Transforming documents with OpenAPI pipelines

Ashley Noel Hinton
ahin017@aucklanduni.ac.nz

Paul Murrell
paul@stat.auckland.ac.nz

Department of Statistics, The University of Auckland

1 A markup transformable document format

2 The `document` markup format

The transformable document format described in this report is an XML file with `document` as the root element. This document has two child elements: `metadata` and `body`.

The `metadata` element contains the document metadata, with elements for the document `title` and `subtitle`, `author` information, `date` of publication, and a `description` section. An example `metadata` element follows:

```
<metadata>
  <title>Today should be a holiday</title>
  <author>
    <name>Ashley Noel Hinton</name>
    <email>ahin017@aucklanduni.ac.nz</email>
  </author>
  <date>25 December 2015</date>
</metadata>
```

The `body` element contains the document's main content. The following elements are used in the same way as they are used in HTML (`https://www.w3.org/TR/html-markup/elements.html`):

- `a` – hyperlink
- `code` – code fragment
- `em` – emphatic stress
- `figcaption` – figure caption

- `figure` – figure with optional caption
- `h1` – heading
- `h2` – heading
- `h3` – heading
- `img` – image
- `li` – list item
- `ol` – ordered list
- `p` – paragraph
- `pre` – preformatted text
- `q` – quoted text
- `section` – section
- `strong` – strong importance
- `ul` – unordered list

The `<url>` element is introduced in the `document` format to indicate a hyperlink where the enclosed URL is both the href and the value. The following code block demonstrates the use of the `url` element:

```
<ul>
  <li>modular</li>
  <li>reusable</li>
  <li>shareable</li>
  <li><url>https://github.com/anhinton/conduit</url></li>
</ul>
```

The resulting output:

- modular
- reusable
- shareable
- `https://github.com/anhinton/conduit`

The document XML format uses <code> elements to indicate blocks of computer code, just as in HTML. Dynamic code chunks which are to be executed are marked using the class attribute to code. For example chunks of R code which are to be executed used the Knitr package are wrapped in a <code> element with class="knitr". An author can also provide a name attribute for the knitr code chunk, as well as knitr options. A document author can also use CDATA sections to wrap code with reserved XML characters. The following code demonstrates how to include an R code chunk to be executed with Knitr:

```
<code class="knitr" name="knitrDemo" options="tidy=FALSE">
<![CDATA[x <- rnorm(n = 10)
mean(x)]]></code>
```

And the following is the result of executing this code chunk:

```
x <- rnorm(n = 10)
mean(x)
```

```
## [1] -0.2431044
```

The document format also makes use of the include element from XInclude (http://www.w3.org/2001/XInclude) namespace to include XML content from external files. This allows document authors to embed other documents which may be authored separately from the main document. There is no simple method of doing this directly in either HTML or Pandoc Markdown.

The next sections describes some simple transformations which can be performed on the document markup format using freely available open source tools. This report was itself written in the document markup format—the source code is available at report.xml.

3 Using OpenAPI pipelines for transformation

The OpenAPI architecture helps to break tasks in data analysis down into small pieces making it easier for people to contribute to a data problem. The goal of the OpenAPI project is to make it easier for people to connect with data. Meaningful steps in a data workflow can be wrapped as modules. Modules can be arranged in pipelines, and shared to be recombined by other authors in their own pipelines. Pipelines and modules can be executed by OpenAPI glue system software. The whole project is open source, and open to contributions from anyone (Introducing OpenAPI, OpenAPI version 0.6).

It is OpenAPI's dividing of tasks into modules which makes it an ideal candidate for handling document transformation. Transforming the document format described in the previouse section can be broken down into several discrete steps:

1. Merge XML code indicated by `xi:include` elements into the document.
2. Convert XML character entities into appropriate characters.
3. Convert document into output format language.
4. Execute embedded chunks of code.

Several technologies already exist for handling each of these steps. For example, the xmllint (`http://www.xmlsoft.org/`) command line tool can replace XInclude code, and replace entities with their values. The xsltproc (ibid.) command line tool can be used to apply an XSL stylesheet to the `document` to produce the desired output format. The Knitr package in R can be used to execute chunks of R code in various document formats.

What the OpenAPI architecture offers is the ability to wrap transformation steps in a module which takes a file as an input, and produces another file as an output. The output of one module can be passed as the intput of another module, thus building a pipeline which describes the entire transformation. Each module in an OpenAPI pipeline specifies its execution language, meaning an OpenAPI pipeline can have access to a wide variety of tools.

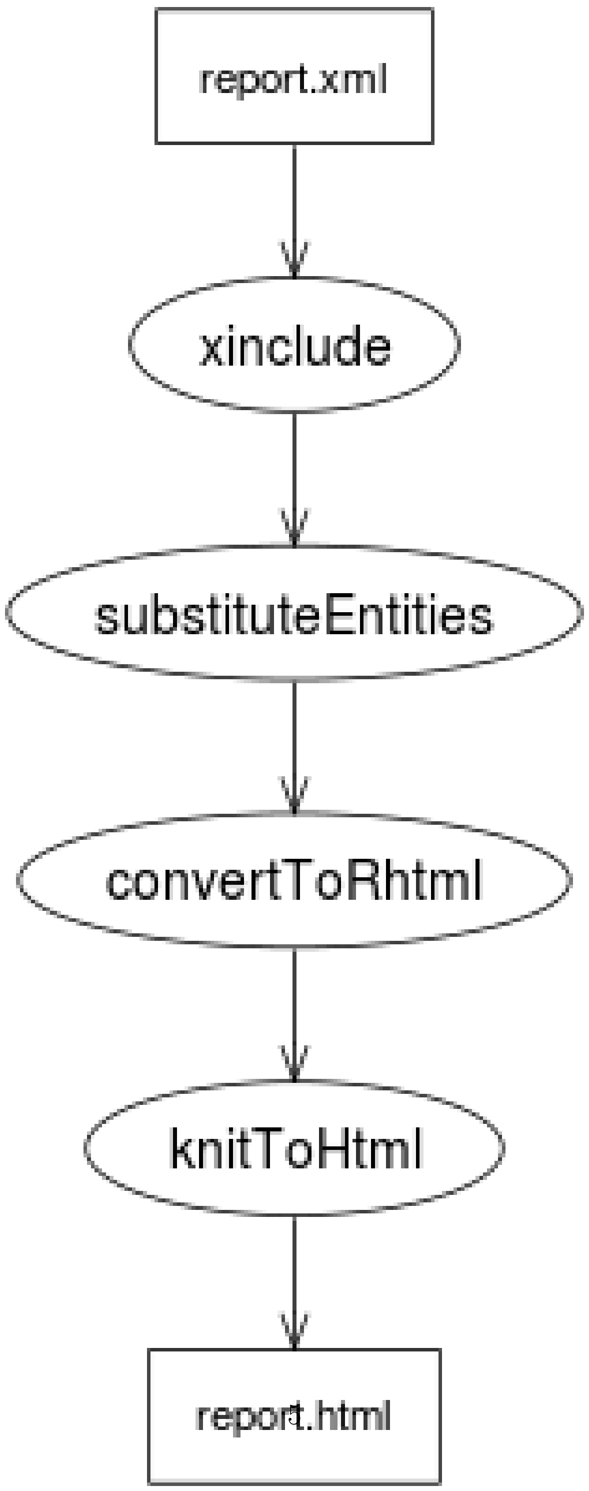
In the following section I will descibe the modules used to transform a tranformable `document` file to HTML output, including the processing of chunks of R code. After this section I will show how this pipeline can be modified to produce PDF and Markdown output from the same source `document`.

3.1 Transformation to HTML

The first pipeline example, `toHtml`, converts a `document` source file to an HTML file with embedded R code, and the results of executing this R code.

3.1.1 Include referenced XML

The first module in the `toHtml` pipeline is the `xinclude` module, which processes all of the `<xi:include>` elements in the source document and produces a document with the reference XML documents inline. The `xinclude` module is a bash-language module, and wraps a bash script which calls the `xmllint` command-line tool to perform the transformation. This module requires four input objects: report which references the source document, `report.xml`; metadataExample which references the XML file `metadataExample.xml`; elementsExample which references the XML file `elementsExample.xml`; and knitrExample which references the XML file `knitrExample.xml`.


report.xml
xinclude
substituteEntities
convertToRhtml
knitToHtml
report.html

The module produces a single output, report, which references the transformed XML file produced by the transformation. The XML source for the `xinclude` module can be found at `transform/toHtml/xinclude.xml`.

3.1.2 XML entities

The second transformation module in the pipeline is the `substituteEntities` module. This module replaces the XML entities in the source document with the values provided in the document's doctype declaration. The `substituteEntities` module is a bash-language module with one input, report, an XML file provided by the `xinclude` module. This module wraps a script which uses the `xmllint` command-line tool to replace the source documents XML entities with their values. The module produces one output, report, which references the transformed source XML file. The XML source for the `substituteEntities` module can be found at `transform/toHtml/substituteEntities.xml`.

3.1.3 Produce `.Rhtml` file

The third module in this pipeline is the `convertToRhtml` module. This module transforms the source document into an HTML document with Knitr R code chunks. This module is a bash-language module with two inputs: report, an XML file provided by the `substituteEntities` module; and toRhtml, which references an XSLT stylesheet file `xsl/toRhtml.xsl`, which describes the transformation. The module wraps a script which uses the `xsltproc` command-line tool to transform the source document into a Knitr HTML file. The module produces one output, report, which references the Knitr HTML file produced in the transformation. The XML source for the `convertToRhtml` module can be found at `transform/toHtml/convertToRhtml.xml`.

3.1.4 Produce `.html` file

The fourth module in the `toHtml` pipeline is the `knitToHtml` module. This module executes the R code chunks in a Knitr HTML file and returns the resulting HTML file. This module is a R-language module with one input, report, a Knitr HTML file provided by the `convertToRhtml` module. The module wraps an R script which call the `knit` function from the Knitr package to execute the R code chunks and returns an HTML file. The module produces two outputs: report, which references the HTML file produced by the module source script, and toHtmlGraph, a PNG image file. update with toPdfGraph AND propagate module to toPdf pipeline. The XML source for the `knitToHtml` module can be found at `transform/toHtml/knitToHtml.xml`.

3.2 Transformation to PDF

The second pipeline example, `toPdf`, converts a `document` source file to a LaTeX file

3.2.1 Substitute LaTeX entity values

The first new module in the `toPdf` pipeline is the `texChars` module. This module replaces the entity definitions in the source document doctype declaration with values appropriate to LaTeX character typesetting—the source document entity values are appropriate to HTML. This module is a bash-language module with one input, report, an XML file provided by the `xinclude` module. The module wraps a bash script which substitutes the entity values using the `sed` command-line tool. The module produces one output, report, which references the transformed XML file. The XML source for the `latexChars.xml` module can be found at `transform/toPdf/latexChars.xml`.

3.2.2 XML entities

The `latexEntities` modules is identical to the `substituteEntities` module in the `toHtml` pipeline. It takes its input from the `texChars` modules, and produces the output report, which references the transformed source document. The source XML for `latexEntities` is the same as for `substituteEntities`, `transform/toHtml/substituteEntities.xml`.

3.2.3 R code comments

The `commentCode` module adds the LaTeX comment character, %, to the beginning of each line of R code identified by `<code class = "knitr"<`, to conform to the Knitr packages standards for R code chunks in a Knitr LaTeX document. This module is an R-language module with its one input, report, provided by the `latexEntities` module. The module wraps a source script which uses the XML package to perform the required transformation on the `code` elements with class set to knitr. The module produces one output, report, which references the transformed source document. The XML source for the `commentCode` module can be found at `transform/toPdf/commentCode.xml`.

3.2.4 Produce `.Rtex` file

The `convertToRtex` module transforms the source XML document into a LaTeX document with Knitr R code chunks. This module is a bash-language module with two inputs: report, which is provided by the `commentCode` module; and toRtex, which references the XSLT stylesheet `xsl/toRtex.xsl`.

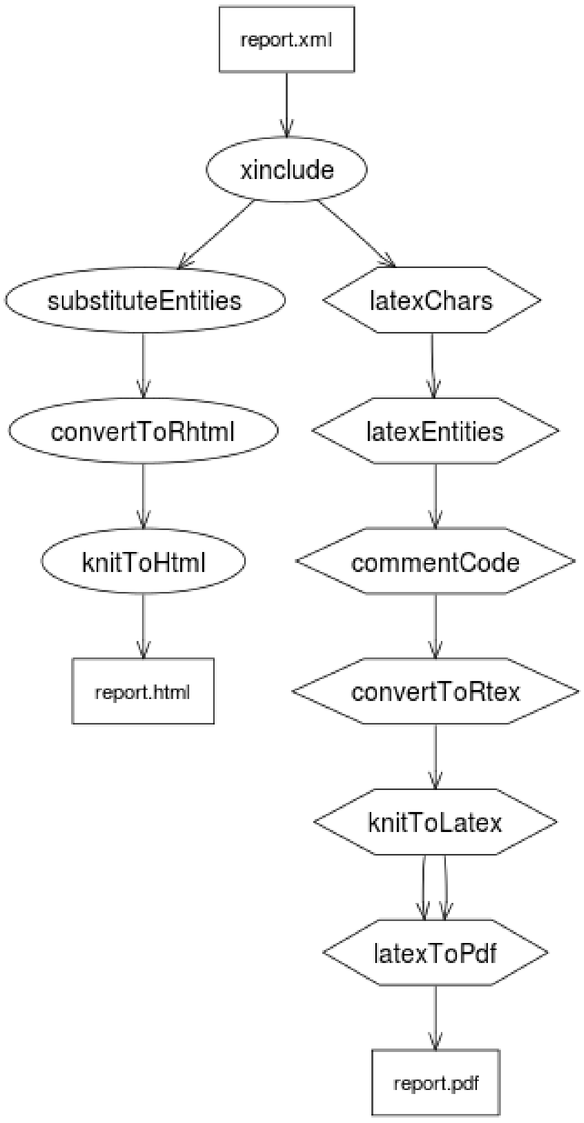


Figure 2: The toPdf pipeline

The module wraps a bash script which transforms the source document to a LaTeX document using the command-line tool `xsltproc`. The module produces one output, report, which references the Knitr LaTeX file resulting from the transformation. The XML source for the `convertToRtex` module can be found at `transform/toPdf/convertToRtex.xml`.

3.2.5 Produce `.tex` file

The `knitToLatex` module executes the chunks of R code in a Knitr LaTeX file and produces a LaTeX file. This is an R-language module with a single input, report, provided by the `convertToRtex` module. The module wraps an R script which call the `knit` function from the Knitr package to execute the R code chunks and returns a LaTeX file. The module produces two outputs: report, which references the LaTeX file produced; and toHtmlGraph, a PNG image file. update with toPdfGraph. The XML source for the `knitToLatex` module can be found at `transform/toPdf/knitToLatex.xml`.

3.2.6 Produce `.pdf` file

The `latexToPdf` module produces a PDF file from a LaTeX source file. This is a bash-language module with two inputs: report and toHtmlGraph, which are provided by the `knitToLatex` module. update with toPdfGraph. The module wraps a bash script which produces a PDF file using the `pdflatex` command-line tool. The module produces one output, report, the PDF file produced from the LaTeX source file. The XML source for the `latexToPdf` module can be found at `transform/toPdf/latexToPdf.xml`.

4 Background

5 Technical requirements

This is an appendix section.