xtpxlib-container

XML Container Handling

Erik Siegel - Xatapult Content Engineering
2019-12-11

0 Table of Contents

0 Xatapult XML Library - XML Container Handling

xtpxlib library - component **xtpxlib-container** - v0.0 2019-12-10
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) 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, requires careful and sometimes heavy programming. The idea of *XML containers* tries to make this all 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) pipelines which are described here.

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.

- You currently cannot control the serialization of the XML documents inside the container. The only thing you can do now is set indenting on or off on a *global* level.
- 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

The root of an 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 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.

2.2 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.

2.3 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>
```

3 XProc Libraries

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

Module	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 XML Schemas

The xtpxlib-container component contains the following XML Schemas:

Module	Description
`container.xsd`	Schema for an XML container.

Table 4-1 - Module overview

4.1 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.