Borges DMS

Self-Documentation

Edited by
Camille Bégnis

Joël Pomerleau
joel@mandrakesoft.com

Christian Roy
croy@mandrakesoft.com

Fabian Mandelbaum
fabman@mandrakesoft.com

Peter Pingus
pp@mandrakesoft.com

Jerry Huynh-Tot
jerry@mandrakesoft.com

Borges DMS: Self-Documentation
Published 2002-04-19

Edited by and Camille Bégnis,Joël Pomerleau, Christian Roy, Fabian Mandelbaum, Peter Pingus, and Jerry Huynh-Tot

Table of Contents

Preface

1. Legal Notice

2. About Borges Documentation

Borges is an XML based extensible document management system powered by open source technologies. It is designed to facilitate the management of multiple languages, content reusability and teamwork.

This manual is intended for Borges users. You will learn how to install Borges, create a new project and use it. This manual also includes a User Reference Guide that further explains the core functionalities such as `conf/` files, etc. Finally, the Programmer's reference manual will look at the inner working of the application such as creating custom DTDs/stylesheets and adding new modules.

Chapter 1. A Revolutionary Concept

1.1. What is Borges?

`Borges` is an open source extensible documents management system aimed at XML aware documentation projects. It's main purpose is to optimize internationalization (many languages, translations), reusable content and teamwork.

The main philosophy behind `Borges` is to provide a convenient tool:

- For beginners: by providing a very simple interface to compile XML DocBook documents into various formats;
- For advanced users: by providing a whole set of customization features allowing to easily twick every single aspect of the system: output formats and layout, custom rules, etc.
- For project managers: by providing a powerful project tracking system to juggle with authors and translators, deadlines, etc.

1.1.1. Features

The supported DTDs are DocBook and TDB (Training DocBook), a subset of the DocBook DTD written for the training manuals of MandrakeSoft. Adding external DTDs is very easy, even though the revision checking system might not work yet with DTDs other than DocBook and TDB.

Currently, the system allows to:

- Compile the source files into PDF, PS and (X)HTML;
- Manage different versions of a single document by easily defining derived versions based on conditional parts;
- Each module is assigned a set of authors: writer/ translators/ proofreaders, each one responsible for one state of a module. Each contributor can easily review his attributions through web pages, and can receive e-mails with his current todo list;
- Track the work in progress. From the whole project (made of various documents) to the most basic components (paragraphs), and their translations;
- Track the state of each module according to six predefined states. (From "written" to "final language proofreading"). Once a task is completed, the corresponding state is passed and the module switches to the next state;

Quick facts about `Borges`:

- automatic management of images in EPS, PNG, JPEG, `DIA`, `XFig` formats;
- automatic management of global and local (per document) external entities;
- automatic management of modules as external entities.

1.2. Choosing Borges

1.2.1. Do I need it?

This section, instead of presenting features, addresses the needs that `Borges` answers.

Constant revision, Multiple languages

If you manage or publish books that need frequent(constant) revision in multi-languages. `Borges` is for you. It will enable you to track changes at the paragraph or block of text level, maximizing translator's and proof reader's time.

Team Leaders

If you manage a team of authors, even scattered around the net, through it's cvs integration, task, revision and languages management, `Borges` will considerably simplify your life.

Reusable Content

If the content you are publishing is reusable, `Borges` is for you. For example, you write a travel guidebook for the USA and would like, from that same content, to publish books on each individual states without having to manipulate your document. You can also publish one book out of many.

Multiple Format Publishing

In todays Internet world, the format you choose to publish your work is something very likely to change. Furthermore, in a *Customer Relationship Management* perspective, it becomes a great asset to deliver content in the format most suitable for your users. It may be a book, it may be a web site, it may be a downloadable PDF... `Borges`, through it's XML and DocBook foundation is specifically tailored to address these needs... You can define layout for all of those formats and really adopt a content provider approach.

1.2.2. Is Borges for me?

Do not think about using `Borges` if you:

- seldom write documents more than 2 pages long;
- seldom have your documents translated;
- don't want to work under another operating system than Windows™;
- get scared when seeing a text mode console;

Do use `Borges` if you:

- happen to manage many big documents;
- have those documents translated in many languages;
- manage a team of many people involved in the production of these documents;
- regularly lose your hair because the documented items change everyday;
- can rely on someone at ease with *GNU/Linux*;
- wish to bring your documentation project and team into new generation of technical documentation with XML and DocBook.

In short, `Borges` will provide you a solution to efficiently manage big documentation projects, bringing higher quality and reducing delays. The counterpart will be some time to spend reading the documentation and getting used to the system. If necessary installing a *GNU/Linux* system will also be needed. If you don't know DocBook, you'll have to learn it as well.

Still interested in the beast? Congratulations! read on, and good luck. You won't regret it!

1.3. Some Vocabulary

We will explain all terms used in `Borges`' documentation: project, author, author initials, document, sub-document, module, module status, atom, atom revision, etc.

Note: The terms are not presented in any particular order.

Author

An author can be the redactor, the translator or the reviewer of a module. Generally speaking, the "author" concept is bound to the creator (in this case, writer) of something, but `Borges` treats translators and reviewers as authors.

See Also: Author Initials, Module.

Author Initials

`Borges` identifies the different authors that participate in a project by their initials. This limits the initials used by different authors of the same project to be *unique*.

If your project has a small group of authors, two-letter initials should be enough, but more letters can be used as long as they are unique.

See Also: Author, Project.

Project

A project is a document or a set of documents you are managing with `Borges`. Usually, a project contains lots of documents.

See Also: Document.

Super-document

Designates a set of modules, structured together to form a book, an article, a user manual; any exhaustive information block about a particular subject.

The super-document is the "master" from which different documents can be generated. The super-document structure is defined in the `master.top.xml` file.

A super-document can contain mutually exclusive informations that will be sorted out by specializing the super-documents into various documents.

See Also: Document.

Document

A document is a compilation of a super-document resulting in a PDF file or (X)HTML file(s). You may choose to compile all your super-document, or parts of it. Documents can be whole books, articles, reference sheets, letters, manuals, etc.

See Also: Compilation, Super-document.

Compilation

Compilation is the process by which a set of source XML files is "transformed" into a PDF or (X)HTML document.

Structuring element

In a super-document, a structuring element is a DocBook element that contains module elements. Typical structuring element are `part` or `chapter`.

See Also: Super-document, Module element.

Module element

In a super-document, a module element is a DocBook element that contains the special

```
<para role="module">
```

child element. A module element will be replaced in the final document by the module content itself. Typical module element are `chapter` or `sect1`.

See Also: Super-document, Module element.

Module

Modules are the parts that compose documents. Usually, a super-document is divided into small chunks called modules to simplify writing, translating, management and content re-use. Chapters, sections, appendices and glossaries are good candidates to become modules.

In fact, `Borges` requires that any structuring element be placed in a module to be able to be translated and to take advantage of the revision management features.

Modules can have some parts flagged, by means of the `condition=` attribute, in order to be excluded from certain compilations. This gives you the ability to create more than one kind of document from *a single set* of modules, improving the content re-use features of `Borges`.

See Also: Document, Super-document, Project.

Original Module

This is used to specify a module which has been written by the module redactor. Translators will use this original module as the base for all translations.

See Also: Module, Translated Module.

Translated Module

Designates a module which is not the original one, but a translation of the original module.

See Also: Module, Original Module.

Module Status

Modules go through different states during their life cycle. Each "state" determines the module's status.

In order to go from one state to another, some operation needs to be performed on the module, for example: writing, translating, spell checking, proofreading, etc.

See Also: Life Cycle.

Atom

Atoms are the XML elements used for checking modifications inside a module. They are the most little possible elements that contain text. Typical DocBook atoms are <title> and <para>.

See Also: Atom Revision.

Atom Revision

Atom's have a revision number used by `Borges` revision management system in order to track changes made into modules at an "atom scale".

See Also: Atom.

Life Cycle

The life cycle of a module is composed by several stages (or states) that a module must go through in order to be considered ready to be released. Currently, `Borges` only supports a fixed life cycle, which is detailed in Section 3.4.1.

See Also: Module, Module Status.

Chapter 2. Quick Start Guide

2.1. Installation

As of now, Borges has only been tested on Mandrake Linux. It should work on any Linux system provided the necessary dependencies are installed. Please inform us of any success or failure on any other system.

2.1.1. Where to get it?

Current versions are published on SourceForge[1]. There, you will find different packagings:

- If you are on an RPM based system, install Borges and Borges-DocBook noarch packages;
- You can also choose to get the tarball (Borges-*.tar.bz2);

The different Borges packages are also part of the Mandrake Linux distribution.

Finally, if you like living dangerously, you can get the current CVS version with following parameters:CVS_RSH=ssh and CVSROOT=:ext:anoncvs@cvs.mandrakesoft.com:/cooker. Then, you can get the module Borges with password cvs.

2.1.2. How to install it?

Just install the RPM packages, or read instructions in the tarball.

> **Note:** Borges installs by default in /usr/share/Borges/.

2.1.3. Dependencies

If you do not install Borges from RPM packages, you'll have to check that the following softwares or libraries are available on your system:

- make
- libxslt-proc from libxslt Gnome project;
- perl;
- perl-XML-Twig, perl-DateManip and perl-XML-LibXML libraries;
- ImageMagick images processor for images transformations;
- dia diagrams editor if you wish to work with dia diagrams;
- xfig diagrams editor if you wish to work with xfig diagrams;
- XFree86-Xvfb is needed for diagrams transformation;
- DocBook DTD XML version 4.1.2 into /usr/share/sgml/docbook/xml-dtd-4.1.2/;
- DocBook DSSSL stylesheets into /usr/share/sgml/docbook/dsssl-stylesheets/;
- DocBook XSL stylesheets into /usr/share/sgml/docbook/xsl-stylesheets/;
- openjade
- tetex-latex
- jadetex

1. http://sourceforge.net/projects/borges-dms/

Tip: If something goes wrong while trying to install `Borges`, make sure that those applications are installed correctly.

2.2. First Steps

`Borges` needs a minimal configuration to work. We will detail the configuration steps necessary to create a project template. Afterwards, the sample document provided with the project template will be compiled into both PDF and HTML and the progress report will be generated.

To configure `Borges` you need to perform the following steps:

1. Define a New Working Directory

   A working directory should be created to hold all your project's files. Let's assume you want to put your files under `My_Project` in your home directory, then you would issue:

   ```
   mkdir ~/My_Project
   ```

   to do so.

   Note: The following steps assume you are under the working directory (`~/My_Project/` in the example).

2. Copy the Template System to The Working Directory

   Now that the working directory has been created, you need to copy a "template system tree" into it. A template directory tree is supplied under the `/usr/share/Borges/template/` directory, so issue

   ```
   cp -a /usr/share/Borges/template/* .
   ```

   to copy the template into your working directory.

3. Create Your Personal Profile

   Each "author" (writer, translator, proofreader, etc.) needs to define his personal profile. `Borges` uses the information in the profile for version management and author credits among other things. The profile is stored in `conf/author.xml` and a sample is provided in the `conf/author.xml.in` file, so just issue

   ```
   cp conf/author.xml.in conf/author.xml
   ```

   and edit `author.xml` with your favorite text editor to suite your personal data. Below you have a sample profile:

   ```
   <?xml version='1.0' encoding='ISO-8859-1'?>
    <author>
     <initials>pp</initials>
     <firstname>Peter</firstname>
     <lastname>Pingus</lastname>
     <mail>peter.pingus@mandrakesoft.com</mail>
     <lang>en</lang>
    </author>
   ```

   Tip: Take a look at the comments in `author.xml` for hints about the meaning of the parameters (In the above sample, the comments are excluded for simplicity reasons).

4. Initialize the System

   Now, `Borges` has to be initialized. To do so, just issue

   ```
   make
   ```

   and directories will be populated with the minimum required files.

5. Generate the Modules Templates

   Now you have the system configured, it is time to test if it works properly. A sample document (called `Sample`) is provided, so we will use that one for testing purposes. Issue

   ```
   make -C manuals/Sample templates LANG=en
   ```

   to generate the modules templates for the sample document.

   > **Tip:** the `-C` argument of the **make** command simply means to make the `templates` target in the `manuals/Sample` directory. You could have run
   >
   > ```
   > cd manuals/Sample; make templates LANG=en
   > ```
   >
   > also.

6. Compile Sample to PDF and Check the Result

   Now you can compile the sample document to PDF to check how it looks. Issue

   ```
   make -C manuals/Sample master.pdf LANG=en
   ```

   to do so, and check the resulting PDF by issuing

   ```
   xpdf manuals/Sample/master.pdf
   ```

   if everything went well, you should see a nice PDF of the sample document. Of course, you can use `Acrobat Reader` instead of `Xpdf` to open the PDF if you prefer to.

7. Compile Sample to HTML and Check the Result

   You can also compile the sample document to HTML. Issue

   ```
   make -C manuals/Sample master.flat.html LANG=en
   ```

   to do so, and check the results by pointing your favorite browser to `~/My_Project/manuals/Sample/master.f`

8. Generate and View the Report

   The report is a tool of `Borges` which informs you about the progress of the work being done in your project for all supported languages. To generate the report for the sample document, issue

   ```
   make -C reports all LANG=en
   ```

   and view the resulting report by pointing your favorite web browser to `~/My_Project/reports/index.html`.

   > **Note:** In all the above examples the `LANG=en` parameter is *mandatory* if your preferred language is other than English (`en`). The preferred language was set in the `author.xml` file, remember?

It was not that hard wasn't it? Now, you can setup `Borges` to work with your own projects.

2.3. Beginning Your Own Project

We will first outline the steps needed to configure `Borges` for a new project and then a step-by-step example will be provided.

2.3.1. Configuring Borges to Start a New Project

> **Warning**
>
> In the following pages it is assumed that `Borges` is already configured properly as explained in Section 2.2 at least up to the point of creating your personal profile.

First, you should clean the files and directories related to the sample document provided with the project template. Issue the following

```
rm -rf manuals/Sample
rm -rf modules/en
rm -rf manuals/images
rm -rf images/en
rm -rf entities/en
```

to do so. You should also clean the main configuration file (see Section 2.3.2.1). You should also define in this file the title for your whole project.

Next, you have to perform the following steps (see Section 2.3.2 below for details):

1. Prepare the master file. The master file outlining your project's structure needs to be created and edited.

   Tip: You can think of the master file as the "skeleton" of your future document.

2. Initialize the repository. This is mandatory for the system to work properly.
3. Define entities. Entities for titles and names (for example, application names, company names, etc.) need to be defined. The importance of entities is explained in Section 2.3.2.4 and in Section 3.1.2.1.
4. Generate the writers' guidelines. The writers' guidelines is a PDF or HTML file compiled from the master file having your project's structure as its content. Once generated the file must be read with an appropriate tool (`Xpdf` or `Acrobat Reader`, for example) to check its validity.
5. Assign tasks to every author. Ideally you should be able now to assign a reponsible for every single task of the life cycle of every module.
6. Write the modules and create images. Now, authors can start writing the different modules that make up your project and creating the modules' associated images (if needed).
7. Check the result. You can check the progress of the work being done on your project (writing, translating, etc.) by compiling the project and reading the resulting PDF from time to time.

In the following section a step by step example is provided to clarify the points detailed above.

2.3.2. Step by Step Example

Let's say you want to start a new book named "My_Book" consisting of a preface and two chapters: the first with two sections and the last one with three sections. You also want to include two appendices and want your book to be translated into French and Spanish.

So, here is what you have to do, step-by-step.

Note: In all the following examples comments in files are excluded for simplicity reasons. Luckily, all configuration files are self-documented so you can always refer to them for an explanation of a particular configuration option.

Note: All examples of command lines to issue assume that the current directory is `~/My_Project/` (you can use **pwd** to check that).

2.3.2.1. Edit the Main Configuration File

`Borges` is designed to handle multiple manuals and languages; to define your project details, it uses a file named `repository.xml` stored under the `conf/` directory.

The `conf/repository.xml` file for your starting project should look like this:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<configuration>
 <repository>
  <title>My Book Project</title>
  <paths>
   <modules>modules</modules>
   <manuals>manuals</manuals>
  </paths>
  <outputs>
   <makefile>/usr/share/Borges/backend/Makefile.DB</makefile>
  </outputs>
  <manuals>
  </manuals>
  <languages>
    <lang>en</lang>
  </languages>
  <revisions>
   <type>
    <name>lproof</name>
    <author>tbn</author>
   </type>
   <type>
    <name>ispell</name>
    <author>tbn</author>
   </type>
   <type>
    <name>pproof</name>
    <author>tbn</author>
   </type>
   <type>
    <name>tproof</name>
    <author>tbn</author>
   </type>
   <type>
    <name>translate</name>
    <author>tbn</author>
   </type>
   <type>
    <name>write</name>
    <author>tbn</author>
   </type>
  </revisions>
 </repository>
</configuration>
```

The file is pretty self-explanatory, however there are some things to note. The `<manuals>` section contains all the documents (one `<manual>` entry per document) handled by `Borges`. The `<languages>` section contains all supported languages for all projects (one `<lang>` entry containing the two letter ISO code of the language per each language).

Note: There is no document defined yet, and no language but the default one you wish to use for your project. Other documents and languages will be added later through the command line.

The `<revisions>` section defines the document's workflow[2] which represents the "life cycle" of the document or the "stages" through which a document must pass.

Tip: `tbn` (To Be Named) *must* be used as the value for `<author>` to tell `Borges` that the person responsible for that revision is not defined yet. You can put here the initials of the person that will be responsible for a specific step by default. Then for all new modules in all languages, this person will be marked as responsible for this specific step.

2.3.2.2. Define the Document Structure

We spoke about "document structure" a lot, right? Well, time has come to define it. We need to create a file named `master.top.xml`. You can copy `/usr/share/Borges/template/manuals/Sample/master.` to `~/My_Project/master.top.xml` and edit it to fit your needs.

The `master.top.xml` file for your project should look like this:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
"/usr/share/sgml/docbook/xml-dtd-4.1.2/docbookx.dtd"[
<!ENTITY % entities SYSTEM "entities">
%entities;
]>

<book id="My_Book" lang="&lang;">
 <title>&book-title;</title>
 <bookinfo>
  <title>&book-title;</title>
  <subtitle>&book-sub-title;</subtitle>
  <revhistory>
   <revision lang="en">
    <revnumber>1</revnumber>
    <date>2002-06-04</date>
    <authorinitials>pp</authorinitials>
    <revremark>First Draft</revremark>
   </revision>
   <revision lang="fr">
    <revnumber>1</revnumber>
    <date>2002-06-14</date>
    <authorinitials>pt</authorinitials>
    <revremark>Begin French Translation</revremark>
   </revision>
   <revision lang="es">
    <revnumber>1</revnumber>
    <date>2002-06-10</date>
    <authorinitials>rp</authorinitials>
    <revremark>Begin Spanish Translation</revremark>
   </revision>
  </revhistory>
  <abstract><para></para></abstract>
  <authorgroup>
   <editor id="pp">
    <firstname>Peter</firstname>
    <surname>Pingus</surname>
    <affiliation>
     <address><email>pp@co.net</email></address>
    </affiliation>
   </editor>
   <author id="pt">
    <firstname>Pierre</firstname>
    <surname>Tremblay</surname>
```

2. At the moment of this writing the workflow is *fixed* and cannot be changed.

```
    <affiliation>
     <address><email>pt@co.net</email></address>
    </affiliation>
   </author>
   <author id="rp">
    <firstname>Reina</firstname>
    <surname>Pingüino</surname>
    <affiliation>
     <address><email>rp@co.net</email></address>
    </affiliation>
   </author>
  </authorgroup>
  <authorblurb>
   <para>&e-mail; doc@co.net</para>
   <para>&web; www.co.net</para>
  </authorblurb>
  <pubdate>2002-06-20</pubdate>
  <copyright>
   <year>2002</year> <holder>Your_Company</holder>
  </copyright>
 </bookinfo>
 <preface id="legal-notice">
  <para role="module">legal-notice-legalnotice </para>  <para>Put
  here the content of the former legal notice section. If not all
  can fit here in the physical page, ask the team. </para>
 </preface>
 <chapter>
  <title>&chapter-1-title;</title>
  <sect1 id="sect-1-1">
   <title>First Section Title</title>
   <para role="module">chap1-sect1-section</para>
   <para>Write the chapter 1, section 1 contents</para>
  </sect1>
  <sect1 id="sect-1-2">
   <title>Second Section Title</title>
   <para role="module">chap1-sect2-section</para>
  </sect1>
 </chapter>
 <chapter>
  <title>&chapter-2-title;</title>
  <sect1 id="sect-2-1">
   <title>Second Chapter First Section Title</title>
   <para role="module">chap2-sect1-section</para>
  </sect1>
  <sect1 id="sect-2-2">
   <title>Second Chapter Second Section Title</title>
   <para role="module">chap2-sect2-section</para>
  </sect1>
  <sect1 id="sect-2-3">
   <title>Second Chapter Third Section Title</title>
   <para role="module">chap2-sect3-section</para>
  </sect1>
 </chapter>
 <appendix id="appendix-1">
  <title>First Appendix Title</title>
  <para role="module">app1-appendix</para>
 </appendix>
 <appendix id="appendix-2">
  <title>Second Appendix Title</title>
  <para role="module">app2-appendix</para>
 </appendix>
</book>
```

Thinking in a more or less “modular” way we can say that, generally speaking, a book has: a title, some information about the book itself (sub-title, authors, editor, revisions, publication date, etc.), a preface, chapters and appendices. So, that is exactly what is represented in the sample `master.top.xml` above, no more, no less.

However, there are some “special” things you might have noticed:

1. The `role="module"` attribute. Whenever `Borges` finds a `para` element having this attribute, it will go up to the element's "father" (usually, a `sectX`, `chapter`, `appendix` element), and will replace that whole "father" with the entity named after the content of that `para`.

   For example, the whole `sect1` element

   ```
   <sect1>
    <title>Some Title</title>
    <para role="module">some_sect-sect1</para>
    <para>Introduce here the Tartempion application, tell why it rocks.</para>
   </sect1>
   ```

   will be replaced by the entity `&some_sect-sect1;`, which in turn refers to the module `some_sect-sect1.xml` stored under the `modules/ll/` directory, where `ll` is the ISO two letter code for the language you want to compile the book in.

   This way of deriving the resulting document directly from the specifications document ensures that there is no discrepancy between specs and final result. Furthermore, the system publishes those direction for writers in the the spec file and in the module templates. They will have disappeared in the final document. Do not hesitate to make those guidelines as lengthy as necessary.

2. There are many entities notably for titles. As this master file will serve as a skeleton for our final document, all texts in it must be enclosed in entities so that the master is language-independent. There is a notable exception however: you remember that module elements will be entirely replaced by the modules themselves. So texts in these elements can stay here in (here our sections titles for example).

Note: You might need to change the XML `SYSTEM` declaration of the DocBook DTD (`"/usr/share/sgml/docbook/xml-dtd`
to fit your system.

2.3.2.3. Insert the New Document

Now that the structure of the document is defined, the system can create the directories and files to support this new document. This is all done in one single command:

```
make adddoc doc=My_Project master=master.top.xml
```

that will create the new `My_Doc` document based on the master file `master.top.xml`. It will actually perform the following tasks:

- Update `conf/repository.xml`;
- create `manuals/My_Doc/` directory and populate it with all needed files and languages directories;
- Make all module templates for this new documents in all defined languages;
- Add all new files to the CVS repository, if available. Note that you'll still need to commit those files by hand;

2.3.2.4. Define Entities

Project and documents entities need to be defined. Project entities are those entities common to all documents, for example: computer program names. Documents entities are those entities used only in a particular document. All entities files are XML files which file names *must* end in `.ent`.

Project entities files go into the `entities/` directory.

Master entities files go into the `manuals/My_Book/ll/` directory where `ll` is the two letter ISO code for the language. All entities defined in `master.top.xml` will have to be defined here.

> **Note:** Global entities are covered more thoroughly in Section 3.1.2.1.

There are two predefined documents entities files, and they are:

- `titles.ent` where entities for titles are defined. Below you have a sample `titles.ent` file:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!ENTITY book-title "My First Book">
<!ENTITY book-subtitle "First Book Using Borges">
<!ENTITY chapter1-title "First Chapter">
<!ENTITY chapter2-title "Second Chapter">
```

- `strings.ent` where other entities used in `master.top.xml` are defined. Below you have a sample `strings.ent` file:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!ENTITY e-mail "E-mail:">
<!ENTITY web "Web:">
```

2.3.2.5. Generate the Writers' Guidelines

Now that all entities are defined, you can generate the writers' guidelines. Issue

```
make -C manuals/My_Book master.top.pdf LANG=en
```

to do so, and check the resulting PDF with your favorite PDF viewer.

> **Tip:** You can also issue
>
> `make -C manuals/My_Book master.top.flat.html LANG=en`
>
> to build the writers' guidelines in HTML.

If all went fine, you should see the book with the table of contents, all chapters and sections with the guidelines you wrote in it.

2.3.2.6. Assign Tasks to Contributors

By default tasks are assigned to the people declared in the main configuration file (Section 3.1.1.4). You may need to reassign tasks, notably those assigned to `tbn`. Consult Section 3.4.1.3 to learn how to do that. However this step is optional.

2.3.2.7. Write the Modules and Create Images

All that is left is to fill your book with content: write the modules and create the images and/or drawings your book will have. If needed, also new entities file(s) have to be created and filled properly.

So, open the modules' XML files (`modules/en/chap2-sect1-section.xml` for the first section of the second chapter of the English book, for example) with your favorite text editor and start filling it with contents. We won't tell you how to use `DocBook` here, there is excellent material about that all other the Internet. Start consulting The DocBook Wiki[3].

If you use entities in your modules, make sure to create a new entities file to hold the modules' entities (`manuals/My_Book/en/acronym-list.ent` for a file having entities for acronyms in English, for example). Consult Section 3.1.2.1 for more information about entities.

`Borges` also supports images and drawings. At the time of this writing, PNG and JPEG (for raster images), EPS (for vector graphics), `DIA` drawings (`.dia` files) and `XFig` drawings were supported. Consult Section 3.1.2.2 for more information about images.

Images and drawings common to all languages should be put in the `images/` directory and images and drawings particular to each language must be put in the `images/ll/` directory, where `ll` is the two letter ISO code for the language.

2.3.2.8. Check the Result

Finally, you have to check the results. Issue

```
make -C manuals/My_Book master.pdf LANG=en
```

to compile the document into PDF and open it with your favorite PDF reader.

You can also compile the document into HTML both as a single (flat) HTML file or as several (chunked) HTML files. Issuing

```
make -C manuals/My_Book master.html LANG=en
```

will compile the document into chunked HTML files. Point your web browser to `~/My_Project/manuals/My_Book` to check the results. Issuing

```
make -C manuals/My_Book master.flat.html LANG=en
```

will compile into a single HTML file. Point your web browser to `~/My_Project/manuals/My_Book/master.flat.` to check the results.

2.3.3. Final Notes

A few things to note:

- Needless to say, the last two sections of Section 2.3.2 should be done "in a loop". There is no need to write all the modules for your book to check how it is looking so far.
- The `LANG=en` parameter passed to the **make** commands in the above sections is only needed if your preferred language is other than English. This is also needed to compile documents in another language than the one declared in your preferences conf file.

3. http://www.docbook.org/wiki/moin.cgi/

Chapter 3. User's Reference manual

3.1. Documents Writing

Following is a review of the configuration files' format and the required elements on `master.top.xml` for the revision system to work. All the necessary elements to create your documents for your projects, from global entities to documents compilation, are detailed in this section also.

3.1.1. Configuration Files

Following is an in-depth review of `Borges`' configuration files and their format.

3.1.1.1. conf/author.xml

This file holds information related to the author (writer, translator, etc.) who will use the system. Each author *must* define it in order to use the revision system and to be able to compile documents.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
 <author>
  <initials>pp</initials>
  <firstname>Peter</firstname>
  <lastname>Pingus</lastname>
  <mail>peter.pingus@mandrakesoft.com</mail>
  <lang>en</lang>
</author>
```

- `<initials>` holds the author's initials. Most likely two lowercase letters, the initials are used as a *unique* identifier to distinguish among different authors. Mandatory.
- `<firstname>` holds the author's name and `<lastname>` holds the author's lastname. Optional.
- `<mail>` holds the author's e-mail address. Optional, but necessary for email alerts
- `<lang>` holds the author's preferred language. Note that we use the word "preferred" because the language can be overridden with the `LANG=` parameter when doing compilations. However, for revision control the language cannot be forced with the `LANG=` parameter and the one defined in `<lang>` is used. Mandatory.

3.1.1.2. conf/manual-default.xml

This file holds the configuration parameters of the style-sheets to use *by default* when producing PDF and HTML output.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
 <configuration>
  <stylesheet>
   <dssslprint>../drivers/docbook-jadetex.dsssl</dssslprint>
   <xslxhtmlchunk>../drivers/docbook-xhtml-chunk.xsl</xslxhtmlchunk>
   <xslxhtmlflat>../drivers/docbook-xhtml.xsl</xslxhtmlflat>
 </stylesheet>
</configuration>
```

- `<dssslprint>` holds the DSSSL style-sheet used for TeX transformation of the document to prepare it for printing. Mandatory.

- <xslxhtmlchunk> holds the XSL style-sheet used for (X)HTML transformation of the document into *different* (X)HTML files for online publication. Mandatory.
- <xslxhtmlflat> holds the XSL style-sheet used for (X)HTML transformation of the document into *a single* (X)HTML file for online publication. Mandatory.

3.1.1.3. manuals/My_Book/conf.xml

This file holds the configuration parameters of the style-sheets to use when producing PDF and HTML output, as well as "aliases" and exclusion information for deriving various documents from a single super-document. By default this file is the copy of the `conf/manual-default.xml` file above.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<configuration>
 <stylesheet>
  <dssslprint>../drivers/docbook-jadetex.dsssl</dssslprint>
  <xslxhtmlchunk>../drivers/docbook-xhtml-chunk.xsl</xslxhtmlchunk>
  <xslxhtmlflat>../drivers/docbook-xhtml.xsl</xslxhtmlflat>
 </stylesheet>
 <manuals>
  <manual id="Some_Book">
   <lang>en</lang>
   <format>pdf</format>
   <exclude>VPN</exclude>
  </manual>
   <manual id="Another_Book">
   <lang>en</lang>
   <format>pdf</format>
   <exclude>VPN</exclude>
   <exclude>booklet</exclude>
  </manual>
 </manuals>
</configuration>
```

Note: <stylesheet> contains the same parameters as before. Please, refer to Section 3.1.1.2 for more information on it.

- <manual> holds configuration information of exclusions for one document. The `id` attribute *must be unique among all projects*. Optional.
- <exclude> holds the name of the "flags" to exclude in conditional-compilation of the document. Mandatory.

With the sample `manuals/My_Book/conf.xml` above, issuing

```
make -C manuals/My_Book Some_Book.pdf
```

will compile a PDF file named `manuals/My_Book/Some_Book.pdf` excluding all elements marked with `condition="VPN"`; issuing

```
make -C manuals/My_Book Another_Book.pdf
```

will compile a PDF file named `manuals/My_Book/Another_Book.pdf` excluding all elements marked with `condition="VPN"` and also all elements marked with `condition="booklet"`.

Please refer to Section 3.1.2.4 for more information on conditional-compilation.

3.1.1.4. conf/repository.xml

This file is the most important configuration files because it's the top configuration file for the whole `Borges` project.

```
<?xml version='1.0' encoding='iso-8859-1'?>
<configuration>
 <repository>
  <title>My Book Project</title>
  <paths>
   <modules>modules</modules>
   <manuals>manuals</manuals>
  </paths>
  <outputs>
   <makefile>/usr/share/Borges/backend/Makefile.DB</makefile>
  </outputs>
  <manuals>
   <manual>My_Book</manual>
  </manuals>
  <languages>
   <lang>en</lang>
   <lang>fr</lang>
   <lang>es</lang>
  </languages>
  <revisions>
   <type>
    <name>lproof</name>
    <author>tbn</author>
   </type>
   <type>
    <name>ispell</name>
    <author>tbn</author>
   </type>
   <type>
    <name>pproof</name>
    <author>tbn</author>
   </type>
   <type>
    <name>tproof</name>
    <author>tbn</author>
   </type>
   <type>
    <name>translate</name>
    <author>tbn</author>
   </type>
   <type>
    <name>write</name>
    <author>tbn</author>
   </type>
  </revisions>
 </repository>
</configuration>
```

- `<title>` holds the project's name. This is used as the title in the reports generated by `Borges` report facilities. Mandatory.
- `<paths>` holds the paths where the different documents and their modules are located. The defaults are a safe bet, so it is recommended not to change them. Mandatory.
  - `<modules>` holds the modules' "top" directory. Mandatory.
  - `<manuals>` holds the documents' "top" directory. Mandatory.
- `<outputs>` holds the location of the template output `Makefile` files. Template output make files are used for the different DTDs `Borges` supports. At the moment of this writing, only the DocBook DTD was supported (hence, the `.DB` file name extension). You can put as many `<makefile>` entries as you like. Mandatory.

- <manuals> holds the directory name of the different documents, with one <manual> entry per document. There *must* be at least one entry. Mandatory.

> **Caution**
>
> It is highly recommended not to add here any manual by hand. Use the **make adddoc** command instead. See Section 2.3.2.3.

> **Tip:** Even if *GNU/Linux* supports the space character in path names, it is *recommended* not to use them here. You can use the hyphen (-) or the underscore (_) as word separators for path names.

- <languages> holds the languages supported by all documents, one <lang> entry per language containing the two letter ISO code (in lowercase) for that language. At least one language *must* be defined. Mandatory.

> **Caution**
>
> It is highly recommended not to add here any language by hand but the first one. Use the **make addlang** command instead. See Section 3.1.4.

- <revisions> holds the revision types managed by the revision system. The order in which they appear define the work-flow of the document's modules. At the moment of this writing, neither the names of the revision types nor their order can be changed, however the initials of the "default" author responsible for a specific revision can be set. Mandatory.
  - <name> holds the name of the revision: `write`, for module's writing; `translate` for module's translation; `tproof` for module's technical proofreading; `pproof` for module's pedagogical proofreading, `ispell` for module's spell-checking and `lproof` for module's idiomatic proofreading. Mandatory.

> **Note:** Translations of the modules will *only* have `translate`, `ispell` and `lproof` revision types.

  - <author> holds the "default" author initials for a revision type. If no author is assigned to a revision type yet, `tbn` (To Be Named) *must* be used. Mandatory.
- <server> holds the parameters related to remote compilation and/or validation of XML files. This is not yet implemented in `Borges`.

3.1.1.5. master.top.xml and the Revision System

The <revhistory> part of the `master.top.xml` file plays an important role in the revision system of `Borges` .

Below you have a sample <revhistory> part:

```
<revhistory>
 <revision lang="en">
  <revnumber>1</revnumber>
  <date>2002-06-04</date>
  <authorinitials>pp</authorinitials>
  <revremark>First Draft</revremark>
 </revision>
 <revision lang="fr">
  <revnumber>1</revnumber>
  <date>2002-06-14</date>
```

```
  <authorinitials>pt</authorinitials>
  <revremark>Begin French Translation</revremark>
 </revision>
 <revision lang="es">
  <revnumber>1</revnumber>
  <date>2002-06-10</date>
  <authorinitials>rp</authorinitials>
  <revremark>Begin Spanish Translation</revremark>
 </revision>
</revhistory>
```

Each `<revision>` entry contains data related to one of the translations of the document. It has a `lang` attribute with the two letter ISO code (in lowercase) of the language. There *must* be at least one such entry which also has the following:

- `<revnumber>` contains the revision number (or edition number) of the document. Generally, it is an integer value starting at `1` and incrementing for each major revision (or edition) of the document. Optional[1].
- `<date>` contains the date at which work on the corresponding language started. This is used by the report facility of `Borges` to estimate finishing dates for the revisions; in the sample above, work has begun on the French revision on June, the 10th, 2002. The format is `YYYY-MM-DD`. Mandatory.
- `<authorinitials>` contains the initials (unique identifier) of the author responsible for that revision. Optional.
- `<revremark>` contains remarks on the revision itself. This remark is *not* rendered with the default DSSSL style-sheet provided by `Borges` for printed documents, so you'll need to customize the style-sheet if you want the remarks to be printed. Optional.

3.1.2. Document Creation Features

In the following sections the document creation features of `Borges` will be detailed. The sections are not presented in any particular order.

3.1.2.1. Global Entities

Global entities are those entities that you intend to use *without any change at all* among all versions of a project and/or among all your projects. They reside under the `entities/` directory and are XML files with file names ending in `.ent`

Put all entities which *neither* change from one language to another *nor* from one document to another under `entities/`.

Put all entities which *do* change from one language to another, *but do not* change from one document to another under `entities/ll/`, where `ll` is the ISO two letter code (in lowercase) for the language in question.

Good candidates for global entities are:

- Company names;
- Program (software) names;
- Operating Systems names.
- Most acronyms[2].

1. This will be used in future versions of `Borges` for major documents revisions.
2. Acronyms are used "almost" without change among all languages/projects. One that does change, for example, is ISDN which is RDSI in Spanish.

3.1.2.2. Images

Including images in your work is as easy as inserting a `<figure>` element in your modules. For example:

```
<figure>
 <title>An Amazing Figure</title>
 <mediaobject>
  <imageobject>
   <imagedata align="center" fileref="images/image_file_name.png"
              format="PNG"/>
  </imageobject>
 </mediaobject>
</figure>
```

will insert a PNG image contained in the file named `image_file_name.png`, aligned in the center of the page with "An Amazing Figure" (without the quotes) as its title.

Needless to say, `Borges` will take care of finding the `image_file_name.png` file for you in the corresponding `images/ll/` directory, where `ll` is the two letter ISO code (in lowercase) of the language the module will be compiled into.

You can also put language-neutral images under the `images/` directory and `Borges` will get them from there.

Images formats available in your documents are PNG (`format="PNG"`), PDF (`format="PDF"`) and EPS (`format="EPS"`). `Borges` will automatically make them available at the right place for you.

Missing Images

In case that you insert an image in a module and you forget to make the image itself, the system will replace it by a default image, so that the compilation is not broken. The image used by default is `images/missing.jpg` and you can replace it by whatever you want.

Additionally, whenever `Borges` finds a missing image, it will report it in the `<manual>.missing.xx.img` text file. So if you just compiled a document (say `UserGuide`) in French and you note some images are missing (showing the default missing image) you can get the list of missing images by printing `manuals/UserGuide/UserGuide.missing.fr.img`. You can also generate directly that file to check no more images are missing by simply running **make -C manuals/UserGuide/ UserGuide.missing.fr.img**

3.1.2.3. Index Support

DocBook is able to generate an automatic index by collecting all index terms found in the source document. Borges will automatically generate such index provided you request it in the master document. If you want an index to be added at the end of your book, simply end your master.top.xml in:

```
   <index id="index">
     <title>Index </title>
     <para>Automatic Index Here.</para>
   </index>
</book>
```

3.1.2.4. Specialized Books for Different Needs

Often you need to make small variations on your book to fit different audiences, for example a technical manual for a family of products with only small differences among each other.

So, instead of writing different books for the different audiences, it would be desirable to have the possibility to write *one* set of modules for all audiences and have different parts excluded from the different documentation for each audience.

Borges makes this possible thanks to "conditional compilation". Conditional compilation allows you to "mark" some parts of your modules or entire modules in order to exclude them in certain compilations, but not in others.

Let's take an example. You are writing a user manual for the Tortoise operating system running on both Intel and Sparc architectures. There are only minor differences between both guides.

You just need to add the condition="i386" attribute to mark an element (section, paragraph, phrase, note, warning, tip, etc.) as being only valid for the Intel version. Likewise mark elements specific to the Sparc version with condition="sparc". If the element appears to be an entire module, add the attribute in the master file:

```
<sect1 condition="i386">
 <title>Some Title</title>
 <para role="module">some_sect-sect1</para>
 <para>Introduce here the Tortoise OS, highlighting Intel specifications.</para>
</sect1>
```

You then need to tell Borges how to derive both userguides from the Tortoise-UserGuide super-document. This is done in the super document configuration file Section 3.1.1.3. For our example, this file could be:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<configuration>
 <stylesheet>
  <dssslprint>../drivers/docbook-jadetex.dsssl</dssslprint>
  <xslxhtmlchunk>../drivers/docbook-xhtml-chunk.xsl</xslxhtmlchunk>
  <xslxhtmlflat>../drivers/docbook-xhtml.xsl</xslxhtmlflat>
 </stylesheet>
 <manuals>
  <manual>
   <exclude>sparc</exclude>
  </manual>
   <manual id="Tortoise-Sparc">
   <exclude>i386</exclude>
  </manual>
 </manuals>
</configuration>
```

That done, issuing

```
make -C manuals/My_Book Tortoise-i386.pdf
```

will compile the whole book into PDF discarding the elements with `condition="sparc"`.

3.1.2.5. Document Validation

From time to time, it is *recommended* to check your modules are valid XML. Issue

```
make -C manuals/My_Book master.validate
```

to validate your whole super-document for your preferred (working) language.

> **Tip:** Use the `LANG=ll` parameter to validate in a language other than your preferred language. `ll` is the ISO two letter code (in lowercase) for the language you want to validate the document into.

At the moment of this writing the validation process is a parsing of the whole XML code of the document, there are no other validation constraints checked yet.

3.1.2.6. Making Translated Paragraphs Transparent to the Revision System

When translating a module into a foreign language it often happens that the translator wants to add a footnote with translator notes or a few clarification words in a separate paragraph. Also, some licenses (for example the GPL and the GFDL) *require* that you include some portion of it in the original language.

`Borges`' automated revisions management system will report differences among the translated module and the original module only because of those added footnotes/paragraphs, even if they are *correct* or *necessary* in some cases; so, how can you make those paragraphs "invisible" to the revision system?

`Borges` solves this "problem" in an elegant way which does not break DocBook compatibility by using the `revision="-1"` attribute. For example:

```
<para revision="-1">En otro idioma</para>
```

will exclude that paragraph (in Spanish, in the example) when comparing against the original looking for differences in revisions.

This way, you can have those "extra" paragraphs in your translation without worrying about the revisions report being wrong all the time just because of them.

3.1.3. Document modification features

Whenever you modify the structure of a super-document it is necessary to inform the system of such modifications. That will particularly build the module templates for the added modules.

Simply run the **make alltemplates** command.

When this is done, to not forget to add the generated templates to your CVS repository, if any.

3.1.4. Adding new languages to the system

When one of your documents needs to be translated, or simply when you decide that one of the documents will need to be translated, it is time to make the system aware of this new language. This is done in one single command:

```
make addlang LANG=ll
```

that will declare the new `ll`[3] language document for the whole project. It will actually per-

3. `ll` being the two letters ISO code for that lanaguage. See ISO 639[4].

form the following tasks:

- Update `conf/repository.xml`;
- create all language specific directories for modules, images, entities, etc.;
- copy entities files from the default one (first in the list of languages in the main configuration file) to the new language directories;
- Make all module templates for this new language for all defined documents;
- Add all new files to the CVS repository, if available. Note that you'll still need to commit those files by hand;

Once this is done translators will have to

1. Translate entities in `manuals/My_Doc/ll/*.ent`;
2. Translate entities in `entities/*.ent`;
3. Translate modules in `modules/ll/*.xml`;
4. Take snapshots and translate diagrams from `images/xx/` to `images/ll/`, `xx` being another language for which images have already been created.i

3.2. Generating Final Documents

Beyond the simple document generation, many advanced features are available to allow the user easily customizing the output formats or generate a set of manuals in a single command. We will detail all that here.

3.2.1. Single Manual Generation

A final manual (in a user readable format) is simply identified by its name followed by a format extension. Four formats with four extensions are available for `DocBook` document sin `Borges`:

Table 3-1. Borges Output Formats

Format	Extension	Description
PDF	.pdf	The famous Adobe PDF format for printable documents with readers available for all platforms.
HTML	.html	Standard HTML format for online publishing, with chunked output: the document is chunked in many different HTML files. In this case `My_Book.html` designates a directory, not a file, holding all the HTML files composing the document. The entry page is `My_Book.html/index.html`
Flat HTML	.flat.html	One single HTML file for the whole document. Can result very big.
PostScript	.ps	for printable documents.

Knowing that all you need to do is to **make** the desired output. For example if you want to get the document `Install-guide-RPM` from the super-document `Install-guide` in English in PDF format, just run:

```
make -C manuals/Install-guide/ Install-guide-RPM.pdf LANG=en
```

3.2.2. Generating Multiple Documents at Once

When one needs to publish all the manuals available in all language for his project, compiling them one after the other in all formats can result harassing. For this reason `Borges` provides a target to automatically compile any combination of manual-language-format.

The synopsis of this command is:

```
make all SUBDOCS="<docs list>" LANGS=""<languages list>" FORMATS=""<formats list>"
```

where:

docs list

: is the list of super-document/document pairs you wish to generate. If you wish to get the manuals `Install-guide-RPM` and `Install-guide-tar` from super-document `Install-guide`, you'll have to use `SUBDOCS="Install-guide/Install-guide-RPM Install-guide/Instal`

languages list

: the list of languages to get the manuals in. use `LANGS="en fr es"` to get all manuals in English, French and Spanish.

formats list

: the list of extensions to compiles the manuals in. If you are interested in PDF and flat HTML output, use `FORMATS="pdf flat.html"`

With this example we would end up with the following command line:

```
make all SUBDOCS="Install-guide/Install-guide-RPM Install-guide/Install-guide-tar" \
LANGS="en fr es" FORMATS="pdf flat.html"
```

Which will result in 2*3*2=12 manuals in `Outputs/`.

3.2.3. Generating a Single Module

When you are working on writing and/or translating a module, you often want to have a look at it in one of the supported output formats. `Borges`' single module compilation feature allows you to do so *without* the need to compile the whole document containing the module in question, thus leaving you more time to do your work instead of waiting long book compilation times.

The command synopsis for compiling a single module is:

```
make -C manuals/module <module_name>.<output_format> [LANG=ll]
```

Note that the directory for single module compilation is always `manuals/module` regardless of which document the module belongs to. This directory is automatically created when `Borges` is initialized. All single module compilation output goes into it.

The `LANG=ll` parameter is optional and it is used to force compilation to occur in a language other than the default one. `ll` is the two letter lowercase ISO code of the language.

For example, after issuing:

```
make -C manuals/module borges-compile-features-sect1.pdf LANG=es
```

you will end up with the PDF file `manuals/module/borges-compile-features-sect1.pdf` with the contents of the `borges-compile-features-sect1` module in Spanish.

3.3. Output Style Customizations

With `Borges` it is very easy to control the way final documents are formatted thanks to `DocBook` customization features. Moreaver it is easy to create new ciustomization layers so that each manual can have its own design.

3.3.1. Customizing Existing Formats

As we already seen in Section 3.1.1.2, te customization layers for all output formats are located in `drivers/` directory. You just need to open the stylesheet corresponding to the format you want to change with your text editor:

`drivers/docbook-jadetex.dsssl`

for PDF and PS formats outputs;

`drivers/docbook-xhtml.xsl`

for flat HTML output format;

`drivers/docbook-xhtml-chunk.xsl`

for chunked HTML output format.

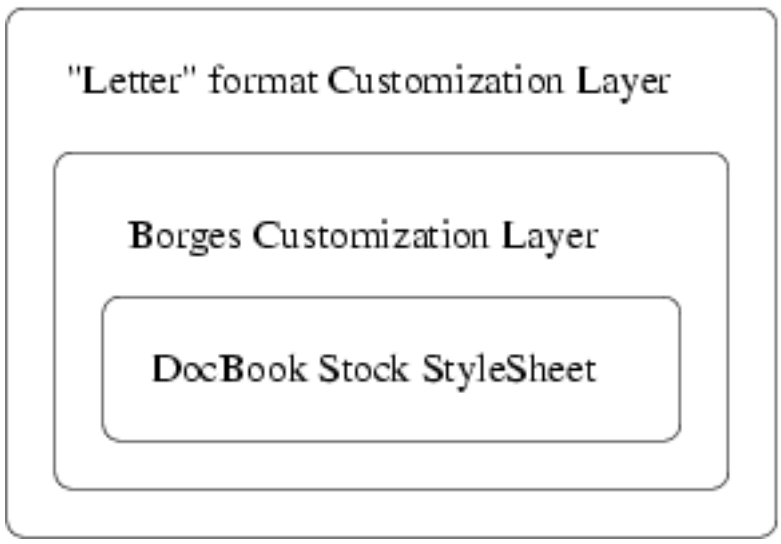
Consult the documentation on how to customize XSL[5] and DSSSL[6] stylesheets if needed.

3.3.2. Creating a New Customization Layer

Having one customization layer per output format might not be enough for some special needs. Let's imagine that there is a manual you want to publish in Europe and in the United States. Therefore you need it in two different paper formats: `A4` and `Letter`. This is done in two simple steps:

1. Create a new customization layer

   This customization layer will be placed on top of `Borges` print customization layer, resulting in the following layers:



   Our new customization layer (`drivers/docbook-jadetex-Letter.dssssl`) would look like:

```
<!DOCTYPE style-sheet PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN" [<!ENTITY do

<style-sheet>
 <style-specification id="print" use="docbook-jadetex">
```

5. http://www.docbook.org/wiki/moin.cgi/DocBookXslStylesheetDocs
6. http://www.docbook.org/wiki/moin.cgi/DocBookDssslStylesheetDocs

```
        <style-specification-body>

        ;;What size paper do you need? A4, A5, USletter or USlandscape?
        (define %paper-type% "USletter")

        </style-specification-body>
       </style-specification>
       <external-specification id="docbook-jadetex" document="docbook-jadetex.dsssl">
      </style-sheet>
```

Now that the customization layer is ready we just need to direct the system to use it in the second step.

2. The default `Borges` print stylesheet uses `A4` paper format. We then need to create a new manual that will use the "Letter" customization layer we just created. This is done in the super document configuration file, for example `manuals/Install-guide/conf.xml`:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
  <stylesheet>
    <dssslprint>../../drivers/docbook-jadetex.dsssl</dssslprint>
    <xslxhtmlflat>../../drivers/docbook-xhtml.xsl</xslxhtmlflat>
    <xslxhtmlchunk>../../drivers/docbook-xhtml-chunk.xsl</xslxhtmlchunk>
  </stylesheet>

 <manuals>
  <manual id="Install-guide-A4">
    <lang>en</lang>
    <format>pdf</format>
  </manual>
  <manual id="Install-guide-Letter">
    <lang>en</lang>
    <format>pdf</format>
    <stylesheet>
      <dssslprint>../../drivers/docbook-jadetex-Letter.dsssl</dssslprint>
    </stylesheet>
  </manual>
 </manuals>
</configuration>
```

In this file, the first `stylesheet` element informs the system that we want to use the `Borges` stylesheets per default. Therefore, the `Install-guide-A4` manual will use `docbook-jadetex.dsssl` with `A4` paper format. However for manual `Install-guide-Letter` we specify that we want to use our customization layer `docbook-jadetex-Letter.dsssl`. The other formats (HTML) will still use the default stylesheets as we did not redefine them.

Once this is done, you can use the Section 3.2.2 feature to generate at once the two different books `Install-guide-A4.pdf` and `Install-guide-Letter.pdf` respectively in `A4` and `Letter` paper formats.

3.4. Revision Management

Revision management is the most interesting feature of `Borges`. It tracks documents on two axes:

1.

Modules status (Section 3.4.1)

Each independant piece of document called a "module" has its own life cycle in `Borges`. Granular module management ensures quality standards while allowing to generate accurate project tracking information.

2.

Translations Freshness (Section 3.4.2)

Translating a document is something common. Updating translation from an ever-changing original is often a nightmare. Thanks to the innovative system brought by `Borges`, it is possible to know whenever a translation needs to be updated, and where exactly the changes are located. This system also ensures that the structure of the document is respected throughout its various translations, while allowing translators to explicitly make additions with respect to the original structure.

Thanks to these tools, hierarchical reports can be generated giving relevant project tracking information for everyone project managers as well as modules authors, translators, contributors.

3.4.1. Modules Life Cycle

3.4.1.1. The underlying philosophy

A module is born when it is referenced for the first time in a master super-document. It reaches its maturity when it passes final language proofreading. Between those two steps it is necessary to bring a module to each one of these steps:

1.

Written

When the redactor has finished writing an original module.

2.

Translated

When a translator has finished the translation of an original module in his language.

3.

Technical proofreading

When a technical expert has read a module, and his remarks have been incorporated.

4.

Pedagogical proofreading

When an education specialist has read a module, and his remarks have been incorporated.

5.

Spell checking

When the module has successfully passed a spell checker.

6.

Language proofreading

When a skilled native speaker has read a module, and his remarks have been incorporated.

After a module has reached this last step, he is regarded as mature and ready for publication. Below is a diagram that better explains the process for an original module and one of its translations.

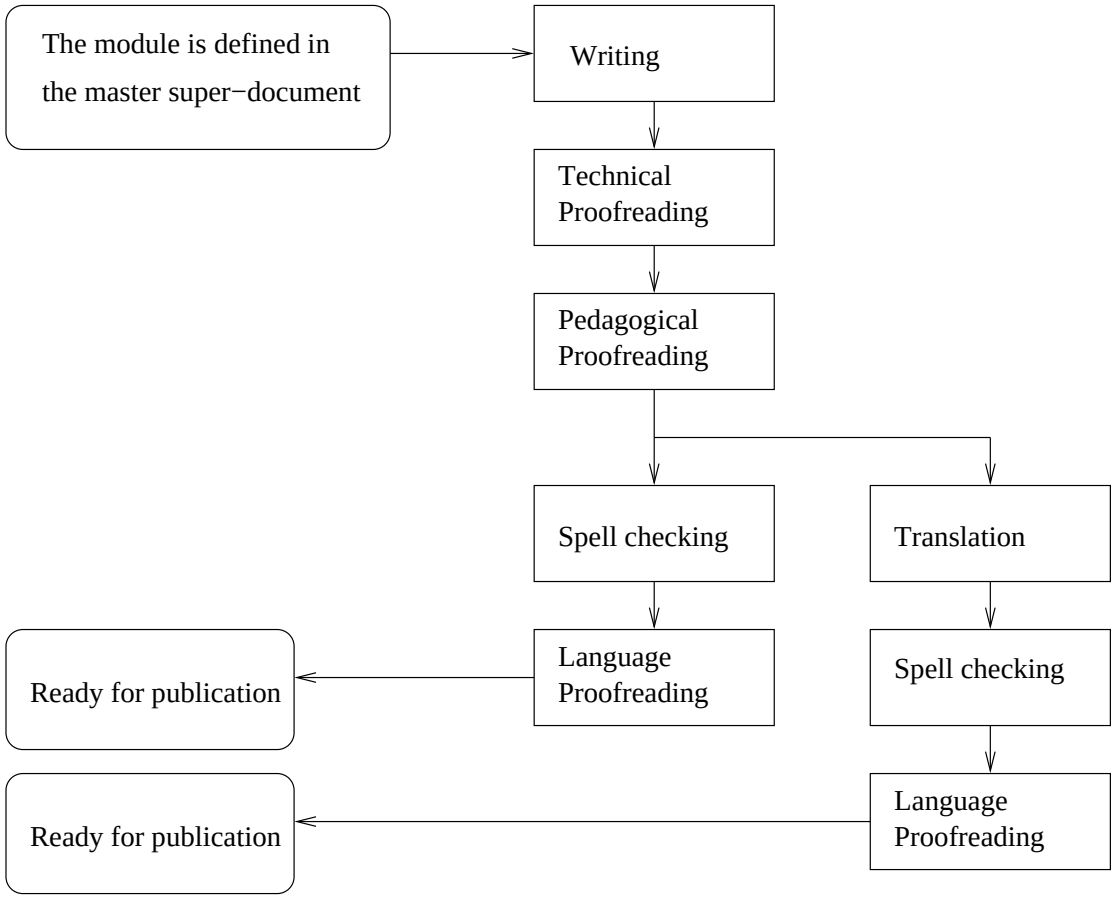


Figure 3-1. Borges' Modules Life Cycle

3.4.1.2. Modules status in practice

The module history is stored directly in the revision history of the module's root element. However it *must not* be edited by hand. For this purpose there are **make** targets that will take care of all the tasks associated with a module's status.

This is the synopsis of the command that will perform the work associated to a task:

```
make <module-name>.revision TYPE=<type>
```

where <type> is one of the following, corresponding to the steps we described just above:

1. `write`
2. `translate`
3. `tproof`
4. `pproof`
5. `ispell`
6. `lproof`

Let's imagine you just modified the module `intro-section.xml` according to the remarks the pedagogical proofreader gave you. The command to issue in the directory owning the file will be:

```
make intro-section.revision TYPE=pproof
```

3.4.1.3. Assigning tasks

In order to get full advantage of `Borges` features, it is recommended to assign all tasks in all languages for all modules. That will ensure no task is left orphan, improving project efficiency.

This operation is achieve with the following command:

```
make <module-name>.revision TYPE=<type>.todo AUTHOR=<ai>
```

where `<ai>` are the initials of the author responsible for that operation. For example if author `pp` is responsible for performing language proofreading on module `intro-section` in spanish, you would issue:

```
make -C modules/es/ intro-section.revision TYPE=pproof.todo AUTHOR=pp
```

3.4.2. Inter-Languages Modules Synchronization

3.4.2.1. The Idea Behind Atom Revisions

To ensure a translation remains up to date with respect to its original, it is necessary to track changes in the original. To track changes in a text, there is basically one and only method: generate the differences between two versions. However this has a big drawback: when the changes are not relevant for the translator (spell and syntactic changes by opposition to semantic changes), one has to extract pertinent bits in a sea of irrelevant information.

To make sure relevant changes are explicitly marked as such, human action is needed. Therefore the redactor changing the meaning of a paragraph will have to explicitly mark that paragraph as modified by incrementing its associated revision attribute. Then the system will be able to spot atoms through a translated module that are not up to date and warn the translator.

3.4.2.2. Authors Duties

The whole system relies on authors good will. If they are contentious at adding revisions all will be fine. Experience has proved that it's easy to make authors aware of the problem, and then all works perfectly.

When a module passes the `pproof` step, it gets ready for translation. The system then automatically adds IDs to any possible atom[7] in that module. Let's follow a specific paragraph of the `passwords` module:

1. After the module has passed the `pproof` step, our paragraph got an automatic ID:

   ```
   <para>
   <screen> root# head -c 6 /dev/urandom | mimencode</screen>
   This will print five random characters on the console, suitable for
   password generation. You can find <command>mimencode</command> in the
   <filename>metamailer</filename> package.</para>
   ```

2. Despite technical proofreading a reader has spotted an error: it should read six and not five random characters. You then correct the error and add a revision ID:

   ```
   <para revision="1">
   <screen> root# head -c 6 /dev/urandom | mimencode</screen>
   This will print six random characters on the console, suitable for
   password generation. You can find <command>mimencode</command> in the
   <filename>metamailer</filename> package.</para>
   ```

7. To get the list of elements that become an atom, consult `/usr/share/Borges/bin/scatter_ids.pl`

3. Later on, you realize there is a mistake in the package name, it is not `metamailer` but `metamail`. Even though the `filename` element is not a default atom, you can assign it an ID and a revision attribute:

   ```
   <para revision="1">
   <screen> root# head -c 6 /dev/urandom | mimencode</screen>
   This will print six random characters on the console, suitable for
   password generation. You can find <command>mimencode</command> in the
   <filename id="metamail-pack" revision="1">metamail</filename> package.</para>
   ```

   This is better than increasing the paragraph revision to 2, as the translator will directly spot the change in the `filename` element without having to search through the whole paragraph.

Tip: If you wish to help translators spot a little change in a big chunk of text, it is better to enclose the modified sentence in the `phrase` element, adding the ID and revision to that reduced element...

Warning

It often happens that an author is forced to modify the structure of a module, even after it has gone to translation. In that case, it may become necessary to assign IDs to possible new elements. The author can choose to assign them manually (ensuring there are no risk of duplicate IDs) or to let the system reassign all IDs throughout the module if there has been many changes. This is made thanks to the following command:

```
make <module-name>.id
```

Obviously, this command will also have to be run on translated modules...

3.4.2.3. How Translators Synchronize Modules

We won't speak about reports generation here, but rather how to read reports and what to do according to the information contained in the reports.

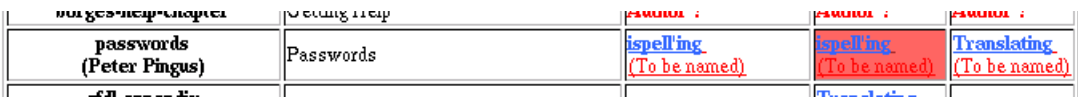


Figure 3-2. An extract of a super-document report

Whenever a translated module becomes obsolete with respect to the original, the corresponding cell in the super-document report table becomes red (Figure 3-2). If you click in the cell, you then get to the module's detailed report (Figure 3-3).

Stats for passwords in fr

Task	Finished on	Author
1.fr.lproof	Pending	(To be named)
1.fr.ispell	Pending	(To be named)
1.fr.translate	2002-06-25	(Peter Pingus)

Changes in IDs/revisions
Side by side not synched elements

Figure 3-3. A Sample Modules' report

In that report, after the revision history table, two links appear:

- Figure 3-4: spots the atoms that differs between the translation and its original;
- Figure 3-5: presents the original and translated atoms that are out of synch side by side.

Modified lines: 2, 5
Added line: None
Removed line: 7

*Generated by **diff2html***
© Yves Bailly, MandrakeSoft S.A. 2001
***diff2html** is licensed under the GNU GPL.*

passwords.ids.xml
8 lines
339 bytes
Last modified : Tue Jun 25 12:21:54 2002

passwords.src-ids.xml
7 lines
294 bytes
Last modified : Tue Jun 25 12:21:54 2002

	passwords.ids.xml		passwords.src-ids.xml
2	<revisions file="/home/pp/doc/modules/en/passwords.xml">	2	<revisions file="/home/pp/doc/modules/fr/passwords.xml">
5	<element id="passwords-pa2" revision="1"/>	5	<element id="passwords-pa2" revision="0"/>
7	<element id="metamail-pack" revision="1"/>		

*Generated by **diff2html** on Tue Jun 25 12:21:54 2002*
Command-line: `/opt/Borges//bin/diff2html --only-changes passwords.ids.xml passwords.src-ids.xml`

Figure 3-4. Changes in IDs/revisions

In this table, the atoms that have been modified in the original clearly appear. The author knows that the element with ID `passwords-pa2` has been modified, while a new element `metamail-pack` has been added.

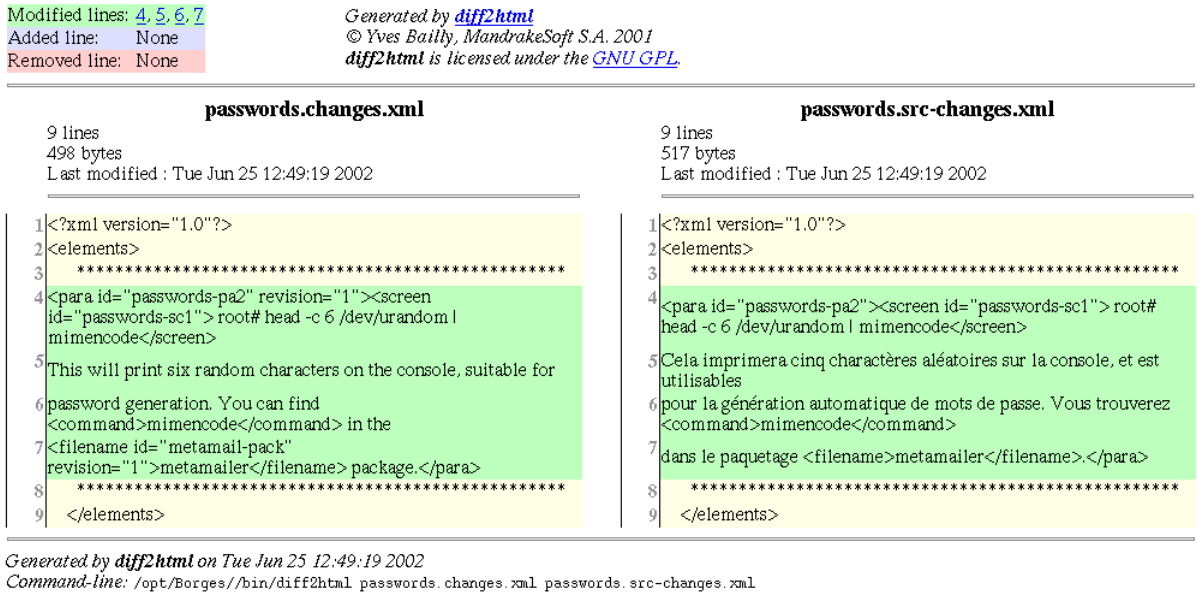


Figure 3-5. Side by side not synched elements

Through this page the translator can open the `modules/fr/passwords.xml` file, search the element `passwords-pa2` and synchronize its content according to the text in the HTML report. Of course the new ID and revision attributes will have to be copied too, so that the system can know the atom has been updated.

3.4.3. Generating Reports

Here is a diagram showing the different HTMLreports generated by `Borges`, and the navigation through them.

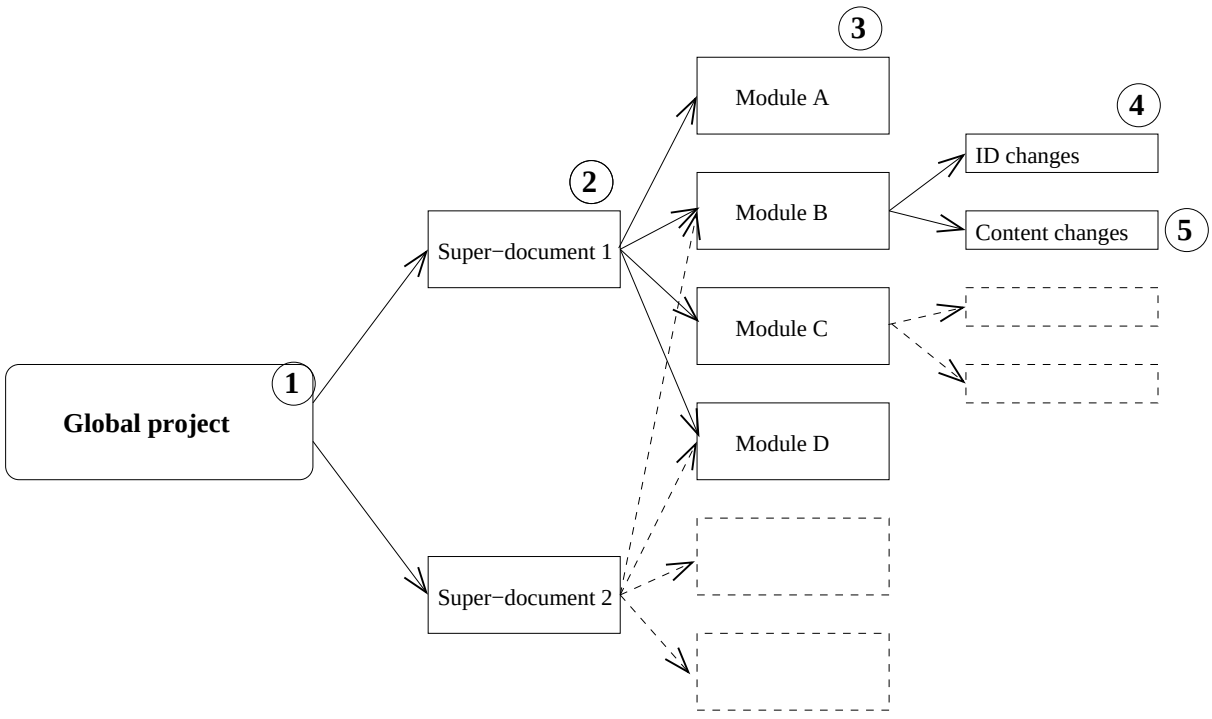


Figure 3-6. The reports generated by Borges

We will now see how to generate each one of those reports and how to read them.

> **Warning**
>
> It is necessary that all the sources in all languages are valid so that the reports get generated correctly.

3.4.3.1. Global Project Report

Generating this report will in fact generate all other reports so that it is possible to consult them starting from the global project report page. You simply need to issue the following command (in the `reports/` directory of your project):

```
make all
```

That will generate the `index.html` file and you will just have to point your browser to that file. For an example of such a report, consult the Overall report for Borges Manuals[8].

The resulting page is self-documented so we won't detail it here. You will however note that, during the compilation process, all rough super-documents have also been generated (see the "Links to the compiled versions of the manuals"). If you wish to only generate the reports without the documents, run

```
make index.html
```

> **Note:** This feature is particularly useful for project managers willing to regularly (through cron jobs) publish on a website the current project status. It is enough to upload the content of the `reports/` directory on a website and point people to it.

3.4.3.2. Super-Document Report

If you wish to generate only the report for a super document (and all dependent reports), run:

```
make master-report.html
```

in the directory of the super-document (e.g. `manuals/My_Doc/`). You can then open `master-report.html` with your favorite browser. For an example of such a report, consult the Detailed report status for Borges-doc[9].

The table in that report has one line per module of the corresponding super-document. There are at least three columns:

1. The name of the module, followed by the name of the original author of that module;
2. The title of the module;
3. Three possibilities for the content of that cell:
   - The task in progress for that module in the language corresponding to that column. If known the person responsible for that task is shown in parenthesis.
   - If no task is available for now the text "Pending" is shown.
   - "OK" means that all tasks required on that module have been passed.

   If the cell appear on a red background, that means this translation is outdated with respect to the original module. See Section 3.4.2.3.

8. http://www.linux-mandrake.com/en/doc/project/Borges/reports/
9. http://www.linux-mandrake.com/en/doc/project/Borges/reports/Borges-doc/master-report.html

A click on the text of that cell will lead to the corresponding Section 3.4.3.3.

3.4.3.3. Module report

There is no need to generate one specific module report, all module reports related to a specific super-document are generated while making the super-document report. This page simply holds the revision history for the module, plus possible links to detailed diff reports if that module happens to be out of synch (see Section 3.4.2.3).

3.4.3.4. ID Changes Report

When a translator is in the process of updating a module, it may be interesting to quickly regenerate the IDs report to check everything is fine. The command to issue is:

```
make <module-name>.ids.html LANG=<xx> manual=<super-document>
```

for example to get the IDs report of the `passwords` module in French as part of the `Borges-doc` super-document, you will need to go into `modules/fr/` and run:

```
make passwords.ids.html LANG=fr manual=Borges-doc
```

You then just need to open `passwords.ids.html` with your browser.

3.4.3.5. Content Changes Report

Same as above, but then the command becomes:

```
make <module-name>.changes.html LANG=<xx> manual=<super-document>
```

Chapter 4. Features for the Project Manager

The previous chapter was dedicated to the working masses. We will now concentrate on a few `Borges` features to assist the projet manager.

The project reports (Section 3.4.3.1) are of great help. But there are additional tools to remind authors of their current tasks (Section 4.1), and to evaluate the work made by each author (Section 4.2).

4.1. Sending Mails to Authors

This very useful feature allows to prepare mails for every authors implied in the project. Those mails will list all tasks the author should be working on currently.

> **Note:** This feature is particularly useful for project managers willing to regularly (through cron jobs) send updated todo lists to their team..

To generate the mails, run:

```
make
mails
```

in the `reports/` directory. That will generate files for each author whom has pending tasks. The filename is the author's email address, and the content is the body of the message. You can then dispatch the mails with the command:

```
for f in $(find . -name \*@\* -exec basename {} \;);
  do cat $f | mail $f -s "Your current tasks list for manuals";
done
```

> **Tip:** You can put additionnal information at the end of the mail, simply by writing your footer in the file `conf/mailfooter.txt`. You can put there information related to the place where to find the modules by CVS, WebCVS, the URL for the compiled version of the documents, etc.

4.2. Accounting Report

This special report is made of tables for each manual and for the the whole project that summarizes the authors contributions for each modules and for each manual.

To generate these reports, simply run **make -C reports/ accounting.html**. Then point your browser to the resulting file `reports/accounting.html`. The table gives all the project's super-documents in column, with the respective contribution of each authors on each line. There are totals on each line for each author, and totals for each manuals, plus a grand total for the whole project on the bottom right corner.

Additionally, you will find under `reports/<Manual-Name>/costs.html` some more details reports per manual which list authors contributions to each module. You can also generate those manuals specific reports by running **make -C manuals/<Manual-Name>/ costs.html**

Now some details on the way those casts are calculated.

The script scans all modules and calculates each contribution with the following formula:

```
C=N*P*W/10
```

```
C=task Cost
N=number of text characters in the module
P=price per translated character
W=task weight
```

the `P` and `W` parameters are defined in `conf/repository.xml`. You should adjust them to your needs.

Chapter 5. Borges and XML Editors (Emacs Rules)

borges-editors-chapter

In introduction tell that any editor should be OK provided it does not interfere with Borges requirements.

Then add a section for Emacs+PSGML which Borges fully supports..

Chapter 6. Borges and CVS Integration

borges-cvs-chapter

Tell that Borges is designed to work fine on a CVS. Tell what to put on CVS and what not. Warn about the problems caused by module templates automatically generated that must be added to the CVS at the same time the modified master.top.xml is updated. Same when adding new languages/manuals.

Chapter 7. Programer's Reference manual

We will get here into Borges internals. This may be of interest for the developer as well as for the user wishing to take advantage of the most advanced feaures of Borges.

If something is not clear enough below, or if you wish to know more, use the source luke. If there's something you definitely not understand, ask on Borges mailing list.

7.1. Makefiles

We will list here the different Makefiles available in Borges source repository and in the implemented repositories[1]. We will detail the way those Makefiles are generated, distributed, etc.

7.1.1. Borges source Makefile

There's only one usable Makefile here. You'll find it at the root of the repository.

This Makefile has two main targets:

doc

compiles the Borges documentation and reports;

install

installs Borges on a system so that users of that system can start documentation projects on it, using Borges. It installs all the scripts and Makefile's and build a repository template so that users can quickly start using Borges.

> **Tip:** Borges gets instralled in /usr/share by default (/usr/share/Borges/). You can change that by passing the *TARGET* parameter to **make**. For example if you wish to relocate Borges to /home/joe/test/Borges/ just run **make install TARGET=/home/joe/test**

you may have noticed that Borges does not need compilation. Indeed all scripts are in perl or bash and do not need compilation.

7.1.2. Documentatin Projects Makefiles

7.1.2.1. What Goes Where

The diagram below shows how the different Makefiles provided by Borges are distributed in the implementation repository.

7.1.2.2. Who Calls Who

The following diagram shows how the Makefiles found in the Borges source repository (on the left) gets distributed into an imaginary project (on the right) with two books Book1 and Book2 in two languages en and fr

1. It is important to distinguish between the Borges source repository, which is the repository holding all the Borges code maintained by its developers; and a simple implementation of Borges, which is a documentation project repository, holding the documentaion source files managed by Borges.

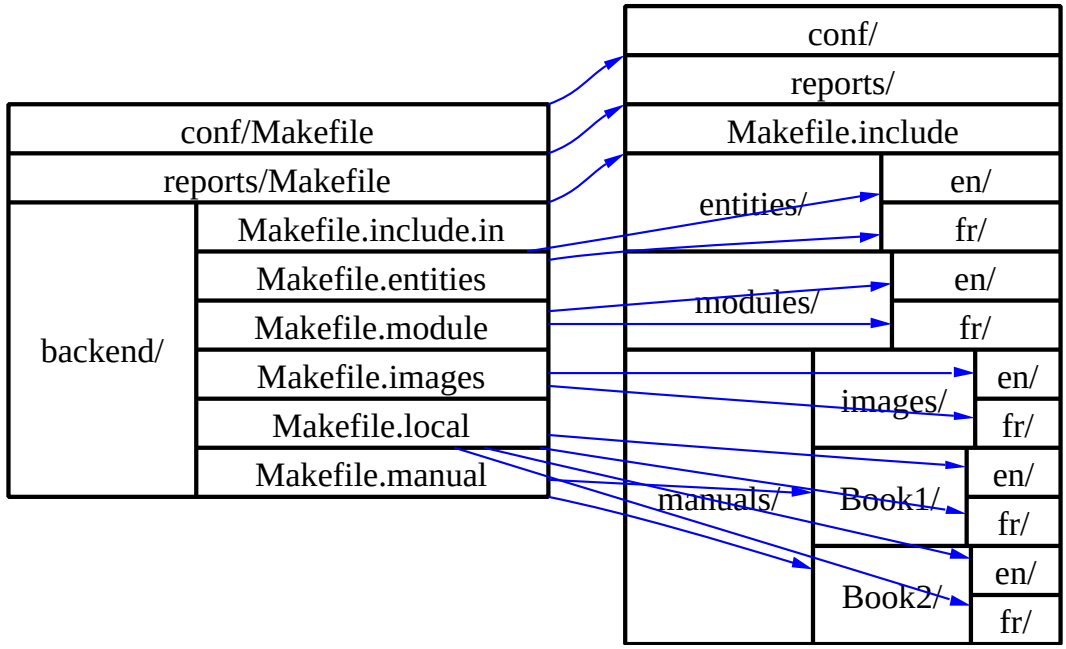


Figure 7-1. Distributing Makefiles

7.1.3. Makefiles in Action

We will show here how Makefiles are linked together. Figure 7-2 shows how Makefiles include each others. An arrow in the diagram means "includes".

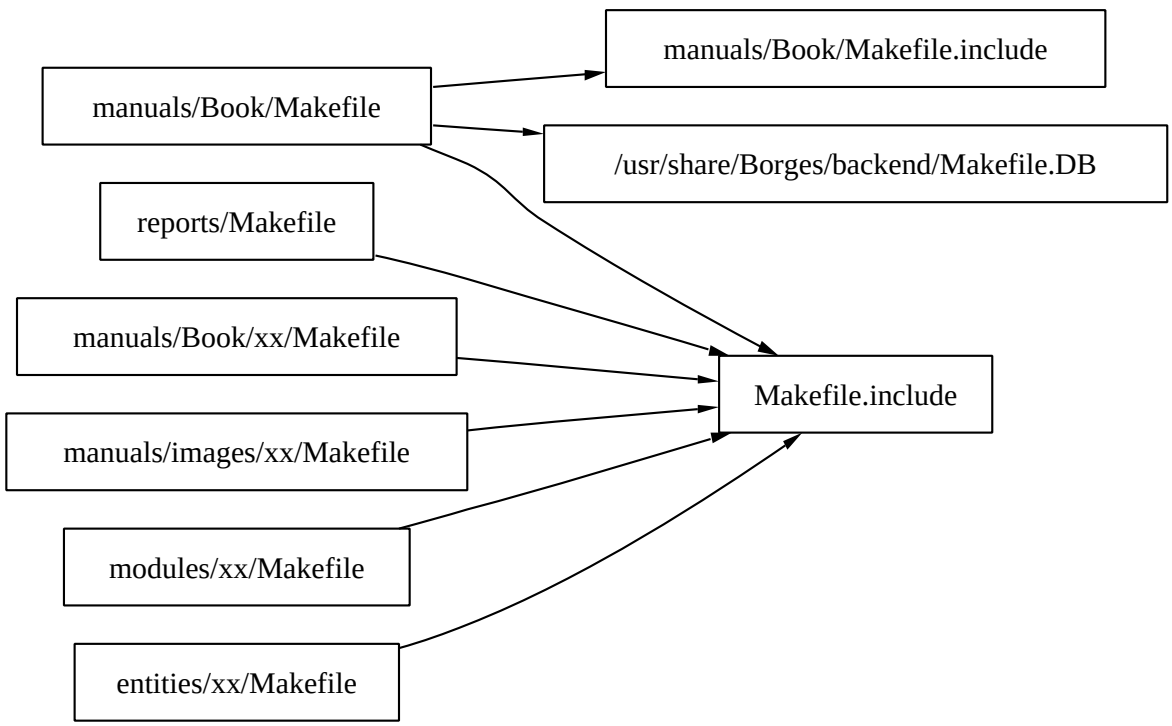


Figure 7-2. Makefiles Relatiunships

All paths are relative to the project root dir unless otherwise stated.

We can distinguish between four types:

Production

The Makefiles on the left are the ones actually used to perform tasks on manuals, modules, images, etc.

manuals/Book/Makefile.include

This Makefile is empty by default. It can be used by advanced users to redefine default manual compilation rules. See Section 7.3 for more details.

Makefile.DB

This Makefile contains the rules to actually transform source XML DocBook files to any desired output format (PDF, HTML, etc.). Advanced users may choose to develop their own `Makefile.XXX` to support another DTD. See Section 7.4 for more details on how to do that.

To determine which Makefile is used to generate output formats, the system looks for the <makefile> element(s) in `conf/repository.xml` and sets the ***`OUTPUTS`*** variable accordingly in the root `Makefile.include` described below.

Makefile.include

This Makefile is automatically generated by the root Makefile. It contains useful information for all other Makefiles, extracted from the environment and notably from the naim configuration file `conf/reposqitory.xml`. It contains also some generic functions and rules.

7.2. The Way a Manual is Generated

To understand the process leading to the generation of a final document, we'll detail here the steps taken to generate an HTML file.

In Figure 7-3 we represent the way from the `master.top.xml` skeleton guideleines to the final HTML book.

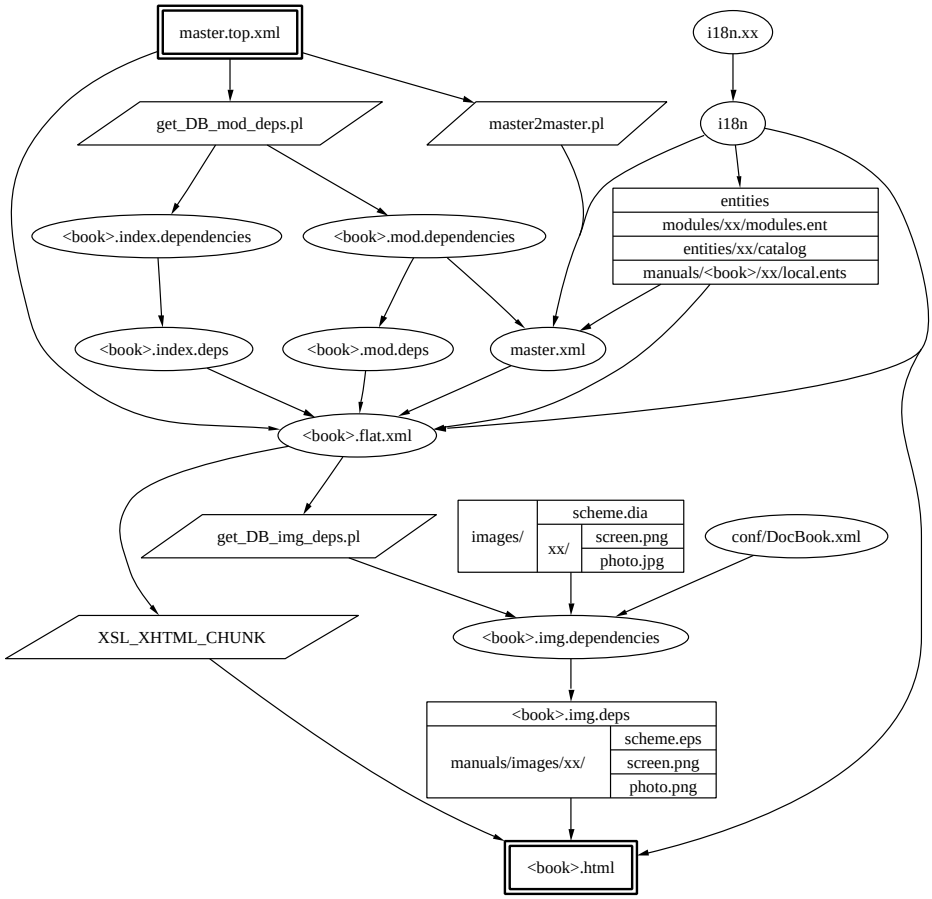


Figure 7-3. Distributing Makefiles

We can distinguish two main steps:

1.

Generation of the `<book>.flat.xml` source file

This file contains all the XML source code necessary to compile the document. It is "flat" because all modules and entities have been expanded into it. To do so it's been necessary to:

- Compute a possible index,
- Check all needed modules are available.
- Catalog all entities.

2.

Actual HTML Compilation

This includes the generation of all necessary images which names are extracted from `<book>.flat.xml`.

7.3. Adding/changing Manuals Rules

It may happen that the rules provided to prepare the `master.flat.xml` are not suited for a particular need. Or the user may want to override some rules for generating output formats.

`Borges` provides a mean to do that easily. One just need to write its custom or tweaked rule in `manuals/<Book>/Makefile.include`. That will add extra functionnality for generating output formats or overwrite default rules.

7.4. Supporting Another DTD than DocBook

To Be Written

Chapter 8. Getting Help

Do not forget to consult the `Borges` Web pages[1].

8.1. Bug Reports, Feature Requests, Patches

Visit the Borges pages on SourceForge[2]. You will be able there to:

- Post bugreports: When ever you think you have discovered a bug in `Borges`, post a detailed bug report here;
- Ask for support: If you are stuck with a problem you cannot find the answer in the documentation, post a support request there;
- Submit patches: You've come with modifications in the source code to fix bugs or ad features to `Borges`? You can submit the patches here.
- Ask for new features: If you wish to see more functionnalities added to `Borges`, propose them here with detailed argumentation.

8.2. Contact

A mailing list is available, simply send a message to the list manager[3] with the command **subscribe borges** in the body. You can also contact `Borges` maintainer[4].

You may also try to see if there are people on the `#borges` IRC channel at

irc.mandrakesoft.com

.

1. http://www.linux-mandrake.com/en/doc/project/Borges/
2. https://sourceforge.net/projects/borges-dms/
3. sympa@moondrake.com
4. documentation@mandrakesoft.com

Chapter 9. Sample Module for Tests

passwords introduction

```
root# head -c 6 /dev/urandom | mimencode
```

This will print six random characters on the console, suitable for password generation. You can find **mimencode** in the `metamailer` package.

Appendix A. Borges Commands Reminder

command-reminder-appendix

This appendix will list all available make commands under Borges, sorted by topic: compilation, revision management, reports generation, etc.

Appendix B. GNU Free Documentation License

B.1. GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

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

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

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

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

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

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

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

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

2. VERBATIM COPYING

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

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

3. COPYING IN QUANTITY

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

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

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

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

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

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

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

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

D. Preserve all the copyright notices of the Document.

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

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

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

H. Include an unaltered copy of this License.

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

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

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

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

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

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

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

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

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

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

5. COMBINING DOCUMENTS

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

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

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

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

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

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

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

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See Copyleft[1] .

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

B.2. How to use this License for your documents

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

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

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

1. http://www.gnu.org/copyleft/