Ciaran McHale

Software that makes it easy to create anthologies
(and other documents) in PDF and HTML formats

Canthology User Guide

Version 1.0, 16 December 2011

www.canthology.org

Availability

This manual is available free-of-charge in the following formats:

- **HTML** for online viewing.
- **PDF** (formatted for A5 paper) for on-screen viewing.
- **Two-up PDF** for printing onto A4 paper.
- **Source** in Canthology format, in case you want to modify or update this manual.

You can find all the above on www.Canthology.org.

Copyright of the Documentation

Copyright of the Software

Copyright of Supporting Files

The `etc` directory hierarchy of this distribution of Canthology contains some files with `".sty"`, `".hva"`, `".tex"` extensions. You are likely to use

some or all of these files when writing a Canthology-based document. These files have permissive copyright licenses that should *not* hinder your ability to apply whatever copyright license you want to your own document. More specifically:

- Most of the `".sty"` files are released under version 1.3c of the LaTeX Project Public License. (A copy of this license is included in Appendix E on page 221.) The only exception is the file `hevea.sty`, which was written by Luc Maranget. As far as I am aware, he considers that file to be in the public domain.
- The `".hva"` files are released without any explicit copyright license attached to them. Please consider these to be in the public domain.
- Files with names of the form `"example-*.tex"` (where $*$ is a wildcard that matches zero or more characters) provide examples of how to carry out particular typesetting tasks. Likewise, files with names of the form `"*-template*.tex"` automate particular typesetting tasks. I release these "example" and "template" files into the public domain.

The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore, all progress depends on the unreasonable man.
— George Bernard Shaw

Canthology is dedicated to unreasonable people
who strive to make the world a better place.

Contents

Chapter 1

Introduction

1.1 What is Canthology?

Canthology is a software application that makes it easy to create an anthology of, for example, poetry, short stories or recipes. The name *Canthology* is an abbreviation of *create anthology*.

Canthology has been developed and tested on Gnu/Linux and Microsoft Windows, but has a good chance of being able to work on other operating systems if you have LaTeX, Tcl and Java installed. (Installation instructions are provided in Chapter 6.)

1.2 Uses of Canthology

The following hypothetical case studies illustrate some of the uses of Canthology.

1.2.1 A Tailor-made Anthology of Poetry for Education

John teaches a one-semester poetry course at university. His course includes the study of twenty poems by eighteenth century poets. In previous years, he required his students to purchase a specific anthology of poems, because that was the only book he knew of that contained all the poems

on his course. However, the book was unpopular with students for two reasons. First, the book was bulky and heavy: it contained hundreds of poems and was over 800 pages long. Second, the book was expensive; students did not like having to pay so much for a book when they made use of just a small fraction of its contents.

Canthology enabled John to eliminate the need for the expensive and bulky book for his course. He achieved this as follows:

1. All the poems on John's course are old enough that the copyright on them has expired. John decided to use an Internet search engine to find the text of the poems online. He quickly found the text of all twenty poems and saved them onto his own computer, each poem in a separate file.

2. John spent a few minutes editing the text of each poem, putting it into the straightforward markup format required by Canthology.

3. Then, John wrote a configuration file that instructed Canthology to combine the twenty files containing poems into an anthology, complete with a table of contents.

4. By adding a particular markup command to each poem, John arranged for the name of each poem's author to be displayed twice: once above the poem, and again in the table of contents (where it appeared underneath the poem's title).

5. The only thing missing was a title page for the anthology. Rather than create one from scratch, John decided to use a *template* title page supplied with Canthology. He updated his configuration file to instruct Canthology to replace the template's placeholders (for the book's title and author/editor) with his desired text.

6. Having edited the configuration file, John executed the `canthology` command. Within a few seconds, his anthology of poetry was created in the form of a PDF file.

Now when John teaches his course, he gives the customised anthology of poems to his students as a PDF file. Some students read it on their

laptop or tablet computers. Students who prefer physical books over electronic ones can print the anthology on a university laser printer and add the printed pages to a ring binder containing their handwritten course notes. The students are happy: printing the tailor-made anthology for the course is much cheaper than buying an 800-page book; and the tailor-made anthology is much slimmer too.

1.2.2 An Anthology of Personal Accounts

Mary is a member of a minority group that faces widespread discrimination. She wants to create an anthology of personal accounts written by other members of her minority group. She has two goals for the anthology: (1) to provide inspiration and hope for isolated members of the minority group; and (2) to educate readers from mainstream society about the prejudice faced by the minority group.

Mary finds people who are willing to contribute to the anthology. She asks each contributor to format their submissions using the simple markup format required by Canthology.

When Mary receives all the contributions, she writes a configuration file that instructs Canthology to combine the contributions into an anthology, complete with a title page and table of contents. Mary has a photograph she wants to use as a background image on the title page of the anthology. Canthology makes it easy to do this.

When the anthology of personal accounts is complete, Mary is faced with several possibilities. She could submit it to a publisher, in the hope of getting it published as a paperback book that could be stocked in shops. Alternatively, with the aid of a print-on-demand company (for example, www.lulu.com), she could self-publish the book and sell it though Amazon. Or she could instruct Canthology to convert the book into HTML format and host it on a website.

1.2.3 Showcasing the Expertise of Employees

Jane is the owner and manager of a consultancy company. Jane is confident that her employees are more highly skilled and provide better cus-

tomer service than employees in most competing companies. However, she has been struggling to find a good way to communicate this message to customers, and she is slowly losing business to inferior competitors who have a larger advertising budget or who undercut her on price.

When Jane discovers Canthology, she develops a plan. Demand for consultancy services tends to be either feast or famine. So, whenever there is a lack of consultancy work to do, Jane gets her underutilised staff to document useful tips in the form of short articles. When a dozen useful tips have been written, Jane uses Canthology to create an anthology, *Tips from the Experts*, in both PDF and HTML formats.

Jane uses the PDF file to print a booklet that is distributed to customers on her company's mailing list. Jane integrates the HTML version of the anthology into her company's website, where it soon becomes indexed by Internet search engines, thus directing potential customers to her company.

1.2.4 Product Manuals

David coordinates the writing of product manuals at a software company. He knows that although Canthology *can* associate a different author with each chapter or section, that is an optional feature of Canthology. Because of this, Canthology can be used to write a "normal" book—such as a product manual—that has only one listed author. Several features of Canthology make David realise it is especially suited for writing product manuals.

First, because Canthology-based documents are written as plain text with markup commands, his technical writers can use text-oriented tools for working with Canthology. These include: their favourite text editors; utilities such as `diff` and `grep`, and a source-code control system, such as SourceSafe, CVS, Subversion, Git or ClearCase.

Second, it is trivially easy for Canthology to produce a document in multiple output formats, including PDF files formatted for different paper sizes, and HTML. Furthermore, it is possible to write a shell script, Makefile or Ant rule to automate this process. His team now produces PDF product manuals formatted for: A4 paper for European customers, US Letter paper for American customers, A5 paper for easier viewing on

a computer screen or tablet computer, and HTML for browsing on his company's website.

While David was evaluating Canthology, he discovered that the *Canthology User Guide* is itself written as a Canthology-based document. He was able to play around with the source of this 200-page document to experiment with the ergonomics of using Canthology and verify how easy it is to produce a product manual in HTML and PDF formats, even for multiple paper sizes.

1.3 How Canthology Works

Canthology is a simplification wrapper for LaTeX (pronounced "lay-tech" or "lah-tech"), so I will start by providing a brief overview of LaTeX, and afterwards explain how Canthology simplifies the use of LaTeX.

1.3.1 A Brief Overview of LaTeX

LaTeX is a markup language for writing documents. This means a LaTeX document consists of plain text with embedded commands that specify how to *markup* (that is, format) the text. As an example, Figure 1.1 shows the start of the LaTeX file that I used for writing this chapter. I have used a **bold** font to highlight the markup commands.

Figure 1.1: LaTeX source for the start of this chapter

```
\chapter{Introduction}

\section{What is Canthology?}

Canthology is a software application that makes it easy to create an
anthology of, for example, poetry, short stories or recipes. The name
\emph{Canthology} is an abbreviation of \emph{create anthology}.

Canthology has been developed and tested on Gnu/Linux and Microsoft
Windows, but has a good chance of being able to work on other operating
systems if you have \LaTeX{}, Tcl and Java installed.
```

As you can see, the `\chapter` command starts a chapter, `\section` starts a section, `\emph` uses italics to emphasise a word or phrase, and `\LaTeX` produces the LaTeX logo. Parameters, if any, to those commands are enclosed in braces.

One benefit of LaTeX is that it enables authors to focus on the content and logical structure of what they are writing, rather than be distracted by formatting issues, such as font sizes and the amount of vertical space to leave after a chapter or sectional title.

When using LaTeX to write a book, it is common practice to write each chapter in a separate file, and then write a "root" file that uses an `\input` command for each of those chapter files. When the book is complete, LaTeX can process the root file (and all the files it inputs) to convert the book into a nicely-formatted document in, say, PDF, PostScript or HTML formats.

It is common for the *start* of a LaTeX document to contain many markup commands (for example, to specify the page dimensions and fonts to be used, and to format a nice looking title page), which can make it difficult to read and edit. However, once that initial hurdle of markup commands has been passed, the rest of the document tends to have only occasional markup commands, and thus is easy to read and edit.

1.3.2 Canthology's Approach to Simplifying LaTeX

Canthology uses a combination of techniques to put a simplification wrapper around LaTeX.

The first simplification technique employed by Canthology is to support the following clear division of labour:

- A *contributor* to an anthology is a person who writes a chapter (or perhaps a section within a chapter). Contributors have to learn how to use just a tiny subset of the markup commands provided by LaTeX. Thus, the LaTeX learning curve for contributors is minimal—typically, less than 30 minutes.

- The *editor* of an anthology needs to learn a slightly larger subset of

LaTeX markup commands, and must also learn how to use Canthology.

- A *typographer* knows how to customise LaTeX to modify the "look and feel" of the output it produces.

Most anthologies contain material from, say, ten or twenty contributors and are organised by just one editor. Thus, simplifying the LaTeX learning curve for the many contributors has a far bigger impact in simplifying the overall project than does simplifying the LaTeX learning curve for one editor. Part I of this manual discusses the very few LaTeX markup commands that will be sufficient for the needs of most contributors.

Another simplification technique relies on the *80/20 Principle*, which is also known as the *Pareto Principle*.[1] Put simply: 80% of the markup commands in a LaTeX document are concentrated in 20% of the document's content. Rather than try to simplify an *entire* LaTeX document, Canthology simplifies *just* the 20% that contains the most markup. These simplifications benefit the editor of an anthology.

The final simplification technique is that the default settings and template files supplied with Canthology are likely to provide "good enough" typography for most uses. This *eliminates* the need for a dedicated typographer. However, if an editor is not satisfied with the range of typographical choices provided by Canthology, then she can read books about LaTeX (such as those suggested in Section 9.2 on page 83) to learn more about how to make the typographical customisations herself. Alternatively, the editor could find somebody with a good knowledge of LaTeX to make the customisations for her.

[1] http://en.wikipedia.org/wiki/Pareto_principle

Part I

Information for Contributors

Introduction to Part I

If you are contributing to an anthology, then you want to focus on writing your contribution. You do *not* want to be distracted by having to adhere to complex instructions on how to format your contribution, or by having to write your contribution with an unfamiliar word processor.

Canthology tries to minimise such distractions. You can use whatever text editor you like best, such as Notepad, UltraEdit, TextMate, Vi or Emacs. Alternatively, you can use your favourite word processor and save your contribution as a plain text file. The only requirement is that you must put some *simple* markup commands in your contribution.

The two short chapters in Part I provide all the information you are likely to need to create a Canthology-compatible contribution.

Chapter 2

Basic Markup

2.1 File Name

You should write your contribution in a file that has a `".tex"` extension, for example, `what-i-did-last-summer.tex`. The name of the file should contain only letters, numbers, hyphens (`-`) and underscores (`_`).

The person who is editing the anthology may suggest a file name that you should use. For example, the editor may wish you to use your own name as the file name (`john-smith.tex`).

2.2 Markup Commands for a Short Story

You should write your contribution as a plain text file with embedded commands that specify how to *markup*, (that is, format) the text. Figure 2.1 shows an example of a short story that might be included in an anthology. The markup commands are shown in a **bold** font. Figure 2.2 shows how the story might be formatted in the final anthology.

Although the example is very short, it illustrates most of the markup commands that a contributor needs to be familiar with.

The `\section*` command takes one parameter (enclosed in braces) that specifies the name of the short story.

Figure 2.1: A short story: markup

```
\section*{What I Did Last Summer}
\sectionAuthorInfo{John Smith}{England}

Last year, I got a summer job working at a local factory.
Unfortunately, with the recession, the manager wasn't hiring anyone
this year. Most of my friends had summer jobs, so I got \emph{very}
bored hanging out by myself.

To keep myself occupied, I decided to do some volunteer work at a local
community centre. While there, I worked at a wide variety of tasks. I
swept the floor. I made refreshments during coffee breaks. I proofread
documents. I ordered new stationary supplies. Occasionally, I staffed
the reception desk. I helped people with poor literacy skills to fill
in forms. I repainted a corridor. And all that was during my
\emph{first week} there!

I volunteered at the centre five days a week, for most of the summer,
taking just a two-week break to go on holidays. It was the most
enjoyable job I've ever had.
```

Figure 2.2: A short story: formatted output

What I Did Last Summer

John Smith, England

Last year, I got a summer job working at a local factory. Unfortunately, with the recession, the manager wasn't hiring anyone this year. Most of my friends had summer jobs, so I got *very* bored hanging out by myself.

To keep myself occupied, I decided to do some volunteer work at a local community centre. While there, I worked at a wide variety of tasks. I swept the floor. I made refreshments during coffee breaks. I proofread documents. I ordered new stationary supplies. Occasionally, I staffed the reception desk. I helped people with poor literacy skills to fill in forms. I repainted a corridor. And all that was during my *first week* there!

I volunteered at the centre five days a week, for most of the summer, taking just a two-week break to go on holidays. It was the most enjoyable job I've ever had.

The `\sectionAuthorInfo` command takes two parameters: the first specifies the name of the contributor, and the second specifies some background information, such as the contributor's location, age or occupation.

Finally, the `\emph` command is used to emphasise a word or phrase, and one or more blank lines are used to separate paragraphs.

By comparing Figures 2.1 and 2.2, you may also notice that (aside from blank lines), it does not matter where you put line breaks in an input file: the LaTeX typesetting system (upon which Canthology is based) will decide the optimal place for lines breaks in the output file.

2.3 Markup for a Chapter-length Contribution

In the previous section, I explained how to format your short story as a *section* (within a chapter). However, if the editor instructs you to submit your contribution as an entire *chapter*, then you should use the `\chapter` and `\chapterAuthorInfo` commands. For example:

```
\chapter{What I Did Last Summer}
\chapterAuthorInfo{John Smith}{England}

Last year, I got a summer job ...
```

If you want to divide your chapter into several sections, then you can use one of the following commands:

```
\section*{title}
\section{title}
\anonymoussection
```

You have already seen the `\section*` command: it formats the *`title`* text *without* a sectional number.

The `\section` command formats the *`title`* text *with* a sectional number. For example, when writing this chapter, I used the following command at the start of this section:

```
\section{Markup for a Chapter-length Contribution}
```

You can use the `\anonymoussection` command to start a new section but not assign a title for it. For example:

```
Then, in the rudest manner he could display, he turned
his back deliberately on Sandeman and walked out of
the room.

\anonymoussection

In the company of Adela he tried to forget the little
contretemps.
```

produces the following output:

> Then, in the rudest manner he could display, he turned his back deliberately on Sandeman and walked out of the room.
>
> * * * * *
>
> In the company of Adela he tried to forget the little contretemps.

As you can see, the `\anonymoussection` command uses blank space and a row of $*$ symbols to mark the end of one section and the start of another.

2.4 Markup Commands for Poetry

The example in Figure 2.3 illustrates the markup you should use for a poem, and Figure 2.4 shows the formatted output. (By the way, this example uses `--` and `---` to produce dashes of various lengths; such dashes will be discussed in Section 3.3 on page 20.)

You use the `\poemtitle` command to specify the title of the poem, and `\poemAuthorInfo` to specify both the name of the contributor and some background information about that contributor, such as location, occupation or age.

Figure 2.3: A poem: LaTeX markup

```
\poemtitle{Carpe Diem}
\poemAuthorInfo{William Shakespeare}{1564--1616}

\begin{verse}[16em]
O mistress mine, where are you roaming? \\*
O stay and hear! your true-love's coming \\*
That can sing both high and low; \\
Trip no further, pretty sweeting, \\*
Journey's end in lovers' meeting--- \\*
Every wise man's son doth know.

What is love? 'tis not hereafter; \\*
Present mirth hath present laughter; \\
What's to come is still unsure: \\*
In delay there lies no plenty,--- \\*
Then come kiss me, Sweet and twenty, \\*
Youth's a stuff will not endure.
\end{verse}
```

Figure 2.4: A poem: formatted output

Carpe Diem

William Shakespeare, 1564–1616

O mistress mine, where are you roaming?
O stay and hear! your true-love's coming
That can sing both high and low;
Trip no further, pretty sweeting,
Journey's end in lovers' meeting—
Every wise man's son doth know.

What is love? 'tis not hereafter;
Present mirth hath present laughter;
What's to come is still unsure:
In delay there lies no plenty,—
Then come kiss me, Sweet and twenty,
Youth's a stuff will not endure.

The poem itself is typeset in the `verse` environment, that is, between the `\begin{verse}` and `\end{verse}` commands. Within a `verse` environment, you should do the following:

- Leave a blank line between each stanza.
- At the end of each line—*except* the last line of a stanza—use the `\\*` command. This forces a line break in the formatted output, while preventing a page break at that line.
- If you feel it is acceptable to have a page break in mid-stanza, then you can use the `\\` command instead of `\\*`. For example, Figure 2.3 uses `\\` when a line ends with a semicolon.

  Note that using `\\` does *not* force a page break at that point; it merely indicates that a page break is acceptable.

There is one last point to note about the typesetting of a poem. The `\begin{verse}` command can take an optional parameter, which, if used, is enclosed in square brackets. You can see an example of this in Figure 2.3: `\begin{verse}[16em]`. The optional parameter specifies a size, which can be specified in a variety of units, including point (`pt`), millimetre (`mm`), centimetre (`cm`), inch (`in`), ex (`ex`) or em (`em`).[1]

If you do *not* specify the optional parameter to the `verse` environment, then the poem will be typeset slightly indented from the left-side margin. For example:

> O mistress mine, where are you roaming?
> O stay and hear! your true-love's coming
> That can sing both high and low;
> ...

If you *do* specify an optional size parameter, then the `verse` environment pretends that the length of each line in the poem is the specified size, and

[1] A point is $1/72.27$ inch. An ex is roughly the height of the letter "x" in the current font. An em is roughly the width of an "M" in the current font, which is often equal to the font size. For example, if the font size is 10pt, then 16em is equivalent to 160pt, but if you change the font size to be 12pt, then 16em will be equivalent to 192pt. People who design document layouts sometimes like to express sizes in ex or em units because this enables the sizes to be scaled automatically if the font size is changed.

centres the poem on the page based on that pretended line length, as shown in Figure 2.4. As a contributor, you should not worry about specifying an accurate value for the optional parameter to the `verse` environment. (If you want, just omit the optional parameter and its square brackets.) If the editor of the anthology decides to centre poems, then she can use a trial-and-error approach with a document previewer to choose a suitable value for the optional parameter to the `verse` environment.

You might want your poem to be formatted in a very specific way, for example, perhaps every second line of the poem should be indented. Arguably, such formatting decisions should be made by the editor of an anthology, rather than by each individual contributor, so there can be consistency of formatting across all poems in the anthology. However, if you feel strongly that your poem should be formatted in a particular way, then you should contact the anthology's editor to express your wishes.

2.5 Special Characters

The markup commands used in Canthology come from the LaTeX typesetting system, upon which Canthology is based. The following characters have special meanings to LaTeX:

```
# $ % ~ _ ^ \ & { }
```

If you need to embed any of those characters in your contribution, then you should use the appropriate command shown in the table below.

Command	Output
\#	#
\$	$
\%	%
_	_
\^{}	^
\&	&
\{	}
\}	{
\textbackslash{}	\
\textasciitilde{}	~

For example:

```
I can see the \# symbol on my telephone keypad.
```

produces the following output:

I can see the # symbol on my telephone keypad.

As you can see from the table, *most* of the special symbols can be typeset by preceding them with \. The most notable exception to that rule is the \ character itself. If you want to use that character in a document, then use \textbackslash{} (the {} indicates that this command does not take a parameter). If you accidentally use \\ instead, then you will force a line break in the output. For example:

```
This is a \backslash{}, but this is a line\\break.
```

produces the following output:

This is a \, but this is a line
break.

The % character starts a comment that continues until the end of the line. Comments are *not* copied into the output document. For example, the following example accidentally uses % instead of \%:

```
Pareto found that 80% of the wealth in Italy was
owned by just 20% of the population.
```

so it produces the following incorrect output:

> Pareto found that 80owned by just 20

The fix is to replace `%` with `\%` in the input document:

```
Pareto found that 80\% of the wealth in Italy was
owned by just 20\% of the population.
```

so it produces the following correct output:

> Pareto found that 80% of the wealth in Italy was owned by just 20% of the population.

Chapter 3

More Markup Commands

3.1 Introduction

The previous chapter introduced a few markup commands that will be sufficient for most contributions to an anthology. This chapter discusses some more LaTeX markup commands that are less likely to be required by contributors.

3.2 Single and Double Quotes

Your keyboard is likely to have three quote symbols: the open single quote (`), the close single quote(') and the double quote ("). You should *not* use the double quote character when writing a LaTeX document. Instead, use two open quote characters (``) to obtain an open double quote (“) and use two close quote characters ('') to obtain a close double quote (”). For example:

```
Some people like `single quotes',
but others prefer ``double quotes''.
```

produces the following output:

> Some people like ‘single quotes’, but others prefer “double quotes”.

3.3 Hyphens and Dashes

You should use the - character when writing a hyphenated word. For example:

```
My sister-in-law saw a man-eating shark in a low-budget
movie.
```

produces the following output:

My sister-in-law saw a man-eating shark in a low-budget movie.

If you are writing a time period, a number range or a telephone number, then you should use `--`. LaTeX typesets this as an *en dash*, which is slightly longer than a hyphen. For example:

```
The years 2002--2006 were happy ones.
This house can accommodate 4--6 people.
My telephone number is 555--26--85--93.
```

produces the following output:

The years 2002–2006 were happy ones. This house can accommodate 4–6 people. My telephone number is 555–26–85–93.

A long dash used for punctuation is called an *em dash*. You use `---` to obtain an em dash. For example:

```
You are the person---the only person---who helped me.
```

produces the following output:

You are the person—the only person—who helped me.

3.4 Footnotes

You can use the `\footnote` command to insert a footnote. It takes one parameter that specifies the text of the footnote. For example:

```
A gnu\footnote{Also known as a wildebeest.} is a type of
animal.
```

produces the following output:

A gnu[1] is a type of animal.

3.5 The quote Environment

A LaTeX construct of the form `\begin{name}...\end{name}` is called a *name* environment. One such construct is the `verse` environment, which I discussed in Section 2.4 on page 13. Another is the `quote` environment, which you can use to quote short pieces of text. For example:

```
In his autobiography, Nelson Mandela writes:
\begin{quote}
No one had ever suggested to me how to go about
removing the evils of racial prejudice, and I had
to learn by trial and error.
\end{quote}
Most people who want to bring about significant
social change face a similar problem: many skills
they need to do so are not taught in schools or
universities.
```

produces the following output:

In his autobiography, Nelson Mandela writes:

> No one had ever suggested to me how to go about removing the evils of racial prejudice, and I had to learn by trial and error.

Most people who want to bring about significant social change face a similar problem: many skills they need to do so are not taught in schools or universities.

[1] Also known as a wildebeest.

By the way, LaTeX ignores excess spaces before and between words. This is useful, because it means you can indent the contents of a `quote` environment to better show the structure of a document. For example, the previous example could be written with some indentation as shown below:

```
In his autobiography, Nelson Mandela writes:
\begin{quote}
    No one had ever suggested to me how to go about
    removing the evils of racial prejudice, and I had
    to learn by trial and error.
\end{quote}
Most people who want to bring about significant
social change face a similar problem: many skills
they need to do so are not taught in schools or
universities.
```

This indentation used inside the `quote` environment does *not* affect the output produced.

3.6 The `itemize` and `enumerate` Environments

You can use the `itemize` environment to create a bullet-point list. Each item in the list is started with the `\item` command. For example:

```
Our priorities are as follows:
\begin{itemize}
\item Relax and have a good time.
\item Find somebody to love.
\item Earn enough money to pay the bills.
\end{itemize}
```

produces the following output:

Our priorities are as follows:

- Relax and have a good time.

- Find somebody to love.
- Earn enough money to pay the bills.

The `enumerate` environment works almost identically to the `itemize` environment, except that each item in the list is prefixed with a number rather than a bullet point. For example:

```
Our priorities are as follows:
\begin{enumerate}
\item Relax and have a good time.
\item Find somebody to love.
\item Earn enough money to pay the bill.
\end{enumerate}
```

produces the following output:

Our priorities are as follows:

1. Relax and have a good time.
2. Find somebody to love.
3. Earn enough money to pay the bill.

3.7 Next Steps

When you have finished writing your contribution, you should send it to the editor of the anthology. Do not worry about the possibility of making mistakes in the markup commands you have used. When the anthology's editor runs your contribution through LaTeX, she will see error messages for incorrect use of markup commands, so she will be able to identify and fix the errors.

Part II

Background Information for Editors

Introduction to Part II

If you are going to be the editor of an anthology, then you will have to learn how to use Canthology. However, you first need to know a little about two underlying technologies: LaTeX and Config4*. The two chapters in Part II provide the relevant information.

Chapter 4

LaTeX History and Variants

4.1 Introduction

Although Canthology provides a simplification wrapper around LaTeX, Canthology does not hide LaTeX completely. Because of this, you will need to be familiar with some LaTeX concepts and terminology. This chapter explains some LaTeX issues that will make it easier to understand how Canthology works.

4.2 A Short History of TeX and LaTeX

Within the printing industry, *type* means a printed character, and *to typeset* means to combine individual types to form words, lines and (eventually) a complete page.

In 1977, a mathematician and computer scientist called Donald Knuth was frustrated at the poor-quality typesetting of a book he had written. He decided to design his own typesetting system so his future books could look better. He spent more than a decade designing and implementing a computer program called TeX (usually pronounced "tech" by English speakers) for typesetting documents.

TeX was developed at a time when there was no standardisation of how to control different types of printers. Knuth worked around this problem

with a two-step approach.

- A person writes a TEX-based document in a file called, for example, `my-document.tex`, and processes this with TEX by executing the command:

  ```
  tex my-document.tex
  ```

  Doing that produces a file called `my-document.dvi`. The `".dvi"` extension stands for "device independent".

- Another program is executed to convert `my-document.dvi` into the format required to drive a particular type of printer.

This two-step approach has enabled TEX to survive technological changes. For example, when computers with high-resolution displays were introduced, on-screen previewers for DVI files were developed. When the PostScript printer-control language was invented, DVI-to-PostScript converters were developed. And when PDF was invented, DVI-to-PDF converters were developed.

Although TEX is powerful and can produce beautiful-looking documents, many people find it difficult to use because the TEX markup commands are low level. Over the years, this has resulted in attempts by different people to implement simplification wrappers around TEX. The most popular of these is called LATEX, which was initially developed during the 1980s. LATEX provides high-level markup commands—such as `\chapter` and `\section`—that are easier to use than the more primitive markup commands provided by TEX.

Processing a LATEX-based document is similar to processing a TEX-based document, except that you use the `latex` command instead of `tex`. For example, the following commands convert `my-document.tex` into `my-document.dvi` and then into `my-document.ps` (that is, a PostScript file):

```
latex my-document.tex
dvips my-document.dvi
```

4.3 Structure of a LaTeX Document

Figure 4.1 shows the high-level structure of a book written in LaTeX.

Figure 4.1: Structure of a LaTeX document

```
\documentclass[12pt,oneside]{book}

\usepackage{color}
\usepackage[english]{babel}
... % Other preamble commands go here

\begin{document}
    \frontmatter
    \input{titlepage.tex}
    \input{copyright.tex}
    \tableofcontents
    \input{preface.tex}

    \mainmatter
    \input{chapter-1.tex}
    \input{chapter-2.tex}
    \input{chapter-3.tex}
    \input{chapter-4.tex}
    \appendix
    \input{appendix-a.tex}
    \input{appendix-b.tex}

    \backmatter
    \input{glossary.tex}
\end{document}
```

The file starts with a `\documentclass` command that specifies the document's *class* (that is, *type*) is a book. This causes LaTeX to load a file called `book.cls` and execute its contents. Doing that causes commands such as `\part`, `\chapter` and `\section` to be defined. The `book.cls` file also defines typographical rules for typesetting a book. By default, those rules specify, among other things, that the document is typeset in a 10 pt font for two-sided printing on US Letter-size paper. The optional arguments (indicated between `[` and `]`) override some of those defaults by specifying use of a 12 pt font and one-sided printing.

The standard distribution of LaTeX contains several class files, including book.cls, report.cls, article.cls and letter.cls. This makes it possible to use LaTeX to create several types of document easily.

The term *package* refers to a file that defines add-on commands for LaTeX. Package files have a ".sty" extension (because package files were originally known as *style* files). The \usepackage command instructs LaTeX to load the ".sty" file specified as a parameter and execute its contents. For example, \usepackage{color} instructs LaTeX to load the file color.sty. Options, if any, to a package are specified between [and].

Preamble refers to the part of a LaTeX file between the \documentclass and \begin{document} commands. As shown in Figure 4.1, the preamble can contain \usepackage commands. The preamble can contain other types of commands too. Some people define a few additional commands in the preamble; but if a lot of new commands are to be defined, then it is best to put them in a package.

The document environment (everything between \begin{document} and \end{document}) contains the actual text of the document. If the document is just a few pages long, then its text might be written directly between \begin{document} and \end{document}. But, as shown in Figure 4.1, multi-chapter documents are often written using a separate file for each chapter, in which case a series of \input commands is used to read and process those separate files.

If the document is a book, then its contents can be grouped into *front matter*, *main matter* and *back matter*. Only chapters and sections in the main matter are numbered. Unnumbered chapters sometimes found in the front matter include *Acknowledgements*, *Preface* or *Foreword*. Those found in the back matter might include *Glossary*, *Index* and *Bibliography*.

Any main-matter chapter or section appearing after \appendix has its title typeset as an appendix rather than as a chapter or section.

4.4 The memoir Class

LaTeX offers the benefit of being significantly easier to use than TeX. However, it also suffers from a drawback: the predefined document classes

(book, article, and so on) adopt a "take it or leave it" attitude to typesetting. For example, users do not have an easy way to adjust the typesetting of chapter or section headings.

Partially to overcome this drawback, Peter Wilson implemented the memoir class. This class provides users with commands to adjust many aspects of document typesetting. Another benefit of the memoir class is that it incorporates the functionality of dozens of popular packages. This means a memoir-based document tends to have fewer \usepackage commands cluttering up its preamble than a similar book-based document. More importantly, it means that documentation for the functionality of many packages can be centralised in the manual for the memoir class [14] rather than be scattered over dozens of short documents (a separate document for each package).

The functionality of the memoir class is a superset of that of the book class. Thus, if you start writing a document using the book class and eventually discover it is too inflexible to suit your needs, then you should be able to switch to using the memoir class without having to make anything more than (at most) a few trivial changes in your existing document.

4.5 Support for Colour and Graphics

The initial development of TeX predated widespread support for colour in computers and printers. It also predated most graphic-file formats, such as EPS, TIFF, GIF, JPEG, PDF and PNG. Despite this, TeX and LaTeX provide good support for the use of colour and graphics in documents. This is because Donald Knuth had the foresight to provide a \special command in TeX. This command writes its argument directly into the DVI file being generated, and it is then up to a DVI-to-whatever converter to interpret that argument. This provides an open-ended extension mechanism that permits TeX (and LaTeX) to support, among other things, the use of colour and graphics in documents. The \special command is also used to implement hypertext links when a document is converted into PDF format.

Over the years, individuals have developed DVI-to-whatever converters independently of each other. The independent nature of their work has

resulted in some DVI converters expecting the argument to a `\special` command to formatted one way, and other DVI converters expecting the argument to be formatted a different way. Obviously, this has the potential to result in widespread incompatibilities. However, a combination of two things save the day.

First, DVI converters *ignore* any `\special` commands that they do not understand. This allows the conversion of documents containing unsupported `\special` commands to degrade gracefully.

Second, packages that provide a simplification wrapper around the low-level `\special` command also protect document authors from the `\special` command's potential to introduce incompatibilities. This can be illustrated with the `graphicx` package [3], which provides commands to scale and rotate text, and to include graphic files. An option to this package specifies which DVI converter you plan to use. For example:

```
\usepackage[dvips]{graphicx}
```

One of the commands provided by this package is `\includegraphics`, which you can use to include (and, optionally, scale) a graphics file:

```
\includegraphics[scale=0.7]{my-diagram}
```

That command executes the `\special` command appropriate for the DVI converter specified in the `\usepackage` command. If the document's author switches from `dvips` to using another DVI converter, then changing the option to the `\usepackage` command will be sufficient to change the `\special` commands executed by `\includegraphics`.

The main irritation I have when using graphics in a LaTeX document is that each DVI converter supports a *subset* of graphic-file formats. For example, I used to use on-screen DVI previewer that could process only `".eps"` files, while the DVI converter I used to produce a PDF file could process `".jpg"`, `".png"` and `".pdf"` files. The lack of overlap in graphic file formats supported by the two converters meant I had to provide the same graphic in several file formats, for example, `my-diagram.eps` and `my-diagram.jpg`. If you encounter this issue, then there are several ways to deal with it.

First, some drawing editor applications can save a graphic in a variety of file formats.

Second, *ImageMagick*[1] is a collection of command-line utilities for manipulating graphic files. Its `convert` utility can read a graphic file in one format and convert it to another file format. For example:

```
convert my-diagram.eps my-diagram.jpg
```

Finally, you could decide to forego use of a DVI previewer and instead use PDF documents for both on-screen previewing and printing.

4.6 Variations of TeX and LaTeX

Over the years, alternative implementations of TeX (and LaTeX) have been developed to adapt to changes in technology.

4.6.1 pdfTeX and pdfLaTeX

pdfTeX (and pdfLaTeX) can generate a PDF file *directly* from a `".tex"` file. For example:

```
pdflatex my-document.tex
```

Many people find this to be more convenient than the two-step process of generating a `".dvi"` file, and then converting this into a `".pdf"` file. The out-of-the-box configuration for Canthology uses `pdflatex`.

4.6.2 XeTeX and XeLaTeX

> The nice thing about standards is that there are so many of them to choose from. Furthermore, if you do not like any of them, you can just wait for next year's model.
> — Andrew S. Tanenbaum

[1] www.imagemagick.org

This has certainly been true for character sets. In brief, a (coded) *character set* is a convention in which numbers are used to denote characters (that is, letters, digits, punctuation, and so on). For decades, there were almost as many character sets as there were countries in the world, and this made it difficult to write, say, a Greek document on a French computer, or vice versa. The lack of an international, standardised character set posed difficulties for writing TEX- or LATEX-based documents that contained, say, accented characters such as á and ö. A two-part approach was used:

1. Commands such as `\'{a}` and `\"{o}` were used in a `".tex"` file to represent accented characters like á and ö. This made it difficult for a person to write non-English text, and also made it difficult for a spell checking program to verify the spelling of words in a `".tex"` file.

2. The output file (in, say, DVI, PostScript or PDF format) would "fake" an accented letter by drawing an *unaccented* letter, and then drawing an accent character over it. Such "fake" accented characters made it difficult, if not impossible, to search through a document for text containing an accented character.

Unicode is a standardised, universal (coded) character set that has been gaining popularity since its introduction in the early 1990s. XeTEX (and XeLATEX) is an redesigned implementation of TEX (and LATEX) that provides built-in support for Unicode. This support eliminates the problems associated with TEX's traditional two-part approach for dealing with accented characters. The `".dvi"` file format does not support the use of Unicode characters, so the designers of XeTEX have defined their own output file format, which has a `".xdv"` extension. An XDV-to-PDF converter is available but on-screen previewers for XDV files are not yet widely available.

4.6.3 Generating HTML from LATEX

The author of a document might want to provide the document as a PDF file for printing, and also as a collection of HTML pages that can be browsed casually on a website. To satisfy this need, several people have

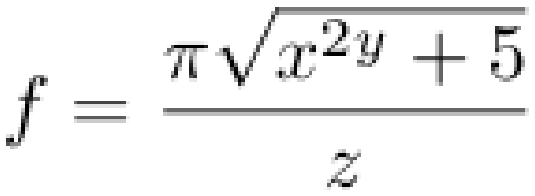
(independently) developed LaTeX-to-HTML translators. Unfortunately, *all* LaTeX-to-HTML translators have limitations. This is unavoidable because HTML is not as powerful as LaTeX, particularly when it comes to typesetting mathematical formulas.

For example, consider the following formula:

$$f = \frac{\pi\sqrt{x^{2y} + 5}}{z}$$

That formula is simple enough that a LaTeX-to-HTML translator should be able to translate it into HTML that displays an accurate representation. However, if you write increasingly complex formulas in LaTeX, then there will come a point where a formula *cannot* be translated into HTML that displays an accurate representation—simply because of limitations in HTML. At this point, rather than generate HTML that displays an inaccurate representation of the formula, a LaTeX-to-HTML converter might use LaTeX to typeset the formula, and then use a utility to render the output as a graphic image. This graphic image is then displayed in a HTML page. The resulting HTML page may look nice. But, if a user "zooms in" to display the web page's text in a larger font size, then the graphic image of the mathematical formula will also be magnified, which will result in it appearing pixelated, as shown in Figure 4.2.

Figure 4.2: A pixelated mathematical formula



Another common limitation is that a LaTeX-to-HTML converter will know how to process built-in LaTeX commands plus the commands defined in a small number of well-known packages, but will not be able to process LaTeX documents that make use of arbitrary packages. Some LaTeX-to-HTML converters permit users to write code to "teach" the converter about commands defined in other packages. Unfortunately, there is

no standardisation across LATEX-to-HTML converters of how to write such teaching code.

Chapter 5

Overview of Configuration Syntax

5.1 Introduction

You control the behaviour of Canthology by editing entries in its configuration file. I will explain the meaning of the entries in the Chapter 7. But first (in this chapter), I will provide an overview of the *syntax* used in the configuration file. This overview will be sufficient to get you started using Canthology. Later on, you might want to read Appendix B to get complete details of the syntax.

By the way, the configuration syntax used by Canthology is called *Config4** (pronounced "config for star").[1] This syntax is not specific to Canthology, so you may see it being used in some other software applications.

5.2 Syntax

Figure 5.1 on the next page provides a simple example of a Config4* configuration file.

Comments, like the one shown in line 1, start with # and continue until the end of the line. Most of the lines in a configuration file contain assign-

[1] www.config4star.org

Figure 5.1: Example configuration file

```
 1  # this is a comment
 2  name = "Fred";
 3  greeting = "hello, " + name;
 4  some_names = ["Fred", "Mary", "John"];
 5  more_names = ["Sue", "Ann", "Kevin"];
 6  all_names = some_names + more_names;
 7  application.defaults {
 8      timeout = "2 minutes";
 9      log {
10          dir = "C:\foo\logs";
11          level = "0";
12      }
13  }
14  one_application {
15      @copyFrom "application.defaults";
16      log.level = "1";
17  }
18  another_application {
19      @copyFrom "application.defaults";
20      timeout = "30 seconds";
21  }
```

ment statements. These are of the form *name*=*value*, where the *value* can be a string (line 2) or a list of strings (line 4). You can use the + operator to concatenate both strings (line 3) and lists (line 6). Assignment statements are terminated with a semicolon.

String values are usually written as a sequence of character enclosed within double quotes, for example, `"Fred"`. Within such a string, `%` acts as an escape character. For example, `%n` denotes a newline character, and `%"` denotes a double quote.[2]

A configuration file can contain named *scopes* (lines 7, 9, 14, and 18 in Figure 5.1). Scopes can be nested (line 9) and re-opened. The scoping operator is what some people call a *full stop*, and others call a *period* or a *dot*. For example, the name `log.level` refers to a variable called `level` inside a scope called `log`. You do not have to explicitly open a scope to define a variable or a nested scope within it. For example, line 7 opens the

[2] An alternative way to write string values is discussed in Appendix B.3 on page 173.

`server.defaults` scope without opening the outer `server` scope. Likewise, line 16 defines `log.level` without explicitly opening the `log` scope.

5.3 Copying Default Values

All keywords (for example, `@include` and `@copyFrom`) start with the `@` symbol: this ensures there can never be a clash between the name of a keyword and the name that you might wish to use for a configuration variable or scope.

The `@copyFrom` statement (lines 15 and 19 in Figure 5.1) copies the entire contents (*name=value* pairs and nested scopes) of the specified scope into the current scope. This provides a simple, yet effective, reuse mechanism. It is *not* an error to assign a new value to an existing variable. This makes it possible to override default values obtained via a `@copyFrom` statement.

5.4 Including Other Files

An `@include` statement (not shown in Figure 5.1) includes the contents of another configuration file into the current one. For example:

```
@include "/tmp/foo.cfg";
```

5.5 Accessing Environment Variables

You can access environmental information in a configuration file. For example, you can use `getenv("CANTHOLOGY_HOME")` to access an environment variable called `CANTHOLOGY_HOME`.

```
install_dir = getenv("CANTHOLOGY_HOME");
```

Part III

Using Canthology to Generate PDF Files

Introduction to Part III

The editor of an anthology needs to know more about Canthology than do the contributors, although the burden of knowledge is still quite slight. This part of the manual explains everything that an editor is likely to need to do when using Canthology to produce PDF documents. Later, in Part IV, I will explain some additional information you will need to know if you want to use Canthology to create HTML documents.

Chapter 6

Installing Canthology

6.1 Prerequisites for Installing Canthology

You should ensure that LaTeX and Java are on your computer before you install Canthology. The combination of LaTeX, Java and Canthology will enable you to generate PDF documents.

If, in addition, you want to use Canthology to generate HTML documents, then you will also need to ensure that you are using a UNIX/Linux computer and have HeVeA and Tcl installed. (Canthology does not support the ability to generate HTML documents on Windows.)

LaTeX distributions are available, free-of-charge, for many operating systems, including Windows, Mac OS X and UNIX/Linux. Details of these distributions can be found at www.latex-project.org/ftp.html.

Java is pre-installed on many computers. To check if you already have Java on your computer (and, if so, which version of Java it is), open a command/shell window and execute the following command:

```
java -version
```

You need Java 1.3 or later to run Canthology. If you do not have a recent-enough version of Java on your computer, then you can download Java from www.java.com.

HeVeA is included in some distributions of LaTeX. You can determine if HeVeA is installed on your computer by opening a shell window and

executing the following command:

```
hevea non-existent-file
```

If you see an error message complaining that `non-existent-file` does not exist, then HEVEA is installed. If HEVEA is *not* installed, then you can download it from http://hevea.inria.fr.

To check if Tcl is installed on your computer, open a shell window and execute the following command:

```
tclsh non-existent-file
```

If you see an error message complaining that `non-existent-file` does not exist, then Tcl is installed. If Tcl is *not* installed, then you can download it from www.tcl.tk/software/tcltk/platforms.html.

6.2 Installing Canthology

Canthology is available, free-of-charge, from www.canthology.org. You can install Canthology with the following steps:

1. Unzip the Canthology distribution into a directory.
2. Set the `CANTHOLOGY_HOME` environment variable to the name of this directory. For example:

```
CANTHOLOGY_HOME=$HOME/canthology          (UNIX)
export CANTHOLOGY_HOME
set CANTHOLOGY_HOME=C:\canthology         (Windows)
```

3. Add the `bin` subdirectory of the Canthology installation to your `PATH` environment variable. For example:

```
PATH=$PATH:$CANTHOLOGY_HOME/bin           (UNIX)
export PATH
set PATH=%PATH%;%CANTHOLOGY_HOME%\bin     (Windows)
```

Chapter 7

A Tour of Canthology's Features

7.1 Introduction

In this chapter, I provide a brief tour of Canthology's features, so you can see how Canthology makes it easy to create professional-looking books. By the way, this is the longest chapter in the manual. Do not be put off by that, because more than half the content of this chapter consists of full-page diagrams that illustrate the output produced by Canthology.

7.2 The Starting-point Configuration File

The operation of Canthology is controlled by a configuration file that defines approximately 20 variables. It would be rather tedious if you had to assign suitable values for all of these variables before you could run Canthology. Thankfully, you don't have to. This is because Canthology can generate a starting-point configuration file that provides you with sensible default values for most of the variables. This significantly reduces the time required to start using Canthology.

The following command instructs Canthology to create a starting-point configuration file called `Canthology.cfg`:

```
canthology -create Canthology.cfg
```

The contents of this file are shown in Figure 7.1.

Figure 7.1: Starting-point configuration file

```
 1  @include getenv("CANTHOLOGY_HOME") + "/etc/defaults.cfg";
 2  anthology1 {
 3      @copyFrom "book:a4";
 4      root_file {
 5          preamble = preamble + [
 6          ];
 7          base_name = "my-anthology" + macro.paperSizeSuffix;
 8          front_matter = [
 9              "\input{titlepage-template-1.tex}",
10              macro.tableofcontents,
11          ];
12          main_matter = [
13          ];
14          back_matter = [
15          ];
16      }
17      substitutions.search_replace_pairs = [
18          # search string          replace string
19          #-----------------------------------------
20      ] + substitutions.search_replace_pairs;
21  }
```

7.2.1 Default Values and Paper Sizes

Line 1 in the starting-point configuration file includes another file that contains many scopes,[1] each of which provides sensible default values. (The `@copyFrom` statement in line 3 copies default values from one of those scopes.) The names of the scopes in the included configuration file can be constructed from information in Table 7.1 on the next page by concatenating the name of a column with a colon and the name of a row. Doing that yields scope names like `book:letter`, `book:a5`, `report:a4`, and so on.

Most of the columns (`book`, `report`, `article`, and so on) in Table 7.1 are names of popular LaTeX document classes. The only exception is

[1] Recall from Section 5.2 on page 36 that a scope is a construct of the form `"name { ... }"`.

Table 7.1: Scopes in `etc/defaults.cfg` with names of the form *`column:row`*

	book	report	article	memoir	memoir-article
letter	✓	✓	✓	✓	✓
a4	✓	✓	✓	✓	✓
a5	✓	✓	✓	✓	✓
a5-trimmed	✓	✓	✓	✓	✓
html-one-page	✓	✓	✓		
html-many-pages	✓	✓	✓		

`memoir-article`, which uses the `memoir` class, but adds some customisation to make that class suitable for writing articles.[2]

Most of the row names denote paper sizes: US Letter (`letter`) paper is commonly used in America, while A4 (`a4`) is commonly used in Europe and some other parts of the world.

A5 (`a5`) paper, which is half the size of A4 paper, is slightly larger than the size of a paperback novel. When writing a document that I expect will be read mostly on a computer screen rather than on paper, I format it for A5 paper because: (1) the small size makes it easy to read text on a computer screen without having to squint; and (2) two A5 pages can be printed size-by-side on a sheet of A4 paper, thus saving on paper.

The `a5-trimmed` row is similar to `a5` except that most of the margins have been trimmed off. The intended use is to format documents for reading on a tablet computer (such as the Apple iPad or an Android device). Trimming unnecessary margins means the text of the document can be scaled up about 10% on the screen, thus making it easier to read.

The first four rows in Table 7.1 instruct Canthology to create a PDF document for the specified paper size.

The last two rows (`html-one-page` and `html-many-pages`) instruct Canthology to create a HTML document, either as a monolithic HTML page or as a collection of HTML pages. I will discuss the generation of HTML documents in Part IV. Unfortunately, the HEVEA tool used to convert LaTeX documents into HTML does not support use of the `memoir` document class.

[2] The `memoir-article` scope adds the following two statements to the preamble of a document:
`\counterwithout{section}{chapter}` `\pagestyle{plain}`

7.2.2 Output Directory and File Name

Each of the scopes in the `etc/defaults.cfg` configuration file defines variables, including `working_dir` and `macro.paperSizeSuffix`. Table 7.2 shows the values assigned to these variables in the various scopes. For example, configuration scopes containing `"a4"` in their name (`book:a4`, `report:a4`, and so on) assign the value `"output-pdf-a4"` to `working_dir` and the value `"-a4"` to `macro.paperSizeSuffix`.

Table 7.2: Working directories and paper sizes for configuration scopes

	working_dir	macro.paperSizeSuffix
letter	"output-pdf-letter"	"-letter"
a4	"output-pdf-a4"	"-a4"
a5	"output-pdf-a5"	"-a5"
a5-trimmed	"output-pdf-a5-trimmed"	"-a5-trimmed"
html-one-page	"output-html"	""
html-many-pages	"output-html"	""

The `working_dir` directory specifies the directory into which Canthology will create and copy files. The `macro.paperSizeSuffix` variable can be used to incorporate the paper size of a document into its file name.

Let's resume our examination of the starting-point configuration file in Figure 7.1 on page 44. As I previously stated, line 1 includes another configuration file that provides 26 scopes, each of which contains sensible default values.

Line 2 opens a configuration scope called `anthology1` (the name of this scope is not important, so you can change it if you wish), and line 3 copies default values from the `book:a4` scope into it.

As you can see in Table 7.2, the `book:a4` scope assigns the value `"output-pdf-a4"` to the `working_dir` variable. Thus, the PDF file created by Canthology will be placed in a subdirectory called `output-pdf-a4`. Within that subdirectory, the name of the PDF file created by Canthology is obtained by appending `".pdf"` to the value of the `root_file.base_name` variable, which is defined on line 7 of Figure 7.1. Thus, the generated PDF file will be `output-pdf-a4/my-anthology-a4.pdf`.

If you run Canthology several times on the configuration file, each time changing the name of the scope in line 3, then you will end up with the same document formatted for different sizes of paper. This can be useful if you intend to make the document available as a download from a website: Europeans will choose the A4-formatted version of the document, while Americans will prefer the US Letter formatted version.

7.2.3 Front, Main and Back Matter

As I explained in Section 4.3 on page 28, LaTeX considers the contents of a book (technically, this means use of the `book` or `memoir` document classes) to be grouped into "front matter, "main matter" and "back matter". The `front_matter` (lines 8–12), `main_matter` (lines 13–14) and `back_matter` (lines 15–16) variables are lists that specify text or commands to be added to those parts of the document. Most of the commands are likely to be `\input` commands to add contributions into the anthology.

The string on line 9 instructs Canthology to `\input` the file `titlepage-template-1.tex` at the start of the book. You do *not* need to create this file, because it is provided in the Canthology distribution. (Later, in Section 8.3.2 on page 72, I will explain how Canthology searches for this file.) Line 10 makes use of the `macro.tableofcontents` variable (which is defined in the `book:a4` scope from which default values were copied). The value of that variable causes a table of contents to be added to the anthology.

7.3 Running Canthology

To get Canthology to process the contents of its configuration file, you run the command:

```
canthology -f Canthology.cfg
```

The `"-f"` command-line option is used to specify the name of the configuration file. If you do not specify this option, then Canthology defaults to using the file `Canthology.cfg`. So, if your configuration file is called

`Canthology.cfg`, then you can tell Canthology to process it by running the (shorter) command:

```
canthology
```

When you run Canthology on the starting-point configuration file shown in Figure 7.1, the PDF file produced contains a title page and a table of contents (which is empty because you have not yet added any chapters or sections into the anthology). The title page of the document is shown in Figure 7.2 on the next page.

7.4 Text Substitutions on the Title Page

As you can see, the title page in Figure 7.2 contains five pieces of placeholder text. Obviously, you would like to replace those with text specific to the anthology you are producing. You can do this by editing the `substitutions.search_replace_pairs` variable in the configuration file, as shown in Figure 7.3 on page 50.

Syntactically, the `substitutions.search_replace_pairs` variable is a list of strings. However, the strings are arranged to form a two-column table. Each row of the table specifies a mapping from placeholder text to the desired text. There are two points worth noting about this table:

- If you do not want some placeholder text, then you can specify `""` as its replacement text. This is done for `"(SUBTITLE-PLACEHOLDER)"`.
- The syntax of the configuration file permits you to break up a long string into two or more shorter strings and use the `+` operator to concatenate them.

Canthology performs substitutions in files whose names match the pattern form `"*.tex"`; in such patterns, `*` denotes zero or more characters. (Later, in Section 8.3.3 on page 75, I will explain how to change this default behaviour.)

Having updated the `substitutions.search_replace_pairs` variable, you can rerun `canthology`. The updated version of the generated title page is shown in Figure 7.4 on page 51.

Figure 7.2: Formatting of `titlepage-template-1.tex` with placeholder text

(TITLE-PLACEHOLDER)

(SUBTITLE-PLACEHOLDER)

(AUTHOR-PLACEHOLDER)

(DESCRIPTION-PLACEHOLDER)

(PUBLISHER-PLACEHOLDER)

Figure 7.3: Configuring substitutions

```
substitutions.search_replace_pairs = [
    # search string               replace string
    #-----------------------------------------
     "(TITLE-PLACEHOLDER)",       "18th Century Poetry",
     "(SUBTITLE-PLACEHOLDER)",    "",
     "(AUTHOR-PLACEHOLDER)",      "John Smith",
     "(DESCRIPTION-PLACEHOLDER)", "Poems for the \emph{Poetry 201} "
                                  + "course, 2011--2012",
     "(PUBLISHER-PLACEHOLDER)",   "Department of English, "
                                  + "ACME University",
] + substitutions.search_replace_pairs;
```

If you dislike the layout provided by `titlepage-template-1.tex`, then you can modify the configuration file to use `titlepage-template-2.tex` or `titlepage-template-3.tex` instead. The results of such modifications are shown in Figures 7.5 and 7.6.

If none of those title page layouts suits your needs, then you have a few more choices:

- Section 9.6 on page 87 will explain how you can add colour or an image (such as a digital photograph) to the background of the title page. Perhaps doing that will be sufficient to give you a pleasing result.

- If you learn some LaTeX markup commands, then you can design your own title page instead of using one of the provided templates. (Later, in Section 9.2 on page 83, I will recommend some useful books for learning more about LaTeX.)

- You might prefer to design a title page using a drawing editor. If so, you can save the drawing as, say, a JPG, PNG or PDF file, and then import that into LaTeX to use as your title page.

For the moment, let's assume that the layout of the title page is good enough for your needs, so we can move on to the main content of the book.

Figure 7.4: Formatting of `titlepage-template-1.tex` with substitutions

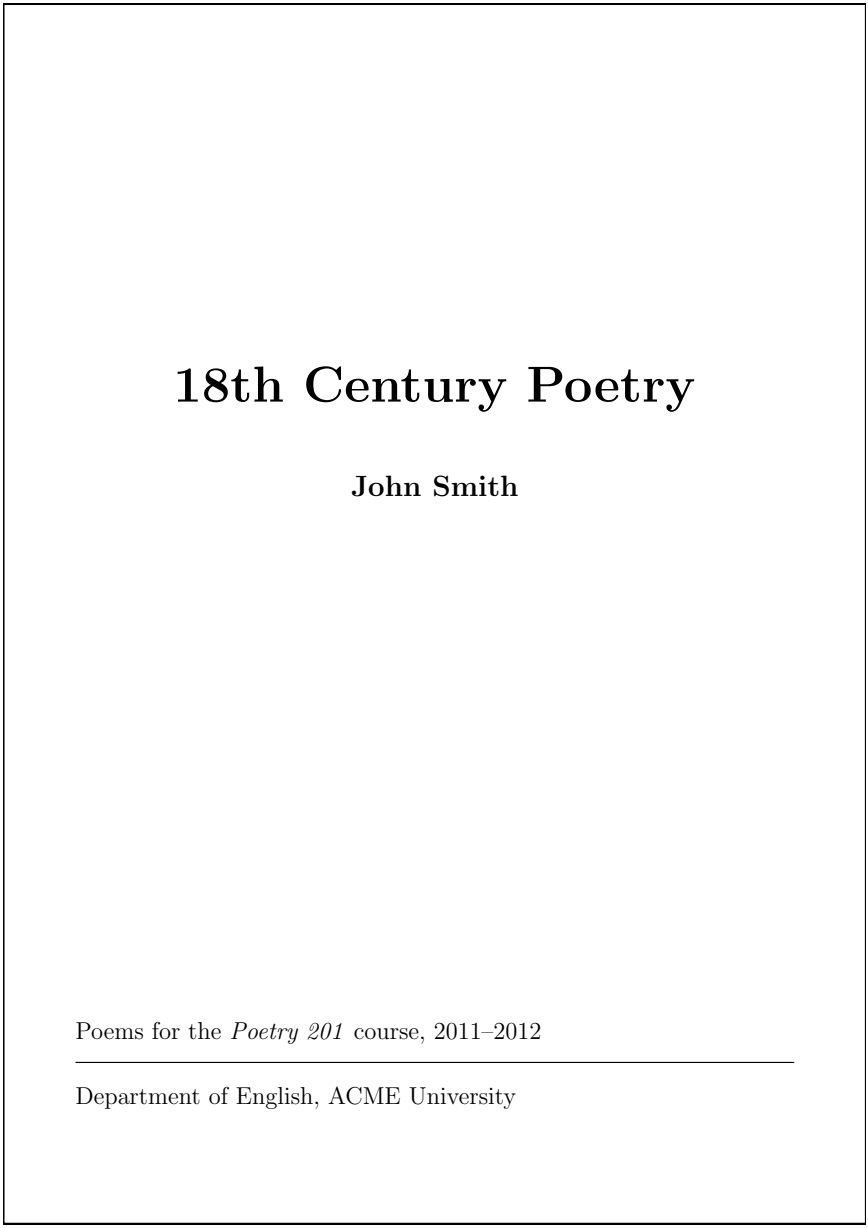


Figure 7.5: Formatting of `titlepage-template-2.tex` with substitutions

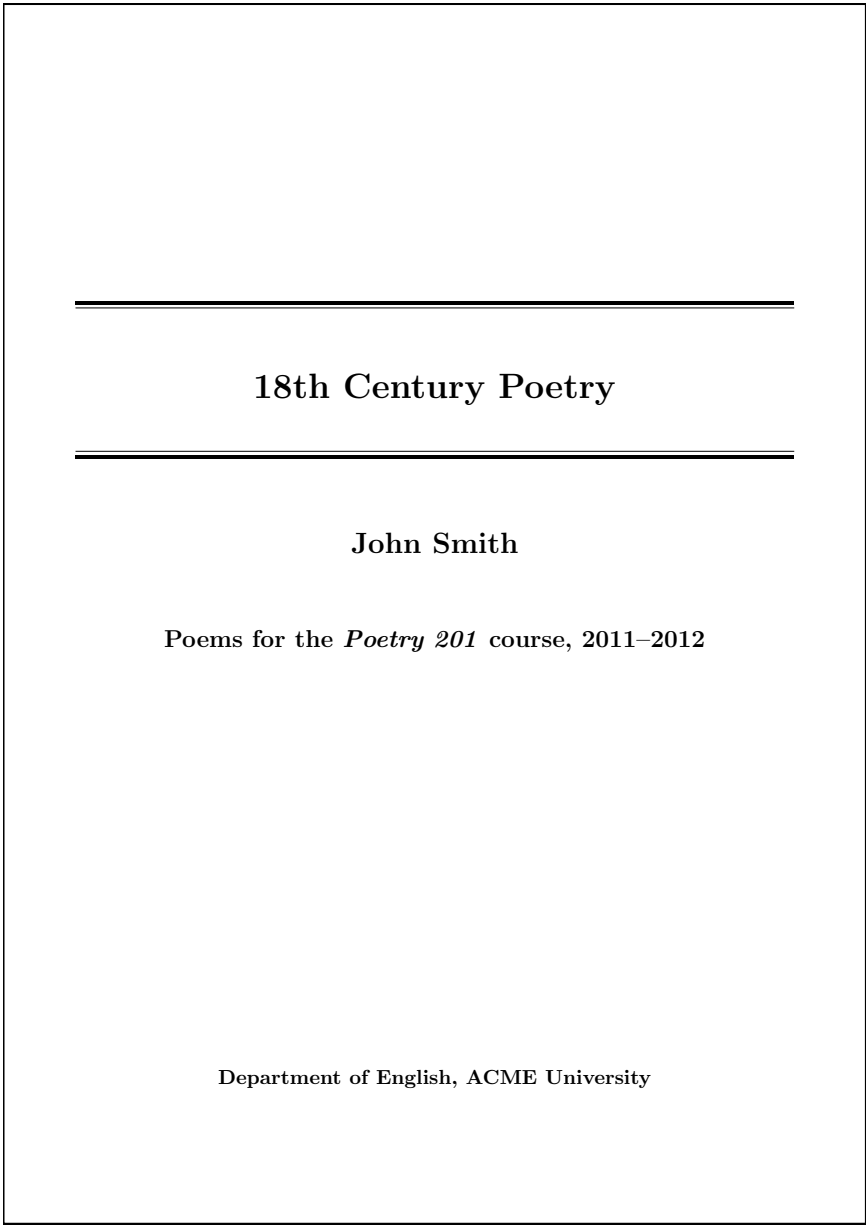
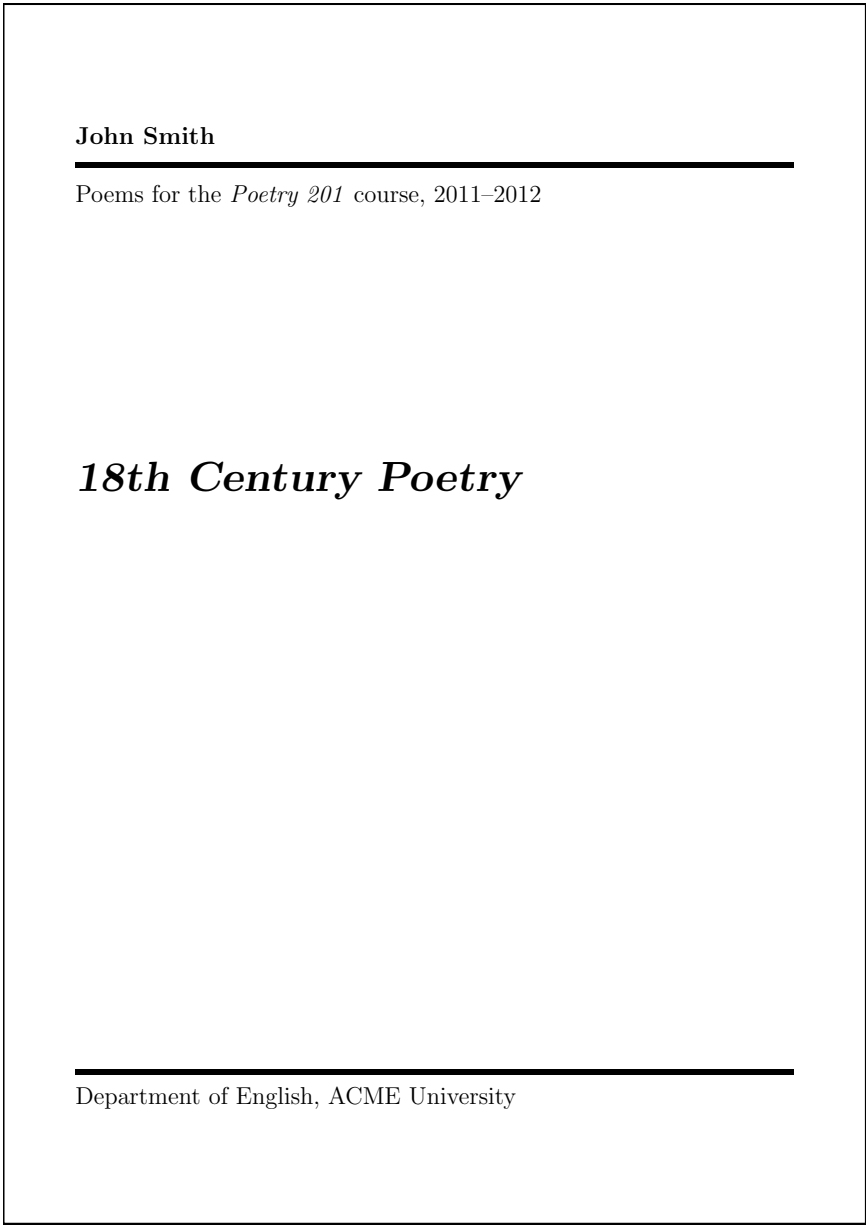


Figure 7.6: Formatting of `titlepage-template-3.tex` with substitutions



7.5 Adding Content

Let's assume you have contributions in files whose names reflect the title of the contribution. For example, the file `what-i-did-last-summer.tex` might look like the following:

```
\chapter{What I Did Last Summer}
\chapterAuthorInfo{John Smith}{England}

... Text of the story omitted for brevity.
```

You can add those contributions to your anthology in a few simple steps.

1. Copy the files into the same directory as `Canthology.cfg` so that Canthology can find them. (Later, in Section 8.3.2 on page 72, I will explain how you can configure Canthology to find files located in other directories.)

2. Modify the `main_matter` variable in `Canthology.cfg` (Figure 7.1 on page 44) to have an `\input` command for each contribution.

3. If your anthology has a large number of chapters, then you might want to group several chapters together to form a larger unit called a *part*. You can do this by adding a command of the form `\part{`*name*`}` to the `main_matter` variable, where *name* is the name of the part.

Figure 7.7 on the next page shows an example configuration file that has been modified according to steps 2 and 3 in the above list; for your convenience, the `\input` and `\part` commands are shown in a **bold** font.

The indentation of `\input` commands after each `\part` command is *not* required. It simply provides a way to indicate the high-level structure of the anthology.

If you run Canthology on this configuration file, it will create a PDF file that contains a title page, table of contents, and eight chapters, most of which are grouped into parts. The generated table of contents is shown in Figure 7.8 on page 56. Notice that use of `\chapterAuthorInfo` commands results in the name of each chapter's author being listed in the table of contents.

Figure 7.7: Configuration file with some content

```
@include getenv("CANTHOLOGY_HOME") + "/etc/canthology-defaults.cfg";
anthology1 {
    @copyFrom "book:a4";
    root_file {
        base_name = "wasting-time" + macro.paperSizeSuffix;
        front_matter = [
            "\input{titlepage-template-1.tex}",
            macro.tableofcontents,
        ];
        main_matter = [
            "\input{introduction.tex}",
            "\part{Wasting Time in Education}",
                "\input{kindergarten.tex}",
                "\input{school.tex}",
                "\input{college.tex}",
            "\part{The Passing of the Seasons}",
                "\input{what-i-did-last-spring.tex}",
                "\input{what-i-did-last-summer.tex}",
                "\input{what-i-did-last-autumn.tex}",
                "\input{what-i-did-last-winter.tex}",
        ];
        back_matter = [
        ];
    }
    substitutions.search_replace_pairs = [
        # search string               replace string
        #-------------------------------------------
         "(TITLE-PLACEHOLDER)",       "Wasting Time",
         "(SUBTITLE-PLACEHOLDER)",    "",
         "(AUTHOR-PLACEHOLDER)",      "Jane Doe",
         "(DESCRIPTION-PLACEHOLDER)", "",
         "(PUBLISHER-PLACEHOLDER)",   "",
    ];
}
```

Each `\part` command results in the name of the part being listed in the table of contents. Each part is automatically numbered using Roman numerals (I, II, III, IV, and so on). In addition, the main body of the book will contain a page displaying a part's number and name. An example of this is shown in Figure 7.9 on page 57.

Figure 7.8: Table of contents

Contents

Figure 7.9: A “part” page

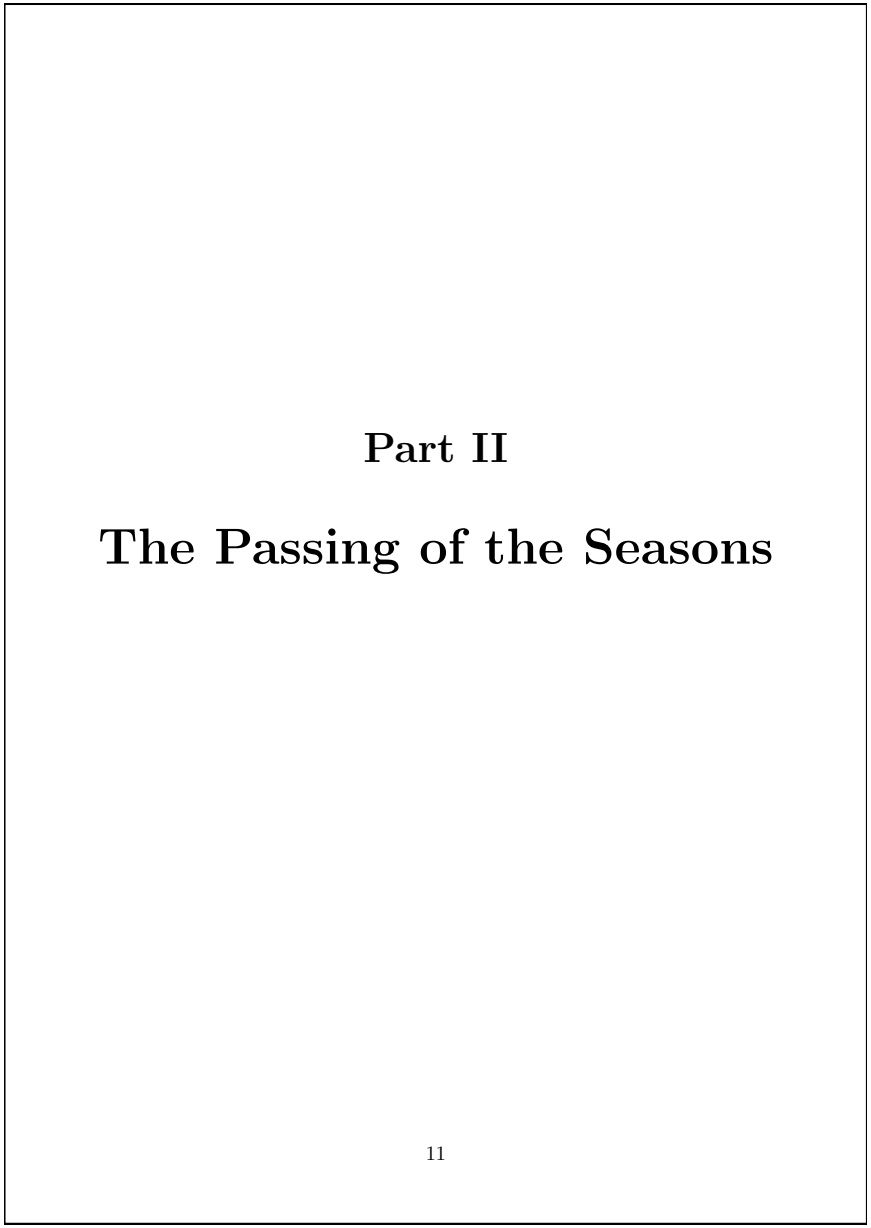


Figure 7.10 on the next page shows the starting page of a chapter. Notice that the use of a `\chapterAuthorInfo` command results in the name and personal details (such as age, location, or occupation) of the author appearing at the top of the page.

7.6 Front and Back Matter

You might want to put some additional material at the very start or very end of your anthology. For example, some books have a preface or forward at the start of the book, and some books have appendices or a glossary at the back.

When writing a preface, foreword, appendix or glossary, you should write it in the same way that you write a chapter, that is, you start it with a `\chapter` command. For example:

```
\chapter{Preface}
... Text of the preface omitted for brevity.
```

Figure 7.11 on page 60 illustrates how to add a preface, some appendices and a glossary to your Canthology configuration file.

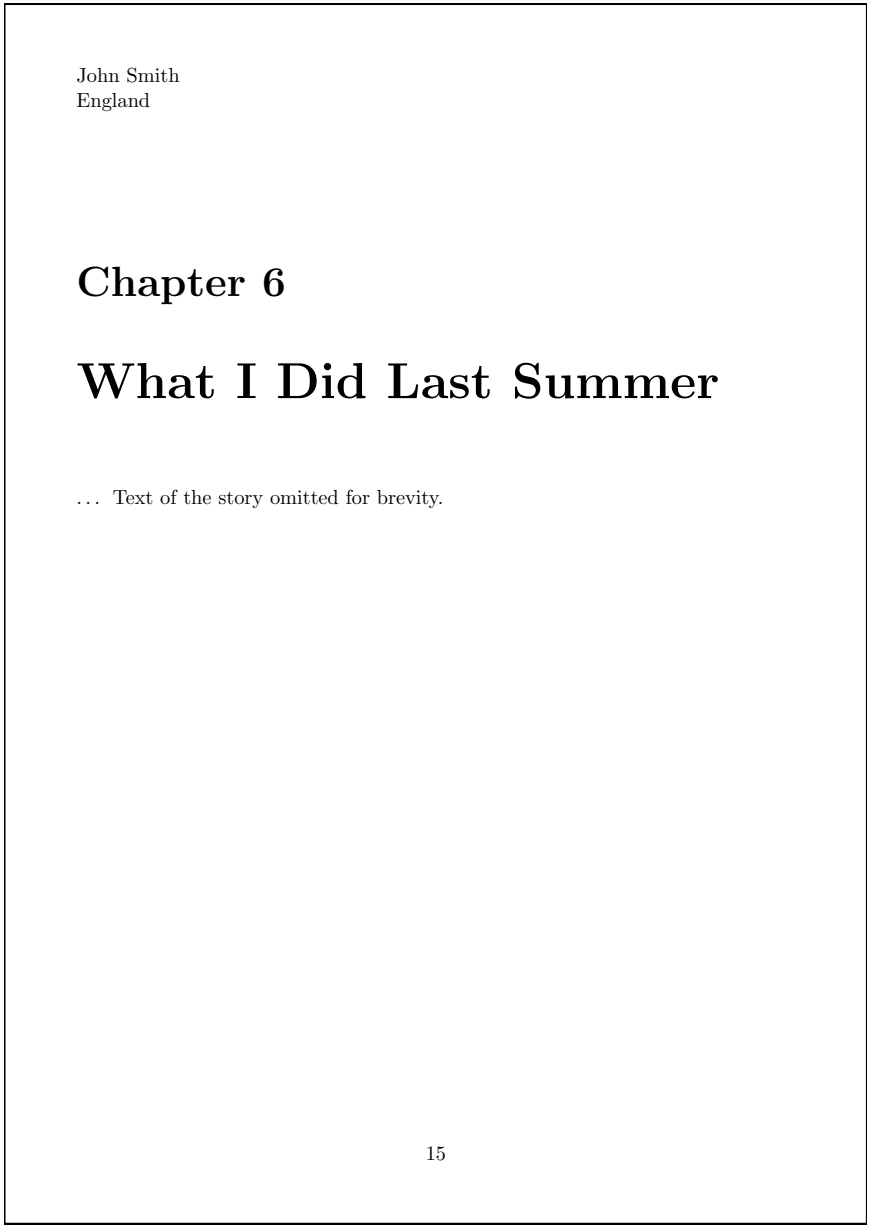
As you can see by focusing on the **bold** text, you add a preface or foreword to the `front_matter` variable in the configuration file, and add the glossary to the `back_matter` variable. Appendices are handled slightly differently: they go at the end of the `main_matter` and are preceded by `macro.startAppendices`.

If you now look at the generated table of contents in Figure 7.12 on page 61, you will see that the preface and glossary are listed as *unnumbered* chapters. In general, chapters and sections in the `front_matter` and `back_matter` are unnumbered, while chapters and sections in the `main_matter` are numbered.

You can verify the unnumbered nature of a front-matter chapter by looking at the preface shown in Figure 7.13 on page 62.

Appendices are placed in the `main_matter` so they can be numbered, but they are "numbered" with letters (A, B, C, and so on) rather than digits to distinguish them from chapters. The use of `macro.startAppendices`

Figure 7.10: Start of a chapter


John Smith
England

Chapter 6

What I Did Last Summer

. . . Text of the story omitted for brevity.

15

Figure 7.11: Adding front and back matter to the configuration file

```
@include getenv("CANTHOLOGY_HOME") + "/etc/canthology-defaults.cfg";
anthology1 {
    @copyFrom "book:a4";
    root_file {
        base_name = "wasting-time" + macro.paperSizeSuffix;
        front_matter = [
            "\input{titlepage-template-1.tex}",
            macro.tableofcontents,
            "\input{preface.tex}",
        ];
        main_matter = [
            "\input{introduction.tex}",
            "\part{Wasting Time in Education}",
                "\input{kindergarten.tex}",
                "\input{school.tex}",
                "\input{college.tex}",
            "\part{The Passing of the Seasons}",
                "\input{what-i-did-last-spring.tex}",
                "\input{what-i-did-last-summer.tex}",
                "\input{what-i-did-last-autumn.tex}",
                "\input{what-i-did-last-winter.tex}",
            macro.startAppendices,
                "\input{statistical-tables.tex}",
                "\input{relevant-laws.tex}",
        ];
        back_matter = [
            "\input{glossary.tex}",
        ];
    }
    substitutions.search_replace_pairs = [
        # search string               replace string
        #--------------------------------------------
         "(TITLE-PLACEHOLDER)",       "Wasting Time",
         "(SUBTITLE-PLACEHOLDER)",    "",
         "(AUTHOR-PLACEHOLDER)",      "Jane Doe",
         "(DESCRIPTION-PLACEHOLDER)", "",
         "(PUBLISHER-PLACEHOLDER)",   "",
    ];
}
```

Figure 7.12: Table of contents with front and back matter

Contents

Figure 7.13: Front and back matter is unnumbered

Preface

... Text of the preface omitted for brevity.

triggers this switch in the numbering convention and also causes a page to be generated that signals the start of the appendices (Figure 7.14 on the next page).

Figure 7.15 on page 65 shows the formatting of an appendix.

7.7 Summary

This chapter has provided a rapid and somewhat superficial tour of Canthology's features. Canthology and LaTeX take care of tedious tasks in typesetting a book, such as the layout of a title page, creating a table of contents, correct numbering of chapters and appendices, and ensuring chapters are *unnumbered* in the front and back matter. This frees you to focus on the contents and structure of the book.

The next chapter takes a deeper look at Canthology, so you can learn how to fine-tune its behaviour.

Figure 7.14: The appendices page

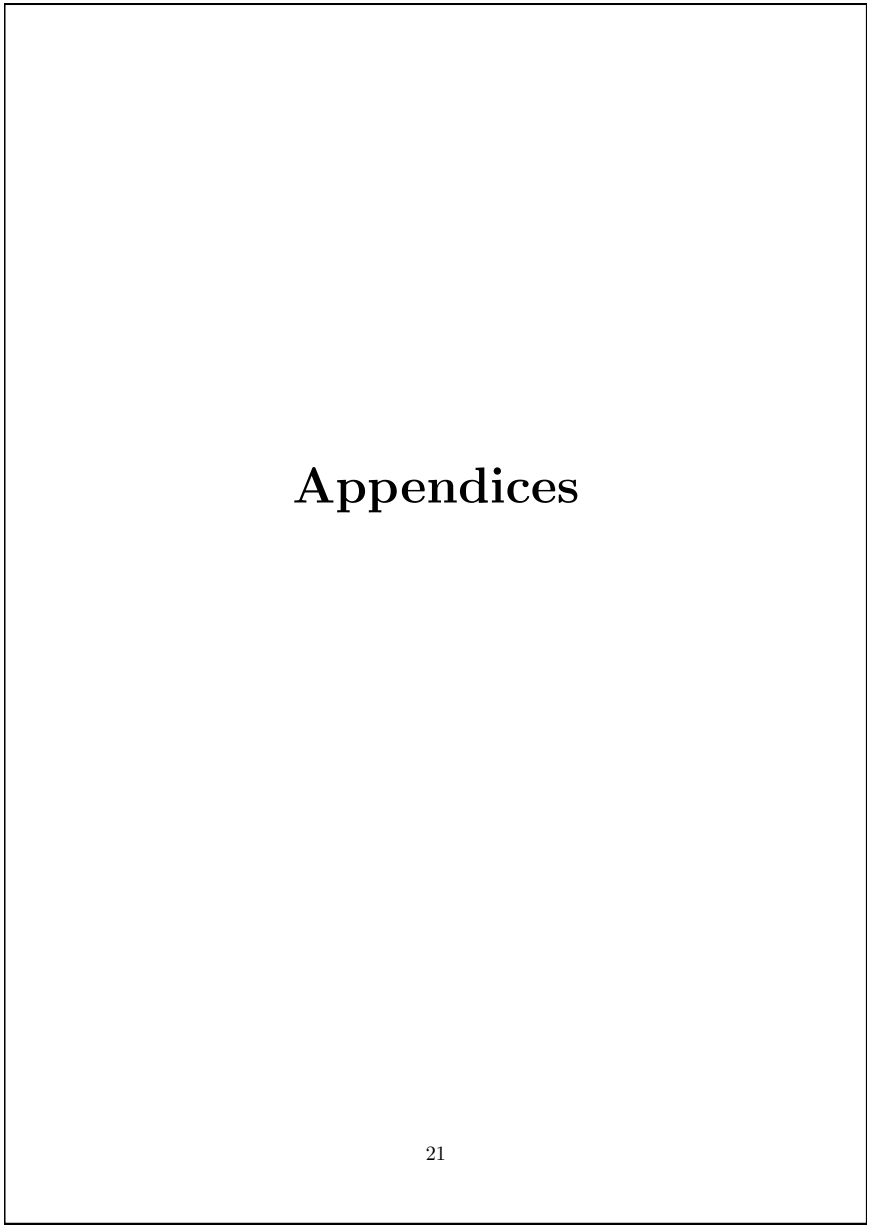


Figure 7.15: Start of an appendix

Appendix A

Statistical Tables

... Text omitted for brevity.

Chapter 8

How Canthology Operates

8.1 Introduction

Chapter 7 used a demonstration-driven approach to illustrate (a subset of) *what* Canthology can do. This chapter explains *how* Canthology does those things.

8.2 Configuration File and Scopes

Consider the following command:

```
canthology -f my-file.cfg -scope doc-1 -scope doc-2
```

The `-f` option instructs Canthology to use `my-file.cfg` as its configuration file. If you omit this option, then Canthology defaults to using `Canthology.cfg` as its configuration file.

The `-scope` option instructs Canthology to generate a document from the configuration variables in the specified scope. You can specify the `-scope` option multiple times; this causes Canthology to generate documents for each of the specified scopes. If you do not specify any scopes, then Canthology generates documents from *all* the top-level scopes that contain a variable called `root_file.base_name`.

8.3 Configuration Variables

Figure 8.1 shows the starting-point configuration file you can obtain by running the command:

```
canthology -create Canthology.cfg
```

Figure 8.1: Starting-point configuration file

```
@include getenv("CANTHOLOGY_HOME") + "/etc/defaults.cfg";
anthology1 {
    @copyFrom "book:a4";
    root_file {
        base_name = "my-anthology" + macro.paperSizeSuffix;
        preamble = [
        ] + preamble;
        front_matter = [
            "\input{titlepage-template-1.tex}",
            macro.tableofcontents,
        ];
        main_matter = [
        ];
        back_matter = [
        ];
    }
    substitutions.search_replace_pairs = [
        # search string            replace string
        #-------------------------------------------
    ] + substitutions.search_replace_pairs;
}
```

The starting-point configuration file is short because the `@copyFrom` statement copies a lot of configuration variables from a scope defined in the `@include`-d file. Figure 8.2 shows a configuration file that explicitly sets *all* configuration variables used by Canthology, although not all the values shown for those variables are the actual default values. Instead, the variables' values have been chosen to facilitate the discussion that follows.

At a high level, Canthology operates as follows:

1. Canthology creates a "working directory" in which it will create some LaTeX files. In particular, Canthology uses information in its

Figure 8.2: Configuration variables

```
anthology1 {
    working_dir = "output-pdf-a4";
    root_file {
        base_name = "my-anthology-a4";
        macro {
            tableofcontents = "\clearforchapter %n"
                            + "\tableofcontents*";
            appendix = "\clearforchapter %n"
                     + "\appendix %n"
                     + "\phantomsection %n"
                     + "\appendixpage";
        }
        documentclass {
            name = "book";
            options = ["12pt", "twoside"];
        }
        package {
            names = ["appendix", "canthology", "geometry"];
            geometry.options = ["paper=a4paper"];
        }
        preamble = [
            "\setcounter{tocdepth}{4}",
        ];
        front_matter = [
            "\input{titlepage.tex}",
            macro.tableofcontents,
        ];
        main_matter = [
            "\input{chapter-1.tex}",
            "\input{chapter-2.tex}",
            macro.appendix,
            "\input{appendix-a.tex}",
            "\input{appendix-b.tex}",
        ];
        back_matter = [
            "\input{glossary.tex}"
        ];
    }

... continued on the next page
```

Figure 8.2 (continued): Configuration variables

```
... continued from the previous page
    copy {
        commands = ["\input", "\usepackage", "\includegraphics"];
        file_extensions = [".tex", ".sty", ".png", ".jpg"];
        search_path = [
            ".",
            getenv("CANTHOLOGY_HOME") + "/etc/latex"
        ];
        extra_files_to_copy = [];
        look_for_copy_commands {
            in_matching_files = ["*.tex", "*.sty"];
            not_in_matching_files = [];
        }
    }
    substitutions {
        in_matching_files = ["*.tex"];
        not_in_matching_files = [];
        search_replace_pairs = [
            "(AUTHOR-PLACEHOLDER)", "Jane Doe",
            "(TITLE-PLACEHOLDER)",  "Modern Fairy Tales",
        ];
    }
    build_commands = [
      "pdflatex -interaction=errorstopmode (ROOT_FILE_BASE_NAME).tex",
      "pdflatex -interaction=errorstopmode (ROOT_FILE_BASE_NAME).tex"
    ];
}
```

configuration file to build a *root* `".tex"` file in the working directory. This root file will contain `\input` commands to add the contributions of the anthology.

2. Canthology copies required support files into the working directory. These support files include all the contributions, graphic files (if any) and some package (`".sty"`) files.

3. Canthology runs some LaTeX-related commands (such as `latex` or `pdflatex`) on the root `".tex"` file to produce the ready-to-print anthology.

Those steps are discussed in the following subsections.

8.3.1 Creating the Root ".tex" File

The working_dir configuration variable specifies the name of the working directory that Canthology should create. By convention, the value of this variable starts with "output-" and the remainder of the value indicates how the document is formatted. For example, the value might be "output-pdf-a4" if you are creating an A4-formatted PDF version of the anthology. This convention makes it easy to create multiple versions of the documents, each one formatted for a different paper size.

The root_file scope contains variables that are used to create a root ".tex" file inside the working directory. The remaining discussion in this subsection is for variables within the root_file scope.

The base_name configuration variable specifies the "base" file name (that is, the file name *without* any extension) for the root ".tex" file. For example, if base_name has the value "my-anthology-a4", then the root file name is my-anthology-a4.tex.

The configuration variables within the documentclass sub-scope are used to create the \documentclass command at the start of the root ".tex" file. For example, consider the following documentclass scope:

```
documentclass {
    name = "book";
    options = ["12pt", "twoside"];
}
```

Those settings result in the following being generated:

```
\documentclass[12pt, twoside]{book}
```

The configuration variables within the package sub-scope are used to generate \usepackage commands. For example, consider the following package scope:

```
package {
    names = ["appendix", "canthology", "geometry"];
    geometry.options = ["paper=a4paper"];
}
```

The names variable specifies that three packages are being used: appendix [15], canthology and geometry [12]. For each of these specified packages, you can *optionally* specify a list of package options. The above example shows a list of options being specified for the geometry package, but no options for the appendix or canthology packages. Those configuration settings result in the following being generated:

```
\usepackage{appendix}
\usepackage{canthology}
\usepackage[paper=a4paper]{geometry}
```

The value of the preamble variable is written to the root ".tex" file immediately after the \usepackage commands.

Canthology ignores everything in the macro sub-scope. The intention is that you can use this sub-scope to define variables whose values are a sequence of LaTeX commands. Then, you can use those variables (as a form of shorthand) in preamble or (more commonly) the front_matter, main_matter or back_matter variables. Figure 8.2 uses a **bold** font to illustrate this.

Having written the value of the preamble variable to the ".tex" file, Canthology then writes "\begin{document}" and "\frontmatter" to the file and follows it with all the strings contained in the front_matter configuration variable. For example, the configuration shown in Figure 8.2 would result in the following being generated:

```
\begin{document}
\frontmatter
\input{titlepage.tex}
\clearforchapter
\tableofcontents
```

Then "\mainmatter" and all the strings in the main_matter configuration variable are written to the ".tex" file. After that, "\backmatter" and all the strings in the back_matter configuration variable are written to the ".tex" file.

Canthology finishes the ".tex" file by writing "\end{document}" to it. Figure 8.3 shows the complete ".tex" file that is generated.

Figure 8.3: The generated `my-anthology-a4.tex` file

```
\documentclass[12pt,twoside]{memoir}

\usepackage{canthology}
\usepackage[paper=a4paper]{geometry}
\setcounter{tocdepth}{4}

\begin{document}

\frontmatter
\input{titlepage.tex}
\clearforchapter
\tableofcontents

\mainmatter
\input{chapter-1.tex}
\input{chapter-2.tex}
\clearforchapter
\appendix
\phantomsection
\appendixpage
\input{appendix-a.tex}
\input{appendix-b.tex}

\backmatter
\input{glossary.tex}
\end{document}
```

8.3.2 Copying Support Files

It is not enough for Canthology to create just the root `".tex"` file in the working directory. Canthology must copy some other files into the working directory too. Here are some examples of other files that may need to be copied:

- Any `".tex"` files that are `\input`-ed by the root file. (And if those `\input`-ed files themselves contain `\input` statements, then Canthology must recursively follow the chain of `\input` commands.)
- Any graphic files, such as diagrams or digital photographs, that are used in an `\includegraphics` command inside a `".tex"` file.

- BIBTEX-based bibliographies, if any, that are used by the document.

LATEX is flexible enough that it is impossible for Canthology to accurately predict the entire set of files that must be copied into the working directory. For this reason, Canthology does not have a hard-coded set of rules for deciding which files need to be copied. Instead, the copying of files is driven by configuration variables in the `copy` scope, as shown in Figure 8.2.

The `copy.commands` variable specifies a list of LATEX commands whose *first* parameter between braces (that is, between `{` and `}`) specifies a file to be copied. For example, consider the following setting of this variable:

```
commands= ["\input", "\usepackage", "\includegraphics"];
```

If Canthology finds the statement `\input{chapter-1.tex}` in a file, then it will copy the `chapter-1.tex` file to the working directory (and recursively search the copied file for other files to be copied). Now let's assume Canthology finds the following statement:

```
\includegraphics[scale=0.7]{my-photograph}
```

Canthology ignores *optional* parameters (enclosed between `[` and `]`) and instead looks at the first parameter enclosed in braces: `my-photograph`. This parameter to the `\includegraphics` command is a file name that may or may not contain a file-name extension (such as `".jpg"` or `".png"`). The `copy.file_extensions` configuration variable enables Canthology to cater for both possibilities:

```
file_extensions = [".tex", ".pdf", ".png", ".jpg"];
```

When Canthology encounters the name of a file (such as `chapter-1.tex` or `my-photograph`) to be copied, Canthology first tries to copy the file whose name is specified. If that fails, then Canthology suffixes the file name with each of the extensions specified in `copy_file_extensions` and tries to copy the resulting file.

The approach discussed above works *if* the file to be copied is specified as the first non-optional parameter to a command. But what if the file is specified in, say, the second or third parameter to a command? As a hypothetical example:

```
\exampleCommand{11}{42}{another-photograph}
```

Or perhaps the file to be copied is a non-LaTeX file required to control the build process. An example of this might be a `Makefile` or an Ant `build.xml` file. You should list such files in the `copy.extra_files_to_copy` configuration variable. For example:

```
extra_files_to_copy= ["another-photograph", "Makefile"];
```

However, it is unlikely you will need to do that frequently. This is because the `copy.commands` variable will be sufficient most of the time.

When Canthology is looking for a file that it must copy, it looks for that file in the list of directories specified in the `search_path` configuration variable:

```
search_path = [
    ".",
    getenv("CANTHOLOGY_HOME") + "/etc/latex"
];
```

The first entry in the list instructs Canthology to look in the current directory. The second directory instructs Canthology to look in the `etc/latex` subdirectory of the Canthology installation. Among other things, that directory contains the template title pages that I discussed in Section 7.4 on page 48.

If Canthology is unable to copy a file because the file is not in any of the directories listed in `copy.search_path`, then Canthology does *not* consider this to be an error. Instead, Canthology assumes that the "missing" file is bundled with a LaTeX distribution, so LaTeX will be able to find it. This is commonly the case with package (`".sty"`) files.

If you are editing an anthology that contains, say, 50 contributions, then you might find it awkward to store all the contributions in a single directory. You might prefer to spread the contributions over several directories, so that each directory contains a subset of the `".tex"` files. (You might have one directory for poems, another for short stories, and so on.) You can do this by listing each of those directories in the `search_path` configuration variable. For example:

```
search_path = [
    "poems", "short-stories, "plays",
    getenv("CANTHOLOGY_HOME") + "/etc/latex"
];
```

When Canthology is copying a file, it needs to decide whether it should search inside the file for nested copy commands. Canthology uses the configuration variables in the `look_for_copy_commands` scope to make this decision:

```
look_for_copy_commands {
    in_matching_files = ["*.tex", "*.sty"];
    not_in_matching_files = [];
}
```

Canthology will search inside a copied file for nested copy commands if: (1) the file name matches at least one pattern in `in_matching_files`, and (2) does *not* match any patterns in `not_in_matching_files`. In a pattern, `*` acts as a wildcard that can match zero or more characters. For example, `"*.tex"` matches file names that end in `".tex"`. These configuration variables provide a simple, yet effective, way to instruct Canthology to look for nested copy commands in `".tex"` and `".sty"` files but not in, say, `".jpg"` or `".png"` files.

8.3.3 Performing Text Substitutions

Variables in the nested `substitutions` scope control how Canthology performs search-and-replace on files. For example:

```
substitutions {
    in_matching_files = ["*.tex"];
    not_in_matching_files = [];
    search_replace_pairs = [
        "(AUTHOR-PLACEHOLDER)", "Jane Doe",
        "(TITLE-PLACEHOLDER)",  "Modern Fairy Tales",
    ];
}
```

Canthology performs substitutions on a copied file if: (1) the file name matches at least one pattern in `in_matching_files`, and (2) does *not* match any patterns in `not_in_matching_files`.

The `search_replace_pairs` variable is used to specify pairs of *search* and *replace* strings. Canthology automatically extends `search_replace_pairs` so that occurrences of `"(ROOT_FILE_BASE_NAME)"` are replaced with the value of the `root_file.base_name` configuration variable.

8.3.4 Running LaTeX-related commands

After Canthology has generated the root `".tex"` file and copied supporting files to the working directory, the only task remaining for Canthology is to run one or more LaTeX-related commands to turn the files into a nicely formatted document in, for example, PDF format. To do this, Canthology executes each command specified by the `build_commands` configuration variable. For example:

```
build_commands = [
  "pdflatex -interaction=errorstopmode "
       + "(ROOT_FILE_BASE_NAME).tex",
  "pdflatex -interaction=errorstopmode "
       + "(ROOT_FILE_BASE_NAME).tex"
];
```

Because of the way LaTeX works, some commands (such as `latex` or `pdflatex`) have to be run twice to correctly resolve cross references and to produce a table of contents. This is why the above example runs `pdflatex` twice.

Canthology uses `substitutions.search_replace_pairs` to perform substitutions on each command that is about to be executed. By doing this, the `"(ROOT_FILE_BASE_NAME)"` text within a command will be replaced with the value of the `root_file.base_name` configuration variable.

If `latex` or `pdflatex` encounters an error in an input file, then, by default, it goes into an interactive mode to ask the user what it should do. The `"-interaction=errorstopmode"` option instructs `latex` and `pdflatex` to

not go into an interactive mode if an error occurs, and instead just print an error message and exit.

8.4 The `etc/defaults.cfg` File

The first line in a starting-point configuration file is:

```
@include getenv("CANTHOLOGY_HOME")+ "/etc/defaults.cfg";
```

An outline of the included `etc/defaults.cfg` file is shown in Figure 8.4. The file contains many scopes; these enable it to provide default values suitable for many combinations of document classes (`book`, `report`, `article` and `memoir`) and paper sizes.

The `etc/defaults.cfg` file serves several important purposes.

First, Canthology makes use of over twenty configuration variables. The `defaults.cfg` file provides sensible default values for most of those variables. This greatly simplifies Canthology for new users.

Second, some people like to write LaTeX-based documents with, say, the `book` class, while other people prefer to use the newer and more flexible `memoir` class. Although the `memoir` class is *mostly* backwards compatible with the `book` class, there are a *few* incompatibilities, which often manifest themselves as slightly differing contents of the root `".tex"` file. For example:

- A `book`-based document might need more `\usepackage` commands than a `memoir`-based document, because the `memoir` class has the functionality of many popular packages built into it.
- In a `memoir`-based document, the sequence of commands used to achieve a particular result—such as guarantee the table of contents appears on a right-hand page, or ensure that the table of contents lists the start of the appendices—is sometimes different to the required sequence of commands in a `book`-based document.

The `etc/defaults.cfg` file can encapsulate users from these subtle differences between `book`- and `memoir`-based documents. This is because

Figure 8.4: Outline of the `etc/defaults.cfg` file

```
default.common { ... } # details omitted for brevity
default.html {
    @copyFrom "default.common";
    ... # details omitted for brevity
}

default.a4-geometry-options { root_file.package.geometry.options = [...]; }
default.a5-geometry-options { ... }
default.a5-trimmed-geometry-options { ... }
default.letter-geometry-options { ... }

book:a4 {
    @copyFrom "default.common";
    @copyFrom "default.a4-geometry-options";
    working_dir = "output-pdf-a4";
    root_file {
        documentclass.name = "book";
        macro { ... } # details omitted for brevity
        package.names = [...];
    }
}
book:a5 {
    @copyFrom "book:a4";
    @copyFrom "default.a5-geometry-options";
    working_dir = "output-pdf-a5";
    macro.paperSizeSuffix = "-a5";
    root_file.documentclass.options = ["10pt", "twoside"];
}
book:a5-trimmed {
    @copyFrom "book:a4";
    @copyFrom "default.a5-trimmed-geometry-options";
    working_dir = "output-pdf-a5-trimmed";
    macro.paperSizeSuffix = "-a5-trimmed";
    root_file.documentclass.options = ["10pt", "twoside"];
}
book:letter {
    @copyFrom "book:a4";
    @copyFrom "default.letter-geometry-options";
}
... continued on the next page
```

Figure 8.4 (continued): Outline of the `etc/defaults.cfg` file

```
... continued from the previous page
book:html-one-page {
    @copyFrom "default.html";
    working_dir = "output-html";
    ... # details omitted for brevity
}
book:html-many-pages {
    @copyFrom "default.html";
    ... # details omitted for brevity
}

report:a4 { ... }
report:a5 { ... }
report:a5-trimmed { ... }
report:letter { ... }
report:html-one-page { ... }
report:html-many-pages { ... }

article:a4 { ... }
article:a5 { ... }
article:a5-trimmed { ... }
article:letter { ... }
article:html-one-page { ... }
article:html-many-pages { ... }

memoir:a4 { ... }
memoir:a5 { ... }
memoir:a5-trimmed { ... }
memoir:letter { ... }
memoir:html-one-page { ... }
memoir:html-many-pages { ... }

memoir-article:a4 { ... }
memoir-article:a5 { ... }
memoir-article:a5-trimmed { ... }
memoir-article:letter { ... }
memoir-article:html-one-page { ... }
memoir-article:html-many-pages { ... }
```

the `memoir`-related scopes specify the packages required by `memoir` and also define `macro.tableofcontents` and `macro.startAppendices` in a `memoir`-compatible way. Likewise, the `book`-related scopes specify `book`-

required packages and define the `macro` variables in a `book`-compatible way.

Of course, *many* of the configuration settings for `memoir` are identical to those required for `book`. This is why the `etc/defaults.cfg` file has a `default.common` scope that defines variables with common values. The `memoir`- and `book`-based scopes use the `@copyFrom` command to access those variables.

Subtle differences are not limited to just use of the `book` or `memoir` classes. Subtle difference in how to write a LaTeX document also arise when creating PDF or HTML documents. The `etc/defaults.cfg` file does its best to encapsulate many of those output-format differences too.

The third and final purpose of the `etc/defaults.cfg` file is to make it possible for people to extend Canthology without having to modify its Java source code. For example:

- There are other document classes that some users might wish to use to write a document.
- The `etc/defaults.cfg` file assumes users want to use `pdflatex` to convert a LaTeX document into a PDF file. Some users might prefer to use, say, `xelatex` or `latex` and `dvipdfmx` to produce a PDF file. Or perhaps a user will want to use `latex` and `dvips` to produce a PostScript file.
- The `etc/defaults.cfg` uses HeVeA to generate HTML (this is discussed in Part IV). However, some people may prefer to use another LaTeX-to-HTML converter.

It should be possible to customise Canthology to support the above by modifying `etc/defaults.cfg`.

8.5 Extending Configuration Variables

As discussed in Section 8.4, one purpose of `etc/defaults.cfg` is to encapsulate subtle differences in the use of particular document classes (such

as `book` and `memoir`) or in creating documents for different output formats (such as PDF and HTML).

One way this encapsulation occurs is by assigning suitable values to the following variables:

```
root_file.package.names
root_file.preamble
substitutions.search_replace_pairs
```

Because of this, it is important to *extend* (rather than *replace*) the values of these variables in the configuration scope for a document. For example, the lines shown in **bold** in Figure 8.5 are likely to result in LaTeX error messages or badly-formatted output when you run `canthology`.

Figure 8.5: Some configuration variables should not be replaced

```
@include getenv("CANTHOLOGY_HOME") + "/etc/defaults.cfg";
anthology1 {
    @copyFrom "book:a4";
    root_file {
        base_name = "my-anthology" + macro.paperSizeSuffix;
        package {
            names = [...];                                    % bad
        }
        preamble = [...];                                     % bad
        front_matter = [...];
        main_matter = [...];
        back_matter = [...];
    }
    substitutions {
        search_replace_pairs = [...];                         % bad
    }
}
```

Instead, it is better to extend the values of those variables by using the list concatenation operator (+) to merge a new value with the existing value of a variable. This is illustrated in Figure 8.6 on the next page.

Figure 8.6: Some configuration variables should be extended

```
@include getenv("CANTHOLOGY_HOME") + "/etc/defaults.cfg";
anthology1 {
    @copyFrom "book:a4";
    root_file {
        base_name = "my-anthology" + macro.paperSizeSuffix;
        package {
            names = [...] + names;                              % good
        }
        preamble = [...] + preamble;                            % good
        front_matter = [...];
        main_matter = [...];
        back_matter = [...];
    }
    substitutions {
        search_replace_pairs = [...] + search_replace_pairs;  % good
    }
}
```

Chapter 9

Next Steps

9.1 Introduction

If you have read all the preceding chapters of this manual, then you should have a good understanding of how to use Canthology. However, you may be wondering, "So far I have learned how to do 95% of what I want to in the anthology I am creating, but how do I learn to do the remaining 5%?" The purpose of this chapter is to answer that question.

9.2 LaTeX Documentation

Recall that Canthology is a simplification wrapper for LaTeX. Because of this, many questions of the form, "How can I do such-and-such with Canthology?" are really questions about how to do such-and-such with LaTeX. To find the answers to such questions, you will probably need to read more about LaTeX.

Many LaTeX books have been published. You can find a list of such books by visiting an on-line bookshop (such as Amazon.co.uk), searching for "latex typesetting", and then reading customer reviews to narrow down your search.

One book I recommend is *LaTeX: a Document Preparation System* [5], which provides a good introduction to LaTeX. Alternatively, some good

introductory guides to LaTeX are available as free downloads [10] [4].

If an introductory book does not answer your questions about how to do such-and-such, then perhaps the functionality you are seeking is provided by one of the numerous add-on packages available for LaTeX. You can search for these on the Comprehensive TeX Archive Network, which is usually referred to by its acronym: CTAN (www.ctan.org).

The LaTeX Companion [8] discusses over 200 packages that are available for LaTeX. You may find it more convenient to borrow or buy a copy of that book and scan through its contents to find a package that serves your needs rather than search on CTAN.

The Memoir Class for Configurable Typesetting User Guide [14] documents the `memoir` class, which is a (mostly) backwards-compatible replacement for the `book` class. The `memoir` class contains the functionality of several dozen popular packages built into it. So, if you use the `memoir` class, you are less likely to have to search for additional packages to satisfy your needs.

9.3 LaTeX Errors

Sooner or later, you will encounter an error in a `".tex"` file, and this will cause LaTeX to report an error message. It is outside the scope of this manual to discuss the LaTeX error-reporting mechanism. Instead, some introductory books on LaTeX discuss this topic in detail [5] [4].

9.4 Defining New Commands

LaTeX provides numerous built-in commands. It also provides facilities for you to define additional commands. Most books and tutorials on LaTeX discuss this topic in detail, so I will provide only a brief overview here.

Let's assume you are writing a phrase book for French in which you use a **bold** font for English words and phrases, and a ***bold italic*** font for their French counterparts. Thus, to obtain the following output:

> The French for **the table** is ***la table***.

you might write:

```
The French for \textbf{the table} is
\textbf{\textit{la table}}.
```

You may consider those LaTeX font-changing commands to be verbose, especially if you will be using them many times in your book. For this reason, you might want to define two commands, `\en` (for typesetting English text) and `\fr` (for typesetting French text), as follows:

```
\newcommand{\en}[1]{\textbf{#1}}
\newcommand{\fr}[1]{\textbf{\textit{#1}}}
```

You use the `\newcommand` command to define a new command. The first argument to `\newcommand` is the name of the new command that is to be defined. If the new command is to take arguments, then the number of arguments is specified in square brackets; thus, `[1]` indicates that the new command will take one argument. The final argument to `\newcommand` is the body of the new command. Within this body, `#1` is a placeholder for the first argument, `#2` is a placeholder for the second argument, and so on. Having defined the above commands, you can use them as follows:

```
The French for \en{the table} is \fr{la table}.
```

Command definitions are usually placed in the preamble of a document, as you can see in Figure 9.1 on the next page.

When using Canthology, you place your command definitions in the preamble of your document by adding them to the `root_file.preamble` variable.

9.5 Implementing a Package

If you need to define a large number of commands, then you may find it inconvenient to define them all in the preamble of your document. An alternative is to define them in as a *package*, which is simply a file (with a `".sty"` extension) that contains the definitions of commands. Actually, the contents of a `".sty"` file are supposed to follow a particular structure to

Figure 9.1: Defining commands in the preamble of a document

```
\documentclass[12pt,twoside]{book}
\usepackage{color}
\usepackage[english]{babel}
\newcommand{\en}[1]{\textbf{#1}}
\newcommand{\fr}[1]{\textbf{\textit{#1}}}

\begin{document}
    \frontmatter
    ... % omitted for brevity
    \mainmatter
    ... % omitted for brevity
    \backmatter
    ... % omitted for brevity
\end{document}
```

enable LaTeX to manage them. Full details of this structure are discussed in *LaTeX 2ε for class and package writers* [11], but you can see a simple example of a package in Figure 9.2.

Figure 9.2: The `example.sty` package file

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{example}
\RequirePackage{color}

\newcommand{\en}[1]{\textbf{#1}}
\newcommand{\fr}[1]{\textbf{\textit{#1}}}
... % definition of other commands omitted for brevity
```

The `\NeedsTeXFormat` command specifies that the package is compatible with the LaTeX 2ε version of LaTeX. The `\ProvidesPackage` command specifies the name of the package. If the commands defined by the package rely upon other packages, then you should use `\RequirePackage` commands to state those dependencies. After that, you can use `\newcommand` to define commands.

When you have finished implementing a package, you can use it in a document by adding a `\usepackage` command in the preamble of the document. For example:

```
\usepackage{example}
```

9.6 Background Graphics

The `eso-pic` package [9] defines commands that enable you to place items on the background of a page. If you combine use of this package with the `color` or `graphicx` packages [3], then you can fill the background of a page—typically the title page—with a colour or a graphics file, such as a digital photograph. Canthology provides a simplification wrapper around the `eso-pic` package that makes it trivial to achieve such effects. The commands provided by Canthology are as follows:

```
\thisPageBackgroundCommand{\command}
\thisPageBackgroundColor{name}
\thisPageBackgroundColor[mode]{specification}
\thisPageBackgroundImage[options]{fileName}
```

You can see an example of the `\thisPageBackgroundCommand` command in Figure 9.3 on the next page.

Lines 6–13 append the `preamble` variable with the definition of a command called `\rectangleBackground`. The body of this command (lines 8–11) uses the `\AtPageLowerLeft` command (defined by the `eso-pic` package [9]) to execute two commands—`\color` and `\rule`—at the lower-left corner of the current page. The `\color` command (defined by the `color` package [3]) sets the foreground colour to be a pale shade of blue. The `\rule` command (documented in most LaTeX books and tutorials) draws a filled-in rectangle that extends one-third the width and the full height of the page.

Having defined the `\rectangleBackground` command, the next step is to execute it to set the background for the title page. You do this by passing the command's name as an argument to `\thisPageBackgroundCommand` (line 15). Figure 9.4 on page 89 shows the resulting title page.

The `\thisPageBackgroundColor` command fills the background of the entire page with the specified colour. For example, line 7 of Figure 9.5

Figure 9.3: The `\thisPageBackgroundCommand` command

```
@include getenv("CANTHOLOGY_HOME") + "/etc/defaults.cfg";
anthology1 {
    @copyFrom "book:a4";
    root_file {
        base_name = "my-anthology" + macro.paperSizeSuffix;
        preamble = preamble + [<%
            \newcommand{\rectangleBackground}{%
                \AtPageLowerLeft{%
                    \color[rgb]{0.8,0.8,1.0}%
                    \rule{.33\paperwidth}{\paperheight}%
                }%
            }
        %>];
        front_matter = [
            "\thisPageBackgroundCommand{\rectangleBackground}",
            "\input{titlepage-template-1.tex}",
            macro.tableofcontents,
        ];
        main_matter = [...];                # omitted for brevity
        back_matter = [...];                # omitted for brevity
    }
    substitutions.search_replace_pairs = [
        ...                                 # omitted for brevity
    ] + substitutions.search_replace_pairs;
}
```

illustrates how you can set a background colour for the title page. The result of doing so are shown in Figure 9.6 on page 91.

The argument to `\thisPageBackgroundImage` is the name of a file containing an image, for example, a digital photograph in JPEG format. The command scales the image to fit the height and width of the paper, and uses the `\includegraphics` command [3] to place this scaled image in the background. For example:

```
\thisPageBackgroundImage{front-cover}
```

If you specify optional arguments, then these will be passed as optional arguments to the `\includegraphics` command. The following example specifies that some of the image is to be trimmed off:

Figure 9.4: Title page with a blue rule on left side

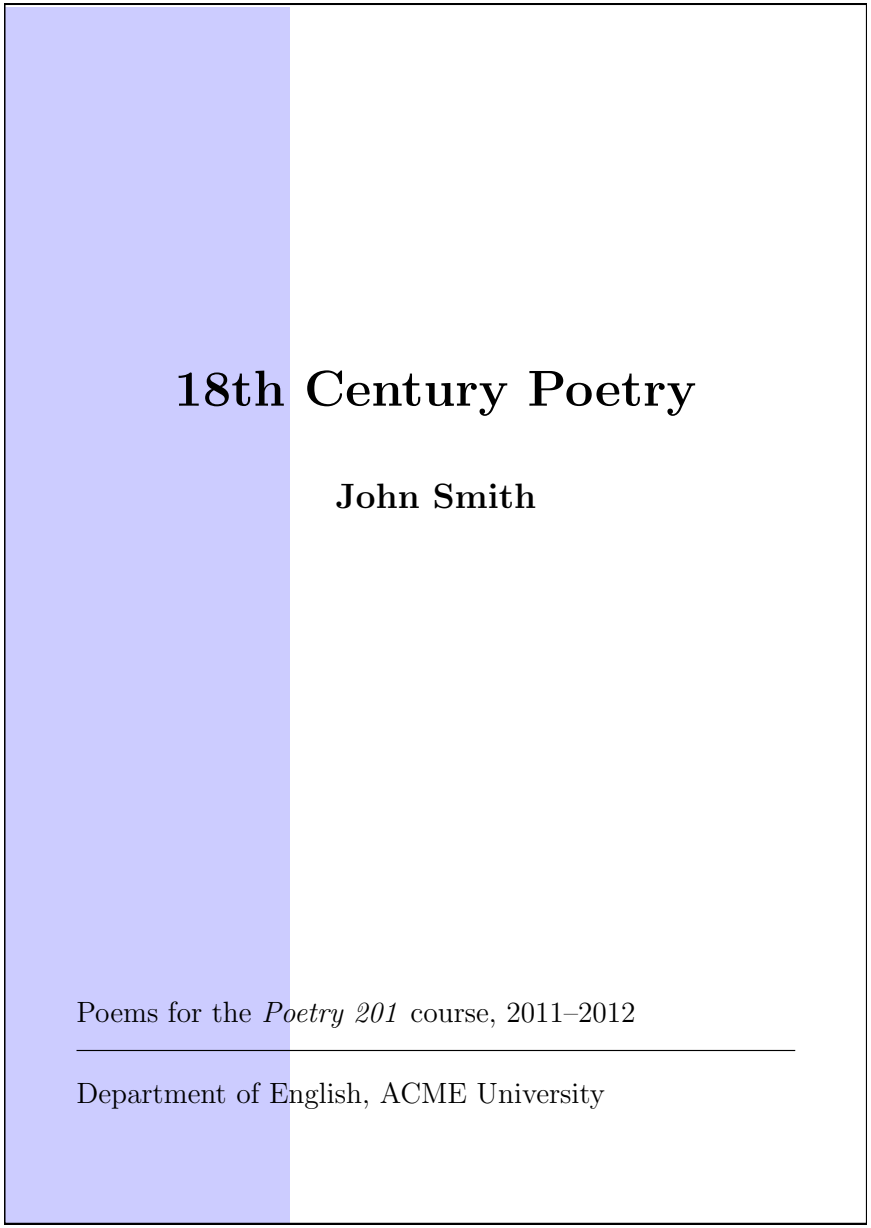


Figure 9.5: The `\thisPageBackgroundColor` command

```
@include getenv("CANTHOLOGY_HOME") + "/etc/defaults.cfg";
anthology1 {
    @copyFrom "book:a4";
    root_file {
        base_name = "my-anthology" + macro.paperSizeSuffix;
        front_matter = [
            "\thisPageBackgroundColor[rgb]{0.8,0.8,1.0}",
            "\input{titlepage-template-1.tex}",
            macro.tableofcontents,
        ];
        main_matter = [...];                # omitted for brevity
        back_matter = [...];                # omitted for brevity
    }
    substitutions.search_replace_pairs = [
        ...                                 # omitted for brevity
    ] + substitutions.search_replace_pairs;
}
```

```
\thisPageBackgroundImage[trim=10 20 30 40]{front-cover}
```

The commands discussed so far in this section are for manipulating the background of the *current* page. Canthology provides corresponding commands for manipulating the background of *every* page (starting with the current one):

```
\everyPageBackgroundCommand{\command}
\everyPageBackgroundColor{name}
\everyPageBackgroundColor[mode]{specification}
\everyPageBackgroundImage[options]{fileName}
```

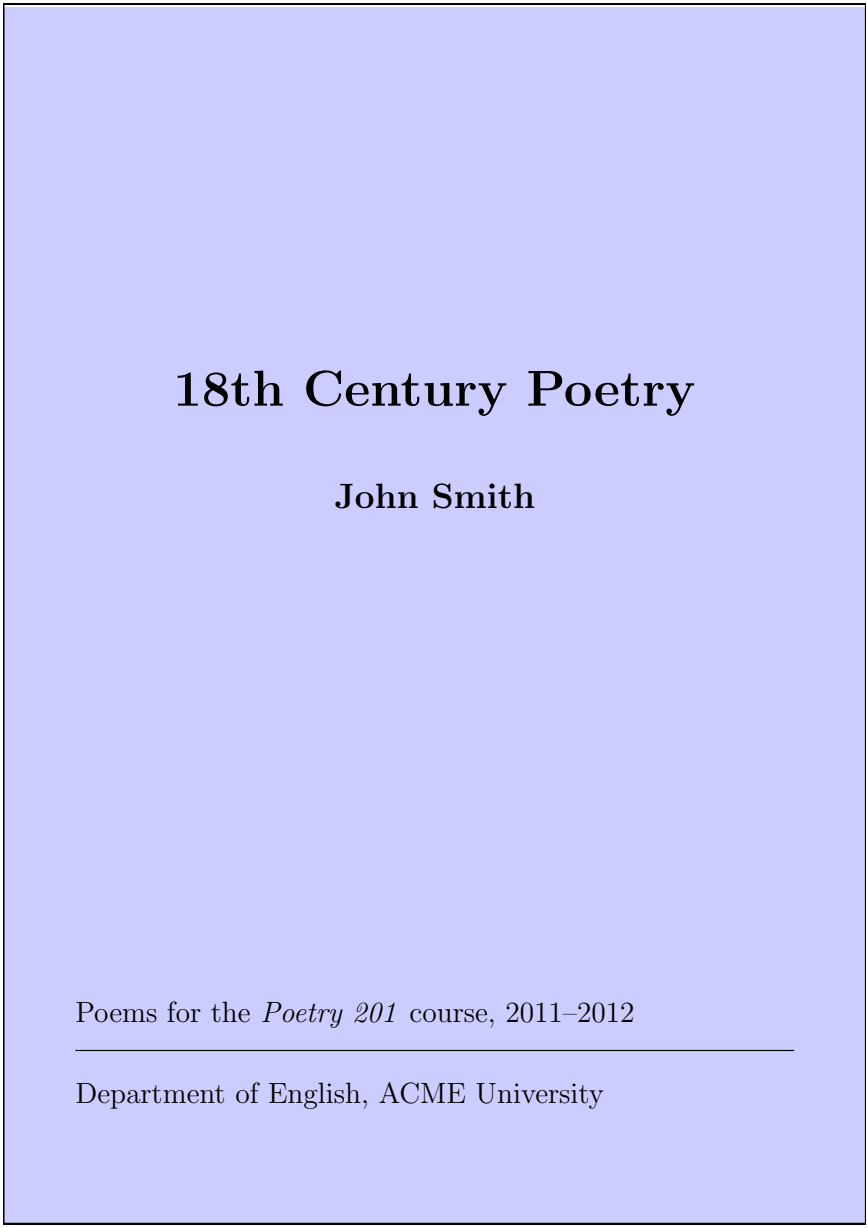
If you use one of the "every page background" commands, then later on you can clear the page background by using the following command:

```
\clearPageBackground
```

9.7 Creating a Document in Multiple Formats

When you write a document, you might want to make it available in multiple formats, for example, formatted for printing on US Letter paper (which

Figure 9.6: Title page with a colour background



is widely used in America), A4 paper (which is widely used in much of the rest of the world), and A5 paper (for on-screen viewing). This goal can be achieved easily with Canthology, as I explain in this section. The discussion focusses on Figure 9.7, which shows the outline of a Canthology configuration file.

Figure 9.7: Outline of a Canthology configuration file

```
1  @include getenv("CANTHOLOGY_HOME") + "/etc/defaults.cfg";
2  anthology1 {
3      @copyFrom "book:a4";
4      root_file {
5          base_name = "my-anthology" + macro.paperSizeSuffix;
6          preamble = preamble + [...];
7          front_matter = [...];
8          main_matter  = [...];
9          back_matter  = [...];
10     }
11     substitutions.search_replace_pairs = [
12         ...
13     ] + substitutions.search_replace_pairs;
14 }
```

A slightly tedious way to produce the document in multiple formats is to run Canthology several times, each time modifying line 3 of the configuration file to copy from a desired scope: `book:a4`, `book:a5` or `book:letter`. The use of `macro.paperSizeSuffix` in line 5 ensures that the generated PDF files will have the paper size embedded in their names.

By making a slight change to the configuration file, as shown in Figure 9.8, you can avoid the need to edit the file every time you want to change the paper size.

Line 2 is a *conditional* assignment statement: the value `"book:a4"` is assigned to the `format` variable *only if* that variable does not already have a value. By default, the `format` variable will not already have a value, so the assignment will take place. However, Canthology provides a "`-set` *`name value`*" command-line option that can be used to assign *`value`* to the *`name`* variable before parsing the configuration file. For example, running Canthology as follows will assign the value `"book:letter"` to `format`:

Figure 9.8: Multi-format Canthology configuration file

```
 1  @include getenv("CANTHOLOGY_HOME") + "/etc/defaults.cfg";
 2  format ?= "book:a4";
 3  anthology1 {
 4      @copyFrom format;
 5      root_file {
 6          base_name = "my-anthology" + macro.paperSizeSuffix;
 7          preamble = preamble + [...];      # omitted for brevity
 8          front_matter = [...];             # omitted for brevity
 9          main_matter  = [...];             # omitted for brevity
10          back_matter  = [...];             # omitted for brevity
11      }
12      substitutions.search_replace_pairs = [
13          ...                               # omitted for brevity
14      ] + substitutions.search_replace_pairs;
15  }
```

```
canthology -set format book:letter
```

The `@copyFrom` statement on line 4 copies from the scope specified by the value of the `format` variable.

The overall effect of those small changes is that the configuration file can now *adapt* itself to the desired output format specified by a command-line option. The following commands show how Canthology can be run three times to produce the document in three different output formats:

```
canthology -set format book:a4
canthology -set format book:a5
canthology -set format book:letter
```

Such a sequence of commands could be placed in a UNIX shell script, a Windows `".bat"` file, a `Makefile` or an Ant `build.xml` file. Doing this makes it possible to automate the generation of a document in multiple output formats.

9.8 The Copyright Page

You might want the page immediately following the title page of a book to contain a copyright notice. The file `example-copyright-template.tex`

(located in the `etc/latex` directory of your Canthology installation) provides a starting-point for such a page. Figure 9.9 on the next page shows what the file looks like when typeset.

Obviously, the typeset page shown is not suitable for use "as is" in a book. Instead, you should copy `example-copyright-template.tex` into the directory containing the `".tex"` files for your book, rename the copied file to be, say, `copyright.tex`, and then edit the file to suit your needs.

As you can see in Figure 9.9, the file contains copyright notices for two popular copyright licenses. If you want to use one of those in your book, then you can just delete the text of the unwanted license from your (copied) `copyright.tex` file.

To add `copyright.tex` to your book, you should edit your Canthology configuration file and modify the `root_file.front_matter` variable so it contains the line shown in a **bold** font below:

```
root_file {
    base_name = "my-anthology" + macro.paperSizeSuffix;
    front_matter = [
        "\input{titlepage-template-1.tex}",
        "\input{copyright.tex}",
        macro.tableofcontents,
    ];
    main_matter = [...];
    back_matter = [...];
}
```

9.9 Pages for Praise and a Dedication in a Book

Some books have a page near the start that quotes praising comments from book reviews. Canthology provides the `\praise` command for typesetting such comments. This command takes two parameters: the name of the person making the comment, and the text of the comment. For example, the following command:

```
\praise{Ann Other}{A very funny book. It contains many
```

Figure 9.9: Formatting of `example-copyright-template.tex`

Use the following text for the GNU Free Documentation Licence.

Copyright © (COPYRIGHT-YEAR-PLACEHOLDER) (COPYRIGHT-OWNER-PLACEHOLDER). Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in Appendix **??**.

Use the following text for the Creative Commons "share-alike" licence.

Copyright © (COPYRIGHT-YEAR-PLACEHOLDER) (COPYRIGHT-OWNER-PLACEHOLDER).

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

Use text like the following if you need to give acknowledgements for the use of, say, a cover photograph.

The cover photograph is the copyright of (COPYRIGHT-COVER-PHOTOGRAPH). Used with permission.

```
laugh-out-loud moments, and a hilarious build-up to a
farcical climax.}
```

is typeset as follows:

> "A very funny book. It contains many laugh-out-loud moments, and a hilarious build-up to a farcical climax."
>
> — Ann Other

Figure 9.10 on the next page shows the `example-praise-template.tex` file that you can find in the `etc/latex` directory of an installation of Canthology. The commands at the start of the file and the end of the file ensure that the page is typeset nicely in both PDF and HTML formats. You should ignore those commands, and just focus on the examples of the `\praise` commands.

The typeset output of `example-praise-template.tex` is shown in Figure 9.11 on page 98.

If you decide you want to have such a page near the start of your book, then you should copy the `example-praise-template.tex` file from the `$CANTHOLOGY_HOME/etc/latex` directory to the directory where you have the `".tex"` files for your book. You might want to rename the copied file to remove the `"example-"` prefix. Then edit the copied file to insert real reviewers' comments. Finally, edit your Canthology configuration file to add line 8 as shown in Figure 9.12 on page 99. Also ensure that the configuration file contains a substitution value for `"(TITLE-PLACEHOLDER)"` (line 16).

Figure 9.12 on page 99 also shows how to add a dedication page to your book. Line 9 inputs `dedication-template.tex`, and line 18 specifies the text of the placeholder to be used. The typeset result is shown in Figure 9.13 on page 100.

9.10 Cross References

When writing documents, I sometimes want to provide a cross-reference to another part of the document. For example, if I want to remind readers

Figure 9.10: The example-praise-template.tex file

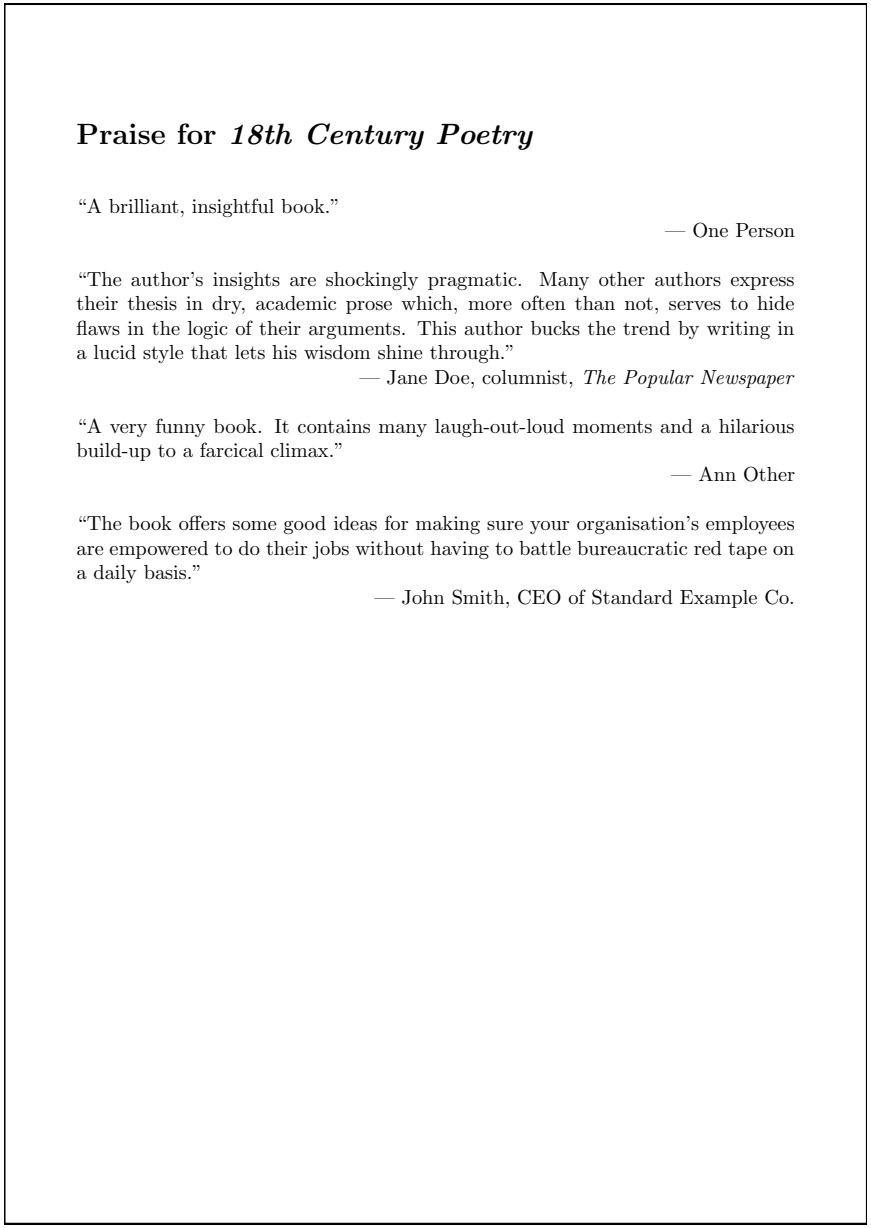
```
\cuthere{now}{Praise for \emph{(TITLE-PLACEHOLDER)}}
\cutname{praise.html}
\section*{Praise for \emph{(TITLE-PLACEHOLDER)}}
\thispagestyle{empty}

\praise{One Person}{A brilliant, insightful book.}

\praise{Jane Doe, columnist, {The Popular Newspaper}}{The author's
insights are shockingly pragmatic. Many other authors express their
thesis in dry, academic prose which, more often than not, serves to
hide flaws in the logic of their arguments. This author bucks the trend
by writing in a lucid style that lets his wisdom shine through.}

\praise{Ann Other}{A very funny book. It contains many laugh-out-loud
moments and a hilarious build-up to a farcical climax.}

\praise{John Smith, CEO of Standard Example Co.}{The book offers some
good ideas for making sure your organisation's employees are empowered
to do their jobs without having to battle bureaucratic red tape on a
daily basis.}

\newpage
```

how to install Canthology, then I might write "You can find details of this in Section 6.2 on page 42". In this section, I explain the basic LaTeX commands for creating such cross references and Canthology's simplification wrapper around those LaTeX commands.

Cross-referencing commands are unlikely to be useful if you are using Canthology to produce, say, an anthology of short stories—because the text in one short story is unlikely to need to cross reference another short story. However, if you are writing a technical article or a product manual, then you are much more likely to want to provide cross references between different parts of the document.

Most books on LaTeX explain how to do cross-referencing in a document. But for the benefit of readers who have yet to read such a book, I provide a brief overview in Section 9.10.1. Readers who are already

Figure 9.11: Formatting of `example-page-template.tex`



Praise for *18th Century Poetry*

"A brilliant, insightful book."

— One Person

"The author's insights are shockingly pragmatic. Many other authors express their thesis in dry, academic prose which, more often than not, serves to hide flaws in the logic of their arguments. This author bucks the trend by writing in a lucid style that lets his wisdom shine through."

— Jane Doe, columnist, *The Popular Newspaper*

"A very funny book. It contains many laugh-out-loud moments and a hilarious build-up to a farcical climax."

— Ann Other

"The book offers some good ideas for making sure your organisation's employees are empowered to do their jobs without having to battle bureaucratic red tape on a daily basis."

— John Smith, CEO of Standard Example Co.

Figure 9.12: Configuration to add praise and dedication pages to a book

```
@include getenv("CANTHOLOGY_HOME") + "/etc/defaults.cfg";
anthology1 {
    @copyFrom "book:a4";
    root_file {
        base_name = "my-anthology" + macro.paperSizeSuffix;
        front_matter = [
            "\input{titlepage-template-1.tex}",
            "\input{praise-template.tex}",
            "\input{dedication-template.tex}",
            macro.tableofcontents,
        ];
        main_matter = [...];
        back_matter = [...];
    }
    substitutions.search_replace_pairs = [
       "(TITLE-PLACEHOLDER)",       "18th Century Poetry",
       "(AUTHOR-PLACEHOLDER)",      "John Smith",
       "(DEDICATION-PLACEHOLDER)",  "For Sam,\\the love of my life.",
        ...
    ] + substitutions.search_replace_pairs;
}
```

familiar with the `\label`, `\ref` and `\pageref` commands can skip to Section 9.10.2 on page 102.

9.10.1 The `\label`, `\ref` and `\pageref` Commands

Creating cross references in a LaTeX document is a two-step process.

1. You use the `\label` command to define a *label* (that is, name) for "something" in a document. The "something" might be a part, chapter, section, subsection, floating figure or a table.

2. Once a label has been defined, you can then use the `\ref` or `\pageref` commands to provide a cross reference to it.

You can see some examples of how to define labels in Figure 9.14 on page 101.

Figure 9.13: Formatting of `dedication-template.tex`

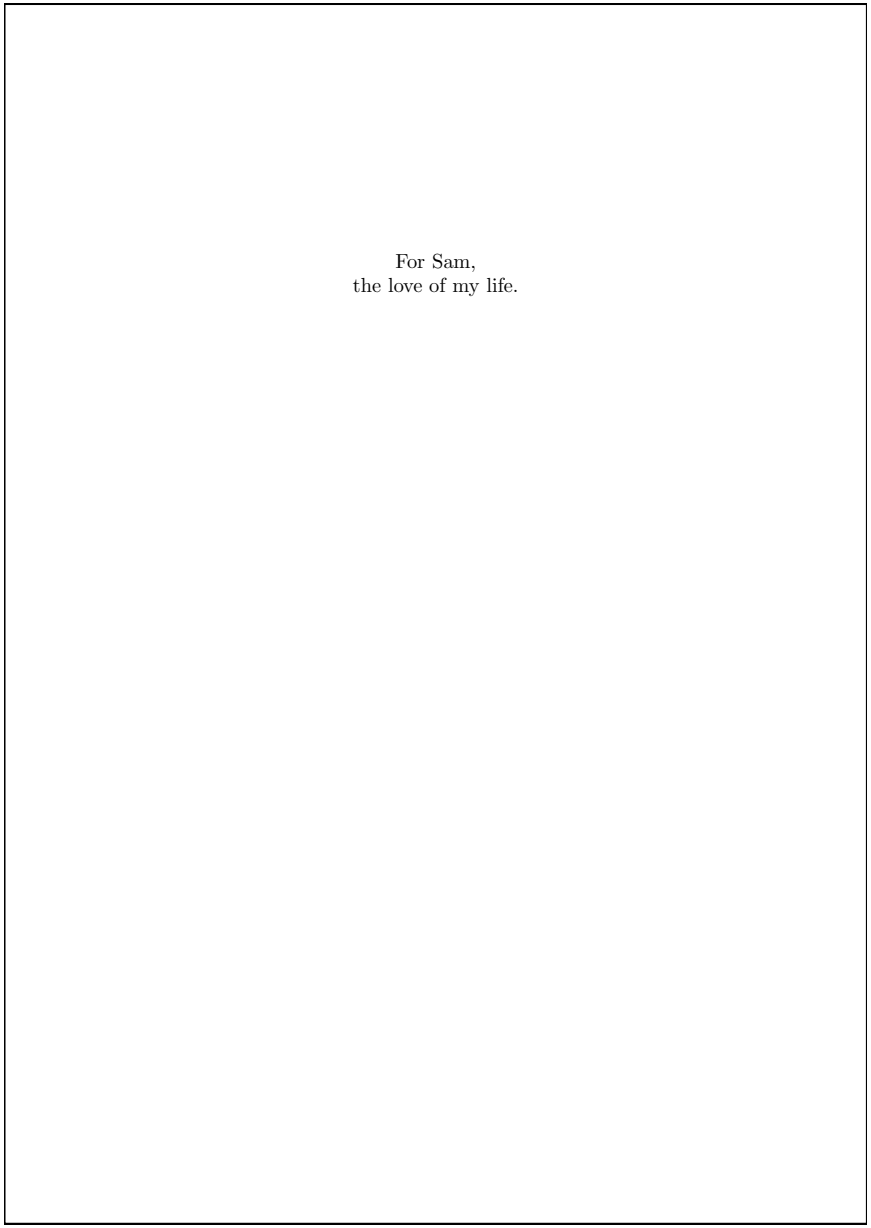


Figure 9.14: Examples of the \label command

```
\chapter{How Canthology Operates}
\label{ch:op}

\section{Introduction}
\label{sect:op:intro}
... % text in section omitted for brevity

\section{Configuration File and Scopes}
\label{sect:op:cfg-file-and-scopes}
... % text in section omitted for brevity
```

Each time you execute a command such as \part, \chapter, \section, \subsection or \caption (used in floating figures and tables), it increments some counters specific to that command. The \label command associates its parameter with the most recently updated counter values.

The \ref command converts a label into the label's counter value. The example below uses the labels defined in Figure 9.14 as parameters to the \ref command:

```
Chapter~\ref{ch:op} starts with
Section~\ref{sect:op:intro} and then continues with
Section~\ref{sect:op:cfg-file-and-scopes}.
```

LaTeX typesets that as follows:

> Chapter 8 starts with Section 8.1 and then continues with Section 8.2.

There are two points worth noting about the above example.

First, ~ is used to insert a non-breakable space between "Chapter" or "Section" and the following number. This is because professional typesetters consider it bad style to allow a line break at such a place.

Second, although you can use whatever names you want for labels, you will find it easier to maintain documents if the names of labels follow a consistent naming scheme. I like to use "ch:" as a prefix for chapter labels, "sect:" as a prefix for section and subsection labels, "fig:" as a prefix for captions in floating figures, and so on. After the prefix, I like the

remainder of the label to be an abbreviation of the name of the chapter, section, figure or whatever.

The `\pageref` command converts a label into a page number. It is typically used as follows:

```
See Section~\ref{sect:op:cfg-file-and-scopes} on
page~\pageref{sect:op:cfg-file-and-scopes} for more
details.
```

9.10.2 Convenience Commands for Cross Referencing

It is possible to use `\newcommand` to define more concise commands for cross-referencing as the example below illustrates:

```
\newcommand{\xp}[1]{Part~\ref{#1}}
\newcommand{\xa}[1]{Appendix~\ref{#1}}
\newcommand{\xc}[1]{Chapter~\ref{#1}}
\newcommand{\xs}[1]{Section~\ref{#1}}
\newcommand{\xf}[1]{Figure~\ref{#1}}
\newcommand{\xt}[1]{Table~\ref{#1}}
\newcommand{\xpp}[1]{Part~\ref{#1} on page~\pageref{#1}}
\newcommand{\xap}[1]{Appendix~\ref{#1} on
                     page~\pageref{#1}}
\newcommand{\xcp}[1]{Chapter~\ref{#1} on
                     page~\pageref{#1}}
\newcommand{\xsp}[1]{Section~\ref{#1} on
                     page~\pageref{#1}}
\newcommand{\xfp}[1]{Figure~\ref{#1} on
                     page~\pageref{#1}}
\newcommand{\xtp}[1]{Table~\ref{#1} on
                     page~\pageref{#1}}
```

The naming scheme used in the above commands is as follows.

- The first letter of the command is `x`, which English speakers sometimes use as an abbreviation for “cross”. In the above commands, it is used as an abbreviation of “cross reference”.

- The next letter indicates what is being cross-referenced. Thus `\xc` produces a cross reference to a chapter, `\xs` produces a cross reference to a section, and so on.
- If the command name consists of three letters ending in `p`, then the comamnd prints "on page ..." after the cross reference.

Earlier, I gave some examples of using the `\ref` and `\pageref` commands:

```
Chapter~\ref{ch:op} starts with
Section~\ref{sect:op:intro} and then continues with
Section~\ref{sect:op:cfg-file-and-scopes}.

See Section~\ref{sect:op:cfg-file-and-scopes} on
page~\pageref{sect:op:cfg-file-and-scopes} for more
details.
```

Using the newly defined `\xc`, `\xs` and `\xsp` commands, those examples can be written more compactly:

```
\xc{ch:op} starts with \xs{sect:op:intro} and then
continues with \xs{sect:op:cfg-file-and-scopes}.

See \xsp{sect:op:cfg-file-and-scopes} for more details.
```

There is further room for improvement. The `varioref` package [7] defines a command called `\vref`, that I will explain by an example. Consider the following statement:

```
I discuss that topic in Section~\vref{sect:something}.
```

LaTeX may typeset that sentence in one several ways, including:[1]

1. I discus that topic in Section 3.9.
2. I discus that topic in Section 3.9 on page 42.

[1] The `varioref` package provides the ability to customise the different variations of wording, but I will discuss only the above examples to give a flavour of the package's ability.

3. I discus that topic in Section 3.9 on the facing page.
4. I discus that topic in Section 3.9 on the next page.
5. I discus that topic in Section 3.9 on the following page.
6. I discus that topic in Section 3.9 on the previous page.
7. I discus that topic in Section 3.9 on the preceding page.

The first wording might be used if the referenced label is on the same page. If the referenced label is two or more pages away, then the second wording is used. If the document is printed in two-sided mode, then the `varioref` package assumes the document will be bound like a book, which means a reader can see two pages at a time. In this case, if the referenced label is on the other visible page, then the third wording (that is, "on the facing page") is used. If the referenced label is only one page away but is *not* on the facing page, then one of the last four variations of wording is used.

If we want to take advantage of this flexibility offered by the `varioref` package, then we can alter the definitions of the `\xpp`, `\xap`, `\xcp`, `\xsp`, `\xfp` and `\xtp` commands as shown below:

```
\newcommand{\xp}[1]{Part~\ref{#1}}
\newcommand{\xa}[1]{Appendix~\ref{#1}}
\newcommand{\xc}[1]{Chapter~\ref{#1}}
\newcommand{\xs}[1]{Section~\ref{#1}}
\newcommand{\xf}[1]{Figure~\ref{#1}}
\newcommand{\xt}[1]{Table~\ref{#1}}
\newcommand{\xpp}[1]{Part~\vref{#1}}
\newcommand{\xap}[1]{Appendix~\vref{#1}}
\newcommand{\xcp}[1]{Chapter~\vref{#1}}
\newcommand{\xsp}[1]{Section~\vref{#1}}
\newcommand{\xfp}[1]{Figure~\vref{#1}}
\newcommand{\xtp}[1]{Table~\vref{#1}}
```

Canthology implements the above commands as shown.

Since the `\vref` command is so flexible, you may wonder why Canthology provides two variations of each cross-referencing command, for

example, the flexible `\xsp` along with its less flexible `\xs` counterpart. There are two reasons for this.

First, let's assume I am writing a paragraph in which I discuss "Figure 2.9 on page 26". The first time I mention the Figure 2.9, I want to specify its page number as a convenience to readers, so I use the `\xfp` command for the cross reference. But if I mention the figure a second (and possibly even a third) time in the same paragraph, then it is redundant (and irritating to readers) to keep specifying its page number, so I use the `\xf` command instead.

Second, there is a boundary case that can sometimes cause `\vref` to fail, in which case LaTeX reports an error message and stops. To understand this, assume that on page 25 of a document, I want to cross-reference Figure 2.9 that happens to be on the next page. In this case, we might expect:

```
Figure~\vref{fig:something}
```

to produce:

> Figure 2.9 on the next page

Most of the time, that is what happens. However, a problem occurs if the text "Figure 2.9 on the next page" is being typeset at the very bottom of page 25 and a page break occurs between "Figure 2.9" and "on the next page". In this case, the text "on the page next" would no longer be valid since that text appears on the *same* page as the figure. In such a case, the `\vref` command reports an error and LaTeX stops. On the rare occasion when this happens, it is useful to replace use of the `\xfp` command with `\xf` to eliminate the error message.

9.10.3 Simplifying Cross References Produced by `\vref`

Although the `varioref` package is very flexible, its assumption that a two-sided document will be bound like a book—and thus two pages will be visible at the same time—is not always correct. For example, a two-sided document may be stapled in the top left-hand corner. Or you might read a PDF version of the document on a computer screen, using a PDF viewer

that is configured to show only one page at a time. In such cases, it can be annoying to read, “Figure 2.9 on the facing page” because there *isn't* a visible facing page.

Another potential annoyance is that the `varioref` package likes to add some variation into the text used for cross references. For example, the `\vref` command sometimes might claim that Figure 2.9 is “on the *next* page” and other times might claim that it is “on the *following* page”. Such variations in phrasing might seem like a good idea, because it helps to avoid monotony. However, it is possible for such a variation in wording to occur between different runs of LaTeX, and the difference in length of the text might cause a bad line break or a page break at unpredictable times. For this reason, it can be useful to intruct the `varioref` package to *not* use variations in phrasing.

The `canthology` package defines a command that instructs `varioref` to avoid: (1) using the phrase “on the facing page”, and (2) alternating between “on the previous/next page” and “on the preceding/following page”. The name of this command is `\simplifyVariorefReferences`.

Part IV

Using Canthology to Generate HTML Files

Introduction to Part IV

If you have read all the preceding chapters, then you know how to use Canthology to create documents in PDF format. Now, in Part IV, I explain some additional information you will need to know if you want to use Canthology to create documents in HTML format.

Chapter 10

Overview of HEVEA

10.1 Introduction

Canthology uses the HEVEA application [6] to convert LaTeX documents into HTML format.[1] In this chapter, I provide background information about HEVEA that is relevant to users of Canthology. This chapter is *not* intended to be a substitute for reading the HEVEA manual, but rather complements it. My advice is to read this chapter to get an overview of HEVEA and how to use it with Canthology. Afterwards, you should read the HEVEA manual to learn how to use HEVEA properly.

Currently, the use of HEVEA with Canthology is not supported on Windows.

10.2 LaTeX-to-HTML Converters

There are several (competing) tools for converting LaTeX-based documents into HTML, including LaTeX2HTML, TtH, TeX4ht, PlasTeX and HEVEA. None of these tools has a 100% success rate in such conversions. This is because there are some LaTeX (and TeX) constructs that have no counterparts in HTML. Nevertheless, a large subset of commonly-used LaTeX

[1] The name HEVEA is a pun on LaTeX. *Hevea Brasilliensis*, also known as *the Pará rubber tree* or simply the *rubber tree*, produces a sap-like substance called *latex*, which is a source of natural rubber.

commands *do* have counterparts in HTML. LaTeX-to-HTML converters tend to work well with LaTeX documents that restrict themselves to such commands.

Some converters provide a way for users to extend the converter with rules for processing LaTeX commands for which the converter does not provide built-in support. Thus, use of a LaTeX-to-HTML converter tends to involve an iterative approach:

1. Run the converter on a LaTeX document.
2. Examine the generated HTML to see if any LaTeX commands were not translated properly. If so, then:
3. Extend the converter with a new rule that enables a troublesome LaTeX command to be correctly converted to HTML. (If that is not possible, then rewrite the LaTeX document to avoid the need to use the troublesome LaTeX command.) Then go back to step 1.

Use of a LaTeX-to-HTML converter can be frustrating, at least initially, because you are likely to spend a lot of time working on step 3 in the above list. However, you will eventually learn which LaTeX commands can be used without causing difficulty for a HTML conversion. This knowledge will make it far easier to write future LaTeX documents that are compatible with the converter.

My favourite LaTeX-to-HTML converter is called HeVeA. In this chapter, I explain how to use Canthology with HeVeA.

10.3 Obtaining and Installing HeVeA

HeVeA is bundled with some LaTeX distributions and available in the application repositories for some Linux distributions. If you cannot get HeVeA from those sources, then you can download it from the HeVeA website.[2] HeVeA was designed for use on Linux and other UNIX-like operating systems. However, a Windows port of HeVeA is also available.[3]

[2] http://hevea.inria.fr/

[3] http://facweb.knowlton.ohio-state.edu/pviton/support/winport.html

10.4 Running HEVEA

The HEVEA distribution contains several several applications that are intended to be used together.

hevea This application translates a `".tex"` document into a monolithic HTML file.

hacha This application splits a monolithic HTML file produced by `hevea` into a collection of HTML files. By default, `hacha` creates a separate HTML file for each chapter of a book, or a separate HTML file for each section of an article. However, commands (defined in the `hevea` package) that you can embed in your `".tex"` document enable you to choose a different granularity of division.

imagen The `hevea` application detects when a `".tex"` document uses image files, for example, in `\includegraphics` commands. Details of these image-using commands are written to a new temporary `".tex"` file. The `imagen` application is then run on this temporary file to convert all the images into a HTML-friendly format, such as GIF or PNG.

esponja A HTML file generated by `hevea` contains HTML markup that can sometimes be needlessly verbose. The `esponja` utility can be run to optimise the HTML markup, thus decreasing the size of the HTML file.

bibhva LATEX has a companion application called `bibtex` for managing bibliography information. Unfortunately, subtle differences between the behaviour of LATEX and HEVEA mean that `bibtex` does not work unaided with HEVEA. The `bibhva` application acts as a wrapper around `bibtex` to resolve this incompatibility.

All but one of the commands described above are compiled applications. The exception is `imagen`, which is a UNIX shell script, and its correct working relies on the presence of many other commands (such as `gs` and `pnmtopng`) that are commonly available on UNIX systems. Such commands are not available by default on Windows. For this reason, HEVEA

can be used with ".tex" documents that may contain graphics on UNIX, but can be used only with image-less ".tex" documents on Windows. Canthology makes this situation even worse: its use of HeVeA relies upon the use of some commands (such as GNU make and a Tcl interpreter) that are widely available on UNIX machines, but are not installed by default on Windows. For this reason, using Canthology with HeVeA is *not* currently supported on Windows.

10.5 How HeVeA Handles Unrecognised Commands

HeVeA implements a large subset of commonly-used LaTeX commands and a small subset of lower-level TeX commands. When HeVeA encounters a command it does not recognise, it prints a warning diagnostic message to the console and ignores the command's name, but it copies the command's arguments, if any, to the output file. For example, assume HeVeA encounters the following in a ".tex" document:

```
I am \emp{very} happy to see you.
```

In this case, HeVeA does not recognise the \emp command (it is a typo, and should be \emph), so it prints a warning message on the console and writes the following to the output HTML file:

I am very happy to see you.

(When using HeVeA with Canthology, you will need to run canthology with the "-d 3" command-line option to see the warning messages reported by HeVeA.)

In practice, many unrecognised commands are due to HeVeA implementing only a subset of LaTeX and TeX commands (rather than typos in the input document).

The "print a warning message and carry on" behaviour can be useful, especially if you are new to HeVeA. This is because it enables you to see that HeVeA correctly converts 90% of your document to HTML, and most of the fouled-up 10% is due to just a small number of unrecognised commands that are used frequently. Thus, if you can (somehow) extend HeVeA

to handle those troublesome commands, then HEVEA will be able to process your entire document.

10.6 Extending HEVEA

Extending HEVEA with new commands usually revolves around implementing packages. You may recall from Section 9.5 on page 85 that LaTeX packages are implemented with `".sty"` files. However, HEVEA ignores `".sty"` files because they are likely to make use of low-level LaTeX or TeX commands that are not supported by HEVEA. Instead, HEVEA packages are implemented in `".hva"` files, which may make use of the subset of LaTeX commands implemented by HEVEA *plus* some additional CSS- and HTML-centric commands provided by HEVEA. Thus, if you want to write a `".hva"` file, you are likely to need to have a working knowledge of HTML and CSS (*cascading style sheets*). The discussion in this section assumes such as working knowledge.

Among HEVEA's CSS- and HTML-centric commands are the following:

```
\newstyle{name}{settings}
\@open{BLOCK}{attributes}
\@close{BLOCK}
```

The `\newstyle` command generates a new CSS style called *`name`* that has the specified *`settings`*. For example, the following code:

```
\newstyle{.example}{margin-left: 4ex; margin-right: 4ex;}
```

results in the following being added to the style sheet for the generated HTML document:

```
.example {margin-left: 4ex; margin-right: 4ex;}
```

The `\@open` command creates an opening tag, containing the specified *`attributes`*, for the specified *`BLOCK`*. Conversely, `\@close` creates a closing tag for the specified *`BLOCK`*. For example, the following code:

```
\@open{DIV}{class="example"}
...
\@close{DIV}
```

adds the following to the output HTML file:

```
<DIV class="example">
...
</DIV>
```

I will now provide a complete worked example to illustrate typical usage of the above HTML-centric commands. I will start by explaining the functionality offered by the `framed` package. Then, I will discuss how a HeVeA version of this package (that is, a file called `framed.hva`) can be implemented.

10.6.1 The `framed` package

The `framed` package [1] defines three environments called `framed`, `shaded` and `leftbar`. For example, the following code:

```
\setlength{\FrameSep}{3pt}
\begin{framed}
    \noindent
    This sentence is boring; it just contains some
    example text. This sentence is boring; it just
    contains some example text.  This sentence is
    boring; it just contains some example text. This
    sentence is boring; it just contains some example
    text.
\end{framed}
```

produces the following output:

This sentence is boring; it just contains some example text. This sentence is boring; it just contains some example text. This sentence is boring; it just contains some example text. This sentence is boring; it just contains some example text.

As you can see, the `framed` environment draws a frame around the contents of the environment. By default, there is a lot of space between the frame and the text it contains. However, in the above example I used the command `\setlength{\FrameSep}{3pt}` to reduce that space, so the frame fits more snugly. In addition, I used `\indent` to prevent the first (and, in this case, only) paragraph within the frame starting with an indented line.

The `shaded` environment does not draw a frame around the contents of its environment. Instead, it paints the background with the color specified by `shadecolor`, which the author of a document must previously have defined via the `\definecolor` command [3]. For example, the following code:

```
\definecolor{shadecolor}{rgb}{0.9,0.9,1.0}
\begin{shaded}
    \noindent
    This sentence is boring; it just contains some
    example text. This sentence is boring; it just
    contains some example text.  This sentence is
    boring; it just contains some example text. This
    sentence is boring; it just contains some example
    text.
\end{shaded}
```

produces the following output:

This sentence is boring; it just contains some example text. This sentence is boring; it just contains some example text. This sentence is boring; it just contains some example text. This sentence is boring; it just contains some example text.

The `leftbar` environment draws a thick black line on the left side of the contents of the environment. For example, the following code:

```
\begin{leftbar}
    \noindent
```

```
    This sentence is boring; it just contains some
    example text. This sentence is boring; it just
    contains some example text.  This sentence is
    boring; it just contains some example text. This
    sentence is boring; it just contains some example
    text.
\end{leftbar}
```

produces the following output:

> This sentence is boring; it just contains some example text. This sentence is boring; it just contains some example text. This sentence is boring; it just contains some example text. This sentence is boring; it just contains some example text.

In addition to the `framed`, `shaded` and `leftbar` environments, the package defines commands that can be used as building blocks for users to define other environments. However, a discussion of those additional commands is outside the scope of this manual.

10.6.2 Implementing `framed.hva`

Although the HeVeA distribution provides `".hva"` implementations of more than twenty popular packages, it does not provide an implementation of the `framed` package. To work around this, I wrote the HeVeA implementation shown in Figure 10.1 on the next page.

The `\newenvironment` command takes three parameters. The first is the name of the environment being defined. The second and third parameters specify commands to be executed at the beginning and ending of the environment.

The definition of the `framed` environment (lines 4–8) uses the `\@open` and `\@close` commands to open and close an HTML `DIV` element whose `class` attribute has the value `"framed"`. The `\newstyle` command on lines 9–15 defines the `framed` CSS class that specifies a one-pixel-wide, solid black border is drawn around the `DIV` element, with 8 pixels of padding between the border and the left and right margins of the text.

Figure 10.1: The `framed.hva` file

```
\ProvidesPackage{framed}
\RequirePackage{color}

\newenvironment{framed}{%
    \@open{div}{class="framed"}%
}{%
    \@close{div}%
}
\newstyle{.framed}{
    border: 1px solid black;
    padding-left: 8pt;
    padding-right: 8pt;
    padding-top: 0pt;
    padding-bottom: 0pt;
}

\newenvironment{leftbar}{%
    \@open{div}{class="leftbar"}
}{%
    \@close{div}
}
\newstyle{.leftbar}{
    border-left: 4px solid black;
    padding-left: 6pt;
    padding-right: 6pt;
    padding-top: 0pt;
    padding-bottom: 0pt;
}

\newenvironment{shaded}{%
    \@open{TABLE}{BORDER="0" CELLPADDING="8" WIDTH="100\%"
                  BGCOLOR=\@getcolor{shadecolor}}
    \@open{TR}{}
    \@open{TD}{}
}{
    \@close{TD}
    \@close{TR}
    \@close{TABLE}
}
```

The `leftbar` environment (lines 17–21) and its accompanying CSS style (lines 22-28) are defined in a similar manner.

The definition of the `shaded` environment (lines 30–39) is more verbose. It creates a `TABLE` element, containing a single cell, in which the background color (that is, the `BGCOLOR` attribute) is set to the value of `\@getcolor{shadecolor}`. The low-level `\@getcolor` command converts the specified color (passed as a parameter) from the specification syntax used by the `color` package into the syntax used in HTML files.

HeVeA does not implement the LaTeX concept of a "length". Instead, it ignores all uses of the `\newlength`, `\setlength` and `\addtolength` commands. For this reason, the `framed.hva` file does not (and cannot) implement support for the `\FrameSep` length. Thus, there is no way for the author of a document to alter the spacing between the box of the `framed` environment and the text inside it. Instead, this spacing is fixed by the CSS `framed` class.

The above worked example illustrates several issues that commonly arise when implementing `".hva"` versions of packages.

First, a person implementing a `".hva"` file may need to have a working knowledge of HTML and CSS.

Second, the implementation of commands in a `".hva"` file is often simpler than their corresponding implementation in a `".sty"` file. This is because HTML and CSS are much simpler (and less powerful) markup languages than LaTeX. In the case of the `framed` package, the LaTeX version is complex because it needs to deal with the possibility of an environment spanning a page break. The HTML implementation of the package is simpler because it does not have to worry about such page breaks.

Finally, it is common for a `".hva"` file to implement a *subset* of the functionality provided in a package. For example, `framed.hva` ignores attempts to set the `\FrameSep` length, and it does not provide building-block commands for users to define other `framed`-like environments.

10.7 The `hevea` Package

A package called `hevea` is distributed with HEVEA. The HEVEA implementation of the `hevea` package defines about 30 commands that enable you to fine-tune how a document is converted into HTML. The LaTeX implementation of the package provides empty implementations of those same commands. For example, the LaTeX implementation of the `\newstyle` command (discussed in Section 10.6 on page 113) does nothing.

10.8 The `latexonly` and `htmlonly` Environments

The `hevea` package defines environments called `latexonly` and `htmlonly`. Any text you place in a `latexonly` environment is used in the document only if you process the document with a LaTeX-related command, such as `latex` or `pdflatex`. Conversely, any text you place in a `htmlonly` environment is used in the document only if you process the document with `hevea`. The following example illutrates the use of those commands:

```
Sentence~1 is in all versions of this manual.
\begin{latexonly}
Sentence~2 is in only the PDF version of this manual.
\end{latexonly}
\begin{htmlonly}
Sentence~3 is in only the HTML version of this manual.
\end{htmlonly}
```

That example results in the following output in the version of this manual you are currently reading:

> Sentence 1 is in all versions of this manual. Sentence 2 is in only the PDF version of this manual.

Although it is good to know that the `latexonly` and `htmlonly` environments exist, in practice, you are likely to use them very infrequently, if at all. For example, this manual is over 200 pages long, yet aside from the above example, I have used the `latexonly` and `htmlonly` environments

just three times. In each case, it was to make a minor adjustment to the visual appearance of the formatted document.[4]

It is important to note that environments work *only* in the main part of a document, that is, between `\begin{document}` and `\end{document}`. In particular, you *cannot* use the `latexonly` and `htmlonly` environments in the preamble of a document to help you define LaTeX- and HeVeA-specific versions of commands. Instead, if you need to define LaTeX- and HeVeA-specific versions of some commands, then you should write `".sty"` and `".hva"` versions of a package. You can find a discussion of how to write packages in Section 9.5 on page 85 and also Section 10.6 on page 113.

10.9 Specifying the Names of HTML Files

Let's assume you run `hacha` to convert a document into a monolithic HTML file, and afterwards you run `hacha` on that file to split it into multiple HTML files. If your document uses the `book` class, then `hacha` creates a separate HTML file for each chapter. Conversely, if your document uses the `article` class, then `hacha` creates a separate HTML file for each section.

By default the names of the HTML files will be the name of the root file of your document (without the `".tex"` extension) followed a three-digit number and then `".html"`. For example, if the root file of your document is `short-stories.tex`, then the HTML files created by `hacha` will have names like `short-stories001.tex`, `short-stories002.tex`, `short-stories003.tex`, and so on.

You can override the default naming scheme by using the `\cutname` command, which is defined in the `hevea` package. This command takes one parameter that specifies the file name that `hacha` should use when splitting the current part of the document into a separate HTML file. You should put a `\cutname` command after each `\chapter` command in a book, or after each `\section` command in an article. For example:

[4] In one case, I used the `latexonly` environment to introduce some extra vertical spacing at the top of the dedications page in the PDF version of this manual. In the other two cases, I wanted to force a line break in the PDF version of this manual, but not have such a line break in the HTML version.

```
\chapter{The Adventure Starts}
\cutname{start.html}
...
\chapter{Disaster Strikes}
\cutname{disaster-strikes.html}
...
\chapter{A Happy Ending}
\cutname{happy-ending.html}
...
```

10.10 Misfeatures of HEVEA

Although HEVEA is very useful, it does have some misfeatures. In this section, I discuss the workarounds that I have developed for some of these.

10.10.1 The `hevea-fix` package

I have implemented a package called `hevea-fix`, which I distribute with Canthology. The `".hva"` implementation of the package redefines some HEVEA commands so they mimic their LATEX counterparts more closely. The `".sty"` implementation of this package is empty because it does not need to modify LATEX. What follows is a brief summary of the capabilities provided by `hevea-fix.hva`.

Unfortunately, HEVEA does not implement `\frontmatter`, `\mainmatter` or `\backmatter`. Because of this, front and back matter such as a preface or glossary appear as *numbered* (rather than as *unnumbered*) chapters or sections. The `hevea-fix.hva` file fixes this problem.

The HEVEA implementation of the `hyperref` package fails to implement the `\phantomsection` command; `hevea-fix.hva` fixes this problem.

The `hevea` application inserts comments into the generated HTML file. These comments instruct `hacha` how to divide the monolithic HTML file into a collection of HTML files—by default, a separate file for each chapter. A bug or misfeature in the `hacha` application causes it to process comments relating to `\part` commands in a strange and unsatisfactory

manner. The `hevea-fix.hva` file resolves this by redefining the HeVeA command that inserts comments into the generated HTML file. The result is that `\part`-related comments are now identical to `\chapter`-related comments, so `hacha` processes them in a satisfactory manner.

LaTeX uses (only) vertical spacing to separate a floating `figure` or `table` from the main body of text. Some authors prefer to have a more visible boundary between a `figure` or `table` and the main body of text. For example, the visible boundary might be a box (such as that provided by the `framed` environment) surrounding the contents of the `figure`, or perhaps horizontal lines drawn above and below the `figure` or `table`. Such authors have to explicitly use LaTeX commands to draw such boundaries. In contrast, HeVeA automatically puts a visible boundary around a `figure` or `table`. This boundary takes the form of horizontal lines above and below the `figure` or `table`. This is a problem because the presence of the HeVeA-provided boundary often visually clashes with whatever boundary has been provided by the author using LaTeX commands. The `hevea-fix.hva` file fixes this problem by redefining a low-level HeVeA command so the horizontal lines are *not* drawn.

LaTeX has commands that can be used to control the co-existence of text and floating figures or tables on a page: `\topfraction`, `\bottomfraction`, `\textfraction` and `\floatpagefraction`. HeVeA neglects to define those commands, which results in warning messages being printed if those commands are used in a document. The `hevea-fix.hva` file fixes this problem by defining dummy versions of those commands.

In LaTeX, there are subtle differences in how the `quote`, `quotation` and `verse` environments are typeset. HTML is not flexible enough to reproduce those subtle differences. For this reason, HeVeA maps both the `quote` and `quotation` environments into a `BLOCKQUOTE` HTML element. That is perfectly reasonable. What I find strange is that HeVeA chooses to not support the `verse` environment. The `hevea-fix.hva` file rectifies this by also mapping the `verse` environment into a `BLOCKQUOTE` HTML element.

10.10.2 Removing the Pseudo Table of Contents

HEVEA implements the `\tableofcontents` command. Figure 10.2 shows a table of contents generated for a document that contains two parts, each with three chapters.

Figure 10.2: Table of contents produced by `hevea`

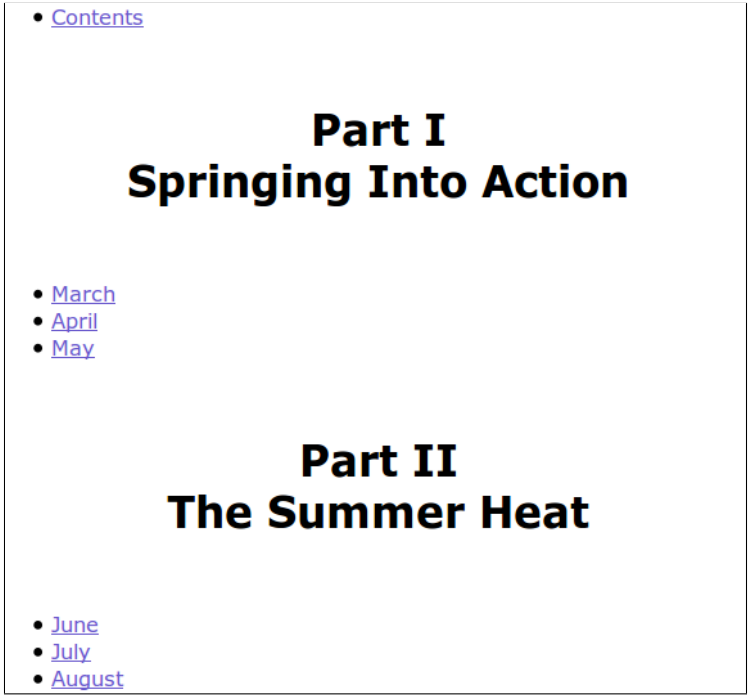
I think you will agree that the table of contents looks nice. Unfortunately, something bizarre happens when you run `hacha` to split a HTML file created by `hevea` into a collection of smaller files. The `hacha` application adds HTML like that shown in Figure 10.3 on the next page to the first HTML file, which is typically the title page of your document.

The result is that your document contains a badly-formatted, *pseudo* table of contents on the title page, plus a *real* table of contents on another HTML page. By the way, the first entry in the pseudo table of contents is a link to the real table of contents!

Unfortunately, `hacha` does not have a command-line option to disable the generation of the pseudo table of contents. However, I have written a script called `remove_pseudo_toc.tcl` that can remove the pseudo table of contents and (optionally) replace it with a link to the real table of contents. This script is located in the `etc/html-common/scripts` directory of Canthology. If you copy the script into a directory containing HTML files generated by `hacha`, then you can use it as follows (assuming `index.html` contains the pseudo table of contents):

```
tclsh remove_pseudo_toc.tcl -contentsname "Contents" \
```

Figure 10.3: Pseudo table of contents produced by `hacha`




```
                    index.html -link "Table of Contents"
```

The `-contentsname` command-line option instructs the script to look in the pseudo table of contents in the specified file for a link called `Contents`. The script will remove the pseudo table of contents and replace it with a link to the real table of contents. The name of this new link is specified by the `-link` command-line option.

If you want to remove the pseudo table of contents and *not* provide a link to the real table of contents, then you run the script as follows:

```
tclsh remove_pseudo_toc.tcl -contentsname "Contents" \
                    index.html -nolink
```

10.10.3 Automating the Workarounds

As I will discuss in Chapter 11, when you use Canthology with HeVeA, Canthology adds `hevea-fix` to the list of packages used by your document, and also arranges for `remove_pseudo_toc.tcl` to be run after `hacha`. In these ways, Canthology automates use of the workarounds.

Chapter 11

Using HEVEA with Canthology

11.1 Introduction

To use HEVEA with Canthology, you need to modify your Canthology configuration file so your document's scope copies from one of the `html`-related scopes previously listed in Table 7.1 on page 45. You can see an example of this in line 3 of Figure 11.1.

Figure 11.1: Configuration file for generating HTML

```
 1  @include getenv("CANTHOLOGY_HOME") + "/etc/defaults.cfg";
 2  anthology1 {
 3      @copyFrom "book:html-many-pages";
 4      root_file {
 5          base_name = "my-anthology" + macro.macro.paperSizeSuffix;
 6          front_matter = [...];        # omitted for brevity
 7          main_matter  = [...];        # omitted for brevity
 8          back_matter  = [...];        # omitted for brevity
 9      }
10      substitutions.search_replace_pairs = [
11          ...                          # omitted for brevity
12      ] + substitutions.search_replace_pairs;
13  }
```

When you run `canthology` on this modified configuration file, the generated HTML file(s) are placed in the `output-html` subdirectory. Within

that directory, the index.html file contains the title page of your document.

The rest of this chapter explains how the Canthology-HEVEA integration works. That knowledge will explain how you can customise some aspects of the generated HTML pages.

11.2 The default.html Configuration Scope

Figure 11.2 on the next page shows a slightly abridged version of the default.html scope in the etc/defaults.cfg file of an installation of Canthology.

The @copyFrom statement (line 2) copies default settings that are shared with other configuration scopes.

On line 3, the string "output-html" is assigned to the working_dir variable.

The macro.paperSizeSuffix variable is assigned the value "" (line 5) because HTML is independent of paper size.

Both hevea and hevea-fix are included in the list of packages used by the document (line 10). The hevea-fix package was discussed in Section 10.10.1 on page 121.

Line 14 adds ".hva" to the copy.file_extensions list. This enables Canthology to find the ".hva" implementations of package when it encounters \usepackage commands.

The extra_files_to_copy variable (lines 15-23) instructs Canthology to copy specific files—used to help turn a document into HTML pages—into the working directory. The first file to be copied is Makefile (line 16), and this is used by the make build command (line 32). The other files to be copied are required for the Makefile to work properly. In particular:

- The insert_html_header_and_footer.tcl script (line 22) inserts html-header.txt and html-footer.txt (lines 18–19) into the start and end of each generated HTML page. You can use this to put, for example, a company logo and contact details across the top and bottom of each HTML page.

Figure 11.2: Outline of the `book:html-many-pages` configuration scope

```
 1  default.html {
 2      @copyFrom "default.common";
 3      working_dir = "output-html";
 4      macro {
 5          paperSizeSuffix = "";
 6          ...
 7      }
 8      root_file {
 9          documentclass.name = "book";
10          package.names = [..., "hevea", "hevea-fix", ...];
11          ... # omitted for brevity
12      }
13      copy {
14          file_extensions = [".hva"] + file_extensions;
15          extra_files_to_copy = [
16              "Makefile",
17              "canthology.css",
18              "html-header.txt",
19              "html-footer.txt",
20              "tidy.conf",
21              "scripts/remove_pseudo_toc.tcl",
22              "scripts/insert_html_header_and_footer.tcl",
23          ];
24          ... # omitted for brevity
25      }
26      build_commands = ["make"];
27      substitutions {
28          search_replace_pairs = [
29            "(MAKEFILE-IMAGEN-OPTIONS-PLACEHOLDER)",
30                                              "-pdf -png -mag 2000",
31            "(MAKEFILE-INSTALL-DIR-PLACEHOLDER)", "/var/www",
32          ];
33      }
34  }
```

- The `remove_pseudo_toc.tcl` script (line 21) was discussed in Section 10.10.2 on page 123. The `Makefile` runs this script to remove the pseudo table of contents from the title page of your document.
- The `canthology.css` file (line 17) is appended to the CSS file for the generated HTML pages.

- The final step in the Makefile is to run the tidy utility to "tidy up" the HTML files and ensure they adhere to the XHTML standard. The tidy.conf file (line 20) specifies configuration options for tidy.

The substitutions.search_replace_pairs variable (lines 28–32) is used to provide values for two variables used in the Makefile.

11.3 The book:html-many-pages Configuration Scope

One variable missing from the default.html scope is copy.search_path, which specifies the directories in which Canthology should look for the files listed in copy.extra_files_to_copy. This is because the value of copy.search_path depends on whether the HTML document will consist of one or many pages.

Figure 11.3 shows that the book:html-many-pages scope copies from the default.html scope (line 2) and then sets copy.search_path appropriately (lines 4–9).

Figure 11.3: Outline of the book:html-many-pages configuration scope

```
 1  book:html-many-pages {
 2      @copyFrom "default.html";
 3      copy {
 4          search_path = [
 5              ".",
 6              getenv("CANTHOLOGY_HOME") + "/etc/html-many-pages",
 7              getenv("CANTHOLOGY_HOME") + "/etc/html-common",
 8              getenv("CANTHOLOGY_HOME") + "/etc/latex",
 9          ];
10      }
11      substitutions {
12          search_replace_pairs = search_replace_pairs + [
13              "(MAKEFILE-TOC-OPTIONS-PLACEHOLDER)",
14                    "-link %"Table of contents%" -contentsname %"Contents%"",
15          ];
16      }
17  }
```

The book:html-one-page scope is similar, but sets copy.search_path

to contain `etc/html-one-page` instead of `etc/html-many-pages` (line 6).

11.4 The `Makefile`

The `Makefile` used to convert the input document into multiple HTML pages is shown in Figure 11.4.

Readers not familiar with Makefiles should read Section 11.4.1 to get an overview of the basic concepts and syntax used in Makefiles. Readers who *are* already familiar with Makefiles can skip ahead to Section 11.4.2 on page 132.

11.4.1 Overview of `Makefile` Concepts and Syntax

A `Makefile` typically contains instructions for turning one or more source-code files into an executable application. In the case of Canthology, a `Makefile` contains instructions for converting `".tex"` files into HTML.

An application called `make` reads a `Makefile` and executes the instructions contained in it.

Within a `Makefile`, a line of the form *`name=value`* defines a variable called *`name`*. The *`value`* of the variable can later be accessed with the syntax `$(`*`name`*`)`. For example, line 1 in Figure 11.4 on the next page defines a variable called `DOC`, and the value of this variable is used in line 9 (and several other lines).

A line starting with *`name:`* defines a *target* called *`name`*. Figure 11.4 contains three targets: `html`, `install` and `clean`. The indented lines immediately following the name of a target are shell commands that need to be executed to "make" that target. The list of shell commands is terminated by a blank line, the name of the next target or the end of the file. Thus, lines 9–26 are the commands for making the `html` target, lines 29–33 are the commands for making the `install` target, and lines 36–40 are the commands for making the `clean` target.

If the name of a target is specified when running `make`, then `make` will execute the commands associated with that target. For example, running `"make html"` executes the commands associated with the `html` tar-

Figure 11.4: The `Makefile` used by the `html-many-pages` scopes

```
DOC=(ROOT_FILE_BASE_NAME)
IMAGEN_OPTIONS=(MAKEFILE-IMAGEN-OPTIONS-PLACEHOLDER)
INSTALL_DIR=(MAKEFILE-INSTALL-DIR-PLACEHOLDER)
TOC_OPTIONS = (MAKEFILE-TOC-OPTIONS-PLACEHOLDER)
HTML_FILES  = `find . -name "*.html"`
GRAPHIC_FILES  = `find . -name "*.jpg" -o -name "*.gif" -o -name "*.png"`

html:  clean
    hevea $(DOC).tex
    if [ -f $(DOC).image.tex ]; then \
        imagen $(IMAGEN_OPTIONS) $(DOC); \
    fi
    if [ `grep -c \\bibliography $(DOC).tex` -ne 0 ]; then \
        bibhva $(DOC); \
        hevea $(DOC).tex; \
    fi
    hevea $(DOC).tex
    hacha -o index.html $(DOC).html
    rm $(DOC).html
    tclsh scripts/remove_pseudo_toc.tcl index.html
    tclsh scripts/insert_html_header_and_footer.tcl \
                                -header html-header.txt \
                                -footer html-footer.txt \
                                $(HTML_FILES)
    cat canthology.css >> $(DOC).css
    -tidy -config tidy.conf -m $(HTML_FILES)

install:
    mkdir -p  $(INSTALL_DIR)
    -chmod 775 $(INSTALL_DIR)
    cp $(HTML_FILES) $(INSTALL_DIR)
    cp $(DOC).css $(INSTALL_DIR)
    -cp -f $(GRAPHIC_FILES) $(INSTALL_DIR)

clean:
    rm -f *.aux *.log *.toc *.out *.dvi $(DOC).pdf *.blg
    rm -f *.haux *.htoc *.hbbl $(HTML_FILES) $(DOC).css
    rm -f $(DOC)[0-9][0-9][0-9].gif $(DOC)[0-9][0-9][0-9].png
    rm -f contents_motif.gif next_motif.gif previous_motif.gif
    rm -f $(DOC).image.tex
```

get, while running `"make clean"` executes the commands associated with the `clean` target. The first target that appears in a `Makefile` is the default target. Thus, in Figure 11.4, the default target is `html`, so running `"make"` is equivalent to running `"make html"`.

If a command is too long to fit on one line, then `\` can be used as a line continuation character. For example, `\` is used to merge lines 10–12 into one long command.

11.4.2 Variables Used in the `Makefile`

The `Makefile` in Figure 11.4 on the previous page defines four variables (lines 1–4) whose values are placeholder strings that will be replaced by real values when Canthology is run. The `(ROOT_FILE_BASE_NAME)` placeholder will be replaced by the value of `root_file.base_name` in the Canthology configuration file. Values for the other placeholder strings are specified in the `substitutions.search_replace_pairs` configuration variable, as can be seen in lines 29–31 of Figure 11.2 on page 128 and in line 13 of Figure 11.3 on page 129.

11.4.3 The `html` Target

The definition of the `html` target (line 8–26 in Figure 11.4 on the previous page) is long, but straightforward. It first runs `hevea` on the root `".tex"` file (line 9). Then, it checks for a particular file whose existence indicates that `imagen` needs to be run (lines 10–12). Next (lines 13–16), it checks if a `\bibliography` command appears in any of the `".tex"` files; if so, it runs `bibhva` to convert the bibliography's contents into LaTeX format, and then runs `hevea` to process the newly generated LaTeX. Afterwards, `hevea` is run again (line 17) to resolve cross references.

At this point, the LaTeX document has been converted into a monolithic HTML page. To split that into multiple HTML pages, the `hacha` utility is run (line 18) and the no-longer-needed monolithic HTML file is deleted (line 19). Then, some Tcl scripts are run to remove the pseudo table of contents (line 20) and insert headers and footers into each HTML page (lines 21–24). The `canthology.css` file is appended to the `".css"` file

created by `hevea` and `hacha` (line 25). Finally, `tidy` is run to convert the HTML pages into XHTML format (line 26).

11.4.4 The `install` Target

The `install` target copies HTML files plus supporting CSS and image files into the directory specified by the `INSTALL_DIR` variable. Line 31 of Figure 11.2 on page 128 sets the default value of this variable to be `"/var/www"`, which is the root directory for web servers on many UNIX machines.

11.4.5 The `clean` Target

The `clean` target uses the UNIX `rm` command to remove generated files.

11.5 Customising the HTML Pages

If you have a working knowledge of the syntax used in HTML and CSS, then it is possible to customise the "look and feel" of the HTML pages generated by Canthology. You can do this by providing your own versions of the `html-header.txt`, `html-footer.txt` and `canthology.css` files, which, as I explained in Section 11.3 on page 129, are used by the `Makefile`.

The default versions of `html-header.txt` and `html-footer.txt` used by the `book:html-many-pages` scope is shown in Figures 11.5 and 11.6.

The `html-header.txt` file uses `div` elements to divide the HTML page into a "banner" area follows by a "content" area (for the main content of the page). The "banner" area is further subdivided into "bannerleft" and 'bannerright" areas.

The `html-footer.txt` file closes the "content" area and adds a `br` element (a line break) to provide a bottom margin, thus ensuring the end of the page's content is not uncomfortably close to the bottom of the window in which it appears.

Figure 11.5: The `html-header.txt` file

```
<div class="banner">
    <div id="bannerleft">
        Change the look-and-feel with your own versions of
        "html-header.txt", "html-footer.txt" and
        "canthology.css".
    </div>
    <div id="bannerright">
        <a href="index.html">title page</a>  
        <a href="contents.html">contents</a>
    </div>
    <div class="endtwocolumntext"></div>
</div>
<div class="content">
```

Figure 11.6: The `html-footer.txt` file

```
</div><br/>
```

The visual formatting of the `div` elements mentioned above is defined in the `canthology.css` file. A discussion of that file is outside the scope of this chapter, but I encourage readers to examine the file to see how the visual formatting rules are specified.

The effect of `html-header.txt` and `html-footer.txt` is illustrated in Figures 11.7 and 11.8, which show the start and end of a HTML page.

You might wish to replace the text in the "bannerleft" area with, say, the logo for your website. Likewise, you might modify `html-footer.txt` to add some text (perhaps contact details for your organisation) to the bottom of HTML pages. If you do not wish to have any header or footer contents, then you can delete all the contents of `html-header.txt` and `html-footer.txt`, thus making them empty files.

Among other things, the `canthology.css` file sets a white background for the HTML page and highlights a hypertext link when the mouse pointer hovers over it. You can create your own version of `canthology.css` to implement a different "look and feel" for the generated HTML pages. This enables you to ensure that the Canthology-generated HTML pages can blend into the overall visual style of a parent website.

Figure 11.7: The start of a HTML page

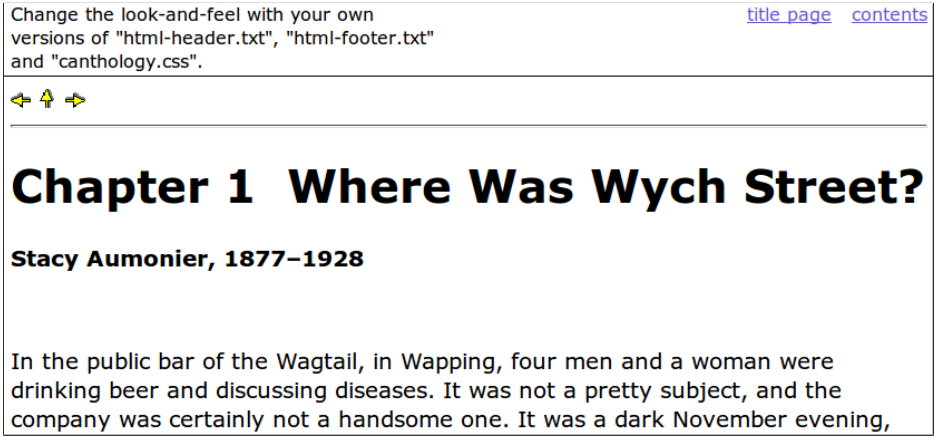
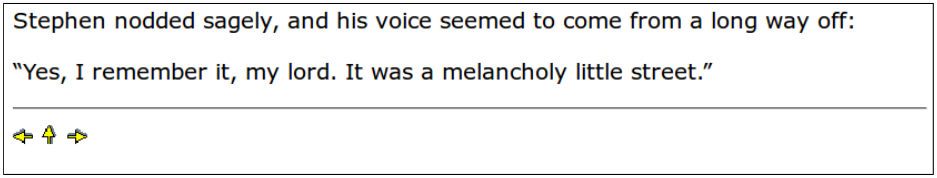


Figure 11.8: The end of the HTML page



11.6 Other HTML Configuration Scopes

The discussion so far has assumed the use of the `book:html-many-pages` configuration scope. The `report:html-many-pages` and `article:html-many-pages` scopes are almost identical—they just use a different value for the `root_file.documentclass.name` configuration variable.

There is another set of configuration scopes called `book:html-one-page`, `report:html-one-page` and `article:html-one-page`. They use a slightly different setting for the `copy.search_path` variable, so they look for support files in the `etc/html-one-page` directory instead of in `etc/html-many-pages`. Because of this, they pick up different versions of `Makefile`, `html-header.txt` and `html-footer.txt`.

The `etc/html-one-page` version of `Makefile` runs `hevea` to convert a

LaTeX document into a single-page HTML file; it does *not* run `hacha` to split the file into multiple HTML pages.

The `etc/html-one-page` version of `html-header.txt` does not contain "title page" or "contents" links.

Chapter 12

Writing Documents Portable to PDF and HTML

12.1 Introduction

The concept of *portability* is often applied to software applications. For example, if an application can run on only one type of computer operating system, then the application is said *not* to be portable. Conversely, if an application can run on several types of operating system (such as Microsoft Windows, Apple Macintosh and Linux), then the application is said to be portable to those operating systems.

The concept of portability can also be applied to documents. It is possible to write a `".tex"` file that can be processed by LaTeX to produce a PDF file, but *cannot* be processed by HeVeA to produce HTML. And vice versa. But, with a bit of planning and effort, it is often possible to write a `".tex"` file that can be processed by LaTeX to produce a PDF file, and also can be processed by HeVeA to produce HTML. I say that such a document is *portable* to both PDF and HTML.

Previous chapters have discussed several ways in which Canthology simplifies the task of writing LaTeX documents that are portable to PDF and HTML. For example:

- Template files shield users from differences in how to format title

pages, dedication pages, and so on, so they look good in both PDF and HTML formats.

- Canthology provides `".sty"` and `".hva"` implementations of several packages.
- Configuration scopes in `etc/defaults.cfg` shield users from significant differences in the shell commands used to convert `".tex"` files into PDF or HTML files.

In this chapter, I start by discussing some additional LaTeX commands, defined in the `canthology` package, that help to increase the portability of documents.

Although Canthology protects users from several portability obstacles, it is impossible for Canthology to provide protection from *every* portability obstacle. For this reason, I finish this chapter by offering some advice on what you can do when you encounter a portability obstacle that is *not* by Canthology.

12.2 How to Define Labels in a Portable Way

Books about LaTeX typically show the `\label` command (Section 9.10.1 on page 99) being used immediately after one of the following commands: `\part`, `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\paragraph`, `\subparagraph` or `\caption`. When you convert your `".tex"` document into a PDF file, the label is associated with the *preceding* command.

Unfortunately, HeVeA works slightly differently. When it converts your `".tex"` document into HTML, the label is associated with whatever text *follows* it. To understand this distinction, consider the following:

```
\chapter{How it All Started}
\label{ch:started}
Once upon a time ...
```

Let's assume another part of the document provides a cross reference to the `ch:started` label, and that a reader viewing the document on a com-

puter screen clicks on that cross reference. If the document is in PDF format, then the PDF document viewer will jump to the chapter title, which is what we want. However, if the document is in HTML format, then the web browser will jump to the first line of text *after* the chapter title (that is, *Once upon a time...*) and the chapter's title will be invisible, just above the top of the browser's window, which is *not* what we want. This problem affects not just the `\chapter` command, but all commands after which you might place a `\label` command.

The HeVeA manual [6] recommends a workaround for this problem: place the `\label` command *inside* the parameter to the `\chapter` command. Using this workaround, the example would be rewritten as follows:

```
\chapter{\label{ch:started}How it All Started}
Once upon a time ...
```

That workaround results in cross references in the HTML version of a document working as they do in the PDF version of a document. However, this workaround is not without problems.

One obvious problem is that placing the `\label` command inside the `\chapter` command decreases readability of the `".tex"` file.

Another *potential* drawback is that I don't think LaTeX was designed with the expectation that a `\label` command would be placed inside a `\chapter` command, so this works by accident rather than by design. As such, it is possible that future modifications to LaTeX may cause this workaround to stop working.

Ideally, we want the `\label` command to appear *after* the `\chapter` command when using LaTeX, but to appear *inside* the `\chapter` command when using HeVeA. The `canthology` package facilitates this by defining a command called `\lchapter` (short for "labelled chapter") that takes two parameters: a chapter title and its label. The `".sty"` (that is, LaTeX) implementation of the `canthology` package defines this command as follows:

```
\newcommand{\lchapter}[2]{\chapter{#1}\label{#2}}
```

As you can see, that definition places the `\label` command *after* the `\chapter` command. In contrast, the `".hva"` (that is, HeVeA) implemen-

tation of the canthology package defines the command so that the \label command is *inside* the \chapter command:

```
\newcommand{\lchapter}[2]{\chapter{\label{#2}#1}}
```

We can use \lchapter to rewrite the start-of-a-chapter example in a more portable way:

```
\lchapter{How it All Started}{ch:started}
Once upon a time...
```

The canthology package defines \lpart, \lsection, \lsubsection, \lsubsubsection, \lparagraph, \lsubparagraph and \lcaption in a similar way to \lchapter.

12.3 Avoid Using the \pageref Command

Consider the following sentence:

```
I discuss this in greater detail in
Section~\ref{sect:something} on
page~\pageref{sect:something}.
```

If you convert your document into PDF, then the above sentence may appear as:

I discuss this in greater detail in Section 2.6 on page 19.

However, if you convert your document into HTML, then two question marks will be used for the page number:

I discuss this in greater detail in Section 2.6 on page ??.

This happens because the concept of page numbers does not make sense in HTML. Thus, HeVeA implements the \pageref command to print two question marks in the hope that a proofreader will notice the problem in the generated HTML.

Thankfully, there is a simple technique that enables us to use \pageref when creating PDF documents and to avoid using it when creating HTML

documents. This technique is to use `\newcommand` to define higher-level commands for typesetting cross references. The definition of these higher-level commands in the `".sty"` implementation of a package *can* use the `\pageref` command, while the definition of the commands in the `".hva"` implementation of a package *avoids* using the `\pageref` command.

In Section 9.10.2 on page 102, I discussed shorthand commands such as `\xsp` and `\xs` for typesetting a cross reference to a section, with and without stating its page number. The definition of these commands in the `".sty"` implementation of a package might be as follows:

```
\newcommand{\xs}[1]{Section~\ref{#1}}
\newcommand{\xsp}[1]{Section~\ref{#1} on
page~\pageref{#1}}
```

Their definitions in the `".hva"` implementation of the same package might be:

```
\newcommand{\xs}[1]{Section~\ref{#1}}
\newcommand{\xsp}[1]{Section~\ref{#1}}
```

Using those definitions, the example sentence used earlier can be written (more compactly) as follows:

```
I discuss this in greater detail in \xsp{sect:something}.
```

That will be typeset in a PDF document as follows:

> I discuss this in greater detail in Section 2.6 on page 19.

and will be typeset in a HTML document as follows:

> I discuss this in greater detail in Section 2.6.

The `".sty"` implementation of the `canthology` package defines the `\xpp`, `\xap`, `\xcp`, `\xsp`, `\xfp` and `\xtp` commands to make use of page numbers (via the `\vref` command of the `varioref` package). The `".hva"` implementation of the `canthology` package defines those commands to *not* attempt to use page numbers.

12.4 The `\ifthenelse` Command

Among other things, the `ifthen` package [2] defines an `\ifthenelse` command that takes three parameters:

```
\ifthenelse{condition}{then-part}{else-part}
```

The `\ifthenelse` command evaluates the *`condition`* parameter. If this evaluates to *true*, then the text or statements in the *`then-part`* parameter are processed. Otherwise, the text or statements in the *`else-part`* parameter are processed.

The `hevea` package [6] defines a boolean variable called `hevea`. This variable is given the value *true* when a document is processed by the `hevea` command, but is given the value *false* when a document is processed by a LaTeX-related command such as `latex` or `pdflatex`.

The combination of the `\ifthenelse` command and the `hevea` boolean variable makes it possible to tailor the appearance or contents of a document for the output format (HTML or PDF). For example, an outline of the `"titlepage-template-*.tex"` files (Section 7.4 on page 48) is shown below:

```
\ifthenelse{\boolean{hevea}}{
    ... % Typeset the title page for HTML format
}{
    ... % Typeset the title page for PDF format
}
```

In this way, a book can have its title page typeset one way in the PDF format of the book, and typeset another way in the HTML format.

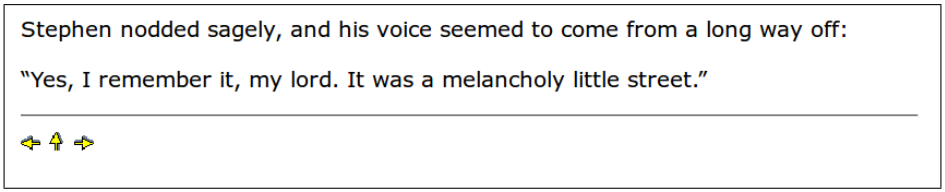
You are likely to need to use the `\ifthenelse` command only rarely. However, for some niche tasks, such as typesetting the title page of a document, it can be invaluable.

12.5 Placement of Captions

A *caption* is a brief description that accompanies, for example, a photograph in a book or article. In some books, a caption appears *above* a figure.

You can see an example of this in Figure 12.1.

Figure 12.1: A caption *above* a figure



In some other books, a caption appears *below* a figure. You can see an example of this is shown in Figure 12.2.

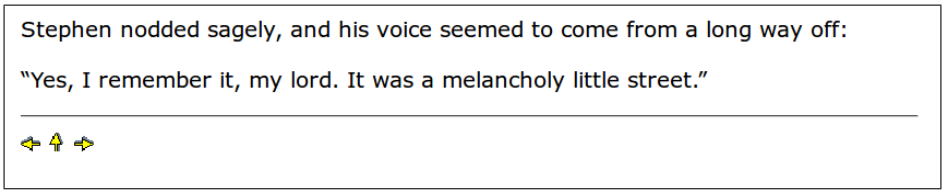


Figure 12.2: A caption *below* a figure

When a document is printed on paper, it is a subjective issue whether a caption looks better above or below a figure. However, if a document might be read on a computer screen, then it is better to put the caption *above* a figure. To understand why, let's assume you are reading a document on a computer screen, and you click on a cross reference to a figure. The view on the screen may change to show the figure's caption (that is, the linked-to text) at the very top of the screen.

- If the caption is *above* the figure, then you will be able to see the figure on the screen.
- If the caption is *below* the figure, then (because the caption is at the top of the screen) the figure will be *above the top of the screen* and hence will be invisible. In such a case, you will have to scroll the document to be able to see the future.

Readers who have viewed HTML pages in a web browser should easily be able to visualise the issue I have described above. However, some readers may assume the issue does *not* apply to PDF documents because the PDF viewer they use shows an entire page of a document at a time. However, many PDF viewers have a menu option that enables you to view a document in one continuous (and scrollable) stream rather than a single page at a time. If a PDF viewer is in "continuous" viewing mode, then the issue I have described above will apply.

12.6 Dealing with Other Portability Problems

Sooner or later, you are likely to encounter a portability obstacle that is *not* handled by Canthology. Typically, you will notice that a particular command is processed one way by LaTeX, but processed a *different* way by HeVeA. Your instinct might be to wonder:

> How can I make this command work the same way in HeVeA as it does in LaTeX?

However, it is often more productive to ask a different question:

> That command (perhaps when combined with other commands) produces the desired result with LaTeX but, unfortunately, not with HeVeA. Might a different command (or sequence of commands) produce the desired result (or at least an acceptable result) with HeVeA?

If you can find such an alternative command (or sequence of commands), then that makes it possible to overcome the portability obstacle with the following:

1. Create a new package, let's call it `example`. Doing this is straightforward. Section 9.5 on page 85 discusses how to create `example.sty` for use with LaTeX; and Section 10.6 on page 113 discusses how to create a corresponding `".hva"` version of a package for use with HeVeA.

2. In `example.sty`, use `\newcommand` to define a command that achieves what you want in a way that is compatible with LaTeX.

3. In `example.hva`, use `\newcommand` to define an identically-named command that achieves what you want in a way that is compatible with HeVeA.

4. Now add the `example` package to the list of package names used by your document.

As an example, recall from Section 12.2 on page 138 that placing a `\label` command *after* a `\chapter` command does what we want in LaTeX, but not in HeVeA. The way Canthology works around this portability problem is to implement a new command called `\lchapter` one way in `canthology.sty` and a different way in `canthology.hva`.

In fact, this approach is the primary tactic that Canthology uses to encapsulate portability obstacles. The `canthology` package defines over 30 commands, most of which are defined one way in `canthology.sty` and a different way in `canthology.hva`.

Part V

Information for Maintainers

Introduction to Part V

The two chapters in Part V provide information that may be of use to people who want to do maintenance work on Canthology, whether by fixing bugs in existing functionality, or by adding new functionality.

Chapter 13 provides an overview of Canthology's architecture. Canthology is a small application, and this is reflected in the conciseness of this chapter.

There are many ways in which Canthology might be improved if people have the time or skills to do so. Chapter 14 outlines some of those ways.

Chapter 13

Architecture of Canthology

13.1 Introduction

This chapter is aimed at people who want to modify Canthology, for example, to fix a bug or extend its functionality. To be able to modify Canthology, you will first need to be familiar with its architecture. Explaining that architecture is the focus of this chapter.

Canthology is comprised of three interacting parts: (1) a Java-based application; (2) supporting files; and (3) the `defaults.cfg` file, which (among other things) configures Canthology to use the subset of supporting files appropriate for the chosen output format (PDF or HTML).

I have already discussed the `defaults.cfg` file in Chapters 8 and 11, so this chapter focusses on the Java-based application and supporting files.

13.2 The Java Application

The `anthology` application contains approximately 1200 lines of Java code (excluding comments and blank lines). Some readers may consider this to be surprisingly concise, considering the functionality provided by the application. This conciseness is due to a combination of two issues.

First, some of Canthology's functionality is not implemented in Java code, but rather is provided by support files, as I will discuss in Sec-

tion 13.3.

Second, and more significantly, the parsing and semantic checking of the configuration file is performed by a separate library (which contains over 8000 lines of code). Thus, Canthology gains the rich functionality of that configuration-file parser "for free".

The configuration parser library is called *Config4J*; this is the Java implementation of *Config4** (pronounced "config for star"). If you want to modify Canthology, then it is useful to first gain an overview of the API of Config4J. You can find such an overview in the Chapter 3 of the *Config4* Getting Started Guide*, which is available on the Config4* website.[1]

13.2.1 Packages and Source-code Files

The Java source code of the `canthology` application resides in a package called `org.canthology.canthology`. The repetition of "canthology" in the package name might appear redundant to some readers. However, it is there to make it easy to create future utility applications that will complement Canthology. For example, if a future version of Canthology provides utility applications called `foo` and `bar`, then the package hierarchy might be as follows:

`org.canthology.canthology`	(source code of `canthology`)
`org.canthology.foo`	(source code of `foo`)
`org.canthology.bar`	(source code of `bar`)

13.2.2 Limited Use of Java Language Features

During my career as a software consultant, I have worked with numerous companies and have noticed that although some companies are quick to upgrade to new versions of development tools, other companies are much slower to do so. It is not unusual for a company to be using a compiler that is five or even ten years old. For this reason, when I develop open-source software, I like to avoid use of relatively new features in a programming language. This approach has the drawback that the source code of my

[1] www.config4star.org

applications may be slightly more verbose than necessary, but it offers the benefit that my applications can be compiled and used by a wide range of companies and individuals, regardless of whether they use new or older compilers.

In practice, I try to limit myself to features available in Java 1.3. One consequence of this is that I avoid using the Java `assert` statement (it was introduced in Java 1.4). Instead, I have written an `assertion()` method that serves a similar purpose.

Another consequence of restricting myself to language features available in Java 1.3 is that I avoid use of generics (they were introduced in Java 5). This results in verbosity when retrieving items from a collection, due to the need for typecasts.

13.2.3 Algorithms and Source-code Files

The source code of the `canthology` application consists of the following five files:

```
Util.java
AssertionError.java
Canthology.java
StartingPointConfig.java     (generated from StartingPointConfig.cfg)
DocumentInfo.java
```

I will now discuss each of those briefly.

`Util.java` and `AssertionError.java`

The `Util` class defines some `static` utility methods. One of these is `Util.assertion()`, which, as I already discussed, I use instead of the Java `assert` statement. This method throws an `AssertionError` if the assertion check fails.

`Canthology.java`

The `Canthology` class defines the `main()` method for the `canthology` application. The code in this class is straightforward. First, it parses

command-line arguments. If a `-create` command-line option is encountered, then it generates a starting-point configuration file and terminates. Otherwise, it creates an empty `Configuration` object (this type is defined by the Config4J library), populates it with *name*=*value* pairs obtained from "`-set` *`name value`*" command-line options, and then parses the configuration file. Finally, for each configuration scope that defines a document, it creates a `DocumentInfo` object (whose constructor validates the configuration information within a configuration scope) and calls `DocumentInfo.process()` to perform the "real work" of Canthology.

`StartingPointConfig.cfg`

A simple, but tedious, way to create a starting-point configuration file is to open the file for writing, use lots of print statements to generate the contents of the file, and then close the file. The problem with this technique is that writing the print statements is tedious and error-prone. Config4J alleviates this tedious work as follows. The `config2j` utility (documented in Chapter 6 of the *Config4* Getting Started Guide*) can read a text file and generate a Java class that provides the contents of the file as an embedded string (accessible through a `getString()` method). The build system uses `config2j` to convert `StartingPointConfig.cfg` into `StartingPointConfig.java`, and that generated file is compiled into the application. Thus, the application code to create a starting-point configuration file becomes trivial:

```
out = new FileWriter(cfgFileName);
out.write(StartingPointConfig.getString());
out.close();
```

If you want `canthology` to generate a starting-point configuration file with different contents, then you should modify `StartingPointConfig.cfg`, and run `ant` to rebuild the application.

`DocumentInfo.java`

The `DocumentInfo` class performs the "real work" of Canthology. Its public API consists of a constructor and the `process()` method.

The constructor creates a schema that specifies what contents are allowed in a document scope, and uses the `SchemaValidator` class (provided by the Config4J library) to validate the document scope against this schema. Finally, the constructor copies configuration variables into instance variables for more convenient access when the `process()` method is invoked.

The `process()` method uses configuration information to perform the following steps:

- Create a root `".tex"` file in the working directory. Search this file for commands such as `\input` that specify the names of other required files, and copy those files into the working directory. Those copied files are recursively searched for commands such as `\input`.
- Files listed in the `copy.extra_files_to_copy` configuration variable are copied into the working directory. These copied files are recursively searched for commands such as `\input` to find other files that need to be copied.
- When the root `".tex"` file is being created, and when other files are being copied, the `substitutions.search_replace_pairs` configuration variable is used to perform a global search-and-replace on the files' contents.
- Finally, each command listed in the `build_commands` configuration variable is executed.

The above steps are straightforward. The only complication is that some steps in the algorithm are inherently recursive.

13.3 Support Files

Support files used by Canthology are stored in the following subdirectories of a Canthology installation:

```
etc/latex
etc/html-common
```

```
etc/html-many-pages
etc/html-one-page
```

I now discuss each of these briefly.

13.3.1 The `etc/latex` Subdirectory

The `etc/latex` directory contains some `".sty"` files (that is, packages) plus some `".tex"` files. The `".sty"` files are:

```
canthology.sty
hevea.sty
hevea-fix.sty
```

The `canthology` package defines commands such as `\chapterAuthorInfo` for specifying information about the author of a contribution in an anthology, and `\thisPageBackgroundColor` for specifying a background colour to be used on, say, the title page of a document. Appendix A provides a complete list of the commands defined in the `canthology` package.

The `hevea` package defines dummy versions of HEVEA commands, so a `".tex"` document that makes use of those commands can be processed by either LATEX or HEVEA. The `hevea` package is distributed as part of the HEVEA distribution. However, a copy of `hevea.sty` is also distributed with Canthology. Doing this enables Canthology to provide an out-of-the-box integration with HEVEA without having to worry about whether HEVEA is installed.

As I discussed in Section 10.10.1 on page 121, I wrote `hevea-fix.hva` to overcome some misfeatures of HEVEA. The `hevea-fix.sty` file is a dummy version of the package for compatibility with LATEX.

The `".tex"` files in the directory are as follows:

```
titlepage-template-1.tex
titlepage-template-2.tex
titlepage-template-3.tex
titlepage-hevea-template.tex
example-copyright-template.tex
copyright-GNU-FDL-1.3.tex
```

```
copyright-GPL-2.0.tex
copyright-GPL-3.0.tex
copyright-LPPL-1-3c.tex
dedication-template.tex
example-praise-template.tex
```

The files with names of the form `"titlepage-template-*.tex"` provide various layouts for the title page of a book. These are called *template* files because they contain placeholder text that can be replaced with text specified in the `substitutions.search_replace_pairs` configuration variable.

You can see an example of a template file in Figure 13.1 on the next page, which shows the contents of the `titlepage-template-1.tex` file. For ease of reference, placeholder text is shown in a **bold** font.

The title page of a document is often typeset with commands that specify rules and vertical spaces. Such commands work fine for a page of fixed dimensions, but often do not make sense for a HTML document. For this reason, it is useful to typeset the title page *one way* if the document is being processed with LaTeX, but typeset the title page *another way* if the document is being processed with HeVeA. This is the reason for the if-then-else statement on line 1 of Figure 13.1. If HeVeA is being used, then the simpler, HTML-friendly formatting commands in `titlepage-hevea-template.tex` are used.

The `example-copyright-template.tex` file provides example text for several copyright licenses. The intention is that the editor of an anthology will make a local copy of this file and edit its contents to suit his or her needs.

One of the example paragraphs in `example-copyright-template.tex` is for version 1.3 of the GNU Free Documentation License (FDL). If the editor of an anthology wishes to use this license, then it is a legal requirement to provide the full text of the FDL as, say, an appendix in the document. The `copyright-GNU-FDL-1.3.tex` file provides the full text of that license and can be used for that purpose. For a similar reason, the `copyright-GPL-2.0.tex` and `copyright-GPL-3.0.tex` files provide the full text of versions 2 and 3 of the Gnu General Public License (GPL),

Figure 13.1: The `titlepage-template-1.tex` file

```
\ifthenelse{\boolean{hevea}}{
    %--------
    % Hevea version
    %--------
    \input{titlepage-hevea-template.tex}
}{
    %--------
    % LaTeX version
    %--------
    \thispagestyle{empty}
    \begin{center}
        \vspace*{0.2\textheight}

        \Huge \textbf{(TITLE-PLACEHOLDER)}

        \ifthenelse{\equal{}{(SUBTITLE-PLACEHOLDER)}}{}{
            \vspace{1cm}
            \Large \textbf{(SUBTITLE-PLACEHOLDER)}
        }

        \vspace{1cm}
        \Large \textbf{(AUTHOR-PLACEHOLDER)}
    \end{center}

    \vfill

    \noindent
    {\large (DESCRIPTION-PLACEHOLDER)}

    \ifthenelse{\equal{}{(PUBLISHER-PLACEHOLDER)}}{}{
        \noindent\rule{\textwidth}{0.2mm}\\
        \noindent{\rule{0mm}{1.1em}(PUBLISHER-PLACEHOLDER)}
    }
    \newpage
}
```

and `copyright-LPPL-1-3c.tex` provides the full text of the LaTeX Project Public License.

The `dedication-template.tex` file can be used to typeset a dedications page near the start of a book.

The `example-praise-template.tex` file provides examples of how to use the `\praise` command to typeset a page of praise for a book.

13.3.2 The `etc/html-*` Subdirectories

Support files used with HEVEA are spread across the following subdirectories:

```
etc/html-common
etc/html-many-pages
etc/html-one-page
```

The `html-common` directory contains the `".hva"` implementations of several packages, including: `canthology`, `framed`, `hevea-fix` and `verse`. This directory also contains: `canthology.css`, which is used for customising the look-and-feel of generated web pages; `tidy.conf`, which is used to configure the `tidy` program when converting generated HTML files into XHTML format; and some Tcl scripts that were discussed in Section 11.3 on page 129.

The `html-one-page` directory contains a `Makefile` that runs `hevea` to create a single-page HTML document. This directory also contains supporting `html-header.txt` and `html-footer.txt` files suitable for use in a single-page HTML document.

The `html-many-pages` directory contains a `Makefile` that runs `hevea` to create a single-page HTML document, and then runs `hacha` to split it into multiple HTML pages. This directory also contains supporting `html-header.txt` and `html-footer.txt` files suitable for use in a multi-page HTML document.

13.4 How Canthology Searches for Support Files

The `copy.search_path` configuration variable specifies which directories will be searched to find supporting files.

Different scopes in the `etc/defaults.cfg` file define different values for the `copy.search_path` variable. For example, scopes used to generate PDF files set `copy.search_path` as follows:

```
copy.search_path = [
    ".",
    getenv("CANTHOLOGY_HOME") "/etc/latex",
];
```

In contrast, scopes used to generate a single-page HTML document set `copy.search_path` as follows:

```
copy.search_path = [
    ".",
    getenv("CANTHOLOGY_HOME") + "/etc/html-one-page",
    getenv("CANTHOLOGY_HOME") + "/etc/html-common",
    getenv("CANTHOLOGY_HOME") + "/etc/latex",
];
```

Scopes that generate a multi-page HTML document use the following setting for `copy.search_path`:

```
copy.search_path = [
    ".",
    getenv("CANTHOLOGY_HOME") + "/etc/html-many-pages",
    getenv("CANTHOLOGY_HOME") + "/etc/html-common",
    getenv("CANTHOLOGY_HOME") + "/etc/latex",
];
```

Chapter 14

Suggestions for Future Work

14.1 Introduction

In this chapter, I discuss several ways in which Canthology might be improved. If you have the time and skills to make any of these improvements, then please do so.

14.2 A Wider Selection of Title-page Templates

As discussed in Section 7.4 on page 48, Canthology provides three template files that make it easy to create the title page of a book. It would be nice to provide a more extensive range of template title pages with future versions of Canthology. If you would like to contribute to this, then you might find some inspiration from Peter Wilson's extensive collection of title pages [13].

Also, I do not claim to be especially gifted at designing title pages, even when borrowing ideas from Peter Wilson's document. If you feel the existing title pages can be tweaked to improve their layout, then please do so.

14.3 Background Graphics for Title Pages

I like having a white background for the title pages of books and manuals that I write. However, many people prefer a title page to have a colourful background picture. For this reason, it might be nice to ship a collection of high-resolution, royalty-free pictures with Canthology that people could use on the title page of documents.

Having said that, I would not like a distribution of Canthology to consist of, say, 1 MB of software, a few more MB of manuals and 400 MB of high-resolution pictures. Perhaps a better idea would be to have a small distribution of Canthology containing just a few sample pictures (so users can play with the `\thisPageBackgroundImage` command), and a separate website that users can browse to access a large collection of pictures. I know such websites already exist, but they are filled mostly with landscape-oriented images, while book covers require portrait-oriented images.

14.4 A Blog-to-LaTeX Converter

Consider the following scenario. Fred has a popular blog, and has been posting articles to it for on a regular basis for several years. He would like to create a "best of" collection of his blog articles, and publish it in book format. He decides Canthology would be a good tool to help him do this. Unfortunately, his blog articles are written using one markup language, while Canthology uses a different markup language (LaTeX), so he has to do the tedious work of converting blog postings from one markup language to another.

A useful addition to Canthology would be a utility that can convert documents from various blog markup formats into LaTeX format.

14.5 Improve the Quality of Generated HTML

Unfortunately, the HTML generated by HeVeA is not 100% compliant with standards. The Canthology `Makefile` that runs `hevea` and `hacha` tries to

improve the quality of the generated HTML by passing it through the `tidy` utility. However, there is still room for improvement. Thus, one way to improve Canthology is to improve HEVEA so it generates better-quality HTML.

14.6 Installers for Various Operating Systems

Installing Canthology is a simple, albeit multi-step, process, which can be described as, "unzip the distribution and set a few environment variables". It would be nice to have platform-specific installers that turn the multi-step process into a single-step process.

14.7 Generate ebooks

Canthology can generate books in PDF and HTML formats. Would it be possible for Canthology to *also* generate books in popular ebook formats? I spent a few weeks investigating this possibility and discovered that it presents some challenges.

14.7.1 Different ebook File Formats

There are approximately 20 different ebook file formats.[1] In some of the popular formats, an ebook is represented as an archive, such as a ZIP file, that holds the following: (1) XML or HTML files that contain the main text of the book; (2) a `".css"` file that defines the visual layout of the text; (3) image files for diagrams and pictures used within the book; and (4) metadata to specify information—such as the title, author and publisher of the book—and impose an organisational structure upon the collection of files in the archive.

I suspect the wide variety of ebook formats is due mainly to each manufacturer of ebook readers wanting to create a proprietary file format for its devices, in the hope of gaining a monopoly on the market for ebooks.

[1] http://en.wikipedia.org/wiki/Comparison_of_e-book_formats

However, as a back-up plan (in case a monopoly is not achieved), ebook readers tend to support a few open or competing file formats too. The result is that almost all ebook readers can be used to view books in the EPUB format. The most notable exception to this is the Amazon Kindle, which does *not* support EPUB. However, the Kindle supports the Mobipocket file format, which is supported on some other ebook readers too. Thus, if you want to make a book viewable on all ebook readers, a pragmatic approach is to produce EPUB and Mobipocket versions of your book, and ignore all the other ebook file formats.

14.7.2 Using HeVeA and Calibre

Calibre[2] is an open-source application for managing collections of ebooks. Calibre provides the ability to convert an HTML document into any of many ebook formats, including EPUB and Mobipocket. This raises the possibility of using HeVeA to convert a LaTeX document into HTML format, and then using Calibre to convert that into an ebook format.

I briefly experimented with this approach. Unfortunately, I was not very happy with the results it produced. Most of the text in my sample document was displayed appropriately in the ebook reader application provided by Calibre. However, the formatting of some text, such as poems, was messed up badly. Initially, I thought there might be a bug in Calibre, but when I started to investigate what might have gone wrong, I learned that both the EPUB and Mobipocket formats support only a subset of HTML and CSS. Perhaps the HTML generated by HeVeA did not fall into that supported subset, so Calibre had to modify the HTML when converting it into an ebook format, and these modifications resulted in the ebook version of my sample document displaying in a way I had not intended.

14.7.3 Tailoring HeVeA to better Support the Generation of ebooks

Perhaps it would be possible to write two sets of `".hva"` files for HeVeA. One set would instruct HeVeA to generate HTML that takes advantage

[2] http://calibre-ebook.com/

of *all* the capabilities of the HTML standard and displays nicely in web browsers. The other set would instruct HeVeA to generate only the subset of HTML elements and tags that are supported in EPUB and Mobipocket.

The two sets of `".hva"` files would reside in different directories, and the `"-I directory"` command-line option could instruct HeVeA to look for `".hva"` files in *one* of those directories. For example, if the two directories are called `browser-support-files` and `ebook-support-files`, then executing the command:

```
hevea -I browser-support-files my-document.tex
```

would produce HTML suitable for viewing in a web browser. In contrast, executing the command:

```
hevea -I ebook-support-files my-document.tex
```

would produce HTML suitable for converting into an ebook format via Calibre.

14.7.4 Playing with HeVeA and Calibre

If you want to experiment with using HeVeA and Calibre to convert LaTeX documents into ebooks, then the following information might help you to get started.

Before using Calibre, you should convert your LaTeX document into a single-page HTML document. You can do this by editing `Canthology.cfg` and ensuring the `@copyFrom` statement copies from the `book:html-one-page` scope. Afterwards, run `canthology`.

The web page below contains documentation on the `ebook-convert` command (provided as part of Calibre):

http://manual.calibre-ebook.com/cli/ebook-convert.html

Within that page, you should scroll down to the "HTML Input" list item, and then click on one of the sub-items, such as "HTML Input to EPUB Output". Doing that will lead you to documentation on the command-line options for performing the desired conversion.

If you want to convert a file called `my-anthology.html` into EPUB format, then the following example illustrates how you might do that (\ denotes a line continuation):

```
ebook-convert my-anthology.html my-anthology.epub \
                  --chapter "//*[name()='h1']" \
                  --page-breaks-before "//h:h1" \
                  --no-default-epub-cover \
                  --chapter-mark none \
                  --authors "Example Name" \
                  --publisher "Example Company"
```

You will probably want to play with different command-line options to see what effect they have.

14.7.5 Small Screen Sizes of ebook Readers

Support for a limited subset of HTML and CSS is not the only challenge when creating an ebook. Another challenge is the small displays of most ebook readers. For example, the diagonal screen size of the Amazon Kindle is is just 6 inches (15 cm). That is slightly smaller than a postcard.

The small screen size of an ebook reader is acceptable for reading a novel, because most novels contain minimal formatting. However, the small screen size is likely to cause problems for technical books, because they often have more ambitious formatting requirements. For example, Figure 11.4 on page 131 is a listing of a `Makefile`. The longest line in that listing is almost 70 characters across, which is too wide to display in its entirety on the screen of most ebook readers (unless you reduce the font size to something that is uncomfortably small to read). Likewise, some technical books contain tables of data that are too big for viewing on an ebook reader.

Perhaps a good rule of thumb is to consider creating ebook versions of novels, but to avoid trying to create ebook versions of product manuals and other technical documentation.

14.8 Providing Customisable Anthologies as Demos

One particular benefit of Canthology is its ability to easily customise an anthology. This benefit would be more readily apparent if Canthology was shipped with some book-length demos that are useful in their own right.

For example, Shakespeare's poems are old enough to be out of copyright, and they can be downloaded (legally) from numerous websites. Let's assume one person took the time to download all those poems, save each poem in a separate `".tex"` file, and typeset them with LaTeX commands. Such a task might be completed in, say, a weekend. Then, a `Canthology.cfg` file could be written to provide a title page and table of contents, and have an `\input` command for each poem. The result, obviously, would be a *complete* anthology of Shakespeare's poems. However, the result would be something else too: a *customisable* anthology of Shakespeare's poems. This customisability would offer benefits:

- Somebody who likes just a subset of the poems could easily delete (or comment out) `\input` commands for the disliked poems, and then rerun `canthology` to obtain an anthology of their favourite poems.

- A student who is studying a handful of Shakespearian poems for a course could trivially compile a very short anthology of just the poems on the course.

Of course, a limitation of such an anthology would be that, by default, it would not contain any critiques of the poems because such critiques tend to be new enough to still be in copyright. However, there would be nothing preventing budding literature geeks from writing their own critiques and contributing them to the anthology.

Customisable anthologies could be useful in other fields too. For example, food lovers might use Canthology to compile a large collection of recipes. The idea of yet another recipe book is not very exciting. But what *is* exciting is distributing the recipes in Canthology format, so people can edit the configuration file to create a personalised book that contains *only* the recipes they like.

14.9 Configuration Support for Additional LaTeX Tools

Currently, the `etc/defaults.cfg` file shipped with Canthology assumes people will use `pdflatex` to convert LaTeX documents into a printable format. However, not everybody uses `pdflatex`. Some people prefer to use `latex` to produce a `".dvi"` file, and then use a post-processing tool to convert that file into a printable format, such as PDF or PostScript. Other people prefer to use `luatex` or `xelatex`. Likewise, not everyone uses HeVeA to generate HTML documents. Some people prefer to use another tool, such as TTH, LaTeX2HTML, TeX4ht, or LaTeXML.

The choice of a LaTeX tool affects not only the `build_commands` configuration variable. It can also affect optional arguments passed to packages, and set-up commands used in the preamble of a document.

It would be nice to see Canthology extended to provide out-of-the-box configuration support for tools other than `pdflatex` and HeVeA. One way to do this would be to extend `etc/defaults.cfg` to contain scopes for each tool. However, doing that might result in the file growing too large to be easily maintained. Another approach would be to provide a separate configuration file for each tool. For example: `etc/pdflatex.cfg` could provide configuration support for `pdflatex`; `etc/hevea.cfg` could provide configuration support for HeVeA; `etc/luatex.cfg` could provide configuration support for `luatex`; and so on. Presumably, configuration settings that are common to many tools could be factored out into, say, `etc/common.cfg` and a tool-specific configuration file would access those settings via an `@include` statement.

14.10 Add Windows Support for Generating HTML

In Section 10.4 on page 111, I explained why the use of Canthology with HeVeA (to produce HTML files) is not supported on Windows. Overcoming this restriction would require several pieces of work to be carried out.

First, somebody would have to enhance the Windows port of HeVeA to contain ports of the UNIX utilities required to run `imagen`.

Second, Canthology uses a `Makefile` to execute the sequence of com-

mands required to generate HTML output. A Windows-compatible replacement (perhaps a `".bat"` file) would need to be written.

Finally, most of Canthology is written in Java, but Canthology also contains some simple Tcl scripts. Ideally, we should write Java replacements for those Tcl scripts. Doing this would eliminate the requirement to have Tcl installed on a computer, which would make it easier to run Canthology on a Windows-based PC.

14.11 A Graphical User Interface

Canthology is simple to use—at least for people who are comfortable executing commands from a command-line prompt. But, of course, many people do *not* know how to execute commands in a UNIX shell or a Windows command window. Such people would find Canthology much easier to use if there was a graphical user interface (GUI) "wrapper" for it.

14.12 A Web Interface

Consider the following scenario. Fred has no Canthology-related software installed on his computer. He briefly looks through this Canthology manual, thinks Canthology might be useful, and decides to try it. But to do so, he first has to install the following software:

- Canthology. The distribution is only a few MB large, so that is not a problem.
- A Java runtime. The distribution is about 20 MB. That is acceptable.
- Tcl. One popular distribution is about 27 MB. That is acceptable.
- A distribution of LaTeX. The distribution usually recommended for Windows (www.tug.org/protext) is about 1.1 GB, while that for Mac OS X (www.tug.org/mactex) is about 1.6 GB.

Having to install more than one GB of software might be enough to dissuade Fred from trying Canthology. But, if Canthology was installed

on a website and he needed only a web browser to use it, then Fred could try Canthology without having to install it. Fred could write some `".tex"` files and upload them to the website. An application on the website could help him create a configuration file. Then, when he clicks on a "create document" button, the web server would run `canthology` on his configuration file and `".tex"` files, and provide him with a downloadable PDF file.

Appendices

Appendix A

Commands in the `canthology` package

The numerous commands defined in the `canthology` package have been discussed in various chapters of this manual. This appendix summarise the names of those commands, and provide cross references to the sections in which the commands are discussed in detail.

Labelled constructs Section 12.2 on page 138

```
\lpart    \lchapter    \lsection    \lsubsection
\lsubsubsection    \lparagraph    \lsubparagraph
\lcaption
```

Anonymous section Section 2.3 on page 12

```
\anonymoussection
```

Author's name and details Sections 2.2–2.4, starting on page 10

```
\chapterAuthorInfo    \sectionAuthorInfo    \poemAuthorInfo
```

Reviewer praise for a book Section 9.9 on page 94

```
\praise
```

Cross references *without* page numbers Section 9.10.2 on page 102

`\xa`	`\xc`	`\xf`	`\xp`	`\xs`	`\xt`

Cross references *with* page numbers Section 9.10.2 on page 102

`\xap`	`\xcp`	`\xfp`	`\xpp`	`\xsp`	`\xtp`

Simplify `varioref` cross references Section 9.10.3 on page 105

`\simplifyVariorefReferences`

Page backgrounds Section 9.6 on page 87

`\thisPageBackgroundColor`	`\everyPageBackgroundColor`
`\thisPageBackgroundImage`	`\everyPageBackgroundImage`
`\thisPageBackgroundCommand`	`\everyPageBackgroundCommand`
`\clearPageBackground`	

Appendix B

Configuration File Syntax

B.1 Introduction

This appendix discusses all the syntax acceptable in configuration files. Figure B.1 provides a formal grammar for *most* of the syntax but, for brevity, the grammar omits some definitions. For example, the lexical definition of comments, strings and identifiers are discussed in text rather than being defined in the grammar of Figure B.1. Likewise, the string and list functions (denoted by `StringFunction` and `ListFunction` in the grammar) are discussed in text rather than being defined in the grammar.

B.2 Comments

A comment starts with the # character and continues until the end of the line, as shown in the example below:

```
# This is a comment
```

Comments are removed by the lexical analyser, which is why they are not mentioned in the formal grammar in Figure B.1.

Figure B.1: Formal grammar of Config4* syntax

Notation: | denotes choice, [...] denotes an optional component, and {...}* denotes 0 or more repetitions.

```
configFile    = StmtList
StmtList      = { Stmt }*
Stmt          = IDENTIFIER "=" StringExpr ";"
              | IDENTIFIER "=" ListExpr ";"
              | IDENTIFIER "{" StmtList "}" [ ";" ]
              | "@include" StringExpr [ "@ifExists" ] ";"
              | "@copyFrom" IDENTIFIER [ "@ifExists" ] ";"
              | "@remove" IDENTIFIER ";"
              | "@error" StringExpr ";"
              | "@if" "(" Condition ")" "{" StmtList "}"
                { "@elseIf" "(" Condition ")" "{" StmtList "}" }*
                [ "@else" "{" StmtList "}" ]
                [ ";" ]
StringExpr    = String { "+" String }*
String        = STRING
              | IDENTIFIER
              | StringFunction
ListExpr      = List { "+" List }*
List          = "[" "]"
              | "[" StringExpr { "," StringExpr }* [ "," ] "]"
              | IDENTIFIER
              | ListFunction
Condition     = OrCondition
OrCondition   = AndCondition { "||" AndCondition }*
AndCondition  = TermCondition { "&&" TermCondition }*
TermCondition = [ "!" ] "(" Condition ")"
              | StringExpr "==" StringExpr
              | StringExpr "!=" StringExpr
              | StringExpr "@in" ListExpr
              | StringExpr "@matches" StringExpr
```

B.3 Strings

There are two ways to write a `STRING`.

The first way is as a sequence of characters enclosed within double quotes. Within such a string, `%` acts as an escape character. The recognized escape sequences are as follows: `%n` denotes a newline character.;

`%t` denotes a TAB character; `%"` denotes a double quote; and `%%` denotes a percent sign. Many programming languages use `\` as an escape character, so the use of `%` may seem strange to some people. However, in my experience, using `\` as an escape character results in awkwardness when writing Windows-style directory names, such as `C:\temp\foo.txt`, which normally have to be written as `C:\\temp\\foo.txt`. Config4* uses `%` as the escape character to avoid this problem.

The second way to write a string is as a (possibly multi-line) sequence of characters enclosed between `<%` and `%>`. No escape sequences are recognised between `<%` and `%>`. If the `<%...%>` notation seems familiar to some readers it is because this notation is borrowed from Java Server Pages (JSP). The `<%...%>` notation is useful if you want to embed, say, a code segment in a configuration file.

You can combine both forms of string by using the string concatenation (`+`) operator. For example:

```
big_string = <%
    ... // some Java code
%> + "<%" + <%
    ... // some more Java code
%>;
```

B.4 Identifiers

An `IDENTIFIER` is a sequence of one or more of the following characters: upper- or lower-case letters, digits, a minus sign (`-`), an underscore (`_`), a colon (`:`), a period (`.`), a dollar sign (`$`), a question mark (`?`), a forward slash (`/`) or a backslash (`\`). There are two comments to be made about this range of allowable characters.

First, one goal of Config4* is to support internationalization, so accented characters (such as `á` and `ö`) and ideographs are permitted in an `IDENTIFIER`. Likewise, the digits permitted in an `IDENTIFIER` include the Roman digits (`0` through to `9`) as well as digits used in other scripts.

Second, a Config4* `IDENTIFIER` should be able to support names not just in many *human* languages, but also names in many *computer* lan-

guages. For example, ensuring that `Foo$Bar`, `X::Y::Z`, and `done?` are valid identifiers makes it possible for a Config4* file to store meta-data about applications written in many popular programming languages, such as C++, Java, Perl and Ruby. Likewise, permitting `/` and `\` in identifiers enables a Config4* file to contain meta-data about file names and (a useful subset of) URLs.

Config4* applies special treatment to any identifier that starts with `"uid-"`, for example, `uid-foo`. The `"uid-"` prefix denotes a *unique identifier*. Config4* modifies the name of a `"uid-"` prefixed variable by inserting a sequence of nine digits and a minus sign after `"uid-"`. For example, `uid-foo` might be changed to `uid-000000042-foo`. The nine-digit number starts at zero and is incremented by one for every encounter of an identifier that has a `"uid-"` prefix.

If Config4* encounters an identifier starting with `"uid-<digits>-"`, then the digits are *replaced* with a newly generated nine-digit number. This is to ensure correct behaviour in pathological cases. For example, consider a configuration file that contains multiple `uid-foo` identifiers. If this file is parsed and the `dump()` operation is used to save the parsed file to, say, `expanded-uid.cfg`, then the newly written file may contain identifiers of the form `uid-<digits>-foo`. Now consider another file of the form:

```
uid-foo { ... };
uid-foo { ... };
uid-foo { ... };
@include "expanded-uid.cfg";
```

When parsing the above file, it is necessary to replace the digits of the `uid-foo` entries contained in the `expanded-uid.cfg` file to ensure they do not conflict with the expanded form of the `uid-foo` entries defined before the `@include` command.

B.5 Assignment statements

An *unconditional* assignment statement takes the form:

```
name = value;
```

A *conditional* assignment statement takes the form:

```
name ?= value;
```

A conditional assignment statement assigns a value to the specified variable only if the variable does not already have a value.

Both conditional and unconditional assignment statements are terminated with a semicolon. A `value` can be either a *string* or a *list* of comma-separated strings inside matching brackets, that is, `[` and `]`.

```
local_domain = "bar.com";          # a string
some_fonts = ["Times", "Courier"]; # a list
```

You can use the + operator to concatenate strings and lists.

```
host = "foo." + local_domain;
all_fonts = some_fonts + ["Ariel", "Symbol"];
```

The above example also illustrates that one variable can be defined in terms of a previously defined variable. For example, the `host` variable is defined by concatenating a string and the `local_domain` variable.

B.6 Scopes

A configuration file can contain named scopes. The following example defines a scope called `server` that contains several assignment statements.

```
server {
    name              = "bankSrv";
    timeout           = "2 minutes";
    diagnostics_level = "2";
}
```

You can optionally place a semicolon after the closing brace of a scope. The reason for this is that a scope looks a bit like a class definition in C++ or Java. A semicolon appears after the class definition in C++, but not in Java.

```
class Foo { ... }; // C++
class Bar { ... }  // Java
```

Being flexible about whether or not a semicolon follows the closing brace of a scope makes it easy for people who come from a C++ *or* Java background.

You *cannot* use an `@include` statement (discussed in Section B.7) inside a scope. Instead, `@include` statements can be used only in the global scope.

The *fully scoped* name of a variable is its *local* name prefixed by the name of its enclosing scope and separated by a full stop/period. In the example at the start of this section, the fully scoped name of `timeout` is `server.timeout`. Use of scopes enables users to type *local* (that is, the short form of) names rather than the longer, *fully scoped* names. The example at the start of this section is equivalent to the following, more verbose example, which does not use scopes:

```
server.name              = "bankSrv";
server.timeout           = "2 minutes";
server.diagnostics_level = "2";
```

You can re-open scopes and nest them arbitrarily. For example:

```
outer {
    inner {
        foo = "Hello, world";
    };
};
outer.inner { # re-opening of scope
        bar = "Goodbye, world";
};
```

When a variable is used in an expression, the search for that variable usually starts at the current scope and works outwards. You can override this search order by prefixing the variable with a full stop/period; doing this instructs Config4* to look for the specified variable in the global scope. For example, the value of `outer.inner.food_1` below is `"apples`

and oranges", while the value of outer.inner.food_2 is "apples and bananas".

```
fruit = "bananas";
outer {
    fruit = "oranges";
    inner {
        food_1 = "apples and " + fruit;
        food_2 = "apples and " + .fruit;
    };
};
```

B.7 The @include statement

An @include statement instructs Config4* to parse the specified configuration file.

```
@include "/tmp/foo.cfg";
```

By default, @include reports an error if the specified file does not exist. However, if you place @ifExists at the end of an @include statement, then @include does not complain about a non-existent file.

```
@include "/tmp/foo.cfg" @ifExists;
```

The @include command can parse not just files, but also the output of executing an external command. This is done by using a string of the form "exec#command" as an argument to @include.

```
@include "exec#curl -sS http://localhost/someFile.cfg";
```

By default, @include reports an error if the specified command exits with an error status. You can instruct Config4* to ignore the unsuccessful execution of an @include command by placing @ifExists at the end of the @include statement.

```
@include "exec#curl -sS http://localhost/someFile.cfg"
         @ifExists;
```

B.8 The @copyFrom statement

The @copyFrom statement takes the following form:

```
@copyFrom "scope";
```

This command copies all the variables and nested scopes from the specified scope into the current scope. The typical use of this command is to copy default values from one scope into several other scopes, as Figure B.2 shows.

Figure B.2: Examples of the @copyFrom statement

```
acme {
    defaults {
        log {
            dir   = "C:\acme\logs";
            level = "0";
        };
        timeout = "2 minutes";
        thread_pool_size = "5";
    };
    app_1 {
        @copyFrom "acme.defaults";
    };
    app_2 {
        @copyFrom "acme.defaults";
        log.level = "1";
    };
    app_3 {
        @copyFrom "acme.defaults";
        thread_pool_size = "10";
    };
};
```

In this example, the acme.defaults scope contains all the configuration variables likely to have similar values in most of the applications (denoted by the scopes acme.app_1, acme.app_2 and acme.app_3). Then, the scope for a particular application, for example, acme.app_1, uses the @copyFrom command to copy the values from the acme.defaults scope. Notice that the acme.app_2 and acme.app_3 scopes copy all the values

from the `acme.defaults` scope, and then selectively override some values.

When using the `@copyFrom` statement, you must specify the *fully scoped* name of the scope to be copied. For example, the `@copyFrom` statements in Figure B.2 specify the scope as `acme.defaults` rather than as just `defaults`. If a configuration file contains deeply nested scopes, then specifying the fully scoped name of a scope to be copied can result in undesirable verbosity. However, Section B.12.6 on page 187 explains how the `siblingScope()` function can reduce such verbosity.

By default, `@copyFrom` reports an error if the specified scope does not exist. However, if you place `@ifExists` at the end of an `@copyFrom` statement, then `@copyFrom` does not complain about a non-existent scope.

```
@copyFrom "acme.defaults" @ifExists;
```

The `@ifExists` form of the `@copyFrom` command can be used to override some variables based on, for example, the operating system, the user running the application or the host on which the application is running.

```
override.pizza { ... }
override.pasta { ... }
fooSrv {
   # Set default values
   ...
   # Modify some values for particular hosts
   @copyFrom "override." + exec("hostname") @ifExists;
}
```

B.9 The `@if`-then-`@else` Statement

Figure B.3 shows some examples of `@if`-then-`@else` statements.

The conditions used in `@if`-then-`@else` statements can be in any of the following formats.

- `"string" == "another string"`
- `"string" != "another string"`

Figure B.3: Configuration file with advanced features

```
 1  production_hosts = ["pizza", "pasta", "zucchini"];
 2  test_hosts       = ["foo", "bar", "widget", "acme"];
 3
 4  @if (exec("hostname") @in production_hosts) {
 5      server_x.port = "5000";
 6      server_y.port = "5001";
 7      server_z.port = "5002";
 8  } @elseIf (exec("hostname") @in test_hosts) {
 9      server_x.port = "6000";
10      server_y.port = "6001";
11      server_z.port = "6002";
12  } @else {
13      @error "This is not a production or test machine";
14  }
15  @if (osType() == "windows") {
16      tmp_dir = replace(getenv("TMP"), "\", "/");
17  } @else {
18      tmp_dir = "/tmp";
19  }
```

- `"string" @in ["a", "list", "of", "string"]`

- `"string" @matches "pattern"`. Within the pattern, $*$ is a wildcard that matches zero or more characters. For example, the condition `"hello" @matches "*lo"` evaluates to true.

- `condition && condition`. This is the boolean AND of two conditions.

- `condition || condition`. This is the boolean OR of two conditions.

- `(condition)`. The parenthesis are used for grouping.

- `!(condition)`. This is the negation of a condition.

B.10 The `@error` Statement

The `@error` statement instructs Config4* to stop parsing and instead report an error.

```
@error "Something has gone wrong";
```

Config4* reports the error by throwing an exception back to application code. The application code should communicate the exception's text message to the user, for example, by writing the text message to a console or displaying it in a dialog box of a graphical user application.

B.11 The `@remove` Statement

The `@remove` statement removes a previously-defined variable or scope. To see why the `@remove` command might be useful, let us assume you want to specify the full path names of several log files that happen to reside in the same directory. It would be tedious to write the full path name of the directory for each log file. Instead, you can define a temporary variable called, say, `_log_dir` and used it as follows:

```
_log_dir = "/path/to/log/dir";
app1_log_file = _log_dir + "/app1.log";
app2_log_file = _log_dir + "/app2.log";
app3_log_file = _log_dir + "/app3.log";
@remove _log_dir;
```

A useful convention shown in the above example is to use an underscore at the start of the name of a temporary variable. This makes it easy to see which variables are "normal" variables and which are temporary ones that will be removed later.

You may be wondering why temporary variables should be removed at all. There are two reasons for this. First, unneeded variables clutter up a configuration file, potentially causing confusion for users. Second, by insisting a configuration scope contain *only* required variables, an application can make use of a schema validator that can perform extensive error checking on the contents of a configuration scope.

B.12 Functions

Table B.1 lists the functions that Config4* provides.

Table B.1: Config4* functions

Function	Return type	Section
`configFile()`	string	B.12.5
`configType("name")`	string	B.12.6
`exec("command")`	string	B.12.3
`exec("command", "default value")`	string	B.12.3
`fileToDir("/path/to/file.txt")`	string	B.12.5
`getenv("name")`	string	B.12.2
`getenv("name", "default value")`	string	B.12.2
`isFileReadable("fileName.txt")`	boolean	B.12.6
`join(["list", "of", "string"], " ")`	string	B.12.4
`osDirSeparator()`	string	B.12.1
`osPathSeparator()`	string	B.12.1
`osType()`	string	B.12.1
`readFile("/path/to/file.txt")`	string	B.12.5
`replace("\a\b\c", "\", "/")`	string	B.12.4
`siblingScope("name")`	string	B.12.6
`split("red green blue", " ")`	list	B.12.4

Config4* considers `(` to be part of a function name, so you cannot place a space before it. For example, Config4* accepts the first statement below, but reports an error for the second statement:

```
x = configFile();  # okay
y = configFile (); # error
```

Treating `(` as being part of a function's name might seem strange, but Config4* does this to guarantee that the names of functions do not conflict with the names of variables or scopes. This makes it possible for future versions of Config4* to provide additional functions without any risk of the newly added functions causing problems for existing configuration files.

The following subsections discuss the functions in logical groupings.

B.12.1 Querying the Operating System

Some of the built-in functions have names starting with `"os"`, which indicates they return information about the operating system environment.

The `osType()` function returns `"windows"` if you are running on a Microsoft Windows-based computer, and `"unix"` if you are running on a UNIX-based computer.

The `osDirSeparator()` function returns the character that the operating system uses as a directory separator. This is `\` on Windows and `/` on UNIX.

The `osPathSeparator()` function returns the character that the operating system uses to separate a list of directories. This is `;` on Windows and `:` on UNIX.

B.12.2 Accessing Environment Variables

The `getenv()` function enables you to access an environment variable. This function can take either one or two parameters. The first parameter is the name of the environment variable to access:

```
example = getenv("FOO_HOME");
```

The second (and optional) parameter to this function is a default value that is used if the specified environment variable does not exist:

```
example = getenv("FOO_HOME", "/tmp");
```

If you do not specify a default value and the specified environment variable does not exist, then Config4* reports an error:

```
someFile.cfg, line 12: cannot access the 'FOO_HOME'
environment variable
```

B.12.3 Executing External Commands

The `exec()` function executes an external command and returns whatever text that command writes to its standard output. This function can take either one or two parameters. The first parameter is the external command to execute, as the following examples illustrate:

```
example_1 = exec("hostname");
example_2 = exec("ls /tmp");
example_3 = exec("ls " + getenv("HOME", "/") );
```

The second (and optional) parameter to this function is a default value that is used if Config4* cannot successfully execute the specified external command:

```
example = exec("hostname", "localhost");
```

If you do not specify a default value and Config4* cannot successfully execute the specified external command, then Config4* reports an error:

```
someFile.cfg, line 3: exec("ls /x/y/z") failed:
ls: /x/y/z: No such file or directory
```

B.12.4 Manipulating Strings and Lists

The example below illustrates the `split()` and `join()` functions:

```
colours_and_spaces = "red green blue";
colour_list = split(colours_and_spaces, " ");
colours_and_commas = join(colour_list, ",");
```

The `split()` function takes two parameters. The first parameter is a string to be broken up into a list of smaller strings. The second parameter indicates a search string; the first string is broken into list elements at each occurrence of this search string. In the above example, `colour_list` is assigned the value `["red", "green", "blue"]`.

The `join()` function is the opposite of `split()`. It takes two parameters; the first parameter is a list and the second parameter is a string. The `join()` function concatenates all the elements of the list using the string as a separator. In the above example, `colours_and_commas` is assigned the value `"red,green,blue"`.

In the above example, the overall effect of using `split()` and `join()` is to *replace* all spaces in a string with commas. To make this easier, Config4* provides a `replace()` function.

```
colours_and_commas=replace("red green blue", " ", ",");
```

The `replace()` function takes three string parameters: *original*, *search* and *replacement*. This function replaces all occurrences of the *search* string in the *original* string with the *replacement* string.

B.12.5 Files and Directories

The `configFile()` function does not take any parameters; it returns the name of the configuration file being parsed.

The `fileToDir()` function takes one parameter—the name of a file—and returns the name of the directory in which that file resides. The returned directory name is guaranteed to not have `/` or `\` at the end. For example, `fileToDir("/tmp/foo.cfg")` returns `"/tmp"`. As the table in Table B.2 shows, the `fileToDir()` function works even for boundary cases, such as for files in the root directory of a file system.

Table B.2: Example results of calling `fileToDir()`

`filename`	`fileToDir(filename)`
`"/tmp/foo.cfg"`	`"/tmp"` (UNIX and Windows)
`"C:\tmp\foo.cfg"`	`"C:\tmp"` (Windows only)
`"foo.cfg"`	`"."` (UNIX and Windows)
`"/foo.cfg"`	`"/."` (UNIX and Windows)
`"\foo.cfg"`	`"\."` (Windows only)
`"C:\foo.cfg"`	`"C:\."` (Windows only)

The combination `fileToDir(configFile())` returns the directory in which the configuration file being parsed resides. This can be useful if you want to write a top-level configuration file that includes other configuration files residing within the same directory. For example:

```
@include fileToDir(configFile()) + "/file1.cfg";
@include fileToDir(configFile()) + "/file2.cfg";
@include fileToDir(configFile()) + "/file3.cfg";
```

This technique can work even if the configuration file is hosted on a web server and is being accessed through the `curl` utility. To see why, let's

assume the top-level configuration file is specified as:

```
exec#curl -sS http://myHost/foo/foo.cfg
```

Config4* will execute that command and parse its output. During this parsing, the `configFile()` function returns:

```
exec#curl -sS http://myHost/foo/foo.cfg
```

The `fileToDir()` function does not check that its parameter is a valid file name; rather it just trims its parameter back to the last occurrence of `/` or `\`, so the result of `fileToDir(configFile())` is:

```
exec#curl -sS http://myHost/foo
```

The first `@include` statement in the example appends `"/file1.cfg"`, so the `@include` statement becomes:

```
@include "exec#curl -sS http://myHost/foo/file1.cfg";
```

One thing to keep in mind is that downloading a multi-part configuration file from a web server will be slower than downloading a monolithic configuration file. It will probably take just a fraction of a second longer to download the multi-part configuration file, so you might think that such an overhead is insignificant. However, in a large organization there might be thousands of users downloading their applications' configuration files from the same web server. In such an organization, all those fractions of a second extra overhead might add up to be a significant overhead.

B.12.6 Miscellaneous Functions

The `configType()` function takes a string parameter that specifies the fully-scoped name of an entry in the configuration file. It returns the value `"string"` if the entry is a string variable, `"list"` if the entry is a list variable, `"scope"` if the entry is a scope, or `"no_value"` if there is no such entry.

The `isFileReadable()` function takes a string parameter that specifies the name of a file. It returns `true` if the file exists and is readable; it returns `false` otherwise. An example of the intended use of this function is shown below:

```
files_to_process = ["file1.txt", "file2.txt"];
@if (isFileReadable("file3.txt")) {
    files_to_process = files_to_process + ["file3.txt"];
}
```

The `siblingScope()` function takes a string parameter that specified the local name of a scope that is a sibling of the current scope. It returns the fully scoped name of the specified scope. This function is provided to simplify a common use case of the `@copyFrom` statement that is shown in Figure B.4.

Figure B.4: Verbose `@copyFrom` statements

```
acme.uk.london.sales {
    defaults {
        timeout = "2 minutes";
        log.level = "1";
    }
    app1 {
        @copyFrom "acme.uk.london.sales.defaults";
    }
    app2 {
        @copyFrom "acme.uk.london.sales.defaults";
       log.level = "0";
    }
    app3 {
        @copyFrom "acme.uk.london.sales.defaults";
       log.level = "0";
    }
}
```

It is common for the `@copyFrom` statement to be used to copy the contents of a scope that is at the same level of nesting—what I call a *sibling* scope. If the sibling scope is deeply nested in the configuration file, then, as shown in Figure B.4, the `@copyFrom` statement can be quite verbose. If, later on, the scope hierarchy is renamed (perhaps by being copy-and-pasted to another part of the configuration file), then all the `@copyFrom` statements will have to be updated to specify the renamed sibling scope. Doing this can be tedious and error-prone.

Figure B.5 shows the configuration file after it has been modified to make use of the `siblingScope()` function. The `@copyFrom` statements in this modified file are more concise and easier to visually verify for correctness. In addition, if the `acme.uk.london.sales` scope is renamed, then the `@copyFrom` statements will continue to work without any need for updating.

Figure B.5: Using `siblingScope()` to get concise `@copyFrom` statements

```
acme.uk.london.sales {
    defaults {
        timeout = "2 minutes";
        log.level = "1";
    }
    app1 {
        @copyFrom siblingScope("defaults");
    }
    app2 {
        @copyFrom siblingScope("defaults");
       log.level = "0";
    }
    app3 {
        @copyFrom siblingScope("defaults");
       log.level = "0";
    }
}
```

Appendix C

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

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

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

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

1. APPLICABILITY AND DEFINITIONS

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

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

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

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the

Document does not identify any Invariant Sections then there are none.

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

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

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

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

The "**publisher**" means any person or entity that distributes copies of

the Document to the public.

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

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

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 (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover

Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

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

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

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

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 fewer than five), unless they release you from this requirement.

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

D. Preserve all the copyright notices of the Document.

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

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

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

H. Include an unaltered copy of this License.

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

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may

omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, 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 to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

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

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

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement

made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

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

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, and that you preserve all their Warranty Disclaimers.

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

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “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, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

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

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, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

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

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

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

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License

"or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your

documents

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

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

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

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

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

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

Appendix D

The GNU General Public License

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you

wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

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

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot

be used to render the program non-free.

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

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells

the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to

require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

   (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7.

This requirement modifies the requirement in section 4 to "keep intact all notices".

(c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

(d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

(a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

(b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

(c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

(d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

(e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is ex-

cluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include

a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

(a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

(b) Requiring preservation of specified reasonable legal notices or

author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

(c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

(d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

(e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

(f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent

sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms

of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

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

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

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,

THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make

it free software which everyone can redistribute and change under these terms.

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

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

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

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

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

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

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program,

if necessary. For more information on this, and how to apply and follow the GNU GPL, see http://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.

Appendix E

The LaTeX Project Public License

LPPL Version 1.3c 2008-05-04

Preamble

The LaTeX Project Public License (LPPL) is the primary license under which the LaTeX kernel and the base LaTeX packages are distributed.

You may use this license for any work of which you hold the copyright and which you wish to distribute. This license may be particularly suitable if your work is TeX-related (such as a LaTeX package), but it is written in such a way that you can use it even if your work is unrelated to TeX.

The section 'WHETHER AND HOW TO DISTRIBUTE WORKS UNDER THIS LICENSE', below, gives instructions, examples, and recommendations for authors who are considering distributing their works under this license.

This license gives conditions under which a work may be distributed and modified, as well as conditions under which modified versions of that work may be distributed.

We, the LaTeX3 Project, believe that the conditions below give you the freedom to make and distribute modified versions of your work that conform with whatever technical specifications you wish while maintaining the availability, integrity, and reliability of that work. If you do not see how to achieve your goal while meeting these conditions, then read the document '`cfgguide.tex`' and '`modguide.tex`' in the base LaTeX distribution for suggestions.

Definitions

In this license document the following terms are used:

Work Any work being distributed under this License.

Derived Work Any work that under any applicable law is derived from the Work.

Modification Any procedure that produces a Derived Work under any applicable law – for example, the production of a file containing an original file associated with the Work or a significant portion of such a file, either verbatim or with modifications and/or translated into another language.

Modify To apply any procedure that produces a Derived Work under any applicable law.

Distribution Making copies of the Work available from one person to another, in whole or in part. Distribution includes (but is not limited to) making any electronic components of the Work accessible by file transfer protocols such as FTP or HTTP or by shared file systems such as Sun's Network File System (NFS).

Compiled Work A version of the Work that has been processed into a form where it is directly usable on a computer system. This processing may include using installation facilities provided by the Work, transformations of the Work, copying of components of the Work, or

other activities. Note that modification of any installation facilities provided by the Work constitutes modification of the Work.

Current Maintainer A person or persons nominated as such within the Work. If there is no such explicit nomination then it is the 'Copyright Holder' under any applicable law.

Base Interpreter A program or process that is normally needed for running or interpreting a part or the whole of the Work.

A Base Interpreter may depend on external components but these are not considered part of the Base Interpreter provided that each external component clearly identifies itself whenever it is used interactively. Unless explicitly specified when applying the license to the Work, the only applicable Base Interpreter is a 'LaTeX-Format' or in the case of files belonging to the 'LaTeX-format' a program implementing the 'TeX language'.

Conditions on Distribution and Modification

1. Activities other than distribution and/or modification of the Work are not covered by this license; they are outside its scope. In particular, the act of running the Work is not restricted and no requirements are made concerning any offers of support for the Work.

2. You may distribute a complete, unmodified copy of the Work as you received it. Distribution of only part of the Work is considered modification of the Work, and no right to distribute such a Derived Work may be assumed under the terms of this clause.

3. You may distribute a Compiled Work that has been generated from a complete, unmodified copy of the Work as distributed under Clause 2 above, as long as that Compiled Work is distributed in such a way that the recipients may install the Compiled Work on their system exactly as it would have been installed if they generated a Compiled Work directly from the Work.

4. If you are the Current Maintainer of the Work, you may, without restriction, modify the Work, thus creating a Derived Work. You may also distribute the Derived Work without restriction, including Compiled Works generated from the Derived Work. Derived Works distributed in this manner by the Current Maintainer are considered to be updated versions of the Work.

5. If you are not the Current Maintainer of the Work, you may modify your copy of the Work, thus creating a Derived Work based on the Work, and compile this Derived Work, thus creating a Compiled Work based on the Derived Work.

6. If you are not the Current Maintainer of the Work, you may distribute a Derived Work provided the following conditions are met for every component of the Work unless that component clearly states in the copyright notice that it is exempt from that condition. Only the Current Maintainer is allowed to add such statements of exemption to a component of the Work.

   (a) If a component of this Derived Work can be a direct replacement for a component of the Work when that component is used with the Base Interpreter, then, wherever this component of the Work identifies itself to the user when used interactively with that Base Interpreter, the replacement component of this Derived Work clearly and unambiguously identifies itself as a modified version of this component to the user when used interactively with that Base Interpreter.

   (b) Every component of the Derived Work contains prominent notices detailing the nature of the changes to that component, or a prominent reference to another file that is distributed as part of the Derived Work and that contains a complete and accurate log of the changes.

   (c) No information in the Derived Work implies that any persons, including (but not limited to) the authors of the original version of the Work, provide any support, including (but not limited to)

the reporting and handling of errors, to recipients of the Derived Work unless those persons have stated explicitly that they do provide such support for the Derived Work.

(d) You distribute at least one of the following with the Derived Work:

i. A complete, unmodified copy of the Work; if your distribution of a modified component is made by offering access to copy the modified component from a designated place, then offering equivalent access to copy the Work from the same or some similar place meets this condition, even though third parties are not compelled to copy the Work along with the modified component;

ii. Information that is sufficient to obtain a complete, unmodified copy of the Work.

7. If you are not the Current Maintainer of the Work, you may distribute a Compiled Work generated from a Derived Work, as long as the Derived Work is distributed to all recipients of the Compiled Work, and as long as the conditions of Clause 6, above, are met with regard to the Derived Work.

8. The conditions above are not intended to prohibit, and hence do not apply to, the modification, by any method, of any component so that it becomes identical to an updated version of that component of the Work as it is distributed by the Current Maintainer under Clause 4, above.

9. Distribution of the Work or any Derived Work in an alternative format, where the Work or that Derived Work (in whole or in part) is then produced by applying some process to that format, does not relax or nullify any sections of this license as they pertain to the results of applying that process.

10. (a) A Derived Work may be distributed under a different license provided that license itself honors the conditions listed in

Clause 6 above, in regard to the Work, though it does not have to honor the rest of the conditions in this license.

(b) If a Derived Work is distributed under a different license, that Derived Work must provide sufficient documentation as part of itself to allow each recipient of that Derived Work to honor the restrictions in Clause 6 above, concerning changes from the Work.

11. This license places no restrictions on works that are unrelated to the Work, nor does this license place any restrictions on aggregating such works with the Work by any means.

12. Nothing in this license is intended to, or may be used to, prevent complete compliance by all parties with all applicable laws.

No Warranty

There is no warranty for the Work. Except when otherwise stated in writing, the Copyright Holder provides the Work 'as is', without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Work is with you. Should the Work prove defective, you assume the cost of all necessary servicing, repair, or correction.

In no event unless required by applicable law or agreed to in writing will The Copyright Holder, or any author named in the components of the Work, or any other party who may distribute and/or modify the Work as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of any use of the Work or out of inability to use the Work (including, but not limited to, loss of data, data being rendered inaccurate, or losses sustained by anyone as a result of any failure of the Work to operate with any other programs), even if the Copyright Holder or said author or said other party has been advised of the possibility of such damages.

Maintenance of The Work

The Work has the status 'author-maintained' if the Copyright Holder explicitly and prominently states near the primary copyright notice in the Work that the Work can only be maintained by the Copyright Holder or simply that it is 'author-maintained'.

The Work has the status 'maintained' if there is a Current Maintainer who has indicated in the Work that they are willing to receive error reports for the Work (for example, by supplying a valid e-mail address). It is not required for the Current Maintainer to acknowledge or act upon these error reports.

The Work changes from status 'maintained' to 'unmaintained' if there is no Current Maintainer, or the person stated to be Current Maintainer of the work cannot be reached through the indicated means of communication for a period of six months, and there are no other significant signs of active maintenance.

You can become the Current Maintainer of the Work by agreement with any existing Current Maintainer to take over this role.

If the Work is unmaintained, you can become the Current Maintainer of the Work through the following steps:

1. Make a reasonable attempt to trace the Current Maintainer (and the Copyright Holder, if the two differ) through the means of an Internet or similar search.

2. If this search is successful, then enquire whether the Work is still maintained.

   (a) If it is being maintained, then ask the Current Maintainer to update their communication data within one month.

   (b) If the search is unsuccessful or no action to resume active maintenance is taken by the Current Maintainer, then announce within the pertinent community your intention to take over maintenance. (If the Work is a LaTeX work, this could be done, for example, by posting to `comp.text.tex`.)

3. (a) If the Current Maintainer is reachable and agrees to pass maintenance of the Work to you, then this takes effect immediately upon announcement.

   (b) If the Current Maintainer is not reachable and the Copyright Holder agrees that maintenance of the Work be passed to you, then this takes effect immediately upon announcement.

4. If you make an 'intention announcement' as described in 2b above and after three months your intention is challenged neither by the Current Maintainer nor by the Copyright Holder nor by other people, then you may arrange for the Work to be changed so as to name you as the (new) Current Maintainer.

5. If the previously unreachable Current Maintainer becomes reachable once more within three months of a change completed under the terms of 3b or 4, then that Current Maintainer must become or remain the Current Maintainer upon request provided they then update their communication data within one month.

A change in the Current Maintainer does not, of itself, alter the fact that the Work is distributed under the LPPL license.

If you become the Current Maintainer of the Work, you should immediately provide, within the Work, a prominent and unambiguous statement of your status as Current Maintainer. You should also announce your new status to the same pertinent community as in 2b above.

Whether and How to Distribute Works under This License

This section contains important instructions, examples, and recommendations for authors who are considering distributing their works under this license. These authors are addressed as 'you' in this section.

Choosing This License or Another License

If for any part of your work you want or need to use *distribution* conditions that differ significantly from those in this license, then do not refer to this license anywhere in your work but, instead, distribute your work under a different license. You may use the text of this license as a model for your own license, but your license should not refer to the LPPL or otherwise give the impression that your work is distributed under the LPPL.

The document '`modguide.tex`' in the base LaTeX distribution explains the motivation behind the conditions of this license. It explains, for example, why distributing LaTeX under the GNU General Public License (GPL) was considered inappropriate. Even if your work is unrelated to LaTeX, the discussion in '`modguide.tex`' may still be relevant, and authors intending to distribute their works under any license are encouraged to read it.

A Recommendation on Modification Without Distribution

It is wise never to modify a component of the Work, even for your own personal use, without also meeting the above conditions for distributing the modified component. While you might intend that such modifications will never be distributed, often this will happen by accident – you may forget that you have modified that component; or it may not occur to you when allowing others to access the modified version that you are thus distributing it and violating the conditions of this license in ways that could have legal implications and, worse, cause problems for the community. It is therefore usually in your best interest to keep your copy of the Work identical with the public one. Many works provide ways to control the behavior of that work without altering any of its licensed components.

How to Use This License

To use this license, place in each of the components of your work both an explicit copyright notice including your name and the year the work was authored and/or last substantially modified. Include also a statement that

the distribution and/or modification of that component is constrained by the conditions in this license.

Here is an example of such a notice and statement:

```
%% pig.dtx
%% Copyright 2005 M. Y. Name
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3 or later is part of all distributions of LaTeX
% version 2005/12/01 or later.
%
% This work has the LPPL maintenance status 'maintained'.
%
% The Current Maintainer of this work is M. Y. Name.
%
% This work consists of the files pig.dtx and pig.ins
% and the derived file pig.sty.
```

Given such a notice and statement in a file, the conditions given in this license document would apply, with the 'Work' referring to the three files '`pig.dtx`', '`pig.ins`', and '`pig.sty`' (the last being generated from '`pig.dtx`' using '`pig.ins`'), the 'Base Interpreter' referring to any 'LaTeX-Format', and both 'Copyright Holder' and 'Current Maintainer' referring to the person 'M. Y. Name'.

If you do not want the Maintenance section of LPPL to apply to your Work, change 'maintained' above into 'author-maintained'. However, we recommend that you use 'maintained' as the Maintenance section was added in order to ensure that your Work remains useful to the community even when you can no longer maintain and support it yourself.

Derived Works That Are Not Replacements

Several clauses of the LPPL specify means to provide reliability and stability for the user community. They therefore concern themselves with the case that a Derived Work is intended to be used as a (compatible or incompatible) replacement of the original Work. If this is not the case (e.g., if a

few lines of code are reused for a completely different task), then clauses 6b and 6d shall not apply.

Important Recommendations

Defining What Constitutes the Work

The LPPL requires that distributions of the Work contain all the files of the Work. It is therefore important that you provide a way for the licensee to determine which files constitute the Work. This could, for example, be achieved by explicitly listing all the files of the Work near the copyright notice of each file or by using a line such as:

```
% This work consists of all files listed in manifest.txt.
```

in that place. In the absence of an unequivocal list it might be impossible for the licensee to determine what is considered by you to comprise the Work and, in such a case, the licensee would be entitled to make reasonable conjectures as to which files comprise the Work.

Bibliography

[1] Donald Arseneau. *The `framed` package*, 2007. http://mirrors.ctan.org/macros/latex/contrib/framed/framed.pdf.

[2] David Carlisle. *The `ifthen` package*, 2001. www.tug.org/texlive/devsrc/Master/texmf-dist/doc/latex/base/ifthen.pdf.

[3] D.P. Carlisle and The LaTeX3 Project. *Packages in the 'graphics' bundle*, 2005. http://ctan.tug.org/tex-archive/macros/latex/required/graphics/grfguide.pdf.

[4] Peter Flynn. *Formatting information: An introduction to typesetting with LaTeX*. A free PDF version of this book is available online: http://latex.silmaril.ie/formattinginformation/beginlatex-a4.pdf.

[5] Leslie Lamport. *LaTeX: a Document Preparation System*. Addison Wesley, second edition, 1994. You should avoid the first edition of this book because it covers the now obsolete version 2.09 of LaTeX. The second edition of the book has been updated for the current version of LaTeX, which is called LaTeX 2_ε.

[6] Luc Maranget. *HeVeA User Documentation*, 2007. http://hevea.inria.fr/distri/hevea-1.10-manual.pdf.

[7] Frank Mittelbach. *The `varioref` package*, 2011. http://mirror.ctan.org/macros/latex/required/tools/varioref.pdf.

[8] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, and Chris Rowley. *The LaTeX Companion*. Addison Wesley,

second edition, 2004. This large book (approximately 1100 pages) discusses over 200 packages that are available for LaTeX.

[9] Rolf Niepraschk. *The `eso-pic` package*, 2010. http://ctan.tug.org/tex-archive/macros/latex/contrib/eso-pic/eso-pic.pdf.

[10] Tobias Oetiker. *The Not So Short Introduction to LaTeX 2ε*. 2010. A free PDF version of this book is available online: http://tobi.oetiker.ch/lshort/lshort.pdf. If you prefer, you can buy a printed copy: www.lulu.com/product/paperback/the-not-so-short-introduction-to-latex/12552267.

[11] The LaTeX3 Project. *LaTeX 2ε for class and package writers*, 1998. www.latex-project.org/guides/clsguide.pdf.

[12] Hideo Umeki. *The `geometry` package*, 2010. http://mirrors.ctan.org/macros/latex/contrib/geometry/geometry.pdf.

[13] Peter Wilson. *Some Examples of Title Pages*, 2010. http://ctan.tug.org/info/latex-samples/titlepages.pdf.

[14] Peter Wilson and Lars Madsen. *The Memoir Class for Configurable Typesetting User Guide*. 2010. This book is available online: www.tex.ac.uk/ctan/macros/latex/contrib/memoir/memman.pdf.

[15] Peter Wilson and Will Robertson. *The `appendix` package*, 2009. http://mirrors.ctan.org/macros/latex/contrib/appendix/appendix.pdf.

Colophon

The contents of this manual were formatted using the markup language of the LaTeX typesetting system.[1] Each chapter was written in its own `".tex"` file, and Canthology[2] was used to create a root `".tex"` file that included all the other `".tex"` files. Canthology then used `pdflatex` to generate PDF versions of the manual, and HeVeA[3] to create a HTML version.

The main font used in the PDF versions of the manual is 10pt Times rather than the default Computer Modern fonts supplied with LaTeX. The decision to use Times was made because the author finds it easier to read on computer screens. Beramono (scaled to 80% of its normal size) is used as the typewriter-like font in examples.

[1] www.latex-project.org
[2] www.canthology.org
[3] http://hevea.inria.fr/