S1000D XSL Stylesheets

README

Содержание

Список таблиц

References

Таблица 1. References

Data Module/Technical publication	Title
None	

1 Introduction

S1000D [1] is an XML based documentation standard for technical publications (технических публикаций). Originally created by a conglomerate of defence companies for documenting military hardware, it is now also being used by civilian aerospace companies. S1000D XSL Stylesheets [2] is a set of XSL stylesheets that can be used to transform an S1000D document (authored as a bunch of XML files) into a PDF file for viewing and printing.

The specification is still evolving and has already has gone through quite a few revisions. The current released issue is 4.0.1 and 4.1 is due sometime soon. The specification decrees not only how a document's content is to be structured but also how the resulting publication should look. The fact that the current issue runs to more than 2750 pages gives an indication of the complexity of the standard. Furthermore, companies that create S1000D documents can apply company specific "Business Rules" that affect both the information content and/or presentation style of the resulting document.

[1] http://www.s1000d.org
[2] http://github.com/smartavionics/S1000D-XSL-Stylesheets

S1000D authoring/publishing systems are available but tend to be geared towards the larger companies that are producing 10,000's of pages of documentation. So at the moment, if you want to produce a modest sized S1000D document you have to either invest (rather heavily) in an authoring/publishing system or pay someone who has the capability to produce S1000D documents to create the document for you. Either way, it's expensive. The goal of the S1000D XSL stylesheet project is to provide a low-cost (in terms of the software cost) means to create page-oriented output (PDF files) from S1000D compliant XML source files.

The S1000D standard describes how the XML **data modules** that make up a given publication's content are stored in a "Common Source DataBase" (CSDB). They don't actually say how the database is implemented or how the data modules are authored and stored in the database but the basic idea is that when a publication is to be created, the required modules are extracted from the database and formatted along with some (possibly automatically generated) front matter to produce the desired result. The content of a given publication is determined by one or more **publication modules** which reference the data modules to be included.

Commercial S1000D publishing systems provide some form of Content Management System (CMS) to act as the CSDB and that would provide versioning and multi-author access facilities. For a small documentation project, the complicated and expensive CMS can be replaced with a decent Software Configuration Management (SCM) system and the modules stored simply as flat files. As described in more detail below, the S1000D XSL stylesheets simply require that a publication's data modules and publication module are combined to make a single XML file which is then processed - how those modules are stored and retrieved is not really relevant.

2 Using the S1000D XSL stylesheets

Generating a PDF file from this top-level XML file is a three stage process:

1 Transform the S1000D XML into DocBook XML using the **s1000dtodb** stylesheet.

2 Transform the DocBook XML into Formatting Objects (FO) XML using the **dbtofo** stylesheet.

3 Process the FO XML using a formatting program such as **fop** or **xep** to produce the PDF file.

An XSLT version 1 compatible processor is required to carry out the transformations. The stylesheets have been tested with the **xalan** and **saxonpe** processors. An **ant** build file is provided that will use either of those programs to do the transformations and either fop or xep to do the formatting. Once this process has been configured, the document can be rebuilt anytime by simply running ant.

3 Requirements

To use the S1000D XSL stylesheets, you need the following items:

S1000D XSL stylesheets	Obtain from the github S1000D-XSL-Stylesheets [4] repository.
Docbook XSL stylesheets	Obtain from SourceForge [6].
	The dbtofo stylesheet is just a customisation layer on top of the standard DocBook XSL stylesheets so you need those stylesheets to use dbtofo. Get the namespace aware version,

[4] https://github.com/smartavionics/S1000D-XSL-Stylesheets
[6] http://sourceforge.net/projects/docbook/files/docbook-xsl-ns/

e.g. docbook-xsl-ns-1.76.1 and unpack the distribution in the top-level directory of the S1000D XSL tree.

An XSLT processor	If you're going the Java route, there's several to choose from - known to work are: – Xalan2 [7] – Saxon HE [8] or Saxon PE & EE [9] The open source version (he) of Saxon is fine but it doesn't support the XSL extension currently used in the stylesheets to resolve graphic file entities. If you use the commercial editions of Saxon (saxon pe, saxon ee) then the extension is supported. See Para 4.3 for more information on this. A non-Java XSLT processor that is known to work is **xsltproc** [10]
Xerces XML Parser	Xerces2 Java [11] is required when using either the xalan or saxon XSLT processors.
resolver.jar	Again, only needed if you are using a Java XSLT processor. It can be found in xml-commons-resolver-latest.zip [12].
FO Processor	There are numerous commercial products that will generate PDF (and other formats) from XML FO input. The stylesheets have been tested with XEP from RenderX [13] product. A free alternative is Apache Fop [14]. Version 1 works well with the stylesheets but has some limitations:
Apache ant	If you want to use the supplied ant build file, you need ant which may well be available as a package on your system or it can be obtained from Apache ant [15].

The ant build file needs to know where the various jars (xalan/saxon, xerces, resolver) reside so it will make configuration easier if those jars are put into the same directory but it's not an absolute requirement.

4 Configuration

Примечание

This section assumes you are using ant with the supplied **build.xml** file.

[7] http://xml.apache.org/xalan-j/
[8] http://saxon.sourceforge.net/
[9] http://www.saxonica.com/
[10] http://xmlsoft.org/XSLT/xsltproc2.html
[11] http://xerces.apache.org/
[12] http://www.apache.org/dist/xml/commons/xml-commons-resolver-latest.zip
[13] http://www.renderx.com/tools/xep.html
[14] http://xmlgraphics.apache.org/ /
[15] http://ant.apache.org

4.1 Configuring the build

Configuration mainly consists of setting those build parameters that you wish to change from the default values. To do that, don't edit build.xml itself but, instead, edit **build.properties**. The most important properties you can set are listed in Таблица 2.

Таблица 2. Configurable build properties

Property	Required/Optional	Value
docname	Required	Name of your top-level XML file without the .xml extension, e.g. `S1000DXSL-README`
s1000d.xsl.home	Required	Name of the top-level directory of the S1000D XSL distribution. Can be a relative or absolute path name.
jars.dir	Optional	Name of the directory in which the jar files are stored. Can be a relative or absolute path name - defaults to '.'
resolver.jar	Optional	Name of the resolver jar file - defaults to `${jars.dir}/resolver.jar`
saxon.jar	Optional	Name of the Saxon jar file - defaults to `${jars.dir}/saxon9he.jar`
xalan.jar	Optional	Name of the Xalan jar file - defaults to `${jars.dir}/xalan2.jar`
xercesimpl.jar	Optional	Name of the Xerces impl jar file - defaults to `${jars.dir}/xercesImpl.jar`
xsltprog	Optional	Name of the XSLT processor to use (`xalan` or `saxon`) - defaults to `xalan`
formatter	Optional	Name of the FO formatter to use (`fop1` or `xep`) - defaults to `fop1`
fop1.cmd xep.cmd	Optional	The actual command to be run to execute the FO formatter - defaults are `fop1` and `xep` respectively.

4.2 Catalog configuration

For the XSLT processor to be able to find the DocBook stylesheets, the catalog file (**catalog.xml**) in the S1000D XSL installation directory needs to have a suitable entry. The supplied file looks like this:

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

<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">

   <nextCatalog catalog="docbook-xsl-ns-1.76.1/catalog.xml"/>
   <nextCatalog catalog="../catalog.xml"/>

</catalog>
```

The first **nextCatalog** element should reference the catalog in the DocBook XSL distribution. Make sure that the pathname there matches where your DocBook XSL files are located.

4.3 Graphics entity map configuration

S1000D documents reference all graphics through XML entities. At some stage in the processing, the entity has to be mapped into a filename which is then passed through to the FO processor so it can include the graphic in the output. Entities can be defined in a local document type definition at the top of a module that uses them. Here's an example:

```
<!NOTATION cgm
  PUBLIC "-//USA-DOD//NOTATION Computer Graphics Metafile//EN">
<!ENTITY ICN-S1000DBIKE-AAA-DA53000-0-U8025-00525-A-04-1 SYSTEM
  "./illustrations/ICN-S1000DBIKE-AAA-DA53000-0-U8025-00525-A-04-1.CGM"
  NDATA cgm>
```

The first line declares `cgm` to be a type of external entity and the second line declares `ICN-S1000DBIKE-AAA-DA53000-0-U8025-00525-A-04-1` to be an entity of that type that references a CGM graphics file at location `./illustrations/ICN-S1000DBIKE-AAA-DA53000-0-U8025-00525-A-04-1.CGM`. So now, within the data module content, you can use `ICN-S1000DBIKE-AAA-DA53000-0-U8025-00525-A-04-1` to specify that graphics file.

The XSLT processor is potentially capable of resolving the entity name into the file name but to do so it must have seen the declarations shown above and for that to happen, it must be using a validating XML parser. At this time, the build file is using a non-validating parser and so when the XSLT processor resolves the graphics entity name it comes out null. Therefore, I have implemented a simple XSL extension function that is used to resolve the graphics entity names.

The way it works is as follows: within the modules that wish to refer to a graphics file you declare a suitable entity for the file - the type and value of the entity are not important as they will both be ignored when the module is processed. Here's the declaration that is used within the PMC for this document that declares the entity that gets used as the company logo in the page headers:

```
<!NOTATION anything SYSTEM "">
<!ENTITY publisher-logo SYSTEM "" NDATA anything>
```

This declares `publisher-logo` to be an entity which we can use within the module to specify a graphics file - it gets used like this:

```
<logo>
  <symbol infoEntityIdent="publisher-logo" reproductionHeight="12mm"/>
</logo>
```

To achieve the desired result, we still have to specify somewhere that `publisher-logo` maps into a particular file name (in this case the file is called `smartavionics-logo.svg`). This mapping of entity names to file names is done with a simple property file called `info-entity-map.txt` that contains one line for each graphics entity you want to resolve. The example file looks like this:

```
publisher-logo = smartavionics-logo.svg
```

If the property file cannot be found or an entity doesn't have an entry in the file, the entity name is passed through as the file name unchanged, i.e. entity name 'foo' maps into 'foo'. The DocBook stylesheets will automatically append a default file extension if a graphics file name has no extension. The default is '.png' so entity 'foo' will become 'foo.png'.

4.3.1 Caveats

– This is all rather experimental and could well change in the future.

– The S1000D standard specifies that all vector graphic files are in CGM format and all bitmap graphic files are in TIFF format. Unfortunately CGM format is not supported by either of the FO processors I have access to and so they cannot be used directly. Potentially, the publishing system could convert CGM files to, say, SVG or EPS for inclusion in the document but as I don't have any means of editing CGM files anyway this capability is not high on my list of improvements.

– The Home Edition of the Saxon XSLT processor does not support the above mentioned extension and so the entity name will get passed through unchanged as described above.

5 Cranking the handle

Before processing, you need to check that the input is valid S1000D XML. If the input isn't valid, expect big trouble. I strongly recommend using an XML aware editor that has validating capability. I am currently using jedit (free) and oxygen (not-free) and many others are available[16].

Once the source is validated, it's just a matter of running 'ant' within the directory containing the build.xml file. You can either do this from a command line or if you are using an editor like jedit you can run the build from within the editor.

It is normal to get a few messages, here's some typical output (unfortunately, some of the lines are long and I have had to wrap them):

```
Buildfile: build.xml

s1000d_to_db:

check.transform.required:

transform.using.xalan:
     [echo] Transforming S1000DXSL-README.xml to S1000DXSL-README-db.xml

db_to_fo:

check.transform.required:

transform.using.xalan:
     [echo] Transforming S1000DXSL-README-db.xml to S1000DXSL-README.fo
     [java] file:/home/smartavionics/S1000D/S1000D_xsl/docbook-xsl-ns-1.76.1/
fo/docbook.xsl; Line #318; Column #16; Making portrait pages on A4 paper
(210mmx297mm)

fo_to_pdf:

check.format.required:

format:

format.using.fop1:
     [exec] 28-Jan-2011 10:35:48 org.apache.fop.events.LoggingEventListene
```

[16] Emacs NXML mode would be usable if the S1000D xsd schemas were available as rnc schemas but I haven't yet succeeded in generating usable rnc files from the xsd files.

```
processEvent
     [exec] WARNING: Font "Symbol,normal,700" not found. Substituting with
"Symbol,normal,400".
     [exec] 28-Jan-2011 10:35:48 org.apache.fop.events.LoggingEventListener
processEvent
     [exec] WARNING: Font "ZapfDingbats,normal,700" not found. Substituting
with "ZapfDingbats,normal,400".
     [exec] 28-Jan-2011 10:35:48 org.apache.fop.events.LoggingEventListener
processEvent
     [exec] WARNING: Line 1 of a paragraph overflows the available area by
802 millipoints. (See position 286:377)
     [move] Moving 1 file to /home/smartavionics/S1000D/S1000D_xsl/sample

build:

BUILD SUCCESSFUL
Total time: 18 seconds
```

If you use an S1000D element that is not yet implemented by the stylesheets you will get a message like this:

```
Unhandled: publication/dmodule/content/faultReporting
```

and the element and its content will be copied through to the output verbatim and displayed in red.

6 Future work

There's still much to do:

– Many elements in the S1000D schema are not yet implemented. I will be implementing elements as and when I need them but if you want to use these stylesheets and need particular elements implementing, let me know. Better still, if you know some XSL, have a go at implementing the elements yourself (all contributions are welcome).

– No attempt has been made to support 'applicability'. I believe that the 4.1 issue of S1000D will have something to say re the formatting aspects of applicability so I am holding off for a while.

– At the present time, the generation of a title page is problematic. S1000D does not currently support any markup to explicitly lay out the elements of a title page. I believe that this may be addressed in 4.1 but in the meantime, one can "cheat" by directly embedding either FO or DocBook elements in the S1000D modules and they will get passed through (thanks to the power of XML namespaces!)

– At this time, if the document contains a "List of effective data modules" module (indicated by the module having an `infoCode` value of `00S`), the stylesheets will automatically generate the content for the module from the data modules that are referenced by the PM. In the future, other front-matter may well be automatically generated.

7 Acknowledgements

This project has been made much easier by the efforts of other people and I would like to say thanks to:

– The creators/maintainers of the DocBook XSL stylesheets and the associated documentation - not only have I used those stylesheets as the basis of the formatting process but I have also learnt a lot about XSL stylesheet writing by looking at that code.

– The creators/maintainers of the various open source tools that are used (xalan, xerces, saxonhe, fop, etc.)

8 Warranty

The file COPYING contains the full warranty and copyright information but the bottom line is that these stylesheets (and any other files in this package) are supplied with no warranty as to fitness of purpose, etc. In particular, the stylesheets are not guaranteed to produce output that is free from errors or omissions. So if you write a document that states something like "ensure that the fuel tank contains at least 10,000L of fuel" and it comes out saying "ensure that the fuel tank contains at least 10L of fuel", Smart Avionics Ltd. will not be liable for the consequences.

End of data module