

Building Books

A Guide to Transforming Novdoc Documents or “Don't panic!”

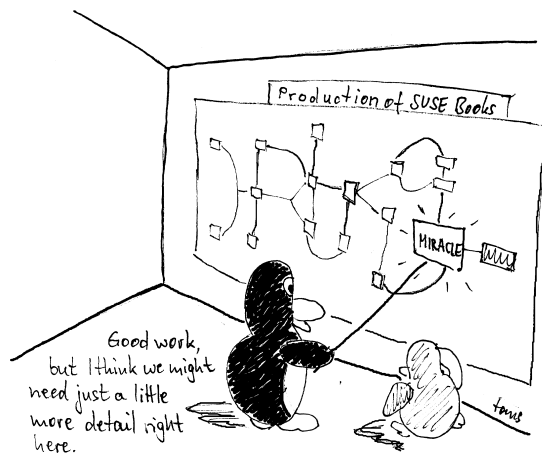
Berthold Gunreben

<berthold DOT gunreben AT suse DOT de>

Thomas Schraitle

<thomas DOT schraitle AT suse DOT de>

\$Id: susemakedoc.xml 547 2010-10-01 07:44:34Z toms \$



Revision History	
Revision 1.0	2005-05
	Initial version
Revision 1.41	2005-06-08
	Updated with suggestions from Dublin
Revision 1.46	2005-06-13
	Updating information about make targets
Revision 1.50	2005-06-21
	Added information about how to configure hyphenation
Revision 1.55	2005-07-11
	Reinserted accidentally delete validation section from bg again
Revision 1.60	2005-07-13
	Added new section about Checking in and Quick Start
Revision	2005-10-11
Added XEP installation, more task oriented descriptions. Reorganized the structure.	
Removed CVS section and moved it into a separate article.	
Revision 546	2010-10-01
	Updated documentation

Contents

1	Quick Start	3
2	Requirements	3
3	Installing XEP	4
4	Setting Up and Checking Your XML Build Environment	5
5	Using the SUSE Make Mechanic	6
6	Troubleshooting	11
7	Background Information	12
8	Notes on Administration	17
9	Notes on Translations	19

A Adding a New Project	20
B Make Targets Reference	22
Glossary	25
Links	26

1 Quick Start

Configuring your system enables you to build books from SUSE. Proceed as follows:

- 1 Check your requirements (see Section 2 (page 3))
- 2 Install XEP (see Section 3 (page 4))
- 3 Setting up your XML environment (see Section 4 (page 5))
- 4 Build your books (see Section 5 (page 6))
- 5 Publish your PDF

2 Requirements

Before you start, you should be aware of some prerequisites. You need:

1. A Linux system. :-)
2. Some rudimentary Linux knowledge. This article cannot explain the basics of Linux.
3. Some knowledge about XML and Subversion (where needed). This article cannot go in-depth into these two technologies, but we provide some links for further information. (See Links (page 26).)
4. The susedoc package. It requires several other packages, for example the DocBook stylesheets.

3 Installing XEP

The `susedoc` package installs everything what you need for building PDFs and HTML. In case you want to build your books with XEP (a commercial formatter from RenderX [<http://www.renderx.com>]) you should have two files:

1. The XEP ZIP archive with the name `xep-X.Y-DATE-docbench.zip`. The placeholders *X* and *Y* are major and minor number of the XEP release, *DATE* corresponds to the date in the format `YearMonthDay`.

This file contains the XEP formatter and the XML editor `oXygen`. You can use both independently of each other.

2. The license key with the name `license.xml` which is mailed to you after purchasing XEP.

To install XEP on your system do the following:

Procedure 1 *Installing XEP on your system*

- 1 Log into your system as you normally do with your preferred user.



XEP and user `root`

Do not install XEP as `root`. Installing XEP as `root` makes it difficult to track errors and other weird combinations.

- 2 Open a terminal window with `Alt + F2` and enter `konsole`.

- 3 Copy your XEP ZIP archive to `/tmp`:

```
cp xep-X.Y-DATE-docbench.zip /tmp
```

- 4 Change the directory:

```
cd /tmp
```

- 5 Unzip the archive:

```
unzip xep-X.Y-DATE-docbench.zip
```

It gives you two files: a `README.txt` and a file named `setup-X.Y-DATE-docbench.zip`.

6 Start the installation dialog with:

```
java -jar setup-X.Y-DATE-docbench.jar
```

7 Choose the correct directory where you want to install XEP and the oXygen XML Editor and select the path to the license file. Proceed with *Install* twice to start the installation process.



Home directory accessed with NFS

Install XEP *always* in a directory which resides on your local harddisc. Don't install XEP in a directory which is accessed by NFS (Network File System). This is the case if you work in a network environment where `/home` is mounted from a server. Otherwise choose a neutral one, like `/local`. Make sure you have write permissions to this directory.

8 Delete the copied archive and the extracted files under `/tmp` where necessary.

9 Extend to `PATH` environment variable with your installation directory from XEP. Open `~/.bashrc` or `~/.bashrc.local` and insert:

```
export PATH="$XEP_INSTALLTION_PATH:$PATH"
```

Replace `XEP_INSTALLTION_PATH` with your XEP installation directory.

4 Setting Up and Checking Your XML Build Environment

Before you build a book, you should check whether all relevant packages are installed. Because it can be a bit difficult to know all packages, we have created a setup script that checks your system for the correct versions.

We assume in this section, you use the directory `/local` for the storage of your files. Whenever you see there is this directory, replace it with your choice.

Proceed as follows:

Procedure 2 *Setting up your XML Build Environment*

1 Access the source code:

- If you have access to the SUSE Subversion repository, see the separate documentation *Using Subversion* [using.svn] (page 26).
- If you don't have access to our Subversion repository, you get an archive from us. Unpack it and move it to your desired location.

2 Change the directory:

```
cd /local/suselx/novdoc/
```

3 Run the following commands:

```
./autogen.sh && ./configure && make
```

First it generates the `configure` and `Makefile` files. If this was successful the `configure` script checks for certain packages and programs, for example:

```
checking for inkscape... inkscape
```

4 Install further packages, if you get an error. The script tells you what you have to install.

5 Run the following command if everything was successful:

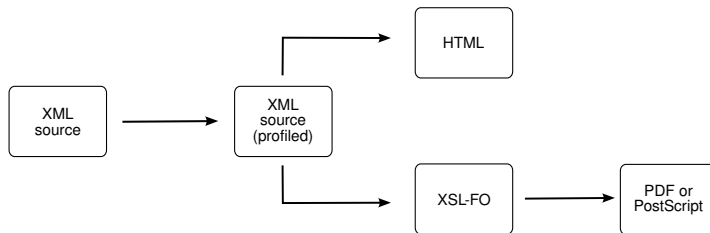
```
make install-xep
```

This installs configuration and hyphenation files into the XEP installation directory.

5 Using the SUSE Make Mechanic

This section explains how to use our build system to create books in HTML and PDF. The entire process of building books from XML sources contains the steps in Figure 1, “Overview of the Entire Process” (page 7).

Figure 1 *Overview of the Entire Process*



The build process looks like:

1. First, you need to access the files from our repository. This gives you a set of directories.
2. You configure your settings in your current shell
3. Make sure that your XML files are *valid*. You cannot build HTML or PDF with invalid files.
4. Your XML source code is *profiled* according to your settings. Find more in Section 7.2, “Profiling Books” (page 13).
5. The profiled XML sources are transformed into HTML or XSL-FO.

5.1 Building HTML

To build a HTML from the XML source code, you will see the following steps:

1. According to your settings, the XML source codes will be profiled first
2. The profiled XML sources are validated
3. After the validation, the XML sources are transformed into HTML

To build HTML do the following:

Procedure 3 *Building HTML*

1 Open a shell with Alt + F2 and type `konsole`

2 Change the directory to your directory structure:

```
cd PATH_TO_YOUR_DIRECTORY
```

3 Set up your shell to configure your environment. In general this is done with:

```
source ENV-SLPROF-html
```

4 Enter:

```
make html
```

This will profile the XML sources and transform this into HTML.

5 Open a browser and access the HTML files under `html`.

5.2 Building PDF

To build a PDF from the XML source code, you will see the following steps:

1. According to your settings, the XML source codes will be profiled first
2. The profiled XML sources are validated
3. After a successful validation, the XML sources are transformed into a FO file. This is an intermediate format
4. XEP reads the FO file, renders it in memory and save it as PDF

To build a PDF do the following:

Procedure 4 *Building PDF*

1 Open a shell with Alt + F2 and type `konsole`

2 Change the directory to your directory structure:

```
cd PATH_TO_YOUR_DIRECTORY
```


3 Set up your shell to configure your environment. In general this is done with:

```
source ENV-SLPROF-print
```

4 Build your PDF with or without cropmarks:

- With cropmarks:

```
make pdf
```

- Without cropmarks:

```
make color-pdf
```

5.3 Transforming Only a Portion of the Book

If you work on only one chapter, transforming the whole book is a waste of time. You can save time if you use the `ROOTID` parameter. For example, if you have a chapter with an `id` attribute value of `chap.foo`, use the following procedure:

Procedure 5 *Transforming Only a Portion of the Book*

1 Open a shell with `Alt + F2` and type `konsole`

2 Set up your shell to configure your environment. In general this is done with:

```
source ENV-SLPROF-print
```

or

```
source ENV-SLPROF-online
```

depending on your desired output

3 Open the chapter, appendix, glossary or preface that you are working on.

4 Check your `id` value by looking at the root element. Memorize this `id`.



Check Your `id` Attribute

This mechanism only works if one of the top level elements (chapter, appendix, preface, or glossary) has an `id`

attribute. You cannot select a chapter without it. Insert an `id` attribute into your top level element. Naming conventions are described in the style guide.

5 Build your output format:

- For HTML:

```
ROOTID=THE_ROOT_ID_VALUE make html
```

- For PDF:

```
ROOTID=THE_ROOT_ID_VALUE make pdf
```

6 Open the result file as `THE_ROOT_ID_VALUE.html` or `THE_ROOT_ID_VALUE.pdf`

Be aware that for HTML a chapter is spread over different files. The file `THE_ROOT_ID_VALUE.html` is just the beginning. However this files contains links to everything inside the chapter.

5.4 Debugging PDFs

In a PDF with lots of pages, it can be difficult to find the correct file if you find an error. For this purpose, we inserted the *Debugging Mechanism*.

You only need to set an environment variable, and the process automatically generates a PDF with filenames in it. Of course, this PDF is only for debugging purposes and is not to be published!

To create a PDF with filenames in it, do the following:

1 Set the environment variable `DRAFT` to `yes`:

```
export DRAFT="yes"
```

2 Make sure that each root element in each file contains the attribute `xml:base` with the filename in it:

```
<sect1 id="sec.foo" xml:base="foo.xml"> ...
```

3 Re-create the PDF by running:

```
make force
```

4 The environment variable is now set. The PDF contains filenames in it.

6 Troubleshooting

6.1 Fixing Errors

Validation errors are fatal when they appear in the profiled source code. If some error occurs, you must modify the original source code in a way that the profiled code becomes valid. Therefore, you have to fix the error in a completely different file than where it is found in the first place. There are two mechanisms that may be used to validate the profiled source code:

```
1. make validate
```

```
2. make bigfile
```

The target `make bigfile` is a workaround for a bug in `xmllint` that occurs in versions before 2.6.15. When you use this target, you find the complete book in `tmp/$(BOOK).xml`. The only way to handle errors then is to look up the error in this huge file, and then find the same error somewhere in the xml sources to fix it.

If you call `make validate`, all validation errors will be listed with the filename and line number of the profiled sources. This means that, depending on the profiling options, the error might be exactly at the same place in the source files, or in a later line. If you have problems finding the error, you can look up the profiled sources at the specified line number and then find the same lines in the original files.

If you get an error that says that, some element does not follow the DTD, you have another way to search for the problem:

```
Element table content does not follow the DTD, expecting
(title , tgroup), got (tgroup )
Document profiled/x86-amd64-em64t_slprof-slpers_0_0/MAIN.box.xml does not
validate
```

In this example, the error first says, that the problem occurs in a table. The error also tells you which XML structure has been found, and what was expected. In this case, the DTD expected a title element inside the table before starting with the actual table content.

Another common problem is having data outside of XML tags. Such an error may look like the following:

```
Element table content does not follow the DTD, expecting
(title , tgroup), got (title CDATA tgroup )
Document profiled/x86-amd64-em64t_slprof-slpers_0_0/MAIN.box.xml does not
validate
```

In this example, there is a character, that is not whitespace between the title of a table and the table content. Please note that sometimes such characters are not displayed because they might not be a printable characters. In such cases, the easiest fix is to remove all whitespace, and then format the xml again.

Most of the problems are found by `make validate`, but a view things are not detected by this method. The FO-processor XEP detects at least one more type of problem: whether index ranges have matching start and end elements. This kind of error is quite clearly described in the output of XEP. If you have any doubt, please compare the specified elements with the original English version.

6.2 Missing Font during build PDF

If you build a PDF and XEP complains about some missing fonts you didn't install the configuration file for XEP. Look at Procedure 4, “Building PDF” (page 8) and begin with Step 2 (page 8).

7 Background Information

You need this section only, if you need more in-depth information of some background information.

7.1 Using make

The command line utility `make` reads a file `Makefile` that includes a rule set for creating and converting files. The `Makefile` in a project directory just includes the file `common.mk` in `suselx/novdoc/make`.

First, select which project to work on. This is done by sourcing an `ENV-` file that contains the needed information to build a certain book.

```
source ENV-styleguide
```

After this, you can use the build environment in your current shell.

7.2 Profiling Books

Some Basics

Before you create your HTML or PDF, you have to make a decision about *profiling*. Profiling describes a method that manages different “versions” in one file.

For example, think of a chapter about installing a program on different processor architectures. Each must use different setup utilities and paths. One solution would be to write one chapter for each architecture. The problem is that there are more things that are similar than things that are different. Maintaining files that only differ in some minor paragraphs or sections is a nightmare.

A solution is to use profiling. In general, you insert each architecture into *one* file and distinguish the different platforms with an XML attribute. The only drawback is that you have to extract the correct version that is “buried” in the file.

To tell the build mechanics about the needed architectures, you need to make sure that the variables `PROFARCH` and `PROFOS` are set properly in your `ENV-` file. Example 1, “A `ENV-*` file” (page 14) shows you the content of the `ENV-` file that sets variables for this document. (See also Appendix A, *Adding a New Project* (page 20).)

Example 1 *A ENV-* file*

```
# ENV file
. .env-profile                                ❶

export MAIN=MAIN.makedoc.xml                 ❷

export PROFARCH="x86; amd64; em64t"          ❸
export PROFOS="slprof; slpers"               ❹
export PROFCONDITION="print"                 ❺
export LAYOUT="flyer"                        ❻
export DISTVER=9.3                           ❼
```

- ❶ Includes system relevant information. This is mandatory, and must be called at the beginning of this file.
- ❷ Sets the main file that contains “references” to other chapters, appendixes, etc. This variable is mandatory.
- ❸ Selects the architecture profiling information. This is an optional variable, but is most likely used in our books.
- ❹ Selects the operating system profiling information. This is an optional variable, but is most likely used in our books.
- ❺ Selects those parts of the book that are to be printed or made available online. Currently used parameters are print and online.
- ❻ Determines the layout of the book. The following values are possible:

flyer

Creates a flyer layout which is used for Quick Starts. It has usually two columns with small margins.

pocket

Creates a pocket layout which is used mostly for the Start-Up guide. It has a smaller page format than the usual guides.

Empty, no value

Selects the default layout.

- ❼ Sets the version of the current distribution. This is used to distinguish versions built with make dist.

As an author or user, you do not need to do anything special. The make process knows which files have been modified and must be profiled again. When changing profiling

variables, the ENV- file has to be sourced again, and a make force command will redo the profiling.

Changes in MAIN* files

The make mechanics can be used not only for Novdoc but for DocBook as well. For this reason, mechanic has to know which stylesheet to use for profiling. We have to use different stylesheets, because we have different DTDs. Each DTD needs a separate profiling stylesheet. In general, they are all the same, but they differ in one part: which DOCTYPE they generate.

To use it, insert the following line into the MAIN, right after the XML declaration:

```
<?xml-stylesheet
  href="urn:x-suse:xslt:profiling:docbook45-profile.xsl"
  type="text/xml"
  title="Profiling step"?>
```

The second line can be only one of the following:

```
urn:x-suse:xslt:profiling:novdoc-profile.xsl
urn:x-suse:xslt:profiling:docbook43-profile.xsl
```

The first line is the default and should be used. Insert the second line only if you use DocBook V4.3.

7.3 Validating Books

Before you transform your XML document, you must take care that the *complete* document is valid. If only one file is invalid, the process stops and you do not get your desired output format. This has nothing to do with the make process itself. It is the design of XML. You can only get predicted results if the input files adhere to a certain structure.

If you want to validate your document without building, use the command

```
make validate
```

The make process takes care of profiling the files then validates the entire document. To validate only single files, use:

```
make validatesingle
```



Validating Single Files

Validating single files can give unexpected results. In general, some sections or files depend on others. This can make validation difficult.

7.4 Adding Images

Add new images under `images/src` in the subdirectory of the correct format. Currently, two formats are supported.

PNG (Portable Network Graphic)

This is a bitmap format that is used in our online versions (HTML) for all kinds of graphics. In the printed books, it is used for screenshots too.

SVG (Scalable Vector Graphic)

This is a vector format that is used in our books (PDF) when available. It can be scaled to different resolutions and you always get very high quality.

If you have a format that is not listed above, convert it into the supported format. For historical reasons, there is a FIG format, too. New files should be created and saved as SVG or PNG.



Support for Other Formats

There is no need to use formats other than PNG. Other bitmap formats, like JPEG or GIF, can easily be converted into PNG. Make sure that you always use PNG.

For other vector formats, check if your graphic program allows exporting SVG. If this is not possible, export to PNG. When in doubt, ask the authors of this guide.

7.5 Transforming Books

Our books can be transformed into two target formats: HTML and PDF. Use HTML for online versions and PDF for a nice printout. To create HTML, type:

```
make html
```


To create PDF, type:

```
make pdf
```

This creates lots of messages. The HTML files are saved into `html`. To view it, point a browser to `index.html` in this directory.

For creating a PDF, just enter `make`. The resulting PDF appears in the current directory and has the name `$(BOOK).pdf`.

8 Notes on Administration

You need this section only, if you need more in-depth information of configuration and administration of the whole build mechanic.

8.1 Hyphenation

You need to read this section only if you want to support a new language that is not available in our Subversion repository. Other users can skip this section.

XEP supports hyphenation in different languages. It uses the same algorithm as LaTeX. Therefore you can use the same files from LaTeX in XEP.

We distinguish between these two possible options:

- Support a new language
- Provide exceptions to hyphenated words in a supported language

Support a new Language

The hyphenation algorithm uses hyphenation patterns to determine possible positions in a word. XEP stores this information in the files `hyphen/*.tex`.

If you want to support a new language, proceed with following steps:

- 1 Copy the respective hyphenation pattern from LaTeX into `hyphen` in your XEP installation directory.

2 Open the XEP configuration file, usually `xep-suse.xml`.

3 Search for the following line:

```
<languages default-language="en-US" xml:base="hyphen/">
```

4 Insert after this line the following XML code:

```
<language name="Czech" codes="cz">  
  <hyphenation pattern="czhyphen_rx.tex"/>
```

In this example, you insert hyphenation patterns for Czech. The information can be found in file `czhyphen_rx.tex`.

5 Repeat Step 1 (page 17) to Step 4 (page 18), if you have other languages.

6 Save the configuration file.

Exceptions of Hyphenation

Sometimes, the algorithm creates unwanted results. For this reason, you can insert exceptions of hyphenation. Do the following:

1 Open the respective file for your language, for example `hyphen.tex` for English.

2 If there is no line beginning with `\hyphenation`, insert the following code:

```
\hyphenation{%
```

3 Insert in the next line as many exceptions as you like. Each exception contains dashes to indicate the possible locations of hyphens. For example:

```
ap-pen-dix%  
man-u-script%  
man-u-scripts%
```

4 Terminate the line with:

```
}
```

5 Save the file.

9 Notes on Translations

If you are a translator, you probably need more specific information. This section gives a brief overview of what you need to know. If in doubt, contact the authors of this document.

9.1 Comments and Remarks

Our XML source code contains comments in two forms: as XML comments and as remark elements. Both are used frequently and look like this:

```
<!-- This is an XML comment -->
...
<remark>Here comes a remark comment</remark>
```

You do not need to translate comments. They are only informative text for writers or even for you. Sometimes there is language-specific information, so we inform you about these issues in a comment or remark.

9.2 Screens

Computer output and input often contains whitespace (space and linebreaks). Make sure that you do not insert or delete spaces inside a `screen` element. Every space here is considered essential.

In general, `screen` contents should be translated only in rare cases.

9.3 Entities

Entities are placeholders for text or other XML structures. An entity name must never be translated! An entity looks like: `&entityname;`.

9.4 IDs in Elements

Copy `id` attributes from the original English source code to the target language. This is important. If you omit this step, correct references cannot be created. An `id` attribute looks like this (bold text):

```
<sect1 id="sec.foo">
...
</sect1>
```

9.5 Be aware of Text between Tags

The “normal” text occurs inside the `para` element. However, there are some locations where text is not allowed. For example, a description of a menu item is marked up with `menuchoice` and `guimenu`. See Example 2, “Correct use of `menuchoice`” (page 20).

Example 2 *Correct use of `menuchoice`*

```
<menuchoice>
  <guimenu>Alt</guimenu>
  <guimenu>F2</guimenu>
</menuchoice>
```

Don't insert text between `</guimenu>` and `<guimenu>` like in Example 3, “Incorrect use of `menuchoice`” (page 20).

Example 3 *Incorrect use of `menuchoice`*

```
<menuchoice>
  <guimenu>Alt</guimenu> with
  <guimenu>F2</guimenu>
</menuchoice>
```

A. Adding a New Project

When adding a new project, you can distinguish two cases. First, you want to reuse some of the existing chapters and, therefore, only need to create a new `MAIN*.xml` and a new `ENV-*` file. If you want to create a completely new project, you need to set up the directory structure and also put the Makefile in place.

The main task of the Makefile is to include the ruleset provided in a system makefile named `common.mk`. You can copy the Makefile from another project and make sure that the path to `novdoc` is correct.

In the `ENV-*` file you need to set several parameters, the following are mandatory:

DTDROOT

This variable contains the absolute path to the `novdoc` directory on your system. This is usually done with a line like:

```
export DTDROOT=$(cd ../../novdoc; pwd)
```

MAIN

This is the MAIN file. The top level file of your document. All required files are included directly in this file, or in one of the included files. This file must exist in the `xml` subdirectory.

BOOK

Use this variable to give your project a name. The resulting PDF will be named like that variable, extended by `.pdf`.

Some necessary variables that are needed by the system must be loaded from the system profile. The current `ENV-*` contains the line:

```
. $DTDROOT/etc/system-profile
```

There are also several optional parameters you can set in this file. It is very important that you set the optional parameters after loading the system profile. The system profile removes all previously set optional values in order to have a clean environment. The following list gives an overview over currently available optional parameters.

PROFOS and PROFARCH

When using profiling attributes in your document, you need to select the needed parts by setting these variables.

FOOPTIONS

The default value of `FOOPTIONS` is `-q`. This changes the behaviour of the FO-Processor (XEP and FOP) to be less verbose about what it does.

COMMENTS

You might want to print the comments you include in a document. To print your remarks in the PDF, set `COMMENTS` variable to 1.

For the various transforming processes, make needs several directories are needed by make. These have to be provided in each project directory.

B. Make Targets Reference

buildbooks (1)

buildbooks — Building Books with make

Make Targets

Use the following syntax:

```
make [TARGET]
```

```
make ROOTID={ID_VALUE} [BUILDTARGET]
```

The optional *TARGET* argument can be a BUILDTARGET, or some other target. The BUILDTARGETS can have one or more of the following names:

`pdf`

Creates PDF with cropmarks and with grey images. This is the default target.

`color-pdf`

Creates PDF without cropmarks and with color images.

`force`

This is just like the `pdf` target, but every source file is profiled again. This is useful when you make changes to the build environment that make cannot detect.

`html`

Creates HTML. The resulting document can be found in the `html` subdirectory with `index.html` being the starting point. If you use a *ROOTID*, the starting point will be `html/${ID_VALUE}.html`.

The following are common targets where *ROOTID* does not have any effect:

`check`

Print the values of several variables that are used in the Makefile.

`clean`

Remove all profiled files as well as temporary files. Does not remove books that have been made.

`configure`

Starts a reconfiguration of the currently used `xep-suse` script. This is necessary after an update, when a change in the `xep-suse.xml` file has changed. If unsure, just run the command. See also the target `update`.

`directories`

Create a template directory structure. When creating a completely new project, you may just create the main directory, and copy the file `Makefile` from an existing project. Make sure that the path to `common.mk` in `Makefile` is correct and run `make directories`.

`dist-html`

Create `html` and `desktop` packages. These are needed to submit the packages of the distribution. Resulting files are `$(BOOK)_$(LANG)-desktop.tar.bz2` and `$(BOOK)_$(LANG)-html.tar.bz2` where `$(LANG)` is selected by the `lang` attribute of `$(MAIN)`.

`filelist`

Creates a list of all `xml` files used in the current book. You might want to add the `entity` file to complete this list.

`dist-xml`

Create packages of the unprofiled `xml` files and of the profiled `xml` files in `.zip` archives. The resulting name is `$(BOOK)_$(LANG)-src.zip` for the unprofiled `xml` and `$(BOOK)_$(LANG).zip` for the profiled version. If one of these files already exists, it is renamed with the old script.

`dist-graphics`

Create package of the `PNG` and `SVG` files that are used in the current project. The name of the archive is `$(BOOK)_$(LANG)-graphics.tar.bz2` where `$(LANG)` is selected by the `lang` attribute of `$(MAIN)`.

`showgfx`

Print a list of the images that are missing from the currently selected book.

`show-remarks`

Show all the remarks in a selected book. In `oXygen`, the displayed `xpath` may also be used to jump to the remark in the source `xml` file.

update

Get all updates of the stylesheets and make mechanics. After this, the profiling step of all source files is performed during the next rebuild. See also the target `configure`.

validate

Update the profiling step and validate the profiled book.

validatesingle

Validate each individual file of the XML source, rather than the entire book.

The variable `ID_VALUE` can contain any chapter or section id. In this case, the basename of the resulting pdf or html will be this id.

Glossary

DTD (Document Type Definition)

A document type definition (DTD) is a set of declarations that conform to a particular markup syntax. It describes a class, or type, of SGML or XML documents. With a DTD, XML documents can be validated. This is useful if you want to check the allowed structure for a certain XML instance.

profiling

Profiling describes a filtering mechanism. An XML document can hold different versions in *one* file, like different architectures, platforms, operating systems, or user levels. The author inserts profiling information into XML documents at specific places and the profiling mechanism selects only those parts that are relevant. This solves the problem of having lots of files with only minor differences. The author has to maintain only one file, but has to take care of the correct profiling information.

Profiling information is inserted with the attributes `os` and `arch`.

project profile

Project profiles contain information needed to select the correct profiling information. The first step to build a book is to *source* the project profile in your current shell. This sets all the relevant variables needed by our Makefiles. See Also profiling.

valid

Valid means that an XML document is well-formed and adheres to a certain DTD. See Also DTD (Document Type Definition), well-formed.

well-formed

An XML document is well-formed if a start tag and an end tag have the same name, each tag contains only valid characters according to the XML specification, the nesting is correct (every start tag must have a corresponding end tag), and attribute values are enclosed by single (') or double quotes (").

XSL-FO (XSL Formatting Objects)

XSL-FO (or just FO) is a specification for describing page layouts. The book production process needs it as an intermediary file for building PDFs.

Links

[docbook-tdg] *DocBook—The Definitive Guide*. <http://www.docbook.org/tdg/en/html/docbook.html>. Norman Walsh and Leonard Mueller.

[cvs] *Version Control with Subversion*. <http://svnbook.red-bean.com/>. Ben Collins-Sussman, Brian W. Fitzpatrick, and Michael Pilato.

[using.svn] *Using Subversion*. Berthold Gunreben and Thomas Schraitle. .