else	First paragraph: page 7 Unnumbered table: page 8

Table 3-4 - Examples of `<xref>` usage

3.6 Other constructs

- **To-be-done marker:** Start a to-be-done marker with `[TBD` and end it with `]`. For instance: `[TBD this needs to be done...]`

3.7 Fixed-width column mechanism

FOP (in the current version, 3Q19) cannot compute the column-widths automatically. It divides the space and you can set a fixed column-width (with `colspec/@colwidth`). For the case that a column contains code stuff (text in a fixed-width font) *and* you want the column-width to be dependent on the text in such a column, there is a (unfortunately a bit complicated) mechanism for this.

The fixed-width column mechanism consists of two parts:

Dynamically compute the column width

This part is optional.

Add a `role` attribute to the `<colspec>` element with, as one of the roles, `code-width-cm:min-max`, where `min` and `max` are (positive) doubles. For instance `<colspec role="code-width-cm:1.2-4">`. `min` and `max` are the minimum and maximum column-widths, expressed in cm.

The PDF conversion will now look in all the contents of this particular column for entries `<code role="code-width-limited">`. Based on the length of these entries it computes an optimal column-width, but always between `min` and `max`.

Output code width-limited

If a table entry contains contents in a `<code role="code-width-limited">` element, it tries to make it fit within the available column-width. If necessary the line is split to prevent overflowing of table cell contents.

This is (currently) not completely fool-proof: if the contents contains whitespace or hyphens, it is assumed to line-break correctly by itself. That, of course, does not guarantee correct results. So it may need a little experimenting before things look right.

A column that contains `<code role="code-width-limited">` contents *must* have a column width set *in cm* (either directly with `<colspec colwidth="...cm">` or by the dynamic mechanism described above).