

xtpxlib-xdoc

An xtpxlib component for generating documentation

0 Table of Contents

0 Documentation generation with xtpplib-xdoc	2
1 Description	3
1.1 The main toolchain	3
2 Instructions	5
2.1 Getting started	5
2.2 Validation	5
2.3 Parameter substitution	5
2.4 xdoc transforms	6
3 xdoc transforms	8
3.1 Running an xdoc transform	8
3.2 Built-in xdoc transformations	8
3.2.1 XProc (1.0) pipeline: code-docgen-dir.xpl	8
3.2.2 XProc (1.0) pipeline: code-docgen.xpl	9
3.2.3 XProc (1.0) pipeline: include-docbook.xpl	10
3.2.4 XProc (1.0) pipeline: xml-description.xpl	10
3.3 Writing your own xdoc transformations	10
4 XProc Pipelines	12
4.1 XProc (1.0) pipeline: docbook-to-pdf.xpl	12
4.2 XProc (1.0) pipeline: docbook-to-xhtml.xpl	13
4.3 XProc (1.0) pipeline: xdoc-to-docbook.xpl	13
4.4 XProc (1.0) pipeline: xdoc-to-pdf.xpl	13
4.5 XProc (1.0) pipeline: xdoc-to-xhtml.xpl	14
5 XProc Libraries	15
5.1 XProc (1.0) library: xtpplib-xdoc.mod.xpl	15
5.1.1 Step: xdoc:markdown-to-docbook	15
6 DocBook dialect	16
6.1 Supported root elements	16
6.2 Document information	16
6.3 Chapter/Section structure	16
6.4 Block constructions	16
6.5 Inline elements	18
6.6 Other constructs	20
6.7 Fixed-width column mechanism	20

0 Documentation generation with xtpplib-xdoc



xtpplib library - component **xtpplib-xdoc** - **v1.1.1** (2020-10-15)
Xatapult Content Engineering - <http://www.xatapult.com> - +31 6 53260792
Erik Siegel - erik@xatapult.com

xtpplib-xdoc is part of the **xtpplib** library. **xtpplib** contains software for processing XML, using languages like XSLT and XProc. It consists of several separate components, all named **xtpplib-***. Everything can be found on GitHub (<https://github.com/xatapult>).

The **xtpplib-xdoc** component contains an XProc (1.0) based DocBook publication toolchain.

- Starting point is some narrative written in DocBook, with the following extensions:
 - Parameter references that are expanded (for dates, times, phrases, names, etc.)
 - Special elements that trigger conversions. These conversions can insert generated DocBook into the source. For instance complex tables, documentation, etc.
- The resulting "pure" DocBook can be used for further processing.
- The component contains specific pipelines for converting the DocBook to PDF and XHTML

Installation and usage information can be found on **xtpplib**'s main website <https://www.xtpplib.org>.

Technical information:

Component documentation: <https://xdoc.xtpplib.org>

License: GNU GENERAL PUBLIC LICENSE - Version 3, 29 June 2007

Git URI: [git@github.com:xatapult/xtpplib-xdoc.git](https://github.com:xatapult/xtpplib-xdoc.git)

Git site: <https://github.com/xatapult/xtpplib-xdoc>

This component depends on:

- [xtpplib-container](#) (Support for XML containers (multiple files wrapped into one))
- [xtpplib-common](#) (Common component: Shared libraries and IDE support)

1 Description

Have you ever struggled with producing technical documentation for your software, content model or anything else? Big chance that you have had to deal with repeating constructs: Explaining XML elements and attributes, documenting functions, procedures and variables, etc. The same constructs over and over again, usually with complex tables, little pieces of program listings or other things that are difficult to keep consistent and maintain. The `xtpplib` component `xtpplib-xdoc` tries to alleviate this problem. `xtpplib-xdoc`'s starting point is *narrative* documentation written in [DocBook 5.1](#). On top of this it adds a number of extensions. This source format, DocBook + extensions, is called *xdoc*.

The `xtpplib-xdoc` XProc (1.0) pipelines turns the *xdoc* format into "pure" DocBook. From there it can be converted into PDF or HTML using standard DocBook technology. The `xtpplib-xdoc` component itself also contains conversions into PDF (through XSL-FO) and HTML. These work out of the box but, especially the PDF one, uses a layout that might not be what you want or need. But since the source is available you can tweak it to your heart's desire.

`xtpplib-xdoc` currently allows two types of extensions on top of DocBook:

Parameter expansion

Parameters, coming from some parameter source, are expanded. This useful for, for instance, status information, dates/times, standard words and phrases, etc. This uses the [parameter mechanism](#) as introduced in `xtpplib`'s [common component](#).

Transforms

The so-called [xdoc transforms](#) convert something, usually some piece of XML, into DocBook and insert the result back in the main document. This is extremely useful for *consistent* and *repeating* documentation generation.

Curious to see it in action. Want to know more? Checkout the "Instructions" on page 5 section.

1.1 The main toolchain

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

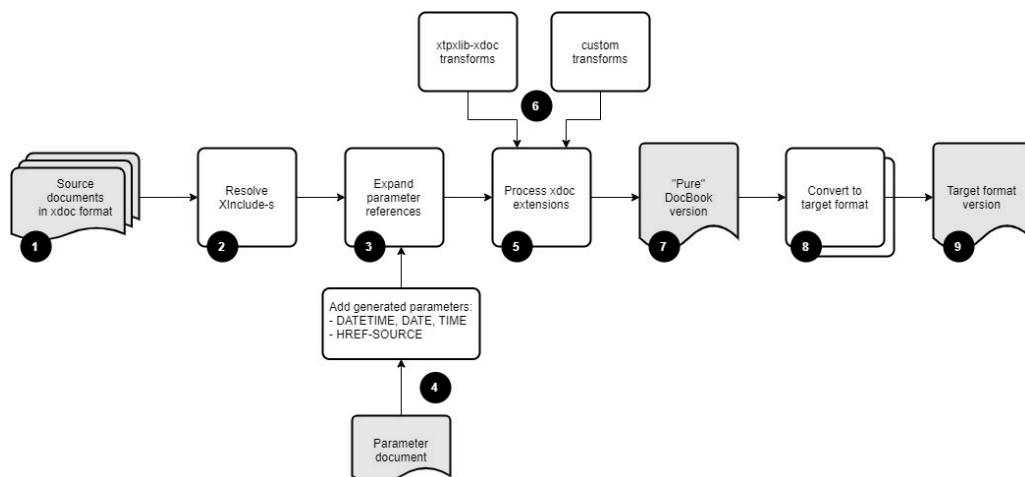


Figure 1-1 - `xtpplib-xdoc`'s main toolchain

1. The `xtpplib-xdoc` module uses a format called *xdoc* as its source format. The basis of *xdoc* is [DocBook 5.1](#). On top of this *xdoc* adds extensions for parameter handling and code/markup/text generation.
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 document per chapter.
Another application of the XInclude processing is to get the data in for the *xdoc* transform 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 document. This document must use the format as handled by the [parameter mechanism](#) of xtpxlib's [common component](#).
- The toolchain automatically creates some parameters.

See [here](#) for details and usage instructions

5. Next the so-called *xdoc transforms* are processed. A transform consists of an `<xdoc:transform>` element (the namespace prefix `xdoc:` must be bound to `http://www.xtpxlib.nl/ns/xdoc`). An XSLT stylesheet or XProc (1.0) pipeline is triggered that gets this `<xdoc:transform>` element (with all attributes and child elements) as input and results in the injection of generated DocBook.

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

- Transformations that are built into the **xtpxlib-xdoc** component. These are generic transformations for, for instance, documenting XML structures or generating code documentation. An overview of these can be found in "Built-in xdoc transformations" on page 8.
- Your own transformations. Guidelines on how to write these can found in "Writing your own xdoc transformations" on page 10.

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 both PDF and HTML (see the [docbook-to-pdf](#) and [docbook-to-xhtml](#) pipelines). These transformations can only handle a [subset](#) of the full DocBook standard. The result will be rather specific for the **xtpxlib-xdoc** component and might not be directly usable for other use-cases. To amend this you can copy-and-adapt these transformations or use some other DocBook conversion.

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

Information about the pipelines that implement this toolchain can be found [here](#).

2 Instructions

2.1 Getting started

The `template/` sub-directory of `xtpxlib-xdoc` contains several template files that can be used as a starting point. These templates also declare the necessary namespace `http://www.xtpxlib.nl/ns/xdoc`, bound to the prefix `xdoc:`.

Use one of the XProc (1.0) [processing pipelines](#) to process an `xdoc` source into DocBook, PDF or HTML. For instance `xdoc-to-docbook` will turn your `xdoc` source into "pure" DocBook.

2.2 Validation

The `xtpxlib-xdoc` component contains an enhanced DocBook NVDL schema, `xsd/docbook/docbook.nvdl`, that allows the `xdoc` extensions.

The template files in the `template/` sub-directory reference this schema. Don't forget to change this reference and keep it valid if you copy such a template to a directory of your own!

2.3 Parameter substitution

The `xdoc` framework performs parameter substitution. `${parameter-name}` and `{$parameter-name}` (both mean the same) are substituted with the parameter's value (if it exists). Substitution takes place in attribute and text values. To stop such a `${...}` or `{$...}` construction from being substituted, *double* the opening curly brace (`{ { }`).

The `xdoc` toolchain automatically creates a number of parameters:

Parameter	Description	Example value(s)
DATETIME	The date and time the toolchain executed in YYYY-MM-DD hh:mm:ss format.	*** Unhandled element encountered: <code xml:id="d2225e572"> (phase: block)
DATE	The date part of the DATETIME parameter.	*** Unhandled element encountered: <code xml:id="d2225e596"> (phase: block)
TIME	The time part of the DATETIME parameter.	*** Unhandled element encountered: <code xml:id="d2225e620"> (phase: block)
HREF-SOURCE	The main source's filename.	C:/my/path/sourcedoc.xml /my/path/sourcedoc.xml

Table 2-1 - Parameters added by the `xtpxlib-xdoc` toolchain

To specify your own parameters, create an XML document that looks like this:

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

The parameter XML format has several additional features, like filtering and grouping values. It's also namespace independent and might be embedded in a bigger document. See the format's [description](#) for more information.

A reference to such a parameter document must be passed as option `href-parameters` to one of the [processing pipelines](#).

To see which parameters are available in your `xdoc` pipelines, add the following to your document:

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

The documentation you're looking at is also produced with the `xdoc` mechanism. The result of a parameter dump during its build process is:

Parameter	Value
DATE	2020-10-15
DATETIME	2020-10-15 14:01:41
HREF-SOURCE	C:/Data/Erik/work/xatapult/xtpxlib-xdoc/doc/source/xtpxlib-xdoc-chapter-instructions.xml
TIME	14:01:41
active-components	xtpxlib-common xtpxlib-container xtpxlib-xoffice xtpxlib-xdoc
author-email-address	erik@xatapult.com
author-name	Erik Siegel
component-current-release-date	2020-10-15
component-current-release-version	1.1.1
component-display-name	xtpxlib-xdoc
component-documentation-uri	https://xdoc.xtpxlib.org
component-git-site-uri	https://github.com/xatapult/xtpxlib-xdoc
component-git-uri	git@github.com:xatapult/xtpxlib-xdoc.git
component-name	xtpxlib-xdoc
component-title	DocBook publication toolchain
library-name	Xatapult XML Library
license	GNU GENERAL PUBLIC LICENSE - Version 3, 29 June 2007
owner-company-git-site-uri	https://github.com/xatapult
owner-company-name	Xatapult Content Engineering
owner-company-phone	+31 6 53260792
owner-company-website	http://www.xatapult.com

Table 2-2 - Parameters

You can also specify `type="comment"`. As the name implies, the parameters will be added as an XML comment so you'll have to dive into the produced DocBook to see them.

2.4 xdoc transforms

An xdoc transform is an XSLT stylesheet or XProc (1.0) pipeline that is triggered from your source document and inserts generated DocBook contents. There are several of these transforms built into the xtpxlib-xdoc component but its also easy to write one of your own. Detailed information can be found [here](#).

As an example: xtpxlib-xdoc contains transforms to extract documentation from XML documents and programs. Among others, the documentation sections for the [pipelines](#) and [libraries](#) are generated with this. Now assume you want to insert the documentation of the [xdoc-to-docbook](#) pipeline somewhere in a document of your own. You could do this by adding a reference to the [xdoc-to-docbook](#) transform to your xdoc source:

```
<xdoc:transform href="$xdoc/code-docgen.xpl" filecomponents="2">
  <xi:include href="../../../xpl/xdoc-to-docbook.xpl"/>
</xdoc:transform>
```

The result will be:

XProc (1.0) pipeline: xdoc-to-docbook.xpl

File: xpl/xdoc-to-docbook.xpl

Type: xdoc:xdoc-to-docbook

Pipeline that transforms a DocBook source containing xdoc extensions into "pure" DocBook format.

Port	Type	Primary?	Description
source	in	yes	The DocBook source with <code>xdoc</code> extensions
result	out	yes	The resulting DocBook

Option	Rq?	Default	Description
href-parameters		()	Optional reference to a document with parameter settings. See here for details.
parameter-filters		()	Optional filter settings for processing the parameters. Format: <code>name=value name=value ...</code>

3 xdoc transforms

An `xdoc` transform is an XSLT stylesheet or XProc (1.0) pipeline that is triggered from your source document and inserts generated DocBook contents. There are several of these [transforms built into the xtpxlib-xdoc component](#) but its also easy to [write one of your own](#).

3.1 Running an xdoc transform

The `<xdoc:transform>` 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.

```
<xdoc:transform href = xs:anyURI
  (any)? >
  <!-- (Optional) XML to transform and/
  or an <xi:include> element to load this from an external source. -->
</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 8.
(any)	?		Often transformations specify additional attributes on the <code><xdoc:transform></code> element to parametrize their functionality. Any additional attribute is allowed here.

3.2 Built-in xdoc transformations

The `xtpxlib-xdoc` has several transforms built in. You can easily reference these by prefixing their name with `$xdoc`, for instance `$xdoc/code-docgen.xpl`.

Module/Pipeline	Description
code-docgen-dir.xpl	Runs the \$xdoc/code-docgen.xpl transform over multiple files in a directory.
code-docgen.xpl	Takes an XML document (XSL, XSD, XProc, ordinary XML) and generates documentation out of it.
include-docbook.xpl	Takes an XML document and unwraps the root element from it. It then copies all the children to the output, with the exception of any <code>db:info</code> elements.
xml-description.xpl	Takes a document that <i>describes</i> an XML document, using special markup for this, and turns this into DocBook.

Table 3-2 - Module overview

3.2.1 XProc (1.0) pipeline: code-docgen-dir.xpl

File: `transforms/code-docgen-dir.xpl`

Runs the [\\$xdoc/code-docgen.xpl](#) transform over multiple files in a directory.

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

```
<xdoc:transform href="$xdoc/code-docgen-dir.xpl"
  dir="..."
  depth="..."
  filter="..."
  toc-only="..."
  id-suffix="..." >
```

- **@dir:** Directory to process
- **@depth:** (integer, default -1) The depth in traversing the directory tree.
 - When `le 0`, `@dir` and all its subdirectories are processed.
 - When `eq 1`, only `@dir` is processed.
 - When `gt 1`, the sub-directories up to this depth are processed.
- **@filter:** optional regexp filter (e.g. get only XProc files with `filter="\.xpl$"`)
- **@toc-only:** (boolean, default `false`) Whether to produce a ToC table only.
- **@id-suffix:** Optional suffix for creating an id based on the filename.

All (other) attributes are passed to `code-docgen.xpl`.

Port	Type	Primary?	Description
source	in	yes	The triggering <code>xdoc:transform</code> element.
result	out	yes	The resulting DocBook output.

3.2.2 XProc (1.0) pipeline: code-docgen.xpl

File: `transforms/code-docgen.xpl`

Type: `xdoc:code-docgen`

Takes an XML document (XSL, XSD, XProc, ordinary XML) and generates documentation out of it.

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

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

- **@filecomponents:** (integer, default 0) 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:** (integer, default 0) 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).
- **@add-table-titles:** (boolean, default `false`) Whether to add titles to generated tables.
- **@sublevels:** (boolean, default `true`) If `true` only the main section will be a "real" section. All sublevels will become bridgeheads.
- **@id:** Optional identifier of this section. If absent the id will become the document's filename, optionally suffixed with `@id-suffix`.
- **@id-suffix:** Optional suffix for creating an id based on the filename.

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

All descriptions and documentation sections 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 pipelines and libraries
- XML Schemas: Uses the global annotation and lists the global elements using their annotations.

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

3.2.3 XProc (1.0) pipeline: include-docbook.xpl

File: `transforms/include-docbook.xpl`

Takes an XML document and unwraps the root element from it. It then copies all the children to the output, with the exception of any `db:info` elements.

It is the responsibility of the author to make sure that everything that results is in the DocBook (<http://docbook.org/ns/docbook>) namespace!

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

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

Port	Type	Primary?	Description
source	in	yes	The document containing the parts to include, wrapped in an <code>xdoc:transform</code> element.
result	out	yes	The resulting DocBook output.

3.2.4 XProc (1.0) pipeline: xml-description.xpl

File: `transforms/xml-description.xpl`

Takes a document that *describes* an XML document, using special markup for this, and turns this into DocBook.

A schema for this markup format 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 <code>xdoc:transform</code> element.
result	out	yes	The resulting DocBook output.

3.3 Writing your own xdoc transformations

- To add an `xdoc` transform of your own you need to write an XSLT stylesheet or an XProc pipeline.
- Such a stylesheet or transformation gets the **full** `<xdoc:transform>` element as its input. It can inspect the `<xdoc:transform>` root element itself for its attributes.
- The output of the stylesheet/pipeline must be the resulting (valid!) DocBook.
- If the resulting DocBook contains multiple elements you can wrap them all in an `<xdoc:GROUP>` element to make the result a single well-formed document. The `<xdoc:GROUP>` wrapper is removed later on by the `xdoc` processing.
- You must reference your stylesheet or pipeline using `<xdoc:transform>`'s `href` attribute.

Here is a simple example of something that is actually quite useful. Tables in DocBook are complex things. When you to have format the same kind of data over and over again in a table, it becomes very boring and hard to keep consistent and maintainable. Using `xdoc` transforms you can automate the data to DocBook part.

Assume we have, all over the document, weather data, that comes in this source format:

```
<weather-data>
  <data city="Amsterdam" temp="20"/>
  <data city="Stavanger" temp="-5"/>
</weather-data>
```

You want to show this as tables. The following XSLT stylesheet (called `transform-weather-data.xsl`) will do the xdoc transform trick. As its input it gets the weather data wrapped in the `<xdoc:transform>` element (see below).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="3.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xdoc="http://www.xtpxlib.nl/ns/xdoc"
  xmlns="http://docbook.org/ns/docbook">

  <xsl:template match="/">
    <table>
      <title>Example weather data</title>
      <tgroup cols="2">
        <colspec colwidth="4cm"/>
        <colspec/>
        <thead>
          <row>
            <entry>
              <para>City</para>
            </entry>
            <entry>
              <para>Temperature (C)</para>
            </entry>
          </row>
        </thead>
        <tbody>
          <xsl:for-each select="/xdoc:transform/weather-data/data">
            <row>
              <entry>
                <para>
                  <xsl:value-of select="@city"/>
                </para>
              </entry>
              <entry>
                <para><xsl:value-of select="@temp"/></para>
              </entry>
            </row>
          </xsl:for-each>
        </tbody>
      </tgroup>
    </table>
  </xsl:template>

</xsl:stylesheet>
```

In your document you add:

```
<xdoc:transform href="path/to/transform-weather-data.xsl">
  <xi:include href="path/to/your/weather/data.xml"/>
</xdoc:transform>
```

And the result for this example is:

City	Temperature (C)
Amsterdam	20
Stavanger	-5

Table 3-7 - Example weather data

4 XProc Pipelines

The xtpxlib-xdoc component contains the following XProc (1.0) pipelines:

Module/Pipeline	Description
docbook-to-pdf.xpl	This turns Docbook (5.1) into a PDF using FOP.
docbook-to-xhtml.xpl	This turns Docbook (5.1) into XHTML.
xdoc-to-docbook.xpl	Pipeline that transforms a DocBook source containing xdoc extensions into "pure" DocBook format.
xdoc-to-pdf.xpl	Convenience pipeline: Combines the xdoc-to-docbook and the docbook-to-pdf steps in one.
xdoc-to-xhtml.xpl	Convenience pipeline: Combines the xdoc-to-docbook and the docbook-to-xhtml steps in one.

Table 4-1 - Module overview

4.1 XProc (1.0) pipeline: docbook-to-pdf.xpl

File: `xpl/docbook-to-pdf.xpl`

Type: `xdoc:docbook-to-pdf`

This turns Docbook (5.1) into a PDF using FOP.

All necessary xdoc pre-processing (usually with [xdoc-to-docbook.xpl](#)) must have been done.

It will only convert a [partial DocBook tagset](#).

If you don't use [xdoc-to-docbook.xpl](#), you have to make sure to get correct `xml:base` attributes in, so the pipeline can find includes and images. The following XProc (1.0) code takes care of that:

```
<p:xinclude>
  <p:with-option name="fixup-xml-base" select="true()" />
</p:xinclude>
<p:add-attribute attribute-name="xml:base" match="/*">
  <p:with-option name="attribute-value" select="/reference/to/source/document.xml"/>
</p:add-attribute>
```

Port	Type	Primary?	Description
source	in	yes	The docbook source document, fully expanded (with appropriate <code>xml:base</code> attributes)
result	out	yes	The resulting XSL-FO (that was transformed into the PDF).

Option	Rq?	Default	Description
chapter-id		' '	Specific chapter identifier to output.
fop-config		<code>resolve-uri('../..//xtpxlib-common/data/fop-default-config.xml', static-base-uri())</code>	Reference to the FOP configuration file
global-resources-directory		()	Images that are tagged as <code>role="global"</code> are searched here (discarding any directory information in the image's URI)
href-pdf	yes		The name of the resulting PDF file (must have <code>file://</code> in front).
href-xsl-fo		()	If set, writes the intermediate XSL-FO to this href (so you can inspect it when things go wrong in FOP)
main-font-size		10	Main font size as an integer. Usual values somewhere between 8 and 10.
output-type		'a4'	Output type. Use either <code>a4</code> or <code>sb</code> (= standard book size)
preliminary-version		<code>false()</code>	If <code>true</code> , adds a preliminary version marker and output any <code>db:remark</code> elements. If <code>false</code> , output of <code>db:remark</code> elements will be suppressed.

4.2 XProc (1.0) pipeline: docbook-to-xhtml.xpl

File: `xpl/docbook-to-xhtml.xpl`

Type: `xdoc:docbook-to-xhtml`

This turns Docbook (5.1) into XHTML.

All necessary `xdoc` pre-processing (usually with [xdoc-to-docbook.xpl](#)) must have been done.

It will only convert a [partial DocBook tagset](#).

The resulting XHTML will not be directly useable, post-processing the result into a complete and correct HTML page is necessary. The result of this pipeline consists of nested `div` elements. There is no surrounding `html` or `body` element.

Port	Type	Primary?	Description
source	in	yes	The docbook source document.
result	out	yes	The resulting XHTML

4.3 XProc (1.0) pipeline: xdoc-to-docbook.xpl

File: `xpl/xdoc-to-docbook.xpl`

Type: `xdoc:xdoc-to-docbook`

Pipeline that transforms a DocBook source containing `xdoc` extensions into "pure" DocBook format.

Port	Type	Primary?	Description
source	in	yes	The DocBook source with <code>xdoc</code> extensions
result	out	yes	The resulting DocBook

Option	Rq?	Default	Description
<code>href-parameters</code>		<code>()</code>	Optional reference to a document with parameter settings. See here for details.
<code>parameter-filters</code>		<code>()</code>	Optional filter settings for processing the parameters. Format: <code>name=value name=value ...</code>

4.4 XProc (1.0) pipeline: xdoc-to-pdf.xpl

File: `xpl/xdoc-to-pdf.xpl`

Type: `xdoc:xdoc-to-pdf`

Convenience pipeline: Combines the [xdoc-to-docbook](#) and the [docbook-to-pdf](#) steps in one.

Port	Type	Primary?	Description
source	in	yes	The DocBook source with <code>xdoc</code> extensions
result	out	yes	Some XML report about the conversion

Option	Rq?	Default	Description
<code>chapter-id</code>		<code>' '</code>	Specific chapter identifier to output.
<code>fop-config</code>		<code>resolve-uri('../..//xtpxlib-common/data/fop-default-config.xml', static-base-uri())</code>	Reference to the FOP configuration file
<code>global-resources-directory</code>		<code>()</code>	Images that are tagged as <code>role="global"</code> are searched here (discarding any directory information in the image's URI)
<code>href-docbook</code>		<code>()</code>	If set, writes the intermediate full DocBook to this href (so you can inspect it when things go wrong)
<code>href-parameters</code>		<code>()</code>	Optional reference to a document with parameter settings. See here for details.
<code>href-pdf</code>	yes		The name of the resulting PDF file
<code>href-xsl-fo</code>		<code>()</code>	If set, writes the intermediate XSL-FO to this href (so you can inspect it when things go wrong in FOP)

Option	Rq?	Default	Description
main-font-size		10	Main font size as an integer. Usual values somewhere between 8 and 10.
output-type		'a4'	Output type. Use either a4 or sb (= standard book size)
parameter-filters		()	Optional filter settings for processing the parameters. Format: name=value name=value ...
preliminary-version		false()	If true, adds a preliminary version marker and output any db:remark elements. If false, output of db:remark elements will be suppressed.

4.5 XProc (1.0) pipeline: xdoc-to-xhtml.xpl

File: xpl/xdoc-to-xhtml.xpl

Type: xdoc:xdoc-to-xhtml

Convenience pipeline: Combines the [xdoc-to-docbook](#) and the [docbook-to-xhtml](#) steps in one.

Port	Type	Primary?	Description
source	in	yes	The DocBook source with xdoc extensions
result	out	yes	The resulting XHTML

Option	Rq?	Default	Description
href-parameters		()	Optional reference to a document with parameter settings. See here for details.
parameter-filters		()	Optional filter settings for processing the parameters. Format: name=value name=value ...

5 XProc Libraries

The xtpxlib-xdoc component contains the following XProc (1.0) library module:

Module/Pipeline	Description
xtpxlib-xdoc.mod.xpl	Library with support pipelines for xdoc and related conversions.

Table 5-1 - Module overview

5.1 XProc (1.0) library: xtpxlib-xdoc.mod.xpl

File: `xplmod/xtpxlib-xdoc.mod/xtpxlib-xdoc.mod.xpl`

Library with support pipelines for xdoc and related conversions.

Prefix	Namespace URI
xdoc	<code>http://www.xtpxlib.nl/ns/xdoc</code>

5.1.1 Step: xdoc:markdown-to-docbook

Converts the contents of `xdoc:MARKDOWN` elements into DocBook.

This pipeline checks the incoming XML for `xdoc:MARKDOWN` elements. The contents of these elements is assumed to contain Markdown. The pipeline tries to convert this into DocBook. The `xdoc:MARKDOWN` element is removed/unwrapped.

The following rules apply:

- The contents of an `xdoc:MARKDOWN` element is stringified (so any child elements are lost).
- The resulting text can be indented, using space characters only (no tabs!). The non-empty line with the *minimum* indent is assumed to be its left margin.
- Only simple Markdown is supported. Specifically:
 - Inline markup for emphasis, bold, code, etc.
 - Links. A link target starting with a % is handled as an *internal* link (the `@xml:id` of something in the encompassing DocBook).
 - Code blocks (using three consecutive back-ticks)
 - Headers (these are all converted into the same DocBook `bridgehead` elements)
- Specifically not supported (yet?) are tables.

If you add an `header-only="true"` attribute to the `xdoc:MARKDOWN` element, only the first paragraph will be output.

Port	Type	Primary?	Description
source	in	yes	Any XML that might contain <code>xdoc:MARKDOWN</code> elements for conversion.
result	out	yes	The same XML but with the <code>xdoc:MARKDOWN</code> element's contents converted into DocBook.

6 DocBook dialect

The xtpxlib-xdoc component uses [DocBook 5.1](#) as its source and target vocabulary. However, *for generating output* (see the [docbook-to-pdf](#) and [docbook-to-xhtml](#) pipelines) 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.

6.1 Supported root elements

Both the <book> and the <article> root element are supported.

For [docbook-to-pdf](#) conversion: A <book> root results in a book-like output (with a front page, ToC, etc.). The <article> root results in something more memo style.

6.2 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.

6.3 Chapter/Section structure

- For books, <preface>, <chapter>, <appendix> and <sect1> to <sect9> are recognized and handled. Anything above <sect3> will not be numbered.
- In articles only <sect1> to <sect9> are allowed.

6.4 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

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

Table 6-1

- **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="nonnumber"` to a table to stop it from getting a number:

Blurp	Blorb				
Example	of				
an	unnumbered table				
An <code><entrytbl></code>	<table> <tr> <td>1</td><td>2</td></tr> <tr> <td>3</td><td>4</td></tr> </table>	1	2	3	4
1	2				
3	4				

Unnumbered table

A table can have multiple `<tgroup>` elements.

You can add a nested table in a cell using the `<entrytbl>` element (currently for PDF only).

`<spanspec>` elements are ignored.

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 20.

- **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.
- For PDF generation it is possible to use so-called *callouts* to draw attention to parts of a program listing. These callouts can become links (both ways) using the right markup. For example:

```
xquery version "3.0" encoding "UTF-8";
module namespace x101log = "http://www.exist-db.org/book/namespaces/exist101"; ❶
declare function x101log:add-log-message($message as xs:string)
as empty-sequence() ❷
...
};
```

❶ The module namespace definition at the top defines ...

❷ We declare a function that returns `empty-sequence()`?

- **Figures:** Both `<figure>` and `<informalfigure>` are allowed. Width and height can be set on the image data.



Figure 6-1 - An example of a figure... (this in fixed width)

Add `role="nonnumber"` to a `<figure>` to stop it from getting a number.

In running the conversion pipelines, you can specify (as an option) a special "global" directory that contains global images (and other resources). When an image is located in this global directory add a `role="global"` to the `<figure>` element. Any directory information in `@fileref` is ignored.

- **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, warnings & cautions:**

NOTE:

This is a note! ... (`<note>`)

WARNING:

This is a warning! ... (`<warning>`)

CAUTION:

This is a caution! ... (`<caution>`)

If you add a `<title>` element, the standard title will be replaced by its contents.

- **Sidebars & tips:**

Title of the sidebar

Contents of the sidebar. ... (`<sidebar>`)

Title of the tip

Contents of the tip. ... (`<tip>`)

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

Lorem ipsum dolor sit amet, consectetur adipiscing elit. ...

Example 6-1 - Example of an example

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

- **Block quotes:**

Example of a `<blockquote>` element's output...

6.5 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.


Lot's of other elements that have to do with programming (like `<function>` and `<varname>`) have the same effect.

- **<link>**: Outputs some link (e.g. a web address). Use one of:

- @xlink:href for an external web address.
- @linkend for an internal id.

The visible contents will consist of either the contents of the <link> element or (if empty) the contents of @xlink:href or @linkend. Like [this](#) or like this <http://www.xatapult.nl>.

For HTML, add role="newpage" to get a new page/tab when clicking on the link.

- **<inlinemediaobject>**: Inserts an inline image , like this.

In running the conversion pipelines, you can specify (as an option) a special "global" directory that contains global images (and other resources). When an image is located in this global directory add a role="global" to the <inlinemediaobject> element. Any directory information in @fileref is ignored.

- **<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**
- **<quote>**: Use for adding a quote: "To be or not to be..."
- **<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 6-3

For HTML, add role="newpage" to get a new page/tab when clicking on the link.

- **<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 16
To a chapter or appendix	chapter # or appendix #
To a section	"Document information" on page 16
To a table (with a number), like this one	table 6-4
To a figure (with a number)	figure 6-1
To an example (with a number)	example 6-1
To anything else	First paragraph: page 16 Unnumbered table: page 17

Table 6-4 - Examples of `<xref>` usage

- `<footnote>` Adds a footnote¹
- There are lots of elements that are ignored. For instance all the `<gui...>` elements, `<orgname>` and many more (but the list is not (yet) DocBook complete).

6.6 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...]`

6.7 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).

1. This is a footnote's text!