Publican 1.6

Users Guide

Publishing books, articles, papers and multi-volume sets with DocBook XML

Don Domingo

Brian Forté

Rüdiger Landmann

Joshua Oakes

Joshua Wulf

Publican 1.6 Users Guide
Publishing books, articles, papers and multi-volume sets with DocBook XML
Edition 1.6

Author	Don Domingo	ddomingo@redhat.com
Author	Brian Forté	bforte@redhat.com
Author	Rüdiger Landmann	r.landmann@redhat.com
Author	Joshua Oakes	joakes@redhat.com
Author	Joshua Wulf	jwulf@redhat.com
Editor	Brian Forté	bforte@redhat.com
Editor	Rüdiger Landmann	r.landmann@redhat.com
Technical Editor	Jeff Fearn	jfearn@redhat.com
	Josef Hruška	

This book will help you install **Publican**. It also provides instructions for using Publican to create and publish DocBook XML-based books, articles and book sets. This guide assumes that you are already familiar with DocBook XML.

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F1`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F7`** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click

[1] https://fedorahosted.org/liberation-fonts/

> **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **ssh *username*@*domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.
>
> The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.
>
> To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;
```

```
import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' won't cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: *https://bugzilla.redhat.com/bugzilla/enter_bug.cgi?product=Fedora&version=rawhide&component=publican* against the product **fedora**. When submitting a bug report, be sure to mention the manual's identifier: *publican*

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Introduction

Publican is a tool for publishing material authored in DocBook XML. This guide explains how to create and build books and articles using **Publican**. It is not a general DocBook XML tutorial; refer to *DocBook: The Definitive Guide* by Norman Walsh and Leonard Muellner, available at *http://www.docbook.org/tdg/en/html/docbook.html* for more general help with DocBook XML.

Publican began life as an internal tool used by Red Hat's Documentation Group (now known as Engineering Content Services). On occasion, this legacy is visible.

Design

Publican is a publication system, not just a DocBook processing tool. As well as ensuring your DocBook XML is valid, **Publican** works to ensure your XML is up to publishable standard.

The branding functionality allows you to create your own presentation rules and look, overriding many parts of the default style to meet your publishing needs. Choices executed in code, however, are not changeable.

Entities, for example, can be validly defined in any XML file. However, to ensure the DTD declaration is present, valid and standardized, **Publican** rewrites the declaration in every XML file before it builds a book or article. Consequently, all entities declared in all XML files are lost. **Publican**, therefore, requires you define entities in the ***Doc_Name*.ent** file (refer to *Section 3.1.6, "Doc_Name.ent"*).

As publishing workflows grow, unrestrained entity definition leads to entity duplication and other practices that cause maintenance difficulties. Consolidating entity definitions in a single, predictable place alleviates these maintenance issues and helps the automation of the build process stay robust.

Entities also present an essentially insurmountable obstacle to quality translation (refer to *Section 3.1.6.1, "Entities and translation"*). Consequently, while we are not reducing the ***Doc_Name*.ent** file's functionality, we are no longer considering requests to add functionality or features associated with entity use.

Installing Publican

1.1. Installing Publican on Linux operating systems

Important — Availability in repositories
The procedures documented in this section assume that **Publican** and its various dependencies are available in repositories to which your system has access.

1.1.1. Installing Publican on Fedora

1. Open a terminal.
2. Change to the root user: **`su -`**
3. Run the following command to install the *publican* package and the *publican-doc* documentation package:

```
yum install publican publican-doc
```

Several brand packages are available for use with **Publican**. Run the following command as the root user to install packages for building branded books:

```
yum install publican-brand
```

Replace *brand* with, for example, **`redhat`**, **`fedora`**, **`jboss`**, **`ovirt`**, or **`gimp`**. Refer to *Chapter 4, Branding* for more information on branding.

1.1.2. Installing Publican on Red Hat Enterprise Linux 5

Important — Unsupported software
Publican is not part of the Red Hat Enterprise Linux distribution. Therefore, Red Hat does not offer support for **Publican**.

Important — Dependencies available only internally to Red Hat
Installing **Publican** on Red Hat Enterprise Linux 5 requires a number of dependencies that are presently available only in yum repositories that are internal to Red Hat.

1. Open a terminal.
2. Change to the root user: **`su -`**
3. Run the following command to install the *publican* package and the *publican-doc* documentation package:

```
yum install publican publican-doc
```

Several brand packages are available for use with **Publican**. Run the following command as the root user to install packages for building branded books:

```
yum install publican-brand
```

Replace *brand* with, for example, **redhat**, **fedora**, **jboss**, **ovirt**, or **gimp**. Refer to *Chapter 4, Branding* for more information on branding.

1.1.3. Installing Publican on Ubuntu

Important — New in 10.4 "Lucid Lynx"

Publican is new in Ubuntu 10.4 "Lucid Lynx".

1. Open a terminal.
2. Run the following command to install the *publican* package:

```
sudo apt-get install publican
```

1.1.4. Installing Publican on Debian

Warning — Complete this procedure

Complete every step of this procedure. If you do not undo the changes that you make to the **/etc/apt/sources.list** file as described, your system might become unstable.

Publican is not available in the current stable version of Debian (version 5.0, "Lenny"), but is available in the current testing version ("Squeeze"). To install **Publican** on a computer that runs Debian, temporarily enable access to the **squeeze** repository. When you enable access to this repository, you allow your computer to install newer software and newer versions of existing software than what is available in the current stable version of Debian. However, not all of the software available in the testing repository has completed quality assurance testing yet. If you do not disable access to this repository after you install **Publican**, the next time that your system updates, it will replace software packages on your system with newer but possibly untested versions of those packages that it downloads from the testing repository.

1. Open a terminal.
2. Open your **/etc/apt/sources.list** file in a text editor. For example, to edit the file in **gedit** run:

```
sudo gedit /etc/apt/sources.list
```

3. Add the following line to the end of the file:

```
deb http://ftp.debian.org/debian/ squeeze main
```

4. Save the file and close the text editor.

5. Run the following command to update the list of packages available to your computer:

```
sudo apt-get update
```

6. Run the following command to install the *publican* package:

```
sudo apt-get install publican
```

7. Open your **/etc/apt/sources.list** file again, and delete the extra line that you added in this procedure.

Note that until the release of "Squeeze" as the stable version of Debian, you must manually enable and disable access to the testing repository as described in this procedure whenever a new version of **Publican** becomes available in the testing repository. You can find up-to-date information about the status of **Publican** for Debian at *http://packages.debian.org/squeeze/publican*, including the version number of **Publican** available in the repository (1.0 at the time of writing).

When "Squeeze" becomes the stable version of Debian, you will not need to enable or disable access to extra repositories to install **Publican** on systems that run that version of the operating system.

1.2. Installing Publican on Windows operating systems

1. Download the Publican installer from *https://fedorahosted.org/releases/p/u/publican/*.

2. Browse to the folder to which you downloaded **Publican-Installer-*version*.exe**.

3. Double-click the **Publican-Installer-*version*.exe** file.

4. The installer presents you with a series of license agreements. All of the files that constitute a **Publican** installation are available under a free license. However, because different licenses are more suitable for certain parts of **Publican** than others, the **Publican** files are not all available under the same free license. Each license grants you a different set of rights and responsibilities when you copy or modify the files in your **Publican** installation. We chose this combination of licenses to allow you to use **Publican** as freely as possible and to allow you to choose whatever license you prefer for the documents that you publish with **Publican**.

   Read the terms of the various license agreements. If you agree to their terms, click **I Agree** on each of them, otherwise, click **Cancel**.

5. The installer offers to install several components: **Publican** itself (labeled **Main** in the installer window), a number of *brands* (including **RedHat**, **JBoss**, and **fedora**), and two DocBook components (the DocBook *Data Type Definition* (DTD) and DocBook *Extensible Stylesheet Language* (XSL) stylesheets). The three brands are grouped under the collapsible heading **Brands** and the DocBook components are grouped under the collapsible heading **DocBook** in the installer window. Refer to *Chapter 4, Branding* for an explanation of brands in **Publican**.

Publican uses the DTD and the XSL stylesheets to render XML documents in other presentation formats (such as HTML and PDF). If you do not install these components, **Publican** must download this data from the Internet every time it processes a document, which creates lengthy delays.

All components are selected by default. Click the checkboxes to deselect any components that you do *not* require and click **Next** to continue.

6. By default, the installer software creates a folder named **Publican** within the ***%ProgramFiles%*** folder of your computer — typically **C:\Program Files\Publican**. You can manually edit the path displayed in the **Destination Folder** box to select a different folder.

7. When you are satisfied with the destination folder, click **Install**.

   The installer displays a progress bar as it installs **Publican**. To see more detailed information about the progress of the installation, click **Show details**.

8. When the process finishes, the installer notifies you with the message **Completed**.

   Click **Close** to close the installer.

Publican commands

Publican is a command-line tool. To use **Publican** on a computer with a Linux operating system, you must either start a terminal emulator program (such as **GNOME Terminal** or **Konsole**) or switch to a virtual console. To use **Publican** on a computer with a Windows operating system, run the **cmd** command from the **Start menu** to open a command prompt.

Publican commands take one of the following formats:

publican *command_option*
: The *command_option* is any of several options for the **publican** command itself.

publican *action action_options*
: The *action* is an action for **Publican** to perform, such as creating the XML files for a new document or building a HTML document from a document's XML files. The *action_options* apply to the *action*, such as specifying the language of a document.

publican *command_option action action_options*
: Some *command_options* affect the output of *actions*, for example, whether **Publican** should use ANSI colors in its output.

2.1. Command options

The options for the **publican** command are:

--help
: This option displays a help message, a condensed version of the contents of this chapter.

--man
: This option displays the man page for **Publican**, which includes the same information as the **--help** option supplies, in addition to information about licensing and dependencies.

--help_actions
: This option displays a list of valid **Publican** *actions*.

-v
: This option displays the version number of your **Publican** installation.

--config *file*
: This option allows you to specify a config file for a document, in place of the default **publican.cfg**.

--nocolours
: This option disables ANSI colors in **Publican** logging.

--quiet
: This option disables all logging.

2.2. Actions

Publican can perform the following actions:

build
: transforms XML to other formats (for example: PDF, single-page HTML, or multiple-page HTML). Refer to *Section 3.4, "Building a document"* for more details and a description of the available options.

clean
: removes all files and folders in the **`tmp/`** subdirectory. The **`tmp/`** subdirectory is created after running the **`publican build`** command to build a document, such as **`publican build --formats=html --langs=en-US`**.

clean_ids
: changes all IDs to a standard format. This format is *`Book_Name-title`*. For example, a section with a title of **`First Section`** in a book named **`Test_Book`** will have the following ID after running **`make clean_ids`**: `<section id="Test_Book-First_Section">`

Warning: `make clean_ids`

To make translation easier, **`make clean_ids`** uses the first four characters of the tag as a prefix for the ID. Consequently, you must check out the latest versions of the XML source and translations before running this command.

If you do not have the current versions of the PO files checked out before running **`make clean_ids`**, the XML and PO files will no longer in synchrony with each other. In this case, all links in the PO files must be manually updated.

cleanset
: removes local copies of remote books in a distributed set. Refer to *Section 5.2, "Distributed sets"* for details of using distributed sets.

create
: creates a new book, article, or set. Refer to *Chapter 3, Creating a document* for details of creating a book or article, and to *Chapter 5, Using sets* for details of using sets.

create_brand
: creates a new brand. Refer to *Section 4.2, "Creating a brand"* for details of creating a brand.

help_config
: displays help text for the configuration file contained in each book or brand, **`publican.cfg`**. Refer to *Section 3.1.1, "The publican.cfg file"* for more detail.

installbrand
: configures a brand for installation. Refer to *Section 4.1, "Installing a brand"* for details of installing a brand.

lang_stats --lang=*`language_code`*
: generates a translation report for the language specified by *`language_code`*.

old2new
: creates a **`publican.cfg`** file based on the **`Makefile`** of a book, article, or set originally created with a developmental version of **Publican** (versions up to and including **Publican 0.45**). Refer to *Section 3.4.1, "Building a document created with Publican 0"* for more detail.

package
: packages a book, article, set, or brand for shipping as an RPM package. Refer to *Section 3.5, "Packaging a book"* and *Section 4.4, "Packaging a brand"* for more detail.

print_banned
: prints a list of DocBook tags banned by **Publican**. Refer to *Appendix A, Disallowed elements and attributes* for a discussion of banned tags.

print_known
: prints a list of DocBook tags supported by **Publican**. *Supported* are those tags whose output has undergone at least cursory verification for quality when used in **Publican** — refer to *Appendix A, Disallowed elements and attributes*.

printtree
: prints a tree of the XML files included with the `<xi:include>` tag in a book, article, or set.

print_unused
: prints a list of the XML files *not* included with the `<xi:include>` tag in a book, article, or set.

update_po
: updates the *portable object* (PO) files. Refer to *Section 3.3, "Preparing a document for translation"* for more detail.

update_pot
: updates the *portable object template* (POT) files. Refer to *Section 3.3, "Preparing a document for translation"* for more detail.

Creating a document

This chapter describes creating books and articles: the main configuration files, example document files, and how to build a document.

Use the **publican create** command to create a new document, including all the necessary files for the document.

The following is a list of valid options that you can append to the **publican create** command. For example, **publican create --help**, **publican create --name New_Book**, and so on:

--help
print a list of all **publican create** command options.

--name *Doc_Name*
replace *Doc_Name* with the name of the book or article. This variable must not contain any spaces. For example, the command **create_book --name Test_Book** creates a book named **Test_Book** with all the necessary files to build the book, and sets the *BOOKID* in the **Test_Book.ent** file.

--lang *Language_Code*
replace *Language_Code* with the language code of the language in which the book or article will be authored. If you do not specify a language, **Publican** defaults to **en-US** (American English). The **--lang** option sets the *xml_lang* in the **publican.cfg** file. Refer to *Section 3.1.1, "The publican.cfg file"* for more information on **publican.cfg** parameters and *Appendix D, Language codes* for more detail on language codes.

--version *version*
replace *version* with the version number of the product that the book describes. For example, for Red Hat Enterprise Linux 5.1 you would use **5.1**. The default version is **0.1**. The **--version** option sets the <pubsnumber> tag in the **Book_Info.xml** or **Article_Info.xml** file. For more information refer to *Section 3.1.2, "Book_Info.xml"*.

--edition *edition*
replace *edition* with the edition number of the book. This number indicates to users when a new edition of the book is released. The initial *general availability* (GA) release of the book should be edition **1.0**. The default value is **0**. The **--edition** option sets the <edition> tag in the **Book_Info.xml** or **Article_Info.xml** file. For more information refer to *Section 3.1.2, "Book_Info.xml"*.

--product *Product_Name*
replace *Product_Name* with the product name. This variable must not contain any spaces. For example, set this to **Fedora** for core Fedora documentation, and the name of the product for other products, for example, **Fedora_Directory_Server**. The **--product** option sets the <product name> tag in the **Book_Info.xml** or **Article_Info.xml** and the *PRODUCT* in the ***Doc_Name*.ent** file.

--type Article --name *Article_Name*
create an article instead of a book. Replace *Article_Name* with the article name. This variable must not contain any spaces. The **--type** option sets the *type* in the **publican.cfg** file. Refer to *Section 3.1.1, "The publican.cfg file"* for more information on **publican.cfg** parameters.

--type Set --name *Set_Name*
create a set of documents instead of a book. Replace *Set_Name* with the set name. This variable must not contain any spaces. The **--type** option sets the *type* in the **publican.cfg** file. Refer to *Section 3.1.1, "The publican.cfg file"* for more information on **publican.cfg** parameters and to *Chapter 5, Using sets* for details on using sets.

--brand *brand*
replace *brand* with RedHat, fedora, JBoss, oVirt, or GIMP. The **--type** option sets the *brand* in the **publican.cfg** file. Refer to *Section 3.1.1, "The publican.cfg file"* for more information on **publican.cfg** parameters. This option requires the appropriate **Publican** brand package to be installed. For example, to build Red Hat branded books, you must install the *publican-redhat* package. Refer to *Section 4.1, "Installing a brand"* for instructions on installing brand packages for **Publican**. If you do not specify a brand, **Publican** uses its built-in, default brand. Refer to *Chapter 4, Branding* for more information.

Before running the **publican create** command, use the **cd** command to change into the directory where you want the book to be created. For example, to create a book named **Test_Book** in the **my_books/** directory, run the following commands:

```
cd my_books/
publican create --name Test_Book
```

To see the results of this command on a computer with a Linux operating system, run the following:

```
ls
```

The output should be similar to the following:

```
en-US publican.cfg
```

3.1. Files in the book directory

If you run the command **publican create --name Test_Book --lang en-US**, **Publican** creates a directory structure and required files, similar to the following:

- **publican.cfg**
- **en-US** (directory)
  - **Test_Book.xml**
  - **Test_Book.ent**
  - **Revision_History.xml**
  - **Preface.xml**
  - **Chapter.xml**
  - **Book_Info.xml**
  - **Author_Group.xml**
  - **images** (directory)

- **icon.svg**

3.1.1. The publican.cfg file

Note — Customizing output

If you maintain multiple versions of a document, you can create a configuration file for each version. When building or packaging the document, you can use the **--config** to specify a configuration file other than the **publican.cfg** file and therefore a different set of parameters to use in a particular build. For example:

```
publican build --formats html,pdf --langs en-US,de-DE,hu-HU --config community.cfg
```

The **publican.cfg** file configures build options, and is located in the root of the book directory. The following is an example **publican.cfg** file, with a description of **publican.cfg** parameters following afterwards:

```
# Config::Simple 4.59
# Mon Sep 28 16:38:14 2009

xml_lang: en-US
type: Book
brand: common
```

Default parameters

xml_lang

specifies the language of the source XML files, for example, **en-US**, as set by the **--lang** option for **publican create**.

type

specifies the type of document — a DocBook <article>, DocBook <book>, or DocBook <set>, as set by the **--type** option for **publican create**.

brand

sets the *brand* of the document, for example, **RedHat**, **fedora**, **JBoss**, **oVirt** or **GIMP** , as set by the **--brand** option for **publican create**. If you do not specify a brand, **Publican** uses its default brand. Refer to *Chapter 4, Branding* for more information.

Advanced parameters

arch

filters output by computer *architecture*. For example, if you set **arch: x86_64** in the **publican.cfg** file, **Publican** will only include XML elements tagged with the equivalent attribute, such as <para arch="x86_64">.

Use with caution

As with conditional tagging more generally, **arch** can cause great difficulties when translating documents. Refer to *Section 3.6.1, "Conditional tagging and translation"* for an explanation of the issues.

arch set for root nodes

If the root node of an XML file is excluded by the *arch* attribute, your document will not build, because empty files are not valid XML. For example, if **Installation_and_configuration-PPC.xml** contains a single chapter:

```
<?xml version='1.0' encoding='utf-8' ?>
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://
www.oasis-open.org/docbook/xml/4.5/docbookx.dtd" [
]>
<chapter id="chap-Installation_and_configuration_on_PowerPC" arch="PowerPC">
<title>Installation and configuration on PowerPC</title>

[text of chapter]

</chapter>
```

and this chapter is included in **User_Guide.xml** with an <xi:include> tag, the document will not build with **condition: x86** set in the **publican.cfg** file.

To exclude this chapter, add the *arch* attribute to the <xi:include> tag in **User_Guide.xml**, not to the <chapter> tag in **Installation_and_configuration-PPC.xml**.

xrefs and the arch attribute

If an <xref> points to content not included in the build due to the *arch* attribute, the build will fail. For example, with **arch: x86** set in the **publican.cfg** file, **publican build --formats=pdf --langs=en-US** will fail if the book has the tag <xref linkend="Itanium_installation"> pointing to <section id="Itanium_installation" arch="IA64">.

books
: specifies a space-separated list of books used in a remote set. Refer to *Section 5.2, "Distributed sets"* for more information on distributed sets.

brew_dist
: specifies the build target to use for building the desktop RPM package in **Brew**, Red Hat's internal build system. This parameter defaults to **docs-5E**. Refer to *Section 3.5.2, "The **publican package** command"* and *Section 4.4, "Packaging a brand"* for more information on building RPM packages.

chunk_first
: controls whether the first section should appear on the same page as its parent when rendered in HTML. To make the first section appear on the same page as its parent, set this parameter to **chunk_first: 1**. Otherwise, the parameter defaults to **0**, and the first section starts a new HTML page.

chunk_section_depth
: controls the point at which **Publican** no longer splits sub-subsections onto a new page when rendering HTML. By default, this value is set to **4**.

classpath
: sets the path to the *Java archive* (*jar*) files for **FOP**. **Publican** relies on Apache **FOP** — a Java application — to render documents as PDF files. The default path for **FOP**'s jar files on a computer with a Linux operating system is: **/usr/share/java/ant/ant-trax-1.7.0.jar:/usr/share/java/xmlgraphics-commons.jar:/usr/share/java/batik-all.jar:/usr/share/java/xml-commons-apis.jar:/usr/share/java/xml-commons-apis-ext.jar**

common_config
: sets the path to the **Publican** installation. The default location on a computer with a Linux operating system is **/usr/share/publican**. On a computer with a Windows operating system, the default location is **%SystemDrive%/%ProgramFiles%/publican** — most usually **C:/Program Files/publican**.

common_content
: sets the path to the **Publican** *Common Content* files. These files provide default formatting, plus some boilerplate text and generic graphics. The default location on a computer with a Linux operating system is **/usr/share/publican/Common_Content**. On a computer with a Windows operating system, the default location is **%SystemDrive%/%ProgramFiles%/publican/Common_Content** — most usually **C:/Program Files/publican/Common_Content**.

condition
: specifies conditions on which to prune XML before transformation. Refer to *Section 3.6, "Conditional tagging"* for more information.

Root nodes and conditional tagging

If the root node of an XML file is excluded with a conditional, your document will not build, because empty files are not valid XML. For example, if **Installation_and_configuration_on_Fedora.xml** contains a single chapter:

```
<?xml version='1.0' encoding='utf-8' ?>
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://
www.oasis-open.org/docbook/xml/4.5/docbookx.dtd" [
]>
<chapter id="chap-Installation_and_configuration_on_Fedora" condition="Fedora">
<title>Installation and configuration on Fedora</title>

[text of chapter]

</chapter>
```

and this chapter is included in **User_Guide.xml** with an <xi:include> tag, the document will not build with **condition: Ubuntu** set in the **publican.cfg** file.

To exclude this chapter, add a condition to the <xi:include> tag in **User_Guide.xml**, not to the <chapter> tag in **Installation_and_configuration_on_Fedora.xml**.

xrefs and conditional tagging

If an <xref> points to content not included in the build due to conditional tagging, the build will fail. For example, with **condition: upstream** set in the **publican.cfg** file, **publican build --formats=pdf --langs=en-US** will fail if the book has the tag <xref linkend="betasection"> pointing to <section id="betasection" condition="beta">.

confidential
: marks a document as confidential. When this parameter is set to **1**, **Publican** adds the text specified by the *confidential_text* parameter (by default, **CONFIDENTIAL**) to the foot of each HTML page and the head of every page in a PDF document. The default value is **0** (no header or footer).

confidential_text
: specifies the text to use when the *confidential* parameter is set to **1**. The default text is **CONFIDENTIAL**.

cvs_branch
: the CVS branch into which to import the SRPM. Specify this parameter when packaging a document with the **--cvs** option — refer to *Section 3.5.2, "The **publican package** command"*.

cvs_pck
: the CVS package into which to import the SRPM. Specify this parameter when packaging a document with the **--cvs** option — refer to *Section 3.5.2, "The **publican package** command"*.

cvs_root
: the CVS root into which to import the SRPM. Specify this parameter when packaging a document with the **--cvs** option — refer to *Section 3.5.2, "The **publican package** command"*.

debug
: controls whether **Publican** should display debugging messages as it works. When set to its default of **0**, **Publican** does not display debugging messages. Change this value to **1** to view these messages.

doc_url
: provides a URL for the documentation team for this package. In multi-page HTML output, **Publican** links to this URL at the top right of each page, through the **image_right.png** image in the **Common_Content/images** directory for the brand. This parameter defaults to **https://fedorahosted.org/publican**

docname
specifies the document name. If set, this value overrides the content of the <title> tag in the **Book_Info.xml** file when you package a document. This value must include only contain upper- and lower-case un-accented letters, digits, and the underscore and space characters ('a–z', 'A–Z', '0'–'9', and '_' and ' ').

dt_obsoletes
specifies the desktop packages that this package obsoletes. Refer to *Section 3.5, "Packaging a book"*.

dtdver
specifies the version of the DocBook XML *Document Type Definition* (DTD) on which this project is based. **Publican** defaults to version 4.5. The specification for DocBook XML DTD version 4.5 is available from *http://www.docbook.org/specs/docbook-4.5-spec.html*.

A different DTD might slow your build
When you install **Publican**, you also install a local copy of the DocBook XML DTD version 4.5 and accompanying *Extensible Stylesheet Language* (XSL). If you set a version of the DTD for which there is no local support, **Publican** must download the appropriate DTD and XSL from an online source every time that it builds the document. Building your document is delayed while this download takes place. The combined size of the required files is around 70 MB.

ec_id
sets the ID for an **Eclipse** help plugin. Every **Eclipse** help plugin must have a unique ID, and these generally follow Java package naming conventions — refer to *http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html*. By default, **Publican** sets the ID to *org.product.docname*. The ID that you set here also determines the directory name for this plugin in the **plugin** directory.

ec_name
sets the name of an **Eclipse** help plugin. This is the human-readable name visible in the help list in **Eclipse**. This name does not need to be unique or to follow a special convention. By default, **Publican** sets the name to *product docname*.

ec_provider
sets the provider name for an **Eclipse** help plugin. This should be your name, or the name of your project or organization. This name is presented to users and does not need to be unique or follow a special convention. By default, **Publican** sets the provider name to *Publican-Publican version*.

edition
specifies the edition number for this document. If set, this value overrides the content of the <edition> tag in the **Book_Info.xml** file when you package a document. This value must include only digits and the period ('0'–'9' and '.').

generate_section_toc_level
controls the section depth at which **Publican** will generate a table of contents. At the default value of **0**, **Publican** will generate tables of contents at the start of the document and in parts, chapters, and appendixes, but not in sections. If (for example) the value is set to **1**, tables of contents also

appear in each "level 1" section, such as sections 1.1, 1.2, 2.1, and 2.2. If set to **2**, tables of contents also appear in "level 2" sections, such as sections 1.1.1, 1.1.2, and 1.2.1.

ignored_translations
: specifies translations to ignore. If you build or package a book in a language specified by this parameter, **Publican** ignores any translations that exist for this language, and builds or packages the book in the language of the original XML instead. Refer to *Section 3.3, "Preparing a document for translation"*.

license
: specifies the license this package uses. By default, **Publican** selects the GNU Free Documentation License (GFDL). Refer to *Section 3.5, "Packaging a book"*.

max_image_width
: specifies the maximum width allowable for images in the document in pixels. By default, **Publican** scales down any images wider than 444 pixels so that they fit within this limit. Keeping images no wider than 444 pixels ensures that they present no wider than the right-hand margin of the text in HTML output and that they fit within the pages of PDF output.

Important — 444 pixels is the maximum safe width

Do not use the *max_image_width* parameter if your images contain important information. Images wider than 444 pixels might lead to poorly presented HTML and to PDF output that it is unusable because images have run off the page and are presented incomplete to the reader.

Conversely, images lose quality when scaled down in HTML and PDF.

To safeguard the quality of your images, crop or scale them so that they are no wider than 444 pixels before including them in a document.

os_ver
: specifies the operating system for which to build packages. **Publican** appends the value that you provide here to the RPM packages that it builds. For example, the default value is **.el5**, which signifies Red Hat Enterprise Linux 5 and operating systems derived from it. Refer to *Section 3.5, "Packaging a book"* and *Section 4.4, "Packaging a brand"*.

prod_url
: provides a URL for the product to which this document applies. In multi-page HTML output, **Publican** links to this URL at the top left of each page, through the **image_left.png** image in the **Common_Content/images** directory for the brand. This parameter defaults to **https://fedorahosted.org/publican**.

product
: specifies the product to which this documentation applies. If set, this value overrides the content of the <productname> tag in the **Book_Info.xml** file when you package a document. This value must include only contain upper- and lower-case un-accented letters, digits, and the underscore and space characters ('a–z', 'A–Z', '0'–'9', and '_' and ' ').

release
: specifies the release number of this package. If set, this value overrides the value of <pubsnumber> in the **Book_Info.xml** file when you package a document. This value must include only digits (‘0’–‘9’).

repo
: specifies the repository from which to fetch remote books that form part of a distributed set. Refer to *Section 5.2, “Distributed sets”*.

scm
: specifies the version control (or *source code management*) system used in the repository in that stores the remote books in a distributed set. At present, **Publican** can use only **Subversion** (SVN), and therefore uses **SVN** as its default setting. Refer to *Section 5.2, “Distributed sets”*.

show_remarks
: controls whether to display remarks in transformed output. By default, this value is set to **0**, which causes **Publican** to hide remarks. Set this value to **1** to display remarks.

show_unknown
: controls whether **Publican** reports unknown tags when processing XML. By default, this value is set to **1**, so **Publican** reports unknown tags. Set this value to **0** to hide this output. ~~**Publican** ignores this parameter in *strict mode*.~~

src_url
: specifies the URL at which to find tarballs of source files. This parameter provides the **Source:** field in the header of an RPM spec file. Refer to *Section 3.5, “Packaging a book”*.

strict
: ~~sets **Publican** to use *strict mode*, which prevents the use of tags that are unusable for professional output and translation. By default, the **strict** parameter is set of **0**, which disables strict mode. To enable strict mode, set this parameter to **1**~~ Strict mode is no longer enforced.

tmp_dir
: specifies the directory for **Publican** output. By default, this is set to **tmp**, which creates a directory named **tmp** inside the directory that holds your article or book.

toc_section_depth
: controls the depth of sections that **Publican** includes in the main table of contents. By default, this value is set to **2**. With the default setting, sections 1.1 and 1.1.1 will appear in the main table of contents, but section 1.1.1.1 will not. (Note that the first digit in these examples represents a chapter, not a section).

version
: specifies the version number of that product to which this document applies. If set, this value overrides the content of the <productnumber> tag in the **Book_Info.xml** file when you package a document. This value must include only digits and the period (‘0’–‘9’ and ‘.’).

web_brew_dist
: specifies the **brew** build target to use for building the web RPM packages. **Brew** is the internal build system used by Red Hat. By default, this value is set to **docs-5E**, representing documentation packages for Red Hat Enterprise Linux 5. Refer to *Section 3.5, “Packaging a book”*.

web_obsoletes
specifies packages that the web RPM obsoletes. Refer to *Section 3.5, "Packaging a book"*.Refer to *Section 3.5, "Packaging a book"*.

Help from the command line
Run the **publican help_config** command in the root directory of a book for a summary of these parameters.

3.1.2. Book_Info.xml

Article_Info.xml and Set_Info.xml
This description of the **Book_Info.xml** file applies to **Article_Info.xml** and **Set_Info.xml** files too. However, for the sake of simplicity, the file is referred to as **Book_Info.xml** throughout this section.

Packages other than RPM packages
This section discusses packaging documents for distribution through the **RPM Package Manager**. However, when you use the **publican package** command, **Publican** generates a tarball that you can use to build a package to distribute through different package manager software. If you run **publican package** on a computer on which **rpmbuild** is not installed, **Publican** still generates the tarball, even though it cannot then generate an RPM package from that tarball.

The **Book_Info.xml** file contains the key metadata concerning a book: the book's ID; title; subtitle; author and edition number. It also contains the name and version of the product that is documented, and an abstract.

Aside from constituting much of a book's front matter, this metadata is also used when building books as RPM packages. Usually, if you distribute a book as an RPM package, several of the tags included by default in **Book_Info.xml** must have appropriate data within them, and that data must conform to the requirements of the RPM format. You can override the data in these tags by using equivalent fields in the **publican.cfg** file, as discussed in this section.

Unless overridden in the **publican.cfg** file, data from seven of the default tags in **Book_Info.xml** is required to build books as RPMs. Most immediately, the file name of a book built as an RPM package is constructed as follows: ***productname-title-productnumber-language-edition-pubsnumber*.src.rpm**. Everything but *language* above is pulled from **Book_Info.xml** — you specify *language* when you build the book. As well, the <subtitle> and <abstract> are used in the RPM spec file, to provide the *Summary:* field in the header and the *%description* field, respectively.

An example **Book_Info.xml** file, for the **Test_Book** book, is presented below. Details regarding this file, and what the RPM format requirements are for each tag, follow.

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

```
<!DOCTYPE bookinfo PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-open.org/
docbook/xml/4.5/docbookx.dtd" [
<!ENTITY % BOOK_ENTITIES SYSTEM "Users_Guide.ent">
%BOOK_ENTITIES;
]>
<bookinfo id="book-Users_Guide-Users_Guide" lang="en-US">
 <title>Users Guide</title>
  <subtitle>Publishing books, articles, papers and multi-volume sets with DocBook XML</
subtitle>
  <productname>Publican</productname>
  <productnumber>1.6</productnumber>
  <edition>1.6</edition>
  <pubsnumber>1</pubsnumber>
  <abstract>
  <para>
   This book will help you install <application>Publican</application>. It also provides
 instructions for using Publican to create and publish DocBook XML-based books, articles and
 book sets. This guide assumes that you are already familiar with &D_B; XML.
  </para>

 </abstract>
  <keywordset>
  <keyword>publican</keyword>
   <keyword>docbook</keyword>
   <keyword>publishing</keyword>

 </keywordset>
  <subjectset scheme="libraryofcongress">
  <subject>
   <subjectterm>Electronic Publishing</subjectterm>

  </subject>
   <subject>
   <subjectterm>XML (Computer program language)</subjectterm>

  </subject>

 </subjectset>
  <corpauthor>
  <inlinemediaobject>
   <imageobject>
    <imagedata fileref="Common_Content/images/title_logo.svg" format="SVG" />
   </imageobject>
    <textobject>
    <phrase>Team Publican</phrase>
   </textobject>

  </inlinemediaobject>

 </corpauthor>
  <mediaobject role="cover">
  <imageobject remap="lrg" role="front-large">
   <imagedata fileref="images/cover_thumbnail.png" format="PNG" width="444" />
  </imageobject>
   <imageobject remap="s" role="front">
   <imagedata fileref="images/cover_thumbnail.png" format="PNG" width="444" />
  </imageobject>
   <imageobject remap="xs" role="front-small">
   <imagedata fileref="images/cover_thumbnail.png" format="PNG" width="444" />
  </imageobject>
   <imageobject remap="cs" role="thumbnail">
   <imagedata fileref="images/cover_thumbnail.png" format="PNG" width="444" />
  </imageobject>
```

```
 </mediaobject>
  <xi:include href="Common_Content/Legal_Notice.xml" xmlns:xi="http://www.w3.org/2001/
XInclude" />
  <xi:include href="Author_Group.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
</bookinfo>
```

`<bookinfo id="`*`book_id`*`">`, `<articleinfo id="`*`article_id`*`">`, `<setinfo id="`*`set_id`*`">`
: The document ID is used internally and is not displayed to readers when the book is built. If you run the **`publican clean_ids`** command, any manually entered ID, including this one, changes to a *`Doc_Name-Title`* format, where *`Title`* is the title of the associated book, article, section, or chapter.

`<productname>`*`productname`*`</productname>`
: The name of the product or product stream to which the book, article, or set applies, for example: **`Red Hat Enterprise Linux`** or **`JBoss Enterprise Application Platform`**. When building a book as an RPM package, data in the <productname> tag is used as part of the file name of the package.

  Override this tag with the *`product`* variable in the **`publican.cfg`** file if the name of your product contains non-Latin characters, accented Latin characters, or punctuation marks other than the underscore.

Permitted characters

Publican uses data in this tag to generate part of the file name for RPM packages, unless overridden by data in the **`publican.cfg`** file. If you do not override this tag in the **`publican.cfg`** file, this tag must only contain upper- and lower-case un-accented letters, digits, and the underscore and space characters (‘a–z’, ‘A–Z’, ‘0’–‘9’, and ‘_’ and ‘ ’) if you plan to build packages with **Publican**.

`<title>`*`title`*`</title>`
: Obviously enough, the title of the document (for example, *Server Configuration Guide*). The title appears under the product name in both HTML and PDF editions. A title is required to build an RPM package. When building a book as an RPM package the title is used as the part of the file name of the package.

  Override this tag with the *`docname`* variable in the **`publican.cfg`** file if the title of your document contains non-Latin characters, accented Latin characters, or punctuation marks other than the underscore.

Permitted characters

Publican uses data in this tag to generate part of the file name for RPM packages, unless overridden by data in the **`publican.cfg`** file. If you do not override this tag in the **`publican.cfg`** file, this tag must only contain upper- and lower-case un-accented letters, digits, and the underscore and space characters (‘a–z’, ‘A–Z’, ‘0’–‘9’, and ‘_’ and ‘ ’) if you plan to build packages with **Publican**.

`<subtitle>`*`subtitle`*`</subtitle>`

Again, plainly enough, the subtitle tag contains a book's subtitle: an alternative, and commonly explanatory title for the book (for example: *Server Configuration Guide: Preparing Red Hat Enterprise Linux for Production Use*). The subtitle appears under the title in both HTML and PDF editions. A subtitle is also required to make a book available as an RPM package. When building a book as an RPM package, the subtitle is used as the *Summary* in the RPM spec file. The **rpm -qi** returns the contents of several spec file fields, including the **Summary** field.

`<productnumber>`*`productnumber`*`</productnumber>`

The version number of the product the book covers, for example '5.2' for Red Hat Enterprise Linux 5.2 and '4.3' for JBoss EAP 4.3.

Running the **publican create --name *Doc_Name* --version *version*** command correctly configures the product number.

Override this tag with the *version* variable in the **publican.cfg** file if the product version is anything other than a number.

Permitted characters

Publican uses data in this tag to generate part of the file name for RPM packages, unless overridden by data in the **publican.cfg** file. If you do not override this tag in the **publican.cfg** file, this tag must only contain numbers and the period ('0–9' and '.') if you plan to build packages with **Publican**.

`<edition>`*`edition`*`</edition>`

This is the edition number of the book. The first edition of the book should be 1.0 (unless you use 0.*x* for pre-release versions of a book). Subsequent editions should increment the 1.*x* to indicate to readers that the book is a new edition. The edition changes the version number in the file name when building a book with the **publican package** command.

For example, setting the edition to **1.2** and building the book using the **publican package --binary --lang=en-US** command creates an RPM file named ***productname*-*title*-*productnumber*-en-US-1.2-0.src.rpm**.

Running the **publican create --name *Doc_Name* --edition *x.y*** command correctly configures the edition.

Override this tag with the *edition* variable in the **publican.cfg** file if the edition of your document is identified by anything other than a number.

Permitted characters

Publican uses data in this tag to generate part of the file name for RPM packages, unless overridden by data in the **publican.cfg** file. If you do not override this tag in the **publican.cfg** file, this tag must only contain numbers and the period ('0–9' and '.') if you plan to build packages with **Publican**.

`<pubsnumber>`*`pubsnumber`*`</pubsnumber>`

The pubsnumber changes the release number (the last digit) when building a book with the **publican package** command. For example, setting the pubsnumber to **1** and building the book

using the **publican package --binary --lang=en-US** command creates an RPM file named ***productname*-*title*-*productnumber*-en-US-*edition*-1.src.rpm**.

Override this tag with the *release* variable in the **publican.cfg** file if the release number of your document contains anything other than whole numbers.

Permitted characters

Publican uses data in this tag to generate part of the file name for RPM packages, unless overridden by data in the **publican.cfg** file. If you do not override this tag in the **publican.cfg** file, this tag must only contain numbers ('0–9') if you plan to build packages with **Publican**.

`<abstract><para>`*abstract*`</para></abstract>`

A short overview and summary of the book's subject and purpose, traditionally no more than a paragraph long. The abstract appears before the table of contents in HTML editions and on the second page of PDF editions. When a book is built as an RPM package, the abstract sets the *Description* field of the RPM's spec file. This makes the abstract for a package available via the **rpm -qi** command.

You can add extra metadata to the **Book_Info.xml** file of a document, to support specific features in various output formats:

`<keywordset>`, `<keyword>`

Terms tagged with `<keyword>` and placed within a `<keywordset>` are added to a `<meta name="keywords">` entry in the head of HTML files and to the **Keywords** field of the properties of a PDF document.

`<subjectset>`, `<subject>`

Terms tagged with `<subject>` and placed within a `<subjectset>` are added to the **Subject** field of the properties of a PDF document and in the metadata of an ebook in EPUB format.

Consider using a *controlled vocabulary* when defining the subject of your document, for example, the *Library of Congress Subject Headings* (LCSH). Identify the chosen vocabulary with the *scheme* attibute in the `<subjectset>` tag, for example, `<subjectset scheme="libraryofcongress">`. You can search for LCSH subject headings through the Library of Congress *Authorities & Vocabularies* page: *http://id.loc.gov/authorities/search/*.

`<mediaobject role="cover" id="epub_cover">`

Use a `<mediaobject>` tag with the **role="cover"** and **id="epub_cover"** attributes to set cover art for an ebook in EPUB format. For example:

```
<mediaobject role="cover" id="epub_cover">
 <imageobject role="front-large" remap="lrg">
  <imagedata width="600px" format="PNG" fileref="images/front_cover.png"/>
 </imageobject>
 <imageobject role="front" remap="s">
  <imagedata format="PNG" fileref="images/front_cover.png"/>
 </imageobject>
 <imageobject role="front-small" remap="xs">
  <imagedata format="PNG" fileref="images/front_cover.png"/>
 </imageobject>
 <imageobject role="thumbnail" remap="cs">
  <imagedata format="PNG" fileref="images/front_cover_thumbnail.png"/>
```

```
 </imageobject>
</mediaobject>
```

As with all the other images in your document, place the cover images in the **images** subdirectory.

3.1.2.1. RPM packages, editions, impressions and versions

As noted above, the default **Book_Info.xml** used by **Publican** includes an <*edition*> tag.

If you distribute a book as an RPM package, the data placed within this tag sets 'the first two digits of the version number in the RPM file name.

So, an edition of '1.0' becomes a version of '1.0'.

Book_Info.xml also includes the <*pubsnumber*> tag. Any data placed within this tag changes the release number of RPM-packaged books.

A book with an edition of 1.0 and a pubsnumber of 5, would be version 1.0, release 5 (1.0-5).

The edition and pubsnumber are not tied to the <*productnumber*> tag also found in **Book_Info.xml**: <*productnumber*> denotes the version number of the product being documented or otherwise written about.

It is entirely possible to have a 2nd edition of a book pertaining to a particular version of a product.

In bibliography, two copies of a book are the same edition if they are printed using substantially the same type-set master plates or pages. ('Substantially' offers some allowance for typo corrections and other inconsequential changes.)

Book collectors routinely conflate 'first edition' with 'first print run', while bibliographers pay attention to the text commonly placed in the front matter of a book, which calls a 2nd print run off the same (or substantially the same) plates a '1st edition, 2nd impression' or '1st edition, 2nd printing'.

We recommend following bibliographic practice in this regard. When using **Publican** to re-publish a book from 'substantially the same XML', increment the <*pubsnumber*> tag, not the <*edition*> tag. It functions as a near-equivalent to the impression or printing number of traditional publishing.

As for changing the edition number, we recommend changing this in the same circumstances traditional publishers change the edition of a work: when it is revised and re-written significantly. What constitutes significant, and how much re-writing is needed to increment an edition number by a whole number and how much is needed to increment it by one-tenth of a whole number, is a matter of editorial discretion.

3.1.3. Author_Group.xml

Author_Group.xml is not required but is the standard place to record author, editor, artist and other credit details. The following is an example **Author_Group.xml** file:

```
<?xml version='1.0'?>
<!DOCTYPE authorgroup PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-open.org/
docbook/xml/4.5/docbookx.dtd" [
]>

<authorgroup>
 <corpauthor>FF0000 Headgear Documentation Group</corpauthor>
 <author>
```

```
  <firstname>Dude</firstname>
  <surname>McDude</surname>
  <affiliation>
   <orgname>My Org</orgname>
   <orgdiv>Best Div in the place</orgdiv>
  </affiliation>
  <email>dude.mcdude@myorg.org</email>
 </author>
</authorgroup>
```

Author_Group.xml does not have to contain all of the above information: include as much or as little as required.

3.1.4. Chapter.xml

Articles and chapters

DocBook articles cannot contain chapters. If you use the **--type=article** option with **publican create**, **Publican** does not create a **Chapter.xml** file. Use sections to organize content within articles.

Refer to *DocBook: The Definitive Guide* by Norman Walsh and Leonard Muellner available at *http://www.docbook.org/tdg/en/html/docbook.html* for details of the different ways that sets, books, articles, parts, chapters, and sections interact. In particular, note that articles can be stand-alone documents, or can be incorporated into books.

The **Chapter.xml** file is a template for creating chapter files. Chapter files contain the content that make up a book. The following is a chapter template (**Chapter.xml**) that is created by the **publican create** command. Note the **DOCTYPE** is set to **chapter**:

```
<?xml version='1.0'?>
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-open.org/
docbook/xml/4.5/docbookx.dtd" [
]>

<chapter id="MYBOOK-Test">
 <title>Test</title>
 <para>
  This is a test paragraph
 </para>
 <section id="MYBOOK-Test-Section_1_Test">
  <title>Section 1 Test</title>
  <para>
   Test of a section
  </para>
 </section>

 <section id="MYBOOK-Test-Section_2_Test">
  <title>Section 2 Test</title>
  <para>
   Test of a section
  </para>
 </section>

</chapter>
```

This chapter has two sections, **Section 1 Test** and **Section 2 Test**. Refer to *http://docbook.org/tdg/en/html/chapter.html* for further information about chapters.

Note

The chapter file should be renamed to reflect the chapter subject. For example, a chapter on product installation could be named **Installation.xml**, whereas a chapter on setting up a piece of software would be better called **Setup.xml** or **Configuration.xml**.

3.1.5. Doc_Name.xml

The ***Doc_Name*.xml** file contains **xi:include** directives to include the other necessary XML files for the document, including chapters or sections contained in other XML files. For example, a book's ***Doc_Name*.xml** file brings together chapters that are contained in separate XML files.

The following is an example ***Doc_Name*.xml** file that describes a DocBook book — note the *DOCTYPE* is set to **book**.

```
<?xml version='1.0'?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-open.org/docbook/
xml/4.5/docbookx.dtd" [
]>

<book>
 <xi:include href="Book_Info.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <xi:include href="Preface.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <xi:include href="Chapter.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <xi:include href="Revision_History.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <index />
</book>
```

This example loads the **Book_Info.xml**, **Preface.xml**, **Chapter.xml**, and **Appendix.xml** XML files.

Important

The order in which chapters are listed matters. When this example book is built, **Book_Info.xml** will precede **Preface.xml** which will precede **Chapter.xml**, and so on.

The ***Doc_Name*.xml** file is not limited to using **xi:include** directives. You can create documents with a single XML file. The following is an example of a book created using a single XML file:

```
<book>

<chapter>
```

```
<title>Chapter 1</title>
<para>
 A paragraph in Chapter 1.
</para>
<section id="section1">
<title>Chapter 1 Section 1</title>
 <para>
  A paragraph in Section 1.
 </para>
</section>
<section id="section2">
<title>Chapter 1 Section 2</title>
 <para>
  A paragraph in Section 2.
 </para>
</section>
</chapter>

<chapter>
<title>Chapter 2</title>
<para>
 A paragraph in Chapter 2.
</para>
</chapter>

</book>
```

This book contains two chapters. Chapter one contains two sections. Refer to *http://docbook.org/tdg/en/html/section.html* for further information about sections, and *http://docbook.org/tdg/en/html/book.html* for further information about books.

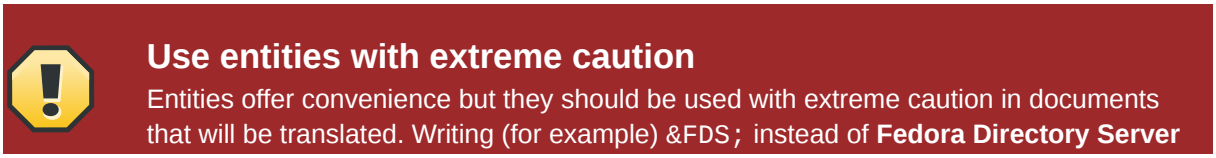
3.1.6. Doc_Name.ent

The ***Doc_Name*.ent** file is used to define local entities. The *YEAR* and *HOLDER* entities are used for copyright information. By default, **Publican** sets *YEAR* to the current year, and inserts a message into *HOLDER* to remind you to specify the copyright holder for the document. If the *YEAR* and *HOLDER* entities are missing altogether, the document will not build.

Other entities might be required by the *brand* applied to your document. For example, the **Publican** brand for Fedora documents uses the entity **BOOKID** to specify how readers should refer to a document when they submit feedback about it.

```
<!ENTITY PRODUCT "MYPRODUCT">
<!ENTITY BOOKID "MYBOOK">
<!ENTITY YEAR "2008">
<!ENTITY HOLDER "YOUR NAME GOES HERE">
```

3.1.6.1. Entities and translation



saves the writer time but transformed entities do not appear in the *portable object* (PO) files that translators use. Complete translations of documents containing entities are, as a consequence, impossible.

Entities present special obstacles to translators and can preclude the production of high-quality translations. The very nature of an entity is that the word or phrase represented by the entity is rendered exactly the same way every time that it occurs in the document, in every language. This inflexibility means that the word or word group represented by the entity might be illegible or incomprehensible in the target language and that the word or word group represented by the entity cannot change when the grammatical rules of the target language require them to change. Furthermore, because entities are not transformed when XML is converted to PO, translators cannot select the correct words that surround the entity, as required by the grammatical rules of the target language.

If you define an entity — `<!ENTITY LIFT "Liberty Installation and Formatting Tome">` — you can enter **`&LIFT;`** in your XML and it will appear as **`Liberty Installation and Formatting Tome`** every time the book is built as HTML, PDF or text.

Entities are not transformed when XML is converted to PO, however. Consequently, translators never see **`Liberty Installation and Formatting Tome`**. Instead they see **`&LIFT;`**, which they cannot translate.

Consider something as simple as the following English sentence fragment being translated into a related language: German.

> As noted in the *Liberty Installation and Formatting Tome*, Chapter 3…

A translation of this might be as follows:

> Wie in dem *Wälzer für die Installation und Formatierung von Liberty*, Kapitel 3, erwähnt…

Because there is no text missing, the title can be translated into elegant German. Also, note the use of ‘dem’, the correct form of the definite article ('the') when referring to a Wälzer ('tome') in this grammatical context.

By contrast, if entities are used, the entry in the PO file says:

```
#. Tag: para
#, no-c-format
msgid "As noted in the <citetitle>&LIFT;</citetitle>, Chapter 3…"
msgstr ""
```

The translation of this would probably run thus:

```
#. Tag: para
#, no-c-format
msgid "As noted in the <citetitle>&LIFT;</citetitle>, Chapter 3…"
msgstr "Wie in <citetitle>&LIFT;</citetitle>, Kapitel 3, erwähnt…"
```

And the presentation would be thus:

> Wie in *Liberty Installation and Formatting Tome*, Kapitel 3, erwähnt…

This, of course, leaves the title in English, including words like 'Tome' and 'Formatting' that readers are unlikely to understand. Also, the translator is forced to omit the definite article ‘dem’, a more general construction that comes close to a hybrid of English and German that German speakers call Denglisch or Angleutsch. Many German speakers consider this approach incorrect and almost all consider it inelegant.

Equivalent problems emerge with a fragment such as this:

> However, a careful reading of the *Liberty Installation and Formatting Tome* afterword shows that…

With no text hidden behind an entity, a German translation of this might be:

> Jedoch ergibt ein sorgfältiges Lesen des Nachworts für den *Wälzer für die Installation und Formatierung von Liberty*, dass…

If an entity was used to save the writer time, the translator has to deal with this:

```
#. Tag: para
#, no-c-format
msgid "However, a careful reading of the <citetitle>&LIFT;</citetitle> afterword shows that…"
msgstr ""
```

And the translation would be subtly but importantly different:

```
#. Tag: para
#, no-c-format
msgid "However, a careful reading of the <citetitle>&LIFT;</citetitle> afterword shows that…"
msgstr "Jedoch ergibt ein sorgfältiges Lesen des Nachworts für <citetitle>&LIFT;</citetitle>,
 dass…"
```

When presented to a reader, this would appear as follows:

> Jedoch ergibt ein sorgfältiges Lesen des Nachworts für *Liberty Installation and Formatting Tome*, dass…

Again, note the missing definite article (den in this grammatical context). This is inelegant but necessary since the translator can otherwise only guess which form of the definite article (den, die or das) to use, which would inevitably lead to error.

Finally, consider that although a particular word never changes its form in English, this is not necessarily true of other languages, even when the word is a *proper noun* such as the name of a product. In many languages, nouns change (*inflect*) their form according to their role in a sentence (their grammatical *case*). An XML entity set to represent an English noun or noun phrase therefore makes correct translation impossible in such languages.

For example, if you write a document that could apply to more than one product, you might be tempted to set an entity such as `&PRODUCT;`. The advantage of this approach is that by simply changing this value in the ***`Doc_Name`*`.ent`** file, you could easily adjust the book to document (for example) Red Hat

Enterprise Linux, Fedora, or CentOS. However, while the proper noun *Fedora* never varies in English, it has six different forms in Czech, depending on one of seven ways that you can use it in a sentence:

Case	Usage	Form
Nominative	the subject of a sentence	Fedora
Genitive	indicates possession	Fedory
Accusative	the direct object of a sentence	Fedoru
Dative	the indirect object of a sentence	Fedoře
Vocative	the subject of direct address	Fedoro
Locative	relates to a location	Fedoře
Instrumental	relates to a method	Fedorou

Table 3.1. 'Fedora' in Czech

For example:

- Fedora je linuxová distribuce. — Fedora is a Linux distribution.
- Inštalácia Fedory — Installation of Fedora
- Stáhnout Fedoru — Get Fedora
- Přispějte Fedoře — Contribute to Fedora
- Ahoj, Fedoro! — Hello Fedora!
- Ve Fedoře 10… — In Fedora 10…
- S Fedorou získáváte nejnovější… — With Fedora, you get the latest…

A sentence that begins S Fedora získáváte nejnovější… remains comprehensible to Czech readers, but the result is not grammatically correct. The same effect can be simulated in English, because although English nouns lost their case endings during the Middle Ages, English pronouns are still inflected. The sentence, 'Me see she' is completely comprehensible to English speakers, but is not what they expect to read, because the form of the pronouns **me** and **she** is not correct. **Me** is the accusative form of the pronoun, but because it is the subject of the sentence, the pronoun should take the nominative form, **I**. Similarly, **she** is nominative case, but as the direct object of the sentence the pronoun should take its accusative form, **her**.

Nouns in most Slavic languages like Russian, Ukrainian, Czech, Polish, Serbian, and Croatian have seven different cases. Nouns in Finno–Ugaric languages such as Finnish, Hungarian, and Estonian have between fifteen and seventeen cases. Other languages alter nouns for other reasons. For example, Scandinavian languages inflect nouns to indicate *definiteness* — whether the noun refers to '*a* thing' or '*the* thing' — and some dialects of those languages inflect nouns both for definiteness *and* for grammatical case.

Now multiply such problems by the more than 40 languages that **Publican** currently supports. Other than the few non-translated strings that **Publican** specifies by default in the ***Doc_Name*.ent** file, entities might prove useful for version numbers of products. Beyond that, the use of entities is tantamount to a conscious effort to inhibit and reduce the quality of translations. Furthermore, readers of your document in a language that inflects nouns (whether for case, definiteness, or other reasons)

will not know that the bad grammar is the result of XML entities that you set — they will probably assume that the translator is incompetent.

3.1.7. Revision_History.xml

The **publican package** command searches for the first XML file in the document's XML directory containing a **<revhistory>** tag. **Publican** then uses that file to build the RPM revision history.

3.2. Adding images

Store images in the **images** subdirectory in the directory that holds your XML files. Use **./images/*image-name*** to insert images into a book. The following is an example that inserts the **testimage.png** image:

```
<mediaobject>
<imageobject>
 <imagedata fileref="./images/testimage.png" />
</imageobject>
<textobject><phrase>alternate text goes here</phrase></textobject>
</mediaobject>
```

Ensure that you supply a <textobject> so that your content remains accessible to people with visual impairments. In certain jurisdictions, you might have a legal responsibility to provide this accessibility — for example, if you or your organization must comply with Section 508 of the United States *Rehabilitation Act of 1973*.[1]

If your book contains images that need to be localized — for example, screenshots of a user interface in a language other than the original language of your book — place these images in the **images** subdirectories for each language directory. Make sure that the image file in the translated language has the same name as the image file in the original language. When you build the book in the translated language, **Publican** uses the file from the **images** subdirectory of the translated language instead of the file from the **images** subdirectory of the original language.

Images wider than 444 pixels present poorly in HTML because they often go beyond the right margin of the text. Similarly, images wider than 444 pixels often go beyond the right margin of PDF pages and are cropped so that only the left side of the image is visible. Therefore, by default, **Publican** creates HTML and PDF output that instructs web browsers and PDF viewers to scale down any images larger than 444 pixels wide. Note, however, that images lose quality significantly when scaled in this way. For best results, scale or crop your images in image editing software so that they are no more than 444 pixels wide before you place them in a document.

Image file locations

Publican only uses images in the **images** subdirectory of your XML directory and corresponding images in the **images** subdirectories of your translated languages. Images stored in other directories directories do not work.

[1] Refer to *http://www.section508.gov/*

PNG files in PDF documents

Publican depends on an external application, **FOP**, to render documents as PDF files. At present, some versions of **FOP** contain a bug that alters the colors in certain images in PNG format. Specifically, 32-bit PNG images are rendered correctly, while 24-bit PNG images are not.

If you notice that **Publican** produces a PDF file that contains images with incorrect colors, convert the original PNG files to 32-bit PNG format by adding an *alpha channel* to the image and rebuild the book. If your chosen image manipulation software does not include an option specifically labeled **`Add alpha channel`**, the option might be labeled **`Add transparency`** instead.

3.3. Preparing a document for translation

Support for localization of documents was a key consideration in the design of **Publican**. The general translation workflow for documents developed in **Publican** is as follows:

1. Complete the XML of a document.

   The XML for this version of the document should now be considered ‘frozen’. If your document is stored in a version-controlled repository, you should now move this version into a separate directory or branch. This allows writers to begin work on subsequent versions of the document in one branch, while providing a stable base for translation in another branch.

2. Generate *portable object template* (POT) files from the XML files:

   ```
   $ publican update_pot
   ```

   If this is the first time that POT files have been created for this document, **Publican** creates a new subdirectory, named **`pot`**. The **`pot`** subdirectory holds a POT file for each XML file in the document. If **Publican** has created POT files for this document previously, **Publican** updates the existing POT files to reflect any changes in the XML since the POT files were last updated.

   ### Remove unused XML files

   Publican generates a POT file for every XML file in the XML directory, whether the XML file is used in the document or not. If you transform unused XML files into POT files, you waste the time and effort of volunteer translators, and waste money if you are paying for translations.

   Use the **`publican print_unused`** command to generate a list of XML files that are not used in your document.

3. Generate *portable object* (PO) files from the POT files to begin translation into a particular language:

   ```
   $ publican update_po --langs=language_code
   ```

where *language_code* is the code for the target language. Refer to *Appendix D, Language codes* for more information about language codes. You can provide multiple language codes, separated by commas, to generate PO files for more than one language at a time. For example:

```
$ publican update_po --langs=hi-IN,pt-BR,ru-RU,zh-CN
```

If this is the first time that PO files have been created for a particular language, **Publican** creates a new subdirectory, named with the language code that you specified with the **--langs=** option. This subdirectory holds a PO file for each POT file in **pot** subdirectory. If **Publican** has created PO files for this language previously, **Publican** updates the existing PO files to reflect any changes in the POT files since the PO files were last updated. You can update existing PO files in every subdirectory with the **--langs=all** option:

```
$ publican update_po --langs=all
```

Remove unused POT files

Publican generates a PO file for every POT file in the **pot** directory, whether the POT file is based on a corresponding XML file that is used in the document or not, or whether a corresponding XML file even exists. If you transform POT files for unused or deleted XML files into PO files, you waste the time and effort of volunteer translators, and waste money if you are paying for translations.

When you generate PO files, **Publican** presents you with a warning for any POT files that do not have corresponding XML files, but will generate the PO file nevertheless. However, **Publican** will not warn you if a POT file exists for an XML file that is not used in the document.

4. Translators translate the *strings* contained in the PO files.

5. Build the document in the target language, for example:

```
$ publican build --formats=html,html-single,pdf --langs=is-IS,nb-NO
```

or package it in the target language, for example:

```
$ publican package --lang=is-IS
```

You can build the document in all languages for which you have translations with the **--langs=all** option, but note that you must package each language individually. Refer to *Section 3.4, “Building a document”* for more information on building a document, and *Section 3.5, “Packaging a book”* on packaging a document.

3.4. Building a document

Note — Customizing output

The parameters set in the document configuration file (by default, **publican.cfg**) allow you to control many aspects of the way in which a document is presented — refer to *Section 3.1.1, "The publican.cfg file"*.

If you maintain multiple versions of a document, you can create a configuration file for each version. When building the document, you can use the **--config** to specify which configuration file (and therefore which set of parameters) to use in a particular build, for example:

```
publican build --formats html,pdf --langs en-US,de-DE,hu-HU --config community.cfg
```

To build a document:

1. Confirm the *YEAR* and *HOLDER* entities have been configured in the ***Doc_Name*.ent** file, as described in *Section 3.1.6, "Doc_Name.ent"*.

2. Change into the root directory of the document. For example, if the document was named **Test_Book** and was located in the **books/** directory, run the following command:

   ```
   cd books/Test_Book
   ```

3. Run a test for any errors that would stop the book from building in your chosen language, for example:

   ```
   publican build --formats=test --langs=en-US
   ```

4. Run the following command to build the book:

   ```
   publican build --formats=formats --langs=languages
   ```

   Replace *formats* with a comma-separated list of the formats that you want to build, for example, **--formats=html,html-single,pdf**. Replace *langs* with a comma-separated list of the languages that you want to build, for example, **--langs=en-US,sv-SE,uk-UA,ko-KR**.

Formats for the **build** action

html

Publican outputs the document as in multiple HTML pages, with each chapter and major section on a separate page. **Publican** places an index at the start of the document, and places navigational elements on each page.

Use the *chunk_first* and *chunk_section depth* parameters in the **publican.cfg** file to control how how **Publican** chunks sections in this format.

html-single

Publican outputs the document as a single HTML page with the table of contents near the top of the page.

`html-desktop`
: **Publican** outputs the document as a single HTML page with the table of contents located in a separate pane on the left side of the document.

`pdf`
: **Publican** outputs the document as a PDF file.

`test`
: **Publican** validates the XML structure of the book, but does not transform the XML into another format.

`txt`
: **Publican** outputs the document as a single text file.

`epub`
: **Publican** outputs the document as an e-book in EPUB format.

`eclipse`
: **Publican** outputs the document as an **Eclipse** help plugin. Refer to *Section 3.1.1, "The publican.cfg file"* for details of setting the *`id`*, *`name`*, and *`provider-name`* parameters.

The following examples demonstrate commonly used **`publican build`** commands:

`publican build --help`
: List available **`publican build`** options for building a book.

`publican build --formats=test --langs=`*`languages`*
: Check that the book can be built correctly. Build **`--formats=test`** before running any other **`publican build`** command, and before checking a book back into a version-controlled repository from which other contributors might download it.

`publican build --formats=html --langs=`*`languages`*
: Build the book in multi-page HTML format. The HTML output will be located in the ***`Doc_Name`***`/tmp/`***`language`***`/html/` directory. Each chapter and major section is placed in a separate HTML file. You can control the depth at which **Publican** places subsections into separate HTML files with the **`chunk-section-depth`** parameter in the **`publican.cfg`** — refer to *Section 3.1.1, "The publican.cfg file"*.

`publican build --formats=html-single --langs=`*`languages`*
: Build the book in single-page HTML format. The output will be a single HTML file located in the ***`Doc_Name`***`/tmp/`***`language`***`/html-single/` directory.

`publican build --formats=pdf --langs=`*`languages`*
: Build the book as a PDF file. **Publican** relies on an external application, **FOP** to render PDF. Therefore, building PDF might not be available on all systems, depending on the availability of **FOP**. The output will be a single PDF file located in the ***`Doc_Name`***`/tmp/`***`language`***`/pdf/` directory.

`publican build --formats=html,html-single,pdf --langs=`*`languages`*
: Build the book in multi-page HTML, single-page HTML, and PDF formats.

3.4.1. Building a document created with Publican 0

Documents produced with early versions of **Publican** (versions up to and including 0.45) did not have a **publican.cfg** file; a similar set of parameters was defined in a **Makefile**. Before you build such a document in a current version of **Publican** (version 0.99 onwards), you must convert the **Makefile** into a **publican.cfg** file. **Publican** can do this conversion automatically:

1. Change into the document directory, the one that holds the **Makefile**.
2. Run **publican old2new**. **Publican** parses the **Makefile** and creates a **publican.cfg** file with equivalent parameters wherever available.

When you run **publican old2new**, **Publican** does not alter or delete the original **Makefile**. A **Makefile** and a **publican.cfg** file can coexist in the same document.

3.5. Packaging a book

Packages other than RPM packages

This section discusses packaging documents for distribution through the **RPM Package Manager**. However, when you use the **publican package** command, **Publican** generates a tarball that you can use to build a package to distribute through different package manager software. If you run **publican package** on a computer on which **rpmbuild** is not installed, **Publican** still generates the tarball, even though it cannot then generate an RPM package from that tarball.

Note — Customizing output

The parameters set in the document configuration file (by default, **publican.cfg**) allow you to control many aspects of the way in which a document is presented and packaged — refer to *Section 3.1.1, "The publican.cfg file"*.

If you maintain multiple versions of a document, you can create a configuration file for each version. When packaging the document, you can use the **--config** to specify which configuration file (and therefore which set of parameters) to use in a particular build, for example:

```
publican package --lang hi-IN --config community.cfg
```

Publican not only builds documentation as HTML and PDF files, but it can package these files for distribution to individual workstations and to web servers as *RPM packages*. RPM packages are used to distribute software to computers with Linux operating systems that use the **RPM Package Manager**. These operating systems include Red Hat Enterprise Linux, Fedora, Mandriva Linux, SUSE Linux Enterprise, openSUSE, Turbolinux, and Yellow Dog Linux, to name just a few.

3.5.1. Types of RPM packages

Publican can produce both *source RPM packages* (*SRPM packages*) and *binary RPM packages*. Furthermore, both SRPM packages and binary RPM packages can be configured to deploy to workstations or web servers.

3.5.1.1. Source RPM packages and binary RPM packages

An SRPM package contains the source code used to generate software rather than the software itself. To use an SRPM package, a computer must *compile* the source code into software — or in this case, into documents. SRPM packages of **Publican** documents contain XML files rather than finished documents. To install documentation from the SRPM package to a computer, the computer must have **Publican** installed on it. When you try to install the SRPM package on a computer that does not have **Publican** installed, the **RPM Package Manager** looks for **Publican** in the software repositories that are available to it. The **RPM Package Manager** installs **Publican** first, so that it can build and install the document contained in the SRPM package. If the **RPM Package Manager** cannot find and install **Publican**, installation of the SRPM package will fail.

Conversely, binary RPM packages contain software — or in this case, a document — that is ready to copy to a location in the computer's file system and use immediately. The contents of the binary RPM package do not need to be compiled by the computer onto which they are installed, and therefore, the computer does not need to have **Publican** installed.

3.5.1.2. Desktop packages and web packages

Publican can package documents for reading on a computer workstation (a *desktop RPM package*) or to install on a web server and publish on the world-wide web (a *web RPM package*). The desktop RPM package of a **Publican** document and the web RPM package of the same document differ in that the desktop RPM package installs documentation only for local use on a computer, while the web RPM installs documentation for local use, but also to be served to the World Wide Web.

Desktop RPM packages of **Publican** documents contain the documentation in single-page HTML format. Documents distributed in these packages are installed in a subdirectory of **`/usr/share/doc/`**, the location specified by the *Filesystem Hierarchy Standard* (*FHS*) for 'Miscellaneous documentation'.[2] The desktop RPM package also contains a *desktop file*, to be placed in **`/usr/share/applications/`**. This file enables *desktop environments* such as GNOME and KDE to add the installed document to their menus for ease of reference by users.

Web RPM packages of **Publican** documents contain the documentation in single-page HTML, multi-page HTML, and PDF formats. They are installed in a subdirectory of **`/var/www/html/`**, a common *document root* for web servers. Note that the web SRPM package generates both a web binary RPM package and desktop binary RPM package.

3.5.2. The `publican package` command

Use the **`publican package --lang=`*`Language_Code`*** command to package documents for distribution in the language that you specify with the **`--lang`** option. Refer to *Appendix D, Language codes* for more information about language codes.

If you run **`publican package`** with no options other than the mandatory **`--lang`** option, **Publican** produces a web SRPM package. The full range of options for **`publican package`** is as follows:

`--lang` *`language`*
: specifies the language in which to package the documentation.

Refer to *http://www.pathname.com/fhs/pub/fhs-2.3.html#USRSHAREARCHITECTUREINDEPENDENTDATA*

Incomplete translations

If translation in a particular language is not complete by the scheduled release date, consider marking the language with the *ignored_translations* parameter in the document's **publican.cfg** file. The package will be named appropriately for the language, but will contain documentation in the original language of the XML rather than a partial translation. When translation is complete, remove the *ignored_translations* parameter, increase the release number in the **Project-Id-Version** field in the **Book_Info.po** file for that language, and generate the package again. When you distribute the revised package, it becomes available to replace the original untranslated package.

--config *filename*
specifies that **Publican** should use a configuration file other than the default **publican.cfg** file.

--desktop
specifies that **Publican** should create a desktop RPM package rather than a web RPM package.

--brew
specifies that **Publican** should push the completed package to **Brew**. **Brew** is the build system used internally by Red Hat; this option is meaningless outside Red Hat.

--scratch
when used together with the **--brew** and **--desktop** options, specifies that a SRPM package should be built as a *scratch build* when sent to **Brew**. Scratch builds are used to verify that an SRPM package is structured correctly, without updating the package database to use the resulting package.

--short_sighted
specifies that **Publican** should build the package without including the version number of the software (*version* in the **publican.cfg** file) in the package name.

Omitting the software version number

Much software documentation is version-specific. At best, the procedures described in the documentation for one version of a product might not help you to install, configure, or use a different version of the product. At worst, the procedures described in the documentation for one version might be misleading or even harmful when applied to a different version.

If you distribute documentation as RPM packages without version numbers in the package names, the **RPM Package Manager** on users' computers will automatically replace any existing documentation with the documentation for the latest version; users will not have access to documentation for more than one version of the software at a time. Be certain you want this outcome.

--binary
specifies that **Publican** should build the package as a binary RPM package.

--cvs
specifies that **Publican** should import the generated SRPM package into CVS. This requires *cvs_root*, *cvs_pkg*, and *cvs_branch* be set in the configuration file.

After you run the **publican package** command, **Publican** outputs completed SRPM packages to the document's **tmp/rpm** directory, and completed binary RPM packages to the document's **tmp/rpm/noarch** directory.

By default, **Publican** documentation packages are named ***productname-title-productnumber-[web]-language-edition-pubsnumber.[build_target].noarch.file_extension***. **Publican** uses the information in the document's **publican.cfg** file to supply the various parameters in the file name, and then information in the **Book_Info.xml** file for any parameters missing from the **publican.cfg** file. Refer to *Section 3.1, "Files in the book directory"* for details of configuring these files. Additionally:

- web RPM packages include the element **-web-** between the product version and the language code.
- SRPM packages have the file extension **.src.rpm** but binary RPM packages have the file extension **.rpm**
- binary RPM packages include ***[build_target]*.noarch** before the file extension, where *[build_target]* represents the operating system and version that the package is built for as set by the *os_ver* parameter in the **publican.cfg** file. The **noarch** element specifies that the package can be installed on any system, regardless of the system architecture.
- use of the **--short_sighted** option removes the ***-productnumber-*** from the package name.

3.5.2.1. The publican package command — Example usage

The following examples illustrate some common options, illustrated with the *Foomaster 9 Configuration Guide*, edition 2, revision 6.

publican package --lang=cs-CZ
produces a web SRPM package named *Foomaster-Configuration_Guide-9-web-cs-CZ-2-6.src.rpm* that contains documentation in Czech.

publican package --desktop --lang=cs-CZ
produces a desktop SRPM package named *Foomaster-Configuration_Guide-9-cs-CZ-2-6.src.rpm* that contains documentation in Czech.

publican package --binary --lang=cs-CZ
produces both a web binary RPM package named *Foomaster-Configuration_Guide-9-web-cs-CZ-2-6.el5.noarch.rpm* and a desktop binary RPM package named *Foomaster-Configuration_Guide-9-cs-CZ-2-6.el5.noarch.rpm* that contain documentation in Czech, built for the Red Hat Enterprise Linux 5 operating system.

publican package --desktop --binary --lang=cs-CZ
produces a desktop binary RPM package named *Foomaster-Configuration_Guide-9-cs-CZ-2-6.el5.noarch.rpm* that contains documentation in Czech, built for the Red Hat Enterprise Linux 5 operating system.

`publican package --desktop --short_sighted --lang=cs-CZ`
produces a desktop SRPM package named *Foomaster-Configuration_Guide-cs-CZ-2-6.src.rpm* that contains documentation in Czech. This package will replace any Configuration Guides for previous versions of **Foomaster** that exists on a system. Users cannot have access to both the *Foomaster 8 Configuration Guide* and the *Foomaster 9 Configuration Guide*.

3.6. Conditional tagging

In some cases you may need to maintain multiple versions of a book; for example, a HOWTO guide for product FOO can have an upstream version and an enterprise version, with very subtle differences between them.

Publican makes it easy to manage differences between multiple versions of a book by allowing you to use a single source for all versions. *Conditional tagging* allows you to make sure that version-specific content only appears in the correct version; that is, you *conditionalize* the content.

To conditionalize content in a book, use the tag attribute **`condition`**. For example, let's say the book *How To Use Product Foo* has an "upstream", "enterprise", and "beta" version:

```
<para condition="upstream">
This paragraph will only appear in the upstream version of How To Use Product Foo.
</para>

<para condition="enterprise">
This paragraph will only appear in the enterprise version of How To Use Product Foo.
</para>

<para condition="beta">
This paragraph will only appear in the beta version of How To Use Product Foo.
</para>

<para condition="beta,enterprise">
This paragraph will only appear in the beta and enterprise versions of How To Use Product Foo.
</para>
```

To build a specific version (and thereby capture all content conditionalized for that version), add the **`condition: version`** parameter to the **`publican.cfg`** file and run the **`publican build`** command as normal. For example, if you add **`condition: upstream`** to the **`publican.cfg`** file of *How To Use Product Foo* and run:

```
publican build --formats=pdf --langs=en-US
```

Publican will capture all tags that use **`condition="upstream"`** attribute and build *How To Use Product Foo* in as a PDF file in American English.

Root nodes and conditional tagging

If the root node of an XML file is excluded with a conditional, your document will not build, because empty files are not valid XML. For example, if **`Installation_and_configuration_on_Fedora.xml`** contains a single chapter:

```
<?xml version='1.0' encoding='utf-8' ?>
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-
open.org/docbook/xml/4.5/docbookx.dtd" [
]>
<chapter id="chap-Installation_and_configuration_on_Fedora" condition="Fedora">
<title>Installation and configuration on Fedora</title>

[text of chapter]

</chapter>
```

and this chapter is included in **User_Guide.xml** with an <xi:include> tag, the document will not build with **condition: Ubuntu** set in the **publican.cfg** file.

To exclude this chapter, add a condition to the <xi:include> tag in **User_Guide.xml**, not to the <chapter> tag in **Installation_and_configuration_on_Fedora.xml**.

xrefs and conditional tagging

If an <xref> points to content not included in the build due to conditional tagging, the build will fail. For example, with **condition: upstream** set in the **publican.cfg** file, **publican build --formats=pdf --langs=en-US** will fail if the book has the tag <xref linkend="betasection"> pointing to <section id="betasection" condition="beta">.

3.6.1. Conditional tagging and translation

Use conditional tagging with great caution

Use conditional tagging only with great caution in books that you expect to be translated, as conditional tagging creates extra difficulties for translators.

Conditional tagging creates difficulty for translators in two ways: it obscures context in the *portable object* (PO) files through which translators work, and it makes proofreading more difficult for translators who are not deeply familiar with your book and all the conditions that you have set.

PO files do not include attributes from tags. When translators open the PO file for the example from *How To Use Product Foo* in *Section 3.6, "Conditional tagging"*, they see:

```
#. Tag: para
#, no-c-format
msgid "Content that only appears in the upstream version of How To Use Product Foo."
msgstr ""

#. Tag: para
#, no-c-format
msgid "Content that only appears in the enterprise version of How To Use Product Foo."
msgstr ""

#. Tag: para
#, no-c-format
```

```
msgid "Content that only appears in the beta version of How To Use Product Foo."
msgstr ""

#. Tag: para
#, no-c-format
msgid "Content that only appears in the beta and enterprise versions of How To Use Product
 Foo."
msgstr ""
```

In this example, the only paragraphs where the meaning flows logically from one to the next is between paragraphs three and four. Because both of these paragraphs appear in the book for the beta version of the product, they (hopefully) make sense together. Beyond that, the use of conditionals here requires translators to translate individual small chunks of content without the ability to follow the context from one paragraph to the next. When translators must work under these conditions, the quality of the translation will suffer, or the time required — and therefore the cost of translation — will increase.

Furthermore, unless the translators who work on your book know how to configure **Publican**'s **`publican.cfg`** file and are aware of the valid conditions for your book, they cannot proofread their work. Without that knowledge, when translators proofread a document, they will wonder why they cannot find text that they know they translated and can find easily in the PO file. If you must use conditionals in your book, you must be prepared to provide a greater degree of support to your translators than you would otherwise provide.

As an alternative to conditionals, consider maintaining separate versions of your book in separate branches of a version-controlled repository. You can still share XML files and even PO files between the various branches as necessary, and some version control systems allow you to share changes readily among branches.

3.7. Pre-release software and draft documentation

Completed documentation for pre-release software is not the same thing as draft documentation.

Drafts are unfinished versions of a book or article, and their unfinished state is unrelated to the status of the software they document.

In both circumstances, however, it is important to make the status of the software, documentation or both clear to users, developers, readers and reviewers.

3.7.1. Denoting pre-release software

Documentation for pre-release software, especially pre-release software being distributed to testers, customers and partners, should carry a clear mark denoting the beta-status of the software.

To create that mark do the following:

1. Add the software's pre-release version number, or equivalent state information, to the subtitle in your **`Book_Info.xml`** file. Place this additional text in `<remark>` tags. For example:

   ```
   <subtitle>Using Red Hat Enterprise Warp Drive<remark> Version 1.1, Beta 2</remark></
   subtitle>
   ```

2. add `show_remarks` to the **`publican.cfg`** file and set it to '1':

```
show_remarks: 1
```

When you build your book with this <remark> tag and the show_remarks setting in place, the pre-release nature of the software is displayed clearly and unmistakably. PDF builds display the remark on their cover and title pages. HTML builds (both single-page HTML and multiple-page HTML) display the remark near the beginning of **index.html**.

Because this approach makes no changes to the information in **Book_Info.xml** used to generate RPMs, it also ensures there is no ambiguity in the RPM subsystem's operation.

Important

It is the writer's responsibility to remove the <remark> tag and its contents and remove or turn off show_remarks when documentation is updated for use with the release version of the software.

3.7.2. Denoting draft documentation

Unfinished documentation made available to others for review should be labeled clearly as such.

- To add the draft watermark to your documentation add the **status="draft"** attribute to the <article>, <book> or <set> tag in your document's root node. For example:

```
<book status="draft">
```

By default, your root node is the <book> tag in your ***Doc_Name*.xml** file.

If you are working on an article or set, the root node is the <article> or <set> tag in ***Doc_Name*.xml**.

Adding the **status="draft"** attribute causes each page of the document to show the draft watermark. This is by design.

Even if you change only a portion of a work before sending it out for review, marking every page as draft will encourage reviewers to report errors or typos they spot in passing. It will also ensure non-reviewers who encounter the work do not mistake a draft for a finished version.

3.7.3. Denoting draft documentation of pre-release software

To denote unfinished documentation of pre-release software properly, do both previously noted procedures.

Branding

Brands are collections of files that **Publican** uses to apply a consistent look and style to HTML and PDF output. They provide boilerplate text that appears at the beginning of documents, images such as logos, and stylistic elements such as color schemes. **Publican** ships with one brand, **`common/`**. Documentation projects can produce and distribute brands to their contributors, either as a package (for example, an RPM package) or as an archive (for example, a tarball or ZIP file).

4.1. Installing a brand

Publican brands for Fedora, Genome, and oVirt documents are available as RPM packages in Fedora. Similarly, Red Hat internally distributes RPM packages containing **Publican** brands for GIMP, JBoss, and Red Hat documents. Providing that you have access to the relevant repositories, you can install these brands on a computer that runs Red Hat Enterprise Linux or Fedora — or an operating system derived from either — with the command **`yum install publican-`*`brand`*** or with a graphical package manager such as **PackageKit**.

If you use **Publican** on an operating system that does not use RPM packages, your documentation project might provide its brand in another format. Whatever the format in which the brand is supplied, you must place the brand files in a subdirectory of the **Publican** Common Content directory. By default, this directory is located at **`/usr/share/publican/Common_Content`** on Linux operating systems and at **`%SystemDrive%/%ProgramFiles%/Publican/Common_Content`** on Windows operating systems — typically, **`C:/Program Files/Publican/Common_Content`**

Each currently available brand is distributed under a brand-specific license as follows:

To install the brand:

1. If the brand was supplied to you in an archive of some kind, for example, a tarball or ZIP file, unpack the brand into a new directory on your system.
2. Change into the directory in which you created or unpacked the brand:

   ```
   cd publican-brand
   ```

   where *`brand`* is the name of the brand.
3. Build the brand:

   ```
   publican build --formats xml --langs all --publish
   ```
4. Install the brand:

   ```
   sudo publican installbrand --path path
   ```

   where *`path`* is the path to the **Publican** Common Content files. For example, on a Linux system, run:

   ```
   sudo publican installbrand --path /usr/share/publican/Common_Content
   ```

   or on a Windows system, run

```
sudo publican installbrand --path "C:/Program Files/Publican/Common_Content"
```

5.

6.

Brand	License	Package	Comment
common	*GFDL Version 1.2*[1]	publican	GPL compatible license. No options.
RedHat	*CC-BY-SA 3.0*[2]	publican-redhat	
Fedora	*CC-BY-SA 3.0*[3]	publican-fedora	
JBoss	*CC-BY-SA 3.0*[4]	publican-jboss	No Options.
oVirt	*OPL 1.0*[5]	publican-ovirt	No Options.
GIMP	*GFDL Version 1.2*[6]	publican-gimp	GPL compatible license. No options.
Genome	*OPL 1.0*[7]	publican-genome	No Options.

Table 4.1. Current Brands and their packages

4.2. Creating a brand

Use the **create_brand** action to create a new brand. When you create a new brand, you must give it a name and specify the original language for the brand's XML files. The **--name** option provides the name, and the **--lang** option specifies the language. The complete command is therefore:

```
publican create_brand --name=brand --lang=language_code
```

Publican creates a new subdirectory named **publican-*brand***, where *brand* is the brand that you specified with the **--name** option.

For example, to create a brand called **Acme**, which will have its Common Content XML files written originally in American English, run:

```
publican create_brand --name=Acme --lang=en-US
```

Publican creates the brand in a subdirectory named **publican-Acme**.

To configure your new brand, search for the word **SETUP** in the default files that **Publican** creates and edit the files to provide the missing details. On Linux operating systems, you can search for the word **SETUP** in these files with the command:

```
grep -r 'SETUP' *
```

4.3. Files in the brand directory

Running the **publican create_brand --name=*brand* --lang=*language_code*** command creates a directory structure and the required files. The brand directory initially contains:

- **COPYING**
- **defaults.cfg**
- **overrides.cfg**
- **publican.cfg**
- **publican-*brand*.spec**, where *brand* is the name of the brand.
- **README**
- a subdirectory for the brand's XML files, CSS stylesheets, and default images. The subdirectory is named with the language code of the original language of the brand (for example, **en-US**). These files are:
  - **Feedback.xml**
  - **Legal_Notice.xml**
  - the **css** subdirectory, which contains:
    - **overrides.css**
  - the **images** subdirectory, which contains 43 images in both raster (PNG) and vector (SVG) formats.

4.3.1. The publican.cfg file

The **publican.cfg** file in a brand serves a similar purpose to the **publican.cfg** file in a document — it configures a number of basic options that define your brand.

version
: specifies the version number for the brand. When you create the brand with **publican create_brand**, the version number is set to **1.0**. Update the version number here in the brand **publican.cfg** file and in the brand **publican.spec** file when you prepare a new version of the brand.

  Note that this parameter is unrelated to the version number of documents built with this brand. For example, the *Fedora 12 Installation Guide* has its version set as **12** in its **publican.cfg** file, but might be built with version 1.0 of the *publican-fedora* brand.

xml_lang
: specifies the language of the source XML files for the brand's Common Content, for example, **en-US**, as set by the **--lang** option for **publican create_brand**.

release
: specifies the release number for the brand. When you create the brand with **publican create_brand**, the release number is set to **0**. Update the version number here in the brand **publican.cfg** file and in the brand **publican.spec** file when you prepare a new release of an existing version of the brand.

type
: when set to **type=brand**, this parameter identifies the contents of this directory as a brand, rather than a book, article, or set.

brand
specifies the name of the brand, as set by the **--name** option for **publican create_brand**.

4.3.2. The defaults.cfg file and overrides.cfg file

Every document built in **Publican** has a **publican.cfg** file in its root directory, which configures build options for the document. Refer to *Section 3.1.1, "The publican.cfg file"* for a full description of these options. The **defaults.cfg** file and **overrides.cfg** file in a brand supply default values for any of the parameters that you can otherwise set with a document's **publican.cfg** file.

When you build a document with a particular brand, **Publican** first applies the values in the brand's **defaults.cfg** file before it applies the values in the document's **publican.cfg** file. Values in the document's **publican.cfg** file therefore override those in the brand's **defaults.cfg** file.

Publican next applies the values in the brand's **overrides.cfg** file, which therefore override any values in the brand's **defaults.cfg** file and the document's **publican.cfg** file.

Use the **defaults.cfg** file to set values that you routinely apply to your brand but want to allow writers to change in particular books; use the **overrides.cfg** file for values that you do not want to allow writers to change.

4.3.3. publican-*brand*.spec file

Some Linux operating systems use the **RPM Package Manager** to distribute software, in the form of *RPM packages*. In general terms, an RPM package contains software files compressed into an archive, accompanied by a *spec file* that tells the **RPM Package Manager** how and where to install those files.

When you create a brand, **Publican** generates the outline of an RPM spec file for the brand. The automatically generated spec file provides you with a starting point from which to create an RPM package to distribute your brand. Refer to *Section 4.4, "Packaging a brand"* to learn how to configure the spec file and use it to produce an RPM package.

4.3.4. README

The **README** file in an SRPM package includes a brief description of the package.

4.3.5. COPYING

The **COPYING** file in an SRPM package contains details of the copyright license for the package.

4.3.6. Common Content for the brand

Inside the brand directory is a subdirectory named after the default XML language for brand, as set with the **--lang** option when you created the brand. This subdirectory contains XML and image files that override the default Common Content provided with **Publican**. Customizing these files provides your brand with its distinctive appearance, including its color scheme and logos.

4.3.6.1. Feedback.xml

The **Feedback.xml** file is included by default in the preface of every book produced in **Publican**. It invites readers to leave feedback about the document. Customize this file with the contact details of

your project. If your project uses a bug tracking system such as **Bugzilla**, **JIRA**, or **Trac**, you could include this information here.

4.3.6.2. Legal_Notice.xml

The **Legal_Notice.xml** file contains the legal notice that appears at the beginning of every document produced by **Publican**. Insert the details of your chosen copyright license into this file. Typically, this might include the name of the license, a short summary of the license, and a link to the full details of the license.

4.3.7. The css subdirectory

The **css** subdirectory contains a single file: **overrides.css**.

4.3.7.1. overrides.css

The **overrides.css** file sets the visual style for your brand. Values in this file override those in **Publican**'s **Common_Content/common/*xml_lang*/css/common.css** file.

4.3.8. The images subdirectory

The **images** subdirectory contains 43 images in both *portable network graphics* (PNG) and *scalable vector graphics* (SVG) format. These images are placeholders for various navigation icons, admonition graphics, and brand logos. They include:

image_left
: is a logo for the product to which this document applies. It appears at the top left corner of HTML pages, where it contains a hyperlink to a web page for the product, as defined by *prod_url* in the **publican.cfg** file for the document. Consider setting *prod_url* in the brand's **defaults.cfg** or **overrides.cfg** file.

image_right
: is a logo for the team that produced this documentation. It appears at the top right corner of HTML pages, where it contains a hyperlink to a web page for the documentation team, as defined by *doc_url* in the **publican.cfg** file for the document. If all the documentation for this brand is produced by the same team, consider setting *doc_url* in the brand's **defaults.cfg** or **overrides.cfg** file.

title_logo
: is a larger version of your product logo, which appears on the title page of PDF documents and at the start of HTML documents.

note, **important**, **warning**
: are icons that accompany the XML admonitions <note>, <important>, and <warning>.

dot, **dot2**
: are bullets used for <listitem>s in <itemizedlist>s.

stock-go-back, **stock-go-forward**, **stock-go-up**, **stock-home**
: are navigation icons for HTML pages.

`h1-bg`
is a background for the heading that contains the name of your product, as it appears at the very beginning of a HTML document.

`watermark_draft`
is a watermark that appears on pages of draft documentation. Refer to *Section 3.7.2, "Denoting draft documentation"*.

4.4. Packaging a brand

Packages other than RPM packages

This section discusses packaging documents for distribution through the **RPM Package Manager**. However, when you use the **`publican package`** command, **Publican** generates a tarball that you can use to build a package to distribute through different package manager software. If you run **`publican package`** on a computer on which **rpmbuild** is not installed, **Publican** still generates the tarball, even though it cannot then generate an RPM package from that tarball.

After you create a brand (as described in *Section 4.2, "Creating a brand"*), **Publican** can help you to distribute the brand to members of your documentation project as *RPM packages*. RPM packages are used to distribute software to computers with Linux operating systems that use the **RPM Package Manager**. These operating systems include Red Hat Enterprise Linux, Fedora, Mandriva Linux, SUSE Linux Enterprise, openSUSE, Turbolinux, and Yellow Dog Linux, to name just a few.

Publican can produce both *source RPM packages* (*SRPM packages*) and *binary RPM packages*. As part of this process, it also creates the *spec file* — the file that contains the details of how a package is configured and installed.

An SRPM package contains the source code used to generate software rather than the software itself. To use an SRPM package, a computer must *compile* the source code into software. SRPM packages of **Publican** brands contain the configuration files, XML files, and image files that define the brand in its original language, plus the PO files that generate the Common Content files in translated languages. To install a brand from its SRPM package to a computer, the computer must have **Publican** installed on it. When you try to install the SRPM package on a computer that does not have **Publican** installed, the **RPM Package Manager** looks for **Publican** in the software repositories that are available to it. The **RPM Package Manager** installs **Publican** first, so that it can build and install the brand contained in the SRPM package. If the **RPM Package Manager** cannot find and install **Publican**, installation of the SRPM package will fail.

Conversely, binary RPM packages contain software — in this case, a **Publican** brand — that is ready to copy to a location in the computer's file system and use immediately. The contents of the binary RPM package do not need to be compiled by the computer onto which they are installed, and therefore, the computer does not need to have **Publican** installed.

To package a brand, use the **`publican package`** command in the brand directory. When used without any further options, **Publican** produces an SRPM package. The options for packaging a brand are as follows:

`--binary`
specifies that **Publican** should build the package as a binary RPM package.

`--brew`
: specifies that **Publican** should push the completed package to **Brew**. **Brew** is the build system used internally by Red Hat; this option is meaningless outside Red Hat.

`--scratch`
: when used together with the **`--brew`** option, specifies that a SRPM package should be built as a *scratch build* when sent to **Brew**. Scratch builds are used to verify that a SRPM package is structured correctly, without updating the package database to use the resulting package.

The **`--lang`**, **`--desktop`** and **`--short_sighted`** options that apply when you package books (described in *Section 3.5, "Packaging a book"*) are meaningless when you package brands. In particular, note that although the **`--lang`** option is mandatory when you package a book, you do not need to use it when you package a brand.

By default, **Publican** brand packages are named **`publican-`*`brand-version-release`*`.[`*`build_target`*`].[noarch].`*`file_extension`***. **Publican** uses the information in the **`publican.cfg`** file to supply the various parameters in the file name. Refer to *Section 4.3.1, "The publican.cfg file"* for details of configuring this file. Additionally:

- SRPM packages have the file extension **`.src.rpm`** but binary RPM packages have the file extension **`.rpm`**
- binary RPM packages include ***`[build_target]`*`.noarch`** before the file extension, where *`[build_target]`* represents the operating system and version that the package is built for as set by the *`os_ver`* parameter in the **`publican.cfg`** file. The **`noarch`** element specifies that the package can be installed on any system, regardless of the system architecture.

Using sets

A *set* is a collection of books, published as a single output. The *Services Plan* for example is a set comprised of many books such as the *Developer Guide*, *Engineering Content Services Guide* and the *Engineering Operations Guide* to name just a few. The **create_book** command creates a template for a set by setting the *type* parameter to **Set**. There are two types of sets, *stand-alone sets* and *distributed sets*.

5.1. Stand-alone sets

A stand-alone set contains the XML files for each book, all of which are located inside the directory of the set.

The procedure that follows will guide you through the process of creating a stand-alone set named *My Set* located in a directory called **books/My_Set/**. The set will contain *Book A* and *Book B* both of which will be manually created inside the **books/My_Set/en-US** directory.

Procedure 5.1. Creating a stand-alone set

1. Run the following command in a shell in the **books/** directory to create a set named **My_Set** branded in the Red Hat style and in which the XML will be written in American English.

```
publican create --type=Set --name=My_Set --brand=RedHat --lang=en-US
```

2. **cd** into the **My_Set/en-US** directory and create two directories called **Book_A** and **Book_B**.

```
cd My_Set/en-US
mkdir Book_A Book_B
```

3. **cd** into the **books/My_Set/en-US/Book_A** directory. Create and edit the **Book_A.xml**, **Book_Info.xml**, and any other xml files required for your book such as those required for individual chapters. Ensure that **Book_A.xml** contains the correct `xi:include` references to all of your xml files in the directory. For example, if *Book A* contained **Book_Info.xml** and **Chapter_1.xml**, the **Book_A.xml** file would look like this:

```
<?xml version='1.0'?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
"http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd" [
]>

<book>
   <xi:include href="Book_Info.xml" xmlns:xi="http://www.w3.org/2001/XInclude"></
xi:include>
   <xi:include href="Chapter_1.xml" xmlns:xi="http://www.w3.org/2001/XInclude"></
xi:include>
</book>
```

4. Use the same process for *Book_B*, located in the **books/My_Set/en-US/Book_B** directory, as per the step above.

5. Open the **books/My_Set/en-US/My_Set.xml** file in an editor. For each book in the set, add an `xi:include` reference to the primary xml file from the book. The primary xml file for *Book A* will be **Book_A.xml** and for *Book B*, **Book_B.xml**. The **My_Set.xml** file should now look like this:

```
<?xml version="1.0"?>
<!DOCTYPE set PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
"http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd" [
]>

<set>
 <xi:include href="Set_Info.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <xi:include href="Preface.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <xi:include href="Book_A/Book_A.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <xi:include href="Book_B/Book_B.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <xi:include href="Revision_History.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
</set>
```

6. Test your set by running the **publican build --formats=test langs=en-US** command.

5.2. Distributed sets

A *distributed set* contains books that are located in a version-controlled repository. Although several version control systems exist, this version of **Publican** supports only one: **Subversion** (**SVN**). By setting the repository location and titles of the included books in the **publican.cfg** file, each book can be exported to build the entire set. The procedure that follows will guide you through the process of creating a set named *My Set* containing *Book A* and *Book B*.

Important

The following procedure assumes that *Book A* and *Book B* already exist and are available in your **SVN** repository. Currently **Publican** only supports **SVN**.

Procedure 5.2. Creating a set

1. Run the following command in a shell to create a set named **My_Set** branded in the Red Hat style and in which the XML will be written in American English.

```
$ publican create --type=Set --name=My_Set --brand=RedHat  --lang=en-US
```

2. Add the following lines to the **publican.cfg** file:

```
books   = Book_A Book_B
repo = http://PATH-TO-YOUR-SVN-REPOSITORY
scm = SVN
```

3. Open the **My_Set.xml** file in an editor. For each book in the set, add an `xi:include` reference to the primary XML file from the book. The primary XML file for *Book A* will be **Book_A.xml** and for *Book B*, **Book_B.xml**. The **My_Set.xml** file should now look like this:

```
<?xml version="1.0"?>
<!DOCTYPE set PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
"http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd" [
]>

<set>
 <xi:include href="Set_Info.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <xi:include href="Preface.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <xi:include href="Book_A/Book_A.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <xi:include href="Book_B/Book_B.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <xi:include href="Revision_History.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
</set>
```

4. Test your set by running the **`publican build --formats=test --langs=en-US`** command.

Important

When building a set, the **`publican clean_ids`** command will be run over each book because of the constraint that IDs must be unique across all books. Be careful of creating IDs that rely on content that may not be available when building books independently of the set.

Frequently Asked Questions

Q: How do I add a language to my book?

A: Run **`publican update_po --langs=`*`language`***, where *`language`* is the code for the new language that you want to add. You can add more than one language at a time, with the language codes separated by commas. For example, **`publican update_po --langs=ja-JP`** creates the Japanese language directory and Japanese PO files, and **`publican update_po --langs=ja-JP,ko-KR`** creates directories and PO files for both Japanese and Korean.

Q: What if I do not want to use the country code? For example, can I run **`publican update_po --langs=es,de,fr`**?

A: Yes — this command works. However, if you omit the country code, the output might be unpredictable when **Publican** or a brand has definitions for more than one regional variety of a language — for example, **`zh-CN`** (Simplified Chinese as used in the People's Republic of China) and **`zh-TW`** (Traditional Chinese as used in the Republic of China, on Taiwan). Even when only one variety is currently defined, it is always safest to include the country code so that, for example, a future update of **Publican** does not suddenly cause your German (**`de-DE`**) documents to switch to Schweizerdeutsch (Swiss German, **`de-CH`**) Common Content and headings.

Q: How do I update all po files?

A: Run the **`publican update_po --langs=all`** command.

Q: Where can I get a complete list of **Publican**'s build options?

A: Run the **`publican build --help`** command.

Q: Where can I get a complete list of parameters that can be set in the **`publican.cfg`**?

A: Run the **`publican help_config`** command in a directory that holds any **Publican** document.

Q: Where are the **Publican** common files located?

A: By default, they are in **`/usr/share/publican/`** on Linux operating systems and in **`%SystemDrive%/%ProgramFiles%/publican/Common_Content`** on Windows operating systems — typically, **`C:/Program Files/publican/Common_Content`**.

Q: I have extensive code samples for my book, how can I include them without having to XML escape everything?

A: The best way to do this is to create a directory named **`extras`** in your source language directory and use an `xi:include` to pull in the code file.

Procedure 6.1. Including code samples

1. Create the extras directory

```
mkdir en-US/extras
```

2. Copy the code file to the extras directory

```
cp ~/samples/foo.c en-US/extras/.
```

3. `xi:include` the sample file in your xml file

```
<programlisting>
<xi:include parse="text" href="extras/foo.c" xmlns:xi="http://www.w3.org/2001/
XInclude" />
</programlisting>
```

4. You can now edit **`en-US/extras/foo.c`** in your favorite editor without having to be concerned about how it will affect the XML.

Q: Is it possible to include arbitrary files in tarballs and RPM packages?

A: Yes. If you make a directory named **`files`** in your source language directory it will be included in any tarballs or SRPM packages that **Publican** creates.

Important

The **`files`** directory will not be available during the validation process so you can not `xi:include` or otherwise embed any files in this directory in your XML.

Q: Why does **Publican** give me warnings about unknown tags?

A: This warning informs you that you are using a tag whose output has not been tested for attractiveness, XHTML 1.0 Strict compliance, or Section 508 (Accessibility) compliance.

Q: ~~Which brands enable strict mode?~~ Strict mode is no longer enforced.

A: ~~Currently the Red Hat and JBoss brands enable strict mode.~~

Q: I get an error saying **Batik** is not in the classpath but **Batik** is installed! What is wrong?

A: We believe this is due to classpath issues caused by having different JRE and JDK versions installed. Sometimes this can be fixed by upgrading your JDK to the same version of your JRE.

Sometimes this issue can be revealed by running **`alternatives --config java`** and **`alternatives --config javac`**, if the versions are different then selecting the same version in both can fix this problem.

Some Java installs do not set-up the **`alternatives`** environment correctly, no fix has been determined for this situation.

Q: I get an error **`Exception in thread "main" java.lang.OutOfMemoryError: Java heap space`** when trying to build PDF. What is wrong?

A: The default memory allocated for Java is not big enough to build your PDF. You need to increase the memory allocated to **FOP**. Before running **`make`** run **`echo "FOP_OPTS='-Xms50m -Xmx700m'" > ~/.foprc`**. This sets the initial heap space to 50 MB and allows it to grow to a maximum of 700 MB.

Q: Previous versions of **Publican** removed empty <para> tags. Does **Publican** still do this?

A: No. **Publican** previously removed empty <para> tags while it transformed XML because empty <para> tags broke earlier translation toolchains used within Red Hat and the Fedora Project. Empty <para> tags are valid DocBook XML, and **Publican** no longer removes them.

Q: What happened to the spell check?

A: Early versions of **Publican** (up to and including 0.45) ran a spell check while transforming a document's XML. Due to negative feedback from users, this feature was dropped.

Q: Why don't <segmentedlist>s work when I build PDFs?

A: Check the number of columns in your <segmentedlist>s. When <segmentedlist>s are formatted as tables, the DocBook XSL limits the number of columns to two, and **Publican** formats <segmentedlist>s as tables.

Q: What happened to the colors in my images in this PDF?

A: This is the result of a bug in **FOP** that distorts colors in 24-bit PNG images. Convert your images to 32-bit PNG images to work around the problem.

Q: When I build my document, I get an error about an 'undefined language' — what's wrong?

A: Code highlighting in **Publican** is generated with the **Syntax::Highlight::Engine::Kate** Perl module. If you specify a language in a <programlisting> tag that **Syntax::Highlight::Engine::Kate** does not recognize, you receive an error when you build your book. The first lines of the error message are similar to:

```
undefined language: JAVA at /usr/lib/perl5/vendor_perl/5.10.0/Syntax/Highlight/Engine/
Kate.pm line 615.
cannot create plugin for language 'JAVA'
```

Note that **Syntax::Highlight::Engine::Kate** is very strict about names of languages and is case sensitive. Therefore, <programlisting language="Java"> works, but <programlisting language="java"> and <programlisting language="JAVA"> do not. The error message that you receive identifies the problematic language attribute.

Refer to *http://search.cpan.org/~szabgab/Syntax-Highlight-Engine-Kate-0.06/lib/Syntax/Highlight/Engine/Kate.pm#PLUGINS* for the full list of languages that **Syntax::Highlight::Engine::Kate** supports, including their expected capitalization and punctuation.

Q: Why does Jeff call Isaac 'Ivan'?

A: Because Jeff's memory is pants!

Appendix A. Disallowed elements and attributes

Supported, unsupported, and disallowed

Not every *element* (tag) and attribute that works with **Publican** is *supported*. Specifically, not every tag has been tested with regards its effect on the presentation of a document once it has been built in HTML or PDF.

Publican works with almost all DocBook 4.5 elements and their attributes, and most of these elements are *supported*. Supported elements and attributes are those whose presentation in **Publican** HTML and PDF output has been tested and is of an acceptable quality.

Other elements and attributes that are not known to be harmful or redundant but which have not been tested for quality are *unsupported*. If material within a particular DocBook tag does not look correct when you build a document in HTML or PDF, the problem could be that the transformation logic for that tag has not yet been tested. Build the document again and examine **Publican**'s output as the document builds. **Publican** presents warnings about unsupported tags that it encounters in your XML files.

Finally, a small group of elements and attributes are *disallowed*. These elements and attributes are set out below, each accompanied by rationale explaining why it is disallowed.

Use the command **`publican print_known`** to print a list of tags that **Publican** supports, and the command **`publican print_banned`** to print a list of tags that are banned in **Publican**.

A.1. Disallowed elements

`<caution>`, `<tip>`

DocBook XML supports five *admonitions* of varying severity: `<tip>`, `<note>`, `<important>`, `<caution>`, and `<warning>`. Taken together, these represent a very fine-grained set of distinctions. It is unlikely that these fine distinctions can be applied consistently within a document, especially when more than one person writes or maintains the document. Moreover, this level of granularity is meaningless to readers. By design, **Publican** disallows the `<tip>` and `<caution>` elements, these elements being the two most redundant in the set.

Use `<note>` instead of `<tip>`, and use either `<important>` or `<warning>` instead of `<caution>`. Some criteria by which you might select a suitable level of severity are presented in the ‘Document Conventions’ section of the preface of books produced with **Publican**'s default brand.

`<entrytbl>`

Publican depends on an external application, **FOP**, to render PDF documents. At present, **FOP** does not support nested tables, so attempts to build PDF files from **Publican** documents that contain nested tables fail.

Nested tables are therefore disallowed at least until they are supported in **FOP**. If you planned to include a nested table in your document, reconsider your data structure.

`<glossdiv>`, `<glosslist>`

This tag set presents terms in glossaries in alphabetical order; however, the terms are sorted according to the original language of the XML, regardless of how these terms are translated into any other language. For example, a glossary produced with `<glossdiv>`s that looks like this in English:

A

`Apple` — an *apple* is…

G

`Grapes` — *grapes* are…

O

`Orange` — an *orange* is…

P

`Peach` — a *peach* is…

looks like this in Spanish:

A

`Manzana` — la *manzana* es…

G

`Uva` — la *uva* es…

O

`Naranja` — la *naranja* es…

P

`Melocotonero` — el *melocotonero* es…

In a translated language that does not share the same writing system with the original language in which the XML was written, the result is even more nonsensical.

`<inlinegraphic>`

This element presents information as a graphic rather than as text and does not provide an option to present a text alternative to the graphic. This tag therefore hides information from people with visual impairments. In jurisdictions that have legal requirements for electronic content to be accessible to people with visual impairments, documents that use this tag will not satisfy those requirements. Section 508 of the *Rehabilitation Act of 1973*[1] is an example of such a requirement for federal agencies in the United States.

Note that `<inlinegraphic>` is not valid in DocBook version 5.

`<link>`

The `<link>` tag provides a general-purpose hyperlink and therefore offers nothing that the `<xref>` and `<ulink>` tags do not, for internal and external hyperlinks respectively. The `<link>` tag is disallowed due to its redundancy.

Refer to *http://www.section508.gov/*

`<olink>`

The `<olink>` tag provides cross-references between XML documents. For `<olink>`s to work outside of documents that are all hosted within the same library of XML files, you must provide a URL for the document to which you are linking. In environments that use `<olink>`s, these URLs can be supplied either as an XML entity or with a server-side script. **Publican** produces documents intended for wide dissemination in which URLs are always necessary for cross-references. Therefore, the `<olink>` tag offers no advantage over the `<ulink>` tag, and is disallowed due to its redundancy.

A.2. Disallowed attributes

`<[element] xreflabel="[any_string_here]">`

The presence of an `<xreflabel>` attribute reduces the usability of printed versions of a book. As well, attribute values are not seen by translators and, consequently, cannot be translated.

For example, if you have the following:

```
<chapter id="ch03" xreflabel="Chapter Three">
 <title>The Secret to Eternal Life</title>
 <para>The secret to eternal life is…</para>
</chapter>

[more deathless prose here]

…see <xref linkend="ch03"> for details.
```

when your XML is built to HTML, the `<xref>` tag becomes an HTML anchor tag as follows:

```
…see <a href="#ch03">Chapter Three</a> for details.
```

The text contained by the anchor tag is the same as the data in the `<xreflabel>` attribute. In this case, it means that readers of printed copies have less information available to them.

You could work around this if you make the value of the `<xreflabel>` attribute the same as the text within the `<title></title>` element tags. However, this duplication increases the risk of typo-level errors and otherwise offers no underlying improvement. And it still reduces the amount of information presented to readers of printed copies.

The following XML:

```
<chapter id="ch03" xreflabel="The Secret to Eternal Life">
 <title>The Secret to Eternal Life</title>
 <para>The secret to eternal life is…</para>
</chapter>

[more deathless prose here]

…see >xref linkend="ch03"> for details.
```

Will result in an HTML anchor tag as follows:

```
…see <a href="#ch03">The Secret to Eternal Life</a> for details.
```

This isn't as informative as the text presented to a reader if you do not use an <xreflabel> attribute. The following:

```
<chapter id="ch03">
 <title>The Secret to Eternal Life</title>
 <para>The secret to eternal life is…</para>
</chapter>

[more deathless prose here]

…see <xref linkend="ch03"> for details.
```

transforms the <xref> element as follows when built to HTML:

```
…see <a href="#ch03">Chapter 3: The Secret to Eternal Life</a> for details.
```

More important, however, are the translation problems that <xreflabel> tags cause. Attribute values are not seen by translators. Consequently, they are not translated. Consider the second example above again:

```
<chapter id="ch03" xreflabel="The Secret to Eternal Life">
 <title>The Secret to Eternal Life</title>
 <para>The secret to eternal life is…</para>
</chapter>

[more deathless prose here]

…see <xref linkend="ch03"> for details.
```

In English, the <xref> is still transformed into an anchor tag as follows:

```
…see <a href="#ch03">The Secret to Eternal Life</a> for details.
```

Someone reading the German version, however, will have this as their underlying HTML:

```
…Sehen Sie <a href="#ch03">The Secret to Eternal Life</a> für Details.
```

If the <xreflabel> attribute is not used, the title and chapter indicator, both properly translated, appear to the reader. That is, the following:

```
<chapter id="ch03">
```

```
 <title>The Secret to Eternal Life</title>
 <para>The secret to eternal life is…</para>
</chapter>

[more deathless prose here]

…see <xref linkend="ch03"> for details.
```

will, after translation, present thus to a German-speaking reader:

```
…Sehen Sie <a href="#ch03">Kapitel 3: Das Geheimnis des ewigen Lebens</a> für Details.
```

This is, not surprisingly, what we want.

The *xreflabel* attribute is therefore disallowed.

`<[element] endterm="[any_string_here]">`

The *endterm* attribute allows you to present hyperlinked text other than the name of the section or chapter to which the hyperlink points. As such, it decreases the usability of printed versions of documents, and causes difficulty for translators.

The text presented in an element (such as an <xref>) that contains the *endterm* attribute is taken from a <titleabbrev> tag in the target chapter or section. Although the content of the <titleabbrev> tag is available to translators in the document's PO files, it is removed from the context of the <xref>. The absence of this context makes reliable translation impossible in languages that mark prepositions or articles for grammatical number and grammatical gender.

For example, if you have the following:

```
<chapter id="The_Secret">
 <title>The Secret to Eternal Life</title>
 <titleabbrev id="final">the final chapter</titleabbrev>

 <para>The secret to eternal life is…</para>
</chapter>

[more deathless prose here]

The solution is in <xref linkend="The_Secret" endterm="final"/>.
```

The text surrounding the <xref> presents in the English version of the document as:

> The solution is in **the final chapter**.

A translator sees the <titleabbrev> in a PO file as:

```
#. Tag: titleabbrev
#, no-c-format
msgid "the final chapter"
msgstr ""
```

and sees the text that contains the <xref> elsewhere in the PO file (or, more likely, in a completely different PO file) as:

```
#. Tag: para
#, no-c-format
msgid "The solution is in <xref linkend="The_Secret" endterm="final"/>."
msgstr ""
```

The translator has no way of telling what will be substituted for <xref linkend="The_Secret" endterm="final"/> when the document builds, so a translation in Italian might read:

```
#. Tag: para
#, no-c-format
msgid "The solution is in <xref linkend="The_Secret" endterm="final"/>."
msgstr "La soluzione è in <xref linkend="The_Secret" endterm="final"/>."
```

Note the preposition **in**.

If the translator rendered **the final chapter** in Italian as **l'ultimo capitolo**, the result when the document builds will read:

> La soluzione è in **l'ultimo capitolo**.

This result is comprehensible, but inelegant, because Italian combines some of its prepositions with its definite articles. More elegant Italian would be:

> La soluzione è nell'**ultimo capitolo**.

Without knowing what text will appear in place of <xref linkend="The_Secret" endterm="final"/>, the translator into Italian cannot know whether to leave the preposition **in** to stand by itself, or which of seven different possible combinations with the definite article to use: **nel**, **nei**, **nello**, **nell'**, **negli**, **nella**, or **nelle**.

Furthermore, note that the combined preposition and article also poses a problem with regard to whether this word should be placed in the text surrounding the <xref>, or in the <titleabbrev>. Whichever of these two solutions the translator selects will cause problems when the *endterm* appears in other grammatical contexts, because not all Italian prepositions can combine with the definite article in this way.

Due to the problems that *endterm* presents for translation, **Publican** disallows this attribute.

Appendix B. Command summary

Command options

`publican --help`
displays help

`publican --man`
displays the manual page

`publican --help_actions`
displays a list of *actions*

`publican --v`
displays the **Publican** version number.

`--config file`
specifies a config file for a document, in place of the default **`publican.cfg`**.

`--nocolours`
disables ANSI colors in **Publican** logging.

`--quiet`
disables all logging.

Actions

`publican build`
transforms XML into a document. Options:

--formats
comma-separated list of formats to build (mandatory).

--langs
comma-separated list of languages to build (mandatory).

--embedtoc
embeds a table of contents into HTML output.

--publish
sets up built content for publishing.

`publican clean`
removes the temporary directories from a document directory.

`publican clean_ids`
indents XML files neatly, and rebuilds element IDs.

`publican cleanset`
removes local copies of remote books that are part of a set.

`publican create`
creates a new book, article, or set. Options:

--name
: the name of the document (mandatory).

--product
: the documented product.

--version
: the version of the documented product.

--edition
: the edition of the document.

--brand
: the brand for the document.

--lang
: the language in which the XML will be authored.

--type
: the type of document — article, book, or set.

`publican create_brand`
: creates a new brand. Options:

--name
: the name of the document (mandatory).

--lang
: the language in which the XML will be authored.

`publican help_config`
: displays a list of parameters for the **`publican.cfg`** file.

`publican installbrand`
: configures a brand for installation. Option:

--path
: path to the **Publican** Common Content files. By default, **`/usr/share/publican/Common_Content`** on Linux operating systems and at **`%SystemDrive%/%ProgramFiles%/Publican/Common_Content`** on Windows operating systems — typically, **`C:/Program Files/Publican/Common_Content`**

`publican lang_stats`
: generates a translation report for a language.

--lang
: the language for which the report will be generated.

`publican old2new`
: constructs a **`publican.cfg`** file based on the **`Makefile`** of a **Publican 0** document.

`publican package`
: packages a document or brand for distribution. Options:

--lang
: the language to package (mandatory for documents, meaningless for brands).

--desktop
: specifies that a document RPM package should be built for desktop use (meaningless for brands).

--brew
: pushes a package to the **Brew** build system (meaningless outside Red Hat).

--cvs
: specifies that **Publican** should import the generated SRPM package into CVS.

--scratch
: used in conjunction with **`--brew`** to specify a *scratch build* (meaningless outside Red Hat).

--short_sighted
: builds the package without the product version number in the package name.

--binary
: builds the package as a binary RPM package rather than a source RPM package.

publican print_banned
: prints a list of DocBook tags banned by **Publican**.

publican print_known
: prints a list of DocBook tags supported by **Publican**.

`publican printtree`
: displays a tree of the XML files included in a document.

publican print_unused
: prints a list of the XML files *not* included with the `<xi:include>` tag in a book, article, or set.

`publican update_pot`
: updates the POT files in a document.

`publican update_po`
: updates the PO files in a document.

--langs
: comma-separated list of languages to update, or ‘all’ to update all (mandatory).

Appendix C. publican.cfg parameters

Every book, article, document set, or brand has a **publican.cfg** file in its root directory. Parameters that can be set in the **publican.cfg** file are:

docname
: the document name, set by the **--name** option.

version
: the product version, set by the **--version** option.

xml_lang
: the language of the source XML files, set by the **--lang** option.

edition
: the edition number for this documentation, set by the **--edition** option.

type
: the type of document — a DocBook <article>, DocBook <book>, or DocBook <set>, set by the **--type** option.

brand
: the *brand* of the document, set by the **--brand** option.

product
: the product to which this documentation applies, set by the **--product** option.

arch
: the computer *architecture* for this document.

books
: a space-separated list of books used in a remote set.

brew_dist
: the build target to use for building the desktop RPM package in **Brew**. (Default: **docs-5E**)

chunk_first
: whether the first section should appear on the same page as its parent when rendered in HTML. (Default: **0** — the first section starts a new HTML page).

chunk_section_depth
: the point at which **Publican** no longer splits sub-subsections onto a new page when rendering HTML. (Default: **4**)

classpath
: the path to the jar files for **FOP**. (Default for Linux operating systems: **/usr/share/java/ant/ant-trax-1.7.0.jar:/usr/share/java/xmlgraphics-commons.jar:/usr/share/java/batik-all.jar:/usr/share/java/xml-commons-apis.jar:/usr/share/java/xml-commons-apis-ext.jar**)

common_config
the path to the **Publican** installation. (Default for Linux operating systems: **/usr/share/publican**, default for Windows operating systems: **%SystemDrive%/%ProgramFiles%/publican** — most usually **C:/Program Files/publican**)

common_content
the path to the **Publican**'s *Common Content* files. (Default for Linux operating systems: **/usr/share/publican/Common_Content**, default for Windows operating systems: **%SystemDrive%/%ProgramFiles%/publican/Common_Content** — most usually **C:/Program Files/publican/Common_Content**)

condition
conditions on which to prune XML before transformation.

confidential
marks a document as confidential. (Default: **0** — not confidential).

confidential_text
sets the text with which to mark a document as confidential. (Default: **CONFIDENTIAL**).

cvs_branch
the CVS branch into which to import the SRPM.

cvs_pkg
the CVS package into which to import the SRPM. This parameter takes the literal string **__LANG__** and replaces it with the langauge supplied to the package action at run time.

cvs_root
the CVS root into which to import the SRPM.

debug
whether **Publican** should display debugging messages as it works. (Default: **0** — suppress messages)

doc_url
URL for the documentation team for this package. (Default: **https://fedorahosted.org/publican**)

dt_obsoletes
the desktop packages that this package obsoletes.

dtdver
the version of the DocBook XML *Document Type Definition* (DTD) on which this project is based. (Default: **4.5**)

ec_id
the ID for an **Eclipse** help plugin (Default: *product*.*docname*)

ec_name
the name of an **Eclipse** help plugin (Default: *product docname*)

ec_provider
the provider name for an **Eclipse** help plugin (Default: *Publican-Publican version*)

generate_section_toc_level
: the section depth at which **Publican** generates a table of contents. (Default: **0** — no tables of contents in sections)

ignored_translations
: translations to ignore.

license
: the license this package uses. (Default: GNU Free Documentation License).

max_image_width
: the maximum width allowable for images in the document. (Default: **444** — 444 pixels wide)

Important — 444 pixels is the maximum safe width

Do not use the *max_image_width* parameter if your images contain important information. Images wider than 444 pixels might lead to poorly presented HTML and to PDF output that it is unusable because images have run off the page and are presented incomplete to the reader.

Conversely, images lose quality when scaled down in HTML and PDF.

To safeguard the quality of your images, crop or scale them so that they are no wider than 444 pixels before including them in a document.

os_ver
: the operating system for which to build packages. (Default: **.el5** — Red Hat Enterprise Linux 5)

prod_url
: URL for the product to which this document applies. (Default: **https://fedorahosted.org/publican**)

release
: the release number of this package. Defaults to the value of **xml_lang**, fetched from the title tag in **xml_lang/TYPE_Info.xml** or **Project-Id-Version** in **lang/TYPE_Info.po**.

repo
: the repository from which to fetch remote books that form part of a distributed set.

scm
: the version control system used in the repository in that stores the remote books in a distributed set. (Default: **SVN**)

show_remarks
: whether to display remarks in transformed output. (Default: **0** — hide remarks)

show_unknown
: whether **Publican** reports unknown tags when processing XML. (Default: **1** — report unknown tags)

src_url
: URL at which to find to find tarballs of source files.

strict
: ~~use *strict mode* (Default: **0** — not strict)~~ Strict mode is no longer enforced.

tmp_dir
: the directory for **Publican** output. (Default: **tmp**)

toc_section_depth
: the depth of sections that **Publican** includes in the main table of contents. (Default: **2**)

web_brew_dist
: the **brew** build target to use for the web RPM package. (Defaults to **docs-5E**)

web_obsoletes
: packages that the web RPM package obsoletes.

Appendix D. Language codes

Region subtags

The only part of the XML language tag that is mandatory in **Publican** is the *language subtag*. However, **Publican** is designed with the assumption that you will routinely include the *region subtag* when you identify languages. In many languages, spelling and vocabulary vary significantly from region to region. If you do not specify the regional variety of a language in which your document is authored or into which it is translated, you might obtain unexpected results when you build the document in **Publican**.

Other language codes

The system of codes used to identify languages in the XML standard is not the only system of languages codes in use in the world today. However, because **Publican** strives to comply with the XML standard, these are the only codes that **Publican** supports. In particular, note that the codes used in the GNU tools (identified by their use of underscores and the `@` symbol to separate elements — for example, **`en_GB`** or **`sr_RS@latin`**) do not comply with the XML standard and therefore do not work with **Publican**.

Publican is an XML publication tool and therefore is designed to use the language codes — or *tags* — that the World Wide Web Consortium (W3C) designated in the XML specification.[1] These codes are defined in the Internet Engineering Task Force (IETF) document *BCP 47: Tags for Identifying Languages*.[2]

Language tags are built from one of more *subtags*, separated from one another by hyphens. In order of appearance within a language tag, these subtags are:

`language-script-region-variant`

BCP 47 also allows for considerable customization of language tags for special purposes through the use of *extension subtags* and *private-use subtags*. Extension subtags allow for finer-tuning of existing subtags, but must be registered with the IETF (none are currently registered). Private-use subtags are introduced by **`x-`** and do not need to be registered. Private-use subtags aside, a subtag is valid if it appears in the registry of subtags maintained by the IETF through the Internet Assigned Numbers Authority (IANA).[3] Although **Publican** will accept any language tag that is valid under the rules presented in BCP 47, it is designed around the assumption that language tags for documents will most usually take the form ***`language-region`***. A brief description of subtags follows:

language subtag

The language subtag comprises two or more lower-case letters and is the only mandatory part of the language tag. For most widely spoken languages, the language subtag is a two-letter code identical with the language codes specified in ISO 639-1, [4] for example, **`zh`** (Chinese), **`hi`** (Hindi), **`es`** (Spanish), and **`en`** (English). Where no two-letter code exists in ISO 639-1, the

http://www.w3.org/TR/REC-xml/#sec-lang-tag
http://tools.ietf.org/html/bcp47
http://www.iana.org/assignments/language-subtag-registry
http://www.infoterm.info/standardization/iso_639_1_2002.php

language subtag is usually a three-letter code identical with the codes specified in ISO 639-2,[5] for example, **bal** (Balochi), **apk** (Kiowa Apache), and **tpi** (Tok Pisin). Finally, a small number of language subtags appear in the IANA registry that have no ISO 639-1 or ISO 639-2 equivalent, such as subtags for the constructed languages **qya** (Quenya) and **tlh** (Klingon), and for the occult language **i-enochian** (Enochian). This last example also illustrates a small number of language subtags *grandfathered* into the registry that do not match the two-letter or three-letter pattern of codes derived from the ISO 639 standards.

Extended language subtags

RFC 5646: Tags for Identifying Languages[6] issued in September 2009 allows for *extended language subtags* to follow the language subtag. Extended language subtags are three-letter codes that represent languages that share a close relationship with a language already represented by a language subtag. For example, **yue** represents Cantonese, but this subtag must always be used with the language subtag associated with it (Chinese), thus: **zh-yue**. The IETF does not yet recognize RFC 5646 as "Best Common Practice", nor are these subtags part of the XML standard yet.

script subtag

The script subtag comprises four letters — the first one in upper case, the other three in lower case — and defines a writing system. These codes are identical with the four-letter codes specified in ISO 15924.[7] The script subtag is used to identify languages that are commonly written with more than one writing system; the subtag is omitted when it adds no distinguishing value to the language tag overall. For example, **sr-Latn** represents Serbian written with the Latin alphabet and **sr-Cyrl** represents Serbian written with the Cyrillic alphabet; **az-Arab** represents Azerbaijani written in Arabic script and **az-Cyrl** represents Azerbaijani written with the Cyrillic alphabet. Conversely, French should not be represented as **fr-Latn**, because French is not commonly written in any script other than the Latin alphabet anywhere in the world.

region subtag

The region subtag comprises either two upper-case letters (for regions that conform to national boundaries) or three digits (for other areas, such as trans-national regions). The two-letter subtags are identical with those from ISO 3166-1[8], for example, **AT** (Austria), **TZ** (Tanzania), and **VE** (Venezuela). The three-digit region subtags are based on those in UN M.49, [9] for example, **015** (Northern Africa), **061** (Polynesia), and **419** (Latin America and the Caribbean).

variant subtag

Variant subtags identify well-defined, recognizable variants of a language or script and can include upper-case letters, lower-case letters, and numerals. Variant subtags that start with a letter must be at least five characters long, and those that start with a numeral must be at least four characters long. Most variant subtags can only be used in combination with specific subtags or combinations of subtags. Variant subtags do not harmonize with any other standard; they are each the result of a separate registration with the IETF by an interested person or group.

Under the present standard, dialects of several languages are designated with variant subtags, for example, **nedis** denotes Nadiza (also known as Natisone), a dialect of Slovenian. This tag must

http://www.loc.gov/standards/iso639-2/
http://www.unicode.org/iso15924/
http://www.iso.org/iso/country_codes.htm
http://unstats.un.org/unsd/methods/m49/m49.htm

be used in conjunction with the language subtag for Slovenian, thus: **`sl-nedis`**. In September 2009, the IETF issued a Request for Comments (RFC) that (amongst other things) proposes that dialects be represented by language extension subtags attached to language subtags.[10]

Most variant subtags mark a particular orthography, most usually as a result of an official spelling reform or a significant work documenting the language. Examples (with their required language subtags) include: **`fr-1606nicot`** (French as documented by Jean Nicot in 1606), **`de-1901`** (German spelling codified by the 2nd Orthographic Conference in 1901) and **`be-1959acad`** (Belarusian as codified by the Orthography Commission in 1959).

Finally, some variant subtags denote a particular variant of a system of writing or transliteration. For example, **`zh-Latn-wadegile`** is Chinese written in the Latin alphabet, according to the transliteration system developed by Thomas Wade and Herbert Giles; **`ja-Latn-hepburn`** is Japanese written in the Latin alphabet using the transliteration system of James Curtis Hepburn.

Publican includes support for the following languages:

- ar-SA — Arabic
- as-IN — Assamese
- ast-ES — Asturian
- bg-BG — Bulgarian
- bn-IN — Bengali (India)
- bs-BA — Bosnian
- ca-ES — Catalan
- cs-CZ — Czech
- da-DK — Danish
- de-CH — German (Switzerland)
- de-DE — German (Germany)
- el-GR — Greek
- es-ES — Spanish
- fa-IR — Persian
- fi-FI — Finnish
- fr-FR — French
- gu-IN — Gujarati
- he-IL — Hebrew
- hi-IN — Hindi

http://tools.ietf.org/html/rfc5646

- hr-HR — Croatian
- hu-HU — Hungarian
- id-ID — Indonesian
- is-IS — Icelandic
- it-IT — Italian
- ja-JP — Japanese
- kn-IN — Kannada
- ko-KR — Korean
- lv-LV — Latvian
- ml-IN — Malayalam
- mr-IN — Marathi
- nb-NO — Norwegian (Bokmål orthography)
- nl-NL — Dutch
- or-IN — Oriya
- pa-IN — Punjabi
- pl-PL — Polish
- pt-BR — Portuguese (Brazil)
- pt-PT — Portuguese (Portugal)
- ru-RU — Russian
- si-LK — Sinhalese
- sk-SK — Slovak
- sr-Cyrl-RS — Serbian (Cyrillic script)
- sr-Latn-RS — Serbian (Latin script)
- sv-SE — Swedish
- ta-IN — Tamil
- te-IN — Telugu
- th-TH — Thai
- uk-UA — Ukrainian
- zh-CN — Chinese (People's Republic of China, implicitly simplified Han script)

- zh-TW — Chinese (Republic of China, implicitly traditional Han script)

Appendix E. Revision History

Revision 1.6-1 Mon May 24 2010 Rüdiger Landmann *r.landmann@redhat.com*

Update Ubuntu installation instructions

Revision 1.6 Fri May 7 2010 Rüdiger Landmann *r.landmann@redhat.com*

Revise action and option nomenclature
Document print_known, print_banned, and print_unused actions
Correct and expand documentation on installing a brand
Document max_image_width and confidential_text parameters
Document Eclipse help plugin format and supporting parameters

Revision 1.5 Fri Feb 26 2010 Rüdiger Landmann *r.landmann@redhat.com*

Document --config option

Revision 1.4 Wed Feb 17 2010 Jeff Fearn *jfearn@redhat.com*

remove obsolete reference to path to the DocBook catalog files. BZ#565498.
document CVS options.

Revision 1.3 Mon Dec 7 2009 Rüdiger Landmann *r.landmann@redhat.com*

Add an FAQ entry about code highlighting errors.
Add a section about valid formats.
Update author list.
More specific installation instructions for Ubuntu; add installation instructions for Debian. BZ#542711
Metadata in the Book_Info.xml file

Revision 1.2 Fri Nov 27 2009 Jeff Fearn *jfearn@redhat.com*

Document lang_stats action. BZ#540696.

Revision 1.1-1 Thu Nov 26 2009 Jeff Fearn *jfearn@redhat.com*

Fix wrong docs for condition usage. BZ#540691

Revision 1.1 Thu Oct 22 2009 Rüdiger Landmann *r.landmann@redhat.com*

Fix various small inconsistencies and general clean up

Revision 1.0 Tue Oct 13 2009 Rüdiger Landmann *r.landmann@redhat.com*

Updated for Publican 1.0

Revision 0.5 Thu Dec 18 2008 Jeff Fearn *jfearn@redhat.com*

Added appendix on Makefile parameters
Added entry to FAQ about java heap space.

Revision 0.4 Tue Nov 25 2008 Brian Forté *bforte@redhat.com*

Added "Pre-release and draft documentation" section.

Revision 0.3 Fri Oct 10 2008 Don Domingo *ddomingo@redhat.com*

Adding "Conditional Tagging" section.

Revision 0.2 Fri Sep 05 2008 Brian Forté *bforte@redhat.com*

General edits and updates related to Publican 0.36 release. Also, new section added to Chapter 3.3.

Revision 0.1.1 Fri Jun 06 2008 Murray McAllister *mmcallis@redhat.com*

Updated Branding to note addition of oVirt and GIMP brands

Revision 0.1 Fri May 16 2008 Jeff Fearn *jfearn@redhat.com*

Updated FAQ

Revision 0.0 Thu Dec 13 2007 Murray McAllister *mmcallis@redhat.com*

Initial content release