Consodoc publication server user's guide

Contents

1 Introduction

Consodoc is a system which automates XML publishing. The base use case is converting XML to beautiful PDF through TeXML and LaTeX, but Consodoc is adaptable to other processes as well.

Exceptional features:

- XML is converted to PDF through TeXML and LaTeX.
- It's possible to manually tune the generated layout.
- Tunings are remembered and re-applied during re-generations.
- Consodoc is based on the popular build system SCons.

Consodoc stands for "**cons**tructor **o**f **doc**umentation". Consodoc home page is http://consodoc.com/.

2 Quick start

After installing Consodoc, enter the folder `examples/guide`. In contains the following files:

- `in/guide.xml`: this User's Guide in XML format
- `in/guide.patch`: patch for the LaTeX file
- `in/guide.pdf`: expected PDF result
- `support/guide.xsl`: XSLT script to convert XML to TeXML
- `support/guide.cls`: LaTeX class file

To build PDF, say

```
$ cdoc pdf
```

or just

```
$ cdoc
```

If the build is finished successfully, the folder `out` contains the file `guide.pdf`. The folder `tmp` contains temporary files.

View the PDF file `out/guide.pdf`.

To delete the folders `tmp` and `out`, say

```
$ cdoc -c pdf
```

or just

```
$ cdoc -c
```

It's possible to build only an intermediate file, without going further. Use the short name of a corresponding step. For example, to only preprocess XML, say:

```
$ cdoc pp
```

3 Creating new projects

Create a new directory and a set of subdirectories. Recommended layout:

- `in/`: input files, such as source XML documents
- `out/`: output files, such as final PDF documents (created automatically)
- `tmp/`: temporary files (created automatically)
- `support/`: support files, such as XSLT scripts and LaTeX class files.

Create the project file `SConstruct`:

- specify the input XML file (parameter `in_file`) and XSLT script (parameter `in_xslt`) for converting from XML to TeXML,
- set other options and paths to files.

The build process is defined using the standard scons build files. To learn how to use scons and its build files, refer to the "SCons User Guide". The tool `cdoc` itself is just a wrapper for the tool `scons`.

Here is a typical `SConstruct` file for Consodoc:

```
import Consodoc
Consodoc.default_process(
 in_file = 'in/text.xml',
 in_xslt = 'support/convert.xsl'
)
```

4 Suggested workflow

From the high-level point of view, the development process is three-step:

- get some result,
- get a good result,
- get the beautiful result.

At the beginning, the first step might be hard, as the XSLT code for converting XML to TeXML might be with faults, and you get incorrect LaTeX code.

If you are lucky, the only errors you get are the missed images. To get a summary of the missed files, issue the following command:

```
$ cdoc missed
```

To get some result, you might prefer to play with the LaTeX code instead of fixing XSLT. Time to time, you want to re-check log files for the list of errors. The corresponding command is:

```
$ cdoc errors
```

If the LaTeX code doesn't compile under Consodoc, check the environment which Consodoc uses to run LaTeX. To get the command prompt with this environment, issue the following command:

```
$ cdoc texenv
```

After you get some result, check the PDF for layout problems. If you can fix the problems by improving the XSLT code, do it. However, some problems can be fixed only manually. The recommended approach is to use the DVI file. To get one, issue the following command:

```
$ cdoc dvi
```

Open the DVI file from the folder `tmp` in the `xdvi` viewer. If it doesn't open due to unresolved dependencies on a custom `TEXMF` tree, ajust the environment:

```
$ cdoc texenv2
```

When you see a problem in the DVI file, use the combination `Ctrl-Mouse1` to make `xdvi` open an editor. The editor will load the LaTeX code and position the cursor near the source of the problem.

While correcting the LaTeX code, remember the changes you made:

```
$ cdoc patch
```

After you get a good result, start tuning it. The log file should contain a lot of warnings about overfull boxes. To re-display the warnings, issue the command:

```
$ cdoc warnings
```

The most often used LaTeX commands to tune the layout are: `\enlargethispage`, `\newpage`, `\hfuzz`, `\\`, `\discretionary`.

PDF and DVI files are re-generated each time you changed the sources, but Consodoc doesn't track external dependencies, such as changes in the custom `TEXMF` tree. If you want to force re-generation of the DVI or PDF, issue the following commands:

```
$ cdoc repdf
$ cdoc redvi
```

When you are satisfied with the layout, remember the changes you made:

```
$ cdoc patch
```

If you use a version control system, put the patch file under its control.

5 Before describing details

Terminology.

- **Step**. An atomic action, which either successful, either failed.
- **Process**. A set of steps.
- **Project**. A definition of a process, options, input and output files and everything other to build the documentation.

The text refers to the variabes defined in the scons build files. These references look so:

- Simple reference: `$(variable_name)`.
- Several variables at once: `$(tmpdir)/$(filename)`.
- There are also functions in the form `$(function parameters)`: `$(basename $(in_file))`.

Three functions are defined:

- `nodir` returns the file name without the directory. For example, `$(nodir in/guilde.xml)` results in "`guide.xml`".
- `basename` returns the file name without the directory and the extension. For example, `$(basename in/guilde.xml)` results in "`guide`".
- `noext` returns the file name without th extension. For example, `$(noext in/guilde.xml)` results in "`in/guide`".

Note that this syntax is used only in this Guide. Actual substitutions in the scons build files look different.

6 Common parameters

Main parameters:

`in_file`: input XML file. Must be set by the user.

`in_xslt`: XSLT program to convert from XML to TeXML. Must be set by the user.

`in_patch`: patch for the LaTeX file. Default: `$(noext $(in_file)).patch`.

Directories:

`indir`: input files. Default: `in`.

`outdir`: output files. Default: `out`.

`tmpdir`: temporary files. Default: `tmp`.

`supportdir`: support files. Default: `support`.

`basedir`: base directory for `$(indir)` and family. Default: absolute path to the scons build file.

Programs:

`xmllint`. Default: `xmllint`.

`xsltproc`. Default: `xsltproc`.

`texml`. Default: `texml`.

`diff`. Default: `diff`.

`patch`. Default: `patch`.

`dvilatex`. Default: `latex`.

`pdflatex`. Default: `pdflatex`.

7 Predefined steps

7.1 Preprocess the input

Short name: “`pp`”.

Parameters:

`pp_in`: input XML file. Default: `$(in_file)`.

`pp_out`: output XML file. Default: `$(tmpdir)/$(nodir $(pp_in))`.

Action:

Resolve all entites and XIncludes and save the result as `$(pp_out)`.

7.2 TeXML from XML

Short name: “`texml`”.

Parameters:

`texml_in`: input XML file. Default: `$(pp_out)`.

`texml_out`: output TeXML file.
Default: `$(tmpdir)/$(basename $(texml_in)).texml`.

`texml_xslt`: XSLT script to convert XML to TeXML. Default: `$(in_xslt)`.

`texml_params`: an associative array of parameters to the XSLT program. Default: `None`.

Action:

Run XSLT processor to transform `$(texml_in)` to `$(texml_out)` with help of the XSLT program `$(texml_xslt)`.

7.3 pre-[La]TeX from TeXML

Short name: “`pretex`”.

Parameters:

`pretex_in`: input TeXML file. Default: `$(texml_out)`.

`pretex_out`: output LaTeX file.
Default: `$(tmpdir)/$(basename $(pretex_in)).tex.orig`.

`pretex_params`: list of parameters, for example:
“`['--encoding utf8', '-a']`”. Default: `None`.

Action:

Execute the TeXML processor to convert `$(pretex_in)` to `$(pretex_out)`.

7.4 Patch work

7.4.1 Introduction

Automatically generated documentation might contain problematic places, in which manual re-layout is desired. Consodoc is tailored to support manual intervention on the level of LaTeX files.

Obviously, the system mustn't lose the user's changes when re-generating PDF. Consodoc remembers and applies changes using the diff and patch mechanism.

Summary of the situations:

- There is no patch, so nothing to apply
- Patch is applied:
  - successfully
  - with errors
- The user has made changes, and the patch file should be generated

This functionality adds complexity to the build process. A set of supporting files is used to assist the patch step:

- `$(patch_error)` is a lock-file to prevent further steps in case of patch errors.
- `$(patch_tex)` and `$(patch_tex_copy)` are used to detect if the user has manually edited the LaTeX code.
- `$(patch_tex)` and `$(patch_tex_orig)` are used to generate a patch file.

For convenience of describing the process, the step "Patch Work" is splitted on three virtual steps:

- "Forward Order",
- "Reverse order", and
- "Problems".

7.4.2 Patch work, common

Parameters:

`patch_file`: the patch file. Default: `$(in_patch)`.

`patch_tex_orig`: the TeX file without user changes, as generated on the step “LaTeX from TeXML”. Default is `$(pretex_out)`.

`patch_tex_copy`: the TeX file without user changes, after copying or applying the patch to `$(pretex_out)`. Default is `$(patch_tex).copy`.

`patch_tex`: the TeX file with user changes. Default: `$(tmpdir)/(basename $(patch_tex_orig)`. For example, if `$(patch_text_orig)` is `file.tex.orig`, then `$(patch_tex)` is `file.tex`.

`patch_tex_rej`: the list of failed patch hunks. Default, can’t be changed: `$(patch_tex).rej`.

`patch_error`: lock-file to indicate patching failure. Default:
`$(tmpdir)/$(basename $(patch_file)).patch-lock`

`patch_error_message`: text of notification about errors. Default: “File `$(patch_tex)` is patched with problems. Correct the rejections manually, delete the file `$(patch_error)` and re-run generation.”

`patch_reminder_message`: text of notification about user-made changes. Default: “TeX code in `$(patch_tex)` was changed. Consider generating a patch.”

7.4.3 Patch work, problems

This step is a substep of the steps “Patch Work, Forward Order”, “Patch Work, Reverse Order” and “DVI or PDF from LaTeX”.

If the file `$(patch_error)` does exist, notify the user about the problem using the message `$(patch_error_message)`. Stop the generation process.

7.4.4 Patch work, forward order

Short name: “`tex`”.

If `$(patch_tex_copy)` already exists, check that content of `$(patch_tex)` is the same. If not, print the warning `$(patch_reminder_message)`.

If `$(patch_file)` doesn’t exist:

- copy `$(patch_tex_orig)` to `$(patch_tex)` and `$(patch_tex_copy)`,

If `$(patch_file)` exists:

- apply the patch `$(patch_file)` to `$(patch_tex_orig)` and save the result as `$(patch_tex)`.
- If error, create the file `$(patch_error)` and execute the step “Patch Work, Problems”.
- If success,
  - copy `$(patch_tex)` to `$(patch_tex_copy)`, and
  - delete the files `$(patch_error)` and `$(patch_error_rej)`.

7.4.5 Patch work, reverse order

Short name: “`patch`”.

- If `$(patch_tex)` and `$(patch_tex_copy)` are different, and
- `$(patch_error)` doesn’t exist,

then

- (re)generate the patch `$(patch_file)` from `$(patch_tex)` and `$(patch_tex_orig)`, and
- copy `$(patch_tex)` to `$(patch_tex_copy)`.

After the patch file is created or updated, consodoc thinks that LaTeX code and PDF are out of date. It’s a bit annoying, but reasonable behaviour.

7.5 DVI or PDF from LaTeX

Short name: “`tmpdvi`” or “`tmppdf`”.

There are two very similar steps. The only difference is that parameter names are a bit different (`tmpdvi_xxx` vs `tmppdf_xxx`).

Parameters:

`tmppdf_in`: input LaTeX file. Default: `$(patch_tex)`.

`tmppdf_out`: output PDF file. Default, can’t be changed: `$(noext $(tmppdf_in)).pdf` for `tmppdf`, `$(noext $(tmpdvi_in)).dvi` for `tmpdvi`.

`tmppdf_log`: TeX log file. Default, can’t be changed:
`$(noext $(tmppdf_in)).log`.

`tmppdf_aux`, `tmppdf_outout`, `tmppdf_toc`: TeX auxiliary files. Default, can’t be changed: `(noext $(tmppdf_in)).aux`, `(noext $(tmppdf_in)).out`,
`(noext $(tmppdf_in)).toc`, correspodingly.

`tmppdf_image_dirs`: array of path names for images. Default: `['images/pdf']` for `tmppdf`, `['images/eps']` for `tmpdvi`.

`tmppdf_rerun_max`: maximal number of re-running TeX. Default: `4`.

`tmppdf_include_dirs`: array of path names for included files. Default: `$(supportdir)`.

`tmppdf_rerun_message`: notification message before re-running TeX. Default: "re-running TeX".

`tmppdf_rerun_nomore_message`: Default: "After `$(tmppdf_rerun_max)` attempts, re-run is still required. Human inspection is required."

Actions:

If the file `$(patch_error)` exists, go to the step "Patch Work, Problems".

Before running LaTeX, set the environment variable `TEXINPUTS`. It consist of "`:`"-separated (UNIX) or "`;`"-separated (Windows) list of paths, which come from the variables `$(tmppdf_include_dirs)`, `$(tmppdf_image_dirs)`, and the original `TEXINPUTS`. Paths in `$(tmppdf_include_dirs)` and `$(tmppdf_image_dirs)` are relative to the `$(basedir)`. Before adding them to `TEXINPUTS`, Consodoc converts them to the absolute paths.

Run LaTeX in quiet, batch mode using a helper object. The helper object checks LaTeX logs and detects if re-run is required and collects warnings and errors.

Check if re-run is required. If required, repeat generation. Repeat until there is no need to re-run or `$(tmppdf_rerun_max)` attempts are performed. The latter is the error. Print the message `$(tmppdf_rerun_nomore_message)`. Handle the error.

Print the warnings. Print the errors. If errors exist, it's an error.

In case of error, delete the resulted PDF file.

7.6 Copy the result

Short name: "`pdf`" or "`dvi`".

Again, two very similar steps with the difference only in the names of parameters (`dvi_xxx` vs `pdf_xxx`).

Parameters:

`pdf_in`: input file. Default: `$(tmppdf_out)`.

`pdf_out`: output file. Default: `$(outdir)/$(basename $(pdf_in))`.

Actions:

If the file `$(patch_error)` exists, go to step "Patch Work, Problems".

Otherwise, copy `$(pdf_in)` to `$(pdf_out)`.

8 Predefined processes

At the moment, the only one process is defined. It consist of the following ordered steps:

- Preprocess the Input (`pp`),
- TeXML from XML (`texml`),
- LaTeX from TeXML (`pretex`),
- Patch Work, Forward Order (`tex`),
- DVI or PDF from LaTeX (`tmppdf` or `tmpdvi`),
- Move the Result (`pdf` or `dvi`).

The virtual step “Patch Work, Reverse Order” (`patch`) isn’t ordered and executed on demand.

9 Installation

Consodoc is developed and tested under Linux, but it’s written with portability in mind. Windows and Mac users are encouraged to install the prerequisite software, set the paths to the software in the Consodoc configuration files and test Consodoc.

Consodoc is developed and tested under SCons pre-release version 0.96.90. With each new pre-release version of SCons, Consodoc development is continued under the new version. Consodoc also supports SCons 0.96.1 and the development version.

Prerequisites:

- `diff` and `patch`: should be installed by default on the most Linux systems.
- `xmllint` and `xstproc`: should be installed by default on the most Linux systems.
- `latex` and `pdflatex`: should be available on the most Linux systems.
- `python`: version 2.3 or later, should be available on the most Linux systems.
- `scons`: version 0.96.1 or later, should be available on the most Linux systems.
- `texml`: can be downloaded from http://getfo.org/texml/.

Download Consodoc distribution tarball file `consodoc-N.N.N.tar.gz` from consodoc.com. Unpack the tarball:

```
$ tar zxf consodoc-N.N.N.tar.gz
```

Enter the folder `consodoc-N.N.N` and execute the command:

```
$ python setup.py install
```

If execution is failed with the message like “No module named distutils”, make sure that Python development libraries are installed (the RPM package name might be `python-devel`).

The installation script copies Consodoc files to `/usr/share/consodoc-N.N.N`, documentation to `/usr/share/doc/consodoc-N.N.N` and creates the program `/usr/bin/cdoc`.

It’s possible to use Consodoc locally, without installing it. To do so,

- edit the script `cdoc_local`: set the correct path to the Consodoc files.
- use the modified script.

10 License

The text of this User’s Guide is covered by the GNU Free Documentation License.

The text of the Consodoc license agreement follows.

10.1 License agreement for Consodoc Publishing Server

This license agreement (the “Agreement”) is entered into between you, as a private person or a company (the “Licensee”, “You”) and Oleg Parashchenko (“We”), a private person doing business as Datahansa, having his registered address Uljanovskaja Street 4, Sankt-Peterburg, 198504, Russia. By installing, copying or otherwise using Consodoc Publishing Server (the “Software”) the Licensee agree to be bound by the terms of the Agreement.

10.1.1 Free software dual license

You may use Consodoc under the terms of the GNU General Public License (GPL), version 2.

The following clauses specify the act of running the Software and the output from the Software, which GPL does not cover.

- The output from the Software can be copied, distributed or modified only under the terms of the GPL or any other OSI-approved free license.
- The output from the Software contains invisible marks, which allow identifying the use of the free Consodoc license.

The rest of the text applies only to the commercial Consodoc license.

10.1.2 Per-project licensing

For each project, you should purchase a project-bound Consodoc license. The output from the Software contains invisible marks, which allow us to identify the license details.

The term “Project” isn’t formally defined. Instead, common sense and the fair use approach are applied. Here are a few examples that indicate you have different projects:

- different teams,
- different managers,
- the accounts department handles the projects as different.

10.1.3 Upgrades

We charge for the new versions of the Software. However, the upgrade fee is only 50% of the difference between the price of the new version and the price of your version.

10.1.4 Custom development

You are allowed to modify Consodoc source code for your needs, and allowed to distribute and sell the modifications, if you meet all of these conditions.

- The modified files must carry prominent notices stating that you changed the files and the date of any changes.
- Each recepient must purchase a Consodoc licenses in advance.

10.1.5 Support

We provide support only for installing and running the Software. XSLT, TeXML, LaTeX and other development under Consodoc are subject to a separate support and maintenance fee.

10.1.6 Privacy

The Licensee grants us the right to store contact details and other related information in order for us to contact the Licensee occasionally regarding the Software. We will not use the contact details and other related information for any other purpose.

10.1.7 No warranty

There is no warranty for the software, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program “as is” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

End of Terms and Conditions