xtpxlib-container

XML Container Handling

Erik Siegel - Xatapult Content Engineering
2020-10-14

0 Table of Contents

0 Xatapult XML Library - XML Container Handling

xtpxlib

xtpxlib library - component **xtpxlib-container** - **v1.0.B** (2020-04-09)
Xatapult Content Engineering - http://www.xatapult.com - +31 6 53260792
Erik Siegel - erik@xatapult.com

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

XML Containers provide support for working with multiple related files by wrapping them into a single one. Binary files are referenced instead of included.

The container structure is standardized. Once contents is in a container it's easy to analyze, change and/or write back. It can also be used to create a whole file structure, in a container, and then write it out to disk or zip file.

The `xtpxlib-container` component has XProc (1.0 and 3.0) pipelines for:

- Reading the contents of a zip file or directory structure into a container
- Writing a container out to a zip file or disk

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

Technical information:

Component documentation: https://container.xtpxlib.org
License: GNU GENERAL PUBLIC LICENSE - Version 3, 29 June 2007
Git URI: `git@github.com:xatapult/xtpxlib-container.git`
Git site: https://github.com/xatapult/xtpxlib-container
This component depends on:

- xtpxlib-common (Common component: Shared libraries and IDE support)

1 Description

In working with XML or HTML, it is often the case that you need to work with multiple documents *at the same time*. And in most cases that's rather awkward. For instance, an XSLT transformation processes a single document. And yes, of course you can get others by calling the `doc()` function or produce others using `<xsl:result-document>`. But, especially when there are a lot of relations between the documents, this requires careful and sometimes heavy programming. The idea of *XML containers* tries to make this more manageable.

An XML container (as handled/used by this module) is an XML structure that holds other XML documents and references to binary files. Here is a short example:

```
<document-container xmlns="http://www.xtpxlib.nl/ns/container"
  timestamp="2019-12-12T12:11:10"
  href-target-path="/my/website/location">

  <document href-target="index.html">
    <html> … </html>
  </document>

  <document href-target="page1.html">
    <html> … </html>
  </document>

  <external-document href-source="/my/resources/image.jpg" href-target="resources/image.jpg"/>
</document-container>
```

Example 1-1 - Example of an XML container

This example shows a container, probably generated by some pipeline or XSLT stylesheet, that contains the contents of a simple website. All two pages and some image are there. Running this container through `xtlcon:container-to-disk` will write it to the path indicated in `/*/@href-target-path`: `/my/website/location`. The documents `index.html` and `page1.html` come from the container, the binary `image.jpg` is copied from the indicated source location. Because everything, every separate file, is in (or referenced in) a single encompassing document, lots of things get easier: creating or checking internal referencing, making classes consistent, etc. An XSLT stylesheet that gets this as its main document has access to *all* information.

1.1 Applications

As it turned out, the whole idea of working with multiple documents in an XML container had several applications:

- An important application of the zip format is its use as an overarching storage format for applications. For instance, most office suites do this: a Microsoft Word `.docx` or Excel `.xlsx` file is actually a zip file with many smaller files inside (most of them in XML format). There are many other examples.

  Trying to interpret such a zip file and get something meaningful out of it can be a nightmarish experience, especially if you want to follow the standard (and not rely on some file naming convention some engineer cooked up and might change). It takes following links through several files to the place the actual interesting information is stored.

  But if you run such a file through `xtlcon:zip-to-container` you get all files in a single encompassing one, making it much, much easier to follow internal links and find the right information. The `xtpxlib-xoffice` component does exactly this: it contains pipelines to get the contents of Word and Excel files in an easier to interpret XML format.
- Going even further with this, it is now much easier to *change* or even *create* such a horribly complex Word, Excel or other kind of office zip file:
  - Read a (template) office document in using `xtlcon:zip-to-container`.
  - Change what you need to change (text, spreadsheet cell values, etc.). Leave the rest, with all this complex linking and other stuff you don't really need to understand, alone.
  - Write it to a resulting zip file using `xtlcon:container-to-zip`.
- A file structure that needs to end up on disk or in a zip file can be created easily using this XML container mechanism.

1.2 Working with containers

The container format is described here Working with XML container documents is done using XProc 1.0 or XProc 3.0 pipelines.

WARNING: The container formats and processing features differ between the 1.0 and the 3.0 version! More about this in the container format description.

There are some notable missing features in the current container handling. These are not impossible to implement, the need for them just hasn't arisen yet.

- When writing a zip file you cannot control the compression (different ones, on or off). This means that this mechanism currently can't produce e-books (which require an uncompressed first file).
- You can't work with binary contents *inside* the container, for instance when its `base64` encoded.

2 Container format

WARNING: The container formats and processing features differ between the XProc 1.0 and the 3.0 version!

- Description of the container format for XProc 1.0
- Description of the container format for XProc 3.0

2.1 XProc 1.0 container format

The schema for the XProc 1.0 container format can be found in `xsd/container.xsd`.

The root of an XProc 1.0 XML container document is an `<xtlcon:document-container>` element. The prefix `xtlcon:` must be bound to the namespace `http://www.xtpxlib.nl/ns/container` (`xmlns:xtlcon="http://www.xtpxlib.nl/ns/container"`).

```
<xtlcon:document-container timestamp = xs:dateTime
                           href-source-zip? = xs:string
                           href-target-zip? = xs:string
                           href-target-zip-result? = xs:string
                           href-target-zip-tmpdir? = xs:string
                           href-source-path? = xs:string
                           href-target-path? = xs:string
                           href-target-result-path? = xs:string
                           (any)? >
  ( <xtlcon:document> |
    <xtlcon:external-document> |
    <(any)> )*
</xtlcon:document-container>
```

Attribute	#	Type	Description
`timestamp`	1	`xs:dateTime`	The timestamp when this container was initially created/generated.
`href-source-zip`	?	`xs:string`	When the container was read from a zip file (using `xtlcon:zip-to-container`), this attribute holds the href of this zip file.
`href-target-zip`	?	`xs:string`	Holds the name of the zip file for writing the container to (using `xtlcon:container-to-zip`).
`href-target-zip-result`	?	`xs:string`	After the container is written to a zip file using `xtlcon:container-to-zip`, this attribute will hold the full canonical filename of the zip file.
`href-target-zip-tmpdir`	?	`xs:string`	After the container is written to a zip file using `xtlcon:container-to-zip`, this attribute will hold the full canonical name of the temporary directory used for this process (probably not very useful).
`href-source-path`	?	`xs:string`	When the container was read from a directory (using `xtlcon:directory-to-container`), this attribute holds the href of this directory.
`href-target-path`	?	`xs:string`	Holds the name of the directory for writing the container to (using `xtlcon:container-to-disk`).
`href-target-result-path`	?	`xs:string`	After the container is written to a directory file using `xtlcon:container-to-disk`, this attribute will hold the full canonical name of the directory.
`(any)`	?		Any other attributes are allowed, so additional information can be added for use during processing.

Child element	#	Description
`xtlcon:document`	*	A document inside the container structure. See "Internal documents" on page 6.
`xtlcon:external-document`	*	An external document, referenced from the container structure. See "External documents" on page 6.
`(any)`	*	Any other elements are allowed, so additional information can be added for use during processing.

2.1.1 Internal documents

An *internal* document is a document whose contents is inside the container document. This will in most cases be XML documents, but text is also possible. It must be surrounded by an `<xtlcon:document>` element:

```
<xtlcon:document href-source? = xs:string
                 href-source-result? = xs:string
                 href-target? = xs:string
                 href-target-result? = xs:string
                 mime-type? = xs:string
                 (any)? >
  <(any)>
</xtlcon:document>
```

Attribute	#	Type	Description
`href-source`	?	`xs:string`	href of the source for this document. When this document comes from a zip file, it holds the href of the file *in* the zip.
`href-source-result`	?	`xs:string`	After processing holds the full canonical name of the source file.
`href-target`	?	`xs:string`	href of the target for this document.
`href-target-result`	?	`xs:string`	After processing holds the full canonical name of the target file.
`mime-type`	?	`xs:string`	Some specific values for this attribute trigger special conversions on output. See "Document MIME type values" on page 7.
`(any)`	?		Any other attributes are allowed, so additional information can be added for use during processing.

Child element	#	Description
`(any)`	1	Root element + contents of the document.

2.1.2 External documents

An *external* document is a document that is only referenced from the container. Usually binary files but anything goes. The referencing is done using an `<xtlcon:external-document>` element:

```
<xtlcon:external-document (attributes-from-internal-document)?
                          href-source-zip? = xs:string
                          href-source-zip-result? = xs:string
                          not-in-global-source-zip? = xs:boolean >
  <(any)>
</xtlcon:external-document>
```

All attributes of an internal document plus the following:

Attribute	#	Type	Description
`(attributes-from-internal-document)`	?		See internal documents.
`href-source-zip`	?	`xs:string`	Reference to the source zip file for this document. If present overrides `/*/@href-source-zip`
`href-source-zip-result`	?	`xs:string`	After processing holds the full canonical name of the source zip file.
`not-in-global-source-zip`	?	`xs:boolean`	Default: `false` When set to `true`, the global zip file `/*/@href-source-zip` is not used. This is necessary to allow references to external files that, when a global zip file is used, come from elsewhere.

Child element	#	Description
(any)	1	Any other elements are allowed, so additional information can be added for use during processing.

2.1.3 Document MIME type values

The xtlcon:document/@mime-type attribute (see Internal documents) can trigger some special treatment of the contents of the element:

mime-type="application/pdf" *and* the root element is <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
: The document is assumed to be an XSL-FO document and the FOP XSL-FO processor is called to create a PDF document. The pipelines (both xtlcon:container-to-disk and xtlcon:container-to-zip) allow you to specify an FOP configuration file.

mime-type="text/plain"
: The contents of the <xtlcon:document> element will be stringified and stored as text. The schema requires some root element to be present as child of <xtlcon:document> but this is ignored:

```
<xtlcon:documnent mime-type="text/plain" …>
<dummy-root>Text to be stored
another line…</dummy-root>
</xtlcon:document>
```

2.2 XProc 3.0 container format

The container format for XProc 3.0 allows documents in XML, HTML, text and JSON format. It can also process JSON as XML.

The schema for the XProc 3.0 container format can be found in xsd/container-xpl3.xsd.

The root of an XProc 3.0 XML container document is an <xtlcon:document-container> element. The prefix xtlcon: must be bound to the namespace http://www.xtpxlib.nl/ns/container (xmlns:xtlcon="http://www.xtpxlib.nl/ns/container").

> **NOTE:**
> When you get the container back after processing by one of the pipelines, you'll find most href related attributes (attributes that start with href-) copied to an attribute with the same name plus an underscore in front. The values of these added attributes will be the fully expanded and canonicalized references.
>
> For instance: When the container has an attribute href-target-zip="../out.zip", after processing you'll find an attribute like _href-target-zip="file:///ful/path/to/out.zip" on the result.

```
<xtlcon:document-container timestamp = xs:dateTime
                           href-source-path? = xs:string
                           href-target-path? = xs:string
                           href-source-zip? = xs:string
                           href-target-zip? = xs:string
                           (any)? >
  ( <xtlcon:document> |
    <xtlcon:external-document> |
    <(any)> )*
</xtlcon:document-container>
```

Attribute	#	Type	Description
timestamp	1	xs:dateTime	The timestamp when this container was initially created/generated.
href-source-path	?	xs:string	When the container was read from a directory (using xtlcon:directory-to-container), this attribute holds the href of this directory.
href-target-path	?	xs:string	Contains the path for the directory when writing the container to (using xtlcon:container-to-disk).

Attribute	#	Type	Description
`href-source-zip`	?	`xs:string`	When the container was read from a zip file (using `xtlcon:zip-to-container`), this attribute holds the href of this zip file.
`href-target-zip`	?	`xs:string`	Holds the path for the zip file when zipping the container's contents (using `xtlcon:container-to-zip`).
`(any)`	?		Any other attributes are allowed, so additional processing information can be added.

Child element	#	Description
`xtlcon:document`	*	A document inside the container structure. See "Internal documents" on page 8.
`xtlcon:external-document`	*	An external document, referenced from the container structure. See "External documents" on page 9.
`(any)`	*	Any other elements are allowed (in a different namespace), so additional processing information can be added.

2.2.1 Internal documents

An *internal* document is a document whose contents is *inside* the container document. It must be surrounded by an `<xtlcon:document>` element.

```
<xtlcon:document href-source? = xs:string
                 href-target? = xs:string
                 href-source-zip? = xs:string
                 content-type? = xs:string
                 serialization? = xs:string
                 (any)? >
  <(any)>
</xtlcon:document>
```

Attribute	#	Type	Description
`href-source`	?	`xs:string`	href of the source for this document. Will be filled by `xtlcon:directory-to-container` and the `xtlcon:zip-to-container` pipelines. When this document comes from disk, it holds the the relative filename against the value of the container's `/*/@href-source-path`. When this document comes from a zip file, it holds the (relative) href of the file *in* the zip.
`href-target`	?	`xs:string`	href of the target location for this document. Used by the `xtlcon:container-to-disk` and `xtlcon:container-to-zip` pipelines. When writing to disk, a relative filename is made absolute against the container's `/*/@href-target-path`. When writing to a zip, it *must* be a relative filename. This will become the path of the file *in* the zip file.
`href-source-zip`	?	`xs:string`	If present, overrides the value of the container's `/*/@href-source-zip`
`content-type`	?	`xs:string`	Content (MIME) type of the document. Will be filled by `xtlcon:directory-to-container` and the `xtlcon:zip-to-container` pipelines. If not present when writing a container, `text/xml` is assumed for internal documents

Attribute	#	Type	Description
`serialization`	?	`xs:string`	Serialization settings for this document, expressed as a JSON map. To allow the usual double quoted attribute values ("…"), all single quotes will be converted to double quotes before JSON parsing is done. For instance: `serialization="{'indent': true}"`
`(any)`	?		Any other attributes are allowed, so additional processing information can be added.

Child element	#	Description
`(any)`	1	Document contents (For text and JSON does not need to be XML).

2.2.2 External documents

An *external* document is a document that is only referenced from the container. Usually binary files but anything goes. The referencing is done using an `<xtlcon:external-document>` element:

```
<xtlcon:external-document href-source? = xs:string
                          href-target? = xs:string
                          href-source-zip? = xs:string
                          not-in-zip? = xs:boolean
                          (any)? >
  <(any)>*
</xtlcon:external-document>
```

Attribute	#	Type	Description
`href-source`	?	`xs:string`	href of the source for this document. Will be filled by `xtlcon:directory-to-container` and the `xtlcon:zip-to-container` pipelines. When this document comes from disk, it holds the the relative filename against the value of the container's `/*/@href-source-path`. When this document comes from a zip file, it holds the (relative) href of the file *in* the zip.
`href-target`	?	`xs:string`	href of the target location for this document. Used by the `xtlcon:container-to-disk` and `xtlcon:container-to-zip` pipelines. When writing to disk, a relative filename is made absolute against the container's `/*/@href-target-path`. When writing to a zip, it *must* be a relative filename. This will become the path of the file *in* the zip file.
`href-source-zip`	?	`xs:string`	If present, overrides the value of the container's `/*/@href-source-zip`
`not-in-zip`	?	`xs:boolean`	Forces this document to load from disk, even when a zip file reference is specified.
`(any)`	?		Default: `false` Any other attributes are allowed, so additional processing information can be added.

Child element	#	Description
`(any)`	*	Any contents allowed for additional processing purposes.

3 XProc 1.0 Libraries

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

Module/Pipeline	Description
container.mod.xpl	XProc library with steps for handling xtpxlib containers.

Table 3-1 - Module overview

3.1 XProc (1.0) library: container.mod.xpl

File: `xplmod/container.mod/container.mod.xpl`

XProc library with steps for handling xtpxlib containers.

Prefix	Namespace URI
xtlcon	http://www.xtpxlib.nl/ns/container

Step	Description
xtlcon:container-to-disk	Writes the contents of a container to disk.
xtlcon:container-to-zip	Writes the contents of a container to a zip file.
xtlcon:directory-to-container	Reads a directory into a container. All XML files will be read into the container, all other files will be included/referenced as external contents.
xtlcon:zip-to-container	Reads a zip file into a container. All XML files will be read into the container, all other files will be included/referenced as external contents.

3.1.1 Step: xtlcon:container-to-disk

Writes the contents of a container to disk.

Port	Type	Primary?	Description
source	in	yes	The container to process.
result	out	yes	The input container, but with changes that reflect the writing process.

Option	Rq?	Default	Description
href-fop-config		resolve-uri('../../../xtpxlib-common/data/fop-default-config.xml', static-base-uri())	Optional reference to an Apache FOP configuration file. Must be absolute! When not present a default file will be used.
href-target		''	Base path where to write the container. When you specify this it will have precedence over a /*/@href-target-path.
indent-xml		false()	Whether to indent the XML we create or not.
remove-target		true()	Whether to attempt to remove the target directory before writing.

3.1.2 Step: xtlcon:container-to-zip

Writes the contents of a container to a zip file.

Port	Type	Primary?	Description
source	in	yes	The container to process.
result	out	yes	The input container, but with all the changes in links, paths, etc.

Option	Rq?	Default	Description
href-fop-config		''	Optional reference to an Apache FOP configuration file. Must be absolute! When not present a default file will be used.
href-target-zip			Base path where to write the container. When you specify this it will have precedence over /*/@href-target-zip.
indent-xml		false()	Whether to indent the XML we create or not.

3.1.3 Step: xtlcon:directory-to-container

Reads a directory into a container. All XML files will be read into the container, all other files will be included/referenced as external contents.

Port	Type	Primary?	Description
`result`	out	yes	The output container.

Option	Rq?	Default	Description
`add-document-target-paths`		`true()`	Adds (relative) source paths as the target paths to the individual documents.
`href-source-directory`	yes		Reference to the directory to read.
`href-target-path`		`''`	Optional target path to record on the container.

3.1.4 Step: xtlcon:zip-to-container

Reads a zip file into a container. All XML files will be read into the container, all other files will be included/referenced as external contents.

Port	Type	Primary?	Description
`result`	out	yes	The output container.

Option	Rq?	Default	Description
`add-document-target-paths`		`true()`	Adds source paths as the target paths to the individual documents.
`href-source-zip`	yes		Reference to the zip file to read.
`href-target-path`		`''`	Optional target path to record on the container.

4 XProc 3.0 Support

The xtpxlib-container component contains the following support (pipelines and/or libraries) for XProc 3.0:

Module/Pipeline	Description
container-to-disk.xpl	Writes an xtpxlib container structure to disk. A base path must be provided, either as option $href-target path or as /*/@href-target.
container-to-zip.xpl	Writes an xtpxlib container structure to a zip file. A name for the output zip can be specified in either option $href-target-zip or on the container in /*/@href-target-zip.
directory-to-container.xpl	Loads a directory (with optional sub-directories) into an xtpxlib container structure.
zip-to-container.xpl	Loads the contents of a zip file (directory depth can be set) into an xtpxlib container structure.

Table 4-1 - Module overview

4.1 XProc (3.0) pipeline: container-to-disk.xpl

File: `xpl3mod/container-to-disk/container-to-disk.xpl`

Type: `xtlcon:container-to-disk`

Writes an xtpxlib container structure to disk. A base path must be provided, either as option `$href-target` path or as `/*/@href-target`.

Port	Type	Primary?	Description
source	in	yes	The container to process.
result	out	yes	The input container structure with additional shadow attributes filled.

Option	Type	Rq?	Default	Description
href-target-path	xs:string?		()	Base path where to write the container. When you specify this it will have precedence over a /*/@href-target-path.
remove-target	xs:boolean		true()	Whether to attempt to remove the target directory before writing.

4.2 XProc (3.0) pipeline: container-to-zip.xpl

File: `xpl3mod/container-to-zip/container-to-zip.xpl`

Type: `xtlcon:container-to-zip`

Writes an xtpxlib container structure to a zip file. A name for the output zip can be specified in either option `$href-target-zip` or on the container in `/*/@href-target-zip`.

Port	Type	Primary?	Description
source	in	yes	The container to process.
result	out	yes	The input container structure with additional shadow attributes filled.

Option	Type	Rq?	Default	Description
href-target-zip	xs:string?		()	Nem of the zip file to write. When you specify this it will have precedence over a /*/@href-target-zip.

4.3 XProc (3.0) pipeline: directory-to-container.xpl

File: `xpl3mod/directory-to-container/directory-to-container.xpl`

Type: `xtlcon:directory-to-container`

Loads a directory (with optional sub-directories) into an xtpxlib container structure.

Port	Type	Primary?	Description
`result`	out	yes	The resulting container structure.

Option	Type	Rq?	Default	Description
`add-document-target-paths`	`xs:boolean`		`false()`	Copies the relative source path as the target path `@target-path` for the individual documents. The idea behind this is that in some cases you want to write almost the same structure back to disk. Recording the relative source path as the target path makes this easier: you don't have to set it explicitly.
`depth`	`xs:integer`		`-1`	The sub-directory depth to go. When lt `0`, all sub-directories are processed.
`exclude-filter`	`xs:string*`		`'\.git/'`	Optional regular expression exclude filters. By default, `.git` directories are excluded.
`href-source-directory`	`xs:string`	yes		URI of the directory to read.
`href-source-directory`	`xs:string`		`resolve-uri('test/', static-base-uri())`	

Option	Type	Rq?	Default	Description
`href-target-path`	`xs:string?`		`()`	Optional target path to record on the container. The idea behind this is that this makes it easier to write the container back to another location on disk, the target path is already there.
`include-filter`	`xs:string*`		`()`	Optional regular expression include filters.
`json-as-xml`	`xs:boolean`		`if ($develop) then $develop-load-json-as-xml else false()`	When JSON files are loaded (`option $load-json` is `true`): whether to add them to the container as XML or as JSON text. It will set the entry's content type to `application/json+xml`.
`load-html`	`xs:boolean`		`if ($develop) then $develop-load-html else false()`	Whether to load HTML files.
`load-json`	`xs:boolean`		`if ($develop) then $develop-load-json else false()`	Whether to load JSON files.
`load-text`	`xs:boolean`		`if ($develop) then $develop-load-text else false()`	Whether to load text files.
`override-content-types`	`array(array(xs:string))?`		`()`	Override content types specification (see description of `p:directory-list`).

4.4 XProc (3.0) pipeline: zip-to-container.xpl

File: `xpl3mod/zip-to-container/zip-to-container.xpl`

Type: `xtlcon:zip-to-container`

Loads the contents of a zip file (directory depth can be set) into an xtpxlib container structure.

Port	Type	Primary?	Description
`result`	out	yes	The resulting container structure

Option	Type	Rq?	Default	Description
`add-document-target-paths`	`xs:boolean`		`false()`	Copies the relative source path as the target path `@target-path` for the individual documents. The idea behind this is that in some cases you want to write almost the same structure back to disk or zip. Recording the relative source path as the target path makes this easier: you don't have to set it explicitly.
`depth`	`xs:integer`		`-1`	The sub-directory depth to go. When lt `0`, all sub-directories are processed.
`exclude-filter`	`xs:string*`		`'\.git/'`	Optional regular expression exclude filters. By default, `.git` directories are excluded.
`href-source-zip`	`xs:string`	yes		URI of the directory to read.
`href-source-zip`	`xs:string`		`resolve-uri('test/test-contents.zip', static-base-uri())`	

Option	Type	Rq?	Default	Description
`href-target-path`	`xs:string?`		`()`	Optional target path to record on the container. The idea behind this is that this makes it easier to write the container back to another location on disk, the target path is already there.
`include-filter`	`xs:string*`			Optional regular expression include filters.
`json-as-xml`	`xs:boolean`		`if ($develop) then $develop-load-json-as-xml else false()`	When json files are loaded (`option $load-json` is `true`): whether to add them to the container as XML or as JSON text. It will set the appropriate entry's content type to `application/json+xml`.
`load-html`	`xs:boolean`		`if ($develop) then $develop-load-html else false()`	Whether to load HTML files.
`load-json`	`xs:boolean`		`if ($develop) then $develop-load-json else false()`	Whether to load JSON files.
`load-text`	`xs:boolean`		`if ($develop) then $develop-load-text else false()`	Whether to load text files.
`override-content-types`	`array(array(xs:string))?`		`()`	Override content types specification (see description of `p:archive-manifest`).

5 XML Schemas

The xtpxlib-container component contains the following XML Schemas:

Module/Pipeline	Description
container-xpl3.xsd	Schema for an XML container (XProc 3 based pipelines)
container.xsd	Schema for an XML container.

Table 5-1 - Module overview

5.1 XML Schema: container-xpl3.xsd

File: xsd/container-xpl3.xsd
Target namespace: http://www.xtpxlib.nl/ns/container
Schema for an XML container (XProc 3 based pipelines)

Element	Description
document-container	Root element for a document container.

5.2 XML Schema: container.xsd

File: xsd/container.xsd
Target namespace: http://www.xtpxlib.nl/ns/container
Schema for an XML container.

Element	Description
document-container	Root element for a document container.