RFC Beautification Working Group
Internet-Draft
Intended status: Informational
Expires: October 12, 2013

R. Gieben
Google

April 10, 2013

Writing I-Ds and RFCs using Pandoc
draft-gieben-pandoc2rfc-01

Abstract

This document presents a technique for using a Markdown syntax variant, called Pandoc, as a source format for documents in the Internet-Drafts (I-Ds) and Request for Comments (RFC) series.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 12, 2013.

Copyright Notice

Table of Contents

1. Introduction

This document presents a technique for using Pandoc syntax as a source format for documents in the Internet-Drafts (I-Ds) and Request for Comments (RFC) series.

Pandoc [Pandoc] is an "almost plain text" format and therefor particularly well suited for editing RFC-like documents. The syntax itself is a superset of the syntax championed by Markdown [Markdown].

Pandoc2rfc generates XML compatible with [RFC2629].

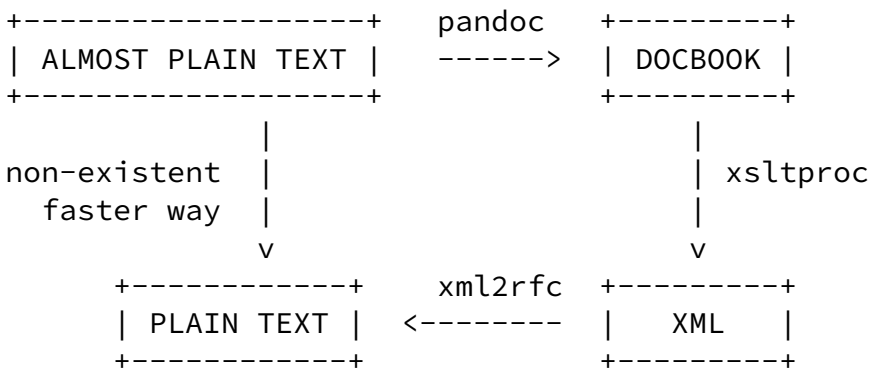
2. Pandoc to RFC

During the last few years people have been developing markup languages that are very easy to remember and type. These languages have become known as "almost plain text"-markup languages. One of the first was the Markdown ([Markdown]) syntax. One that was developed later and incorporates Markdown syntax and a number of extensions is Pandoc ([Pandoc]). The power of Pandoc comes from the

fact that it can be translated to numerous output formats, including, but not limited to: HTML, EPUB, (plain) Markdown and DocBook XML.

Pandoc2rfc allows authors to write in (the simple) Pandoc syntax which is then transformed to XML and given to xml2rfc. The

conversions are, in some way amusing, as we start off with (almost) plain text, use elaborate XML and end up with plain text again.



Attempt to justify Pandoc2rfc.

Figure 1

The XML generated (the output after the "xsltproc" step in Figure 1) is suitable for inclusion in either the "middle" or "back" section of an RFC.

Even though Pandoc2rfc abstracts away a lot of XML details, there are still a few places left where some XML needs to be edited. Most notably in the "front" section of an RFC.

The simplest way to start is to create a template XML file and include the appropriate XML for this "front" section:

```
<?xml version='1.0' ?>
<!DOCTYPE rfc SYSTEM 'rfc2629.dtd' [
<!ENTITY pandocMiddle PUBLIC '' 'middle.xml'>
<!ENTITY pandocBack   PUBLIC '' 'back.xml'>
<!ENTITY rfc.2629     PUBLIC '' 'reference.RFC.2629.xml'>
]>
```

```
<rfc ipr='trust200902' docName='draft-gieben-pandoc2rfc-01'>
 <front>
    <title>Writing I-Ds and RFCs using Pandoc</title>
</front>
<middle>
    &pandocMiddle;
</middle>
<back>
    <references title="Normative References">
         &rfc.2629;
    </references>
    &pandocBack;
```



```
</back>
</rfc>
```

A very minimal template.xml.

Figure 2

In this case you need to edit three documents:

1. "middle.txt" - contains the main body of text;

2. "back.txt" - holds appendices;

3. this "template.xml" - probably a fairly static file, among other things, it holds the authors and the references.

The up to date source code for Pandoc2rfc can be found at [Pandoc2rfc], this includes the stylesheet "transform.xsl" used for the transformation.

2.1. Dependencies

Pandoc2rfc needs "xsltproc" [XSLT] and "pandoc" [Pandoc] to be installed. The conversion to xml2rfc XML is done with a stylesheet based on XSLT version 1.0 [W3C.REC-xslt-19991116]. See Figure 4 in Appendix C for the contents of that stylesheet.

When using the template from Figure 2 xml2rfc version 2 must be used.

3. Building an Internet-Draft

Assuming the setup from Section 2, we can build an I-D as follows (in a Unix-like environment):

```
pandoc -st docbook middle.txt | xsltproc --nonet transform.xsl - \
    > middle.xml
pandoc -st docbook back.txt | xsltproc --nonet transform.xsl - \
    > back.xml

# Create text output
xml2rfc template.xml -f draft.txt --text

# ... or create HTML output
xml2rfc template.xml -f draft.html --html

# ... or create (self-contained) XML output
xml2rfc template.xml -f draft.xml --exp
```


Building an I-D with Pandoc2rfc.

Figure 3

Note that the output file names ("middle.xml" and "back.xml") must match the names used as the XML entities in "template.xml". The Pandoc2rfc source repository includes a shell script that incorporates the above transformations. Creating a "draft.txt" or a "draft.xml" can be done with "pandoc2rfc *.txt" and "pandoc2rfc -X *.txt" respectively.

4. Supported Features

Almost everything xml2rfc can do is supported, see Table 1 in Appendix B for a "cheat sheet".

- Sections with an anchor and title attributes
- Lists
  - style="symbols", use "* " for each item;

    * style="numbers", use digits: "1. " for each item;

    * style="empty", use "#. " for each item;

    * style="format %i", use roman lowercase numerals: "ii. ";

    * style="format (%d)", use roman uppercase numerals "II. ";

    * style="letters", use lower- or uppercase letters: "a. " and "A. " (note: two spaces);

    * style="hanging", use the Pandoc definition list syntax;

- Spanx style="verb", style="emph" and style="strong", respectively use: "`text`", "_text_" or "**text**";

- Block quote - not supported by xml2rfc, so this is converted to "<list style="hanging">" paragraph;

- Figure/artwork with an anchor and postamble (Section 6.1);

- Tables with an anchor and postamble (Section 6.2);

- References (Section 6.3)

    * external (eref);

    * internal (xref):

        + section (handled by Pandoc);

        + figures (handled by XSLT);

        + tables (handled by XSLT).

- Citations, by using internal references;

- Indexes, by (ab)using footnotes (Section 6.4).

5. Unsupported Features

- Pandoc markup in the caption for figures/artwork. Pandoc markup

for table captions _is_ supported;

- Crefs: for comments, but you can use HTML comments: "<!-- ... -->" in the Pandoc source files;

- Preamble in figures and tables.

6. Pandoc Style

In some cases the Pandoc syntax is slightly misused to get the desired output, in the following paragraphs we will detail these. Also the meta data feature (Pandoc's "Title Block" extension) of Pandoc is not used in Pandoc2rfc. This information must be put in the "template.xml".

6.1. Figure/Artwork

Indent the paragraph with 4 spaces as mandated by Pandoc. Note that xml2rfc supports a caption with "<artwork>". If you add a "@Figure: some text" as the last line, the artwork gets a "<postamble>" with the text after "@Figure: ". It will also be possible to reference the artwork. If such a caption is supplied the artwork will be centered on the page. If a caption is needed but the figure should not be centered use "@figure: ".

6.2. Tables

A table can be entered by using Pandoc's table syntax. You can choose multiple styles as input, but they all are converted to the same style (plain "<texttable>") table in xml2rfc.

The table caption is _always_ translated to a "<postamble>". The "<preamble>" tag isn't supported. If a table has a caption, it will also be possible to reference the table.

6.3. References

Pandoc provides a syntax that can be used for references. Its syntax is repeated in this paragraph. Any reference like: "[Click

here](URI)", is an external reference. An internal (i.e. see Section X) reference is typeset with: "[Click here](#localid)" or "[](#localid)".

For referencing RFCs (and other citations), you will need add the reference source in the template, as an external XML entity, Figure 2 provides an example. After that you can use: "[](#RFC2629)"

Note that referencing figures/artworks and tables is done slightly different. The reason for this is that Pandoc does not have native support/syntax for this. We work around it, by "faking" the reference in the XSLT. Thus for referencing figures/artworks and tables, you need:

- To take the first 10 characters of the caption (i.e. this is the text _after_ the string "Table: " or "@Figure: ");
- Translate spaces and single quotes ' to a minus "-";
- Translate uppercase letters to lowercase;
- For tables prefix the anchor with "tab:" and for figures use "fig:".

The figure from Section 2 will get "fig:a-very-min" as an anchor.

Note that duplicate anchors are an XML validation error which will make xml2rfc fail. These are not detected during the XSL transformation.

6.4. Indexes

The footnote syntax of Pandoc is slightly abused to support an index. Sub items are also supported. Use an exclamation mark ("!") to separate them:

```
[^1]: item!sub item
```


7. Acknowledgements

The following people have helped shape Pandoc2rfc: Benno Overeinder, Erlend Hamnaberg, Matthijs Mekking and Trygve Laugstoel.

This document was prepared using Pandoc2rfc.

8. Security Considerations

This document raises no security issues.

9. IANA Considerations

This document has no actions for IANA.

10. Normative References

[Markdown] Gruber, J., "Markdown", 2004, <http://daringfireball.net/projects/markdown/>.

[Pandoc2rfc] Gieben, R., "Pandoc2rfc git repository", October 2012, <http://github.com/miekg/pandoc2rfc>.

[Pandoc] MacFarlane, J., "Pandoc, a universal document converter", 2006, <http://johnmacfarlane.net/pandoc/>.

[RFC2629] Rose, M.T., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[W3C.REC-xslt-19991116] Clark, J., "XSL Transformations (XSLT) Version 1.0", World Wide Web Consortium Recommendation REC-xslt-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xslt-19991116>.

[XSLT] Veillard, D., "The XSLT C library for GNOME", 2006, <http://xmlsoft.org/XSLT/xsltproc2.html>.

Appendix A. Changelog

[This section should be removed by the RFC editor before publishing]

A.1. -00

1. Initial document.

A.2. -01

1. Lots of updates;

2. Added the stylesheet use in an appendix.

Appendix B. Cheat Sheet

Textual construct	Pandoc syntax	xml2rfc Syntax
Section Header	"# Section"	"<section title= ...>"
Unordered List	"* item"	"<list style="symbols">"
Unordered List	"#. item"	"<list style="empty">"
Ordered List	"1. item"	"<list style="numbers">"
Ordered List	"a. item"	"<list style="letters">"
Ordered List	"ii. item"	"<list style="format %i">"
Ordered List	"II. item"	"<list style="format (%d)">"
Ordered List	"A. item"	"<list style="format (%C)">"
Definition List	Definition	"<list style="hanging">"
Emphasis	"_text_"	"<spanx style="emph">"
Strong Emphasis	"**text**"	"<spanx style="strong">"
Verbatim	"`text`"	"<spanx style="verb">"
Block Quote	"> quote"	"<list style="hanging">"
Index	Footnotes	"<iref item="index"/>"
Table	Tables	"<texttable>"
Figure/Artwork	Code Blocks	"<figure><artwork>"
External Reference	"[Click](URI)"	"<eref target="..."/>"
Internal Reference	"](#id)"	"<xref target="id">"
Figure Reference	"](#fig:...)"	"<xref target="..."/>"
Table Reference	"](#tab:...)"	"<xref target="..."/>"
Citations	"](#RFC2119)"	"<xref target="RFC2119">"

The most important textual constructs that can be used in Pandoc2rfc.

Table 1

Appendix C. XSLT

This is the XSLT with Git version "f0cc985e291ae25a24c91934b8192930df1ad6f6", that is used for the transformation.

```
<?xml version="1.0"?>
```



```
<!-- vim: set shiftwidth=1 tabstop=1: -->
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:exsl="http://exslt.org/common" version="1.0"
 extension-element-prefixes="exsl">
<!--
    (c) Miek Gieben 2013, Licensed under the GPL version 2.
-->
  <xsl:output method="xml" omit-xml-declaration="yes"/>
  <xsl:variable name="smallcase" select="'abcdefghijklmnopqrstuvwxyz'"/>
  <xsl:variable name="uppercase" select="'ABCDEFGHIJKLMNOPQRSTUVWXYZ'"/>
  <xsl:variable name="ure" select="'igure: '"/>
  <xsl:variable name="Fig" select="concat('@F', $ure)"/>
  <xsl:variable name="fig" select="concat('@f', $ure)"/>
  <xsl:template match="/">
   <xsl:comment>
    This document was prepared using Pandoc2rfc
    https://github.com/miekg/pandoc2rfc
   </xsl:comment>
   <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="article">
   <xsl:apply-templates/>
  </xsl:template>
<!--
Remove the article info section, this should be handled
in the <front> matter of the draft
-->
  <xsl:template match="articleinfo">
  </xsl:template>
<!-- Use footnotes for indexes (iref) -->
  <xsl:template match="footnote">
   <xsl:element name="iref">
    <xsl:choose>
     <xsl:when test="contains(./para, '!')">
      <xsl:attribute name="item">
       <xsl:value-of select="substring-before (normalize-space(
        translate(./para, '&#10;', ' ')), '!')"/>
      </xsl:attribute>
```

```
         <xsl:attribute name="subitem">
          <xsl:value-of select="substring-after (normalize-space(
           translate(./para, '&#10;', ' ')), '!')"/>
         </xsl:attribute>
        </xsl:when>
        <xsl:otherwise>
         <xsl:attribute name="item">
          <xsl:value-of select="normalize-space(
           translate(./para, '&#10;', ' '))"/>
```



```
         </xsl:attribute>
        </xsl:otherwise>
       </xsl:choose>
      </xsl:element>
    </xsl:template>
  <!-- Merge section with the title tags into one section -->
    <xsl:template match="section | simplesect |
     sect1 | sect2 | sect3 | sect4 | sect5">
     <section>
      <xsl:attribute name="title">
       <xsl:value-of select="normalize-space(
        translate(./title, '&#10;', ' '))"/>
      </xsl:attribute>
      <xsl:attribute name="anchor">
       <xsl:value-of select="@id"/>
      </xsl:attribute>
      <xsl:apply-templates/>
     </section>
    </xsl:template>
  <!--
  Transform a <para> to <t>, not in lists, then it is discarded
  -->
    <xsl:template match="para | simpara">
     <xsl:choose>
      <xsl:when test="ancestor::orderedlist">
       <xsl:if test="position() &gt; 2">
        <vspace blankLines="1"/>
       </xsl:if>
       <xsl:apply-templates/>
      </xsl:when>
      <xsl:when test="ancestor::itemizedlist">
       <xsl:if test="position() &gt; 2">
```

```
        <vspace blankLines="1"/>
       </xsl:if>
       <xsl:apply-templates/>
      </xsl:when>
      <xsl:when test="ancestor::variablelist">
       <xsl:if test="position() &gt; 2">
        <vspace blankLines="1"/>
       </xsl:if>
       <xsl:apply-templates/>
      </xsl:when>
      <xsl:when test="ancestor::tbody">
       <xsl:apply-templates/>
      </xsl:when>
      <xsl:otherwise>
       <t>
        <xsl:apply-templates/>
```



```
       </t>
      </xsl:otherwise>
     </xsl:choose>
    </xsl:template>
  <!--
  Transform a <listitem> to a <t> for lists, except in description lists
  -->
    <xsl:template match="listitem">
     <xsl:choose>
      <xsl:when test="parent::varlistentry">
       <xsl:apply-templates/>
      </xsl:when>
      <xsl:otherwise>
       <t>
        <xsl:apply-templates/>
       </t>
      </xsl:otherwise>
     </xsl:choose>
    </xsl:template>
  <!--
  Transform lists, for lists in list we do not put it in a new <t></t>
  -->
    <xsl:template match="orderedlist">
     <xsl:choose>
      <xsl:when test="contains(@numeration,'arabic')">
```

```
        <xsl:choose>
         <xsl:when test="ancestor::orderedlist">
          <list style="numbers">
           <xsl:apply-templates/>
          </list>
         </xsl:when>
         <xsl:when test="ancestor::itemizedlist">
          <list style="numbers">
           <xsl:apply-templates/>
          </list>
         </xsl:when>
         <xsl:when test="ancestor::variablelist">
          <list style="numbers">
           <xsl:apply-templates/>
          </list>
         </xsl:when>
         <xsl:otherwise>
          <t>
           <list style="numbers">
            <xsl:apply-templates/>
           </list>
          </t>
         </xsl:otherwise>
```



```
        </xsl:choose>
       </xsl:when>
       <xsl:when test="contains(@numeration,'lowerroman')">
        <xsl:choose>
         <xsl:when test="ancestor::orderedlist">
          <list style="format %i.">
           <xsl:apply-templates/>
          </list>
         </xsl:when>
         <xsl:when test="ancestor::itemizedlist">
          <list style="format %i.">
           <xsl:apply-templates/>
          </list>
         </xsl:when>
         <xsl:when test="ancestor::variablelist">
          <list style="format %i.">
           <xsl:apply-templates/>
          </list>
```

```
          </xsl:when>
          <xsl:otherwise>
           <t>
            <list style="format %i.">
             <xsl:apply-templates/>
            </list>
           </t>
          </xsl:otherwise>
         </xsl:choose>
        </xsl:when>
        <xsl:when test="contains(@numeration,'upperroman')">
         <xsl:choose>
          <xsl:when test="ancestor::orderedlist">
           <list style="format (%d)">
            <xsl:apply-templates/>
           </list>
          </xsl:when>
          <xsl:when test="ancestor::itemizedlist">
           <list style="format (%d)">
            <xsl:apply-templates/>
           </list>
          </xsl:when>
          <xsl:when test="ancestor::variablelist">
           <list style="format (%d)">
            <xsl:apply-templates/>
           </list>
          </xsl:when>
          <xsl:otherwise>
           <t>
            <list style="format (%d)">
```



```
             <xsl:apply-templates/>
            </list>
           </t>
          </xsl:otherwise>
         </xsl:choose>
        </xsl:when>
        <xsl:when test="contains(@numeration,'upperalpha')">
         <xsl:choose>
          <xsl:when test="ancestor::orderedlist">
           <list style="format %C.">
            <xsl:apply-templates/>
```

```
          </list>
         </xsl:when>
         <xsl:when test="ancestor::itemizedlist">
          <list style="format %C.">
           <xsl:apply-templates/>
          </list>
         </xsl:when>
         <xsl:when test="ancestor::variablelist">
          <list style="format %C.">
           <xsl:apply-templates/>
          </list>
         </xsl:when>
         <xsl:otherwise>
          <t>
           <list style="format %C.">
            <xsl:apply-templates/>
           </list>
          </t>
         </xsl:otherwise>
        </xsl:choose>
       </xsl:when>
       <xsl:when test="contains(@numeration,'loweralpha')">
        <xsl:choose>
         <xsl:when test="ancestor::orderedlist">
          <list style="letters">
           <xsl:apply-templates/>
          </list>
         </xsl:when>
         <xsl:when test="ancestor::itemizedlist">
          <list style="letters">
           <xsl:apply-templates/>
          </list>
         </xsl:when>
         <xsl:when test="ancestor::variablelist">
          <list style="letters">
           <xsl:apply-templates/>
          </list>
```



```
         </xsl:when>
         <xsl:otherwise>
          <t>
           <list style="letters">
```

```
          <xsl:apply-templates/>
         </list>
        </t>
       </xsl:otherwise>
      </xsl:choose>
     </xsl:when>
     <xsl:otherwise>
      <xsl:choose>
       <xsl:when test="ancestor::orderedlist">
        <list style="empty">
         <xsl:apply-templates/>
        </list>
       </xsl:when>
       <xsl:when test="ancestor::itemizedlist">
        <list style="empty">
         <xsl:apply-templates/>
        </list>
       </xsl:when>
       <xsl:when test="ancestor::variablelist">
        <list style="empty">
         <xsl:apply-templates/>
        </list>
       </xsl:when>
       <xsl:otherwise>
        <t>
         <list style="empty">
          <xsl:apply-templates/>
         </list>
        </t>
       </xsl:otherwise>
      </xsl:choose>
     </xsl:otherwise>
    </xsl:choose>
   </xsl:template>
   <xsl:template match="itemizedlist">
    <xsl:choose>
     <xsl:when test="ancestor::orderedlist">
      <list style="symbols">
       <xsl:apply-templates/>
      </list>
     </xsl:when>
     <xsl:when test="ancestor::itemizedlist">
      <list style="symbols">
       <xsl:apply-templates/>
```

```
        </list>
       </xsl:when>
       <xsl:when test="ancestor::variablelist">
        <list style="symbols">
         <xsl:apply-templates/>
        </list>
       </xsl:when>
       <xsl:otherwise>
        <t>
         <list style="symbols">
          <xsl:apply-templates/>
         </list>
        </t>
       </xsl:otherwise>
     </xsl:choose>
   </xsl:template>
<!--
Hanging lists are specified as <variablelist>
-->
   <xsl:template match="variablelist">
    <xsl:choose>
     <xsl:when test="ancestor::orderedlist">
      <list style="hanging">
       <xsl:apply-templates/>
      </list>
     </xsl:when>
     <xsl:when test="ancestor::itemizedlist">
      <list style="hanging">
       <xsl:apply-templates/>
      </list>
     </xsl:when>
     <xsl:when test="ancestor::variablelist">
      <list style="hanging">
       <xsl:apply-templates/>
      </list>
     </xsl:when>
     <xsl:otherwise>
      <t>
       <list style="hanging">
        <xsl:apply-templates/>
       </list>
      </t>
     </xsl:otherwise>
    </xsl:choose>
   </xsl:template>
   <xsl:template match="varlistentry">
    <t>
```

```
      <xsl:attribute name="hangText">
```



```
       <xsl:value-of select="normalize-space(translate(./term,
        ' &#9;&#13;&#10;', ' '))"/>
      </xsl:attribute>
  <!--
  OPTION: enable this to get a newline after the hangText
  -->
  <!-- <xsl:element name="vspace"/> -->
      <xsl:apply-templates select="./listitem"/>
     </t>
    </xsl:template>
  <!--
  Transform <link> to <xref> crosslinks
  -->
    <xsl:template match="link">
     <xref>
      <xsl:attribute name="target">
       <xsl:value-of select="@linkend"/>
      </xsl:attribute>
      <xsl:apply-templates/>
     </xref>
    </xsl:template>
  <!--
  Transform <ulink> to <eref> links
  -->
    <xsl:template match="ulink">
     <eref>
      <xsl:attribute name="target">
       <xsl:value-of select="@url"/>
      </xsl:attribute>
      <xsl:apply-templates/>
     </eref>
    </xsl:template>
  <!--
  Transform <blockquote> to <list style="hanging">
  -->
    <xsl:template match="blockquote">
     <t>
      <list style="hanging" hangIndent="3">
       <xsl:apply-templates/>
      </list>
```

```
    </t>
   </xsl:template>
  <!--
  Transform <screen> and <programlisting> to <figure><artwork>
  -->
   <xsl:template match="screen | programlisting">
    <figure>
     <xsl:choose>
```



```
       <xsl:when test="contains(., $Fig)">
        <xsl:attribute name="anchor">
         <xsl:text>fig:</xsl:text>
         <xsl:value-of select="translate(
          translate(substring(normalize-space(
          translate( substring-after(., $Fig) ,
          &quot;&#10;'&quot;, &quot;  &quot;)), 1, 10),
          &quot; &quot;, &quot;-&quot;), $uppercase, $smallcase)"/>
        </xsl:attribute>
  <!-- If there is an caption, center the figure -->
           <xsl:attribute name="align">
            <xsl:text>center</xsl:text>
           </xsl:attribute>
           <artwork>
            <xsl:value-of select="substring-before(., $Fig)"/>
           </artwork>
           <postamble>
            <xsl:value-of select="substring-after(., $Fig)"/>
           </postamble>
          </xsl:when>
          <xsl:when test="contains(., $fig)">
           <xsl:attribute name="anchor">
            <xsl:text>fig:</xsl:text>
            <xsl:value-of select="translate(
             translate(substring(normalize-space(
             translate( substring-after(., $fig) ,
             &quot;&#10;'&quot;, &quot;  &quot;)), 1, 10),
             &quot; &quot;, &quot;-&quot;), $uppercase, $smallcase)"/>
           </xsl:attribute>
           <artwork>
            <xsl:value-of select="substring-before(., $fig)"/>
           </artwork>
           <postamble>
```

```
      <xsl:value-of select="substring-after(., $fig)"/>
     </postamble>
    </xsl:when>
    <xsl:otherwise>
     <artwork>
      <xsl:value-of select="."/>
     </artwork>
    </xsl:otherwise>
   </xsl:choose>
  </figure>
 </xsl:template>
 <xsl:template match="title"/>
 <xsl:template match="literal">
  <spanx style="verb">
   <xsl:apply-templates/>
```



```
  </spanx>
 </xsl:template>
 <xsl:template match="emphasis">
  <xsl:choose>
   <xsl:when test="contains(@role,'strong')">
    <spanx style="strong">
     <xsl:apply-templates/>
    </spanx>
   </xsl:when>
   <xsl:otherwise>
    <spanx style="emph">
     <xsl:apply-templates/>
    </spanx>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:template>
 <xsl:template match="literal" mode="post">
  <spanx style="verb">
   <xsl:apply-templates mode="post"/>
  </spanx>
 </xsl:template>
 <xsl:template match="emphasis" mode="post">
  <xsl:choose>
   <xsl:when test="contains(@role,'strong')">
    <spanx style="strong">
     <xsl:apply-templates mode="post"/>
```

```
          </spanx>
         </xsl:when>
         <xsl:otherwise>
          <spanx style="emph">
           <xsl:apply-templates mode="post"/>
          </spanx>
         </xsl:otherwise>
        </xsl:choose>
       </xsl:template>
  <!-- Tables -->
    <xsl:template match="table | informaltable">
     <texttable>
  <!-- If there is a caption, fake an anchor attribute -->
       <xsl:if test="./caption">
        <xsl:attribute name="anchor">
         <xsl:text>tab:</xsl:text>
         <xsl:value-of select="translate(
          translate(substring(normalize-space(translate(
          ./caption, &quot;&#10;'&quot;, &quot;  &quot;)), 1, 10),
          &quot; &quot;, &quot;-&quot;), $uppercase, $smallcase)"/>
        </xsl:attribute>
       </xsl:if>
```



```
       <xsl:if test="./title">
        <xsl:attribute name="anchor">
         <xsl:text>tab:</xsl:text>
         <xsl:value-of select="translate( translate(
          substring(normalize-space(translate(./title,
          &quot;&#10;'&quot;, &quot;  &quot;)), 1, 10),
          &quot; &quot;, &quot;-&quot;), $uppercase, $smallcase)"/>
        </xsl:attribute>
       </xsl:if>
       <xsl:apply-templates/>
       <xsl:if test="./title">
  <!-- create postamble of the title -->
         <postamble>
          <xsl:apply-templates select="./title" mode="post"/>
         </postamble>
        </xsl:if>
        <xsl:if test="./caption">
  <!-- create postamble of the caption -->
         <postamble>
```

```
          <xsl:apply-templates select="./caption" mode="post"/>
         </postamble>
        </xsl:if>
       </texttable>
      </xsl:template>
<!-- Table headers -->
  <xsl:template match="table/thead/tr/th |
   informaltable/thead/tr/th">
   <ttcol>
    <xsl:attribute name="align">
     <xsl:value-of select="@align"/>
    </xsl:attribute>
<!--
Every even position() need to be dealt with:
2 look back to 1, 4 look back to 2, etc.
-->
      <xsl:if test="position() = 2">
       <xsl:call-template name="get_col">
        <xsl:with-param name="column" select="1"/>
       </xsl:call-template>
      </xsl:if>
      <xsl:if test="position() = 4">
       <xsl:call-template name="get_col">
        <xsl:with-param name="column" select="2"/>
       </xsl:call-template>
      </xsl:if>
      <xsl:if test="position() = 6">
       <xsl:call-template name="get_col">
        <xsl:with-param name="column" select="3"/>
```



```
       </xsl:call-template>
      </xsl:if>
      <xsl:if test="position() = 8">
       <xsl:call-template name="get_col">
        <xsl:with-param name="column" select="4"/>
       </xsl:call-template>
      </xsl:if>
      <xsl:if test="position() = 10">
       <xsl:call-template name="get_col">
        <xsl:with-param name="column" select="5"/>
       </xsl:call-template>
      </xsl:if>
```

```
        <xsl:if test="position() = 12">
         <xsl:call-template name="get_col">
          <xsl:with-param name="column" select="6"/>
         </xsl:call-template>
        </xsl:if>
        <xsl:if test="position() = 14">
         <xsl:call-template name="get_col">
          <xsl:with-param name="column" select="7"/>
         </xsl:call-template>
        </xsl:if>
        <xsl:if test="position() = 16">
         <xsl:call-template name="get_col">
          <xsl:with-param name="column" select="8"/>
         </xsl:call-template>
        </xsl:if>
        <xsl:apply-templates/>
       </ttcol>
      </xsl:template>
      <xsl:template name="get_col">
       <xsl:param name="column"/>
       <xsl:if test="../../../col[$column]">
        <xsl:attribute name="width">
         <xsl:value-of select="../../../col[$column]/@width"/>
        </xsl:attribute>
       </xsl:if>
      </xsl:template>
  <!-- Table headers for CALS tables -->
    <xsl:template match="table/tgroup/thead/row/entry">
     <ttcol>
      <xsl:if test="position() = 2">
       <xsl:call-template name="get_colspec">
        <xsl:with-param name="column" select="1"/>
       </xsl:call-template>
      </xsl:if>
      <xsl:if test="position() = 4">
       <xsl:call-template name="get_colspec">
```



```
        <xsl:with-param name="column" select="2"/>
       </xsl:call-template>
      </xsl:if>
      <xsl:if test="position() = 6">
       <xsl:call-template name="get_colspec">
```

```
        <xsl:with-param name="column" select="3"/>
       </xsl:call-template>
      </xsl:if>
      <xsl:if test="position() = 8">
       <xsl:call-template name="get_colspec">
        <xsl:with-param name="column" select="4"/>
       </xsl:call-template>
      </xsl:if>
      <xsl:if test="position() = 10">
       <xsl:call-template name="get_colspec">
        <xsl:with-param name="column" select="5"/>
       </xsl:call-template>
      </xsl:if>
      <xsl:if test="position() = 12">
       <xsl:call-template name="get_colspec">
        <xsl:with-param name="column" select="6"/>
       </xsl:call-template>
      </xsl:if>
      <xsl:if test="position() = 14">
       <xsl:call-template name="get_colspec">
        <xsl:with-param name="column" select="7"/>
       </xsl:call-template>
      </xsl:if>
      <xsl:if test="position() = 16">
       <xsl:call-template name="get_colspec">
        <xsl:with-param name="column" select="8"/>
       </xsl:call-template>
      </xsl:if>
  <!-- If the entry itself has align, we use that -->
       <xsl:if test="@align">
        <xsl:attribute name="align">
         <xsl:value-of select="@align"/>
        </xsl:attribute>
       </xsl:if>
       <xsl:apply-templates/>
      </ttcol>
     </xsl:template>
     <xsl:template name="get_colspec">
      <xsl:param name="column"/>
      <xsl:if test="../../../../tgroup/colspec[$column]">
       <xsl:attribute name="align">
        <xsl:value-of
         select="../../../../tgroup/colspec[$column]/@align"/>
```

```
      </xsl:attribute>
<!-- Optionally colwidth, translate * to % -->
      <xsl:if test="../../../../tgroup/colspec[$column]/@colwidth">
       <xsl:attribute name="width">
        <xsl:value-of select="translate(
         ../../../../tgroup/colspec[$column]/@colwidth, '*', '%')"/>
       </xsl:attribute>
      </xsl:if>
     </xsl:if>
    </xsl:template>
<!-- Table headers for CALS tables, Pandoc 1.9.x+ -->
  <xsl:template match="informaltable/tgroup/thead/row/entry">
   <ttcol>
    <xsl:if test="position() = 2">
     <xsl:call-template name="get_colspec_informal">
      <xsl:with-param name="column" select="1"/>
     </xsl:call-template>
    </xsl:if>
    <xsl:if test="position() = 4">
     <xsl:call-template name="get_colspec_informal">
      <xsl:with-param name="column" select="2"/>
     </xsl:call-template>
    </xsl:if>
    <xsl:if test="position() = 6">
     <xsl:call-template name="get_colspec_informal">
      <xsl:with-param name="column" select="3"/>
     </xsl:call-template>
    </xsl:if>
    <xsl:if test="position() = 8">
     <xsl:call-template name="get_colspec_informal">
      <xsl:with-param name="column" select="4"/>
     </xsl:call-template>
    </xsl:if>
    <xsl:if test="position() = 10">
     <xsl:call-template name="get_colspec_informal">
      <xsl:with-param name="column" select="5"/>
     </xsl:call-template>
    </xsl:if>
    <xsl:if test="position() = 12">
     <xsl:call-template name="get_colspec_informal">
      <xsl:with-param name="column" select="6"/>
     </xsl:call-template>
    </xsl:if>
    <xsl:if test="position() = 14">
     <xsl:call-template name="get_colspec_informal">
      <xsl:with-param name="column" select="7"/>
     </xsl:call-template>
    </xsl:if>
```

```
       <xsl:if test="position() = 16">
        <xsl:call-template name="get_colspec_informal">
         <xsl:with-param name="column" select="8"/>
        </xsl:call-template>
       </xsl:if>
   <!-- If the entry itself has align, we use that -->
        <xsl:if test="@align">
         <xsl:attribute name="align">
          <xsl:value-of select="@align"/>
         </xsl:attribute>
        </xsl:if>
        <xsl:apply-templates/>
       </ttcol>
      </xsl:template>
      <xsl:template name="get_colspec_informal">
       <xsl:param name="column"/>
       <xsl:if test="../../../../tgroup/colspec[$column]">
        <xsl:attribute name="align">
         <xsl:value-of
          select="../../../../tgroup/colspec[$column]/@align"/>
        </xsl:attribute>
   <!-- Optionally colwidth, translate * to % -->
        <xsl:if test="../../../../tgroup/colspec[$column]/@colwidth">
         <xsl:attribute name="width">
          <xsl:value-of
           select="translate(
           ../../../../tgroup/colspec[$column]/@colwidth,
           '*', '%')"/>
         </xsl:attribute>
        </xsl:if>
       </xsl:if>
      </xsl:template>
      <xsl:template match="table/tbody/tr/td |
       informaltable/tbody/tr/td |
       table/tgroup/tbody/row/entry |
       informaltable/tgroup/tbody/row/entry">
       <c>
        <xsl:apply-templates/>
       </c>
      </xsl:template>
   <!-- CALS table -->
     <xsl:template match="table/tbody/row/entry">
```

```
      <c>
       <xsl:apply-templates/>
      </c>
     </xsl:template>
    </xsl:stylesheet>
```


XSLT used for the transformation.

Figure 4

Author's Address

R. (Miek) Gieben
Google

Email: miek@google.com