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

John Rye
jrt746@clear.net.nz

Borges DMS: Self-Documentation
Published 2004-04-19

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

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 document 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 you to easily tweak 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 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 is currently tighty-linked with the DocBook DTD.

Currently, the system allows you to:

- Compile the source files into PDF, PS and (X)HTML whether they be complete documents or just modules (component parts of documents);
- Compile multilingual documents. This allows you to have, for example, a booklet in 2, 3 or more languages.
- Manage different versions of a single document by easily defining derived versions based on conditional parts;
- Assign work to the project's participants. Each module is assigned a set of authors: writers, translators and proofreaders; each one responsible for one state of a module. Each contributor can easily review his attributions through web pages, and receive e-mails with his currently assigned tasks;
- Track the work in progress. From the whole project (made of various documents) to the most basic components (paragraphs), and their translations. Notify the project's administrator by e-mail about the work not yet assigned;
- Define the workflow of the modules. The workflow is the set of states a module goes through during its "life". Different projects might have different workflow needs, and different enterprises have different workflows. Usually, the workflow begins with the "write" task and finishes with a "lang_proofread" task.
- Track the state of each module according to the workflow. Once a task is completed, the corresponding state is passed and the module switches to the next state;
- Perform work on modules through a web frontend. Project participants can work on modules (write, translate, proofread) using a web frontend which greatly simplifies participation of "remote" authors in a project allowing them to work without the need to have `Borges` installed on their computers.

Quick facts about `Borges`:

- Automatic management of images in EPS, PNG, JPEG, `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 multiple 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 state 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 operating systems other 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 into 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 a new generation of technical documentation with XML and DocBook.

In short, Borges will provide you with a solution to efficiently manage large 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 may 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 elements are `part` or `chapter`.

See Also: Super-document, Module element.

Module element

In a super-document, a module element is a DocBook element which 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 elements 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 smallest 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 the `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 of 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 successes or failures on any other systems.

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 do I install it?

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

> **Note:** Borges installs by default in /usr/share/Borges/. If that doesn't suits you, please see Section 7.5.1.

2.1.3. Dependencies

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

- make
- libxslt-proc;
- perl;
- perl-XML-Twig, perl-DateManip and perl-XML-LibXML libraries;
- ImageMagick images processor for images transformations;
- xfig diagrams editor if you wish to work with xfig diagrams;
- DocBook DTD XML version 4.2 into /usr/share/sgml/docbook/xml-dtd-4.2/;
- DocBook DSSSL stylesheets into /usr/share/sgml/docbook/dsssl-stylesheets/;
- DocBook XSL stylesheets into /usr/share/sgml/docbook/xsl-stylesheets/;
- openjade
- tetex-latex
- jadetex

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

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

2.2. First Steps

Borges provides an easy procedure to start with a new documentation project. We will detail the configuration steps necessary to create a project template. Afterwards, the sample document provided with Borges will be compiled into both PDF and HTML and the progress report will be generated.

To start with a new Borges repository, you need to perform the following steps:

1. Create the project skeleton

   Borges provides a simple script to create a new project skeleton. Let's assume you want to put your files under My_Project in your home directory, then you would issue:

   ```
   /usr/share/Borges/bin/configure ~/My_Project
   ```
   to do so.

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

2. Initialize the System with the Provided Sample

   Now, Borges has to be initialized with a new document to work with. To do so with the provided sample, just issue:

   ```
   make adddoc doc=My_Book master=/usr/share/Borges/Sample/master.top.xml
   ```
   and directories will be populated with the minimum required files.

3. Compile My_Book to PDF and Check the Result

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

   ```
   make -C manuals/My_Book My_Book.pdf LANG=en
   ```
   to do so, and check the resulting PDF by issuing

   ```
   xpdf manuals/My_Book/My_Book.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.

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

4. Compile My_Book to HTML and Check the Result

   You can also compile the sample document to HTML. Issue

   ```
   make -C manuals/My_Book My_Book.flat.html LANG=en
   ```
   to do so, and then check the results by pointing your favorite browser to ~/My_Project/manuals/My_Book/My_

5. 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.

It was not that hard was 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

First, you should create a new project skeleton based on the provided template as described in Section 2.2:

```
$ /usr/share/Borges/bin/configure ~/New_Project en 2.1
      $ cd ~/New_Project
```

> **Note:** You should replace `en` by the language code you wish to use by default on your system, if it's not English. Likewise `2.1` is the first release number of the documentation you are about to start working on. `1.0` will be used if nothing is provided.

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

1. Declare the languages expected to be used in this project besides the default one.
2. 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.

3. Insert your new document into the project.
4. List the initial contributors meant to work on the project.
5. 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 2.3.2.6 and in Section 3.1.2.1.
6. 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 should be read with an appropriate tool (`Xpdf` or `Acrobat Reader`, for example) to check its validity.
7. Assign tasks to every contributor. Ideally you should be able now to assign a reponsible for every single task of the life cycle of every module.
8. 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).
9. 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 one appendix 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. You can consult Section 3.1.1 for details about configuration files.

Note: All examples of command lines to issue assume that the current directory is `~/New_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.

Note: It is not mandatory to modify the main configuration file at that point. You can keep default values for now and come back here when you actually need to change parameters.

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

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <repository>
    <title>Documentation Project</title>
    <paths>
      <modules>modules</modules>
      <manuals>manuals</manuals>
    </paths>
          <doctype>-//OASIS//DTD DocBook XML V4.2//EN</doctype>
    <dtd>http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd</dtd>
    <outputs>
      <makefile>$DISTDIR/backend/Makefile.DB</makefile>
    </outputs>
    <manuals>
    </manuals>
    <languages>
      <lang>en</lang>
    </languages>
    <revisions>
      <original>
 <type role="1time">
   <name>write</name><author>tbn</author><weight>10</weight>
 </type>
 <type role="2time">
   <name>update</name><author>tbn</author><weight role="proportional">8</weight>
 </type>
 <type>
   <name>tproof</name><author>tbn</author><weight>4</weight>
 </type>
 <type role="2translate">
   <name>pproof</name><author>tbn</author><weight>2</weight>
 </type>
 <type>
   <name>ispell</name><author>tbn</author><weight>1</weight>
 </type>
 <type>
   <name>lproof</name><author>tbn</author><weight>4</weight>
 </type>
     </original>
     <translation>
 <type role="1time">
   <name>translate</name><author>tbn</author><weight>8</weight>
 </type>
 <type>
   <name>synch</name><author>tbn</author><weight role="proportional">6</weight>
 </type>
```

```
  <type>
    <name>ispell</name><author>tbn</author><weight>1</weight>
  </type>
  <type>
    <name>lproof</name><author>tbn</author><weight>4</weight>
  </type>
       </translation>
       <cost>0.01</cost>
     </revisions>
   </repository>
 </configuration>
```

The file is pretty self-explanatory (comments have been removed here for clarity sake), 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.

> **Warning**
>
> The `<manuals>` and `<languages>` list is handled by Borges and you *must not* modify it by hand. Same goes for the `<borges>` element which is used to record the `Borges` version used by the repository.

The `<revisions>` section defines the document's workflow, which represents the "life cycle" of modules or the "stages" through which each modules part of a document must pass. See Section 3.1.1.5 for more information.

2.3.2.2. Add the Languages to be Used

This is done in one single command. As we want to add both French and Spanish besides English, we will have to run:

```
make addlang NEWLANG=fr
make addlang NEWLANG=es
```

The `addlang` target will actually perform the following tasks:

- Update `conf/repository.xml`;
- Create all directories meant to hold language specific files (in `modules/` `entities/` and `images/`) and populate them with all default files;
- Make all module templates for this new languge for all defined documents;
- Add all new files to the CVS repository, if available.

The `addlang` target has more options, consult Section 3.1.4 for more information.

2.3.2.3. 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/Sample/master.top.xml` to `~/New_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-4.2/docbookx.dtd"[
<!ENTITY % entities SYSTEM "entities">
%entities;
```

```
]>
<book id="My_Book" lang="&lang;">
  <title>&amp;book-title;</title>
  <preface role="module" id="legal-notice">
    <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>&amp;chapter-1-title;</title>
    <sect1 role="module" id="chap1-sect1-section">
      <title>First Section Title</title>
      <para>Write the chapter 1, section 1 contents</para>
    </sect1>
    <sect1 role="module" id="chap1-sect2-section">
      <title>Second Section Title</title>
      <para>Write in here the chapter 1, section 2 contents</para>
    </sect1>
  </chapter>
  <chapter>
    <title>&amp;chapter-2-title;</title>
    <sect1 role="module" id="chap2-sect1-section">
      <title>Second Chapter First Section Title</title>
      <para>Write in here the chapter 2, section 1 contents</para>
    </sect1>
    <sect1 role="module" id=""chap2-sect2-section>
      <title>Second Chapter Second Section Title</title>
      <para>Write in here the chapter 2, section 2 contents</para>
    </sect1>
    <sect1 role="module" id="chap2-sect3-section">
      <title>Second Chapter Third Section Title</title>
      <para>Write in here the chapter 2, section 3 contents</para>
    </sect1>
  </chapter>
  <appendix role="module" id="app1-appendix">
    <title>First Appendix Title</title>
    <para>Write in here interesting appendix informations</para>
  </appendix>
</book>
```

Thinking in a more or less "modular" way we can say that, generally speaking, a book has: a title, a preface, chapters and appendices. So, that is exactly what is represented in the sample `master.top.xml` above, no more, no less.

Finally, our master document looks like a standard DocBook document. However, there is something essential to be noted: The `role="module"` attribute. The elements (usually, `sectX`, `chapter`, `appendix`) having this attribute, denotes the modules handled by `Borges`.

For example, the whole `sect1` element

```
      <sect1 role="module" id="chap1-sect1-section">
    <title>First Section Title</title>
    <para>Write the chapter 1, section 1 contents</para>
  </sect1>
```

will be used to generate the module's template. It will show as is in the document's guidelines but will be replaced by the module's content as written by its author in the module `chap1-sect1-section.xml`. Note that the module's name is taken from the structuring element ID.

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 directions for the 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.

Note: You might need to change the XML `SYSTEM` declaration of the DocBook DTD (`"/usr/share/sgml/docbook/xml-dtd` to suit your system. Note that if you use another version of the DTD, you will need to update the `doctype` and `dtd` elements of the main configuration file to fit the version number.

2.3.2.4. 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_Book master=~/New_Project/master.top.xml
```

this will create the new `My_Book` document based on the master file `~/New_Project/master.top.xml` (the file path *must* be absolute). It will actually perform the following tasks:

- Update `conf/repository.xml`;
- create `manuals/My_Book/` directory and populate it with all needed file and language directories;
- Make all module templates for this new document in all defined languages;
- Add all new files to the CVS repository, if available.

2.3.2.5. List Initial Contributors

Each "contributor" (writer, translator, proofreader, etc.) must now be known from the system. `Borges` uses this information version management and author credits among other things. Contributors are listed in `conf/authors.xml` and is filled with default values initially. So just edit `authors.xml` with your favorite text editor to enter your staff. Below is a sample profile:

```
<?xml version="1.0" encoding="UTF-8"?>
<authorgroup>
      <editor id="cb">
        <firstname>Camille</firstname><surname>Bégnis</surname>
        <affiliation>
          <address><email>camille@some_company.com</email></address>
        </affiliation>
      </editor>
      <author id="pp">
        <firstname>Peter</firstname><surname>Pingus</surname>
        <affiliation>
          <address><email>peter@pingus.com</email></address>
        </affiliation>
      </author>
</authorgroup>
```

Replace existing data with your own, possibly removing the `<author>` element if you are currently alone working on that project. Make sure to use unique IDs.

see Section 3.1.1.2 for more information about the structure of this file.

If you did not do it already it is time to tell to the system who you are in `conf/author.xml` (see Section 3.1.1.1).

2.3.2.6. Define Entities

> **Note:** This step is optional and can be performed in a loop during documents writing.

Project and document entities need to be defined. Project entities are those entities common to all documents, for example: computer program names. Document entities are those entities used only in a particular document. All entities files are XML files. Entities 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.

When you add a new super-document to the repository, strings needed to be translated found in the master file are automatically transformed into entities that are created in `ll/strings.ent`. You then just need to open that file and write your own content in it. By default `Borges` sets entities content to `FILL ME: entity-name`.

Below you have a sample `strings.ent` file:

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

2.3.2.7. 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.

2.3.2.8. Assign Tasks to Contributors

By default tasks are assigned to the people declared in the main configuration file (Section 3.1.1.5). 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.

2.3.2.9. Write the Modules and Create Images

All that is left now is to fill your book with content: write the modules and create the images and/or drawings your book will contain. If needed, 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 content. We won't tell you how to use `DocBook` here, there is excellent material about that all over the Internet. Start consulting The DocBook Wiki[2].

If you use entities in your modules, make sure you create a new entities file to hold the modules' entities (`entities/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 writing, PNG and JPEG (for raster images), EPS (for vector graphics), 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.

2.3.2.10. 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 `~/New_Project/manuals/My_Boo` 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 `~/New_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 looks 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 needed to compile documents in another language than the one declared in your `author.xml` profile conf file.

2. 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 that copy of the repository. A `Borges` project is meant to be stored in a CVS repository, and used by many people. Each author *must* customize this file in its own copy of the repository 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>
  <lang>en</lang>
</author>
```

- `<initials>` holds the author's identifier. This has been defined in `conf/authors.xml`. If not, it must be done: see Section 3.1.1.2.
- `<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.

3.1.1.2. conf/authors.xml

Each "contributor" (writer, translator, proofreader, etc.) must be known from the system. `Borges` uses this information version management and author credits among other things. Contributors are listed in `conf/authors.xml` and is filled with default values initially. So just edit `authors.xml` with your favorite text editor to enter your staff. Below is a sample profile:

```
<?xml version="1.0" encoding="UTF-8"?>
<authorgroup>
      <editor id="cb">
        <firstname>Camille</firstname><surname>Bégnis</surname>
        <affiliation>
          <address><email>camille@some_company.com</email></address>
        </affiliation>
      </editor>
      <author id="pp">
        <firstname>Peter</firstname><surname>Pingus</surname>
        <affiliation>
          <address><email>peter@pingus.com</email></address>
        </affiliation>
      </author>
</authorgroup>
```

That file should strictly follow the default structure and:

- The <editor> contributor type is used to store the project manager profile. There must be one and only one. This person will notably receive tasks not assigned to anybody else so that he can assign pending tasks.
- The <author> contributor type is used for all other people working on the project. There can be as many as needed.
- Each contributor must have a unique and meaningfull id used to identify him in the system. It should be composed of letters only.
- The email address will be used to send tasks the contributor is currently meant to perform (see Section 4.3).

3.1.1.3. conf/manual-default.xml

This file will be used as a template to generate document specific configuration files (Section 3.1.1.4). It holds the configuration parameters of the style-sheets to use *by default* when producing PDF and HTML output. Please refer to Section 3.3 for details on how to customize the default style-sheets to fit your particular needs.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<configuration omf.seriesid="">
  <makefile>$DISTDIR/backend/Makefile.DB</makefile>
  <style format="pdf" toolchain="jade">../../drivers/docbook-jadetex.dsssl</style>
  <style format="flat.html" toolchain="xsl">../../drivers/docbook-xhtml.xsl</style>
  <style format="html" toolchain="xsl">../../drivers/docbook-xhtml-chunk.xsl</style>

  <document id="">

    <style format="pdf"/>
    <style format="html"/>

    <language lang="en">
      <style format="pdf" toolchain="jade">../../drivers/db-jadetex-letter-en.dsssl</style>
    </language>
    <language lang="fr"/>
    <language lang="es"/>

    <exclude>multilingual</exclude>
  </document>

</configuration>
```

- <makefile> designates the Makefile holding the rules specific to a particular document type. The default one is for DocBook documents. $DISTDIR references to the location where Borges is installed (/usr/share/Borges/ by default. That allows you to redefine a complete new set of targets and use them easily on a per document basis.
- <style>: in this example, the first three <style> elements hold the style-sheet used for the three output formats provided by default with Borges. Under the <document> element, <style> tells which formats should be generated when making all the documents. Finally, it is possible to define a specific stylesheet for a specific language and a specific document in the way done with the last <style> element of the above example.
- <document> holds the configuration for the first sub document. The id attribute must be left empty and will be automatically filled at document insertion time.
- <language> tells in which languages that sub document is meant to be translated. That information will be used for tasks dispatching and documents generation.
- <exclude> see Section 3.1.1.4 for explanation. The multilingual flag is excluded by default. It can be used in modules to mark data to be included only in multilingual documents (see Section 3.1.2.5).

Remember that this file just holds the default configuration for the documents that will be inserted in the project. You will most probably need to adjust that configuration after document insertion (Section 3.1.1.4).

3.1.1.4. manuals/My_Book/conf.xml

This file holds the configuration of a specific documnent, and notably style-sheets to use when producing PDF or HTML output, as well as "aliases" and exclusion information for deriving various documents from a single super-document. File is based on the `conf/manual-default.xml` file (Section 3.1.1.3).

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<configuration omf.seriesid="7a5f2ef8-3239-11d8-8574-a94a75e630dd">
  <makefile>$DISTDIR/backend/Makefile.DB</makefile>
  <style format="pdf" toolchain="jade">../../drivers/docbook-jadetex.dsssl</style>
  <style format="flat.html" toolchain="xsl">../../drivers/docbook-xhtml.xsl</style>
  <style format="html" toolchain="xsl">../../drivers/docbook-xhtml-chunk.xsl</style>

  <document id="My_Book">

    <style format="pdf"/>
    <style format="html"/>

    <language lang="en">
      <style format="pdf" toolchain="jade">../../drivers/db-jadetex-letter-en.dsssl</style>
    </language>
    <language lang="fr"/>
    <language lang="es"/>

    <exclude>Mac</exclude>
    <exclude>multilingual</exclude>
  </document>

  <document id="My_Book_Mac">

    <style format="pdf"/>
    <style format="html"/>

    <language lang="en">
      <style format="pdf" toolchain="jade">../../drivers/db-jadetex-letter-en.dsssl</style>
    </language>
    <language lang="fr"/>
    <language lang="es"/>

    <exclude>PC</exclude>
    <exclude>multilingual</exclude>
  </document>

  <document id="ML-Doc" multilang="//part">
    <style format="pdf"/>
    <language lang="en"/>
    <language lang="fr"/>
    <language lang="es"/>
    <exclude>unilingual</exclude>
  </document>
</configuration>
```

We have already seen (Section 3.1.1.3) the meaning of the elements of that file. We will now detail the use the the `<exclude>` element and see how this document has been configured.

`<exclude>` holds the name of the "flags" to exclude in conditional-compilation of the document. See Section 3.1.2.4 for detailed explanation.

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

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

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

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

will compile a PDF file named `manuals/My_Book/My_Book_Mac.pdf` excluding all elements marked with `condition="PC"`.

Finally, there is a third document available in PDF only, and the `multilang` attribute tells us it's in fact a document containing various parts in various languages in the same book. In this case, the document `ML-Doc.pdf` will have its body (<part> here) repeated in English, French and Spanish. Please refer to Section 3.1.2.5 for more information.

3.1.1.5. conf/repository.xml

This file is the most important configuration file because it's the top configuration file for the whole `Borges` project. We will detail here a sample configuration file and see what one can modify by hand.

```
 <?xml version='1.0' encoding='iso-8859-1'?>
<configuration>
  <repository>
    <title>Documentation Project</title>
    <borges>0.11.2</borges>
    <paths>
      <modules>modules</modules>
      <manuals>manuals</manuals>
    </paths>
    <doctype>-//OASIS//DTD DocBook XML V4.2//EN</doctype>
    <dtd>http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd</dtd>
    <outputs>
      <makefile>$DISTDIR/backend/Makefile.DB</makefile>
    </outputs>
    <manuals>
      <manual>Book</manual>
      <manual status="inactive">Took</manual>
    </manuals>

    <pool id="Printer">
      <document id="Book/Book">
        <language lang="en">
          <style format="pdf"/>
        </language>
        <language lang="fr">
          <style format="pdf"/>
        </language>
      </document>
    </pool>

    <languages>
      <lang>en</lang>
      <lang>fr</lang>
      <lang>es</lang>
    </languages>

    <revisions>
      <original>
        <type role="1time">
          <name>write</name><author>tbn</author><weight>10</weight>
        </type>
        <type role="2time">
          <name>update</name><author>tbn</author><weight role="proportional">8</weight>
        </type>
        <type>
          <name>tproof</name><author>tbn</author><weight>4</weight>
        </type>
        <type role="2translate">
          <name>pproof</name><author>tbn</author><weight>2</weight>
        </type>
        <type>
          <name>ispell</name><author>tbn</author><weight>1</weight>
        </type>
        <type>
          <name>lproof</name><author>tbn</author><weight>4</weight>
        </type>
      </original>
      <translation>
```

```
        <type role="1time">
          <name>translate</name><author>tbn</author><weight>8</weight>
        </type>
        <type>
          <name>synch</name><author>tbn</author><weight role="proportional">6</weight>
        </type>
        <type>
          <name>ispell</name><author>tbn</author><weight>1</weight>
        </type>
        <type>
          <name>lproof</name><author>tbn</author><weight>4</weight>
        </type>
      </translation>
      <cost>0.01</cost>
    </revisions>
  </repository>
</configuration>
```

We will review now, section by section, what you can change and what far.

- `<title>` holds the qproject's name. This is used as the title in the reports generated by `Borges` report facilities. Mandatory.
- `<doctype>` holds the symbolic `DOCTYPE` of the DTD used for all the documents and modules in this project. There can be only one DTD per project. Make sure you use the same DTD in your documents master files. If you change that you'll need to update accordingly all of your master files. Mandatory.
- `<dtd>` holds the URI of the DTD specified by the `<doctype>` parameter above. 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 provided the rules in it do not conflict. Mandatory.

  Note: you can also specify the `Makefile` to be used on a super-document basis (see Section 3.1.1.4).

- `<manuals>` holds the directory name of the different documents, with one `<manual>` entry per document. This is automatically managed by the `adddoc` target (Section 2.3.2.4). .

  Caution

  It is highly recommended not to add here any manual by hand. Use the **make adddoc** command instead. See Section 2.3.2.4.

  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.

  In our example, the document `Took` is marked with attribute `status="inactive"`. That means this manual won't ba taken into account when generating reports or outputs.
- `<pool>`: this element that can be repeated as much as needed is used to define document subsets for a specific publication media. See Section 3.2.2.2.
- `<languages>` holds the languages supported by all documents, one `<lang>` entry per language containing the two letter ISO code (in lowercase) for that language.

  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.

There are two things you can do here:

- Change the order of the languages: the first one is the default one, and will be used in some rare places when the system needs a default language. The order then determines the order of the languages in reports.
- As seen above for documents, a `status="inactive"` attribute can be added to a `<lang>` element to temporarily deactivate it..

- `<revisions>` holds the revision types managed by the revision system. The order in which they appear defines the work-flow of the document's modules. The workflow presented here is the default one provided by `Borges`. It is possible to change steps name and number, removing or adding new steps. We will analyse first how the default workflow (Figure 3-1) is coded and then see how it is possible to customize it. The workflow is devided in two sections:

  original

  This is the workflow followed by original modules. There are two special states in that workflow. The one marked with `role="1time"` is only required one time (typically the first writing) and won't be required for module updates. Passing the state marked as `role="2translate"` will trigger the translation for all other languages, so that translators can begin their work. Finally, the task marked with `role="2time"` will be available for future releases only after the module is written.

  translation

  This is the workflow to be followed by translated modules. The state marked with `role="1time"` is only required one time (typically the initial translation) and won't be required again for module updates.

  See Figure 3-1 to see how those tasks articulate with each others. Each workflow is made of tasks with the following information:

  - `<name>` holds the name of the revision for the system. This is the meaning we give them for the default workflow: `write`, for module's initial writing; `translate` for module's initial 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. After that comes `update` for original modules updates in future releases, and `synch` for synchronization of the translations with the original.
  - `<author>` holds the "default" author initials or identifier for a revision type. If no author is assigned to a revision type yet, `tbn` (To Be Named) *must* be used.
  - `<weight>` is used to estimate the relative cost of a task with respect to writing task. For example the pedagogic proofreading (`pproof`) task has a relative weight of 2 with respect to `write` (10), meaning that we estimate it costs 5 times less to do pedagogic proofreading than writing a text (see Section 4.4). Of course you can use whatever values you wish here.

    You may have noted that some weights are added the attribute `role="proportional"`. That means that the cost of this task will be weighted by the actual changes done on this module. See Section 4.4 to get details on the way this is calculated.

  The `<revisions>` element is repeated for each language defined in the system. This is to allow defining specific authors for every language. See Section 3.1.4 to learn how to define those authors at language addition time.
- `<cost>` is the estimated cost per written character for the writing task (see Section 4.4). All other costs will be derived from this one according to respectives weights. Adjust to your own value.

3.1.1.6. 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. The first entry has been automatically created at document creation time (see Section 2.3.2.4) following ones are added by translators to record the date at which they began translating a document:

- <revnumber> contains the revision number (or edition number) of the document. It is the one corresponding to the current document release (see Section 3.4.3). Mandatory.
- <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 in 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 in 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[1].

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 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 for 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. If the required format is not available `Borges` will take care of converting the image to the needed format automatically. Supported formats as input image files are:

.png
: Portable Network Graphic (bitmap);

.jpg
: Compressed bitmap usable for photos;

.eps
: Encapsulated PostScript (Vector);

.pdf
: Encapsulated Portable Document Format (Vector);

.fig
: native output format of the `xfig` diagram application.

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 with whatever you want.

1. Acronyms are used "almost" without change between all languages/projects. One that does change, for example, is ISDN which is RDSI in Spanish.

Additionally, whenever `Borges` finds a missing image, it will report it in the `<manual>.missing.xx.img` text file. So if you have 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 that file directly to check that no other 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 an 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 between them.

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 books 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 only 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 user guides from the `Tortoise-UserGuide` super-document. This is done in the super document configuration file Section 3.1.1.4. 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>
   <lang>en</lang>
   <format>html</format>
   <exclude>sparc</exclude>
  </manual>
   <manual id="Tortoise-Sparc">
   <lang>en</lang>
   <format>html</format>
   <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. Miltilingual Documents

3.1.2.6. Document Validation

From time to time, it is *recommended* you check that 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 super-document into.

Validation of plain `master` document does not mean that all your sub-documents will be actually valid. To ensure that the `Tortoise-i386` document is actually valid (taking into account exclusions and indexes), issue:

```
make -C manuals/My_Book Tortoise-i386.flat.validate
```

Finally, if you wish to validate against one single module, run:

```
make -C modules/en tortoise-install-sect1.validate
```

> **Caution**
>
> At the moment, module validation does not check cross references (<xref>) validity.

3.1.2.7. Making Translated Paragraphs Transparent to the Revision System

When translating a module into a another 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 revision management system will report differences between 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 every 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.

If your project is CVS enabled, the new module templates will automatically be added to the CVS repository. See Section 6.2.1.

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 NEWLANG=ll
```

will declare the new `ll`[2] language document for the whole project. It will actually perform 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;

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

- 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/ll/*.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.

3.2. Generating Final Documents

Beyond the simple document generation, many advanced features are available to allow the user to easily customize the output formats or to 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` documents in `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, containing 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 a very big file.
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 One Time

3.2.2.1. For the Whole Project

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 be 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>"]
```

Without the SUBDOCS option, this command will generate all sub documents as defined in the conf.xml files (Section 3.1.1.4) of all the documents defined for the whole project (Section 2.3.2.4).

If you supply a SUBDOCS list, only the specified super-document/document pairs will be generated. If you wish to get only 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/Install-guide-tar"

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"
```

Which will result in all manuals in Outputs/, sorted by language and document.

3.2.2.2. For a Documents Subset

Documents from a single project are often published separately for different purposes or publishing medias. This feature allows to define pools of documents in specific formats and languages, that can be used to generate in a single commands all the output documents associated, or an archive containing the sources for just those deocuments.

We have seen at Section 3.1.1.5 that we can define elements like this:

```
<pool id="Printer">
  <document id="Book/Book">
    <language lang="en">
      <style format="pdf"/>
    </language>
    <language lang="fr">
      <style format="pdf"/>
    </language>
  </document>
</pool>
```

Note about the format of this element: The <style> element must always specify the format attribute. It must be enclosed in a <language> element (with a mandatory lang attribute) which must be enclosed in a <document> element (with an id attribute). The id attribute identifies the document/subdocument pair concerned. Each of these elements can be repeated as much as needed to reach the exact combination of document, language and format needed.

Once a pool is defined in conf/repository.xml with appropriate id, one can start the generation of the documents defined in it in a single command:

```
make all POOL=Printer
```

That will generate and place in Outputs/Printer all the documents defined in the pool with ID Printer.

Likewise, the following command:

```
make archive POOL=Printer
```

Will create an archive called Printer-9.2.tar.bz2 (9.2 being replaced by the current project release) containing a striped down version of the project repository allowing only to generate the documents defined in the Printer pool with **make all**.

3.2.2.3. For one super-document

A feature also allows you to generate all documents (in all formats and languages defined in the `conf.xml` file) associated to a specific super-document.

```
make -C manuals/Install-guide/ all
```

does the job for the `Install-guide` super-document for example. The resulting files will be stored in the `manuals/Install-guide/Outputs/` directory.

3.2.2.4. For one document

Finally you may want to generate all formats and languages outputs associated to a single document as defined in the `conf.xml` file. Simply run

```
make -C manuals/Install-guide/ Install-guide-RPM.all
```

The resulting files will be stored in the `manuals/Install-guide/Outputs/` directory.

3.2.3. Generating a Single Module

When you are working on writing and/or translating a module, you will 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 out the long book compilation times.

The command synopsis for compiling a single module is:

```
make -C manuals/module module MOD=<module_name> FORMAT=<output_format> [LANG=ll] [exclude=foo
```

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, and all single module compilation outputs go 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.

The `exclude=` parameter is optional and it is used to exclude elements from input XML files. See Section 3.1.1.4.

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.

3.2.4. OMF Support

The `Open Metadata Framework` is an initiative of ibiblio[4] to provide an open cataloging system for documentation. It allows various documentation projects to present their documents through a common catalog interface, making browsing and searching much easier. See the OMF home[5] for more information. The ScrollKeeper[6] project offers a common interface for managing OMF files.

To generate the OMF file for a specific sub-document, simply run, for example:

4. http://www.ibiblio.org/
5. http://www.ibiblio.org/osrt/omf/
6. http://scrollkeeper.sourceforge.net/

```
make -C manuals/Install-guide/ Install-guide-RPM.omf
```

This will generate in a single file all the OMF metadata necessary to catalog all the actual output files (all languages and all formats) derived from the specified document, according to the content of the `conf.xml` file.

The actual cataloging metadata is extracted from the master document header. The following table shows the DocBook elements (from inside the `info` element (`bookinfo`, `articleinfo`, etc.) of the master file) used to fill the information in the different OMF elements.

Table 3-2. Correspondence between DocBook and OMF elements

OMF element	DocBook equivalent
creator	first author
title	title: subtitle
date	pubdate
description	abstract
type	releaseinfo
coverage/@architecture	@arch
coverage/@os	@os
rights/@type	legalnotice[1]/@conformance
rights/@license	legalnotice[1]/@role
rights/@holder	copyright/holder

Consult the http://www.ibiblio.org/osrt/omf/omf_elementsOMF Specification to learn about the meanings of the OMF elements.

> **Note:** The OMF files are automatically generated when all the formats associated with specific documents are generated (**make all**)

3.3. Output Style Customizations

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

3.3.1. Customizing Existing Formats

As we already seen in Section 3.1.1.3, the 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.

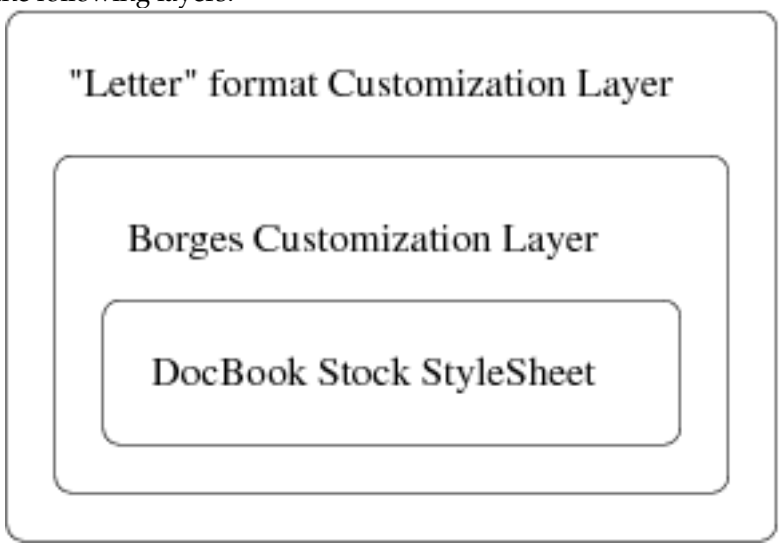
Consult the documentation on how to customize XSL[7] and DSSSL[8] stylesheets if needed.

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 both Europe and in the United States. Therefore you need 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">
  <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="UTF-8"?>
<configuration>
  <stylesheet>
    <dssslprint>../../drivers/docbook-jadetex.dsssl</dssslprint>
    <xslxhtmlflat>../../drivers/docbook-xhtml.xsl</xslxhtmlflat>
    <xslxhtmlchunk>../../drivers/docbook-xhtml-chunk.xsl</xslxhtmlchunk>
  </stylesheet>
```

7. http://www.docbook.org/wiki/moin.cgi/DocBookXslStylesheetDocs
8. http://www.docbook.org/wiki/moin.cgi/DocBookDssslStylesheetDocs

```
 <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 the same time, 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.

Module status (Section 3.4.1)

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

2.

Translation Freshness (Section 3.4.2)

Translating a document is something common. Updating translations for 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 module authors, translators, contributors, etc.

In addition to module revisions it is necessary to handle whole document revisions (when a new version of the documented product is released for example).

3.4.1. Module Life Cycle

3.4.1.1. The underlying philosophy

A module is born when it is referenced for the first time in a master 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[9]:

1.

Written

When the redactor has finished writing an original module.

2.

Translated

When a translator has finished the translation of an original module to 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.

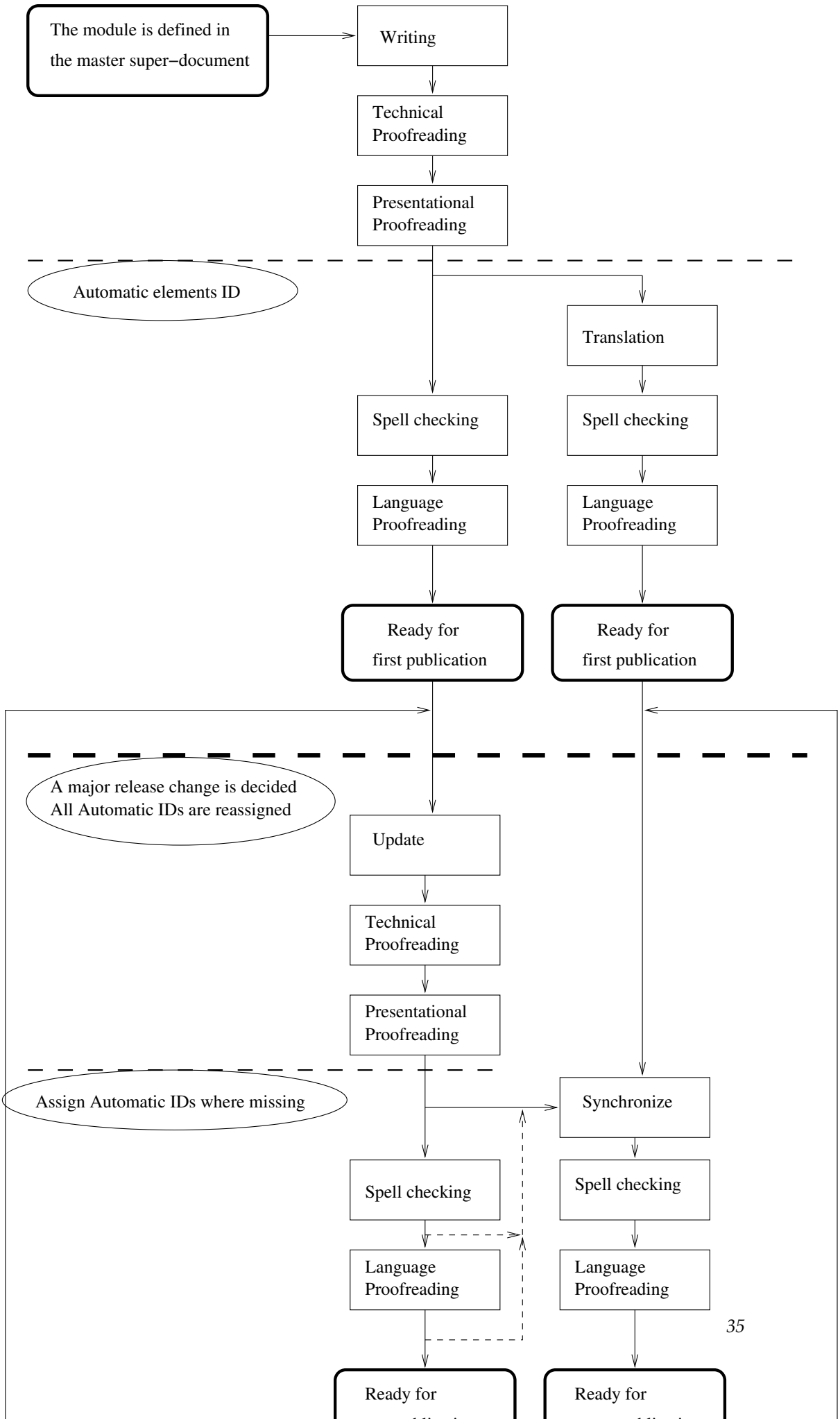
6.

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, it 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.

9. this is for the default `Borges` modules workflow.


The module is defined in
the master super−document
Writing
Technical
Proofreading
Presentational
Proofreading
Automatic elements ID
Translation
Spell checking
Spell checking
Language
Proofreading
Language
Proofreading
Ready for
first publication
Ready for
first publication
A major release change is decided
All Automatic IDs are reassigned
Update
Technical
Proofreading
Presentational
Proofreading
Assign Automatic IDs where missing
Synchronize
Spell checking
Spell checking
Language
Proofreading
Language
Proofreading
Ready for
Ready for

This figure is the illustration of the information contained in Section 3.1.1.5, and requires a few comments:

- Just after the original module passes the presentational proofreading status, all automatic IDs in it are (re)assigned, as well for all its translations. This resets the automatic IDs numbering from one to the other release.
- After a major release change is decided the writing and translation states are not required anymore, simply because this only needs to bo done once. Instead they are replaced by the "Update" and "Synchronise" steps.
- The "Synchronise" step can be reactivated even after the original has passed the "Presentational Proofreading" status. This is to allow the translations to be updated with respect to the original if this one is to be altered afterwards. This is represented by dotted arrows in the diagram. See Section 3.4.2. Note that the subsequent steps for the translation (spell and language proof) are not reactivated if they had been done already.

> **Note:** This is the description of the default workflow as proposed by `Borges` by default. It is possible to define a customized workflow by defining it in Section 3.1.1.5.

3.4.1.2. Module Status in Practice

The module history is stored in a hidden file, and *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 made by the pedagogical proofreader. The command to issue in the directory containing 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 that you assign all tasks in all languages for all modules. That will ensure that no task is left orphaned, improving project efficiency.

This operation is achieved 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
```

3.4.1.4. Customizing the Modules Workflow

To Be Written

3.4.2. Inter-Language Module 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 only one 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 in 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 conscientious 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 becomes ready for translation. The system then automatically adds IDs to any possible atom[10] 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>
   ```

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
   ```

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

```
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 risks of duplicate IDs) or to let the system reassign all IDs throughout the module if there have 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 talk about report generation here, but rather about how to read the reports and what to do according to the information contained in the reports.

passwords (Peter Pingus)	Passwords	ispell'ing (To be named)	ispell'ing (To be named)	Translating (To be named)

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

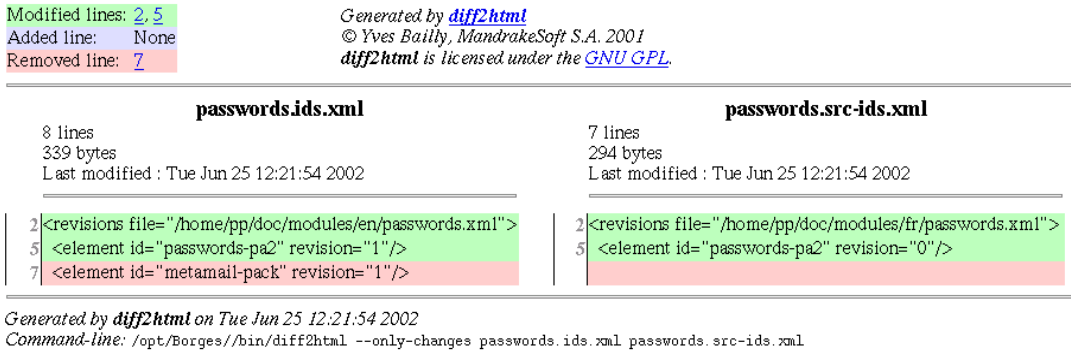


Figure 3-4. Changes in IDs/revisions

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

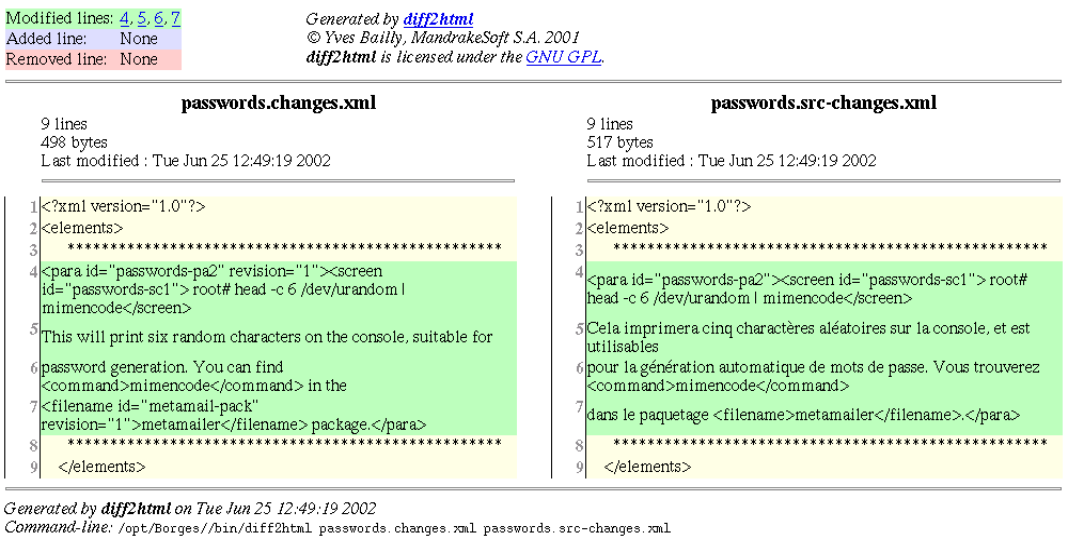


Figure 3-5. Side by side not synched elements

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 will know the atom has been updated.

3.4.3. Project Major Release

When a new version of a software or product is released, its accompanying documentation generally also needs big changes. That's in this case that `Borges` offers a documents major release feature, allowing you to restart the production cycle on existing modules.

When you have created your documents you have assigned them an initial release number (see Section 2.3.2.4). Then all tasks that have been performed on associated modules since then are linked to this precise release number. Increasing the release number of a project will create a new set of tasks for every single module of the project with the new release number.

All this is performed in one single command:

```
make release REL=2.0
```

Note: This process can take a fair amount of time if there are a lot of modules and languages.

It is recommended to change the release number of all documents of a same project at once. Indeed, if two documents with two different release numbers use a same module, it will result in conflicts when generating reports. Though if you prefer to manage document release numbers separately you can with the following command:

```
make -C manuals/My_Manual/ release REL=2.0
```

3.4.4. Generating Reports

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

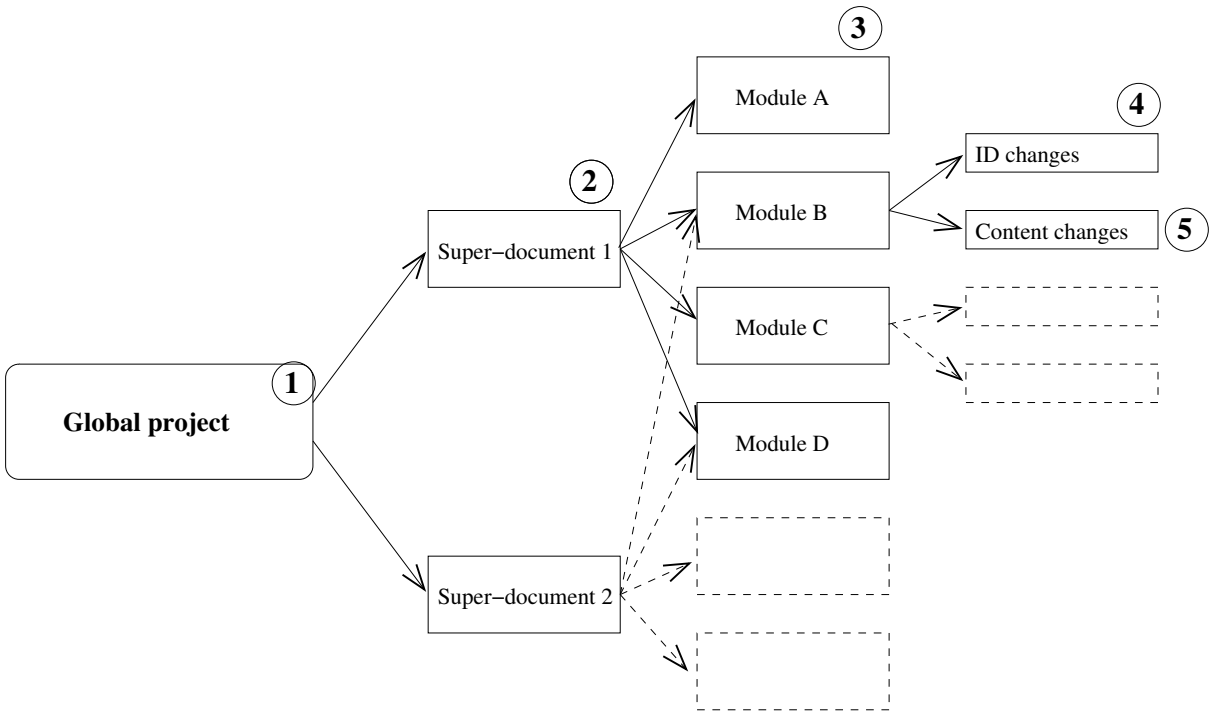


Figure 3-6. The reports generated by Borges

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.

3.4.4.1. 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
```

This will generate the `index.html` file and you just have to point your browser to that file. For an example of such a report, consult the Overall report for Borges Manuals[11].

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 web site the current project status. It is enough to upload the content of the `reports/` directory on a web site and point people to it.

3.4.4.2. 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[12].

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 a cell appears with a red background, it means this translation is outdated with respect to the original module. See Section 3.4.2.3.

   A click on the text of that cell will lead to the corresponding Section 3.4.4.3.

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

3.4.4.3. 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 contains 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).

3.4.4.4. 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.

3.4.4.5. Content Changes Report

Same as above, but 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 project manager.

The project reports (Section 3.4.4.1) are a great help. But there are additional tools to remind authors of their current tasks (Section 4.3), and to evaluate the work made by each author (Section 4.4).

4.1. Server Side Repository

Many of the following features are meant to be run periodically. Therefore it is highly recommended to create a special user on a server machine that would act as a robot. It would have a local copy of the project repository, and would be directed to periodically run tasks. This solution prevents possible conflicts with your own working repository.

4.2. Automatically Compile and Publish Reports

To ease automatic publication of reports, `Borges` provides a single `publish` target, wich will:

1. Clean the repository and get a fresh copy from CVS,
2. Generate reports and/or output documents,
3. Transfer them to a web server.

The command synopsis is simple:

```
make publish [PUBTYPE={report output}]
```

If the `PUBTYPE` argument is not provided, both reports and outputs will be generated and published. If you wish to update only one of those, specify it with `PUBTYPE` argument. For example if you wish to only update the reports on your website and not outputs, run **make publish PUBTYPE=report**.

This command is a very good candidate for periodic (`cron`) scheduling. This is a `crontab` example which updates reports every hour, and updates reports and output documents twice a day.

Example 4-1. Crontab Publishing

```
0 * * * *    nice make -C /home/r2d2/Borges/doc/ publish PUBTYPE=report
30 8,12 * * * *    nice make -C /home/r2d2/Borges/doc/ publish
```

4.3. Sending Mails to Authors

This very useful feature allows you to prepare mails for every author listed in the project. Those mails will list all tasks the author should be working on currently. Then Those mails will be sent, provided a mail server is available.

> **Caution**
>
> This feature is only available if a local mail server is available (on the same machine where the robot stands). If you wish to only generate the mails and send them on your own, you can simply run **make mails** in `reports/`.

To send the mails, run: **make sendmails** in the `reports/` directory. That command will:

1. Generate tasks reports„
2. Build contributors mail according to reports,
3. Send messages through local mail server.

This command is a very good candidate for periodic (`cron`) scheduling. This is a `crontab` example which send mails every morning of working days.

Example 4-2. Crontab Mails Delivering

```
30 7 * * 1,2,3,4,5    nice make -C /home/r2d2/Borges/doc/reports sendmails
```

4.3.1. Adding Information on the Mail Footer

You can put additional information at the end of the mail, simply by writing your footer in the file `conf/mailfooter.txt`.

`Borges` actually provides a mechanism to automatically fill this footer with useful information for your contributors. You can find an example in `/usr/share/Borges/template/conf/mailfooter.txt`

```
P.S.
You can make your modifications directly from
@frontendurl-perso@

You can consult the compiled version of the manuals from
@outputsurl@
And current status at
@reportsurl@

Those modules corresponds to the files you can find in CVS with following parameters:
CVS_RSH=@CVS_RSH@
CVSROOT=@CVSROOT@

Do not hesitate to contact the documentation manager for any further
information.
```

You can simply copy this file to your robot's `conf/` directory, and eventually customize it. Words between `@` will be replaces by their value as defined in `conf/publish.xml`.

When using the **make sendmails** command, if file `conf/mailfooter.txt` exists it will be automatically appended to every mail sent to authors.

4.4. Accounting Reports

4.4.1. Project Report

This special report is made of tables for each manual and for the the whole project that summarizes the authors contributions for each module 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 shows 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 manual, 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 detailed reports per manual which list authors contributions to each module. You can also generate those manual 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.

4.4.2. Authors Report

This report shows the same information but in an author's point of view: it simply lists all tasks done by each author and evaluates their cost. This report is generated by running:

```
make -C reports/ contributions.html
```

Chapter 5. Borges and XML Editors

5.1. Which Editor Should I Use?

`Borges`' modules are plain text XML files, so *any* text editor (`Emacs`, `Vi`, you-name-it) should be OK. `Emacs` is the favorite editor of `Borges`' authors, mainly because its PSGML module makes working with XML files really easy.

Nothing prevents you from using a GUI-based graphical editor, provided that it does not interfere with `Borges`' requirements and management features. To enhance compatibility with such editors, `Borges` provides a mechanism to export modules in a format suitable to such editors, and then import them back into `Borges` format. See Section 5.3.

However, a "pure" text editor is highly recommended because it offers the ultimate flexibility when working with XML files.

5.2. Emacs+PSGML

As mentioned before, `Emacs` with its PSGML module makes your life easier when working with XML files. We will explain some basic PSGML commands and DTD features in the following sections.

5.2.1. Installing PSGML

You can download PSGML from the PSGML home page[1] or, if you are a *Mandrake Linux* user and have the "contribs" source or CD configured in the Software Manager, simply issuing **urpmi psgml** will install PSGML.

5.2.2. DTD-Awareness

PSGML mode is DTD-aware. This means that when using PSGML you will always produce well-formed and valid XML files. Please refer to The XML FAQ[2] for more information about the meaning of the terms "well-formed" and "valid".

PSGML should be told somehow about the DTD your module intends to conform to, in order to be "aware" of it. To do so, `Borges` inserts the following at the end of every module's XML source file:

```
<!-- Keep this comment at the end of the file
Local variables:
mode: xml
sgml-parent-document: ("../../manuals/module/en/psgml-top.xml" "root_element")
End:
-->
```

where `mode: xml` guarantees that `Emacs` will enter XML mode after auto-loading PSGML when you open the module's source XML file with it, and `root_element` should be replaced by the module's root element, for example `chapter` if the module in question is a chapter.

> **Tip:** In case you are wondering, the `psgml-top.xml` filename is automatically created under the `manuals/module/` directory when configuring `Borges`.

1. http://psgml.sourceforge.net
2. http://www.ucc.ie/xml/faq.xml

5.2.3. Basic PSGML Commands

PSGML mode adds powerful commands which take the burden of typing element tags and/or element attributes when working with plain-text XML files.

> **Note:** The following table lists *some* PSGML commands without any order nor preference, please refer to the PSGML documentation[3] for a complete list of available commands.

> **Tip:** When you see something like **Ctrl**-**C**-**Ctrl**-**E** it means that you have to press the Control key plus the C key, and right afterwards the Control key plus the E key.

Table 5-1. PSGML Commands

Command	Keyboard Shortcut or Menu Entry	Description
Insert Element	**Ctrl-C-Ctrl-E**	Inserts an element constrained by the DTD. That is, only elements allowed by the DTD can be inserted. If you press the **Tab** key, a list of valid elements is shown in `Emacs`' mini-buffer. If the element to be inserted requires attributes you will be prompted to enter their values in `Emacs`' mini-buffer.
Validate File	**Ctrl-C-Ctrl-V**	Loads the DTD, parses it (if not already parsed), and then presents the external validation command in `Emacs`' mini-buffer. Pressing **Enter** will proceed to validate the file showing a list of validation errors.
Next Trouble Spot	**Ctrl-C-Ctrl-O**	This one includes the validation one, but instead of showing a list of errors it stops when encounters the first error and shows the error message in `Emacs`' mini-buffer. This is the preferred way to validate files.
End Element	**Ctrl-C-/**	Ends the current open element. Actually, the Insert Element command inserts opening and ending tags where appropriate, but if you type the element name and want to close it without typing the whole closing tag, then this one comes handy.
List Valid Tags	**Ctrl-C-Ctrl-T**	Lists all the valid tags that can be inserted at the current cursor position. This is useful when working with "complex" DTDs (like `DocBook`) where some elements can have dozens of elements inside them, and you cannot expect to know them all by heart. It can be used as a quick DTD "reminder".
Fold Element	**Ctrl-C-Ctrl-F-Ctrl-E**	Folds the element the cursor is at. This is very handy when working with big files to have a quick view of the document's (or part of a document's) structure. The effect of folding an element is that only the opening tag and one line of content ending in ellipsis (. . .) is shown.
Unfold Element	**Ctrl-C-Ctrl-U-Ctrl-E**	Unfolds the folded element the cursor is at.

3. http://www.lysator.liu.se/~lenst/about_psgml/psgml.html

Command	Keyboard Shortcut or Menu Entry	Description
Insert Attribute	**Markup⟶Insert Attribute**	Shows a list of the attributes valid for the element the cursor is at from where you can chose the one you want to insert. If the attribute selected requires a value you will be prompted for it in `Emacs`' mini-buffer.

5.3. "WYSIWYG" XML Editors

The modules, as handled by `Borges`, are not directly parsable by other programs because they are meant to be called from a master file, which contains all needed references to DTD and external entities. That's why `Borges` provides a mean to add the necessary information into those module files, so that any XML editor should be able to open it. After the file has been edited, another operation transforms back the module file in a `Borges` suitable format.

This procedure has been tested with XMLmind XML Editor[4], and should work with other XML editors such as Morphon[5] and many other editors[6].

1. Export the desired module

   This is done on the command line:

   ```
   make -C modules/xx/ module_name.edit.noents.xml
   ```

   replacing `xx` and `module_name` by the language and the module name you wish to edit. That generates a file (`modules/xx/module_name.edit.noents.xml`) suitable for editing with any XML editor, provided the configuration files are properly setup (see Section 3.1.1.5).

2. Edit the module

   Simply open the file resulting from previous operation (`modules/xx/module_name.edit.noents.xml`) in your editor.

   Caution

   There is one important limitation to be aware of: as some editors do not handle properly external entities, they are "escaped", so that they are not recognized as entities by the editor. For example, the entity `&borges;` as found in a XML file, will be transformed to `&borges;` in the editable file, so that it will *appear* as `&borges;` in the WYSIWIG editors and not replaced by the entity value. When using entities in such editors, this same syntax will have to be used.

   When editing is done, it is enough to save the file locally and go to next step:

3. Import the module back into `Borges`

   To transform back the module to a format compatible with `Borges`, the following command will have to be run:

   ```
   make -C modules/xx/ module_name.revertedit
   ```

   this will move the content of the modified module from `modules/xx/module_name.edit.noents.xml` back to `modules/xx/module_name.xml`. It is now possible to validate the module or send it to the CVS server.

4. http://www.xmlmind.com/xmleditor/
5. http://www.morphon.com/xmleditor/index.shtml
6. http://wiki.docbook.org/topic/DocBookAuthoringTools

Chapter 6. Borges and CVS Integration

`Borges` is designed to integrate flawlessly into a CVS environment. We will detail here the initial procedure to start a new project with CVS support and look in detail how to use this new repository with CVS.

6.1. Starting a New Project on CVS

The principle is quite straightforward: you create an initial repository, you import it to a CVS server, and voilà! We will detail here the steps up to the point where you add your first document to the project.

First of all you must have a working CVS repository and access to it. If your organization does provide one to you ask for access to it and set the CVSROOT environment variable accordingly. If not it is quite easy to create a local CVS repository, refer to your CVS documentation and set the CVSROOT environment variable.

1. Create a new project skeleton

   This is the same command as usual, we're here creating a project in `~/my_doc/` with French as default language:

   ```
   /usr/share/Borges/bin/configure ~/my_doc/ fr
   ```

2. Import the project skeleton to the CVS repository:

   `Borges` povides a special wrapper to perform that task:

   ```
   cd ~/my_doc/ ; make cvsinit PROJECT=MyNewProject
   ```

   You will see all files being added to the CVS. Ones this is done you have to retrieve your own copy of the CVS module `MyNewProject` that has just been created.

3. Checkout the new CVS module to start working

   Make sure the CVSROOT variable is properly set. We will checkout our local copy of the New project in `~/cvs/`:

   ```
   cd ~/cvs/
   cvs checkout MyNewProject
   ```

4. Initialize the new copy

   The last step consists in preparing this working copy so that you can use it as any other `Borges` project:

   ```
   cd MyNewProject
   ./configure
   ```

   All is now ready, you can add your first document to the system. Read the next section to learn the changes that occur in `Borges` working with CVS

6.2. What changes when using CVS

When a `Borges` repository is integrated in a CVS environment, the bahavior of some targets is modified and some others become available. We detail this specificities in here.

> **Tip:** If you prefer `Borges` not to issue any CVS command, even when you are working inside a CVS environment, add the option `CVS=no` to you **make** command lines.

6.2.1. Commands with Modified Behavior

Basically some commands that add new templates to the project add those new files automatically to the CVS.

Tip: If you don't want to mirror your changes on the CVS server when issuing commands locally, simply add option `CVS=no` to your command line when running following commands. Don't forget then to commit manually your changes to CVS afterwards.

adddoc

When adding a new document all template files created by `Borges` (entities, modules, image directories, etc.) are automatically added and committed to CVS. The same happens for the `master.top.xml` and all needed `Makefile`s. All this is done for all active languages.

addlang

The same occurs here, for all modules needed by all documents, as well as entities.

alltemplates

This target, executed in a super-document directory, generates all new module templates for all languages and adds them to CVS. This is useful when you just have added new modules to an existing super-document.

module.revision TYPE=pproof

When a module passes the `pproof` state (or whatever it is named according to step marked as `2translate`) its content is copied to translation modules. `Borges` then automatically commits the translation modules to CVS.

module.revision

When you pass a revision on a module (even a `.todo` one) it is automatically committed with a standard log message indicating the task passed and the contributor. You can override this default message by providing your own changelog in the optional parameter *`CVSLOG`*.

6.2.2. New Useful Commands

Some new commands appear to ease the management of CVS sources from within `Borges`.

checkout

This target simply retrieves latest version from CVS and reconfigures the repository.

cvsinit

This target imports the current `Borges` repository to a CVS server. Consult Section 6.1.

master.top.commit

This target, executed in a super-document directory, will commit a modified super-document (`master.top.xml`) to CVS *checking previously it is fully valid.*

<module>.commit

This target, executed in a modules directory (`moduless/en/` f.e.), will commit a modified module (`<module>.xml`) to CVS *checking previously it is fully valid.*

commit [modules="module1 module2 ..."]

This target, executed in a modules directory (`moduless/en/` f.e.), will validate and commit all modified modules in current directory. Specifying the `modules` option will perform the action only on specified modules.

<image.ext>.commit

This target, executed in an images directory (`images/en/` f.e.), will add and commit a modified image (`<image.ext>`) to CVS *checking previously it is fully valid.*

<entity>.commit

This target, executed in an entities directory (`entities/en/` or `manuals/My_Book/en/` f.e.), will add and commit a modified entities file (`<entity>.ent`) to CVS *checking previously it is fully valid.*

Tip: If you prefer not to provide the change log in the CVS editor (normally `vi`) every time you commit a file to CVS, you can provide your comments directly on the command line thanks to the `CVSLOG` option. For example

```
make mymodule.commit CVSLOG="Fixed the foo option syntax"
```

would directly commit changes made on module `mymodule` with specified chqangelog, without opening the text editor.

Chapter 7. Programmer's Reference manual

Here we will get into the Borges internals. This may be of interest for the developer as well as for the user wishing to take advantage of the most advanced features 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 do 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 builds a repository template so that users can quickly start using Borges.

> **Tip:** Borges gets installed 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. Documentation 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, containing the documentation source files managed by Borges.

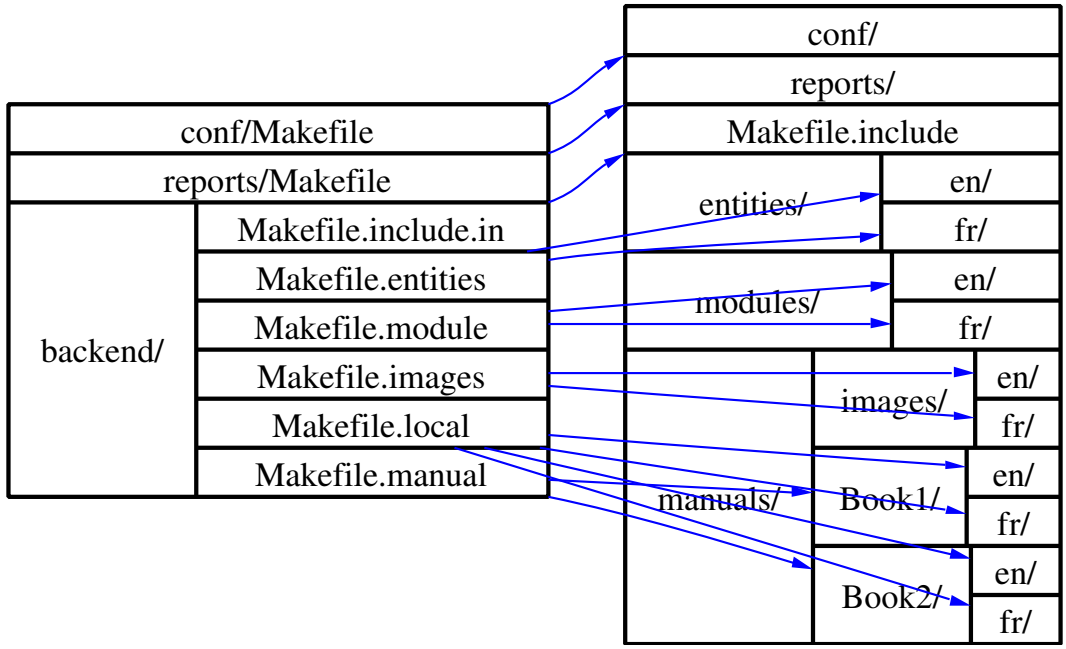


Figure 7-1. Distributing Makefiles

7.1.3. Makefiles in Action

Here we show you how Makefiles are linked together. Figure 7-2 shows how Makefiles include each other. An arrow in the diagram means "includes".

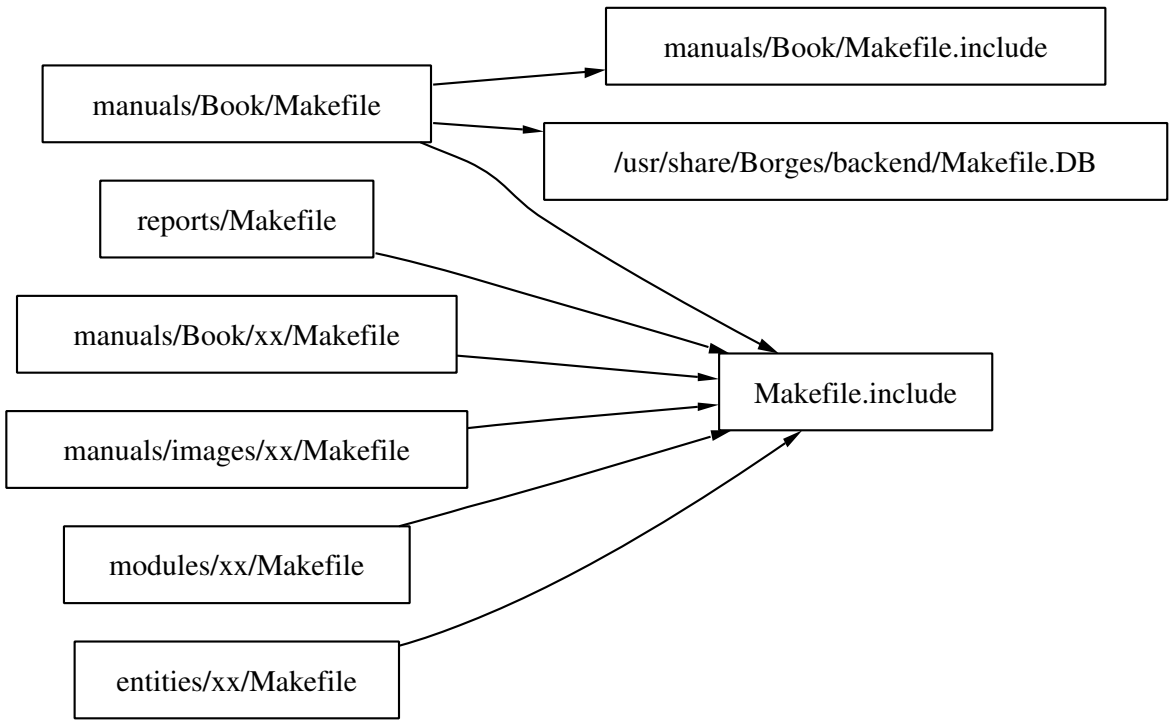


Figure 7-2. Makefiles Relationships

All paths are relative to the project root directory 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 main configuration file `conf/reposqitory.xml`. It also contains 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 guidelines to the final HTML book.

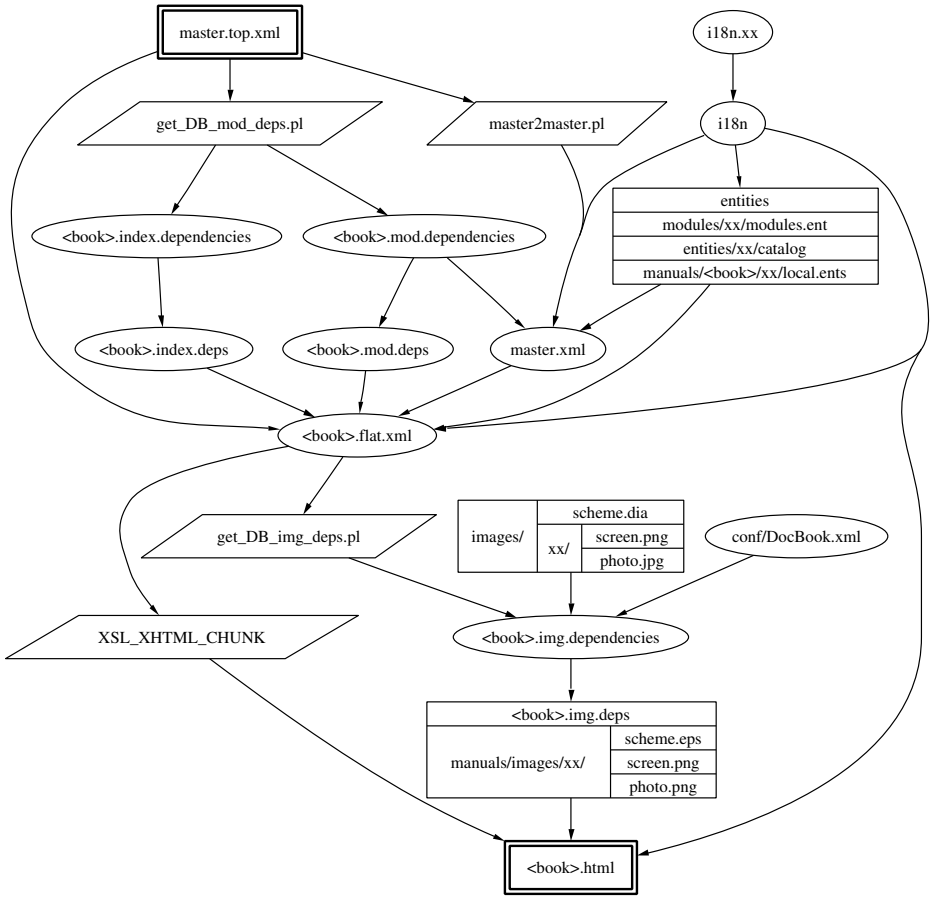


Figure 7-3. Distributing Makefiles

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

7.4. Supporting Another DTD than DocBook

To Be Written

7.5. Notes on Borges Installation

All is done so that the Borges backend can easily be installed anywhere on a system, including in a user's home directory. Efforts are also made so that it can be installed on other platforms than Mandrake Linux. If you try to install Borges on *GNU/Linux* and it doesn't work, we'll be glad to hear from you.

7.5.1. Installing Borges on an Unusual Path

According to the "Filesystem Hierarchy Standards", Borges installs itself by defaults into /usr/share/Borges/ when running the **make install** command in Borges repository. Changing the DESTDIR parameter will overwrite this behavior.

Example 7-1. Installing Borges in Home Dir

```
$ make install DESTDIR=~/BORGES
```

7.5.2. Adapting Borges to unusual Environment

Borges is developped under Mandrake Linux and expects various files to be available at a specific location. The first step is to make sure all the dependencies listed at Section 2.1.3 are correctly installed and working. If your PATH is correctly set, Borges should be able to acces the various executables he needs.

We will now review the external files Borges needs to work.

7.5.2.1. DTDs and StyleSheets

The template repository and sample files provided with Borges currently refer directly to files in your local filesystem. Catalogs are not currently used but nothing prevents you from using them. To adapt the paths to your local configuration, you'll have to edit files

```
template/drivers/docbook-index.dsssl
template/drivers/docbook-jadetex.dsssl
template/drivers/docbook-xhtml-chunk.xsl
template/drivers/docbook-xhtml.xsl
Sample/master.top.xml
```

> **Note:** It is recommended to use URI references and catalogs whenever possible, to allow documents generation to work on systems where DocBook is not at the expected place.

7.5.2.2. Other Control Files

openjade is a tool used for SGML transformations. Borges needs to access some of the files distributed with openjade. Those files are listed in template/conf/DocBook.xml and their path must be adapted to your local configuration. This file also holds the access path for the collateindex.pl DocBook script.

Note: As the local configuration can change from one user to another, inside the same documentation project, it is possible for users to adapt access path in their own `conf/author.xml` configuration file.

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 bug reports: 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 and you cannot find the answer in the documentation, post a support request there;
- Submit patches: You've come up with modifications to the source code to fix bugs or add features to `Borges`? You can submit the patches here.
- Ask for new features: If you wish to see more functionalities added to `Borges`, suggest them here with your detailed arguments.

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. mailto:sympa@moondrake.com
4. mailto: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

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

A.1. General purpose maintainance commands

Initialize the system

`./configure [BORGESDIR=/path/to/Borges]`

prepares the system to accept comands. Optionnaly, one can specify the path to the Borges installation if it is different from the default location (`/usr/share/Borges`).

Clean repository

Three versions exist:

make clean

Removes output documents and reports.

make cleaner

Additionally, that removes the images prepared for publication in output documents (in `manuals/images/`).

make superclean

This command will do its best to remove all files that are not present in the CVS repository. **./configure** will have to be run afterwise to make the system functional again.

Retrieve Latest Version

`make checkout`

This will update all file with latest modifications made by others on the CVS server. It will retrieve new files and directories too.

A.2. Output Documents Compilation

Generate all possible documents defined in a project

`make all [Pool=<pool_name>]`

All outputs will be stored in `Outputs/`. Optionnaly one can specify a documents pool name (defined in `conf/repository.xml`) to generate only the documents defined in that pool.

Document Compilation

`make -C manuals/<Doc_Name> <Doc_Name>.<output_format> [options...]`

will compile the whole document named `Doc_Name` in `output_format` format. Please refer to Table 3-1, for more information on supported output formats and below for more information on `[options...]`.

Module Compilation

`make -C manuals/module module MOD=<module_name> FORMAT=<output_format> LANG=<ll> EXCLUDE=`

will compile only the module named `module_name`, in the `output_format` format, in language "ll". Please refer to Table 3-1, for more information on supported output formats and below for more information on `[options...]`.

Common Compilation Options

The following options can be used with any compilation command, in place of `[options...]`, to change default compilation parameters:

LANG=language

Compilation is done for language `language`, where `language`is the two lowercase letter ISO code for the language in question.

DSSSL_JADETEX=../../drivers/alternative_stylesheet.dsssl

Uses `alternative_stylesheet.dsssl` as the DSSSL stylesheet. Note that the path `../../drivers/` is *mandatory* because of the `-C` option used with the **make** command.

A.3. Revision Management Commands

Passing a task

```
make -C modules/<ll>/ <module>.revision TYPE=<task>
```

where:

ll

is the ISO code of the languge concerned

module

Is the module name (without the .xml extension)

task

is the task that is being passed as specified in the reports (by default, one of:write, update, synch, tproof, pproof, ispell, lproof)

Assign a task

```
make -C modules/<ll>/ <module>.revision TYPE=<task>.todo AUTHOR=<initials>
```

The only changes with respect to above command is that we add `.todo` as a prefix to the task to be assigned. Additionaly, we specify the initials of the author responsibla for that task. Note that authors are defined in `conf/authors.xml`.

Assign IDs automatically

```
make -C modules/<ll>/ <module>.id
```

This command may be excpetionnally needed when an author adds content to a module after the `pproof` task has been passed. This will ensure translators add this content to translations.

Assign a task for all the modules of a specific document

```
make -C manuals/<manual> revisions [LANG=<ll>] TYPE=<task>.todo AUTHOR=<initials>
```

The options are tha same as for assigning a task to a single module (see above).

A.4. Module Edition Commands

Make a standalone module for XML editors

```
make -C modules/<ll>/ <module>.edit.noents.xml
```

Will generate in `modules/<ll>/<module>.edit.noents.xml` an XML file suitable for edition in any XML editor.

Incorporate an edited module back into Borges

```
make -C modules/<ll>/ <module>.revertedit
```

This will take edited file `modules/<ll>/<module>.edit.noents.xml` and move changes back into `modules/<ll>/<module>.xml` so that changes are taken into account by Borges. Do not forget to commit the module if neede.

Commit a module to CVS

```
make -C modules/<ll>/ <module>.commit
```

That will additionally check the module's validity before committing.

Commit an entites file to CVS

```
make -C entity/<ll>/ <entities_module_name>.commit
```

That will additionally check the entities file validity before committing the `entity/ll/entities_module_nam` file.

A.5. Reports Generation Commands

Generate the whole project report

```
make -C reports index.html
```

To be browsed from `reports/index.html`

Generate a single document report

```
make -C manuals/<manual> master-report.html
```

Will generate the report for document `<manual>` in `manuals/<manual>/master-report.html`

Generate a module's synchronization reports

```
make -C modules/<ll>/ <module>.ids.html
```

Will generate the synchronization reports for module `<module>` in language `<ll>`. Resulting reports are `modules/<ll>/<module>.ids.html` and `modules/<ll>/<module>.changes.html`. Empty if no changes.

A.6. Project Management Commands

Declare a new language to work on for the whole repository

```
make addlang NEWLANG=<ll>  NEWAUTHORS="authortask1 authortask2 authortask3 ..."
```

where:

<ll>

is the ISO code (generally two lowercase letters) corresponding to the language to be added. Make sure that code corresponds to the code used by DocBook stylesheets in `/usr/share/sgml/docbook/xsl-stylesheets/common/`.

"authortask1 authortask2 authortask3 ..."

Is the list of default authors associated to each of the tasks as defined in the modules workflow in `conf/repository.xml`.

Add a new document to the repository

```
make adddoc doc=<doc_name> master=</path/to/master.top.xml>
```

Will create a new document called `doc_name` based on the master file located at `/path/to/master.top.xml`. Beware to use absolute path. The system will take care of extracting entities from the master (if needed), and create all modules templates based on the contents of the master.

Change the repository release number

```
make release [NEWREL=10.1]
```

Bumps up the repository release number and reinitializes all workflows. if `NEWREL` is not specified, the current minor revision number is incremented (aka 9.2 becomes 9.3).

> **Caution**
>
> This command is quite heavy and will drastically change most files on the repository. Use with care. It automatically updates the CVS repository too.

Modify an existing document struture

```
make -C manuals/<manual> alltemplates
```

This is needed when one adds new modules to a document structure in `manuals/<manual>/master.top.xml`. That will generate all needed templates in all languages for the new modules. Note that this is not needed if the new modules already exists in another document.

Setting the work start date for a document in a specific language

```
make -C manuals/<manual> startwork [LANG=<ll>] [DATE=<YYYY-MM-DD>]
```

That will set the start date of work on document `literamanual` in language `ll`. If the *`DATE`* parameter is not specified, the current date is used.

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/