User Guide

Tanja Roth <ta-ro@opensuse.org>
Stefan Knorr <sknorr@suse.de>

User Guide

by Tanja Roth and Stefan Knorr
Technical review: Thomas Schraitle, Frank Sundermeyer

@DAPS_VERSION@

Abstract

(PDF [pdf/book.daps.user_color_en.pdf])

DAPS (DocBook Authoring and Publishing Suite) helps technical writers to author and publish documentation written in DocBook XML. DAPS is a command line based software for Linux* and released as open source.

The DAPS User Guide is a comprehensive guide for technical writers using DAPS. It guides you through creating, editing, managing and publishing your documents—be it a short article by a single author or a large documentation project written by multiple authors.

Table of Contents

About This Guide viii
Target Audience viii
Available Documentation viii
Feedback viii
Documentation Conventions viii
About the Making of This Document ix
1. System Requirements and Installation 1
System Requirements 1
Hardware Requirements 1
Software Requirements 1
Additional Software 1
Installation 2
Installing DAPS on openSUSE 2
Installing DAPS on SUSE Linux Enterprise 2
Installing DAPS on Other Linux Distributions 3
2. Conceptual Overview 4
Supported DocBook Versions 4
Key Features 4
DAPS Configuration 5
Defining Documentation Projects 5
Directory Structure 6
Source Files 6
The `build` Subdirectory 6
Key Files 7
Single Deliverables (Article or Book) 8
Multiple Deliverables: Articles or Books in a Set 12
Basic DAPS Syntax 15
3. Editing DocBook XML 17
Basic Structural Elements 17
Choice of Editor 17
Spell Checking 17
Checking Links to Web Pages 19
Profiling—Support for Document Variants 20
Keeping Track of Your Documentation Project 20
For More Information 21
4. Generating Output Formats 22
Validating Your XML Sources 22
Basic Syntax for Generating Output 23
Supported Output Formats 23
Advanced Output Options 24
5. Image Handling 26
Supported Image Types 26
DIA 26
EPS 27
FIG 27
JPEG 27
PDF 27
PNG 27
SVG 28
Source Images and Generated Images 28
File Name Requirements 28
Referencing Images 29
DAPS Commands for Managing Images 29
6. Modularizing Documentation Projects 32
Splitting up Documents into XIncludes 32

Declaring Entities in a Separate File 32
Profiling—Support for Document Variants 33
Introduction to DocBook Profiling 33
Using Profiling with DAPS 35
Combining Entities and Profiling 39
7. Review and Translation Processes 43
Including Remarks, or Draft Watermarks in the Output 43
Creating XML Big Files 43
8. Packaging and Deploying Your Documentation 44
Creating a TAR Archive with All Sources (Including Graphics) 44
Generating a Distributable HTML Archive 44
Generating Desktop, Document, or Page Files 44
9. Customizing Layout of the Output Formats 45
Modifying Individual XSLT Processor Parameters 45
Specifying the Layout for ASCII Text Output 46
Customizing the DocBook Stylesheets 46
10. Configuring DAPS 47
11. Troubleshooting 48
Glossary 49
A. Migration of Existing DocBook Projects 52
B. Editor-specific Information 53
Emacs—Macros for Inserting DocBook Elements 53
Editing XML/DocBook Files with the Vim Editor 53
jEdit—Spell Check on the Fly 53
C. What's New? 55
D. GNU Licenses 56
GNU General Public License 56
Preamble 56
GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION 56
How to Apply These Terms to Your New Programs 59
GNU Free Documentation License 60
PREAMBLE 60
APPLICABILITY AND DEFINITIONS 61
VERBATIM COPYING 62
COPYING IN QUANTITY 62
MODIFICATIONS 62
COMBINING DOCUMENTS 64
COLLECTIONS OF DOCUMENTS 64
AGGREGATION WITH INDEPENDENT WORKS 64
TRANSLATION 64
TERMINATION 65
FUTURE REVISIONS OF THIS LICENSE 65
ADDENDUM: How to use this License for your documents 65

List of Figures

3.1. Example Output of **daps checklink** 19

List of Tables

4.1. DAPS Output Commands and Formats 23
6.1. Output of Example 6.4 34
6.2. Output of Example 6.4 (Without Profiling) 34
6.3. Output of Example 6.5 35
6.4. Profiling Attributes (DocBook) and Profiling Parameters (DAPS) 35
6.5. Output Variants of Example 6.15 / Example 6.16 Combined With Example 6.14 42

List of Examples

2.1. Required Directory Structure 6
2.2. Build Directory 7
2.3. MAIN file of an Article (DocBook 4.x) 8
2.4. MAIN file of an Article (DocBook 5.x) 9
2.5. Basic DC File for an Article 10
2.6. MAIN file of a Book (DocBook 4.x) 10
2.7. MAIN file of a Book (DocBook 5.x) 11
2.8. DC File For a Book with Custom Layout 11
2.9. MAIN file of a Set (DocBook 4.x) 12
2.10. DC File For a Book in a Set 12
2.11. DC File For Another Book in the Same Set 13
2.12. DC File for a Set 14
2.13. DAPS Syntax 15
3.1. Default Output of a File Listing DAPS Command 21
3.2. Pretty-printed Output of a File Listing DAPS Command 21
4.1. Parser Output For Validation Errors (`xref` to unknown ID) 22
5.1. Image Reference in an XML File 29
5.2. Default Output of an Image-related DAPS Command 31
5.3. Pretty-printed Output of an Image-related DAPS Command 31
6.1. Separate Entity File `entity-decl.ent` 32
6.2. Referencing A Separate Entity File 32
6.3. Referencing Entity Files Within an Entity File 33
6.4. Product-specific Profiling (One Attribute) 34
6.5. Product-specific Profiling (Multiple Attributes) 34
6.6. Profiling PI in a DocBook 4.5 File 36
6.7. Profiling PI in a DocBook 5.0 File 36
6.8. XML File With Profiling Attributes (DocBook 4.x) 37
6.9. MAIN file With PI for Profiling (DocBook 4.5) 37
6.10. MAIN file With PI for Profiling (DocBook 5.0) 38
6.11. DC File with Profiling for Home Edition 38
6.12. DC File with Profiling for Professional Edition 38
6.13. DC File with Profiling for Professional Edition (OEM Version) 39
6.14. Separate Entity File with Profiling Attributes 39
6.15. XML File with `&productname;` and `&productnumber;` Entities (DocBook 4.5) 40
6.16. XML File with `&productname;` and `&productnumber;` Entities (DocBook 5.0) 41
9.1. Adjusting the Layout of Variable Lists 46
B.1. A `varlistentry` Element 53

About This Guide

DAPS (DocBook Authoring and Publishing Suite) helps technical writers to author and publish documentation written in DocBook XML. DAPS is a command line based software for Linux* and released as open source.

Target Audience

This document is intended for users who want to make efficient use of DocBook XML for editing and publishing their documentation—be it documentation sets, individual books, or articles. Key knowledge of XML and DocBook and of using the Bash Shell (or command line interfaces in general) is required.

Available Documentation

This guide contains links to additional documentation resources. The following manuals are available for DAPS:

DAPS Quick Start
: The DAPS Quick Start is a short introduction to DAPS for technical writers. It includes step-by-step instructions for key editing and publishing tasks.

User Guide
: The DAPS User Guide is a comprehensive guide for technical writers using DAPS. It guides you through creating, editing, managing and publishing your documents—be it a short article by a single author or a large documentation project written by multiple authors.

Feedback

We want to hear your comments and suggestions about DocBook Authoring and Publishing Suite (including this guide and the other documentation included with DAPS). You can contact us on the `#opensuse-doc` IRC channel on `irc.freenode.net` or in the discussion forum at http://sourceforge.net/p/daps/discussion/. For bugs or enhancement requests, open an issue at https://github.com/openSUSE/daps/issues/new. A user account at https://github.com is needed.

Patches and user contributions are welcome!

Documentation Conventions

The following typographical conventions are used in this manual:

- `/etc/passwd`: directory names and file names
- *`placeholder`*: replace *`placeholder`* with the actual value
- `PATH`: the environment variable PATH
- **ls**, `--help`: commands, options, and parameters
- `user`: users or groups
- **Alt**, **Alt** + **F1**: a key to press or a key combination; keys are shown in uppercase as on a keyboard
- File, File + Save As: menu items, buttons
- *Dancing Penguins* (Chapter *Penguins*, ↑Another Manual): This is a reference to a chapter in another manual.

About the Making of This Document

This documentation is written in DocBook (see http://www.docbook.org) and generated by DAPS.

Chapter 1. System Requirements and Installation

This chapter describes:

- on which computers DAPS will work,
- how to install DAPS on openSUSE or SUSE Linux Enterprise, and
- how to build and install DAPS on other Linux* distributions.

System Requirements

DAPS itself is a lean solution that does not require a lot of system resources. However, it does use components that may need a stronger processor and more RAM, for example, for creating PDF output files. Although not required, DAPS benefits from multi-core processors.

Hardware Requirements

RAM
: The required amount of RAM mostly depends on the volume of your documentation projects. For creation of PDF output, at least 2 GB of RAM are recommended.

CPU
: If you have multiple or very large documentation projects, a machine with multiple cores is recommended.

Hard Disk Space
: The disk space consumed mostly depends on the amount of your documentation sources and the number of output formats you want to generate.

Software Requirements

DAPS runs on any modern Linux system. It has not been attempted to port DAPS to Windows* or Mac OS X* yet.

When installing DAPS as an RPM package (on any SUSE-based system), dependencies on other software packages are automatically resolved during installation. No additional action is required.

Additional Software

In addition to DAPS, you need the following software:

- An XML (or text) editor of your choice.
- For generating PDF output: an FO formatter, like FOP [http://projects.apache.org/projects/fop.html] or XEP [http://www.renderx.com]. The FO formatter Antenna House Formatter [http://www.antennahouse.com] is currently not supported. Whereas FOP is an open source product, both XEP and Antenna House are commercial products.

To add further components like version management or a workflow mechanism for your projects, use DAPS in combination with the following software:

- Any version management system, like CVS, Subversion, Mercurial or Git.

Together with the software components mentioned above, DAPS can be used as a fully-fledged authoring and content management system for documentation projects based on DocBook.

Installation

The DocBook Authoring and Publishing Suite can be installed and used on any Linux distribution. Currently, DAPS is available as an RPM package for the openSUSE distribution and for SUSE Linux Enterprise products. Eventually, packages for other distributions may become available. For the latest status update and installation instructions, refer to https://github.com/openSUSE/daps/blob/master/INSTALL.adoc.

Installing DAPS on openSUSE

There are a few ways to install DAPS on openSUSE. To always stay up-to-date with the latest version of DAPS install the `daps` package from the `Documentation:Tools` repository as outlined below.

You may also use the `daps` package that shipped with your version of openSUSE. However, you then might miss the latest features and bug fixes in DAPS.

The quickest way to install DAPS is using the **zypper** command.

Procedure 1.1. Installing DAPS via Zypper From Documentation:Tools

1. Open a browser and enter the following URL: http://download.opensuse.org/repositories/Documentation:/Tools
2. Select your distribution and product number to make the browser show the URL for the respective repository.
3. Copy the URL from the address bar.
4. Open a terminal.
5. Add the repository with the following zypper command:

   ```
   root # zypper ar -f URL Documentation:Tools
   ```

   Replace *URL* with the URL you pasted from your browser.
6. Install DAPS with the following zypper command:

   ```
   root # zypper in --from Documentation:Tools daps
   ```

   In order to install DAPS you have to trust the `Documentation:Tools` repository.

Installing DAPS on SUSE Linux Enterprise

Starting with SUSE Linux Enterprise 12, DAPS is also available for SUSE Linux Enterprise. The DAPS package is provided by the SUSE Software Development Kit (SDK), a free extension for SUSE Linux Enterprise. You need to install it as add-on (or extension). You can install the SDK (without any physical media) as an extension after the registration of your system at SUSE Customer Center. If you prefer to install from a physical medium, proceed as follows:

1. Download the installation media for the SDK from http://download.suse.com/.
2. Install the SDK as an add-on product to SUSE Linux Enterprise.

   For details on how to install add-on products (with or without physical media), see the *SUSE Linux Enterprise 12 Deployment Guide*, available at http://www.suse.com/documenta-

tion/. Refer to chapter *Installing Add-On Products*, https://www.suse.com/documentation/sles-12/book_sle_deployment/data/cha_add-ons.html.

3. Install the `daps` package, using either the YaST Software Management module or the following command:

   ```
   root # zypper in daps
   ```

   Dependencies on other software packages are automatically resolved during installation.

Installing DAPS on Other Linux Distributions

For the latest status update and installation instructions, refer to https://github.com/openSUSE/daps/blob/master/INSTALL.adoc.

Chapter 2. Conceptual Overview

This chapter describes:

- the features that make DAPS stand out, from creating multiple output formats to automatically profiling documents,
- how to configure DAPS, and
- the basics of working with DAPS.
- the basic syntax of **daps** commands.

Supported DocBook Versions

DAPS supports DocBook 4.x and DocBook 5.x.

Key Features

DAPS supports technical writers in the editing, translation and publishing process of DocBook XML files (in the following, simply called as XML files):

Output Formats (Single-source Publishing)
: DAPS lets you publish your XML sources in several different output formats, for example: HTML, HTML-single, PDF, EPUB, text, and man pages. For details, refer to Chapter 4, *Generating Output Formats*.

Custom Layouts
: By default, DAPS uses the DocBook stylesheets to generate the output formats. But DAPS also supports custom layouts for your documentation projects (or for individual books within your set). Thus your XML documents can be published in different layouts without having to change the sources or the configuration.

  Apart from that, DAPS allows you to change individual layout parameters by passing string parameters to **xsltproc** for HTML or PDF builds —without even touching the stylesheets. For details about custom layouts, refer to Chapter 9, *Customizing Layout of the Output Formats*.

Editor Macros
: For Emacs, DAPS includes a set of macros for easy insertion of complex DocBook elements like `variablelist`, `figure`, `table` or `indexterm`. Instead of inserting the child elements successively, you will get a "skeleton" that includes all required child elements and is ready to be filled with contents. For details, refer to the section called "Emacs—Macros for Inserting DocBook Elements".

Validating
: Validating XML files within in a book or set exceeds validation of the current XML file, as links (`xref` elements) or XIncludes need to be resolved, too. With DAPS, you can check validity of all files that belong to a documentation project with a single command. For details, refer to the section called "Validating Your XML Sources".

Spell Check
: DAPS supports spell checking of your XML sources with aspell from the command line. Depending on the XML editor you use, you can also integrate a custom aspell dictionary into your editor. For details, refer to the section called "Spell Checking" and the section called "jEdit—Spell Check on the Fly".

Link Checker
: To make sure that all external links in your XML sources are still available (and do not give a `404` error or similar), DAPS also includes a link checker (based on `checkbot`). Use it to create a

report of all links that caused some kind of warning or error. For details, refer to the section called “Checking Links to Web Pages”.

Image Handling
: DAPS provides sophisticated image handling support. For example, it automatically transforms images referenced in your XML files into different formats, list all source images referenced in your XML files, list any missing images or check if all image names are unique. You can also forward those lists to your preferred image viewer to conveniently browse through the images. For details, refer to Chapter 5, *Image Handling*.

Profiling (Conditional Text)
: If you have similar products to document and want to generate multiple documentation variants from your XML files, you can do so with the help of conditional text (or `profiling`, as it is called in DocBook). For example, you can profile certain parts of your XML texts for different (processor) architectures, operating systems, vendors or target groups. DAPS supports profiling. Use the `PROF*` keys defined in `/etc/daps/config` to define which information should be included in the output. For details, refer to Chapter 6, *Modularizing Documentation Projects*.

Review and Translation Processes
: DAPS offers several features to simplify review and translation processes. By adding a single parameter, you can generate output that contains remarks for writers, reviewers, and translators. By default, remarks are suppressed in the final output version. You can also generate preview versions of your documentation with a `DRAFT` watermark appearing on the output. For handing over your files to review or translation, DAPS can create TAR archives of the XML sources and graphics. For details, refer to Chapter 7, *Review and Translation Processes*.

Packaging and Deployment
: For deploying the documentation as RPM packages and integrating it into KDE and GNOME desktop environments and into Web user interfaces (via JSP), DAPS offers a number of options to produce the corresponding output. For example, you can create source packages, HTML TAR archives, color PDFs and desktop and document files with the **daps package-*** commands. For details refer to Chapter 8, *Packaging and Deploying Your Documentation*.

DAPS Configuration

DAPS can be customized to a large degree: per system, per user, and per document. The configuration file `/etc/daps/config` lists all parameters that can be configured, including a short description for each parameter. Parameters are always defined as `KEY="VALUE"` pairs. Any parameter can be set in various locations, which are listed below in ascending order with regard to their hierarchy. If conflicting values are set for the same parameter, the value defined in the next higher hierarchy level takes precedence. Values defined on the command line always take precedence over values set in any other locations.

- `/etc/daps/config` (system-wide configuration file)
- `~/.config/daps/dapsrc` (user-specific configuration file)
- DC (doc config) file of the documentation project (for settings specific to a document or documentation set)
- on the fly at the command line by specifying options to a **daps** command.

Defining Documentation Projects

The easiest way to set up a new documentation project from scratch is to use the DAPS initialization script **daps-init**. For instructions how to do so, refer to Procedure 2, “Using **daps-init** ”. The script automatically creates the Key Files and Directory Structure that you need to get started with DAPS.

To migrate existing DocBook projects so that you can manage and publish them with DAPS, follow the step-by-step instructions in Appendix A, *Migration of Existing DocBook Projects*.

Directory Structure

For DAPS to work out of the box, your XML files and images must be organized in a specific structure within your documentation directory. Example 2.1 shows the required structure including the key files for a DAPS documentation project. You can also create multiple documentation directories for individual documentation projects, but they all need the substructure outlined below.

Source Files

Example 2.1. Required Directory Structure

```
YOUR_DOC_DIR/ ❶
     |--DC-* ❷
     |--images/
     |    |--src/❸
     |    |  |--dia/
     |    |  |--eps/
     |    |  |--fig/
     |    |  |--jpg
     |    |  |--pdf/
     |    |  |--png/
     |    |  |--svg/
     |--xml/❹
     |    |--MAIN*.xml❺
          |--*.xml
```

❶ "Working directory" for the respective documentation project (in the following also called `project directory` or `documentation directory`).

❷ On the topmost level of your project directory, store the Doc Config (DC) file defining your documentation project. You can store multiple DC files here (for multiple books belonging to the same documentation project, or DC files for various documentation projects). For more information, refer to the section called "Key Files".

❸ Top-level directory for any original images that you want to use in the documentation project. Contains subdirectories for images in various formats. Any images to be referenced in the XML sources must be put in the respective subdirectories. For basic information about referencing images, refer to the section called "Referencing Images".

❹ Directory holding the XML MAIN file and all other XML files for the documentation project. If you declare entities in one or more external files (for example, in `entity-decl.ent`), put the entity declaration files here, too.

❺ The MAIN file of the documentation project. It contains the "starting point" (the highest-level object) of your documentation project and includes "references" to other books, chapters, appendixes, etc. For more information, refer to the section called "Key Files".

Structure of the `xml` and `image/src/*` Directories

Avoid subdirectories within the `xml` and `image/src/*` directories. Referencing or including files from subdirectories within those directories can lead to unpredictable results with DAPS.

The `build` Subdirectory

To strictly discriminate between all source content added by users and the content generated by DAPS, DAPS uses a `build` directory. When generating output from your documentation project for the first time, DAPS adds a `build` directory to your documentation directory. It is located parallel to the `xml`

and `images` subdirectories. (If desired, the name and path of the `build` directory can be changed with the parameter *`BUILD_DIR`* in `/etc/daps/config` or `~/.config/daps/dapsrc`.)

The `build` directory is structured as follows:

Example 2.2. Build Directory

```
YOUR_DOC_DIR ❶
     |--build/ ❷
     |--NAME_OF_DC1/ ❸
     |--NAME_OF_DC2/ ❸
     |--.images/ ❹
     |--.profiled/ ❺
     |--.tmp/ ❻
```

❶ "Working directory" for the respective documentation project.
❷ Directory that holds all contents build by DAPS.
❸ For each of your documentation deliverables, DAPS creates a subdirectory. It is named after the respective DC from which you build the book, article or set. For example, the output for `DC-`*`NAME`* is written to *`YOUR_DOC_DIR`*`/build/`*`NAME`*. All formats that have been generated from the DC (PDF, HTML, TXT, EPUB etc.) can be found there. A `log` subdirectory stores log files for each output format that has been generated by DAPS.
❹ Directory holding the images created by DAPS.
❺ Directory holding the profiled XML sources created by DAPS.
❻ Directory holding temporary files created by DAPS (for example, the FO files).

Key Files

The following key files define a documentation project so that it can be processed by DAPS:

MAIN File
: A DocBook XML file in the `xml` directory. It contains the "starting point" (the highest-level object) of your documentation project (for example, `book` or `article`). For larger documentation projects, it is good practice to name the file `MAIN-`*`PROJECTNAME`*`.xml`, but you can use any other file name as well. Other XML files may be included into the MAIN file via `<xi:include/>` statements.

Doc Config (DC) File
: A configuration file in the project directory. It defines several parameters for your documentation deliverable (for example, the MAIN file, layout variants, or which profiling information to use). Of the multiple parameters that can be set in the DC file, the only one required is *`MAIN`*, pointing to the XML file that you want to process. Usually, you create one DC file per book or article. For a documentation `set` (a collection of books), multiple DC files can be defined. This allows you to set different parameters and different values for individual books in the set.

In the following sections, find examples for MAIN and DC files, together with background information on some key parameters that can be used in DC files. The examples are sorted according to use cases:

- Small documentation projects, consisting of Single Deliverables (Article or Book)
- Larger documentation projects, consisting of Multiple Deliverables: Articles or Books in a Set

Differences DocBook 4 versus DocBook 5

The examples below differ slightly with regards to the respective DocBook version. One of the main differences is the header of the XML files. Apart from that, the `articleinfo` and `bookinfo` elements no longer exist in DocBook 5. They have been replaced by the generic `info` element.

Single Deliverables (Article or Book)

The most elementary case of a documentation project is probably a white paper or article. Typically, its content can be contained in a single XML file with `article` as the root element. In this case, this single XML file would be the MAIN file as it specifies the highest-level object in your documentation project (`article`). Apart from document title and body, the file can contain other information such as a legal notice, release information, author data etc. An article may be structured into sections (by use of `section` elements or `sect1`, `sect2` etc.).

Creating an Example Document

The command **daps-init** allows you to automatically set up an example article or book, together with a DC file, as described in Procedure 2, “Using **daps-init** ”. Use the `--docbook4` or the `--docbook5` option to define the Docbook version to use. The examples below are based on the output of **daps-init**, but vary deliberately in some details to show key parameters that you might want to add or change.

Find simple examples for DocBook 4 and DocBook 5 in Example 2.3 and Example 2.4, respectively.

Example 2.3. MAIN file of an Article (DocBook 4.x)

```
<?xml version="1.0" encoding="UTF-8"?>
[...]

<article lang="en" id="art.template">
 <title>Article Template</title>
 <subtitle>generated by DAPS</subtitle>
 <articleinfo>
  <releaseinfo>Version 0.1</releaseinfo>
  <releaseinfo>Revision: 0</releaseinfo>
  <releaseinfo>
   Build Date: <?dbtimestamp format="B d, Y"?>
  </releaseinfo>
  <legalnotice>
   <para>
    <ulink url="http://www.gnu.org/licenses/fdl-1.3-standalone.html">
    GNU Free Documentation License</ulink>
   </para>
  </legalnotice>
 </articleinfo>
 <abstract>
  <para>
   You may use this file as a template. For a complete DocBook reference
   see <citetitle>DocBook: The Definitive Guide</citetitle>, available at
   <ulink url="http://www.docbook.org/tdg/en/html/docbook.html"/>.
  </para>
 </abstract>
 <sect1 id="sec.template.examples">
  <title>Examples: The most commonly used DocBook XML constructs</title>
  <para>
   I am a paragraph in a section 1.
  </para>
  <sect2 id="sec.template.examples.lists">
   <title>Lists</title>
   <para>
    This section 2 showcases 3 types of lists.
   </para>
```

```
   [...]
  </sect2>
 </sect1>
</article>
```

Example 2.4. MAIN file of an Article (DocBook 5.x)

```
<?xml version="1.0" encoding="UTF-8"?>
[...]

 <article xml:id="art.template" xml:lang="en"
  xmlns="http://docbook.org/ns/docbook"
  xmlns:xlink="http://www.w3.org/1999/xlink">

<info>
 <title>Article Template</title>
 <subtitle>generated by DAPS</subtitle>
 <info>
  <releaseinfo>Version 0.1</releaseinfo>
  <releaseinfo>Revision: 0</releaseinfo>
  <releaseinfo>
   Build Date: <?dbtimestamp format="B d, Y"?>
  </releaseinfo>
  <legalnotice>
   <para>
    <link xlink:href="http://www.gnu.org/licenses/fdl-1.3-standalone.html">
    GNU Free Documentation License</ulink>
   </para>
  </legalnotice>
 </info>
 <abstract>
  <para>
   You may use this file as a template. For a complete DocBook reference
   see <citetitle>DocBook: The Definitive Guide</citetitle>, available at
   <link xlink:href="http://www.docbook.org/tdg/en/html/docbook.html"/>.
  </para>
 </abstract>
 <sect1 xml:id="sec.template.examples">
  <title>Examples: The most commonly used DocBook XML constructs</title>
  <para>
   I am a paragraph in a section 1.
  </para>
  <sect2 xml:id="sec.template.examples.lists">
   <title>Lists</title>
   <para>
    This section 2 showcases 3 types of lists.
   </para>
   [...]
  </sect2>
 </sect1>
</article>
```

Let us assume, the XML file shown in Example 2.3 or Example 2.4 is named `MAIN-DAPS-example-article.xml` and you want to publish it using the default DocBook layout. To generate output, you usually create a DC file per article or book, specifying a number of parameters such as the MAIN file or which layout to use. Of the multiple parameters that can be set in the DC file, the only one required is *`MAIN`*, pointing to the XML file that you want to process. Therefore, a very basic DC file for the article in Example 2.3 or Example 2.4 could look as follows:

Example 2.5. Basic DC File for an Article

```
## Doc config file for the DAPS example document
## See /etc/daps/config for documentation of the settings below
##

## Mandatory Parameter
MAIN="MAIN-DAPS-example-article.xml"
```

Specifies the XML MAIN file. It contains the highest-level object (root element) of your documentation project. The MAIN file must be located in *YOUR_DOC_DIR*/xml/. Therefore, you only need to specify the MAIN's file name in the DC file (no path).

The example above is a bit artificial, though: If you do not want to specify any further parameters (apart from the MAIN file), you can also set the `--main` parameter as a command line option when generating the output format. In that case, you can do completely without a DC file. For details, refer to Chapter 4, *Generating Output Formats*.

In case your documentation project consists of a single book, instead of an article (as assumed before), the basic setup of MAIN file and DC file is similar:

Example 2.6. MAIN file of a Book (DocBook 4.x)

```
<?xml version="1.0" encoding="UTF-8"?>
[...]

<book id="book.template" lang="en">
 <bookinfo>
  <title>Book Template</title>
  <subtitle>generated by daps</subtitle>
  <productname>Book Template</productname>
   <legalnotice>
   <para>
    <ulink url="http://www.gnu.org/licenses/fdl-1.3-standalone.html">
    GNU Free Documentation License</ulink>
   </para>
  </legalnotice>
 </bookinfo>
 <chapter id="cha.template.examples">
  <title>Examples: the most commonly used DocBook XML constructs</title>
  <abstract>
   <para>
    You may use this file as a template. For a complete reference on DocBook
    see <citetitle>&tdg;</citetitle>, available at
    <ulink url="http://www.docbook.org/tdg/en/html/docbook.html"/>.
   </para>
  </abstract>
  <para>
   I am a paragraph in a chapter.
  </para>
  <sect1 id="sec.template.examples.lists">
   <title>Lists</title>
   <para>
    This is a section 1.
   </para>
  </sect1>
 </chapter>
</book>
```

Example 2.7. MAIN file of a Book (DocBook 5.x)

```
<?xml version="1.0" encoding="UTF-8"?>
[...]

<book xml:id="book.template" xml:lang="en"
      xmlns="http://docbook.org/ns/docbook"
      xmlns:xlink="http://www.w3.org/1999/xlink">
 <info>
  <title>Book Template</title>
  <subtitle>generated by daps</subtitle>
  <productname>Book Template</productname>
   <legalnotice>
   <para>
    <link xlink:href="http://www.gnu.org/licenses/fdl-1.3-standalone.html">
    GNU Free Documentation License</ulink>
   </para>
  </legalnotice>
 </info>
 <chapter xml:id="cha.template.examples">
  <title>Examples: the most commonly used DocBook XML constructs</title>
  <abstract>
   <para>
    You may use this file as a template. For a complete reference on DocBook
    see <citetitle>&tdg;</citetitle>, available at
    <ulink url="http://www.docbook.org/tdg/en/html/docbook.html"/>.
   </para>
  </abstract>
  <para>
   I am a paragraph in a chapter.
  </para>
  <sect1 xml:id="sec.template.examples.lists">
   <title>Lists</title>
   <para>
    This is a section 1.
   </para>
  </sect1>
 </chapter>
</book>
```

In the above example, the book's contents are also contained in a single XML file, however, this time with `book` as the root element. In contrast to an article, books can contain more structural levels: they are usually divided into `chapter` elements (that may contain sections and subsections) as outlined in Example 2.6 or Example 2.7 . In addition to chapters, books may also contain other structural elements such as `preface`, `glossary`, and `appendix`. A further additional structural level is called `part`. For a complete reference, see *DocBook: The Definitive Guide*, available at http://www.docbook.org/tdg/en/html/docbook.html.

Let us assume the XML file shown in Example 2.6 or Example 2.7 is named `MAIN-DAPS-example-book.xml` and you want to publish it in a custom layout. To generate output, you would create a DC file pointing to the MAIN file of the book, and additionally specify a set of custom stylesheets.

Example 2.8. DC File For a Book with Custom Layout

```
## Doc config file for the DAPS example book
## See /etc/daps/config for documentation of the settings below

## Mandatory Parameter
```

```
MAIN="MAIN-DAPS-example-book.xml" ❶

## Optional Parameters

## Custom Stylesheets
## (if not defined the DocBook stylesheets will be used)

STYLEROOT="/usr/share/xml/docbook/stylesheet/custom/xslt" ❷
```

❶ Specifies the XML MAIN file. It contains the highest-level object (root element) of your documentation project. The MAIN file must be located in *YOUR_DOC_DIR*/xml/. Therefore, you only need to specify the MAIN's file name in the DC file (no path).

❷ For a custom layout, use the *STYLEROOT* parameter to specify the (absolute or relative) path to the directory containing the custom stylesheets. Using absolute paths is recommended for DC files.

Multiple Deliverables: Articles or Books in a Set

If your documentation project consists of multiple books in a set, the MAIN file is the one that contains the set element. In the following example, the components of the set (individual books) are not part of the MAIN file, but have been put into separate document files (book*.xml). Those are then assembled in the MAIN file using XIncludes. Note that this is not specific to sets—it is mainly a means of modularizing your documents. You can also use XIncludes for splitting up books, articles or chapters into separate files. For more information, refer to the section called “Splitting up Documents into XIncludes” and *Physical Divisions: Breaking a Document into Separate Files* [http://www.docbook.org/tdg51/en/html/ch02.html].

Example 2.9. MAIN file of a Set (DocBook 4.x)

```
<?xml version="1.0" encoding="UTF-8"?>
[...]

<set lang="en">
 <title>DAPS Documentation</title>
 <xi:include href="book_daps_user.xml"
             xmlns:xi="http://www.w3.org/2001/XInclude"/>
 <xi:include href="book_daps_quickstarts.xml"
             xmlns:xi="http://www.w3.org/2001/XInclude"/>
 <!--<xi:include href="book_daps_developer.xml"
             xmlns:xi="http://www.w3.org/2001/XInclude"/>-->
</set>
```

For a documentation set (a collection of books), multiple DC files can be defined. This allows you to set different parameters and values for individual books in the set. By specifying a different *ROOTID* in the DC file, you define which book of the set is to be built. You can also specify different layouts or output modes (such as draft or annotated versions) for individual books in the same documentation set.

The following two DC files are those of the DAPS Quick Start and the DAPS User Guide— both guides belong to the same documentation set, but use different layouts.

Example 2.10. DC File For a Book in a Set

```
## Doc config file for DAPS Quick Start
## See /etc/daps/config for documentation of the settings below

## Mandatory Parameter
MAIN="MAIN.DAPS.xml" ❶

## Optional Parameters
## ROOTID
```

```
## If MAIN contains a set with several books and/or articles, use
## a separate DC-file for each book/article and set ROOTID to
## the id of the respective <book>/<article> element of the document
## This will enable you to build individual books/articles rather than
## the whole set
## See http://www.docbook.org/tdg/en/html/set.html for more information
## on sets
ROOTID="art.daps.quick" ❷

## Custom Stylesheets
## (if not defined the DocBook stylesheets will be used)

STYLEROOT="/usr/share/xml/docbook/stylesheet/suse/xslt/flyer" ❸
#FALLBACK_STYLEROOT="" ❹
```

❶ Specifies the XML MAIN file. It contains the highest-level object (root element) of your documentation project. The MAIN file must be located in *YOUR_DOC_DIR*`/xml/`. Therefore, you only need to specify the MAIN's file name in the DC file (no path).

❷ Defines the root ID of the element to be used for creating an output format. Usually, you specify the root ID of a `book` or `article` element here.

In this example, `art.daps.quick` is the root ID of the DAPS Quick Start, contained in `MAIN.DAPS.xml`.

❸ For a custom layout, use the *STYLEROOT* parameter to specify the (absolute or relative) path to the directory containing the custom stylesheets. Using absolute paths is recommended for DC files.

In this example, the parameter specifies the path to a directory with SUSE-specific stylesheets for the flyer layout that is used by the DAPS Quick Start.

❹ Allows you to define a fallback which is used in case the custom stylesheets defined with *STYLEROOT* cannot be accessed. In case neither the stylesheets specified with *STYLEROOT* nor with *FALLBACK_STYLEROOT* can be accessed, DAPS uses the default DocBook layout.

In this example, no fallback is specified and the parameter is disabled.

Example 2.11. DC File For Another Book in the Same Set

```
## Doc config file for DAPS User Guide
## See /etc/daps/config for documentation of the settings below

## Mandatory Parameter
MAIN="MAIN.DAPS.xml" ❶

## Optional Parameters
## ROOTID
## If MAIN contains a set with several books and/or articles, use
## a separate DC-file for each book/article and set ROOTID to
## the id of the respective <book>/<article> element of the document
## This will enable you to build individual books/articles rather than
## the whole set
## See http://www.docbook.org/tdg/en/html/set.html for more information
## on sets
ROOTID="book.daps.user" ❷

## Custom Stylesheets
## (if not defined the DocBook stylesheets will be used)

STYLEROOT="/usr/share/xml/docbook/stylesheet/suse/xslt/" ❸
#FALLBACK_STYLEROOT="" ❹
```

```
## Formatter
# Specify which PDF formatter to use. Currently only fop or xep are supported

FORMATTER="xep" ❺

##Draft Mode
# Turns on DRAFT watermarks in PDF or HTML builds when set to "yes"
# Is ignored for any other output format and has no effect on profiling.
# This value can be set to "yes" using the -d switch on the command line
# Also see COMMENTS and REMARKS
#
DRAFT="yes" ❻
```

❶ Specifies the XML MAIN file. It contains the highest-level object (root element) of your documentation project. The MAIN file must be located in *YOUR_DOC_DIR*/xml/. Therefore, you only need to specify the MAIN's file name in the DC file (no path).

❷ Defines the root ID of the element to be used for creating an output format. Usually, you specify the root ID of a book or article element here.

In this example, book.daps.user is the root ID of the DAPS User Guide, contained in MAIN.DAPS.xml.

❸ For a custom layout, use the *STYLEROOT* parameter to specify the (absolute or relative) path to the directory containing the custom stylesheets. Using absolute paths is recommended for DC files.

In this example, the parameter specifies the path to a directory with SUSE-specific stylesheets that is used by the DAPS User Guide.

❹ Allows you to define a fallback which is used in case the custom stylesheets defined with *STYLEROOT* cannot be accessed. In case neither the stylesheets specified with *STYLEROOT* nor with *FALLBACK_STYLEROOT* can be accessed, DAPS uses the default DocBook layout.

In this example, no fallback is specified and the parameter is disabled.

❺ Specifies the PDF formatter to use.

For supported formatters, refer to the section called “Software Requirements”. In this example, XEP is specified as PDF formatter.

❻ When set to yes, a DRAFT watermark appears in PDF or HTML outputs of the document.

If your documentation project contains cross-references between the individual books in a set, it is useful to define an additional DC file —*without* the *ROOTID* parameter. Use this (generic) DC to generate HTML outputs containing all hyperlinks between the individual books (or for creating file lists of all source files and images used in the set). Find an example DC file in Example 2.12, “DC File for a Set”.

Example 2.12. DC File for a Set

```
## Doc config file for the DAPS Documentation Set
## See /etc/daps/config for documentation of the settings below

## Mandatory Parameter
MAIN="MAIN.DAPS.xml" ❶

## Optional Parameters
## ROOTID
## If MAIN contains a set with several books and/or articles, use
## a separate DC-file for each book/article and set ROOTID to
## the id of the respective <book>/<article> element of the document
## This will enable you to build individual books/articles rather than
## the whole set
## See http://www.docbook.org/tdg/en/html/set.html for more information
```

```
## on sets
#ROOTID="" ❷

## Custom Stylesheets
## (if not defined the DocBook stylesheets will be used)

STYLEROOT="/usr/share/xml/docbook/stylesheet/suse/xslt/" ❸
#FALLBACK_STYLEROOT="" ❹

## enable sourcing
export DOCCONF=$BASH_SOURCE ❺
```

❶ Specifies the XML MAIN file. It contains the highest-level object (root element) of your documentation project. The MAIN file must be located in *YOUR_DOC_DIR*/xml/. Therefore, you only need to specify the MAIN's file name in the DC file (no path).

❷ Defines the root ID of the element to be used for creating an output format. Usually, you specify the root ID of a book or article element here.

In this example, no *ROOTID* is set. This allows to build the complete documentation set, with the output containing all hyperlinks between the individual books.

❸ For a custom layout, use the *STYLEROOT* parameter to specify the (absolute or relative) path to the directory containing the custom stylesheets. Using absolute paths is recommended for DC files.

In this example, the parameter specifies the path to a directory with SUSE-specific stylesheets.

❹ Allows you to define a fallback which is used in case the custom stylesheets defined with *STYLEROOT* cannot be accessed. In case neither the stylesheets specified with *STYLEROOT* nor with *FALLBACK_STYLEROOT* can be accessed, DAPS uses the default DocBook layout.

In this example, no fallback is specified and the parameter is disabled.

❺ Enabling this parameter allows you to source the DC file on the Bash with DAPS. Sourcing a DC file (formerly called ENV file) was necessary to work with the documentation environment provided by susedoc, DAPS's predecessor.

Basic DAPS Syntax

Before moving forward, let's get familiar with the basic syntax of the **daps** command:

```
tux:~> daps [--global-options] subcommand [--command-options] [arguments]
```

Example 2.13, “DAPS Syntax” shows an example command that generates HTML output. Global options are used to specify the level of verbosity, and the Doc Config file for creating the output.

Example 2.13. DAPS Syntax

```
tux:~> daps❶ --debug❷ -d❸ DC-daps-example html❹ --static❺
```

❶ Main command: **daps**

❷ Global Option --debug: Sets the highest verbosity level (number of messages shown during the transformation process from XML to HTML).

❸ Global Option -d: Defines the relative or absolute path to the Doc Config file. In this example, **daps** is called in the same directory that holds the Doc Config file.

❹ Subcommand **html**: Defines the output format to create.

❺ Command option --static: Tells DAPS to copy CSS and image files to the same location like the HTML files. For more information, see Table 4.1, “DAPS Output Commands and Formats”.

Specifying the DC File

For execution of most commands, DAPS needs to know which DC file to use. Usually, you do so by specifying a DC file with the global option -d. For example:

```
tux:~> daps -d PATH_TO_DC_FILE pdf
```

Only in the following cases you may omit the `-d` option:

- If your documentation directory contains only one DC file. In that case, DAPS automatically uses the corresponding file.
- If you have specified a default DC file to use in `~/.config/daps/dapsrc` (as a value for *DOCCONF_DEFAULT*). In that case, DAPS automatically uses the corresponding file, unless you specify a different one on the command line.
- If you want to call the help function.

To view the global options and the available subcommands for DAPS, use the command:

```
tux:~> daps help
```

For a short help text on a specific *subcommand*, use:

```
tux:~> daps help  subcommand
```

For example, if you want more information about generating HTML output, run:

```
tux:~> daps help html
```

Chapter 3. Editing DocBook XML

This chapter describes:

- how to choose an editor for editing DocBook XML files,
- how to check the contents of your DocBook files for mistakes, and
- how to adapt your documentation to fit multiple similar product versions at once,
- how to get and keep track of your files included a documentation project.

Basic Structural Elements

If you have worked with DocBook before, you know about the typical top-level elements for documents, `book` and `article`. For larger documentation projects, another typical top-level element is `set` (a collection of books).

To define the individual components of a book, use structural elements such as `part`, `chapter`, `preface` or `appendix`. Chapters are usually subdivided into sections (`section` elements or `sect1`, `sect2` etc.). Smaller structural units are `para` (for paragraphs), or list elements such as `orderlist`, `itemizedlist`, or `variablelist`.

If you have set up your documentation project from scratch with **daps-init**, you can explore the example documents that are installed within the directory structure. They show the most commonly used DocBook XML constructs.

Choice of Editor

As DAPS does not include any editor software, you are completely free in the choice of your XML editor. While you can use your text editor of choice, it is helpful if the editor supports editing XML in accordance with the schema you use. Several open source editors can be extended with plug-ins for automatic tag insertion and completion, insertion of `xref` elements and for checks if the XML document is well-formed. If you are already familiar with vi or Emacs, you can configure them to support XML editing mode. If you prefer an editor with a graphical user interface, jEdit [http://www.jedit.org/] is a good choice.

XML elements can be nested deeply. Constructs like `variablelist`, `table` or `image` often have a lot of child elements—some of them required, some optional. If you have an editor with schema support, it will tell you which elements are allowed at the current cursor position, but nevertheless it is cumbersome if you need to insert the child elements of complex XML constructs consecutively.

Most editors allow you to define or record macros which you can use for automatically inserting empty “skeletons” for a complex XML construct. For Emacs, DAPS already includes macros for adding DocBook elements. For details, refer to the section called “Emacs—Macros for Inserting DocBook Elements”.

Spell Checking

DAPS comes with a spell checker that is optimized for DocBook documents: Tags and attributes are excluded from the check so that you can focus on the content of the document. The spell checker is based on aspell and can be run from the command line. By default, it starts in interactive mode, but you can also run it in “batch” mode where it dumps a sorted list of misspelled words to standard output. DAPS also allows you to specify a custom dictionary and the language to use for spelling.

In the following, find some examples on how to spell check with DAPS. All options discussed below can be combined with each other, except for `--file` and `--rootid` which exclude each other.

spellcheck Options and XIncludes

All options discussed below can be combined with each other, except for `--file` and `--rootid` which exclude each other.

The **spellcheck** command always follows `xi:includes`, even when using the `--file` option.

Spell Checking Files in a Documentation Project

```
tux:~> daps -d PATH_TO_DC_FILE spellcheck
```

Spell checks all files in the documentation project with the default dictionary (`en_US`). One by one, the files are opened in interactive mode and checked with aspell. To abort spell checking of the current file, press **X**. The spell check continues with the next file in the project.

Uses the ROOTID defined in the specified DC file as starting point. You can restrict the spell check to parts of the set, such as a `book`, `article`, `glossary`, `appendix`, `part`, or `chapter` element. To do so, specify the ID of the respective element with the `--rootid` option:

```
tux:~> daps -d PATH_TO_DC_FILE spellcheck --rootid=ID
```

Spell Checking a Single XML File

```
tux:~> daps -d PATH_TO_DC_FILE spellcheck --file PATH_TO_XML_FILE
```

Checks the specified file (plus all files included with XIncludes on this level) using the default dictionary. Suggests alternative spellings for each misspelled word and waits for user interaction.

Spell Checking XML Files in Languages Other than English

```
tux:~> daps -d PATH_TO_DC_FILE spellcheck --lang=LANG \
  [--file PATH_TO_XML_FILE]
```

Checks the specified documentation project or file with the dictionary for *LANG* (make sure the specified aspell dictionary is installed). Suggests alternative spellings for each misspelled word and waits for user interaction. The language code used for the `--lang` option is the same that is used for the `LANG` environment variable and matches the directory names in `/usr/share/locale`.

Spell Checking XML Files in Batch Mode

```
tux:~> daps -d PATH_TO_DC_FILE spellcheck --list \
  [--file PATH_TO_XML_FILE]
```

Checks the specified documentation project or file. Returns a list of misspelled words to standard output. You can use the `--list` option to easily collect a list of words that are unknown to aspell and use this output as basis for a custom aspell word list or dictionary.

Spell Checking XML Files with an Additional Custom Dictionary

```
tux:~> daps -d PATH_TO_DC_FILE spellcheck --extra-dict=PATH_TO_CUSTOM_DICT \
[--file PATH_TO_XML_FILE]
```

Checks the specified documentation project or file with the default dictionary plus the additional custom dictionary specified with `--extra-dict`.

For your convenience, you can integrate daps-susespell (plus a custom aspell dictionary, if needed) into your XML editor, so that spelling is checked “on the fly” during editing. Consult your editor's documentation on how to integrate a custom dictionary. If you use jEdit, proceed as outlined in the section called “jEdit—Spell Check on the Fly”.

Checking Links to Web Pages

To make sure that all external links (such as HTTP, HTTPS and FTP links) in your XML sources are valid (and do not give a `404` error or similar), DAPS also includes a link checker. It is based on `checkbot`, see **man 1 checkbot** for more information. The link checker searches for the `url` attribute in `ulink` elements and checks links included there. Use it to create a report of all links that caused some kind of warning or error.

checklink follows XIncludes

The **checklink** command always follows `xi:include` statements, even when using the `--file` option.

Checking Links in a Documentation Project

```
tux:~> daps -d PATH_TO_DC_FILE checklink
```

Uses the ROOTID defined in the specified DC file as starting point. Checks the `ulink` elements in all files belonging to the documentation project. The resulting HTML report `*checkbot-localhost.html` can be opened in a browser, see Figure 3.1, “Example Output of **daps checklink**”. For opening the results directly in the browser, add the `--show` option.

If your DC file references a documentation `set`, you probably do not want to check all files belonging to the set. You can restrict the check to parts of the set, such as a `book`, `article`, `glossary`, `appendix`, `part`, or `chapter` element. To do so, specify the ID of the respective element with the `--rootid` option:

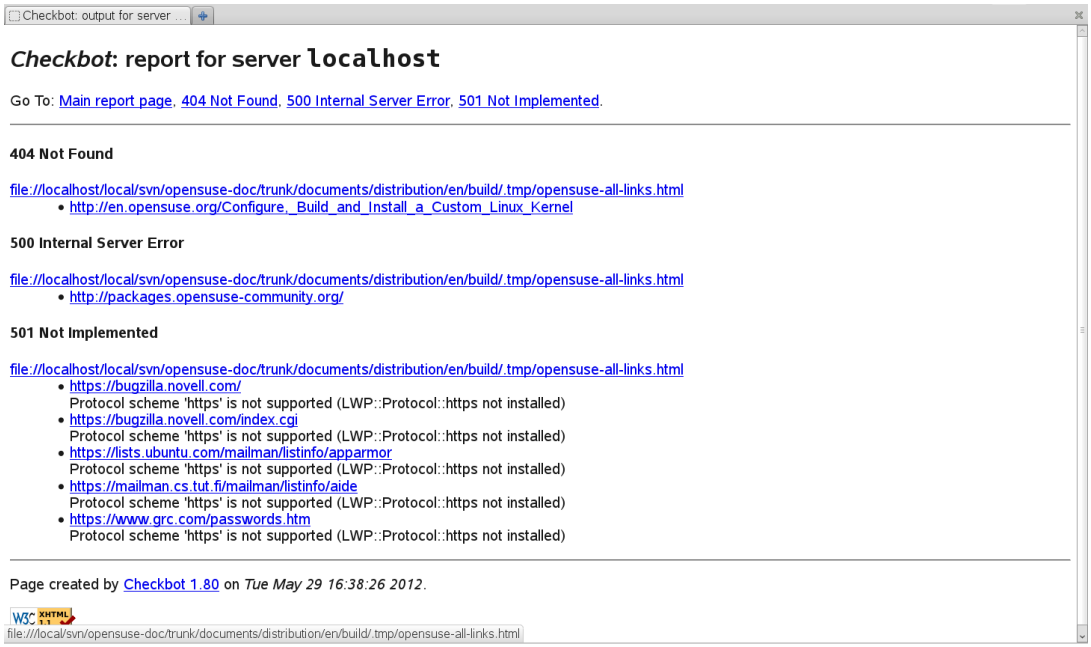
```
tux:~> daps -d PATH_TO_DC_FILE checklink --rootid=ID
```

Checking Links in a Single XML File

```
tux:~> daps -d PATH_TO_DC_FILE checklink --file=PATH_TO_XML_FILE
```

Checks the `ulink` elements in the specified file. At the end of the check, DAPS returns an HTML file with a list of all links which caused some kind of warning or error. Open the resulting `checkbot-localhost.html` file in a browser.

Figure 3.1. Example Output of daps checklink



Opening the Link Check Report

To directly open the link check report, use the DAPS command output as an argument for a browser, for example the command line Web browser **w3m**:

```
tux:~> w3m -dump $(daps -d PATH_TO_DC_FILE checklink)
```

Profiling—Support for Document Variants

Similar products often share a considerable amount of features and differ in details only. It is therefore convenient to apply the same approach to the documentation of similar products or product families: Share most of the XML source code and only differentiate text snippets where necessary. DocBook allows you to create documentation variants from the same pool of XML sources by means of *profiling*.

In DocBook XML files you can mark some elements as conditional by using profiling attributes. When processing the files to generate output, specify which conditions apply to the output. The stylesheets will then include or exclude the marked text, according to the conditions.

Profiling allows you to keep both common and product-specific content in one XML file and select at production time which information to include in the output.

For details, refer to the section called “Profiling—Support for Document Variants”.

Keeping Track of Your Documentation Project

To help you keep track of the files and images included in your documentation project, DAPS provides some useful commands.

Listing All Files in a Documentation Project

```
tux:~> daps -d PATH_TO_DC_FILE list-srcfiles
```

Lists all source files (XML sources, images, entity declarations, and the DC file) used by the document specified with the DC file.

Checking for Superfluous Files

```
tux:~> daps -d PATH_TO_DC_FILE list-srcfiles-unused
```

Lists all files that are *not* referenced in the document specified by the DC file, but are available in the `xml` and `images/src` subdirectories. This command is useful if you want to clean up your source files and want to know which files are no longer needed for the documentation project.

Use a `set` to Check for Superfluous Files

If you are storing multiple DC files in the same project directory, use the DC file of a `set` for this check. As it contains all articles and books in the project directory, this makes sure that any files found during the check are indeed unnecessary.

Checking Where an ID or File is Included

```
tux:~> daps-d PATH_TO_DC_FILE list-file --rootid
     ID
```

Lists the file that contains the ID specified with the mandatory parameter `--rootid`. For example:

```
tux:~> daps -d DC-daps-example list-file --rootid sec.template.examples.images
```

```
The ID "sec.template.examples.images" appears in:
/DOC_DIR_PATH/xml/MAIN-daps-example.xml
```

For the image-related commands, seethe section called “DAPS Commands for Managing Images”.

Output of File Listings

By default, the output of the listing commands is a list of file names (including the absolute path), all printed in one line with the file names separated by blanks. This default output format is useful for piping (or copying and pasting) the output for use with another command.

If you need a pretty printed output where each file name is listed on a separate line, use the `--pretty` option.

See Example 3.1, “Default Output of a File Listing DAPS Command” and Example 3.2, “Pretty-printed Output of a File Listing DAPS Command” for a comparison of both outputs.

Example 3.1. Default Output of a File Listing DAPS Command

```
tux:~> daps -d PATH_TO_DC_FILE list-srcfiles
YOUR_DOC_DIR/DC-daps-example YOUR_DOC_DIR/images/src/png/example1.png YOUR_DOC_DIR/imag
```

Example 3.2. Pretty-printed Output of a File Listing DAPS Command

```
tux:~> daps -d PATH_TO_DC_FILE list-srcfiles --pretty
YOUR_DOC_DIR/DC-daps-example
YOUR_DOC_DIR/images/src/png/example_png1.png
YOUR_DOC_DIR/images/src/png/example_png2.png
YOUR_DOC_DIR/xml/MAIN-daps-example.xml
```

Using Pretty-Printed Output for Counting

To count the number of files listed in a given output, you can also combine the *`--pretty`* parameter with the **wc** command:

```
tux:~> daps -d PATH_TO_DC_FILE list-srcfiles --pretty | wc -l
```

For Example 3.2, the command above would return 4 as a value.

For More Information

For a complete DocBook reference see *DocBook: The Definitive Guide* [http://www.docbook.org/tdg/en/html/docbook.html].

Useful tips and tricks around using DocBook and the DocBook stylesheets can be found in *The DoCookBook—Recipes for DocBook Developers and Writers*, available at http://doccookbook.sourceforge.net/.

Chapter 4. Generating Output Formats

This chapter describes:

- how to validate your XML files,
- the basic command syntax for generating output formats,
- which output formats you can generate with DAPS,
- how to do partial builds of your documentation, and
- how to specify the MAIN file at the command line (instead of using a DC file).

Validating Your XML Sources

Generating any output requires that your XML files are valid. As soon as any output command is executed, DAPS automatically runs a validation check first. If it fails, DAPS returns the parser errors, including information about the type of error, the respective file name and the line number where the error occurred. In addition, DAPS shows the path to the profiled XML sources and the total number of errors.

Example 4.1. Parser Output For Validation Errors (`xref` to unknown ID)

```
daps_user_concept.xml:60: element xref: validity error:
IDREF attribute linkend references an unknown ID "itl.daps.user.inst.other.req"
Document /local/svn/daps-svn/daps/doc/build/.profiled/x86-amd64-em64t_osuse_/
MAIN.DAPS.xml does not validate
make: *** [validate] Error 3
```

Validating is Done in Build Directory

Validation is always done in the `build` directory and in the profiled sources, as indicated by the path above (`/local/[...]/build/.profiled/[...]/MAIN.DAPS.xml`). However, you need to fix the validation error in the sources located in your `xml` directory, otherwise the fixes will not take effect. Profiling is similar to conditional text. For details, refer to the chapter about modularizing document projects in the DAPS User Guide.

As soon you use profiling in your DocBook sources, validation within your XML editor is not reliable. Furthermore, validating XML files within a `book` or `set` often exceeds validation of the current XML file, as links (`xref` elements) or XIncludes need to be resolved, too.

DAPS can handle all those cases because of the built-in `xmllint` validator. By default, `remark` elements and XML comments are ignored during validation. However, if you intend to create a (draft) output including remarks or comments, you need to include them for validation—see the example commands below.

To validate all files that belong to your documentation project, DAPS only needs to know which DC file to use. Use the `-d` option to specify it.

Validating All XML Files in Your Book, Article or Set

```
tux:~> daps -d PATH_TO_DC_FILE validate
```

If the XML files are not valid, DAPS will return the parser errors. If validation was successful, DAPS returns: `All files are valid.`

Validating Your Files Including `Remark` Elements

```
tux:~> daps -d PATH_TO_DC_FILE validate --remarks
```

Basic Syntax for Generating Output

DAPS supports a number of different output formats, including also “exotic” formats like man pages or simple text. Table 4.1 gives an overview.

You can build several output formats (without them interfering with each other in the `build` directory), build your complete documentation project (set, book, or article), or only a part of it (for example, a specific chapter).

Independent of the individual output format you want to create, you need to specify the DC file to use:

```
tux:~> daps -d PATH_TO_DC_FILE OUTPUT_FORMAT
```

For example:

```
tux:~> daps -d DC-daps-example pdf
```

At the end of the transformation process, DAPS shows a message where to find the generated output.

Supported Output Formats

The following table lists the main output formats and their characteristics, and the DAPS subcommands to generate them. Refer to Section for the commands' basic syntax.

Table 4.1. DAPS Output Commands and Formats

Subcommand	Output/Note
pdf	Creates a color PDF. Open the result in a PDF viewer. Requires an FO formatter.
pdf --grayscale	Creates a black-and-white PDF. Open the result in a PDF viewer. Requires an FO formatter. All color images are automatically converted to grayscale images. If you need a PDF for a printing shop, add the `--cropmarks` option. Creation of crop marks is currently only supported by the XEP FO formatter.
html	Creates a subdirectory containing individual HTML files for all chapters of a book (including also preface, glossary or appendix files). The HTML files are named according to the ID of the respective root element. Open the generated `index.html` file in a Web browser to view the generated HTML from the starting point (ROOTID of the top-level element). Images and CSS files are only linked in the resulting directory that contains the HTML files. To copy these files to the same location like the HTML files, use the `--static` option. This is useful for creating distributable HTML builds.
html --single	Creates a single HTML file, named after the DC file used to create the output. Open the generated `*.html` file in a Web browser. Single HTML files are more convenient for full text searches. Images and CSS files are only linked in the resulting directory that contains the HTML files. To copy these files to the same location like the HTML files, use the `--static` option. This is useful for creating distributable HTML builds.

Subcommand	Output/Note
epub	Creates an EPUB 2 document. Open the result in a portable e-book reader (or with a software like Calibre). If you need an EPUB 3 document, add the `--epub3` option.
mobi	Creates an Amazon Kindle e-book in Mobipocket format. Open the result in a portable e-book reader (or with a software like Calibre). Requires Calibre. DAPS first generates an EPUB file which is then converted to `*.mobi` format with Calibre.
webhelp	Creates a DocBook Web Help output. Open the generated `index.html` file in a Web browser to view the generated HTML from the starting point (ROOTID of the top-level element). Experimental feature. Requires a very recent version of the DocBook stylesheets. DocBook Web Help consists of HTML pages with an additional pane, featuring a table of contents and a search function. The table of contents can be expanded and collapsed and is automatically synchronized with the contents pane. The search function weights the search results so that the most relevant results are listed first.
text	Creates an ASCII text output. Open the result in a text editor. All images are removed from the output, but their location is indicated in the text by the respective image base name printed in square brackets. A table of contents is automatically generated and is available at the beginning of the text document.
man	Creates one or multiple man pages. To create man pages, your XML files must contain at least one `refentry`—be it in a `chapter`, `appendix`, or collected in a `reference` element. When processing a DocBook document with multiple `refentry` elements (regardless where they appear), DAPS generates one man page file per `refentry` element. All other parts of the document will be ignored.

The number of output formats may be extended in the future, depending on the output formats that are supported by DocBook stylesheets. For an overview of all output formats, run **daps help**. The available output formats are listed below Subcommands + Generate Books.

By default, DAPS uses the regular DocBook stylesheets, but DAPS also allows you to customize your output formats in a very flexible way. For details, refer to Chapter 9, *Customizing Layout of the Output Formats*.

Advanced Output Options

In the following, find some example commands for special use cases, like doing partial builds of your documentation project or specifying no further parameters than the MAIN file for an output. In the last case, you can do completely without a DC file.

For more advanced output options like including remarks or draft watermarks in the output, creating one big XML file or creating distributable archives, refer to Chapter 7, *Review and Translation Processes*.

Building Only Parts of Your Documentation Project

```
tux:~> daps -d DC-daps-example pdf --rootid=cha.template.examples
```

Instead of always building your complete documentation project (`set`, `book`, or `article`), DAPS also allows you to build only individual parts. The “starting point” of your documentation project is usually defined by the root element of the MAIN file that is referenced in the respective Doc Config. To build only a part of your documentation project, use the `--rootid` option to specify the ID of an individual `book`, `article`, `glossary`, `appendix`, `part`, or `chapter`.

Specifying the MAIN File on the Command Line

```
tux:~> daps -m PATH_TO_MAIN_FILE
```

If you do not need to specify any further parameters than the MAIN file, you can do completely without a DC file. With the `-m` option you can specify the MAIN file defining your document. The options `-m` and `-d` exclude each other.

Opening the Output Directly in Your PDF Viewer

```
tux:~> evince $(daps -d DC-daps-example pdf) &
```

Use the syntax above to open the PDF output directly in a PDF viewer (for example, Evince).

Getting More Information about the Building Process

By default DAPS only shows the path to the resulting file. Several verbosity levels are available. Use the global options `-v*` to specify the level of information you want. It ranges from `-v` (print one line of results) to `-vvv` (print all commands, very verbose). For example, the following command will print all files created during the build process:

```
tux:~> daps -vv -d DC-daps-example pdf
```

For a debug output, use the following:

```
tux:~> daps --debug -d DC-daps-example pdf
```

Chapter 5. Image Handling

This chapter describes:

- which types of images are supported by DAPS,
- the distinction between source images and generated images, and the image directory structure required by DAPS,
- the file name requirements for source images,
- how to reference images in your DocBook files,
- and how to manage images with DAPS commands.

Supported Image Types

Depending on the output format you generate (PDF or HTML, for example), DAPS automatically transforms the source images you provide (which are also referenced in your XML sources) into the appropriate output formats. For example, SVG images are converted to PNG for HTML builds, or color images to grayscale for black-and-white PDFs. You only need to decide which file format to use as source format. Of course, this decision depends on the purpose of the image. For more details, see the DAPS User Guide.

DAPS supports the following types of images:

- DIA (input format only)
- EPS (experimental)
- FIG (input format only)
- JPEG
- PDF (experimental - only works for PDF output and with XEP formatter)
- PNG
- SVG

Image formats can be categorized into pixel-based image formats (also called bitmap formats) and vector-based image formats. In pixel-based image formats the data describes the characteristics of each pixel. In contrast to that, vector-based image formats contain a geometric description that can be rendered smoothly. Vector-based images are resolution-independent—they can be displayed or printed as large or small as you want without showing pixel artifacts.

From the supported image types listed above, only JPEG and PNG are pixel-based image formats.

DIA

Can only be used as input format for DAPS.

DIA is a vector image format which means it is resolution-independent. Images in this format can be displayed or printed as large or small as you want without showing pixel artifacts. The format is suited especially well to creating diagrams. DIA files are XML files that are automatically compressed when saving, thus they are often quite small.

To create DIA files, there is a software of the same name: Dia [https://live.gnome.org/Dia]. Dia is a diagram editor which can be used to draw entity-relationship diagrams, UML diagrams, and flowcharts.

Dia makes it very easy to connect elements, to add text and to use simple fill and border colors. Although it can import SVG files as shapes, it is not useful for freely drawing shapes itself. Complex or effect-laden vector illustrations and information graphics are hard to create with Dia.

EPS

The *Encapsulated PostScript* (EPS) format is a general purpose vector image format. As a Postscript-based format, it is similar to PDF. There is currently no mainstream Linux image editor software that creates EPS files natively, although various applications can export into it.

Where feasible, use SVG files instead of EPS files (also because EPS images only work with XEP formatter at the moment). EPS might occasionally serve as an exchange format with contributors that use Adobe* graphics software.

Limited Support with DAPS

This image format is only supported in combination with the XEP formatter. Using this format might also lead to longer document creation times.

FIG

Can only be used as input format for DAPS.

FIG is a vector image format that can be created with the software Xfig [http://www.xfig.org]. The support for FIG files can help when working with legacy images. However, it is recommended to use SVG format for new illustrations and DIA for new diagrams.

JPEG

The acronym JPEG derives from the *Joint Photographic Experts Group* who created this standard specifically for storing photographic images. JPEG is the most common image format used by digital cameras and widely used for displaying photographs on the Web. Because of its lossy compression algorithm, the resulting file sizes are rather small compared to lossless graphics format such as TIFF, GIF, PNG, or a raw image format. The compression algorithm works very well for photographs and paintings with smooth variations of tone and color. However, JPEG is not well suited for textual or iconic graphics, because sharp contrasts between adjacent pixels in such images can cause noticeable artifacts. It is also not advisable to use JPEG for files that will undergo multiple edits: you will lose image quality each time the file is decompressed and recompressed. This is especially true if the image is cropped or shifted.

PDF

The *Portable Document File* format is a general purpose, page-based fixed-layout document format. PDF is a PostScript-based format. There is a large number of Linux software that can export PDF files natively. PDF files can also be used as an exchange format with contributors that cannot export to SVG.

Limited Support with DAPS

This image format is only supported in combination with the XEP formatter. Using this format might also lead to longer document creation times.

PNG

The *Portable Network Graphics* is a pixel-based format which can be used if you want to use a raster (point-based) image. A good example for when to use PNG files are screenshots and photographs.

PNG files can be created with several applications, including the GIMP graphics editor.

Make PNG Files Smaller with daps optipng

To decrease the file size of PNG images without altering their look, use **daps optipng**. It removes unused colors and alpha channels form your source PNG files. Note that it is the only DAPS command that alters any sources.

To run **optipng** over an entire book's PNG images:

```
tux:~> daps -d PATH_TO_DC_FILE optipng
```

SVG

The *Scalable Vector Graphics* format is a general purpose, vector image format. SVG is an XML format which can be displayed in most browsers and edited in many graphics programs.

Use *Plain SVG* Format

Some SVG editors offer the choice of saving your file in a custom SVG-based format or in *Plain SVG* (*standard SVG*). In this case, always use the plain version. Custom SVG formats might not be compatible with the components used by DAPS for processing SVG files.

XEP and FOP formatter only support SVG 1.0.

A good open source SVG editor is Inkscape [http://inkscape.org/], which is available for most operating systems. You can also create SVG files from many Adobe products, for example, from Illustrator*. SVG is the preferred vector image format for DAPS.

Source Images and Generated Images

DAPS strictly discriminates between source images (any images that have been created outside of DAPS) and images that are generated by DAPS.

This clear distinction is also visible in the file system: source images are stored in a different directory than generated images.

DAPS requires you to use a specific directory structure for images. All images that you reference from your DocBook files must be stored in a subdirectory of the project directory named `images/src/file_extension`. For example, PNG files must be stored under `images/src/png`. If you used **daps-init** to set up your project, the appropriate directories should already exist. For a longer reference to the directory structure, see the section called “Directory Structure”.

From your source images, DAPS automatically generates appropriate image formats for each output format. They are stored in `build/.images/` within the project directory. If an image referenced from your DocBook files is changed, DAPS will notice when trying to build and generate new versions of the image automatically.

For gaining an overview of source images or generated images, and for managing both, DAPS provides different subcommands. For details, refer to the section called “DAPS Commands for Managing Images”.

File Name Requirements

No Spaces and Colons In File and Directory Names
: Avoid spaces and colons in file and directory names. The **make** command in DAPS has trouble understanding them. Use underscores (_) or hyphens (-) instead.

It is good practice to only use the following characters for file or directory names: alphabetic characters [a-z] or [A-Z], numerical characters [0-9], hyphens (`-`), or underscores (`_`).

Unique Image Names
: Always store just one file with a particular name within the `images/src` directory of a project. As DAPS tries to create any missing image formats from original images, it will otherwise not know which one of the duplicate files to use for converting to the missing formats.

  Additionally, having a file called `example.png` and another called `example.svg` in the same documentation project will often lead to questions like: Which file to use where? Do both files display the same content? Are both files current, or is one outdated?

  When invoking DAPS with the parameter **`-v`**, a warning will be printed whenever a file name appears twice within a project. To specifically check for image name clashes upfront, use the **daps list-images-multisrc** subcommand.

  It is a good idea to find a consistent file naming scheme. For example, when documenting software, it might prove helpful to include the name of the application at the beginning of the file name. You can also use prefixes like `screenshot_` and `diagram_` to separate between different types of images.

Referencing Images

As your images need to be located in a defined directory structure, DAPS automatically finds the path to your images. Therefore, referencing images in your XML sources is very straightforward: you must not include any path in the `fileref` attribute—the file name is enough.

Furthermore, DocBook allows you to reference more than one image to distinguish between different output formats. For example, you can add two references pointing to the same file, but using different images widths for PDF and HTML output. Use the `role` attribute to specify the output format, for example `fo` or `html`.

Example 5.1. Image Reference in an XML File

Let us assume you have a source image file named `graphic.dia`. To make DAPS use an SVG version of your image for PDF output (`role="fo"`) and a PNG version for HTML output (`role="html"`), reference the images as follows:

```
<figure>
  <title>Main Window</title>
  <mediaobject>
    <imageobject role="fo">
      <imagedata fileref="graphic.svg" width="70%"/>
    </imageobject>
    <imageobject role="html">
      <imagedata fileref="graphic.png" width="75%"/>
    </imageobject>
  </mediaobject>
</figure>
```

DAPS Commands for Managing Images

DAPS offers several subcommands for managing images in a documentation project. To find out which images are used/not used in a project or referenced in a DocBook file but missing from the file system, DAPS offers subcommands that check your project for such issues.

Apart from that, you can check your source images for non-unique names and reduce the size of your PNGs with an optimizer that recompresses the files to a smaller size.

Subcommands for Image Listing

Listing All Graphics Referenced in a Documentation Project

```
tux:~> daps -d PATH_TO_DC_FILE list-srcfiles --noent --noxml --nodc
```

Lists all graphics used by the DocBook files that are referenced in the document specified by the current DC file.

Listing Missing Graphics in a Documentation Project

```
tux:~> daps -d PATH_TO_DC_FILE list-images-missing
```

Lists all graphics that are referenced in your DocBook files, but could not be found in the file system. In case there are any missing graphics, you cannot build your project.

Checking for Superfluous Graphics

```
tux:~> daps -d PATH_TO_DC_FILE list-srcfiles-unused --noxml
```

Lists all graphics that are *not* referenced in your DocBook files, but are available in the `images/src` subdirectories. This command is useful if you want to clean up your source images and want to know which images are no longer needed for the documentation project.

Use a `set` to Check for Superfluous Files

If you are storing multiple DC files in the same project directory, use the DC file of a `set` for this check. As it contains all articles and books in the project directory, this makes sure that any files found during the check are indeed unnecessary.

Checking for Non-Unique Graphic File Names

```
tux:~> daps -d PATH_TO_DC_FILE list-images-multisrc
```

Checks the `images/src` subdirectories for non-unique base names. For more information, refer to the section called “File Name Requirements”.

Reducing the File Size of PNG Graphics

```
tux:~> daps -d PATH_TO_DC_FILE optipng
```

Recompresses any PNG files in the `images/src/png` directory with the PNG optimizer `optipng` without reducing the image quality.

Deleting All Generated Images for a Documentation Project

```
tux:~> daps -d PATH_TO_DC_FILE clean-images
```

Deletes all images generated for a certain DC file. This is only necessary in rare cases, for example, when a file previously had an incorrect time stamp or when you have changed your global DAPS configuration. In that cases, DAPS might wrongly assume that an already generated file should be inserted into your output format when in reality the file should be re-generated.

Viewing All Images Referenced in a File or Project

```
tux:~> daps -d DC-daps-example getimages --show --viewer ristretto
```

Opens and shows all images referenced in the documentation project in the specified image viewer, for example, Ristretto. Alternatively, specify your preferred image viewer in the *IMG_VIEWER* parameter in the DAPS configuration file. To show only the images referenced in a specific XML file, specify the file with `--file`.

Output of File Listings

By default, the output of the listing commands is a list of file names (including the absolute path), all printed in one line with the file names separated by blanks. This default output format is useful for piping (or copying and pasting) the output for use with another command.

If you need a pretty printed output where each file name is listed on a separate line, use the `--pretty` option.

See Example 5.2, “Default Output of an Image-related DAPS Command” and Example 5.3, “Pretty-printed Output of an Image-related DAPS Command” for a comparison of both outputs.

Example 5.2. Default Output of an Image-related DAPS Command

```
YOUR_DOC_DIR/images/src/dia/example_dia1.dia YOUR_DOC_DIR/images/src/png/example_png1.p
```

Example 5.3. Pretty-printed Output of an Image-related DAPS Command

```
tux:~> daps -d PATH_TO_DC_FILE list-images-multisrc --pretty
YOUR_DOC_DIR/images/src/dia/example_dia1.dia
YOUR_DOC_DIR/images/src/png/example_png1.png
YOUR_DOC_DIR/images/src/png/example_png2.png
YOUR_DOC_DIR/images/src/png/example_png3.png
YOUR_DOC_DIR/images/src/svg/example_svg.svg
YOUR_DOC_DIR/images/src/fig/example_fig1.fig ...
```

Use Pretty-Printed Output for Counting

To count the number of images listed in a given output, you can also combine the *--pretty* parameter with the **wc** command:

```
tux:~> daps -d PATH_TO_DC_FILE list-images-multisrc --pretty | wc -l
```

For Example 5.3, the command above would return 5 as value.

Chapter 6. Modularizing Documentation Projects

This chapter describes:

- how to modularize documents by splitting them into XIncludes,
- how to use a consistent set of entities throughout a documentation project,
- how to create document variants by using profiling, and
- how to combine profiling and entities.

Splitting up Documents into XIncludes

Instead of putting the contents of a complete article or book into the MAIN file, DocBook allows you to divide the text into separate document files. They are then assembled in the MAIN file using `XIncludes` as shown in Example 2.9, “MAIN file of a Set (DocBook 4.x)”. XIncludes can be used for splitting up sets, books, articles or chapters into separate files. For more information, refer to *Physical Divisions: Breaking a Document into Separate Files* [http://www.docbook.org/tdg51/en/html/ch02.html].

XIncludes do not cause any problems with DAPS and are fully supported. For example, **daps** commands like **checklink** or **spellcheck**, for example) also follow XIncludes.

Declaring Entities in a Separate File

When maintaining many documents for a product, it becomes cumbersome to keep a consistent set of entities if you declare entities in the document type declaration of *individual* XML files. For large documentation projects, it is therefore useful to put all entity declarations into a separate file and reference that file in the individual XML files.

Example 6.1. Separate Entity File `entity-decl.ent`

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!ENTITY exampleuser "tux">
<!ENTITY exampleuserII "wilber">
<!ENTITY examplegroup "users">
[...]
```

Example 6.2. Referencing A Separate Entity File

```
<?xml version="1.0"?>
<!DOCTYPE chapter PUBLIC [...]
[
  <!ENTITY % entities SYSTEM "entity-decl.ent"> ❶
  %entities; ❷
]>
<chapter>
<title>Managing User Accounts</title>
[...]
```

❶ Reference to the separate entity declaration file (with the `<!ENTITY>` keyword).
❷ Loads the external entity file declared in the previous line.

For more information, refer to *Modular DocBook Files: Shared text entities* [http://www.sagehill.net/docbookxsl/ModularDoc.html].

Separate entity files do not cause any problems with DAPS— during generation of output, the entities will be treated equally to entities declared in individual XML files.

It is also possible to use multiple entity files by including them into the separate entity file that is referenced in the XML file.

Example 6.3. Referencing Entity Files Within an Entity File

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!ENTITY exampleuser "tux">
<!ENTITY exampleuserII "wilber">
<!ENTITY examplegroup "users">
[...]
<ENTITY % network-entities SYSTEM "network-decl.ent"> ❶
%network-entities; ❷
<ENTITY % more-entities SYSTEM "another-decl.ent"> ❶
%more-entities; ❷
```

❶ Reference to the separate entity declaration file (with the <!ENTITY> keyword).
❷ Loads the external entity file declared in the previous line.

Profiling—Support for Document Variants

Similar products often share a considerable amount of features and differ in details only. It is therefore convenient to apply the same approach to the documentation of similar products or product families: Share most of the XML source code and only differentiate text snippets where necessary. DocBook allows you to create documentation variants from the same pool of XML sources by means of *profiling*.

In DocBook XML files you can mark some elements as conditional by using profiling attributes. When processing the files to generate output, specify which conditions apply to the output. The stylesheets will then include or exclude the marked text, according to the conditions.

Profiling allows you to keep both common and product-specific content in one XML file and select at production time which information to include in the output.

Introduction to DocBook Profiling

DocBook offers profiling attributes for various purposes as illustrated in *Table 26.1. Profiling attributes* [http://www.sagehill.net/docbookxsl/Profiling.html]. All of them are supported by DAPS.

Generally, profiling attributes can be used on many elements—from high-level elements like `book` or `chapter` down to low-level elements like `para`. With the `phrase` element, you can even profile inline elements, like one sentence within a paragraph.

Based on the conditions that you want to apply, select one or more profiling attributes and add them to the text snippets that are conditional. The tagged snippets will only be included in the output if the required condition is fulfilled. Any content that is valid for *all* conditions does *not* need any profiling attributes. The respective content will always be included in the output formats generated from the XML sources. You are free in defining the attribute values (`condition="foo"`), but they must be used consistently in all files belonging to a documentation project.

Example 6.4, “Product-specific Profiling (One Attribute)” shows how to profile product-specific information in a software description. Let us assume we need to write documentation for the fictional software `Frog Sound Recordings`. The software is available in two editions: a basic edition for home users and a professional edition for enterprise customers. Both editions share common features, but some features are only available in the basic or the professional edition, respectively.

Example 6.4. Product-specific Profiling (One Attribute)

```
<simplelist>
  <member>Common Feature 1</member>
  <member>Common Feature 2</member>
  <member>Common Feature 3</member>
  <member condition="basic">Basic Feature 1</member>
  <member condition="prof">Professional Feature 1</member>
  <member condition="prof">Professional Feature 2</member>
</simplelist>
```

When generated for the basic edition or for the professional edition, respectively, the example source code would result in the following output:

Table 6.1. Output of Example 6.4

Basic Edition	Professional Edition
Common Feature 1	Common Feature 1
Common Feature 2	Common Feature 2
Common Feature 3	Common Feature 3
Basic Feature 1	Professional Feature 1
	Professional Feature 2

If the profiling attributes are *not* processed during output, the source code in Example 6.4, “Product-specific Profiling (One Attribute)” would result in the following (identical) output for both editions:

Table 6.2. Output of Example 6.4 (Without Profiling)

Basic Edition	Professional Edition
Common Feature 1	Common Feature 1
Common Feature 2	Common Feature 2
Common Feature 3	Common Feature 3
Basic Feature 1	Basic Feature 1
Professional Feature 1	Professional Feature 1
Professional Feature 2	Professional Feature 2

Let's suppose the professional edition of the software is also available as OEM (original equipment manufacturer) version by the vendor `OEM Company`. It contains additional features that are only available in the OEM version:

Example 6.5. Product-specific Profiling (Multiple Attributes)

```
<simplelist>
  <member>Common Feature 1</member>
  <member>Common Feature 2</member>
  <member>Common Feature 3</member>
  <member condition="basic">Basic Feature 1</member>
  <member condition="prof">Professional Feature 1</member>
  <member condition="prof">Professional Feature 2</member>
  <member condition="prof" vendor="oemcompany">OEM Feature 1</member>
</simplelist>
```

When generated for the professional edition or for the professional edition in the OEM version, respectively, the example source code would result in the following output:

Table 6.3. Output of Example 6.5

Professional Edition	Professional Edition (OEM Version)
Common Feature 1	Common Feature 1
Common Feature 2	Common Feature 2
Common Feature 3	Common Feature 3
Professional Feature 1	Professional Feature 1
Professional Feature 2	Professional Feature 2
	OEM Feature 1

Using Profiling with DAPS

To create multiple documentation variants of the same pool of DocBook files with DAPS, the following requirements need to be fulfilled:

1. XML Files: Profiling Attributes
2. MAIN File: Processing Instruction
3. DC Files: Profiling Parameters

For a comprehensive example showing all requirements in detail, refer to the section called “Profiling Example”.

XML Files: Profiling Attributes

In DAPS, each profiling attribute has a corresponding profiling parameter to be used in the DC file, see the section called “DC Files: Profiling Parameters”. The profiling parameters define which profiling attributes and values to interpret during generation of output.

The names of the profiling parameters are derived from the profiling attributes, but written in uppercase letters and preceded by the prefix `PROF`.

Table 6.4. Profiling Attributes (DocBook) and Profiling Parameters (DAPS)

Attribute Name	Use	Profiling Parameter
`arch`	Computer or chip architecture, such as `i386`.	*PROFARCH*
`audience`	Intended audience of the content, such as `instructor`. Added in DocBook version 5.0.	*PROFAUDIENCE*
`condition`	No preassigned semantics, general purpose attribute.	*PROFCONDITION*
`conformance`	Conformance to standards, as for example `lsb` (Linux Standards Base).	*PROFCONFORMANCE*
`lang`	Language code, such as `de_DE`.	*PROFLANG*
`os`	Operating system.	*PROFOS*
`outputformat`	Output format to which the element applies (for example, print or EPUB). Only available for DocBook 5.1 or later.	*PROFOUTPUTFORMAT*

Attribute Name	Use	Profiling Parameter
`revision`	Editorial revision, such as `v2.1`.	*PROFREVISION*
`revisionflag`	Revision status of the element, such as `changed`. This attribute has a fixed set of values to choose from.	*PROFREVISIONFLAG*
`role`	General purpose attribute, with no preassigned semantics. As the role attribute is not solely "reserved" for profiling but can be used for other purposes in a document, we would recommend to not use this for profiling. For further details, see http://sagehill.net/docbookxsl/ProfilingWithRole.html.	*PROFROLE*
`security`	Security level, such as `high`.	*PROFSECURITY*
`status`	Editorial or publication status, such as `InDevelopment` or `draft`.	*PROFSTATUS*
`userlevel`	Level of user experience, such as `beginner`.	*PROFUSERLEVEL*
`vendor`	Product vendor.	*PROFVENDOR*
`wordsize`	Word size (width in bits) of the computer architecture, such as `64bit`. Added in DocBook version 4.4.	*PROFWORDSIZE*

MAIN File: Processing Instruction

To activate generation of profiled output in DAPS, the following processing instruction (PI) must be included in the header of the MAIN file, before the root element.

Adjusting the DocBook Version in the PI

As the profiling PI includes the DocBook version number, it needs to be adjusted according to the DocBook version you use. See Example 6.6, "Profiling PI in a DocBook 4.5 File" and Example 6.7, "Profiling PI in a DocBook 5.0 File".

Example 6.6. Profiling PI in a DocBook 4.5 File

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

Example 6.7. Profiling PI in a DocBook 5.0 File

```
<?xml-stylesheet
  href="urn:x-daps:xslt:profiling:docbook50-profile.xsl"
  type="text/xml" ?>
```

The MAIN file of a documentation project is the one that is referenced by the *MAIN* parameter in the DC file. If the processing instruction is missing in the MAIN file, any profiling parameters in the DC file will be ignored during generation of the output.

Include PI in All XML Files

Although DAPS only checks for the profiling PI in the MAIN file, it is a good idea to include the PI in all XML files for any projects that need profiling. Otherwise you might forget to move the PI to the respective MAIN file in case of restructuring the XML sources. Having the PI in all XML files does not hurt: Generation of profiled output is only triggered if your DC files contain profiling parameters.

DC Files: Profiling Parameters

Depending on the profiling attributes used in your XML files, a DC file may contain multiple profiling parameters, see Table 6.4, “Profiling Attributes (DocBook) and Profiling Parameters (DAPS)”. Profiling parameters define which of the profiling attributes should be interpreted by DAPS when generating output. For each profiling parameter, set the respective attribute values for which you want to filter during the profiling process. The spelling of the values must be the same that is used in the XML files.

Profiling Example

In the following, find a comprehensive example that shows the basic DAPS profiling requirements in more detail. It is based on the examples in the section called “Introduction to DocBook Profiling” about the fictional software `Frog Sound Recordings`. The software is available in a basic edition, a professional edition and a professional OEM edition, shipped by an OEM vendor. The following example shows *all* files that you need to consider (XML files, MAIN file, and DC file).

Example 6.8. XML File With Profiling Attributes (DocBook 4.x)

```
<?xml version="1.0" encoding="utf-8"?>
[...]

<chapter id="frog.features">
 [...]
 <simplelist>
   <member>Common Feature 1</member> ❶
   <member>Common Feature 2</member> ❶
   <member>Common Feature 3</member> ❶
   <member condition="basic">Basic Feature 1</member> ❷
   <member condition="prof">Professional Feature 1</member> ❸
   <member condition="prof">Professional Feature 2</member> ❸
   <member condition="prof" vendor="oemcompany">OEM Feature 1</member> ❹
 </simplelist>
 [...]
</chapter>
```

❶ Unprofiled list items. The common features 1-3 are available in all software editions or versions.

❷ List item profiled with attribute `condition` and attribute value `basic`. `Basic Feature 1` is only available in the basic software edition for home users.

❸ List item profiled with attribute `condition` and attribute value `prof`. `Professional Feature 1` and `Professional Feature 2` are only available in the professional software edition for enterprise customers.

❹ List item profiled with two attributes: Attribute `condition` with attribute value `prof` and attribute `vendor` with attribute value `oemcompany`. `OEM Feature 1` is only available in the professional OEM software edition for enterprise customers.

Example 6.9. MAIN file With PI for Profiling (DocBook 4.5)

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet
 href="urn:x-daps:xslt:profiling:docbook45-profile.xsl"
 type="text/xml"
 title="Profiling step"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
                         "http://www.docbook.org/xml/4.5/docbookx.dtd"
[<!ENTITY % entities SYSTEM "entity-decl.ent">
%entities;
]>
```

```
<!--the following article is contained in the file art_frog.xml-->

<article lang="en" id="art.frog">
 <title>Frog Sound Recordings</title>
 <subtitle>Product Description</subtitle>
 [...]
</article>
```

If the processing instruction (PI) is missing, any profiling parameters in the DC file will be ignored.

Example 6.10. MAIN file With PI for Profiling (DocBook 5.0)

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet
 href="urn:x-daps:xslt:profiling:docbook50-profile.xsl"
 type="text/xml"
 title="Profiling step"?>
<!DOCTYPE article
[<!ENTITY % entities SYSTEM "entity-decl.ent">
%entities;
]>

<!--the following article is contained in the file art_frog.xml-->

<article xml:lang="en" xml:id="art.frog"
   xmlns="http://docbook.org/ns/docbook">
 <title>Frog Sound Recordings</title>
 <subtitle>Product Description</subtitle>
 [...]
</article>
```

If the processing instruction (PI) is missing, any profiling parameters in the DC file will be ignored.

Use the PI `<?xml-stylesheet href="urn:x-daps:xslt:profiling:docbook51-profile.xsl" type="text/xml" title="Profiling step">` for DocBook version 5.1.

Example 6.11. DC File with Profiling for Home Edition

```
## Doc Config File for Frog Sound Recordings
## (Home Edition)

## Mandatory Parameters
MAIN="art_frog.xml" ❶

## Profiling
PROFCONDITION="basic" ❷
[...]
```

❶ *MAIN* parameter referencing the MAIN file. See Example 6.9 (DocBook 4) or Example 6.10 (DocBook 5).

❷ DAPS profiling parameter for the `condition` profiling attribute. It defines that XML elements tagged with `condition="basic"` are included in the output.

Example 6.12. DC File with Profiling for Professional Edition

```
## Doc Config File for Frog Sound Recordings
## (Professional Edition)
```

```
## Mandatory Parameters
MAIN="art_frog.xml" ❶

## Profiling
PROFCONDITION="prof" ❷
[...]
```

❶ *MAIN* parameter referencing the MAIN file. See Example 6.9 (DocBook 4) or Example 6.10 (DocBook 5).
❷ DAPS profiling parameter for the `condition` profiling attribute. It defines that XML elements tagged with `condition="prof"` are included in the output.

Example 6.13. DC File with Profiling for Professional Edition (OEM Version)

```
## Doc Config File for Frog Sound Recordings
## (Professional Edition, OEM Version)

## Mandatory Parameters
MAIN="art_frog.xml" ❶

## Profiling
PROFCONDITION="prof" ❷
PROFVENDOR="oemcompany" ❸
[...]
```

❶ *MAIN* parameter referencing the MAIN file. See Example 6.9, “MAIN file With PI for Profiling (DocBook 4.5)” or Example 6.10, “MAIN file With PI for Profiling (DocBook 5.0)” .
❷ DAPS profiling parameter for the `condition` profiling attribute. It defines that XML elements tagged with `condition="prof"` are included in the output.
❸ DAPS profiling parameter for the `vendor` profiling attribute. It defines that XML elements tagged with `vendor="oemcompany"` are included in the output.

Combining Entities and Profiling

For maximum flexibility in generating documentation variants from the same source, DAPS also supports the combination of entities and profiling. As you already learned in the section called “Declaring Entities in a Separate File”, it is useful for modularization purposes to declare entities in a separate file and to re-use it in multiple documentation projects.

For multiple use of entities like `&productname;` or `&productnumber;`, declare them in a separate file and add profiling within the entities as shown in Example 6.14. During generation of output, DAPS then automatically replaces the entities with different values during output, depending on the context.

Example 6.14. Separate Entity File with Profiling Attributes

```
<!--the following declarations are contained in the file entity-decl.ent -->

<!ENTITY productname
 '<phrase cond="basic">Frog Sound Recordings (Basic)</phrase>
  <phrase cond="prof">Frog Sound Recordings (Professional)</phrase>
  <phrase cond="prof" vendor="oemcompany">Gecko Sound Recordings (Professional)</phrase

<!ENTITY productnumber
 '<phrase cond="basic">1.0</phrase>
  <phrase cond="prof">4.2</phrase>
```

```
  <phrase cond="prof" vendor="oemcompany">4.21</phrase>'>
```

Procedure 6.1. Using Product Entities with Profiling in XML Files

After declaring the entities as shown in Example 6.14 you can use them throughout your documents. For DAPS to process them correctly, you only need to "introduce" the entities once within the `*info` element of each document as described below. For examples showing the result, see Example 6.9 (DocBook 4) or Example 6.10 (DocBook 5).

1. Open the XML file that contains the root element for your respective document (`set`, `book`, or `article`).
2. If you use DocBook 4, insert a `setinfo`, `bookinfo`, or `articleinfo` element, (respectively) within the root element.

   If you use DocBook 5, insert the generic `info` element within the root element of your documentation. (DocBook 5 does no longer distinguish between `setinfo`, `bookinfo`, or `articleinfo`).
3. Within *all* of the `*info` elements in your documentation, add the following elements: `productname` and `productnumber`.
4. Within the `productname` elements, add the `&productname;` entity.
5. Within the `productnumber` elements, add the `&productnumber;` entity.

Now that you have "introduced" the entities for each document, use the entities in the text wherever you need them.

Example 6.15. XML File with `&productname;` and `&productnumber;` Entities (DocBook 4.5)

```
<?xml version="1.0" encoding="utf-8"?>

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

<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
                         "http://www.docbook.org/xml/4.5/docbookx.dtd"
[<!ENTITY % entities SYSTEM "entity-decl.ent">
%entities;
 ]> ❷

<!--the following article is contained in the file art_frog.xml-->

<article lang="en" xml:id="art.frog">
 <title>Frog Sound Recordings</title>
 <subtitle>Product Description</subtitle>
  <articleinfo> ❸
   <productname>&productname;</productname> ❹
   <productnumber>&productnumber;</productnumber> ❺
 </articleinfo>
  <abstract>
   <para>
    &productname; &productnumber; is a software for recording, editing, ❻
    and mixing audio data.
```

```
   </para>
  </abstract>
 [...]
</article>
```

❶ Processing instruction (PI) in the header of the MAIN file. If it is missing, any profiling parameters in the DC file will be ignored.
❷ Reference to the separate entity declaration file (with a parameter entity).
❸ Element `articleinfo`.
❹ Element `productname` and entity `&productname;`.
❺ Element `productnumber` and entity `&productnumber;`.
❻ Paragraph containing the entities `&productname;` and `&productnumber;`.

Example 6.16. XML File with `&productname;` and `&productnumber;` Entities (DocBook 5.0)

```
<?xml version="1.0" encoding="utf-8"?>

<?xml-stylesheet
 href="urn:x-daps:xslt:profiling:docbook50-profile.xsl"
 type="text/xml"
 title="Profiling step"?> ❶

<!DOCTYPE article
[<!ENTITY % entities SYSTEM "entity-decl.ent">
%entities;
]> ❷

<!--the following article is contained in the file art_frog.xml-->

<article lang="en" id="art.frog">
 <title>Frog Sound Recordings</title>
 <subtitle>Product Description</subtitle>
  <info> ❸
   <productname>&productname;</productname> ❹
   <productnumber>&productnumber;</productnumber> ❺
 </info>
  <abstract>
   <para>
    &productname; &productnumber; is a software for recording, editing, ❻
    and mixing audio data.
   </para>
  </abstract>
 [...]
</article>
```

❶ Processing instruction (PI) in the header of the MAIN file. If it is missing, any profiling parameters in the DC file will be ignored.
❷ Reference to the separate entity declaration file (with a parameter entity).
❸ Element `info`.
❹ Element `productname` and entity `&productname;`.
❺ Element `productnumber` and entity `&productnumber;`.
❻ Paragraph containing the entities `&productname;` and `&productnumber;`.

In any output format, the entities (❹, ❺, ❻) will automatically be replaced with different values. The actual values depend on the profiling parameters contained in the DC file that you use for generating

the output. For an example, refer to Table 6.5. It shows output variants that can be generated from the XML code in Example 6.15 or Example 6.16 plus the entity declaration in Example 6.14 by using different DC files.

Table 6.5. Output Variants of Example 6.15 / Example 6.16 Combined With Example 6.14

DC File	Output
DC File with Profiling for Home Edition	Frog Sound Recordings (Basic) 1.0 is a software for recording, editing, and mixing audio data.
DC File with Profiling for Professional Edition	Frog Sound Recordings (Professional) 4.2 is a software for recording, editing, and mixing audio data.
DC File with Profiling for Professional Edition (OEM Version)	Gecko Sound Recordings (Professional) 4.21 is a software for recording, editing, and mixing audio data.

Chapter 7. Review and Translation Processes

This chapter describes how to simplify review and translation processes with DAPS:

- by including remarks, or draft watermarks in the output,
- or by transforming the multiple DocBook files in your project into one big XML file.

Including Remarks, or Draft Watermarks in the Output

DAPS offers several features to simplify review and translation processes. For example, you can insert `remark` elements in the source code (for editorial remarks or questions to the proofreader) and generate an output format that either includes or suppresses these remarks. You can also generate preview versions of your documentation with a `DRAFT` watermark appearing on the HTML or PDF output.

Availability of Advanced Output Options

Advanced output options are only supported for selected formats. For example, `--draft` and `--remarks` are only available in HTML, HTML-single, PDF, and EPUB output.

Using the `--remarks` option automatically turns draft mode on.

By default, DAPS adds a string to the base name of the output file to flag output formats generated with special options. Example file names are `*_draft_en.pdf` or `*_remarks_*_draft_en.pdf`.

Find a few example commands below:

Including Remarks in the Output

```
tux:~> daps -d PATH_TO_DC_FILE pdf --remarks
```

When generating PDFs with FOP, the contents of the remark elements is shown in italics within the text. XEP supports conversion of remark elements into PDF annotations. This feature is enabled in DAPS by default, but if you want XEP to treat remark elements like FOP does, you can change the respective DAPS parameter. In HTML, HTML-single and EPUB output, the contents of the remark elements is shown in red within the text.

Building PDFs with a `DRAFT` Watermark

```
tux:~> daps -d PATH_TO_DC_FILE pdf --draft
```

Generates a PDF that has a `DRAFT` watermark printed on each page.

Creating XML Big Files

Instead of sending multiple XML files to a proofreader for review, you can transform all files included in your `book` or `set` into one huge DocBook XML file (big file). Use the following command:

```
tux:~> daps -d PATH_TO_DC_FILE bigfile
```

DAPS resolves all XIncludes (replaces them with the referenced content) to create the big file. A message is shown where to find the generated output.

Chapter 8. Packaging and Deploying Your Documentation

This chapter describes:

- how to create TAR archives with all source files, including graphics,
- how to generate distributable HTML archives,
- and how to generate page, desktop, or document files to integrate your documentation in KDE and GNOME desktop environments.

For distributing your output formats in a convenient way, DAPS can automatically create TAR archives of the XML sources (including graphics) and various output formats. DAPS uses `bz2` for high compression of the archives and keeps the directory structure when generating the TAR archives.

Create source packages, HTML TAR archives, color PDFs, or JSP TAR archives with the **daps package-*** commands. By adding the respective options, you can additionally create page files, document files or desktop files for GNOME or KDE desktop environments.

Creating a TAR Archive with All Sources (Including Graphics)

Use the following command to create a distributable TAR archive containing the sources of the complete set, including the graphics:

```
tux:~> daps -d PATH_TO_DC_FILE package-src
```

Generating a Distributable HTML Archive

To generate HTML output and to automatically pack the HTML files, any graphics, and your CSS file into a TAR archive, use the following command:

```
tux:~> daps -d PATH_TO_DC_FILE package-html
```

Generating Desktop, Document, or Page Files

To create files that you can use for the help system of the GNOME and KDE desktop environments, use one of the following options:

- For the GNOME help system Yelp: `--pagefiles`
- For former Yelp versions: `--documentfiles`
- For the KDE3 help system: `--desktopfiles`

For example, to create a distributable HTML archive plus the files for GNOME yelp, use the following command:

```
tux:~> daps -d PATH_TO_DC_FILE package-html --pagefiles
```

Chapter 9. Customizing Layout of the Output Formats

This chapter includes:

- how to modify individual XSLT processor parameters,
- an external reference to a book that deals with common customizations to the DocBook stylesheets.

Modifying Individual XSLT Processor Parameters

If you use the default DocBook layout and want to adjust individual parameters, you can use the `--param "KEY=VALUE"` or the `--stringparam "KEY=VALUE"` option. Both options can be specified multiple times and are supported for the following subcommands (specifying the output formats):

- epub
- html
- man
- pdf
- text
- webhelp

The options will pass on values for XSLT parameters directly to the XSLT processor, which is useful to temporarily overwrite style sheet parameters such as margins, for example.

XSLTPARAM And `--xsltparam`

For compatibility reasons, DAPS still supports the use of `--xsltparam` on the command line, but this option will be removed in future versions. Use `--param "KEY=VALUE"` or `--stringparam "KEY=VALUE"` instead.

However, if you want to specify your settings in the DC file instead of using the command line options, the respective parameter is still named *XSLTPARAM*.

For larger or more complex modifications, such as adjustments of the title page layout, for example, it is advisable to develop your own set of stylesheets instead.

For a list of XSLT parameters to modify, refer to one of the following parameter references at http://docbook.sourceforge.net/release/xsl/current/doc/index.html:

- HTML Parameter Reference: http://docbook.sourceforge.net/release/xsl/current/doc/html/index.html
- FO Parameter Reference: http://docbook.sourceforge.net/release/xsl/current/doc/fo/index.html
- Man Pages Parameter Reference: http://docbook.sourceforge.net/release/xsl/current/doc/manpages/index.html

Example 9.1. Adjusting the Layout of Variable Lists

By default, the DocBook stylesheets display the contents of a `variablelist` as a table. To change this temporarily, set the parameter *`variablelist.as.table`* to the value `0` by executing the following command:

```
tux:~> daps -d   PATH_TO_DC_FILE html \
--param❶ "variablelist.as.table❷=0"❸ \
--param "variablelist.term.break.after=1" \
--stringparam❶ "paper.type=A4"
```

❶ As value of `--param` or `--stringparam`, add an `xsltproc` parameter in the form of *PARAM_NAME*=*VALUE*".

VALUE can be an integer or XPATH expression for `--param` and a string for `--stringparam`.

❷ Name of the parameter to adjust.

❸ Value for ❷. It will be passed on unmodified to the XSLT processor call that creates the `.fo` file or HTML from the profiled XML sources.

Alternatively, you can add the `XSLTPARAM` parameter to the DC file of your project and specify the parameters there. If doing so, also add `--stringparam` or `--param`.

Either specify all parameters in a single line:

```
XSLTPARAM="--stringparam PARAM_NAME=VALUE --param PARAM_NAME=VALUE"
```

or in multiple lines:

```
XSLTPARAM="--stringparam PARAM_NAME=VALUE"
XSLTPARAM="$XSLTPARAM --param PARAM_NAME=VALUE"
```

Specifying the Layout for ASCII Text Output

You can set parameters in `/etc/daps/config` or `~/.config/daps/dapsrc` using the *`TXT_PARAMS`* parameter. Alternatively, set them from command line, using the `--param`/`--stringparam` options. By default, the stylesheets specified with the *`STYLEROOT`* parameter or the `--styleroot` option are used for generating ASCII text output.

However, if you want to ignore any STYLEROOT definitions, set `TXT_IGNORE_STYLEROOT="yes"` in `/etc/daps/config` or `~/.config/daps/dapsrc` or use the `--ignore-styleroot` option on the command line:

```
tux:~> daps -d PATH_TO_DC_FILE text --ignore-styleroot
```

Customizing the DocBook Stylesheets

Useful tips and tricks around using DocBook and the DocBook stylesheets can be found in *The DoCookBook—Recipes for DocBook Developers and Writers*, available at http://doccookbook.sourceforge.net/.

Chapter 10. Configuring DAPS

This chapter shows:

- how to customize DAPS behavior with a local configuration file.

DAPS can be customized to a large degree: per system, per user, and per document. The configuration file `/etc/daps/config` lists all parameters that can be configured, including a short description for each parameter. Parameters are always defined as `KEY="VALUE"` pairs. Any parameter can be set in various locations, which are listed below in ascending order with regard to their hierarchy. If conflicting values are set for the same parameter, the value defined in the next higher hierarchy level takes precedence. Values defined on the command line always take precedence over values set in any other locations.

- `/etc/daps/config` (system-wide configuration file)
- `~/.config/daps/dapsrc` (user-specific configuration file)
- DC (doc config) file of the documentation project (for settings specific to a document or documentation set)
- on the fly at the command line by specifying options to a **daps** command.

For adjusting a few parameters that you want to set to custom values, do *not* edit the system-wide DAPS configuration file. Instead, check if the file `~/.config/daps/dapsrc` already exists. If not, create the file and modify it as described below:

1. Open both `~/.config/daps/dapsrc` and `/etc/daps/config` in a text editor.
2. Select which parameters you want to modify.
3. Copy *only those* to `~/.config/daps/dapsrc`.

Parameter Incompatibilities Between Configuration Files

If copying *all* parameters from `/etc/daps/config` to `~/.config/daps/dapsrc`, this will raise the risk of parameter incompatibilities as soon as you update to a higher DAPS version. As the settings in the custom DAPS configuration file will override the settings in `/etc/daps/config` by default, any parameter incompatibilities between the files may lead to unexpected behavior of DAPS.

4. Save `~/.config/daps/dapsrc` and test if your changes lead to the expected results.

For a list of all parameters including a short description of each parameter, see `/etc/daps/config` in your installed system or https://github.com/openSUSE/daps/blob/master/etc/config.in.

Chapter 11. Troubleshooting

This chapter lists common problems and possible solutions, sorted into categories.

Generating Output

11.1. Profiling does not work as expected?

1. Check the values of your profiling attributes in the XML files: They must use consistent spelling throughout a documentation project. If you assigned multiple values to a profiling attribute, check if the values are separated with a semicolon, for example, `os="linux;unix"`.

2. Check the DC file for your documentation project: Does it contain one or multiple *PROF** parameters? Otherwise DAPS does not know which profiling attributes to interpret. Do the *PROF** parameters match the profiling attributes used in the XML files? Do the values of the *PROF** parameters match the attribute values used in the XML files?

3. Check the MAIN file of your documentation projects: Does its header contain the following line?

   ```
   href="urn:x-daps:xslt:profiling:docbook45-profile.xsl" type="text/xml"
   ```

   If not, any profiling parameters in the DC file will be ignored during generation of the output.

For more details, refer to the section called “Profiling—Support for Document Variants” and http://www.sagehill.net/docbookxsl/Profiling.html.

Miscellaneous

11.1. Why does DAPS not talk to me?

By default DAPS only shows the result of the current subcommnd. To increase the verbosity run **daps** with the option `-v`, `-vv`, or `-vvv`. For the highest verbosity, use the `--debug` option.

11.2. Where to find the log files?

If you should run into problems with DAPS that you cannot classify, check the DAPS log files in *YOUR_DOC_DIR*`/build/`*BOOKNAME*`/log`. A complete log file of the latest **daps *subcommand*** that was executed is available in *YOUR_DOC_DIR*`/build/`*BOOKNAME*`/log/make_`*SUBCOMMAND*`.log`

In case of an error the complete log file will be shown on the screen (STDOUT).

11.3. Are all changes to DAPS backward-compatible?

No. If you have recently updated to a higher DAPS version and afterward experience strange effects that are difficult to debug, check your custom DAPS configuration file (`~/.config/daps/dapsrc`) against the system-wide configuration file (`/etc/daps/config`). Search for any parameters that may have changed. By default, the settings in the custom DAPS configuration file will override the settings in `/etc/daps/config`. Therefore any parameter incompatibilities between the files may lead to unexpected behavior of DAPS.

When switching from DAPS 1.x to DAPS @DAPS_VERSION@, especially check the syntax of any XSLT parameters that you are using (on the command line, in scripts or in DC files). If you have not adjusted the parameters to the new syntax, this may result in strange error messages. For details, refer to Chapter 10, *Configuring DAPS*.

Glossary

Antenna House Formatter
: See FO Formatter.

Conditional Text
: See Profiling.

DTD (Document Type Definition)
: The DTD (Document Type Definition) defines the exact elements, entities attributes and structure available in an XML or HTML document.

DOCTYPE
: The DOCTYPE, or Document Type *Declaration*, not to be confused with a Document Type Definition, contains the information on the DTD to use with an XML document. Therefore, it also defines which particular XML format is for the document.

DAPS (DocBook Authoring and Publishing Suite)
: DAPS provides authors of technical documentation with an easy-to-use tool chain to convert their DocBook documents into various output formats.

DocBook
: DocBook is a semantic markup language for technical documentation published as a DTD.

Entity
: An entity connects one or multiple characters with a unique identifier. One example where this is used is for escaping characters that are necessary for XML markup. A character such as & must be written as the entity `&` in XML.

  You can also declare custom entities.

FO Formatter
: Renders the XSL-FO files which are created by the DocBook XSL stylesheets into various output formats. The output format used most often is likely PDF. Formats that can usually be rendered into include:

  - Page description formats such as PDF, PostScript, and XPS.
  - Different raster and vector image formats such as PNG and SVG.
  - Text documents and Web page documents such as TXT, RTF, and HTML.
  - Internal formats of the formatter.

  Well-known formatters include Apache FOP, XEP, and Antenna House Formatter. Whereas the former is an open source product, the latter two are proprietary solutions. Antenna House Formatter is incompatible with DAPS.

FO (Formatting Objects)
: See XSL-FO.

FOP (Formatting Objects Processor)
: See FO Formatter.

Main Element
: Within this guide, main element refers to any XML element that is commonly used to create a coherent whole in an output format. In other words, either a `book`, an `article`, or a `set`.

PDF (Portable Document Format)
: PDF is a page description format created by Adobe Systems in 1993. Today, it is widely adopted as the standard format for digitally distributed page-oriented documents. A major advantage of PDF is that the formats can be reproduced identically across different platforms.

PI (Processing Instruction)
: PIs can be used to mark certain content as having to be treated differently by writing an instruction enclosed in <? and ?>. This is commonly used within (X)HTML Web pages to mark parts of the file as being written in server-side scripting language PHP.

  In DocBook, Processing Instructions can also be used for somewhat more mundane purposes, such as setting the background color of a preceding image.

Profiling
: Through profiling, you can easily adapt your documentation to different variants of a product. For example, a manufacturer of white-label products might appreciate being able to easily replace the brand name for the product they sell.

  It is possible to further this concept and even replace entire sections of text— for example, depending on a product's target group (if documentation is generated for the entry-level or for the professional version of a product).

Project
: A project consists of all the files that lie in a directory structure as required by DAPS, with the first directory level containing any DC files and subdirectories for `xml` files and `images`. When the first Main Element is built, an additional subdirectory called `build` will be created.

  Such a *project directory* may contain the source files for multiple main elements.

SVG (Scalable Vector Graphics)
: SVG is an XML-based vector graphics format, which is supported by most modern Web browsers.

  Vector graphics formats are different from traditional raster graphics in that they describe the exact shape of an object instead of using the lossy process of subdividing an object into individual raster points (such as pixels).

Stylesheet
: In the context of DocBook, the term *stylesheet* usually refers to the XSLT stylesheets used to transform DocBook documents into their respective output formats.

Transformation
: Data transformation converts data from a source data format into a destination data format. An example is the process of converting a DocBook XML document into HTML by using an XSLT processor.

Validation
: Validation refers to the process of checking whether an XML document is formally correct, for example, checking if all XML tags are properly closed and nested. This is done using a DTD or XML Schema.

  If a document is valid that does not mean that its contents are factually correct or that it is structured as you intended. However, validity does mean that a document can be further processed, for example by a Web browser, or an XSL processor.

XEP
: See FO Formatter.

XInclude
: XIncludes are references to other DocBook files. XIncludes can be used to split one large file into multiple smaller, more manageable files. For example, instead of having an entire book in a single file, you can create one central file from which you can reference individual chapter files.

  When using a version control system within your documentation process, having smaller files can help to avoid version conflicts if you and co-workers are working on different chapters of the same book.

XML Catalog
: XML Catalogs can be used to make DTDs available locally, so they do not need to be downloaded over the network every time they are accessed.

XML Parser
: Also known as an *XML Processor*, an *XML Parser* is used to provide the structural information contained in an XML file to another application.

XOP
: XOP (XML-binary Optimized Packaging) is a W3C (World Wide Web Consortium) recommendation on how to represent binary data inside XML documents.

XML (Extensible Markup Language)
: XML is a markup language with rules to encode documents into a form that is both human-readable and machine-readable.

XSL (Extensible Stylesheet Language)
: XSL is a collective noun used to refer to XSLT, XSL-FO, and the XML Path Language (XPath).
See Also XSLT, XSL-FO.

XSL-FO
: FO, XSL-FO or *Extensible Stylesheet Language-Formatting Objects* is a markup language used to mediate between other XML representations and a page formatting format such as PDF.
See Also XSLT.

XSLT (Extensible Stylesheet Language for Transformations)
: XSLT or *Extensible Stylesheet Language for Transformations* is a language based on XML. It is used to transform XML documents.

Appendix A. Migration of Existing DocBook Projects

This section provides instructions how to migrate existing DocBook projects so that you can use DAPS for managing and publishing them.

Procedure A.1. Making DocBook Projects Compatible with DAPS

1. If your XML files are distributed across several subdirectories, flatten the hierarchy and put all XML files directly into the `xml` subdirectory that is required by DAPS. See Required Directory Structure. Hosting multiple documentation projects in the same `xml` directory is fine as long as the file names are unique. You can put multiple MAIN files there.

2. If you have any XIncludes or entity declaration files, also put them into the `xml` subdirectory.

3. Depending on the file type of your source images, add them to the respective subdirectories in *`YOUR_DOC_DIR`*`/images/src`. The `image` directory and its substructure is required by DAPS. For details, refer to Required Directory Structure.

4. Make sure that the base names of your image files are unique. For details, refer to the section called “File Name Requirements”.

5. Adjust all references of image files, XIncludes, and entity declarations, in the existing XML files to match the structure required by DAPS. The references must not include any absolute or relative path, the plain file name is enough.

6. For each deliverable (book, article, set) that you want to generate from your XML files, create a Doc Config file. For more information, refer to the section called “Key Files”. Find a template for DC files in your installed system in `/usr/share/daps/init_templates/DC-file.template`.

Appendix B. Editor-specific Information

This chapter describes:

- the use of DocBook macros for Emacs,
- how to edit XML/DocBook Files with the Vim Editor
- how to integrate daps-susespell into jEdit.

Emacs—Macros for Inserting DocBook Elements

Most editors allow you to define or record macros which you can use for automatically inserting empty "skeletons" for a complex XML construct as illustrated by Example B.1, "A `varlistentry` Element".

Example B.1. A `varlistentry` Element

```
<varlistentry>
 <term></term>
  <listitem>
   <para></para>
  </listitem>
 </varlistentry>
```

For Emacs, DAPS already includes macros for adding DocBook elements such as `listitem`, `figure`, or `indexterm`. The macros are defined in `docbook_macros.el` and are added to your system during the installation of DAPS. They require that you use one of Emacs' main XML editing modes, either `nxml` or `psgml`.

Procedure B.1. Configuring Emacs to Use the DocBook Macros

1. To load the DocBook macros, open your Emacs customization file (`~/.emacs` or `~/.gnu-emacs`).
2. Insert the following line:

   ```
   (load "/usr/share/emacs/site-lisp/docbook_macros.el" t t)
   ```

3. Save the Emacs customization file and restart Emacs.

For an overview, which macros are available and how to use them, refer to http://en.opensuse.org/openSUSE:Documentation_Emacs_Docbook_Macros.

Editing XML/DocBook Files with the Vim Editor

Find information about on https://github.com/tbazant/xml-vim.

jEdit—Spell Check on the Fly

If you do not want to run **daps spellcheck** from the command line, you can also integrate daps-susespell (plus a custom aspell dictionary, if needed) into your XML editor, so that spelling is checked "on the

fly” during editing. Consult your editor's documentation on how to integrate a custom dictionary. If you use jEdit, proceed as outlined in Procedure B.2, “Integrating daps-susespell into jEdit”.

Procedure B.2. Integrating daps-susespell into jEdit

1. Install and activate the plug-in for spell checking:
   a. Start jEdit and select Plug-ins + Plug-in Manager.
   b. If the Spell Check plug-in is not already installed, install and activate it.
   c. Close and restart jEdit.
2. Configure the plug-in as follows:
   a. Select Plug-ins + Plug-in Options.
   b. In the left navigation pane, select Spell Check + General.
   c. Set Spell-checking engine to Aspell and select the Dictionary to use, for example `en_US`.
   d. If the desired dictionary does not appear in the drop-down box, install the respective aspell dictionary for the language and click Refresh list.
   e. In the left navigation pane, switch to Spell Check + Syntax handling.
   f. In the table, activate the markup entry and click Edit next to it.
   g. In the Token types picker, activate the following entries:
      - NULL
      - COMMENT1
      - LITERAL1
   h. In the left navigation pane, switch to Spell Check + Aspell Engine.
   i. Set the path to the Aspell executable file name. Select Enable markup mode.
   j. To use an additional custom aspell dictionary, specify the path to the custom dictionary in the text box below Additional parameters:

      `--extra-dicts=`*`PATH_TO_CUSTOM_DICT`*

      For example:

      ```
      --extra-dicts=/home/tux/custom_aspell.rws
      ```
   k. Confirm your settings in the plug-in options dialog with OK or Apply.
3. To execute a spell check during editing, select Plug-ins + Spell Check + Highlight misspelled words (or use the key combination assigned to that menu item).

Appendix C. What's New?

For a detailed list of new features, or any changes to DAPS, see the change log of the `daps` package in your installed system. The change log for the latest released DAPS version is also available online at https://github.com/openSUSE/daps/blob/master/ChangeLog.

Appendix D. GNU Licenses

This appendix contains the GNU General Public License version 2 and the GNU Free Documentation License version 1.2.

GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", be-

low, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a). You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b). You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c). If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a). Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b). Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c). Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, 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.

12. 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

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
one line to give the program’s name and an idea of what it does.
Copyright (C) yyyy name of author
```

```
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'. This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License [http://www.fsf.org/licenses/lgpl.html] instead of this License.

GNU Free Documentation License

Version 1.2, November 2002

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with

or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgments”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified

Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgments" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgments”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission

from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgments”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled “GNU
Free Documentation License”.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.