LaTeX for Administrative Work

Version 1.3

Nicola L. C. Talbot

Dickimaw Books
www.dickimaw-books.com

6th September, 2021

The base URL for this document is: http://www.dickimaw-books.com/latex/admin/

Contents

List of Figures

List of Tables

List of Exercises

List of Examples

1. Introduction

This book is aimed at people who want to use LaTeX for administrative work, such as writing correspondence, performing repetitive tasks or typesetting problem sheets or exam papers. If you have never used LaTeX before, I recommend that you first read Volume 1: *LaTeX for Complete Novices* [92], since this book assumes you are already familiar with LaTeX.

As with all the books in this series, this tutorial only shows the basic usage of class files and packages. For more advanced commands, you will need to consult the class or package documentation (see Volume 1 [92, §1.1]). The reason for this is that it would be far too overwhelming for most readers to be presented with every possible option. (Consider, for example, the KOMA-Script manual is over 300 pages and the datatool user guide is over 200 pages at the time of writing.)

Throughout this document there are pointers to related topics in the UK List of TeX Frequently Asked Questions (UK FAQ). These are displayed in the margin in square brackets, as illustrated on the right. You may find these resources useful in answering related questions that are not covered in this book.

[FAQ: What is LaTeX?]

On-line versions of this book, along with associated files and solutions to the exercises, are available from this book's home page. This includes a link to the HTML version of this book that mostly uses `object` instead of `img` tags to include images to allow for more detailed alternative text to assist screen readers. The links in this PDF document are colour-coded: internal links are blue, external links are magenta.

The topics covered by this book range from fairly basic (assumes you know how to load a document class and packages) to advanced. To help you navigate your way around this book, sections have symbols to denote the difficulty level. If you only want to learn how to do straight-forward tasks, such as writing a letter without looking up data, you can skip the harder sections. The symbols are as follows:

Basic concepts. This may include common LaTeX commands described in the earlier volumes or fairly basic commands defined by a simple class or package.

Intermediate. This may include more complicated class or package commands, or there may be a wide range of settings (typically key=value lists) which can appear a little bewildering at first glance. You may also need to use external applications as part of the document build.

Advanced. This may include core TEX commands, internal LATEX kernel commands, or programming concepts.

Most chapters start with the basic or intermediate symbol, but they may progress to harder sections. Some of the exercises have a "More Adventurous" part, which increases the difficulty level. There is, of course, a certain amount of subjectivity in choosing the classifications for each section. What one person may find straight-forward, may be more difficult to understand for someone else, so these are just general guides.

To refresh your memory or for those who haven't read other volumes in this series, throughout this book source code is illustrated in a typewriter font with the word Input placed in the margin, and the corresponding output (how it will appear in the PDF document) is typeset with the word Output in the margin.

Example:
A single line of code is displayed like this:

```
This is an \textbf{example}.
```
Input

The corresponding output is illustrated like this:

This is an **example**.
Output

Segments of code that are longer than one line are bounded above and below, illustrated as follows:

```
Line one\par
Line two\par
Line three.
```

↑ Input

↓ Input

with corresponding output:

Line one
Line two
Line three.

↑ Output

↓ Output

(Commands typeset in blue, such as \par, indicate a hyperlink to the command definition in the summary.)

Command definitions are shown in a typewriter font in the form:

`\documentclass[`⟨*options*⟩`]{`⟨*class file*⟩`}` Definition

In this case the command being defined is called `\documentclass` and text typed ⟨*like this*⟩ (such as ⟨*options*⟩ and ⟨*class file*⟩) indicates the type of thing you need to substitute. (Don't type the angle brackets!) For example, if you want the scrbook class [46] you would substitute ⟨*class file*⟩ with `scrbook` and if you want the `letterpaper` option you would substitute ⟨*options*⟩ with `letterpaper`, like this:

```
\documentclass[letterpaper]{scrbook}
```
Input

When it's important to indicate a space, the visible space symbol ␣ is used. For example:

```
A␣sentence␣consisting␣of␣six␣words.
```
Input

When you type up the code, replace any occurrences of ␣ with a space. [FAQ: Spaces in macros]

Recall from Volume 1 [92, §2] that the comment character `%` is often used to suppress unwanted space caused by the end of line (EOL) character in the source code. For example:

```
Foo%
Bar
```
↑ Input

↓ Input

produces:

FooBar Output

Any applications that need to be run from a command prompt or terminal (see Volume 1 [92, §2.5]) are displayed in the form:

```
pdflatex mydocument.tex
```
Shell

These should be typed at the command prompt not in your LaTeX document.

If a line of code must be typed without any EOL characters but is too long to display on the page, the symbol ↩ is used to indicate a line wrap. Make sure you don't insert a line break at that point.

1.1 Packages and Document Classes

If you already have some basic knowledge of LaTeX, you'll know that a class is loaded using

```
\documentclass[⟨options⟩]{⟨name⟩}
```
Input

and packages are loaded using

`\usepackage[`⟨*options*⟩`]{`⟨*name*⟩`}` Input

If you are using an operating system with case-sensitive filenames (for example, Unix-like systems), the ⟨*name*⟩ part must use the same case as the corresponding filename ⟨*name*⟩`.cls` or ⟨*name*⟩`.sty`. If the case doesn't match, the file won't be found. In the instance of operating systems with case-insensitive filenames (such as Windows) the file will be found but there will most likely be a warning that ⟨*name*⟩ doesn't match the declared class or package name. To assist you, this book will always match the filename case when referring to a class or package, rather than the format used in the class or package's manual. (For example, pgf/tikz rather than PGF/TikZ or pstricks rather than PSTricks.) If I use any upper case characters (for example, Alegreya or DejaVuSerif) then that means the filename actually contains those upper case characters.

Recall from Volume 1 [92, §1.1] that package and class documentation can be accessed via the `texdoc` application. For example:

```
texdoc datatool
```
Shell

In the past couple of years that I've been writing this book, I've submitted more bug reports than I have over the previous twenty or more years of using LaTeX. If it's of any consolation to my fellow coders, I found more bugs in my own code than anyone else's. However this leads me onto a

topic I haven't really covered in the previous two volumes (except, perhaps, briefly in Volume 1 [92, §C]) and that's the difference between commercial software and open source software (aside from the obvious one of price).

When a company decides to produce a new piece of software, they usually hire someone to do some market research to find out what potential customers might need, then they hire a programmer (or group of programmers) to write the code and then the software gets tested by a group of product testers. After the buyer has purchased the software, they usually have access to a support desk, customer helpline or chat. The person on the support desk may just have a crib sheet of stock solutions for common issues and if the issue isn't on that list, the customer will probably be referred to a more technical support person. In a few years' time, the company may decide it's no longer financially viable to continue to provide or support that product. If the customer upgrades their operating system (or buys a new computer with a newer operating system) the software may no longer be able to run (or even install), and customers may find themselves in a position where they can no longer access their data that's stored in a proprietary format.

Open source software, on the other hand, usually starts life when someone has a need for an application or piece of code that doesn't exist or exists but is beyond their budget or exists but doesn't run on their operating system. If that person happens to be able to write code (or, perhaps, decides

to take this opportunity to learn) then they may be able to write something that fixes their problem. If it achieves what they wanted to do, there's a chance it might be useful to someone else with a similar requirement, so they make it publicly available with an open source licence and an "as is" caveat. There are two most likely possibilities: firstly, once the task has been completed, the developer loses interest in the code they wrote or, secondly, the developer makes repeated long-term use of the code they wrote.

Consider the first case. There's no support desk or customer helpline for this code. The developer has other tasks to do and possibly a full-time job to keep them busy. There are few employers who will pay employees to work full-time on something that doesn't generate an income or improve productivity. If your problem isn't quite identical to the developer's, that code might not work. It may need a few patches or adjustments. Since it's open source, if you understand the code you may be able to modify it yourself. If you don't understand it, you might be able to find someone else who can. This kind of tweaking and patching isn't possible with commercial software. However, it may be that you can't fix it yourself and you can't find anyone willing to help you. At which point you may have to look around for another alternative, which can be time-consuming.

Now consider the second case. Again, there's no support desk or customer helpline, but if the developer is regularly using this code in their

work then they have more interest in maintaining and enhancing it. While the code doesn't actually generate any income, it may improve productivity in which case an employer may be more amenable to the developer spending time on updating it. In this case you're in a better position if you use the code and find it doesn't quite work for you.

Although I said that in both cases there's no support desk or customer helpline, there's generally something better: a collection of enthusiastic users who are keen to help others in their spare time. These aren't people just reading stock answers from a crib sheet, they are people who regularly use the code and understand it (the user commands, at least, if not the underlying internals). In the TEX world, you can find them at places like TEX on StackExchange, The LATEX Community or similar forums or on a newsgroup such as `comp.text.tex` [16] or a mailing list (such as the TEX User Group (TUG) lists at `http://tug.org/mailman/listinfo`). These people are usually friendly, as long as you remember that they are volunteers not employees and that repeating the same old answers is tiresome.

Some of the topics covered in this book suggest a number of alternative classes or packages that you could use. When presented with a selection like this, how do you decide which of them fall under the first case and which the second case? The second case is clearly preferable as any issues are more likely to be fixed. You don't really want to invest time on learning how to use a package only to discover there are ancient unresolved bugs

in it.

The first thing to do is check the version number and release date. This information is listed on the Comprehensive TEX Archive Network (CTAN). For example, if you go to http://ctan.org/pkg/datatool you'll see the version number listed in the summary. The release date is in both the README file and at the start of the user guide. If the version number is low, such as 1.0, and the release date is some years ago then the package hasn't had much activity. This may not necessarily be a sign that the package falls under the first case as it may be small enough for bugs to be easily detected with a simple test file, but it may well be an indication that the author created it for a particular document, made it publicly available, but hasn't required it for any new documents. The other thing to look for is a "changes" or "history" file. A long revision list is usually an indication of the second case. (Don't mistake a long list of bug fixes as a bad thing. A package that has ten boolean options requires $2^{10} = 1024$ test files to check that all combinations work correctly, and that doesn't include the possibility of conflicts with other packages.)

The next thing to do is a quick search for the package on one of the sites mentioned above. For example, enter the package name in the search box for TEX on StackExchange or the LATEX Community Forum. Are there many results? If there aren't, that could mean there's not much interest in the package. (Unless, again, it's a very small package that's too trivial

to cause any issues.) If, on the other hand, the result list is long and most of the answers involve hacks using internal commands, then the package most likely falls under the first case. If the result list is long and most of the answers don't use internal commands or indicate that there was a bug that has now been fixed or a new feature has been added that solves the problem, then the package most likely falls under the second case. A little time spent on this type of investigation could save you a lot of grief later on.

Occasionally a package or class that has been under the second case for some years suddenly loses maintenance. This is most likely due to a change of circumstances for the developer. (None of us are immortal or immune to adversity or the ageing process!) Fortunately, in the case of some popular classes or packages other volunteers have agreed to take over maintenance. The open source nature of the code makes this more likely than it would with proprietary code.

1.2 Arara

Volume 2 [95, §1.1.2] introduced `arara`, a Java application that automates the process of building a LaTeX document. Since then, `arara` has evolved into version 4.0 that has useful new features, including conditionals that can be used to determine whether or not an application needs to be run.

To refresh your memory, or for those who haven't read Volume 2 [95, §1.1.2], you need to tell arara how to build your document by placing directives as comments within your source code.

Example:

↑ Input

```
% arara: pdflatex
\documentclass{article}

\begin{document}
\section{A Sample Section}
This is a sample document.
\end{document}
```

↓ Input

The first line of the source code has a directive:

```
% arara: pdflatex
```

Input

This indicates that arara needs to run pdflatex so if the document file is called, say, myDoc.tex, then the PDF file can be built using:

```
arara myDoc
```
Shell

However, it may be that you only want to run pdflatex if the source code has changed. With the new arara version 4.0, you can now add a conditional to the directive:

```
% arara: pdflatex if changed("tex")
```
Input

Now arara will only run pdflatex if the .tex file has changed. You can combine tests using || (boolean or) && (boolean and) or ! (negation). For example:

```
% arara: pdflatex if changed("tex") || missing("pdf")
```
Input

This will now run pdflatex if the .tex file has changed or if the PDF file is missing.

In addition to if (and the logically opposite unless) you can also have loops using while (repeat while the condition is true) or until (repeat until the condition is true).

Available tests that can be used in the conditions are listed below. The argument ⟨*file ref*⟩ indicates either an extension, such as "log", or a file reference, such as toFile("xampl.bib"). In the case of just a file extension, such as "log", the base is the same as the base of the source .tex

file that contains the arara directives. The argument ⟨*regex*⟩ indicates a regular expression.[1]

changed(⟨*file ref*⟩)	Returns true if the file has changed.
unchanged(⟨*file ref*⟩)	Returns true if the file hasn't changed.
exists(⟨*file ref*⟩)	Returns true if the file exists.
missing(⟨*file ref*⟩)	Returns true if the file doesn't exist.
found(⟨*file ref*⟩, ⟨*regex*⟩)	Returns true if the regular expression is found in the file.

Example:
With earlier versions of arara, if your document had cross-references you needed two `pdflatex` directives:

[1] See the Java `Pattern` class documentation for further details.

↑ Input

```
% arara: pdflatex
% arara: pdflatex
\documentclass{article}

\begin{document}
\section{A Sample Section}
Here is a cross-reference to
section~\ref{sec:another}.

\section{Another Section}
\label{sec:another}
\end{document}
```

↓ Input

However, if you make a change that doesn't affect the cross-references, you only need one invocation of `pdflatex`. The log file contains information if a rerun is needed:

```
LaTeX Warning: There were undefined references.

LaTeX Warning: Label(s) may have changed. Rerun to get
cross-references right.
```

So you could check for "undefined" like this:

```
% arara: pdflatex if changed("tex") || missing("pdf")
% arara: pdflatex if found("log", "undefined")
```

Input

Input

A more general purpose expression to search for is "Rerun", which may also be used by some packages if they detect a rerun is required that is unrelated to cross-references.

These tests can be combined in a loop. This can make for a long line in your source code, but luckily another new feature of arara version 4.0 is the ability to break the line using:

```
% arara: -->
```

Input

at the start of each continuation line. For example:

```
% arara: pdflatex while (changed("tex")
% arara: --> || missing("pdf")
% arara: --> || missing("log")
% arara: --> || found("log", "Rerun"))
```

Input

Input

Now arara will repeatedly run pdflatex while the .pdf or .log files are missing or while the .tex file has changed or while the log file contains "Rerun". To prevent an infinite process occurring, arara will break the loop if the condition is still true after ten iterations. (This maximum value can be changed, if required.)

The above example provides a single pdflatex directive in the source code, which is useful if no other applications are required, but this may need to be broken up into multiple directives if another application, such as bibtex, needs to be run.

Example:
This example has a citation from the sample xampl.bib database. (This sample bibliography file should be provided with all modern TeX distributions.)

↑ Input

```
% arara: pdflatex if changed("tex") || missing("pdf")
% arara: bibtex if (missing("bbl") || found("log", "Citation"))
% arara: pdflatex while (found("log", "Rerun"))
\documentclass{article}

\begin{document}
```

```
\section{A Sample Section}
Here is a cross-reference to
section~\ref{sec:another}.
A citation~\cite{book-minimal}.

\section{Another Section}
\label{sec:another}

\bibliographystyle{plain}
\bibliography{xampl}
\end{document}
```

Input

Now arara will only run bibtex if the .bbl file is missing or if the log file contains "Citation", which will pick up the undefined citation warning:

```
LaTeX Warning: Citation `book-minimal' on page 1 undefined
```

Another possible test would be for any changes to the .bib file. In this case, the .bib file doesn't have the same root as the main .tex file, so I can't just do changed("bib"). Instead I need to indicate that I'm specifying the actual file rather than the extension. This can be done using

```
changed(toFile("⟨filename⟩"))
```
Input

where ⟨*filename*⟩ is the name of the file.

In this case, I'm using the `xampl.bib` file, which is in my TEX Live distribution so I'd need the full path name, but if it was in the same directory as my `.tex` file I would just need to do:

```
changed(toFile("xampl.bib"))
```
Input

⚠ The boolean operators `||` and `&&` are short-circuited. This means that under certain circumstances, only the left-hand condition may be evaluated. For example, if you have ⟨*test1*⟩ `||` ⟨*test2*⟩ then if ⟨*test1*⟩ is true, there's no need to evaluate ⟨*test2*⟩ (since both the boolean operations ("true" or "false") and ("true" or "true") evaluate to "true"). Similarly, if you have ⟨*test1*⟩ `&&` ⟨*test2*⟩ then if ⟨*test1*⟩ is false, there's no need to evaluate ⟨*test2*⟩ (since both ("false" and "false") and ("false" and "true") evaluate to "false"). You may therefore need to consider the optimal ordering of any conditionals, such as placing `changed` or `unchanged` at the start of the condition. Alternatively, you can use the non-short-circuited operators `|` or `&`.

2. Managing Data

Many of the topics covered in this book have examples that fetch row-and-column style data from an external source, such as in a spreadsheet or structured query language (SQL) database. For example, in §3 Correspondence you may want to pull the recipient's details from a database or you may want to send a template letter to everyone listed in a spreadsheet (mail merging).

This chapter covers accessing data from a CSV file or spreadsheet or from a MySQL database (the most popular open source database [60]). The MySQL Community Edition is freely downloadable, but there are also commercial versions available. The free version can be fetched from `http://dev.mysql.com/downloads/mysql/`. In addition, this chapter also includes sections on topics that aren't specific to databases, but describe useful utility commands that are used in some of the examples and exercises throughout this book.

⚠ Some of the topics covered in this chapter are quite advanced. If you don't need to fetch data from an external source — for example if you want to write a letter to someone but intend to explicitly write the recipient's

address in your `.tex` file — then you can omit this chapter and skip ahead to §3 Correspondence.

There are a number of packages and applications available on CTAN that can be used to fetch data, see the data-import topic. This book covers the datatool bundle (available on modern TEX distributions such as TEX Live and MiKTEX[1]) and the `datatooltk` Java application (the installer and source are available on CTAN). This book assumes you have the latest version of datatool installed (version 2.23 at time of writing). If you want to try any of the `datatooltk` examples here, ensure you are using the latest version (currently 1.6).

If you are used to writing programming languages, take care with TEX. Although TEX is Turing-complete, it is very different to most programming languages. (In fact, it's a document formatting language rather than a programming language.) In, say, C or Java the source code is a series of assignments or function calls or commands. Data is explicitly assigned to variables. In TEX the source code consists of data interspersed with control sequences.[2] Whitespace forms part of the data (except at the start of lines or after control words). In most programming languages `foo( bar )` is

[1] TEX Live is included in the MacTeX distribution, and MiKTEX is included in the ProTeXt distribution.

[2] Although LATEX adds the restriction that document text is only permitted within the document environment.

equivalent to `foo(bar)` but in TEX (and LATEX) `\foo{ bar }` is usually not the same as `\foo{bar}`.[3] This is because the entire contents of the argument is data not a variable or list of variables. You may not like or agree with TEX's unusual syntax, but no one has so far produced a viable alternative to TEX that has the typesetting power and flexibility of TEX but the structured coding of popular programming languages (although LuaTEX comes close). *Unless told otherwise, assume that spaces aren't ignored.*

2.1 Utility Commands

This section describes utility commands that aren't specific to databases, but are useful when dealing with automation. §2.1.1 describes ways of defining new commands, §2.1.2 describes how to append or prepend code to existing commands and §2.1.3 describes commands that allow you to perform arithmetic calculations. There are some other utility commands concerned with iterating through lists, but these are described later in §2.7.

[3] Try doing `\begin{ document }` instead of `\begin{document}` in a LATEX document.

2.1.1 Macro Definitions

Volume 1 [92, §8] described how to define new commands using \newcommand and how to redefine them using \renewcommand. However there are times when these commands are too limited for the task in hand. For example, these commands only have a local effect, which means that if you use them to define a command within a group (see Volume 1 [92, §2.7]) they cease to be defined when the group ends. There are also cases where you might want to define a command without checking if it has already been defined. For example, suppose you want to define a "scratch" command that's used for temporary storage. If you use \renewcommand, then you must first initialise the scratch command with \newcommand, but if it turns out you never need to use the scratch command, then this definition is redundant and a waste of resources. In this case it's better to define the scratch command only when it's needed without the existence checks imposed by \newcommand and \renewcommand.

[FAQ: Optional arguments like \section]

⚠ Since many of the macros described in this section don't check for the prior existence of the command being defined, it's your responsibility to ensure you don't accidentally overwrite a core command which could mess up your document. Some class and package writers use a prefixing system for their command names to help prevent this.

If you're worried about accidentally overwriting a core command, you

can get TEX to tell you the definition of a command using:

\show ⟨*token*⟩ Definition

This will interrupt the document build (as though an error had been encountered) and display the definition of ⟨*token*⟩ in the transcript. This command can be used for testing purposes but it should be removed once it's provided you with the information you require. The token may be a control sequence or a character, including special characters.

Examples

1. Suppose you're thinking about defining a command called \c but you first want to find out if it already exists:

Input

```
\documentclass{article}

\show\c

\begin{document}

\end{document}
```

Input

When you run LaTeX on this document, the transcript will show:

```
> \c=macro:
->\OT1-cmd \c \OT1\c .
l.3 \show\c

?
```

This tells you that \c is a macro (first line of the above) and it then gives you the definition (second line of the above between -> and the terminating full stop). The rest of the message is just the regular error message prompt, including the line number. If you type h at the prompt for help, it will tell you that this isn't actually an error message.

2. Suppose you're now thinking about defining a command called \kern but you first want to find out if it already exists:

↑ Input

```
\documentclass{article}
```

```
\show\kern

\begin{document}

\end{document}
```

↓ Input

The transcript now shows:

```
> \kern=\kern.
l.3 \show\kern

?
```

This may seem a bit strange as it's not showing you a definition, but in this case (La)TEX is telling you that \kern is a primitive (a core command that's not defined in terms of any other command). If you try redefining this particular command, the results will be disastrous.

3. Suppose you're now thinking about defining a command called \tmp but you first want to find out if it already exists:

```
\documentclass{article}

\show\tmp

\begin{document}

\end{document}
```

Input

Input

The transcript now shows:

```
> \tmp=undefined.
l.3 \show\tmp

?
```

This means that \tmp is undefined.

There's a convenient Perl script called texdef that will display the definition of a command without the need for you to use \show in your document. Syntax:

```
texdef ⟨options⟩ ⟨cs name⟩
```
Shell

where ⟨*cs name*⟩ is the name of the command you want to check without the leading backslash. There are a number of options you can use. For example:

- `-t` ⟨*format*⟩

  This indicates the TEX format. For example, `-t latex` means you're checking a LATEX command. On some systems, there's a convenient shortcut script:

  ```
  latexdef ⟨options⟩ ⟨cs name⟩
  ```
  Shell

  which is equivalent to:

  ```
  texdef -t latex ⟨options⟩ ⟨cs name⟩
  ```
  Shell

- `-c` ⟨*cls name*⟩

  This indicates that you want to check if the command is defined in a document that uses the class ⟨*cls name*⟩.

- -p ⟨*sty name*⟩

  This indicates that you want to check if the command is defined in a document that uses the package ⟨*sty name*⟩.

For a complete list of options either use:

```
texdef --help
```
Shell

or

```
texdoc texdef
```
Shell

Example
Suppose you want to know the definition of the command \forlistloop provided by the etoolbox package:

```
texdef -t latex -p etoolbox forlistloop
```
Shell

This displays:

```
\forlistloop:
macro:#1#2->\expandafter \etb@forlistloop \expandafter {#2}{#1}
```

This means that \forlistloop is a macro that has two arguments (indicated by macro:#1#2) and the code after the arrow -> indicates the definition of \forlistloop.

What about environments? The start of an environment ⟨*envname*⟩, issued by the command \begin{⟨*envname*⟩}, performs some checks, starts a group, stores the environment name in \@currenvir and does \⟨*envname*⟩. The end of the environment \end{⟨*envname*⟩} checks that ⟨*envname*⟩ matches \@currenvir, does \end⟨*envname*⟩ if that command exists, and then closes the group. (This, incidentally, is why \newcommand won't allow you to define a command where the name starts with end as it may interfere with the environment end mechanism.)

Example

Suppose you want to know the definition of the figure environment:

```
texdef -t latex figure endfigure
```

Shell

This displays:

```
\figure:
\long macro:->\@float {figure}
```

```
\endfigure:
\long macro:->\end@float
```

Alternatively you can use the -E switch:

```
texdef -t latex -E figure
```

Shell

which produces the same output.

Recall from Volume 1 [92, §4.5.1] that the font declarations can also be used as environments. In this case, \end⟨*envname*⟩ doesn't exist so \end{⟨*envname*⟩} skips it.

Example

Suppose you want to know the definition of the Large environment:

```
texdef -t latex Large endLarge
```

Shell

This displays:

```
\Large:
\long macro:->\@setfontsize \Large \@xivpt {18}


\endLarge:
undefined
```

This shows the definition of \Large but shows that \endLarge is undefined, which means \Large can also be used as a declaration.

Now that you know how to check that your chosen macro name doesn't already exist, so you won't accidentally cause a perplexing catastrophic error in your document, the rest of this section will look at defining commands, while the next section (§2.1.2) will look at modifying existing commands.

\let⟨*new token*⟩⟨*token*⟩ Definition

This command makes ⟨*new token*⟩ have the same definition as ⟨*token*⟩.

Example:

Input

```
\let\myemph\emph
\myemph{Test}
```

Input

produces:

Test Output

Note that this isn't the same as defining \myemph to use \emph. If \emph is later redefined, \myemph isn't affected.

Example:

Input

```
\let\myemph\emph
\myemph{Test}.

\renewcommand{\emph}[1]{\textbf{#1}}
\myemph{Test}. \emph{Test}.
```

Input

produces:

Output

Test.
Test. **Test**.

Output

The command \let may be prefixed with \global to make the assignment have a global effect.

Example:

Input

```
\newcommand{\myemph}[1]{\texttt{#1}}
\myemph{Test}. \emph{Test}.

{% start a group
  \global\let\myemph\emph
  \myemph{Test}.

  \renewcommand{\emph}[1]{\textbf{#1}}%
  \myemph{Test}. \emph{Test}.
}% end the group

\myemph{Test}. \emph{Test}.
```

Input

produces:

Output

Test. *Test.*
Test.

Test. **Test.**
Test. *Test.*

↓ Output

In this case, \myemph has retained its new definition after the end of the group, but the original definition of \emph has been restored, because the effect of \renewcommand only lasted until the end of the group.

\providecommand{⟨*cs*⟩}[⟨*n-args*⟩][⟨*default*⟩]{⟨*definition*⟩} Definition

This is like \newcommand except that it will only define the command *if it doesn't already exist*. If ⟨*cs*⟩ already exists, no change will be made to it. The syntax is the same as for \newcommand.

Example:

↑ Input

```
\providecommand{\emph}[1]{\textbf{#1}}%
\emph{Test}.
```

↓ Input

This produces:

Test. Output

Since \emph already exists, \providecommand had no effect. Compare this to:

↑ Input

```
\providecommand{\Emph}[1]{\textbf{#1}}%
\Emph{Test}.
```

↓ Input

which produces:

Test. Output

In this case \Emph didn't exist, so it was okay for \providecommand to define it.

\def⟨*cs*⟩⟨*arg syntax*⟩{⟨*definition*⟩} Definition

This defines ⟨*cs*⟩ without checking if it already exists. The ⟨*definition*⟩ part is the same as that for \newcommand, but with \def you don't just specify the number of arguments. Instead you declare the argument syntax in ⟨*arg syntax*⟩ where each parameter is identified by #⟨*n*⟩ (where ⟨*n*⟩ is a number from 1 to 9).

Examples:

1. The simplest form is when you define a command that has no arguments. For example:

```
\def\test{This is a test.}% definition
\test
```

Input

Input

produces:

This is a test.

Output

2. This example defines a command with two arguments:

```
\def\test#1#2{\emph{#1} \textbf{#2}}%
\test{First}{Second}
```

Input

Input

This produces:

First **Second** Output

3. This example defines a rather more complicated command that has the syntax:

`\test` ⟨*arg1*⟩`-`⟨*arg2*⟩`-`⟨*arg3*⟩`\endtest`

This parameter syntax is now `#1-#2-#3\endtest` which is in the ⟨*arg syntax*⟩ part below:

```
\def\test#1-#2-#3\endtest{\emph{#1} \textbf{#2} \texttt{#3}}
```
Input

Note that this doesn't define a command called `\endtest`. The `\endtest` token merely forms part of the argument syntax. This `\test` command can now be used but only with the correct syntax:

```
\test First-Second-Third\endtest
```
Input

This produces:

First **Second** Third

Output

If you attempt to use incorrect syntax, for example, if you try

```
\test{First}{Second}{Third}
```

then you will get a runaway argument error:

```
Runaway argument?
{First}{Second}{Third}
! Paragraph ended before \test was complete.
```

Within ⟨*arg syntax*⟩ the parameters must appear in ascending order (for example, `#2#1` is invalid); parameters can't be repeated (for example, `#1#1` is invalid); intermediate parameters can't be skipped out (for example, `#1#3` is invalid). This is a restriction on ⟨*arg syntax*⟩ not on ⟨*definition*⟩.

`\def` only has a local effect, like `\newcommand`. If you want to globally define a command, then you can use the analogous:

`\gdef`⟨*cs*⟩⟨*arg syntax*⟩`{`⟨*definition*⟩`}`

Definition

There are two more similar commands:

\edef⟨*cs*⟩⟨*arg syntax*⟩{⟨*definition*⟩} Definition

and

\xdef⟨*cs*⟩⟨*arg syntax*⟩{⟨*definition*⟩} Definition

In both cases, the command being defined is set to the full expansion of ⟨*definition*⟩. The first, \edef, only has a local effect. The second, \xdef, has a global effect.

Example:
This example illustrates the difference between the non-expansion \gdef and the expansion \xdef:

↑ Input

```
\def\abc{abc}
\def\xyz{xyz}
{% start group
  \def\abc{ABC}
  \def\xyz{XYZ}
  \gdef\test{\abc; \xyz}
  \xdef\etest{\abc; \xyz}
  \test. \etest.
```

```
}%

\test. \etest.
```

↓ Input

This produces:

ABC; XYZ. ABC; XYZ.
abc; xyz. ABC; XYZ.

↑ Output

↓ Output

In this example, \test was just defined in terms of \abc and \xyz, so when \test is executed, it uses the current definitions of those commands. On the other hand, \etest was defined as the full expansion of \abc and \xyz. This means that if the definitions of \abc and \xyz later change, \etest is unaffected.

⚠ Take care when using \edef and \xdef that the definition doesn't contain fragile commands, as that can cause an error.

If you want to protect against fragile commands, LaTeX provides additional commands analogous to \edef and \xdef:

\protected@edef⟨*cs*⟩⟨*arg syntax*⟩{⟨*definition*⟩} Definition

and

\protected@xdef⟨*cs*⟩⟨*arg syntax*⟩{⟨*definition*⟩} Definition

Note that these are internal commands, so if you need to use them in your document (rather than in a class or package), you need \makeatletter and \makeatother.

[FAQ: \@ and @ in macro names]

Note that \def, \gdef, \edef and \xdef all define short commands, unless the prefix \long is used. For example:

```
\long\def\test#1{\emph{#1}}
```
Input

This now allows a paragraph break within the argument of \test. For further details see *The TEXbook* [45]. (Both \protected@edef and \protected@xdef also define short commands, but they can't be prefixed with \long.)

In addition to the TEX primitives and core LATEX commands described above, the etoolbox package [50] also provides ways of defining commands.

\csdef{⟨*cs name*⟩}⟨*arg syntax*⟩{⟨*definition*⟩} Definition

This is analogous to \def except that you supply the control sequence name ⟨*cs name*⟩ (without the leading backslash) instead of the actual control

sequence. Note that ⟨*cs name*⟩ must be fully expandable. The ⟨*arg syntax*⟩ is the same as for \def.

Defining a control sequence by its name in this manner means that you can have control sequences that include punctuation or digits or you can form the control sequence name on the fly. This is the way that commands with labels work. For example, recall the \newglossaryentry command from Volume 2 [95, §6.1.2]. The first argument is a label, and this label is used in the name of the internal control sequences that store the entry details. The datatool package's database commands described later in this chapter also use a similar technique.

The etoolbox package also provides:

\csgdef{⟨*cs name*⟩}⟨*arg syntax*⟩{⟨*definition*⟩} Definition

analogous to \gdef,

\csedef{⟨*cs name*⟩}⟨*arg syntax*⟩{⟨*definition*⟩} Definition

analogous to \edef,

\csxdef{⟨*cs name*⟩}⟨*arg syntax*⟩{⟨*definition*⟩} Definition

analogous to \xdef,

\protected@csedef{⟨cs name⟩}⟨arg syntax⟩{⟨definition⟩} Definition

analogous to \protected@edef, and

\protected@csxdef{⟨cs name⟩}⟨arg syntax⟩{⟨definition⟩} Definition

analogous to \protected@xdef.

There are also commands analogous to \let:

\cslet{⟨cs name⟩}{⟨cs⟩} Definition

\letcs{⟨cs⟩}{⟨cs name⟩} Definition

and

\csletcs{⟨cs name⟩}{⟨cs name⟩} Definition

where ⟨cs name⟩ is the name of a control sequence and ⟨cs⟩ is a control sequence.

You can use a control sequence by its name with:

\csuse{⟨cs name⟩} Definition

Remember that if the control sequence has been defined to have arguments, you need to specify these afterwards. For example:

```
\csuse{section}{A Sample Section}
```
Input

is equivalent to:

```
\section{A Sample Section}
```
Input

If the control sequence doesn't exist, \csuse{⟨*cs name*⟩} will expand to nothing. An alternative is to use TEX's primitive:

\csname ⟨*cs name*⟩\endcsname
Definition

If the control sequence doesn't exist, \csname will define the control sequence to \relax before using it.

The etoolbox package also provides ways of determining if a command already exists using:

\ifdef{⟨*cs*⟩}{⟨*true part*⟩}{⟨*false part*⟩}
Definition

where ⟨*cs*⟩ is the control sequence whose existence is being tested. Alternative you can specify the control sequence name using:

\ifcsdef{⟨*cs name*⟩}{⟨*true part*⟩}{⟨*false part*⟩}
Definition

In both cases, if the command has been defined, ⟨*true part*⟩ will be processed, otherwise ⟨*false part*⟩ will be processed. You can also use the logically opposite commands which check for non-existence:

\ifundef{⟨cs⟩}{⟨true part⟩}{⟨false part⟩} Definition

where ⟨cs⟩ is a control sequence and

\ifcsundef{⟨cs name⟩}{⟨true part⟩}{⟨false part⟩} Definition

where ⟨cs name⟩ is a control sequence name. Although logically opposite, these commands differ slightly from their existence counterparts as these commands will consider a control sequence undefined if its definition is \relax.

If you don't want to use the etoolbox package, the LaTeX kernel provides the internal commands:

\@ifundefined{⟨cs name⟩}{⟨true part⟩}{⟨false part⟩} Definition

which is similar to \ifcsundef,

\@nameuse{⟨cs name⟩} Definition

which is equivalent to

\csname ⟨cs name⟩\endcsname Input

and

\@namedef{⟨cs name⟩}⟨arg syntax⟩{⟨definition⟩} Definition

which is similar to \csdef.

2.1.2 Hook Management

The etoolbox package [50] comes with some useful hook management tools that allow you to append (or prepend) code to a control sequence's definition. To append ⟨code⟩ to the definition of the command ⟨cs⟩:

\appto⟨cs⟩{⟨code⟩} Definition

or to make the effect global:

\gappto⟨cs⟩{⟨code⟩} Definition

Example

Input

```
\newcommand{\mymacro}{x}%
\mymacro;
\appto\mymacro{AB}%
\mymacro;
```

```
\appto\mymacro{YZ}%
\mymacro.
```

↓ Input

This produces:

x; xAB; xABYZ. Output

Note that no expansion is performed on ⟨*code*⟩. For example:

↑ Input

```
\newcommand{\mymacro}{x}%
\newcommand{\myothermacro}{AB}%
\appto\mymacro{\myothermacro}
\renewcommand{\myothermacro}{YZ}%
\appto\mymacro{\myothermacro}
```

↓ Input

is equivalent to:

```
\newcommand{\mymacro}{x\myothermacro\myothermacro}
```

Input

When you actually use \mymacro it will use whatever the definition of \myothermacro is at the time, which may not be what you intended.

If you want ⟨*code*⟩ expanded before being appended to ⟨*cs*⟩ then you need to use:

\eappto⟨*cs*⟩{⟨*code*⟩} Definition

or its global equivalent:

\xappto⟨*cs*⟩{⟨*code*⟩} Definition

So

↑ Input

```
\newcommand{\mymacro}{x}%
\newcommand{\myothermacro}{AB}%
\eappto\mymacro{\myothermacro}%
\renewcommand{\myothermacro}{YZ}%
\eappto\mymacro{\myothermacro}
```

↓ Input

is equivalent to:

```
\newcommand{\mymacro}{xABYZ}
```

Input

When you use any of the versions that expand the item, if the item contains a command that shouldn't be expanded, you need to use:

\noexpand⟨*cs*⟩ Definition

where ⟨*cs*⟩ is the command that shouldn't be expanded. Fragile commands will typically need to have their expansion suppressed. For example:

↑ Input

```
\newcommand{\mymacro}{x}%
\newcommand{\myothermacro}{AB}%
\eappto\mymacro{\noexpand\footnote{\myothermacro}}
```

↓ Input

is equivalent to:

```
\newcommand{\mymacro}{x\footnote{AB}}
```

Input

Now the fragile \footnote command won't be processed until \mymacro is used later on in the document.

There are similar commands for prepending code to a command:

\preto⟨*cs*⟩{⟨*code*⟩} Definition

or its global equivalent:

\gpreto⟨cs⟩{⟨code⟩} Definition

As with \appto and \gappto, no expansion is performed on ⟨code⟩. If you want ⟨code⟩ expanded you need to use:

\epreto⟨cs⟩{⟨code⟩} Definition

or its global equivalent:

\xpreto⟨cs⟩{⟨code⟩} Definition

The expansion commands, such as \eappto and \epreto, *fully expand* ⟨code⟩. This means that a recursive expansion is performed until everything is expanded except commands that have been prefixed with \noexpand or commands that don't expand, such as primitives or robust commands. For example:

↑ Input

```
\newcommand{\mymacro}{x}%
\newcommand{\mymacroB}{\mymacroC}%
\newcommand{\mymacroC}{\mymacroD}%
\newcommand{\mymacroD}{Z}%
\eappto\mymacro{\mymacroB}
```

↓ Input

is equivalent to:

```
\newcommand{\mymacro}{xZ}
```
Input

It may be that you only want one-level of expansion. In which case you can use the etoolbox command:

\expandonce⟨*cs*⟩ Definition

For example:

```
\newcommand{\mymacro}{x}%
\newcommand{\mymacroB}{\mymacroC}%
\newcommand{\mymacroC}{\mymacroD}%
\newcommand{\mymacroD}{Z}%
\eappto\mymacro{\expandonce\mymacroB}
```
↑ Input

↓ Input

is equivalent to:

```
\newcommand{\mymacro}{x\mymacroC}
```
Input

There are also analogous commands that require the name (without the leading backslash) of the control sequence:

\csappto{⟨cs name⟩}{⟨code⟩} Definition

(appending without expansion)

\csgappto{⟨cs name⟩}{⟨code⟩} Definition

(global version)

\cseappto{⟨cs name⟩}{⟨code⟩} Definition

(appending with expansion)

\csxappto{⟨cs name⟩}{⟨code⟩} Definition

(global version)

\cspreto{⟨cs name⟩}{⟨code⟩} Definition

(prepending without expansion)

\csgpreto{⟨cs name⟩}{⟨code⟩} Definition

(global version)

\csepreto{⟨cs name⟩}{⟨code⟩} Definition

(prepending with expansion)

\csxpreto{⟨cs name⟩}{⟨code⟩} Definition

(global version).

2.1.3 Arithmetic

TEX count registers only allow integer values. Even the registers that store dimensions (such as \parindent) use integer arithmetic as TEX internally stores lengths in terms of its sp unit (65 536 sp = 1 pt). There are, however, some packages that allow you to perform decimal calculations using TEX (see, for example, the calculation topic or the arithmetic topic).

However, first let's look at just integer arithmetic, as that can be performed more efficiently using TEX primitive that using LATEX packages. TEX count registers are defined using:

\newcount⟨register cs⟩ Definition

This not only allocates a new register, but also assigns a control sequence ⟨register cs⟩ that can be used to reference the register. For example:

```
\newcount\mycount
```
Input

allocates a new register that can be referenced using \mycount.

⚠ Be careful not to confuse TEX's \newcount with LATEX's \newcounter command. With the LATEX version, you provide a label as the argument. Internally, \newcounter uses \newcount to create a register where the control sequence is formed from \c@⟨*label*⟩ (where ⟨*label*⟩ is the argument of \newcounter). However, \newcounter performs more than simply assigning a new register. It also provides a command that can be used to format the value of the register (\the⟨*label*⟩), allows the value to be cross-referenced using LATEX's \label/\ref (along with \refstepcounter), and can also automatically reset the value of the register when another counter is incremented. If I only want a scratch variable to calculate integer values, I'll just define a register using \newcount, as it helps to reduce the clutter of (possibly already complicated code) if I can refer to the register control sequence directly, without the cumbersome use of \value or internal commands. (See, for example, the \julianday register in Example 38.)

A register can be assigned a value using the syntax:

⟨*register*⟩=⟨*value*⟩ Definition

For example:

```
\mycount = 25
```
Input

sets the value of the \mycount register to 25. Note that it's sometimes nec-

essary to use \relax after the assignment to prevent TEX from prematurely expanding the following token. ε-TEX provides a primitive for evaluating expressions:

\numexpr⟨*integer expression*⟩ Definition

For example:

```
\mycount=\numexpr(25+5)/3
```
Input

It's usually a good idea to put \relax after ⟨*integer expression*⟩ in case it happens to be followed by something that could be interpreted as part of the expression:

↑ Input

```
\mycount=\numexpr(25+5)/3\relax
-- this assignment is followed by an en-dash.
```

↓ Input

There is a similar ε-TEX primitive for evaluating dimension expressions:

\dimexpr⟨*dimension expression*⟩ Definition

You can display the value of the register using:

\the⟨*register*⟩ Definition

For example:

```
\mycount = 25\relax
\the\mycount
```

Input

Input

produces:

25 Output

This also works with other types of registers. For example:

```
\the\textwidth
```

Input

produces:

271.30533pt Output

There's a similar TEX primitive that works on either a register or a number (either typed explicitly or stored in a macro):

\number⟨*num*⟩ Definition

For example:

↑ Input

```
\mycount = 25
\newcommand{\mynum}{40}%
Register: \number\mycount.
Macro: \number\mynum.
Number: \number46.
```

↓ Input

Produces:

Register: 25. Macro: 40. Number: 46. Output

The contents of a register can be incremented using:

\advance⟨*register*⟩ by ⟨*value*⟩ Definition

where ⟨*value*⟩ may be another register, a macro that expands to a value or a plain number. The by keyword is optional. For example:

```
\mycount = 12
\advance\mycount by -3
Result: \number\mycount.
```

Input

Input

produces:

Result: 9.

Output

The contents of a register can be multiplied by a value using:

\multiply⟨*register*⟩ by ⟨*value*⟩

Definition

As with \advance, the keyword by may be omitted. For example:

```
\mycount = 12
\multiply\mycount by 2
Result: \number\mycount.
```

Input

Input

produces:

Result: 24. Output

The contents of a register can be divided by a value using:

\divide⟨*register*⟩ by ⟨*value*⟩ Definition

As before the by keyword may be omitted. Remember that this uses integer arithmetic. For example:

↑ Input

```
\mycount = 25
\divide\mycount by 3
Result: \number\mycount.
```

↓ Input

produces:

Result: 8. Output

For decimal arithmetic, there are a number of packages available, such as fp [56] and pgfmath [101]. In addition, the LaTeX3 experimental bundle [49] also provides decimal arithmetic. However, it uses LaTeX3 syntax, which is beyond the scope of this book. [FAQ: LaTeX3 programming]

The datatool package provides an interface to either fp or pgfmath. The default is to use fp but you can change this using the `math=pgfmath` package option:

```
\usepackage[math=pgfmath]{datatool}
```
Input

The pgfmath package is part of the pgf bundle, so if you intend loading pgf (or tikz) in your document, it's more efficient to use the pgfmath engine with datatool to avoid the overhead of loading an additional package.

The fp and pgfmath packages use very different syntax to perform the same calculations, but datatool provides the same interface commands regardless of the underlying arithmetical package, so you can switch engines without having to change your code, however you may find minor differences in the results, caused by different levels of precision or rounding.

There are two types of arithmetical commands provided by datatool: those that operate on raw plain numbers that use a full stop as the decimal point with no group separator, and those that operate on locale dependent numbers or currency. The first type are prefixed by `dtl`. For example:

```
\dtlround{⟨cs⟩}{⟨number⟩}{⟨num digits⟩}
```
Definition

This rounds ⟨*number*⟩ to ⟨*num digits*⟩ decimal places and stores the result in the control sequence ⟨*cs*⟩.

The second type are prefixed by `DTL`. For example:

\DTLround{⟨cs⟩}{⟨number⟩}{⟨num digits⟩} Definition

which is the locale-dependent alternative to \dtlround.

The plain versions (prefixed with dtl) all perform local assignments. If you need a global assignment you can use \global\let on the result (see §2.1.1). For example:

↑ Input

```
{% local scope
  \dtlround\mynum{14.39999}{2}% perform rounding
  \global\let\mynum\mynum
}
\mynum
```

↓ Input

The locale versions (prefixed with DTL) come with two alternatives: a local version and a global version. The global versions have the prefix DTLg such as:

\DTLground{⟨cs⟩}{⟨number⟩}{⟨num digits⟩} Definition

which is the global alternative to \DTLround.

For the locale versions, the decimal character defaults to a full stop and the number group separator defaults to a comma, but these can be changed using:

\DTLsetnumberchars{⟨*number group char*⟩}{⟨*decimal char*⟩} Definition

where ⟨*number group char*⟩ is the number group separator and ⟨*decimal char*⟩ is the decimal character. For example, to switch to commas for the decimal character and a full stop for the number group separator:

```
\DTLsetnumberchars{.}{,}
```
Input

Now, any numbers that use this format need to use the control sequences prefixed with `DTL` instead of `dtl`:

↑ Input

```
\DTLsetnumberchars{.}{,}
\DTLround\mynum{1.250,2398}{2}\mynum.
```

↓ Input

This produces:

1.250,24. Output

Example (plain numbers):

```
\dtlround{\mynum}{14.399999999999999}{2}\mynum;
\dtlround{\mynum}{2.5}{2}\mynum.
```

Input

Input

produces:

14.40; 2.50.

Output

Example (locale numbers):

```
\DTLround{\mynum}{2,014.399999999999999}{2}\mynum;
\DTLround{\mynum}{1,002.5}{2}\mynum.
```

Input

Input

produces:

2,014.4; 1,002.5. Output

The locale version \DTLround first converts the formatted number into a plain number, performs the arithmetical operation, and then converts the result back into a formatted number. Therefore, if your numbers are all plain numbers, it's more efficient to use \dtlround instead of \DTLround. Similarly for all the commands described below.

Addition can be performed using:

\dtladd{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the plain version, or

\DTLadd{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the scoped locale version, or

\DTLgadd{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the global locale version. In each case the sum of ⟨number 1⟩ and ⟨number 2⟩ is stored in the control sequence ⟨cs⟩.

Subtraction can be performed using:

\dtlsub{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the plain version, or

\DTLsub{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the scoped locale version, or

\DTLgsub{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the global locale version. In each case ⟨*number 1*⟩ minus ⟨*number 2*⟩ is stored in the control sequence ⟨*cs*⟩.

Multiplication can be performed using:

\dtlmul{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the plain version, or

\DTLmul{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the scoped locale version, or

\DTLgmul{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the global locale version. In each case the product of ⟨*number 1*⟩ and ⟨*number 2*⟩ is stored in the control sequence ⟨*cs*⟩.

Division can be performed using:

\dtldiv{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the plain version, or

\DTLdiv{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the scoped locale version, or

\DTLgdiv{⟨cs⟩}{⟨number 1⟩}{⟨number 2⟩} Definition

for the global locale version. In each case ⟨*number 1*⟩ divided by ⟨*number 2*⟩ is stored in the control sequence ⟨*cs*⟩.

The absolute value can be obtained using:

\dtlabs{⟨cs⟩}{⟨number⟩} Definition

for the plain version, or

\DTLabs{⟨cs⟩}{⟨number⟩} Definition

for the scoped locale version, or

\DTLgabs{⟨cs⟩}{⟨number⟩} Definition

for the global locale version. In each case the absolute value of ⟨*number*⟩ is stored in the control sequence ⟨*cs*⟩.

The negation can be obtained using:

\dtlneg{⟨cs⟩}{⟨number⟩} Definition

for the plain version, or

\DTLneg{⟨cs⟩}{⟨number⟩} Definition

for the scoped locale version, or

\DTLgneg{⟨cs⟩}{⟨number⟩} Definition

for the global locale version. In each case the negative of ⟨number⟩ is stored in the control sequence ⟨cs⟩.

There are also commands available to perform arithmetic operations on a column of data stored in one of datatool's internal database. However, these use the DTL versions and since TEX isn't designed for data management, it's better to perform these calculations in your spreadsheet or when you pull the data from a SQL database.

2.2 Loading Data

Before you can use data from an external source in your document, you must first load it. This section describes how to load data from a CSV file or from a datatool (.dbtex) file. When the data is loaded, it's stored in an

internal datatool database with an associated label. This label is then used to identify the internal database whenever you want to fetch values from it in your document. Throughout the rest of this book, ⟨*db-name*⟩ will be used to indicate this label. It's best to just use the letters `a`, ..., `z`, `A`, ..., `Z` or the digits `0`, ..., `9` within ⟨*db-name*⟩ to avoid accidentally using problematic special characters.

The database is divided up into columns (or fields) and rows where a column can be referenced either by its index (starting from 1) or by its label (or key) and a row can be referenced by its index (starting from 1). Throughout the rest of this book, ⟨*row-idx*⟩ will be used to indicate the row index, ⟨*col-idx*⟩ will be used to indicate the column index and ⟨*col-label*⟩ will be used to indicate the column label. As with ⟨*db-name*⟩, ⟨*col-label*⟩ should not contain any special characters. Since the label is often used in a list context, it's also best to avoid commas.

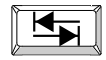
2.2.1 Loading Data From a CSV File

Most spreadsheet applications can export data to a CSV file. By default datatool assumes that the data in this file is separated by commas where the values are optionally delimited with the double-quote character " but if this isn't the case you need to specify the separator using:

\DTLsetseparator{⟨character⟩} Definition

and the delimiter using:

\DTLsetdelimiter{⟨character⟩} Definition

A common alternative is to use the tab character  as a separator but this is awkward to specify in LaTeX, so datatool provides

\DTLsettabseparator Definition

which is the same as \DTLsetseparator{⟨character⟩} where ⟨character⟩ is a tab. Note that this command changes the category code of the tab character. If you later want to treat a tab as a regular space, you can reset the category code after you have loaded the data using:

\DTLmaketabspace Definition

Remember to specify the separator and delimiter characters *before* you load the file.

For example, suppose your data is saved in a CSV file in the form:

```
Surname;First Name;Title;Registration Number
|Smith, Jr|;John;Mr;12345
Brown;Jane;Miss;12346
Brown;Andy;12347
```

```
Adams;|Z\"oe|
```

then in your document, before you load this file, you need to write:

```
\DTLsetseparator{;}\DTLsetdelimiter{|}
```

Input

For convenience, all the examples in this book assume the default comma separator and double-quote delimiter.

⚠ When creating your CSV files be careful of spurious spaces. For example, the following line of data:

```
Brown , Jane , Miss , 12356
```

✘

isn't the same as:

```
Brown,Jane,Miss,12356
```

✔

If the CSV file contains extended characters, make sure the file was saved with the same encoding as your LaTeX document and use the inputenc [39] and fontenc [58] packages.[4] (See Volume 1 [92, §4.3.1].) The sample files that accompany this book all use UTF-8 encoding. *Make sure you load the inputenc package with the `utf8` option before you load any of these CSV files.*

[4] Just use fontspec [75] for XeLaTeX.

Once your data is in a CSV file you can load it into a datatool database using:

\DTLloaddb[⟨*options*⟩]{⟨*db-name*⟩}{⟨*filename*⟩} Definition

where the CSV file is called ⟨*filename*⟩. The argument ⟨*db-name*⟩ is the database label, as described above.

The \DTLloaddb command assumes the CSV file either doesn't contain any of TEX's special characters (see Volume 1 [92, §4.3]) or, if it does, they form correct LATEX code. If this isn't the case, instead of using \DTLloaddb, you can use:

\DTLloadrawdb[⟨*options*⟩]{⟨*db-name*⟩}{⟨*filename*⟩} Definition

This is like \DTLloaddb except that it performs a substitution on nine of the ten special characters. (The backslash always retains its special state.) The mappings are listed in Table 2.1.

You can add extra mappings using:

\DTLrawmap{⟨*string*⟩}{⟨*replacement*⟩} Definition

For example, to replace the character £ with \pounds:

```
\DTLrawmap{£}{\pounds}
```
Input

(Alternatively use the inputenc package.)

Table 2.1 Special character mappings used by \DTLloadrawdb

Character	Mapping
%	\%
$	\$
&	\&
#	\#
_	_
{	\{
}	\}
~	\textasciitilde
^	\textasciicircum

Examples

1. Suppose your CSV file looks like:

```
Experiment,Result
1,$42.08\pm 0.1$
2,$48.03\pm 0.2$
```

In this file, the $ character has been used to indicate in-line maths mode (see Volume 1 [92, §9.1]). This should be left as it is when the data is loaded, so use \DTLloaddb.

2. Suppose your CSV file looks like:

```
Title,Price
"Duck & Goose's Adventures",$10.00
"The Return of Duck & Goose",$11.00
```

Now the characters & and $ are intended literally and should not be interpreted by TEX, so you need to use \DTLloadrawdb to ensure they are converted to the correct commands.

3. Now suppose your CSV file looks like:

```
Title,Price
"Duck & Goose's Adventures",£10.00
"The Return of Duck & Goose",£11.00
```

Again you need to use \DTLloadrawdb but before you do that you may need to define a new mapping for the £ character using \DTLrawmap, as described above. (Or use the inputenc package.)

⚠ The commands \DTLloaddb and \DTLloadrawdb can't read data where there are EOL characters within a cell. For example, neither command can read a CSV file that looks like:

```
Title,Price
"Duck and Goose's
Adventures",10.00
"The Return of
Duck and Goose",11.00
```

If you have a CSV file in this form, you can use datatooltk to convert the CSV file to a datatool (.dbtex) file, described in the next section.

You may have noticed that both \DTLloaddb and \DTLloadrawdb have an optional argument. This is a key=value list. Available keys are as follows:

noheader This is a boolean key. The value can be either false (the CSV file has a header row, as in the examples above) or true (the

CSV file doesn't have a header row). The default is `false`. If you want to set this value you may omit `=true`, so `noheader=true` is the same as just `noheader`.

`keys` Each column must have a unique label assigned to it. This makes it easier to reference but, like the database label, the column label mustn't contain special characters. The default action of `\DTLloaddb` and `\DTLloadrawdb` is to use the value given in the header row as a label. This may be inappropriate, so you can set a different set of labels using this option. The value must be a comma-separated list of labels in the same order as the columns in the CSV file. For example,

```
\DTLloaddb[keys={title,price}]{books}{booklist.csv}
```
Input

(note the braces).

You should use this option if the header row contains special characters. If the CSV file has no header and no label has been specified, a default label is generated in the form

`\dtldefaultkey` ⟨*n*⟩ Definition

where ⟨*n*⟩ is the column number. By default, `\dtldefaultkey` is just "Column", but you can change this by redefining the command (see Volume 1 [92, §8.2]). Note that an empty item in the `keys` list indicates an empty label for that column.

`autokeys` This is a boolean key that was introduced in version 2.22. If true, all the column labels will automatically be assigned the default label `\dtldefaultkey` ⟨*n*⟩ described above. This is useful if you have a lot of columns where the header may contain special characters, and you don't want to have to list every column in the `keys` list. This means that you need to know the column index if you want to reference the data in it.

`headers` Each column not only has a unique label assigned to it, but also has a header or title. The column headers are used in commands such as `\DTLdisplaydb` described in §2.6. The default column headers are taken from the header row in the CSV file but if they aren't appropriate or your file doesn't have a header row, you can use this option to assign headers. As with the `keys` option, described above, the value must be a comma-separated list of header text in the same order as the columns in the CSV file. For example:

```
\DTLloaddb[headers={Book Title,Price (\pounds)}]
 {books}% database label
 {booklist.csv}% filename
```

Input

(note the braces). An empty item in the `headers` list indicates an empty header for that column. For example:

```
\DTLloaddb[headers={Book Title,}]{books}{booklist.csv}
```

Input

indicates that the second column has a blank header.

`omitlines` The value must be a non-negative number indicating how many lines to skip at the start of the CSV file. For example, if the CSV file contains two lines of unwanted material at the start, then you need to use `omitlines=2`.

Examples

1. Suppose your CSV file called `products.csv` looks like:

```
This is a list of products in my shop.
Last edited 2014-01-22

Title,Price (£)
"Duck & Goose's Adventures",10.00
"The Return of Duck & Goose",11.00
```

When you load this data into your document, you need to skip the first three lines as they don't form part of the data. You also need to map the pound symbol (£) and the ampersand (&). Additionally, it's a good idea to provide short unique labels to identify the columns:

↑ Input

```
\DTLrawmap{£}{\pounds}% add mapping for £ symbol
\DTLloadrawdb
 [%
   omitlines=3,% header row is on line 4
   keys={title,price}% column labels
```

```
 ]%
 {products}% database label
 {products.csv}% filename
```

Input

2. Suppose your CSV file called products.csv looks like:

```
"Duck and Goose's Adventures",10.00
"The Return of Duck and Goose",11.00
```

Here there isn't a header row. You could assign labels as in the previous example:

Input

```
\DTLloaddb
 [%
  noheader,% no header row in file
  keys={title,price},% column labels
  headers={Title,Price (\pounds)}% column titles
 ]%
 {products}% database label
 {products.csv}% filename
```

Input

However, if you're not interested in referencing any columns (for example, you just want to display the data in a table, as described in §2.6) you can let datatool assign default labels:

Input

```
\DTLloaddb
 [%
   noheader,% no header row in file
   headers={Title,Price (\pounds)}% column titles
 ]%
 {products}% database label
 {products.csv}% filename
```

Input

2.2.2 Loading Data From a .dbtex File

A datatool (.dbtex) file is a LaTeX file with definitions and assignments of macros and registers used to represent an internal database. Although

this is a plain text file, it's very difficult to read and edit. However it's the fastest way of loading data via datatool. The datatooltk application can read and write this file, but it can also import data from CSV files, Excel .xls files, Open Document .ods spreadsheets or MySQL databases. (You can't export back to those formats.) It can be run either in batch mode from a command prompt (see Volume 1 [92, §2.5]) or as a graphical user interface.

It's simple to load a .dbtex file into a document that uses the datatool package. Just use:

\input{⟨*filename*⟩} Definition

where ⟨*filename*⟩ is the name of the file including the .dbtex extension. (This must come after \usepackage{datatool}.) If you can't remember the label you assigned to the data, you can reference it using:

\dtllastloadeddb Definition

However, a more convenient approach is to use:

\DTLloaddbtex{⟨*cs*⟩}{⟨*filename*⟩} Definition

This inputs ⟨*filename*⟩ and makes the control sequence ⟨*cs*⟩ have the same value as \dtllastloadeddb. For example

```
\DTLloaddbtex{\people}{people.dbtex}
```
Input

works like:

```
\input{people.dbtex}
\let\people\dtllastloadeddb
```
Input

Input

but it checks for the existence of the file people.dbtex and checks that the new command (\people) isn't already defined. Using \DTLloaddbtex is therefore safer than just using \input and \let. When you use commands like \DTLdisplaydb (§2.6) and \DTLforeach (§2.7.1) you can now reference the data using your new command (\people in this example).

The column title can be changed using:

\DTLsetheader{⟨*db-name*⟩}{⟨*col-label*⟩}{⟨*title*⟩}
Definition

where ⟨*db-name*⟩ is the label identifying the data (\people in the above example), ⟨*col-label*⟩ is the label identifying the required column and ⟨*title*⟩ is the new column title.

Example 1. Convert a .csv File to a .dbtex File

⚠ The default CSV escape character is a backslash, which means that if your data contains any control sequences the backslash will need to be doubled. For example, \\pounds instead of \pounds. If this causes a problem, you can change the CSV separator to a different character using datatooltk's application settings. (This doesn't apply to any of the sample files provided with this book as none of them contain any LaTeX commands.)

On page 76 I mentioned that \DTLloaddb and \DTLloadrawdb can't load CSV files where there is an EOL within a cell, but datatooltk can. Let's suppose the file books-multiline.csv contains the following:

```
Title,Price
"Duck and Goose's
Adventures",10.00
"The Return of
Duck and Goose",11.00
```

This can be converted into a datatool (.dbtex) file using one of the following methods:

1. Run datatooltk in batch mode from a command prompt:

```
datatooltk --csv books-multiline.csv
--output books.dbtex
```
Shell

The database label by default is taken from the CSV filename. If you want to change it you can use the `--name` ⟨*db-name*⟩ option:

```
datatooltk --name products2014
--csv books-multiline.csv --output books.dbtex
```
Shell

If you want to convert special characters to commands (using the mapping given in Table 2.1) you need to use the `--map-tex-specials` option:

```
datatooltk --name products2014
--csv books-multiline.csv
--output books.dbtex --map-tex-specials
```
Shell

2. Run `datatooltk` in graphical user interface (GUI) mode. If you have installed `datatooltk` on Windows there should be an entry

in your Start menu that will do this. Otherwise you can run the command `datatooltk-gui`. This should display the window shown in Figure 2.1.

If your CSV file contains special characters that you want converted to LaTeX commands, you need to switch on the map TeX characters via the "TeX" tab in the "Preferences" dialog. (This can be opened using the menu item Edit→Edit Preferences.)

Next use the File→Import→Import CSV menu item. This will open the dialog box shown in Figure 2.2. Select the CSV file and click on "Import". The data should now be visible in the main window as shown in Figure 2.3.

The tab above the data shows the database label (the same as the first mandatory argument of `\DTLloaddb` and `\DTLloadrawdb`). If you want to change the default, you can double-click on the tab which will open a dialog box where you can type in a new label. You can now save the data to a datatool (`.dbtex`) file using the File→Save As menu item.

If you want to import an Excel `.xls` file to a datatool (`.dbtex`) file, you can use the `--xls` and `--sheet` options to `datatooltk`. The argument

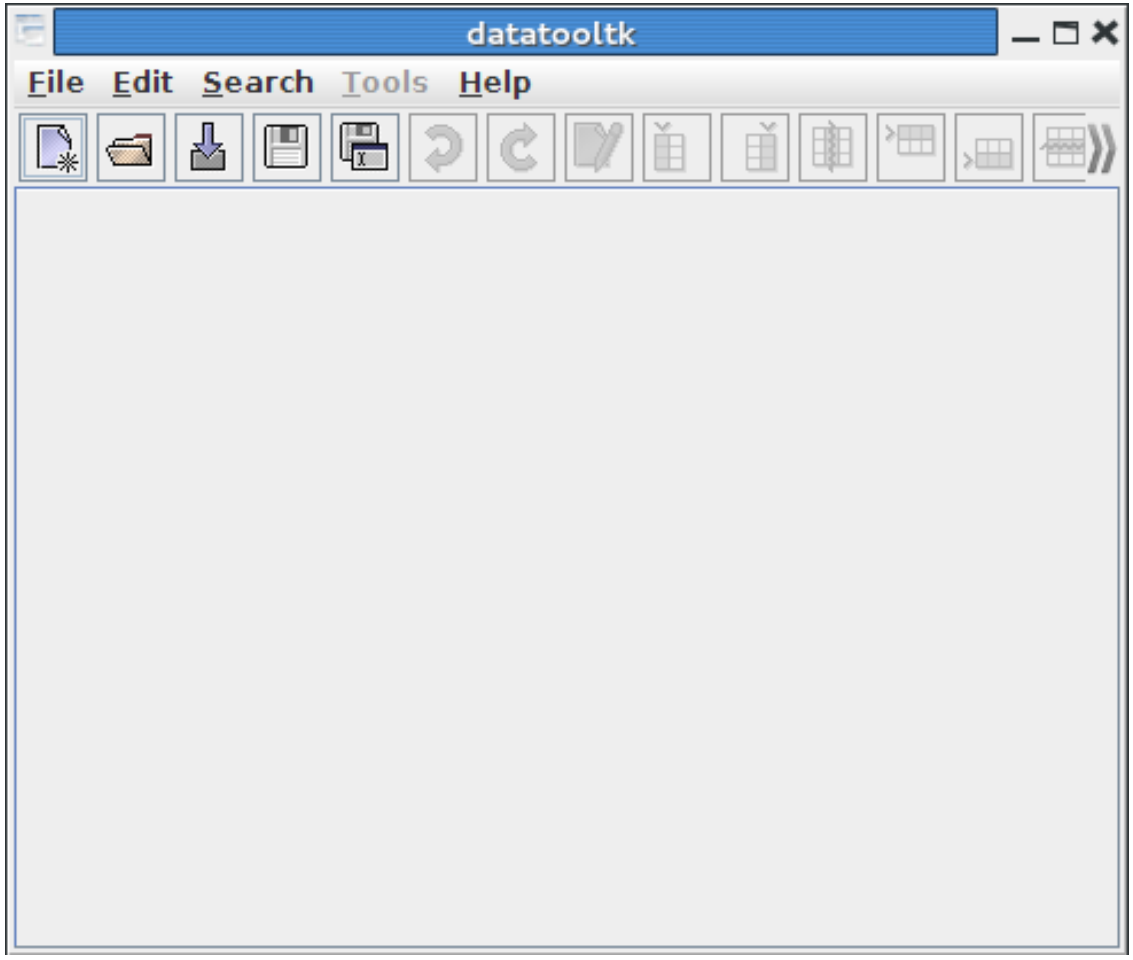


Figure 2.1 `datatooltk` in Graphical Mode

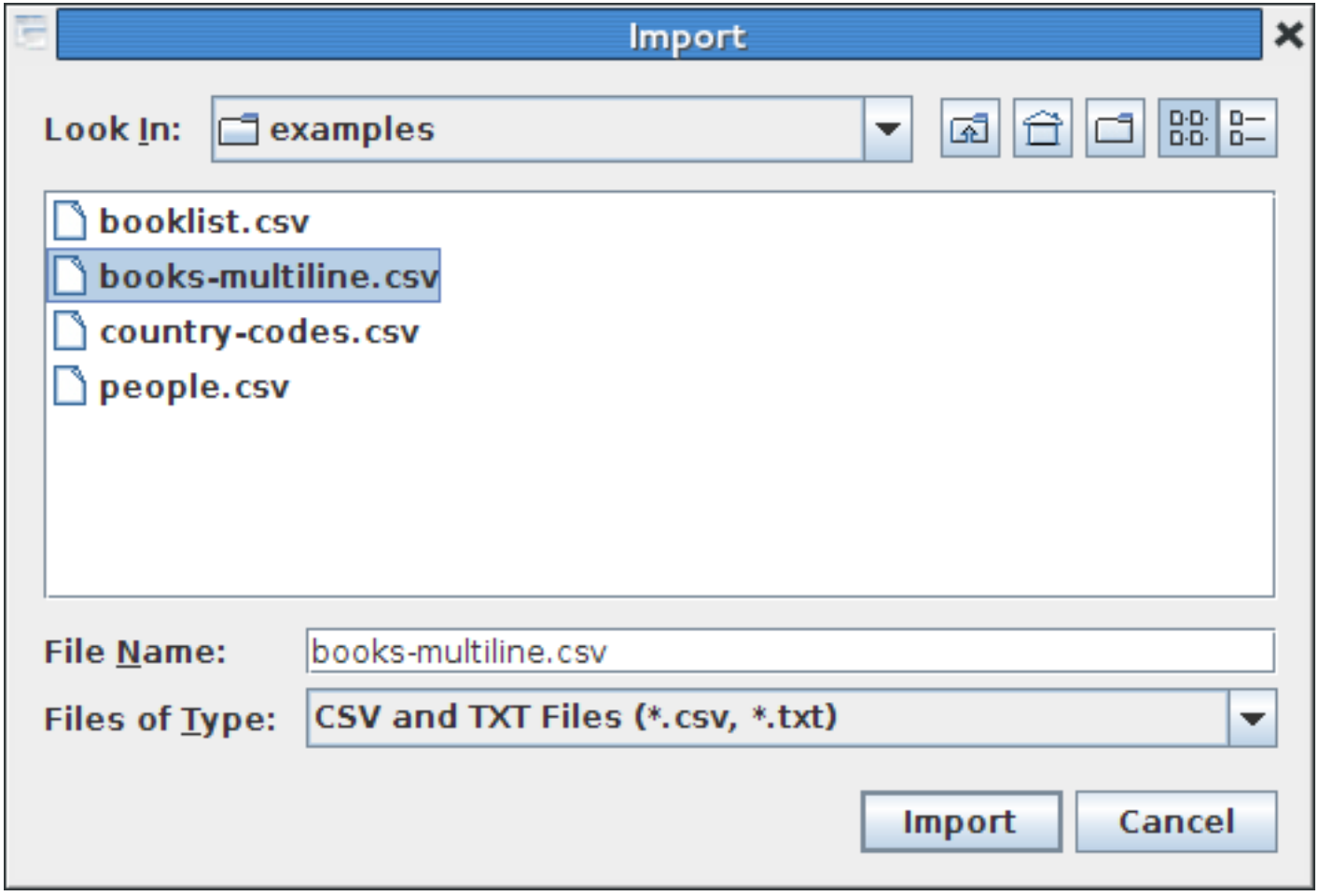


Figure 2.2 Import CSV File

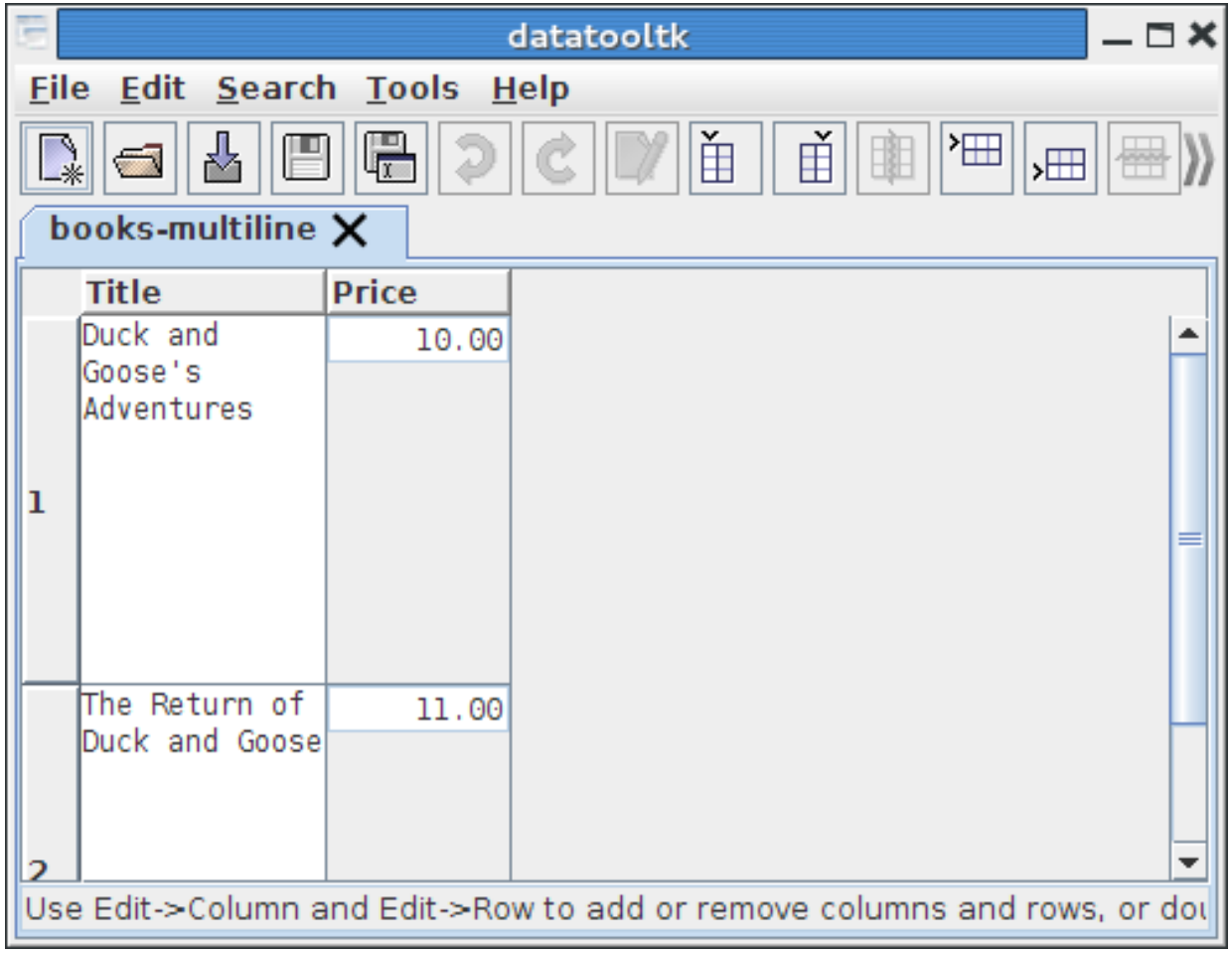


Figure 2.3 Imported CSV Data Shown in Main Window

of --xls is the name of the .xls file and the argument of --sheet is either an integer (starting from 0) indicating the sheet index or a string identifying the sheet label. Note that no formatting information is read from the Excel file. Any font changes or alignments should be made in your LaTeX document. Similarly, you can import an Open Document .ods spreadsheet using the --ods and --sheet options.

Example 2. Convert an .xls Sheet to a .dbtex File
Suppose you have an Excel file called, say, shop.xls and it contains two sheets: "products" and "customers". You can convert, say, the "customers" sheet to a file called customers.dbtex using one of the following methods:

1. Run datatooltk in batch mode from a command prompt:

```
datatooltk --output customers.dbtex --xls shop.xls
--sheet customers
```

Shell

Or:

```
datatooltk --output customers.dbtex --xls shop.xls
--sheet 1
```

Shell

(The customer data is in the second sheet, but indices start from 0 so it's sheet 1.)

2. Run datatooltk in GUI mode using datatooltk-gui, as described above, and set the map TEX characters option and header row settings if required.

   Next use the File→Import→Import Spreadsheet menu item. This will open a file selector dialog box. Select the .xls file (shop.xls in this example) and click on the "Import" button. Another dialog box will appear in which you need to select the required sheet name ("customers" in this example).

   Click on "Okay" and the data will be displayed in the main window. You can then save the data to a .dbtex file using the File→Save As menu item.

Note that formatting information isn't fetched from the spreadsheet. You will have to add any necessary font changing commands when you display the data. This helps to provide a consistent style throughout the document.

You can also use datatooltk to import data from a MySQL database. This is slightly more complicated as you need to tell datatooltk the data-

base, the SQL SELECT statement[5], the user name and password. (You may also need to change the SQL port and host, if different from the defaults.)

Example 3. Importing SQL Data to a .dbtex File
Suppose you have a database called `samples` and in that database you have a table called `books` and let's suppose the user name for the `samples` database is "sampleuser". Then you can import the data into a datatool (`.dbtex`) file using one of the following methods:

1. Run `datatooltk` in batch mode from a command prompt:

```
datatooltk --output books.dbtex --sqluser sampleuser
--sqldb samples --sql "SELECT * FROM books"
```

Shell

   This will prompt you for a password from the console. As before, you can use the `--map-tex-specials` option if required.

2. Run `datatooltk` in GUI mode using `datatooltk-gui`, as described above, and set the map TEX characters option if required.

[5] SQL statements are beyond the scope of this guide, but for further details I recommend you read "Managing & Using MySQL" [70].

Next use the File→Import→Import SQL menu item. This will open the "Import SQL" dialog box. Edit the SELECT statement as required and enter the database and user name in the appropriate fields. For example, as shown in Figure 2.4.

Then click on "Okay" and enter the password when prompted.

The data should now be visible in the main window as illustrated in Figure 2.5.

3. If you want to change the column header details, double-click on the button at the top of the required column. This will display the dialog box shown in Figure 2.6.

4. Once the data has been fetched, you can save it using File→Save As.

Remember that you can use the SELECT statement to filter unwanted rows, sort data or join the data with data from other tables.

If you use arara to build your document, you can use the datatooltk directive, but you must make sure you have at least v4.0 of arara installed.

Examples Using arara

1. Fetch the data from a CSV file called booklist.csv:

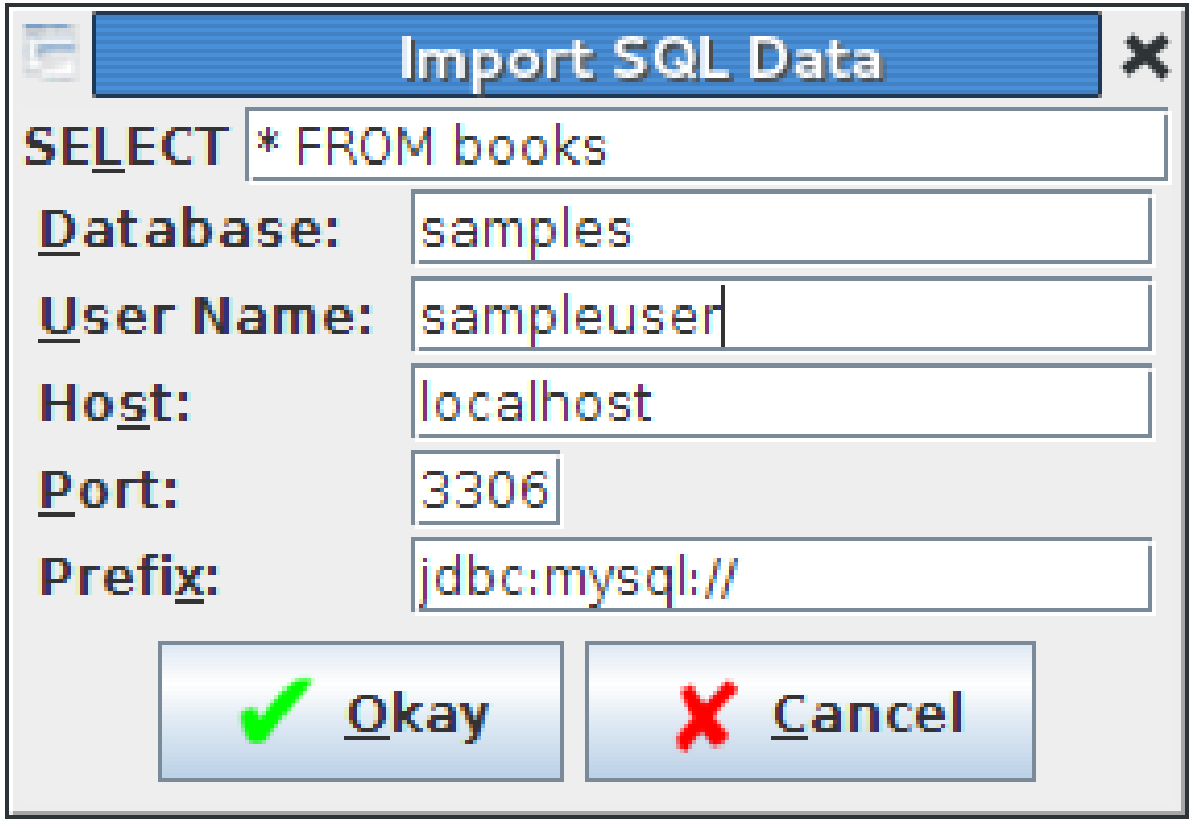


Figure 2.4 Import SQL Data

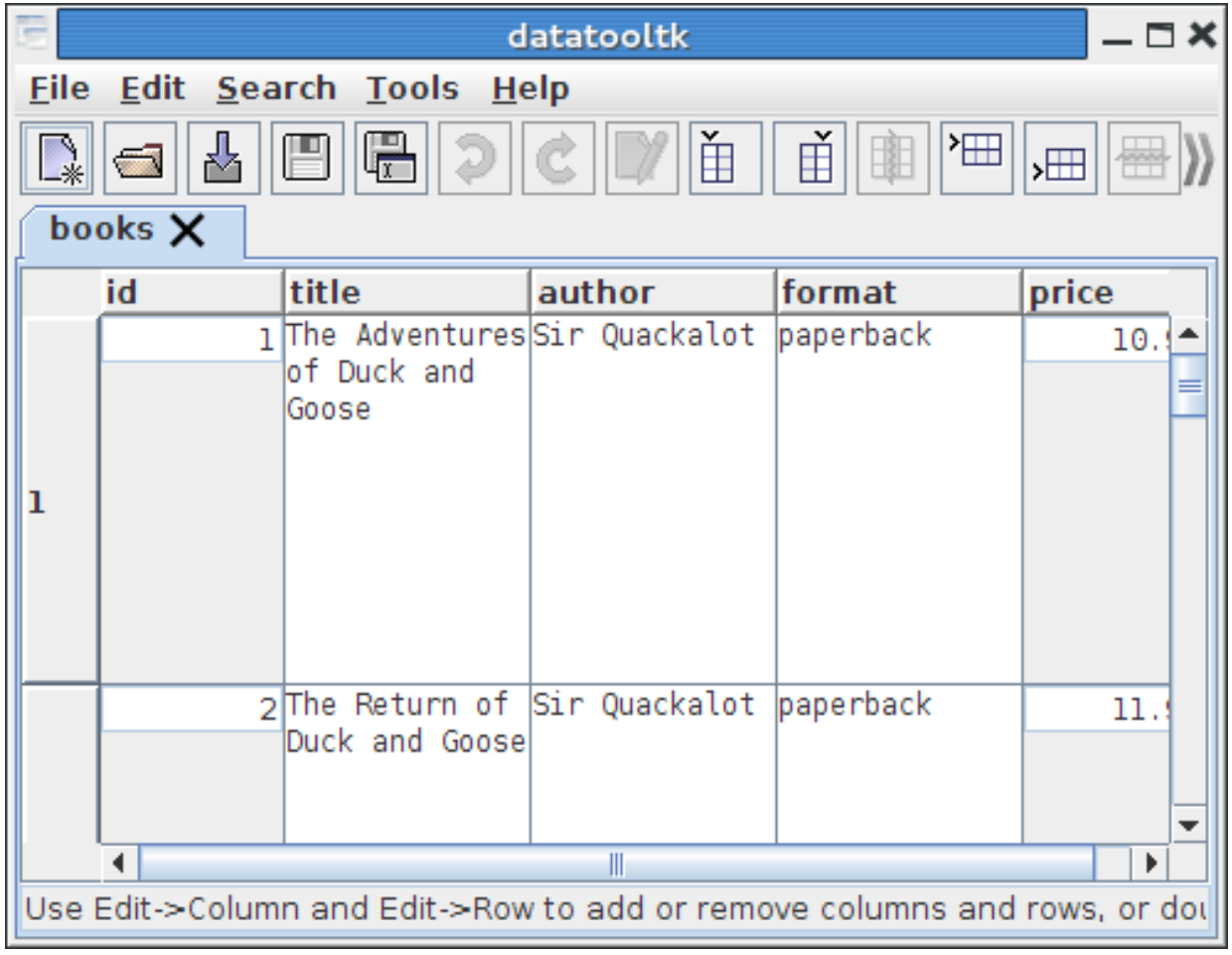


Figure 2.5 Imported SQL Data Shown in Main Window

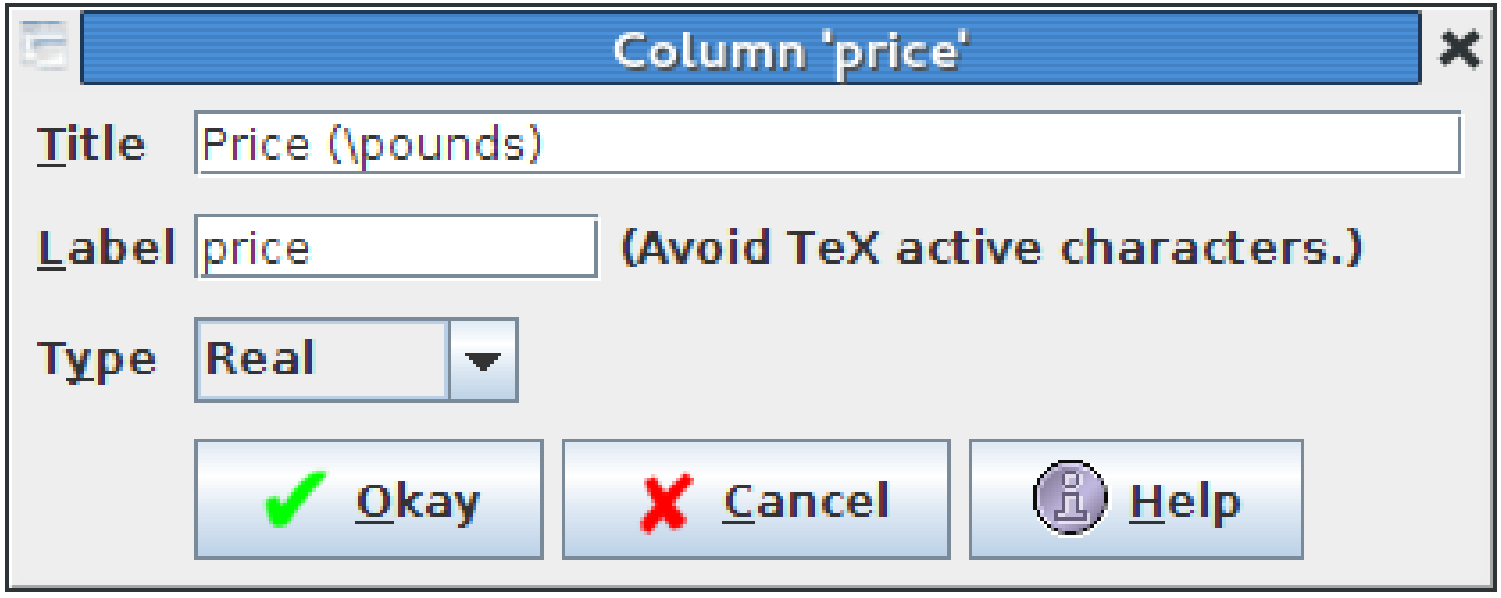


Figure 2.6 Changing the Column Header Text

↑ Input

```
% arara: datatooltk: {
% arara: --> output: books.dbtex,
% arara: --> csv: booklist.csv }
% arara: pdflatex
\documentclass{article}

\usepackage{datatool}

\DTLloaddbtex{\books}{books.dbtex}

\begin{document}
% Do stuff with data.
\end{document}
```

↓ Input

Remember that you can also use conditionals to prevent unnecessary application calls. For example:

↑ Input

```
% arara: datatooltk: {
% arara: --> output: books.dbtex,
% arara: --> csv: booklist.csv }
% arara: --> if changed(toFile("booklist.csv"))
% arara: --> || missing(toFile("books.dbtex"))
% arara: pdflatex while changed("tex")
% arara: --> || changed(toFile("books.dbtex"))
% arara: --> || missing("pdf")
% arara: --> || found("log", "Rerun")
\documentclass{article}

\usepackage{datatool}

\DTLloaddbtex{\books}{books.dbtex}

\begin{document}
% Do stuff with data.
\end{document}
```

↓ Input

2. Fetch the data from the sheet called "products" in an Excel `.xls` file called `shop.xls`:

↑ Input

```
% arara: datatooltk: {
% arara: --> output: products.dbtex,
% arara: --> xls: shop.xls,
% arara: --> sheet: products }
% arara: pdflatex
\documentclass{article}

\usepackage{datatool}

\DTLloaddbtex{\products}{products.dbtex}

\begin{document}
% Do stuff with data.
\end{document}
```

↓ Input

3. Fetch all the data from the table called books in a MySQL database called samples:

Input

```
% arara: datatooltk: {
% arara: --> output: books.dbtex,
% arara: --> sqldb: samples,
% arara: --> sqluser: sampleuser,
% arara: --> sql: "SELECT * FROM books" }
% arara: pdflatex
\documentclass{article}

\usepackage{datatool}

\DTLloaddbtex{\books}{books.dbtex}

\begin{document}
% Do stuff with data.
\end{document}
```

Input

In this case, datatooltk will prompt you for the MySQL password associated with the specified user name. There's a slight difference between running datatooltk directly from a terminal and running it via arara. The terminal usually provides a console for datatooltk to use to prompt for a password, but with arara there's no console so you'll get a dialog box instead (even if you haven't specified the --verbose option when you call arara). See Figure 2.7. This will also happen if you invoke datatooltk from another application, such as TeXworks.

If you're thinking of using a conditional in the arara directive, there's no platform-independent way of determining if the database has been modified. The way the data is stored on the hard disk varies not only according to the operating system but also with the way the table was created (MyISAM, InnoDB, etc). For example, on my Linux computer I could test if the file /var/lib/mysql/samples/books.MYD has changed, but I would need to run arara with sudo in order to access the file, which is unwise.

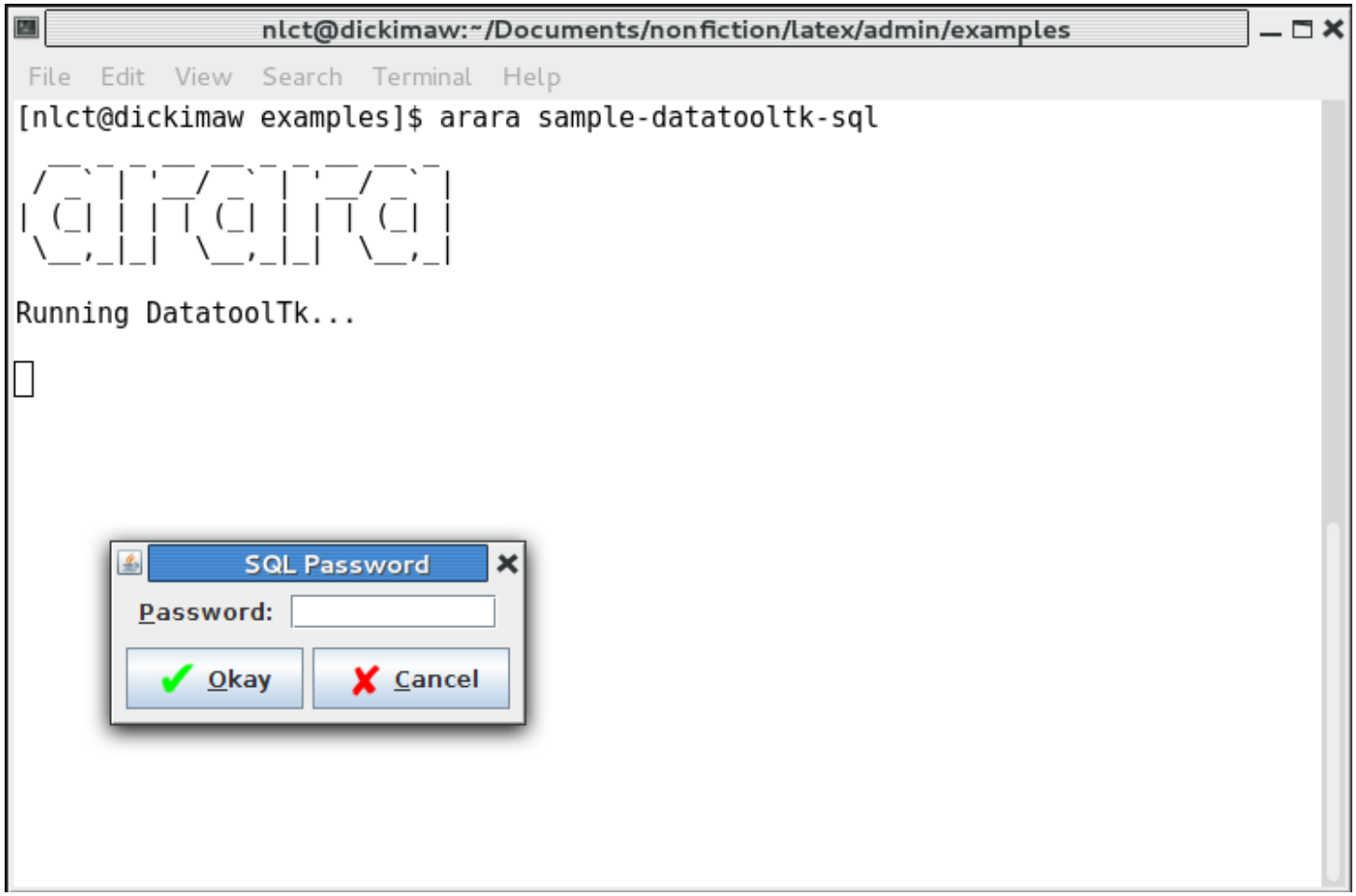


Figure 2.7 Running `datatooltk` via `arara` uses a dialog box to prompt for a MySQL password instead of using a console.

2.3 Security

When you import data from an SQL database, you need to enter the password associated with the SQL user. This must be done each time you run datatooltk with the --sql option. While it is possible to specify the password using the --sqlpassword option, this is not advisable as it will be visible to anyone who happens to be looking over your shoulder (and will probably be remembered by the shell's history). It's also not advisable to include the password in an arara directive since it can be read by anyone with access to the document source.

However, regardless of whether or not you use the --sqlpassword option, any confidential data that was imported from the SQL database will be present in the datatool (.dbtex) file, which other users may be able to access and read. If your operating system supports different permissions for owner and others (such as Unix-based systems) you can use the --owner-only option when invoking datatooltk so that the permissions on the .dbtex are set to read and write only for the owner. For example:

```
datatooltk --output people.dbtex --sqluser sampleuser
--sqldb samples --sql "SELECT * FROM people" --owner-only
```

Shell

On Unix-like systems this is equivalent to

```
datatooltk --output people.dbtex --sqluser sampleuser
--sqldb samples --sql "SELECT * FROM people"
chmod 600 people.dbtex
```

Shell

This means that only the owner of the file (you, if you ran the datatooltk command under your own user name) can read or write the people.dbtex file. Any other users with an account on the same machine should not be able to read or write that file. If your operating system doesn't support different owner and other permissions, the --owner-only option will have no effect.

If you are using arara the directive is:

Input

```
% arara: datatooltk: {
% arara: --> output: people.dbtex,
% arara: --> sqluser: sampleuser,
% arara: --> owneronly: true,
% arara: --> sqldb: samples,
% arara: --> sql: "SELECT * FROM people" }
```

Input

⚠ Even if you take the precaution of setting the permissions on the `.dbtex` file in this way, if you've used the data in your document, that data is also likely to be present in the resulting PDF file. If this data is sensitive, you also need to consider changing the permissions of the PDF file as well and possibly use something like pdftk to add encryption.

If your SQL database contains a mixture of private and public data, but you only want to use the public data in your document, make sure you omit the private data in your `SELECT` statement. For example, instead of using `SELECT *` to fetch all columns replace the asterisk with just the columns you want to fetch. Perhaps you only want to list the countries where you have customers, then you'd just use `SELECT country FROM people` which would only write the country information to the `.dbtex` file and not the individual customer details.

Remember that TEX discards comments in the form

% ⟨*comment text*⟩ Input

(unless the category code of % is changed, for example, by the verbatim environment). Here the comment text doesn't get hidden or embedded within the resulting PDF, but in other cases don't assume that just because you can't see the text you've included in your source code it won't be somehow present in the PDF file. For example

```
\textcolor{white}{some text}
```
Input

If this text is placed on a white background, it won't be visible to the human eye, but the text will be present in the PDF file and can be read by an electronic device such as a screen reader or it can be extracted using a PDF to text tool. The censor package described in §6.4.1 replaces the redacted text with a black rectangle rather than simply obscuring the text by painting a black rectangle on top of it, which means that the redacted text can't be extracted from the PDF.

Unlike Word [88], LaTeX doesn't automatically embed private information or revision logs in the document. With LaTeX you need to explicitly indicate the author or authors. This is done through the `\author` command in the standard classes. PDFLaTeX allows you to add metadata, but again this relies on you explicitly providing the author data. (The hyperref package provides a convenient key to do this called `pdfauthor`, which can be passed as a package option or through `\hypersetup{`⟨*options*⟩`}`. It's possible that other classes or packages may use `\author` or provide a similar command to add the author's name to the metadata, but it's still information provided by you and not automatically picked up from your operating system's environment variables or settings.)

TeX has a shell escape mechanism that allows processes to be spawned during the document build. However, since this can be exploited by malicious code, it's usually disabled completely or just enabled in restricted

mode, which only allows trusted applications (such as `makeindex` or `bibtex`) to be run. You can check the mode on your system by inspecting the log file. For example, if the log file contains "restricted `\write18` enabled" near the start, then TEX was run in restricted mode. The shell escape can be enabled using TEX's `-shell-escape` option, for example:

```
pdflatex -shell-escape myDoc
```
Shell

or disabled using the `-no-shell-escape` option, for example:

```
pdflatex -no-shell-escape myDoc
```
Shell

or the restricted mode can be enabled using `-shell-restricted`, for example:

```
pdflatex -shell-restricted myDoc
```
Shell

Note that `-shell-escape` is normally disallowed.

JavaScript code can be embedded in PDF files, which may be a security issue. The code is run by the PDF viewer when viewing the PDF file, not during the document build. If this is a concern for you, disable this feature in your PDF viewer. As with the author metadata, TEX doesn't automatically embed JavaScript code in the PDF file. You can explicitly embed JavaScript

code into interactive PDF elements, but obviously don't do this if you don't understand the code you're trying to embed.

You have control over the commands you enter into your source code, but what about the document class or packages that you load? If you have the shell escape disabled or restricted, then this automatically prevents any class or package from running dangerous applications. Some classes or packages may embed JavaScript code if their purpose is to create an interactive PDF document. If this concerns you, disable JavaScript in your PDF viewer, as mentioned above. Remember also that class and package files can be viewed in any text editor, so it's possible to inspect the code. While this may not appeal to you, if it's a well-used class or package that's on both MiKTEX and TEX Live, then the chances are that someone else already has.

What about applications such as `arara` and `latexmk`? You can add code to `latexmk` through the `.latexmkrc` file, but if you're writing the code, you should know what that code does. It's possible to embed Java code within the conditional part of `arara` directives, but again if you are writing the code it's up to you to ensure you don't write anything dangerous. If someone else has supplied a LATEX document that contains `arara` directives, you can search for them in the document source file before running `arara` on it. However `arara` will only execute a directive that has a corresponding `.yaml` file in its rules directory, and you can also use `--dry-run` to see what

commands would be executed.

Remember that if the application is available on TeX Live, then it has to have the source available. In fact, since `latexmk` is a Perl script, you can open it in a text editor. The `arara` directives are all defined in `.yaml` files, which again can be viewed in a text editor. An active open source community with a central file repository is far more likely to detect and flag malicious code than any users of proprietary systems.

But what if someone accesses your computer and modifies the `.latexmkrc` or `.yaml` files? In that case, you have far more to worry about than the integrity of your TeX distribution.

Other security issues you might want to consider are discussed in §6.4 and §13 Collaborating on Documents.

2.4 Sorting Data

Once you've loaded your data you can sort it using:

`\dtlsort[`⟨*replacement list*⟩`]{`⟨*criteria*⟩`}{`⟨*db-name*⟩`}{`⟨*handler*⟩`}` Definition

where ⟨*db-name*⟩ is the label that identifies the database. The ⟨*criteria*⟩ argument is a comma-separated list of column labels that indicate the sort order. For example, if you first want to sort on the surname column and

then on the forenames column the ⟨*criteria*⟩ should be surname,forenames (make sure you don't have any unwanted spaces in the list). You can optionally add =⟨*order*⟩ after the column label where ⟨*order*⟩ is either ascending or descending. If omitted, ascending is assumed. For example, to sort in descending order, first by surname and then by forenames, the ⟨*criteria*⟩ should be:

```
surname=descending,forenames=descending
```

The ⟨*handler*⟩ argument is a control sequence that's used for the comparisons. The datatool package comes with four handlers:

1. A case-sensitive comparison:

   \dtlcompare Definition

2. A case-insensitive comparison:

   \dtlicompare Definition

3. English word-ordering comparison (as described by the Oxford Style Manual [73]):

\dtlwordindexcompare Definition

4. English letter-ordering comparison:

\dtlletterindexcompare Definition

The last two are intended for indexes. If you want any further details about those handlers, see the datatool user guide [94]. The first two handlers, \dtlcompare and \dtlicompare, are the ones you're most likely to need for administrative purposes. The datatool package provides convenient shortcuts:

\DTLsort[⟨*replacement list*⟩]{⟨*criteria*⟩}{⟨*db-name*⟩} Definition

which uses \dtlcompare and

\DTLsort*[⟨*replacement list*⟩]{⟨*criteria*⟩}{⟨*db-name*⟩} Definition

which uses \dtlicompare.

The optional argument ⟨*replacement list*⟩ is provided in case null values are encountered. You're unlikely to have null values if you load your data from a CSV file, but you may have null values if you use datatooltk to fetch data from a SQL database.

For example, suppose you want to sort your data according to the author column, but if there's a null value in that column then sort it by the title column, you would need to do

```
\DTLsort[title]{author}{books}
```
Input

(where books is the label identifying your data.)

⚠ TEX isn't designed for data analysis, so sorting your data within your document in this way isn't very efficient. It's therefore better to sort your data before you load it in your document. This can be done using datatooltk's --sort option (in batch mode) or via the Tools→Sort menu item (in GUI mode). Alternatively, if you're fetching data from a SQL database, it's more efficient to append the ORDER BY statement to your SELECT statement.

Examples:

1. Fetch and sort data from a CSV file.

   - Either:

```
\DTLloaddb{books}{booklist.csv}
```
↑ Input

```
\DTLsort{author}{books}
```

Input

- Or use datatooltk:

```
datatooltk --output books.dbtex
--csv booklist.csv --sort author
```

Shell

and then in your document:

```
\DTLloaddbtex{\books}{books.dbtex}
```

Input

2. Fetch and sort data from a SQL database:

```
datatooltk --output books.dbtex --sqluser sampleuser
--sqldb samples --sql "SELECT * FROM books ORDER BY
author"
```

Shell

and then in your document:

```
\DTLloaddbtex{\books}{books.dbtex}
```

Input

2.5 Sample Data

This section describes the sample CSV files, Excel spreadsheet and SQL database tables that will be used in some of the examples and exercises throughout the rest of this book. This includes sample data regarding hypothetical people. These people could represent, for example, students or customers or members of an organisation. This leads on to a couple of points that aren't specific to TEX, but are of a more general nature.

If you intend to store personal data, make sure you are aware of your country's data storage legislation. For example, in the United Kingdom you need to be familiar with the data protection act and you will probably also need to register as a data controller with the Information Commissioner's Office (ICO), although there are some exemptions. (At the time of writing it usually costs £35 per year, but may cost £500 per year, depending on your organisation's size.) If you are an employee, check with your company's administrative office for further details.

The sample data includes both a gender and a title field. The gender is usually not required, unless you need to know whether to use male or female pronouns (for example, "him" or "her") or where the gender has some significance (for example, in medical data). Some people view a request for their gender to be intrusive, but if it is genuinely needed, make sure you include it in the data rather than omitting it and trying to de-

termine the gender from the person's name. Additionally, I recommend you don't use a gender field as a replacement for the title field. It's not advisable to assume "Mr" or "Ms" on the basis of gender. People's preferences are varied. Some don't like the formality of titles at all, or simply don't care how they're addressed, but some women object to being addressed as "Ms". While it's true that some professional women prefer "Ms" as they feel their marital status is no one's business, there are, on the other hand, some women with a professional title, such as "Dr" or "Prof", who object to being "demoted" to a "Ms", particularly if they work in a male-dominated environment and they perceive, rightly or wrongly, that their male colleagues are being shown more respect professionally. There are, of course, plenty of other titles as well, such as "Rev", so I think it's best to find out how people prefer to be addressed. If the data is collected by a third party (such as an online store script) and it doesn't provide you with the person's title, you may want to consider just addressing individuals by both their forename and surname (such as "Dear John Smith") rather than guessing a title to avoid unwittingly causing offence. (Remember there are also people, of certain cultures or age, who object to the informality of being addressed by their first name.)

2.5.1 Sample CSV Files

Some of these files contains UTF-8 characters, so you'll also need to use:

Input

```
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
```

Input

(or just fontspec [75] for XeLaTeX) and make sure your text editor is set to UTF-8 encoding. The sample CSV files are as follows (you can download them from the examples page):

1. `booklist.csv` (download)

   This is a list of sample book titles. They could represent, say, products in a shop or a reading list for students. The `id` column provides a number that uniquely identifies the book. (This would typically be an ISBN in a real life book list or the ISBN might be in an additional column.)

```
id,title,author,format,price
1,"The Adventures of Duck and Goose","Sir Quackalot",paperback,
10.99
```

```
2,"The Return of Duck and Goose","Sir Quackalot",paperback,
19.99
3,"More Fun with Duck and Goose","Sir Quackalot",paperback,
12.99
4,"Duck and Goose on Holiday","Sir Quackalot",paperback,
11.99
5,"The Return of Duck and Goose","Sir Quackalot",hardback,
19.99
6,"The Adventures of Duck and Goose","Sir Quackalot",hardback,
18.99
7,"My Friend is a Duck","A. Parrot",paperback,14.99
8,"Annotated Notes on the ‘Duck and Goose’ chronicles",
"Prof Macaw",ebook,8.99
9,"‘Duck and Goose’ Cheat Sheet for Students",
"Polly Parrot",ebook,5.99
10,"‘Duck and Goose’: an allegory for modern times?",
"Bor Ing",hardback,59.99
```

This file can be loaded using:

```
\DTLloaddb{books}{booklist.csv}
```
Input

2. people.csv (download)

   This is a list of sample people. They could represent, say, customers or students or members of an organization. The id column provides a number that uniquely identifies the person. (For example, a student's registration number.) The subscribed field indicates whether the person wants to be subscribed to some form of mailing list where a value of 1 means they want to be subscribed. The gender column can be either "m" (male) or "f" (female). The dob column is the date of birth in the ISO format ⟨*year*⟩-⟨*month*⟩-⟨*day*⟩.

```
id,forenames,surname,title,address1,address2,town,county,
country,postcode,subscribed,gender,dob
1,Polly,Parrot,Miss,42 The Lane,,Some Town,Noshire,gb,
AB1 2XY,1,f,1970-12-31
2,Mabel,Canary,Mrs,24 The Street,Some Village,Some Town,
Noshire,gb,AB1 2YZ,0,f,1968-01-23
3,Zöe,Zebra,Ms,856 The Avenue,,Some City,CA,us,123456,1,
f,1989-07-16
```

```
4,José,Arara,,Nenhuma Rua,,São Paulo,,br,123457,1,m,
1991-05-30
5,Dickie,Duck,Mr,1 The Street,Another Village,Some City,
Imagineshire,gb,YZ1 2AB,0,m,1952-11-25
6,Fred,Canary,Mr,24 The Street,Some Village,Some Town,
Noshire,gb,AB1 2YZ,1,m,1967-08-04
```

This file can be loaded using:

```
\DTLloaddb{people}{people.csv}
```

Input

3. `country-codes.csv` (download)

   This is a list of country codes. It contains 250 lines. You can download the entire file from the examples page. For brevity, only the header row and the rows containing codes that are referenced in `people.csv` are shown below:

```
code,name
br,Brazil
gb,United Kingdom
us,United States
```

This file can be loaded using:

```
\DTLloaddb{countries}{country-codes.csv}
```
Input

4. `ordergroups.csv` (download)

   This is a list of customer orders. The first column is a number that uniquely identifies the order, the second column is a cross-reference to the `id` field in the `people.csv` file above, indicating the purchaser, the third column is a discount applied to the order, and the fourth column is the cost of postage and packaging.

```
id,customerid,discount,postage
1,2,0,5.00
2,4,2.50,20.00
3,1,0,5.00
```

   (In practice, this would typically have extra fields, such as the dispatch status and order date.)

   This file can be loaded using:

```
\DTLloaddb{ordergroups}{ordergroups.csv}
```
Input

5. `orders.csv` (download)

   This is a list of partial customer orders. Each row represents part of an order for one of the book titles listed in `booklist.csv` by one of the people listed in `people.csv`. There are four columns: the first is a number that uniquely identifies each order part, the second is a number that identifies the entire order (matching the `id` field in `ordergroups.csv`), the third column is a cross-reference to the `id` field in the `booklist.csv` file above, and the fourth column is the quantity ordered.

```
id,groupid,bookid,quantity
1,1,6,1
2,1,1,4
3,2,10,1
4,2,7,20
5,2,8,1
6,3,1,4
7,3,6,5
8,3,7,2
```

The first two rows form a single order (id 1 from the `ordergroups.csv` file). The order consists of two parts: one copy of the book whose id is 6 (the hardback version of "The Adventures of Duck and Goose") and four copies of the book whose id is 1 (the paperback version of "The Adventures of Duck and Goose"). The next three rows form another order (id 2 from `ordergroups.csv`). The order consists of three parts: one copy of the book whose id is 10, twenty copies of the book whose id is 7 and one copy of the book whose id is 8. The last four rows form the third order (id 3 from `ordergroups.csv`). The order consists of three parts: four copies of the book whose id is 1, five copies of the book whose id is 6 and two copies of the book whose id is 7.

This file can be loaded using:

```
\DTLloaddb{orders}{orders.csv}
```

Input

2.5.2 Sample XLS File

Most of the examples in this book will use data from either CSV files or SQL tables, but `datatooltk` can also import from Excel `.xls` files so there is one sample file, `shop.xls`, to illustrate this. The `shop.xls` file has

two sheets called "products" and "customers". The first sheet is shown in Figure 2.8. This has three columns. The second and third columns ("B" and "C") have been set to a currency format. In addition, the third column contains formulae instead of an explicitly entered number.

The data can be fetched via datatooltk. For example, to fetch the data from the first sheet ("products"):

```
datatooltk --output products.dbtex --xls shop.xls
--sheet products
```

Shell

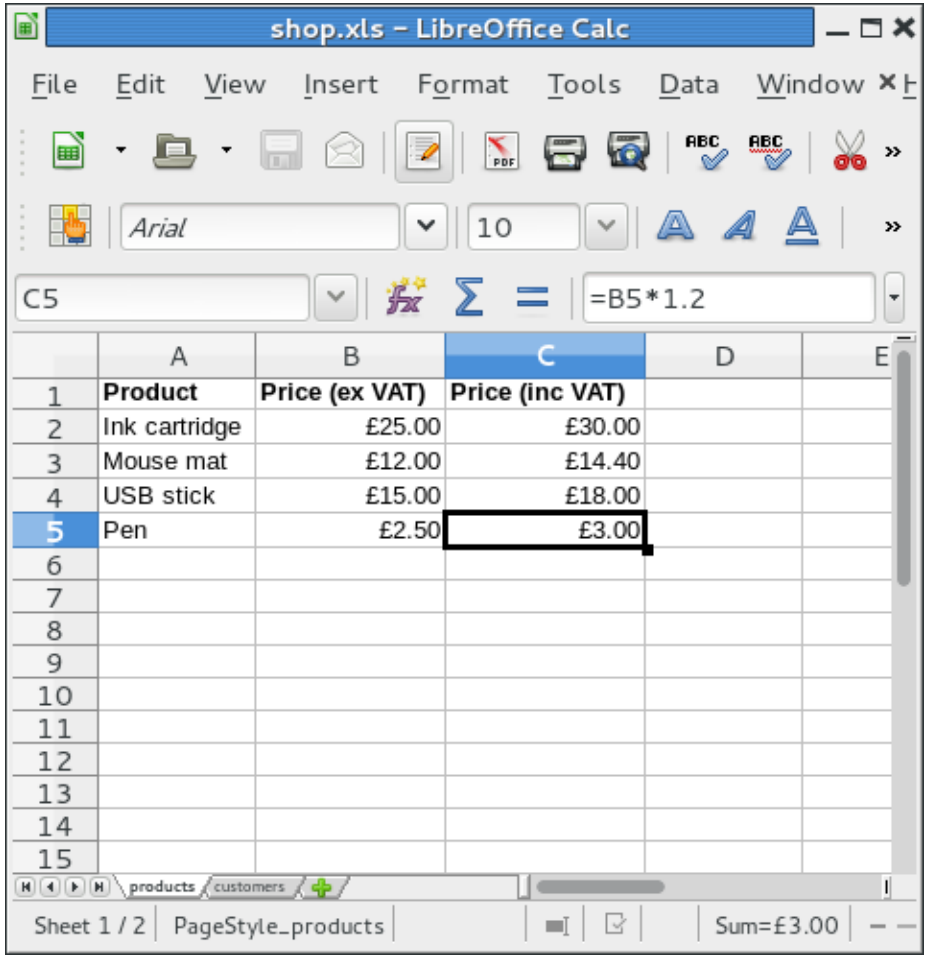
Or using an arara directive:

```
% arara: datatooltk: {
% arara: --> output: products.dbtex,
% arara: --> xls: shop.xls,
% arara: --> sheet: products }
```

Input

Input

The resulting products.dbtex file can be loaded in your document using:



	A	B	C	D	E
1	**Product**	**Price (ex VAT)**	**Price (inc VAT)**		
2	Ink cartridge	£25.00	£30.00		
3	Mouse mat	£12.00	£14.40		
4	USB stick	£15.00	£18.00		
5	Pen	£2.50	£3.00		
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

Figure 2.8 Sample XLS File (First Sheet)

```
\DTLloaddbtex{\products}{products.dbtex}
```
Input

You can also import Open Document `.ods` spreadsheets using a similar method to the above but you need to use `--ods` instead of `--xls`.

2.5.3 Sample SQL Tables

The sample SQL database called "samples" is created using:

```
CREATE DATABASE samples;
```

A sample user called "sampleuser" with the password "sample-passwd" is created using:

```
GRANT SELECT ON samples.* TO 'sampleuser'@'localhost'
IDENTIFIED BY 'sample-passwd';
```

Remember to switch to this database before you try creating the tables in it:

```
USE samples;
```

If you like, you can download `samples.sql` and add this sample database (including the tables below) using:

```
mysql -u root -p < samples.sql
```

Shell

(Alternatively, you can use one of the GUI MySQL tools, such as MySQL Workbench.)

The tables included in this sample database are analogous to the CSV files described in §2.5.1. As with those files, some of the SQL tables include UTF-8 characters, so you'll also need to use:

↑ Input

```
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
```

↓ Input

(or just fontspec [75] for XeLaTeX) and make sure your text editor is set to UTF-8 encoding.

The tables are defined as follows:

1. books

   This table is created using:

   ```
   CREATE TABLE books
   (
   ```

```
 id INT PRIMARY KEY AUTO_INCREMENT,
 title VARCHAR(64),
 author VARCHAR(32),
 format ENUM('paperback', 'hardback', 'ebook'),
 price DECIMAL(5,2)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

The data is added to the table via:

```
INSERT INTO books (title, author, format, price)
  VALUES ('The Adventures of Duck and Goose',
   'Sir Quackalot', 'paperback', 10.99);
INSERT INTO books (title, author, format, price)
  VALUES ('The Return of Duck and Goose', 'Sir Quackalot',
   'paperback', 11.99);
INSERT INTO books (title, author, format, price)
  VALUES ('More Fun with Duck and Goose', 'Sir Quackalot',
   'paperback', 12.99);
INSERT INTO books (title, author, format, price)
  VALUES ('Duck and Goose on Holiday', 'Sir Quackalot',
   'paperback', 11.99);
INSERT INTO books (title, author, format, price)
  VALUES ('The Return of Duck and Goose', 'Sir Quackalot',
```

```
   'hardback', 19.99);
INSERT INTO books (title, author, format, price)
  VALUES ('The Adventures of Duck and Goose', 'Sir Quackalot',
   'hardback', 18.99);
INSERT INTO books (title, author, format, price)
  VALUES ('My Friend is a Duck', 'A. Parrot', 'paperback',
   14.99);
INSERT INTO books (title, author, format, price)
  VALUES (
   'Annotated Notes on the ‘Duck and Goose’ chronicles',
   'Prof Macaw', 'ebook', 8.99);
INSERT INTO books (title, author, format, price)
  VALUES ('‘Duck and Goose’ Cheat Sheet for Students',
   'Polly Parrot', 'ebook', 5.99);
INSERT INTO books (title, author, format, price)
  VALUES ('‘Duck and Goose’: an allegory for modern times?',
   'Bor Ing', 'hardback', 59.99);
```

2. people

   This table is created using:

   ```
   CREATE TABLE people
   ```

```
(
  id INT PRIMARY KEY AUTO_INCREMENT,
  forenames VARCHAR(32) NOT NULL,
  surname VARCHAR(32) NOT NULL,
  title VARCHAR(8),
  address1 VARCHAR(32) NOT NULL,
  address2 VARCHAR(32),
  town VARCHAR(32) NOT NULL,
  county VARCHAR(32),
  country CHAR(2) NOT NULL,
  postcode VARCHAR(32),
  subscribed BIT(1) DEFAULT 0,
  gender ENUM('male', 'female'),
  dob DATE,
  INDEX(id, surname)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

The data is added to the table via:

```
INSERT INTO people
  (forenames, surname, title, address1, address2,
   town, county, country, postcode, subscribed, gender, dob)
  VALUES ('Polly', 'Parrot', 'Miss', '42 The Lane', NULL,
```

```
    'Some Town', 'Noshire', 'gb', 'AB1 2XY', 1, 'female',
    '1970-12-31');
INSERT INTO people
  (forenames, surname, title, address1, address2,
   town, county, country, postcode, subscribed, gender, dob)
  VALUES ('Mabel', 'Canary', 'Mrs', '24 The Street',
   'Some Village', 'Some Town', 'Noshire', 'gb', 'AB1 2YZ',
   0, 'female', '1968-01-23');
INSERT INTO people
  (forenames, surname, title, address1, address2,
   town, county, country, postcode, subscribed, gender, dob)
  VALUES ('Zöe', 'Zebra', 'Ms', '856 The Avenue', NULL,
   'Some City', 'CA', 'us', '123456', 1, 'female', '1989-07-16');
INSERT INTO people
  (forenames, surname, title, address1, address2,
   town, county, country, postcode, subscribed, gender, dob)
  VALUES ('José', 'Arara', NULL, 'Nenhuma Rua', NULL,
   'São Paulo', NULL, 'br', '123457', 1, 'male', '1991-05-30');
INSERT INTO people
  (forenames, surname, title, address1, address2,
   town, county, country, postcode, subscribed, gender, dob)
  VALUES ('Dickie', 'Duck', 'Mr', '1 The Street',
```

```
   'Another Village', 'Some City', 'Imagineshire', 'gb',
   'YZ1 2AB', 0, 'male', '1952-11-25');
INSERT INTO people
  (forenames, surname, title, address1, address2,
   town, county, country, postcode, subscribed, gender, dob)
  VALUES ('Fred', 'Canary', 'Mr', '24 The Street',
   'Some Village', 'Some Town', 'Noshire', 'gb', 'AB1 2YZ',
   1, 'male', '1967-08-04');
```

3. countries

   This is a list of country codes. The table is created using:

```
CREATE TABLE countries
(
  code CHAR(2) PRIMARY KEY NOT NULL,
  name VARCHAR(64) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

   This table contains 249 entries. For brevity, only the rows that contain codes referenced in the people table are shown below:

```
INSERT INTO countries (code, name)
```

```
  VALUES ('br', 'Brazil');
INSERT INTO countries (code, name)
  VALUES ('gb', 'United Kingdom');
INSERT INTO countries (code, name)
  VALUES ('us', 'United States');
```

You can view or download the complete list from the examples web page.

4. ordergroups This is a list of order groups. The table is created using:

```
CREATE TABLE ordergroups
(
  id INT PRIMARY KEY AUTO_INCREMENT,
  customerid INT NOT NULL REFERENCES people (id),
  discount DECIMAL(5,2),
  postage DECIMAL(5,2)
) ENGINE=MyISAM;
```

The data is added to the table via:

```
INSERT INTO ordergroups (customerid, discount, postage)
```

```
 VALUES (2, 0.0, 5.0);
INSERT INTO ordergroups (customerid, discount, postage)
 VALUES (4, 2.5, 20.0);
INSERT INTO ordergroups (customerid, discount, postage)
 VALUES (1, 0.0, 5.0);
```

5. orders

   This is a list of partial orders. The table is created using:

```
CREATE TABLE orders
(
  id INT PRIMARY KEY AUTO_INCREMENT,
  groupid INT NOT NULL REFERENCES ordergroups (id),
  bookid INT NOT NULL REFERENCES books (id),
  quantity INT NOT NULL
) ENGINE=MyISAM;
```

   The data is added to the table via:

```
INSERT INTO orders (groupid, bookid, quantity)
  VALUES (1, 6, 1);
INSERT INTO orders (groupid, bookid, quantity)
```

```
  VALUES (1, 1, 4);
INSERT INTO orders (groupid, bookid, quantity)
  VALUES (2, 10, 1);
INSERT INTO orders (groupid, bookid, quantity)
  VALUES (2, 7, 20);
INSERT INTO orders (groupid, bookid, quantity)
  VALUES (2, 8, 1);
INSERT INTO orders (groupid, bookid, quantity)
  VALUES (3, 1, 4);
INSERT INTO orders (groupid, bookid, quantity)
  VALUES (3, 6, 5);
INSERT INTO orders (groupid, bookid, quantity)
  VALUES (3, 7, 2);
```

This data can be fetched via datatooltk. For example, to fetch the data from the books table:

```
datatooltk --output books.dbtex --sqluser sampleuser
--sqldb samples --sql "SELECT * FROM books"
```

Shell

If you prefer to use arara you can use

```
% arara: datatooltk: {
% arara: --> output: books.dbtex,
% arara: --> sqluser: sampleuser,
% arara: --> sqldb: samples,
% arara: --> sql: "SELECT * FROM books" }
```

Input

Input

The resulting file books.dbtex can now be loaded in your document via:

```
\DTLloaddbtex{\books}{books.dbtex}
```

Input

If required, the column headers can be set using \DTLsetheader. For example:

```
\DTLsetheader{\books}{id}{ID}
\DTLsetheader{\books}{title}{Title}
\DTLsetheader{\books}{author}{Author}
\DTLsetheader{\books}{format}{Format}
\DTLsetheader{\books}{price}{Price (\pounds)}
```

Input

Input

Remember you can use the SELECT statement to sort or filter the data or join with another table. For example, if you only want to fetch customers in the UK and order them by their surname:

```
datatooltk --output people.dbtex --sqluser sampleuser
--sqldb samples --sql "SELECT * FROM people
WHERE country='gb' ORDER BY surname"
```

Shell

2.6 Displaying Tabulated Data

Once you have loaded your data you can display it using:

`\DTLdisplaydb[`⟨*omit list*⟩`]{`⟨*db-name*⟩`}`

Definition

where ⟨*db-name*⟩ is the database label. The optional argument ⟨*omit list*⟩ is a comma-separated list of labels indicating which columns you want omitted from the tabular environment. Make sure you don't have any unwanted spaces in ⟨*omit list*⟩.

The `\DTLdisplaydb` command uses a tabular environment internally and should typically go inside a table environment. If the data is too big to fit on one page, you can instead use:

\DTLdisplaylongdb[⟨*options*⟩]{⟨*db-name*⟩} Definition

This uses the longtable environment defined by the longtable package [11] instead of the tabular environment (see §4.3). As with \DTLdisplaydb, ⟨*db-name*⟩ indicates the label identifying the data. Note that \DTLdisplaylongdb should not be put inside a table environment.

The optional argument ⟨*options*⟩ is a key=value list. The following keys are available:

`caption`	The caption for the longtable.
`contcaption`	The continuation caption for the longtable.
`shortcaption`	The caption to be used in the list of tables.
`label`	The label for this table. (To be used in \ref{⟨*label*⟩}, if required.)
`omit`	Comma-separated list of labels identifying columns to be omitted.
`foot`	The longtable's foot.
`lastfoot`	The foot for the last page of the longtable.

Remember to load the longtable package if you want to use \DTLdisplaylongdb.

Example 4. Display Product List

Suppose I want to display the data from my sample booklist.csv file in a table environment, but I don't want to include the id or author columns:

Input

```
% Load "booklist.csv":
\DTLloaddb
 [headers={Id,Title,Author,Format,Price (\pounds)}]% column headers
 {books}{booklist.csv}

\begin{table}[htbp]
\caption{Products}
\label{tab:products}
\centering
\DTLdisplaydb[id,author]{books}
\end{table}
```

Input

This produces Table 2.2.

Suppose instead I want to use the equivalent books SQL table. If I'm not interested in using the id or format columns at all in the document, I can exclude them when I'm importing the data. From the command line:

Table 2.2 Products

Title	Format	Price (£)
The Adventures of Duck and Goose	paperback	10.99
The Return of Duck and Goose	paperback	11.99
More Fun with Duck and Goose	paperback	12.99
Duck and Goose on Holiday	paperback	11.99
The Return of Duck and Goose	hardback	19.99
The Adventures of Duck and Goose	hardback	18.99
My Friend is a Duck	paperback	14.99
Annotated Notes on the 'Duck and Goose' chronicles	ebook	8.99
'Duck and Goose' Cheat Sheet for Students	ebook	5.99
'Duck and Goose': an allegory for modern times?	hardback	59.99

```
datatooltk --output books.dbtex --sqldb samples
--sqluser sampleuser --sql "SELECT title, author, price
FROM books"
```

Shell

or using an arara directive:

↑ Input

```
% arara: datatooltk: {
% arara: --> output: books.dbtex,
% arara: --> sqldb: samples,
% arara: --> sqluser: sampleuser,
% arara: --> sql: "SELECT title, author, price FROM books" }
```

↓ Input

Once the file books.dbtex has been created I can load it into my document using:

```
\DTLloaddbtex{\books}{books.dbtex}
```

Input

and set the headers using:

Input

```
\DTLsetheader{\books}{title}{Title}
\DTLsetheader{\books}{author}{Author}
\DTLsetheader{\books}{format}{Format}
\DTLsetheader{\books}{price}{Price (\pounds)}
```

Input

(Alternatively, if I import the SQL data using `datatooltk-gui` I can set the header text by double-clicking on the column header as described in §2.2.2.)

The table can now be created in the same way as before. Putting it all together:

Input

```
% Load "books.dbtex":
\DTLloaddbtex{\books}{books.dbtex}

% Set the headers:
\DTLsetheader{\books}{title}{Title}
\DTLsetheader{\books}{author}{Author}
\DTLsetheader{\books}{format}{Format}
```

```
\DTLsetheader{\books}{price}{Price (\pounds)}

% Display the data:

\begin{table}[htbp]
\caption{Products}
\label{tab:products}
\centering
\DTLdisplaydb[id,author]{books}
\end{table}
```

Input

This produces the same result shown in Table 2.2.

A longer database, such as the sample `country-codes.csv` file, would need `\DTLdisplaylongdb`, which requires the longtable package:

Input

```
\usepackage{longtable}
\usepackage{datatool}
```

Input

Now load the data:

```
\DTLloaddb{countries}{country-codes.csv}
```
Input

Later in the document, display the data in a longtable environment:

```
\DTLdisplaylongdb
 [
   caption={A Sample Long Table},% main caption
   contcaption={A Sample Long Table (Continued)},% continuation
   label={tab:countries},% table label
   foot={\emph{Continued on next page}},% table foot
   lastfoot={}% final table foot
 ]
 {countries}
```
Input

The table can then be referenced using:

```
\label{tab:countries}
```
Input

You can download or view the full example document using CSV files, or download or view the example using a SQL database.

The commands \DTLdisplaydb and \DTLdisplaylongdb are useful if you just want to list the data, but if you want to modify the displayed format (for example, if you want to swap columns or highlight a row) then you'll need to construct the contents of the tabular or longtable environments by iterating through the data using \DTLforeach, described in §2.7.1.

⚠ Take care if you have imported your data from an Excel .xls file as the formatting information isn't imported. This may cause an unexpected result when you use \DTLdisplaydb or \DTLdisplaylongdb. This is illustrated in the example below.

Example 5. Displaying Data Imported from a Spreadsheet

Recall the spreadsheet shown in Figure 2.8. Viewed in a spreadsheet application, the second and third columns are displayed as currency with two decimal places. Now suppose I import the data from the "products" sheet using datatooltk:

```
datatooltk --output shop-products.dbtex --xls shop.xls
--sheet products
```

Shell

and load the resulting shop-products.dbtex file in my document:

```
\DTLloaddbtex{\xlsproducts}{shop-products.dbtex}
```

Input

then

Input

```
\begin{table}
  \caption{Data imported from \texttt{shop.xls}}
  \label{tab:xlsproducts}
  \centering
  \DTLdisplaydb{\xlsproducts}
\end{table}
```

Input

produces Table 2.3. Note that the second and third columns are no longer displayed as currency nor are the numbers rounded to two decimal places. (See Example 6 for a different approach that rounds and formats the price columns.)

Table 2.3 Data Imported From `shop.xls`

Product	Price (ex VAT)	Price (inc VAT)
Ink cartridge	25.0	30.0
Mouse mat	12.0	14.399999999999999
USB stick	15.0	18.0
Pen	2.5	3.0

You can download or view this example document.

Exercise 1. Loading and Displaying Data

Try loading data from the sample booklist.csv file or from the books SQL table and displaying it in a table. Then try displaying the sample country-codes.csv file or countries SQL table data in a longtable.

For the More Adventurous:

The datatool package provides some hooks to allow you to make minor modifications to the default layout of \DTLdisplaydb and \DTLdisplaylongdb. These include the commands:

\dtldisplaystarttab Definition

which is done at the start,

\dtldisplayendtab Definition

which is done at the end and

\dtldisplayafterhead Definition

which is done after the header row. In the case of \DTLdisplaylongdb, the hooks \dtldisplaystarttab and \dtldisplayafterhead are used before

and after the header row on *each* page of the longtable, but \dtldisplayendtab is only used on the *last* page of the longtable. You will need to use the foot option if you want to specify code that should appear at the bottom of every page of the longtable.

These three commands default to nothing, but you can redefine them before you display the data. For example, recall from Volume 1 [92, §4.6.3] that the booktabs package [24] provides:

\toprule[⟨*wd*⟩] Definition

for a top horizontal rule,

\bottomrule[⟨*wd*⟩] Definition

for a bottom horizontal rule, and

\midrule[⟨*wd*⟩] Definition

for a horizontal rule between rows. See if you can edit your example document to include these rules in the table displaying the list of products.

You can download or view the solution to this exercise.

2.7 Iteration

[FAQ: Repeating a command *n* times]

[FAQ: Repeating something for each 'thing' in a set]

This section covers iterating over data. The datatool package provides a way of iterating over rows of a database, discussed in §2.7.1, but you may also find you need to iterate over a comma-separated list, so this is discussed in §2.7.2. If you want to manually build up a list and then iterate over it, you may prefer to use etoolbox's internal lists, discussed in §2.7.3. For a more general purpose loop ability, you may prefer to use TEX's `\loop`, discussed in §2.7.4.

2.7.1 Iterating Through a Database

The datatool package provides ways of iterating through a database and performing a task on each row of data. The two main commands are:

`\DTLforeach[`⟨*condition*⟩`]{`⟨*db-name*⟩`}{`⟨*assign*⟩`}{`⟨*body*⟩`}` Definition

and its starred version:

`\DTLforeach*[`⟨*condition*⟩`]{`⟨*db-name*⟩`}{`⟨*assign*⟩`}{`⟨*body*⟩`}` Definition

The unstarred version allows you to modify the data *stored internally* (that is, in TEX's registers used by datatool, not in the original loaded or imported source). As it's more efficient to do any modifications in your spreadsheet

or via SQL these datatool commands aren't covered here. Instead, all the examples in this document will use the read-only starred version, which compiles faster. The parameters for both versions are as follows:

⟨*db-name*⟩ The label identifying the internal database.

⟨*assign*⟩ A comma-separated list of ⟨*cs*⟩=⟨*col-label*⟩ assignments where ⟨*cs*⟩ is a control sequence that can be used as a placeholder in ⟨*body*⟩ and ⟨*col-label*⟩ is the label identifying the required column. *Spaces aren't ignored in this list* (except after ⟨*cs*⟩ as per TEX's normal behaviour). There is no check for the existence of ⟨*cs*⟩ so be careful you don't accidentally overwrite an existing command. You only need to assign control sequences to the columns whose values you intend to use in ⟨*body*⟩. Assignments are performed with the `\global` prefix to ensure that `\DTLforeach` works correctly within a tabular (or similar) environment.

⟨*body*⟩ The code to do for each row of data where the condition given in the optional argument is true.

⟨*condition*⟩ This optional argument should be a conditional that follows the same syntax as the `\ifthenelse` command defined in the ifthen package [10]. For example, you can use:

\equal{⟨*text 1*⟩}{⟨*text 2*⟩} Definition

to test if ⟨*text 1*⟩ is equal to ⟨*text 2*⟩.

The ⟨*body*⟩ is only applied to those rows where the condition is met. The default is \boolean{true}. If you're importing data from a SQL database, then it's better to apply any filtering in the SELECT statement.

You can prematurely terminate the list at the end of the current iteration by placing

\dtlbreak Definition

anywhere within ⟨*body*⟩.

For example, to simply print each surname in the people data:

```
\DTLforeach*{people}{\Surname=surname}{\Surname. }
```
Input

Using the sample people.csv file, this produces:

Parrot. Canary. Zebra. Arara. Duck. Canary. Output

To just print the forenames of the people whose surname is "Canary":

Input

```
\DTLforeach*
 [\equal{\Surname}{Canary}]% condition
 {people}% database
 {\Surname=surname,\Forenames=forenames}% assignments
 {\Forenames. }% body
```

Input

which produces:

Mabel. Fred.

Output

Example 6. Iterating Through Data

Recall from Example 5 that data imported from an Excel `.xls` file doesn't include any of the formatting used by the spreadsheet, so Table 2.3 (produced using `\DTLdisplaydb`) didn't display the numerical data as currency. Instead of using `\DTLdisplaydb` we can use `\DTLforeach*` to display the table and use `\dtlround` (described in §2.1.3) to round the values to two decimal places.

Input

```
\begin{table}
 \caption{Formatted data imported from \texttt{shop.xls}}
 \label{tab:xlsproducts2}
 \centering
 \begin{tabular}{lrr}
 \multicolumn{1}{c}{\bfseries Product} &
 \multicolumn{1}{c}{\bfseries Price (ex VAT)} &
 \multicolumn{1}{c}{\bfseries Price (inc VAT)}%
 \DTLforeach*{xlsproducts}%
 {%
   \Product=Product,%
   \exPrice=Price (ex VAT),%
   \incPrice=Price (inc VAT)%
 }%
 {%
    \\\Product &
    \dtlround{\exPrice}{\exPrice}{2}\pounds\exPrice &
    \dtlround{\incPrice}{\incPrice}{2}\pounds\incPrice
 }%
 \end{tabular}
\end{table}
```

↓ Input

which produces Table 2.4. Note that the new row command \\ is put at the start of ⟨*body*⟩ to ensure a new line starts after the header entries. It's usually best to put \\ at the start of ⟨*body*⟩ as it may cause a problem if it's placed later in that argument. You can download or view the complete document.

Output

Table 2.4 Formatted Data Imported From `shop.xls`

Product	**Price (ex VAT)**	**Price (inc VAT)**
Ink cartridge	£25.00	£30.00
Mouse mat	£12.00	£14.40
USB stick	£15.00	£18.00
Pen	£2.50	£3.00

Exercise 2. Iterating Through Data

Create a document that loads the sample `people.csv` file (or the `people` SQL table) and displays the three tables shown in Tables 2.5, 2.6 and 2.7 using `\DTLforeach*`. You can download or view the solution to this exercise.

Table 2.5 Hardback Books

Id	Author	Title
5	Sir Quackalot	The Return of Duck and Goose
6	Sir Quackalot	The Adventures of Duck and Goose
10	Bor Ing	'Duck and Goose': an allegory for modern times?

Table 2.6 Paperback Books

Id	Author	Title
1	Sir Quackalot	The Adventures of Duck and Goose
2	Sir Quackalot	The Return of Duck and Goose
3	Sir Quackalot	More Fun with Duck and Goose
4	Sir Quackalot	Duck and Goose on Holiday
7	A. Parrot	My Friend is a Duck

Table 2.7 Ebooks

Id	Author	Title
8	Prof Macaw	Annotated Notes on the 'Duck and Goose' chronicles
9	Polly Parrot	'Duck and Goose' Cheat Sheet for Students

2.7.2 Iterating Over a Comma-Separated List

The etoolbox package [50] provides:

`\docsvlist{`⟨*item1,item2,...*⟩`}` Definition

This iterates over the given comma-separated list and does

`\do{`⟨*item*⟩`}` Definition

at each iteration, where ⟨*item*⟩ is the current item in the list. It's up to the user to define `\do` before using `\docsvlist`. For example:

Input

```
\renewcommand{\do}[1]{#1. }%
\docsvlist{Parrot,Canary,Zebra,Arara,Duck}
```

Input

produces:

Parrot. Canary. Zebra. Arara. Duck. Output

Alternatively you can provide your own handler instead of `\do` using

\forcsvlist{⟨handler cs⟩}{⟨list⟩} Definition

For example:

```
\newcommand{\mylistitem}[1]{#1. }%
\forcsvlist{\mylistitem}{Parrot,Canary,Zebra,Arara,Duck}
```

Input

Input

which again produces:

Parrot. Canary. Zebra. Arara. Duck. Output

⚠ The argument \docsvlist (and \forcsvlist) doesn't get expanded, so if you try:

```
\newcommand*{\mylist}{Parrot,Canary,Zebra,Arara,Duck}%
\renewcommand{\do}[1]{#1. }%
\docsvlist{\mylist}
```

Input

✘

Input

then you'll only have a list with a single item (\mylist) so you'll just have the one iteration

```
\do{\mylist}
```

Input

which just produces:

Parrot,Canary,Zebra,Arara,Duck.

Output

Instead you need to make sure the argument is expanded before it's processed by \docsvlist:

```
\newcommand*{\mylist}{Parrot,Canary,Zebra,Arara,Duck}%
\renewcommand{\do}[1]{#1. }%
\expandafter\docsvlist\expandafter{\mylist}
```

↑ Input

✔

↓ Input

The \expandafter commands may look a bit confusing but the syntax is

\expandafter ⟨*token 1*⟩⟨*token 2*⟩

Definition

This makes TEX expand ⟨*token 2*⟩ before it processes ⟨*token 1*⟩, so it's equivalent to:

⟨*token 1*⟩⟨*expansion of token 2*⟩

Therefore

```
\expandafter \docsvlist \expandafter
```

⟨*token 1*⟩ ⟨*token 2*⟩

means that TEX must expand the thing after \docsvlist before it does \docsvlist (step ① in Figure 2.9), but that thing happens to be another \expandafter:

```
\expandafter {\mylist
```

⟨*token 1*⟩ ⟨*token 2*⟩

This means that before TEX processes the left brace character { it must first expand the token after it (step ②), so that \mylist is replaced with its definition (step ③).

So TEX starts out with the first \expandafter, skips over \docsvlist ① and does the second \expandafter which makes TEX skip over the open brace ② and expand \mylist, which replaces \mylist with its definition ③.

Once \mylist has been expanded, TEX then goes back to the \docsvlist command, which is now in the form:

```
\docsvlist{Parrot,Canary,Zebra,Arara,Duck}
```

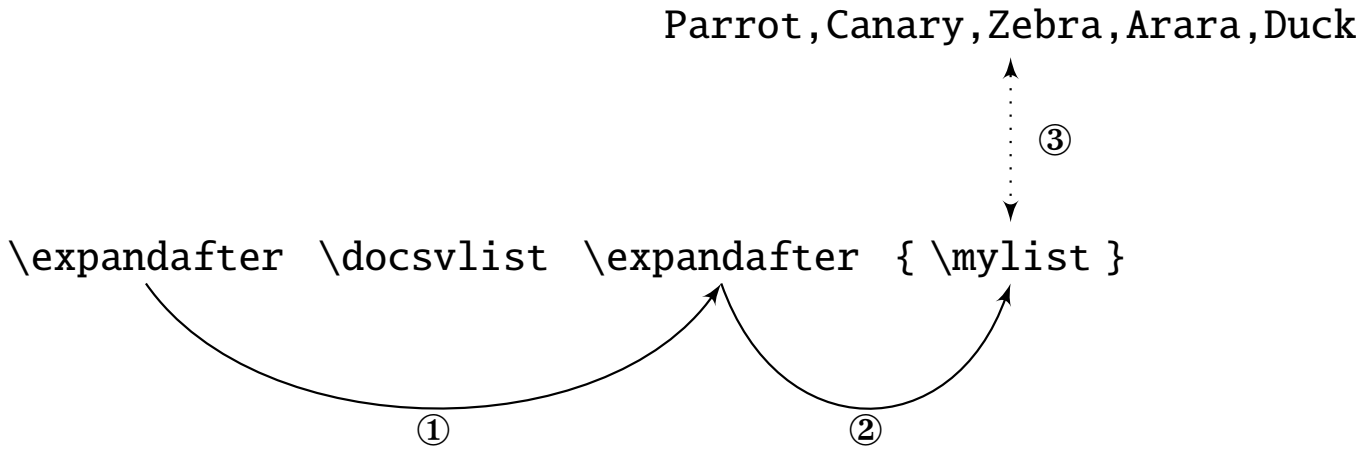


Figure 2.9 Processing \expandafter

In the case of \forcsvlist, this becomes more complicated as the list is the second argument. However, since the first argument is just a single control sequence it doesn't need to be grouped and can therefore be skipped over with another \expandafter. For example:

Input

```
\newcommand*{\mylist}{Parrot,Canary,Zebra,Arara,Duck}%
\newcommand{\mylistitem}[1]{#1. }%
\expandafter\forcsvlist\expandafter\mylistitem\expandafter{\mylist}
```

Input

The datatool package also provides some comma-separated list related commands:

\DTLifinlist{⟨*item*⟩}{⟨*list*⟩}{⟨*true*⟩}{⟨*false*⟩} Definition

This checks if the given item is in ⟨*list*⟩. A one-level expansion is performed on ⟨*list*⟩ but not on ⟨*item*⟩.

\DTLnumitemsinlist{⟨*list*⟩}{⟨*cs*⟩} Definition

This counts the number of non-empty items in ⟨*list*⟩ and stores the result in the control sequence ⟨*cs*⟩. Again, a one-level expansion is performed on ⟨*list*⟩.

Example:

↑ Input

```
\newcommand{\mylist}{Parrot,Canary,Zebra,Arara,Duck}%
Parrot
\DTLifinlist{Parrot}{\mylist}{is}{isn't}
in the list.
Number of items in the list:
\DTLnumitemsinlist{\mylist}{\numitems}\numitems.
```

↓ Input

produces:

Parrot is in the list. Number of items in the list: 5. Output

Spaces

Remember what I mentioned on page 22 about being careful of spaces? Here's an illustration of unexpected behaviour involving spaces in lists:

```
\renewcommand{\do}[1]{``#1''. }%
\docsvlist{ Parrot , Canary , Zebra , Arara , Duck }
```

Input

Input

This produces:

"Parrot ". "Canary ". "Zebra ". "Arara ". "Duck ". Output

The leading spaces have been ignored but the trailing spaces are still present.

The LaTeX kernel has a command that can also iterate through a comma-separated list but it's an internal command:

\@for⟨*cs*⟩:=⟨*list*⟩\do{⟨*body*⟩} Definition

This iterates through ⟨*list*⟩ and assigns the control sequence ⟨*cs*⟩ to the current item in the list so that it can be used as a placeholder in ⟨*body*⟩. Note that in this context \do doesn't refer to etoolbox's \do handler macro but is used as an argument delimiter (so it's like plain TEX syntax rather than LATEX syntax). Since \@for is an internal command it should only be used in a package or class file. If it has to be used in the document, it should be placed between \makeatletter and \makeatother like this:

Input

```
\makeatletter
\@for\thisitem:=Parrot,Canary,Zebra,Arara,Duck\do{\thisitem. }
\makeatother
```

Input

This produces:

Parrot. Canary. Zebra. Arara. Duck. Output

The \@for command expands the list so:

```
\newcommand*{\mylist}{Parrot,Canary,Zebra,Arara,Duck}%
\makeatletter
\@for\thisitem:=\mylist\do{\thisitem. }
\makeatother
```

produces the same result as above. However, now let's look at what happens when we introduce spaces into the list again:

```
\makeatletter
\@for\thisitem:= Parrot , Canary , Zebra , Arara , Duck \do
{``\thisitem''. }
\makeatother
```

This produces:

“ Parrot ”. “ Canary ”. “ Zebra ”. “ Arara ”. “ Duck ”. Output

In this case both the leading and trailing spaces have been retained.

The etextools package [15] also provides commands to iterate over lists. For example:

`\csvloop[`⟨*auxiliary commands*⟩`]{`⟨*list*⟩`}` Definition

The `\csvloop` macro iterates over the comma-separated list given in ⟨*list*⟩ (which may be a macro that expands to a list) and at each iteration does ⟨*auxiliary commands*⟩. If this optional argument is missing, `\do` is assumed, which behaves in the same way as with etoolbox's `\docsvlist` command. So

Input

```
\renewcommand{\do}[1]{#1. }%
\csvloop{Parrot,Canary,Zebra,Arara,Duck}
```

Input

produces

Parrot. Canary. Zebra. Arara. Duck. Output

Now let's try with spaces again:

```
\renewcommand{\do}[1]{``#1''. }%
\csvloop{ Parrot , Canary , Zebra , Arara , Duck }
```

↑ Input

↓ Input

This produces:

“ Parrot ”. “Canary ”. “Zebra ”. “Arara ”. “Duck ”.

Output

In this case the leading space has been retained on the first item, but not on any of the other items, but all the trailing spaces have been retained.

⚠ This section introduces one of etextools commands for illustrative purposes, but some of the commands in etextools and etoolbox conflict. For example, both packages define a command called `\forlistloop` but the syntax is incompatible. Since datatool automatically loads etoolbox this means that datatool and etextools may also conflict.

Another package is pgffor (part of the pgf bundle [101]) which provides:

`\foreach` ⟨*variables*⟩ `[`⟨*options*⟩`]` `in` `{`⟨*list*⟩`}{`⟨*body*⟩`}`

Definition

where ⟨*list*⟩ is either an explicit comma-separated list or a control sequence that expands to a comma-separated list. The syntax can get quite complicated, but the simplest version is in the form:

```
\foreach \thisitem in {Parrot,Canary,Zebra,Arara,Duck}
{\thisitem. }
```

Input

Input

which produces:

Parrot. Canary. Zebra. Arara. Duck.

Output

Now let's try with spaces:

```
\foreach \thisitem in { Parrot , Canary , Zebra , Arara , Duck }
{``\thisitem''. }
```

Input

Input

This produces:

"Parrot ". "Canary ". "Zebra ". "Arara ". "Duck ".

Output

which is the same result as with \docsvlist where the trailing spaces have been retained but not the leading spaces.

This is why it's important to ensure any spurious spaces are removed from comma-separated lists in your source code. It may be that some commands trim all spaces (for example in the optional argument of \usepackage or \documentclass),[6] while some commands only trim leading spaces (as with \docsvlist and \foreach) but others don't trim any spaces (such as \cite, which internally uses \@for). Remember that you can comment out space caused by the EOL character, and spaces at the start of lines are automatically discarded by TEX, so to make your code clearer you can do, for example:

Input

```
\docsvlist
{%
  Parrot,%
  Canary,%
  Zebra,%
  Arara,%
  Duck%
}
```

Input

[6] In fact, *all* spaces are stripped from the optional argument of those two commands, so it's technically possible to do, say, \usepackage[dr aft]{graphics} although I don't recommend you do this.

If you comment an EOL character that would naturally be discarded (for example, following a control sequence) there's no harm done, but if you forget to comment an unwanted EOL character, you can end up with weird spaces in your document or an error message.

Empty Items

In addition to watching out for spurious spaces, you also need to consider what happens if you have an empty item in your list. Do you want empty items skipped or should they be processed either in the same way as the other items or by displaying a missing data symbol, such as an em-dash? Let's look at how the above comma-separated list processing commands deal with this type of situation:

↑ Input

```
\renewcommand{\do}[1]{``#1''. }

\docsvlist{,Parrot,Canary,Zebra,,Arara,Duck,}

\csvloop{,Parrot,Canary,Zebra,,Arara,Duck,}

\foreach\thisitem in {,Parrot,Canary,Zebra,,Arara,Duck,}
{``\thisitem''. }
```

```
\makeatletter
\@for\thisitem:=,Parrot,Canary,Zebra,,Arara,Duck,\do
{``\thisitem''. }
\makeatother
```

↓ Input

The result is:

“Parrot”. “Canary”. “Zebra”. “Arara”. “Duck”.
“Parrot”. “Canary”. “Zebra”. “Arara”. “Duck”.
“”. “Parrot”. “Canary”. “Zebra”. “”. “Arara”. “Duck”. “”.
“”. “Parrot”. “Canary”. “Zebra”. “”. “Arara”. “Duck”. “”.

↑ Output

↓ Output

So \docsvlist and \csvloop both skip empty items but \foreach and \@for don’t.

Exercise 3. Iterating Through a List
Create a document that defines the commands:

Input

```
\newcommand*{\mylistI}{A,B,C,D}
\newcommand*{\mylistII}{a,b,c,d}
```

Input

then create the tabular environment shown in Table 2.8 using \docsvlist to construct columns 2 to 5 for each row.

Table 2.8 Lists for Iteration Exercise

List 1:	A	B	C	D
List 2:	a	b	c	d

Next, redefine \mylistII so the third item is empty and redo the tabular environment. Finally, define a command called \missingdata that does nothing and redefine \mylistII so the third item is \missingdata and redo the tabular environment.

You can download or view the solution to this exercise.

2.7.3 Iteration With etoolbox's Internal Lists

The previous section looked at iterating over a comma-separated list, but it may be that you need to first construct a list before iterating over it. This can be done efficiently via etoolbox's internal lists. These don't use a comma as a separator so it's useful for lists where items may potentially contain commas. For example, suppose I want to make a list called, say, `\mylist`, then I first need to define an empty list:

```
\newcommand*{\mylist}{}
```
Input

An item can be added to the list using:

```
\listadd{⟨list cs⟩}{⟨item⟩}
```
Definition

where ⟨*list cs*⟩ is the command used to store the list (`\mylist` in the above example) and ⟨*item*⟩ is the item to add to the list. Note that the ⟨*item*⟩ doesn't get expanded and a blank item won't be added to the list. For example:

```

\listadd{\mylist}{Parrot}
\listadd{\mylist}{Parrot, Jr}

```
Input

Input

If the item needs to be expanded before being added to the list, you can use:

\listeadd{⟨*list cs*⟩}{⟨*item*⟩} Definition

As with \listadd, a blank item won't be added to the list. For example:

```
\newcommand*{\Name}{Canary}
\listeadd{\mylist}{\Name}
```

Input

Input

There are also similar commands where you supply the name of the list macro without the leading backslash:

\listcsadd{⟨*list csname*⟩}{⟨*item*⟩} Definition

(unexpanded item) and

\listcseadd{⟨*list csname*⟩}{⟨*item*⟩} Definition

(expanded item). For example:

Input

```
\renewcommand*{\Name}{Zebra}
\listcseadd{mylist}{\Name}
\listcsadd{mylist}{Arara}
```

Input

These commands all use local assignments, so they're limited to the current scope. There are analogous commands that use global assignments:

\listgadd{⟨*list cs*⟩}{⟨*item*⟩} Definition

(global version of \listadd)

\listxadd{⟨*list cs*⟩}{⟨*item*⟩} Definition

(global version of \listeadd)

\listcsgadd{⟨*list csname*⟩}{⟨*item*⟩} Definition

(global version of \listcsadd) and

\listcsxadd{⟨*list csname*⟩}{⟨*item*⟩} Definition

(global version of \listcseadd).

You can test if an item is in a list using:

\ifinlist{⟨item⟩}{⟨list cs⟩}{⟨true⟩}{⟨false⟩} Definition

(No expansion is performed on ⟨item⟩.) For example:

↑ Input

```
Parrot
\ifinlist{Parrot}{\mylist}{is}{isn't}
in the list.
```

↓ Input

If you want to test the expansion of an item, you can use:

\xifinlist{⟨item⟩}{⟨list cs⟩}{⟨true⟩}{⟨false⟩} Definition

For example:

↑ Input

```
\newcommand*{\Name}{Parrot}%
\Name␣\xifinlist{\Name}{\mylist}{is}{isn't} in the list.
```

↓ Input

There are also analogous commands where the list control sequence name (without the leading backslash) is supplied:

`\ifinlistcs{`⟨*item*⟩`}{`⟨*list csname*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}` Definition

for the non-expanded version and

`\xifinlistcs{`⟨*item*⟩`}{`⟨*list csname*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}` Definition

for the item expansion version.

Once you've added all your items to the list, you can iterate over the list using:

`\dolistloop{`⟨*list cs*⟩`}` Definition

where ⟨*list cs*⟩ is the control sequence storing the list (`\mylist` in the above examples). If you prefer to supply the control sequence name without the leading backslash, you can use:

`\dolistcsloop{`⟨*list csname*⟩`}` Definition

Both these commands use `\do{`⟨*item*⟩`}` at each iteration, in the same way as for `\docsvlist` described earlier. For example:

↑ Input

```
\newcommand*{mylist}{}%
\listadd{\mylist}{Parrot}%
\listadd{\mylist}{Canary}%
\listadd{\mylist}{Zebra}%
\listadd{\mylist}{Arara}%
\listadd{\mylist}{Duck}%
\renewcommand*{\do}[1]{#1. }%
\dolistloop{\mylist}
```

Input

produces:

Parrot. Canary. Zebra. Arara. Duck.

Output

Alternatively, you can provide your own handler instead of using \do:

\forlistloop{⟨*handler cs*⟩}{⟨*list cs*⟩}

Definition

where ⟨*handler cs*⟩ is the command to use on each iteration of the list and ⟨*list cs*⟩ is the list control sequence. If you prefer to supply the list control sequence name without the leading backslash you can use:

\forlistcsloop{⟨*handler cs*⟩}{⟨*list csname*⟩}

Definition

Exercise 4. Internal Lists

Create a document that loads the sample booklist.csv file or the books SQL table. Then create an internal list that contains a list of all the book titles, without repetition. For example, if a title has both a hardback and paperback edition only add that title once rather than twice.

To test the list, iterate through it and display each item of the list. (If you like, you can just use a paragraph break between items rather than using a tabular environment.) The result should look like:

↑ Output

The Adventures of Duck and Goose
The Return of Duck and Goose
More Fun with Duck and Goose
Duck and Goose on Holiday
My Friend is a Duck
Annotated Notes on the 'Duck and Goose' chronicles
'Duck and Goose' Cheat Sheet for Students
'Duck and Goose': an allegory for modern times?

↓ Output

You can download or view the solution to this exercise.

2.7.4 General Iteration with TEX's \loop

If you want a more general purpose way of repeating a block of code, you can use

\loop ⟨*code*⟩\if... \repeat

Definition

This repeats ⟨*code*⟩ while the given condition is true. The \if... part should be one of TEX's conditionals, such as \ifnum, without the terminating \fi.

Example:
For example, to print the numbers from 1 to 10:

Input

```
\newcount\mycount
\loop
 \advance\mycount by 1\relax
 \the\mycount.
\ifnum\mycount<10
\repeat
```

Input

This produces:

1. 2. 3. 4. 5. 6. 7. 8. 9. 10. Output

2.7.5 Iteration Tips and Tricks

This section describes some advanced techniques that you may or may not need to know, so feel free to skip it.

Recall from §2.7.1 I had:

```
\DTLforeach*{people}{\Surname=surname}{\Surname. }
```
Input

which, when using the sample `people.csv` file, produced:

Parrot. Canary. Zebra. Arara. Duck. Canary. Output

Suppose instead I wanted to produce:

Parrot; Canary; Zebra; Arara; Duck; Canary. Output

If I try:

```
\DTLforeach*{people}{\Surname=surname}{\Surname; }.
```
Input

I get:

Parrot; Canary; Zebra; Arara; Duck; Canary; . Output

(there's an unwanted semi-colon and space before the terminating full stop) and if I try:

```
\DTLforeach*{people}{\Surname=surname}{; \Surname}. Input
```

I get:

; Parrot; Canary; Zebra; Arara; Duck; Canary. Output

(The unwanted semi-colon and space are now at the start.)

Neither or these are quite right. Here's a way of achieving the desired output:

```
\newcommand{\surnamesep}{%
  \renewcommand{\surnamesep}{; }%
}%
\DTLforeach*{people}{\Surname=surname}{\surnamesep\Surname}.
```

↑ Input

↓ Input

This may look a bit weird at first sight, but here's how it works:

- On the first iteration

```
\surnamesep\Surname
```
Input

is equivalent to

```
\renewcommand{\surnamesep}{; }\Surname
```
Input

That is, \surnamesep is redefined to ;␣ (semicolon followed by a space) without displaying anything and the first surname is printed.

- On the next iteration

```
\surnamesep\Surname
```
Input

is equivalent to just

```
; \Surname
```
Input

(because \surnamesep has just been redefined to ;␣) so a semi-colon followed by a space followed by the second surname is printed. Since \surnamesep doesn't get redefined any more, it remains the same for the rest of the loop.

⚠ Be careful if the contents of ⟨*body*⟩ are localised (for example, if it's in a tabular environment, as in Example 6) since \renewcommand only has a local effect. Instead you can use TEX's \gdef command described in §2.1.1:

Input

```
\newcommand{\surnamesep}{%
  \gdef\surnamesep{; }%
}%
\DTLforeach*{people}{\Surname=surname}{\surnamesep\Surname}.
```

Input

Alternatively you can do a global \let after you've redefined the command:

Input

```
\newcommand{\surnamesep}{%
  \renewcommand\surnamesep{; }%
  \global\let\surnamesep\surnamesep
}%
\DTLforeach*{people}{\Surname=surname}{\surnamesep\Surname}.
```

Input

The datatool package defines the command:

\DTLiffirstrow{⟨*true*⟩}{⟨*false*⟩} Definition

designed for use within the ⟨*body*⟩ of \DTLforeach so I could just do:

Input

```
\DTLforeach*
 {people}% database
 {\Surname=surname}% assignment list
 {\DTLiffirstrow{}{; }\Surname}.
```

Input

However, the technique described above can be used in more general situations. For example, suppose I want to use etoolbox's \docsvlist, described in §2.7.2, I could do:

Input

```
\newcommand{\surnamesep}{%
  \renewcommand{\surnamesep}{; }%
}%
\renewcommand\do[1]{\surnamesep#1}%
\docsvlist{Parrot,Canary,Zebra,Arara,Duck}.
```

Input

which produces:

Parrot; Canary; Zebra; Arara; Duck. Output

The datatool package also defines the command:

\DTLiflastrow{⟨*true*⟩}{⟨*false*⟩} Definition

As with \DTLiffirstrow, this command is designed for use within the ⟨*body*⟩ of \DTLforeach (or \DTLforeach*). For example:

↑ Input

```
\DTLforeach*
 {people}% database
 {\Surname=surname}% assignment list
 {%
  \DTLiffirstrow{}{\DTLiflastrow{ and }{, }}%
  \Surname
 }.
```

↓ Input

produces:

Parrot, Canary, Zebra, Arara, Duck and Canary. Output

So how can we do the equivalent for a general comma-separated list rather than using \DTLforeach? One possible method is described in the example below.

Example 7. List of Names

This is slightly more complicated but it uses a similar technique to earlier:

Input

```
\newcommand*{\surnamesep}{}%
\newcommand*{\lastsurname}{}%
\newcommand*{\prelastsurname}{}%
\renewcommand*{\do}[1]{%
  \surnamesep
  \lastsurname
  \renewcommand{\lastsurname}{%
    \renewcommand{\surnamesep}{, }%
    \renewcommand{\prelastsurname}{ and }%
    #1%
  }%
}%
\docsvlist{Parrot,Canary,Zebra,Arara,Duck}%
```

```
\prelastsurname \lastsurname
```

↓ Input

This produces:

Parrot, Canary, Zebra, Arara and Duck

Output

Here's how it works:

1. On the first iteration (`\do{Parrot}`) `\surnamesep` and `\lastsurname` do nothing. Then `\lastsurname` is redefined via:

   ```
   \renewcommand{\lastsurname}{%
     \renewcommand{\surnamesep}{, }%
     \renewcommand{\prelastsurname}{ and }%
     Parrot%
   }
   ```

   (The `#1` has been replaced with the argument passed to `\do`.) So far nothing has been displayed.

   If this was the only item in the list, the loop would end and then:

   - `\prelastsurname` would be done, but this is currently nothing.

- \lastsurname would be done, which is now set to redefine a couple of commands that are no longer needed (\surnamesep and \prelastsurname) and then displays "Parrot".

Therefore, if the list only has one element, it just displays that element. However, since the list has more than one element, we have nothing displayed and we move on to the next item in the list.

2. On the second iteration (\do{Canary}) \surnamesep still does nothing but \lastsurname now does:

```
\renewcommand{\surnamesep}{, }%
\renewcommand{\prelastsurname}{ and }%
Parrot%
```

So \surnamesep gets redefined to do ,␣ (that is a comma followed by a space) and \prelastsurname gets redefined to do ␣and␣. After these redefinitions, the word "Parrot" is then displayed. Next \lastsurname is redefined via:

```
\renewcommand{\lastsurname}{%
  \renewcommand{\surnamesep}{, }%
  \renewcommand{\prelastsurname}{ and }%
```

```
  Canary%
}
```

Therefore, by the end of the second iteration, the only text displayed is "Parrot". If this happened to be the last item in the list, the loop would end and then:

- \prelastsurname would be done, which now displays " and ".
- \lastsurname would be done, which redefines some commands we no longer need (\surnamesep and \prelastsurname) and then displays "Canary".

Therefore, if the list only contained Parrot,Canary then just the text "Parrot and Canary" would be displayed. However, as there are still more items left, the only text displayed so far is "Parrot".

3. On the third iteration (\do{Zebra}) \surnamesep now displays a comma followed by a space and \lastsurname does:

```
\renewcommand{\surnamesep}{, }%
\renewcommand{\prelastsurname}{ and }%
Canary%
```

Then \lastsurname is redefined via:

```
\renewcommand{\lastsurname}{%
  \renewcommand{\surnamesep}{, }%
  \renewcommand{\prelastsurname}{ and }%
  Zebra%
}
```

So we have thus far produced the text "Parrot, Canary". If this happened to be the last item in the list, the loop would end and then:

- \prelastsurname would display " and "
- \lastsurname would redefine some commands that we no longer need (\surnamesep and \prelastsurname) and then display "Zebra".

The remaining iterations follow the same pattern as this third iteration.

If you plan to use this method more than once, you might prefer to define a new command. For example:

↑ Input

```
% set up defaults so we don't get an error
% when we try to redefine these commands
\newcommand*{\surnamesep}{}%
\newcommand*{\lastsurname}{}%
\newcommand*{\prelastsurname}{}%
% define the new command to process a list of names:
\newcommand*{\displaynames}[1]{%
  % initialise:
  \renewcommand*{\surnamesep}{}%
  \renewcommand*{\lastsurname}{}%
  \renewcommand*{\prelastsurname}{}%
  % set up list handler:
  \renewcommand*{\do}[1]{%
    \surnamesep
    \lastsurname
    \renewcommand{\lastsurname}{%
      \renewcommand{\surnamesep}{, }%
      \renewcommand{\prelastsurname}{ and }%
      ##1%
    }%
```

```
  }%
  \docsvlist{#1}%
  \prelastsurname \lastsurname
}
```

Input

Since we have nested definitions of commands that take a parameter we need to be careful how we reference the parameter. In the above #1 refers to the argument of \displaynames (the outer command) and ##1 refers to the argument of \do (the inner command).

In this case, it's better to define your own handler macro and use \forlistloop instead of \docsvlist:

Input

```
% set up defaults so we don't get an error
% when we try to redefine these commands
\newcommand*{\surnamesep}{}%
\newcommand*{\lastsurname}{}%
\newcommand*{\prelastsurname}{}%
% define the handler macro:
\newcommand*{\dodisplayname}[1]{%
  \surnamesep
```

```
  \lastsurname
  \renewcommand{\lastsurname}{%
    \renewcommand{\surnamesep}{, }%
    \renewcommand{\prelastsurname}{ and }%
    #1%
  }%
}%
% define the new command to process a list of names:
\newcommand*{\displaynames}[1]{%
  % initialise:
  \renewcommand*{\surnamesep}{}%
  \renewcommand*{\lastsurname}{}%
  \renewcommand*{\prelastsurname}{}%
  % iterate through list:
  \forcsvlist{\dodisplayname}{#1}%
  % finish off:
  \prelastsurname \lastsurname
}
```

↓ Input

This removes one of the nested redefinitions and the need to use ##1 instead of #1.

You can download or view a complete document.

Exercise 5. Oxford Comma

Some people always use the Oxford comma, some people never use it, and some people only use it where its lack would cause ambiguity. The purpose of this exercise is not to engage in a heated debate over whether or not it should be used. It's simply an exercise in iteration techniques. For those who've never heard of the Oxford comma, it's a comma that's placed after the penultimate item in a list of three or more items before the "and". For example: Parrot, Canary, and Zebra.

For this exercise, see if you can adapt the definition of `\displaynames` from the end of the previous example so that it uses the Oxford comma. To test it:

↑ Input

```
\displaynames{Parrot,Canary,Zebra,Arara,Duck}

\displaynames{Parrot,Canary,Zebra,Arara}

\displaynames{Parrot,Canary,Zebra}

\displaynames{Parrot,Canary}
```

```
\displaynames{Parrot}
```

Input

should produce:

Output

Parrot, Canary, Zebra, Arara, and Duck
Parrot, Canary, Zebra, and Arara
Parrot, Canary, and Zebra
Parrot and Canary
Parrot

Output

You can download or view the solution to this exercise.

2.8 Fetching Data From a Given Row

It may be that you don't want to iterate through the entire data but just want to fetch information from a particular row. The datatool package provides a number of ways to do this, but this book is just going to cover three commands:

\DTLassign{⟨db-name⟩}{⟨row-idx⟩}{⟨assign list⟩} Definition

This applies the assignment list to the row given by ⟨*row-idx*⟩. (Indices start from 1.)

\DTLassignfirstmatch{⟨db-name⟩}{⟨col-label⟩}{⟨value⟩}{⟨assign list⟩} Definition

This applies the assignment list to the first row where the entry in the column identified by ⟨*col-label*⟩ exactly matches the given value. Note that no expansion is performed on ⟨*value*⟩.

\xDTLassignfirstmatch{⟨db-name⟩}{⟨col-label⟩}{⟨value⟩}{⟨assign list⟩} Definition

This applies the assignment list to the first row where the entry in the column identified by ⟨*col-label*⟩ exactly matches a *one-level expansion* of ⟨*value*⟩.

In each case, ⟨*db-name*⟩ is the label identifying the data and ⟨*assign list*⟩ is the comma-separated list of assignments, as used by \DTLforeach and \DTLforeach*.

Example 8. Fetching the Data From Row 1

Suppose I just want information from the first row of data in my sample people.csv file. Then I can use \DTLassign, like this:

Input

```
\DTLassign{people}{1}{%
  \Surname=surname,%
  \Title=title,%
  \AddressI=address1,%
  \AddressII=address2,%
  \Town=town,%
  \County=county,%
  \Postcode=postcode%
}
```

Input

Remember to make sure you comment out the unwanted EOL characters, as shown above, or you'll get an error caused by spurious spaces (recall the note about spaces on page 162). Now that the data has been fetched, it can be used. For example, to just display the details in a tabular environment:

Input

```
\begin{tabular}{l}
\Title\␣\Surname\\
\AddressI\\
```

```
\AddressII\\
\Town\\
\County\\
\Postcode
\end{tabular}
```

↓ Input

This produces:

↑ Output

Miss Parrot
42 The Lane

Some Town
Noshire
AB1 2XY

↓ Output

(You can download or view a complete document.)

Note that there is a blank entry caused by missing data in the `address2` column. If this example was changed to use the `people` SQL table instead, the result would appear as:

```
Miss Parrot
42 The Lane
NULL
Some Town
Noshire
AB1 2XY
```

↑ Output

↓ Output

See §2.9 on how to deal with null or empty entries. (You can also download or view a complete document for the SQL version.)

Remember that if you're importing your data from a SQL database, there's no need to import all the data from the table if you don't require parts of it. Instead you can filter out all the unwanted rows in your SELECT statement. For example, if you wanted to fetch the data for just the customer whose surname is "Parrot", you can do:

```
datatooltk --output customer.dbtex --sqldb samples --sqluser
sampleuser --sql "SELECT * FROM people WHERE surname='Parrot'"
```

Shell

If you're not using SQL then you can fetch the relevant row using the afore mentioned \DTLassignfirstmatch, but it's less efficient.

Example 9. Fetching a Customer's Details
Suppose you only want the details from the customer whose surname matches "Parrot" in the sample people.csv file. This can be fetched using:

↑ Input

```
\DTLassignfirstmatch{people}{surname}{Parrot}{%
  \Surname=surname,%
  \Title=title,%
  \AddressI=address1,%
  \AddressII=address2,%
  \Town=town,%
  \County=county,%
  \Postcode=postcode%
}
```

↓ Input

Now the details have been fetched, it can be used as in the previous example:

Input

```
\begin{tabular}{l}
\Title\␣\Surname\\
\AddressI\\
\AddressII\\
\Town\\
\County\\
\Postcode
\end{tabular}
```

Input

The result is the same as for the previous example. (You can download or view a complete document.)

Remember that \DTLassignfirstmatch performs an exact match without expansion. This means that if you do something like:

Input

```
\newcommand{\Name}{Parrot}
\DTLassignfirstmatch{people}{surname}{\Name}%
{%
```

```
  \Surname=surname,%
  \Title=title,%
  \AddressI=address1,%
  \AddressII=address2,%
  \Town=town,%
  \County=county,%
  \Postcode=postcode%
}
```

↓ Input

Then you'll get an error that no match was found. This is because you're effectively asking TEX to find an entry that contains "\Name", but that control sequence doesn't appear in any of the entries, so there's no match. Instead, you need to use \xDTLassignfirstmatch which will internally replace \Name with its definition ("Parrot").

↑ Input

```
\newcommand{\Name}{Parrot}
\xDTLassignfirstmatch{people}{surname}{\Name}%
{%
  \Surname=surname,%
  \Title=title,%
```

✔

```
  \AddressI=address1,%
  \AddressII=address2,%
  \Town=town,%
  \County=county,%
  \Postcode=postcode%
}
```

↓ Input

Example 10. Fetching a Customer's Details (With Expansion)
In Example 8, I didn't access the country from the data. Let's modify that example so that it fetches the complete address for "Polly Parrot":

↑ Input

```
\DTLassignfirstmatch{people}{surname}{Parrot}{%
  \Surname=surname,%
  \Title=title,%
  \AddressI=address1,%
  \Town=town,%
  \County=county,%
  \Postcode=postcode,%
  \CountryCode=country%
}
```

↓ Input

Now let's try displaying the information:

Input

```
\begin{tabular}{l}
\Title\␣\Surname\\
\AddressI\\
\Town\\
\County\\
\Postcode\\
\CountyCode
\end{tabular}
```

Input

This produces:

Output

Miss Parrot
42 The Lane
Some Town
Noshire
AB1 2XY
gb

Output

If this is intended for, say, a letter to a customer, then the country code really needs to be converted to the country's name. That information is stored in the sample country-codes.csv file, so that also needs to be loaded. Therefore the document should have:

Input

```
\DTLloaddb{people}{people.csv}
\DTLloaddb{countries}{country-codes.csv}
```

Input

Once the \CountryCode has been assigned via \DTLassign, as shown above, the code can be converted to a name:

Input

```
\xDTLassignfirstmatch{countries}{code}%
{\CountryCode}{\CountryName=name}
```

Input

Now the address can be displayed including the country name:

Output

Miss Parrot
42 The Lane
Some Town
Noshire
AB1 2XY
United Kingdom

Output

You can download or view this example.

Remember that if you're using SQL, it's much simpler to combine and filter using the SELECT statement:

```
datatooltk --output customer.dbtex --sqldb samples
--sqluser sampleuser --sql "SELECT title, surname, address1,
address2, town, county, countries.name AS country, postcode
FROM people, countries WHERE surname='Parrot' AND
people.country = countries.code"
```

Shell

This creates a datatool (.dbtex) file called customer.dbtex that only contains the one row of data. The country name is now stored in the column labelled country. So you can just do:

↑ Input

```
\DTLloaddbtex{customer}{customer.dbtex}
\DTLassign{customer}{1}{%
  \Surname=surname,%
  \Title=title,%
  \AddressI=address1,%
  \Town=town,%
  \County=county,%
  \Postcode=postcode,%
  \CountryName=country%
}
```

```
\begin{tabular}{l}
\Title\␣\Surname\\
\AddressI\\
\Town\\
\County\\
\Postcode\\
\CountyName
\end{tabular}
```

↓ Input

This produces the sample result as above. You can download or view this SQL version.

Exercise 6. Fetching a Row of Data

Modify the code from Example 10 so that it fetches the address for Fred Canary from sample `people.csv` file.

You can download or view the solution to this exercise.

2.9 Null and Boolean Values

Recall from Example 8 that there is a difference between the sample people.csv file and the corresponding people SQL table. Some of the entries in the SQL table have null values in the address2 column whereas in the CSV file the values are empty rather than null. You can test for a null value using:

\DTLifnull{⟨*cs*⟩}{⟨*true*⟩}{⟨*false*⟩} Definition

where ⟨*cs*⟩ is a control sequence, ⟨*true*⟩ is what to do if ⟨*cs*⟩ is null and ⟨*false*⟩ is what to do if ⟨*cs*⟩ isn't null. To test for an empty value, you can use:

\ifdefempty{⟨*cs*⟩}{⟨*true*⟩}{⟨*false*⟩} Definition

which is provided by the etoolbox package (automatically loaded by datatool). The datatool package provides the command:

\DTLifnullorempty{⟨*cs*⟩}{⟨*true*⟩}{⟨*false*⟩} Definition

which is just a short cut for:

\ifdefempty{⟨*cs*⟩}{⟨*true*⟩}{\DTLifnull{⟨*cs*⟩}{⟨*true*⟩}{⟨*false*⟩}} Input

The other difference is the way boolean values have been stored. Both the `people.csv` file and the SQL `people` table used "1" to indicate a true value and "0" to indicate a false value in the `subscribed` field, but when the data was fetched from the SQL table, these values were converted to "true" and "false" in the datatool (`.dbtex`) file. There are a number of ways of testing whether a control sequence is equal to "true" or "1". For example, the etoolbox package defines:

`\ifdefstring{`⟨*cs*⟩`}{`⟨*string*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}` Definition

This tests if the control sequence ⟨*cs*⟩ is defined to be ⟨*string*⟩. If it is, ⟨*true part*⟩ is done, otherwise ⟨*false part*⟩ is done. For example:

Input

```
\newcommand*{\Subscribed}{true}%
\ifdefstring{\Subscribed}{true}{Yes}{No}
```

Input

produces:

Yes Output

Similarly

```
\newcommand*{\Subscribed}{1}%
\ifdefstring{\Subscribed}{1}{Yes}{No}
```

Input

Input

produces:

Yes

Output

If you want to test for “true” or “1”, you can combine these:

```
\ifdefstring
 {\Subscribed}{true}% test
 {Yes}% condition true
 {\ifdefstring{\Subscribed}{1}{Yes}{No}}
```

Input

Input

Alternatively you can use another etoolbox command:

\ifboolexpr{⟨*expression*⟩}{⟨*true part*⟩}{⟨*false part*⟩}

Definition

which evaluates ⟨*expression*⟩ and does ⟨*true part*⟩ if true, otherwise it does ⟨*false part*⟩. In this case I’m going to use the `test` syntax:

↑ Input

```
\ifboolexpr
{
  test{\ifdefstring{\Subscribed}{true}} or
  test{\ifdefstring{\Subscribed}{1}}
}
{Yes}{No}
```

↓ Input

(For further details of the syntax used in ⟨*expression*⟩, see the etoolbox documentation [50].) You might find it easier to define a command to do this. For example:

↑ Input

```
\newcommand{\ifcsbool}[3]{%
  \ifboolexpr
  {
    test{\ifdefstring{#1}{true}} or
    test{\ifdefstring{#1}{1}}
  }
  {#2}{#3}%
}
```

↓ Input

This has the syntax:

\ifcsbool{⟨*cs*⟩}{⟨*true*⟩}{⟨*false*⟩}

where ⟨*cs*⟩ is a control sequence. For example:

```
\ifcsbool{\Subscribed}{Yes}{No}
```
Input

or (recall from Volume 1 [92, §8.2] the \ding command provided by pifont [83]):

```
\ifcsbool{\Subscribed}{\ding{52}}{\ding{56}}
```
Input

Example 11. Display Customer List (Null Values)
To illustrate the difference between null and empty values, let's first look at what happens if we load the sample people.csv file and display the data using \DTLdisplaydb:

↑ Input

```
\DTLloaddb{people}{people.csv}

\begin{table}
  \caption{Customers (CSV)}
  \label{tab:peoplecsv}
```

```
  \centering
  \DTLdisplaydb{people}
  [id,forenames,title,country,postcode,gender,dob]% omit these columns
  {people}
\end{table}
```

Input

This produces Table 2.9. Now let's look at what happens if we fetch the people SQL table using:

```
datatooltk --output people.dbtex --sqluser sampleuser
--sqldb samples --sql "SELECT * FROM people"
```

Shell

and load the resulting people.dbtex file:

```
\DTLloaddbtex{\people}{people.dbtex}
```

Input

Now display the data:

Input

```
\begin{table}
  \caption{Customers (SQL)}
  \label{tab:peoplesql}
```

```
  \centering
  \DTLdisplaydb
  [id,forenames,title,country,postcode,gender,dob]% omit these columns
  {\people}
\end{table}
```

↓ Input

This produces Table 2.10.

You can use \DTLifnull or \ifdefempty to check for missing entries in the SQL or CSV data, respectively. Alternatively, use \DTLifnullorempty to adapt for either case. However, we now can't simply use \DTLdisplaydb but we can use \DTLforeach* to typeset the table instead:

↑ Input

```
\begin{table}
  \caption{Customers (Check for Null and Boolean)}
  \label{tab:peoplenullcheck}
  \centering
  \begin{tabular}{lllllc}
    \multicolumn{1}{c}{\bfseries Surname} &
    \multicolumn{1}{c}{\bfseries Address 1} &
    \multicolumn{1}{c}{\bfseries Address 2} &
```

```
    \multicolumn{1}{c}{\bfseries Town} &
    \multicolumn{1}{c}{\bfseries County} &
    \multicolumn{1}{c}{\bfseries Subscribed}%
    \DTLforeach*{people}{\Surname=surname,\AddressI=address1,%
    \AddressII=address2,\Town=town,\County=county,\Subscribed=subscribed}
    {%
      \\\Surname & \AddressI &
      \DTLifnullorempty{\AddressII}{\multicolumn{1}{c}{---}}{\AddressII}
      & \Town &
      \DTLifnullorempty{\County}{\multicolumn{1}{c}{---}}{\County}&
      \ifcsbool{\Subscribed}{\ding{52}}{\ding{56}}%
    }%
  \end{tabular}
\end{table}
```

Input

(Recall \ifcsbool from above.) This produces Table 2.11.

You can download or view a complete document.

Table 2.9 Customers (CSV)

surname	address1	address2	town	county	subscribed
Parrot	42 The Lane		Some Town	Noshire	1
Canary	24 The Street	Some Village	Some Town	Noshire	0
Zebra	856 The Avenue		Some City	CA	1
Arara	Nenhuma Rua		São Paulo		1
Duck	1 The Street	Another Village	Some City	Imagineshire	0
Canary	24 The Street	Some Village	Some Town	Noshire	1

Table 2.10 Customers (SQL)

surname	address1	address2	town	county	subscribed
Parrot	42 The Lane	NULL	Some Town	Noshire	true
Canary	24 The Street	Some Village	Some Town	Noshire	false
Zebra	856 The Avenue	NULL	Some City	CA	true
Arara	Nenhuma Rua	NULL	São Paulo	NULL	true
Duck	1 The Street	Another Village	Some City	Imagineshire	false
Canary	24 The Street	Some Village	Some Town	Noshire	true

Table 2.11 Customers (Check for Null or Empty and Boolean)

Surname	Address 1	Address 2	Town	County	Subscribed
Parrot	42 The Lane	—	Some Town	Noshire	✔
Canary	24 The Street	Some Village	Some Town	Noshire	✘
Zebra	856 The Avenue	—	Some City	CA	✔
Arara	Nenhuma Rua	—	São Paulo	—	✔
Duck	1 The Street	Another Village	Some City	Imagineshire	✘
Canary	24 The Street	Some Village	Some Town	Noshire	✔

3. Correspondence

This chapter covers writing letters (including mail merging), envelope labels and faxes. There are a number of classes you can use for this purpose. See, for example, the letter topic. (If you're writing in German, you may want to consider dinbrief [8].)

[FAQ: Letters and the like]

For brevity, this book will only look at four such classes:

letter This is one of the base LaTeX classes that comes with all TeX distributions. It's not very flexible, but it's simple to use and may suit your purposes if you're happy with the default layout. Since it's one of the base classes, it's stable and shouldn't conflict with common packages.

scrlttr2 This is one of the KOMA-Script classes and is included here because the first two volumes in this series looked at other classes in that bundle. This is more complicated to use than letter but is more flexible. At the time of writing this, the current version of the scrlttr2 class is 3.14 (2014-10-28).

newlfm This class is less flexible than scrlttr2 but can also be used for memos (see §6.1) and faxes. At the time of writing this, the current version of the newlfm class is dated 2009-04-11.

isodoc This class is more flexible than newlfm but less complicated than scrlttr2 and can also be used to write invoices (see §4.1). At the time of writing this, the current version of the isodoc class is 1.06 (2014-07-26).

⚠ Note that it's usually considered inappropriate to have floats within a letter, so letter-style classes, such as letter, scrlttr2 and newlfm, don't define the figure or table environments. You can, however, just use a tabular environment. In this case, you might want to place the tabular environment within the center environment to centre it and produce a small vertical gap above and below. The isodoc class inherits the figure and table environments from the underlying base article class, but that kind of floating material is liable to interfere with the fixed blocks such as the address information.

3.1 Writing a Letter Using the letter Class

This section describes how to use the basic letter class. As with all the base LaTeX classes, the paper size defaults to the US letter size, but this can be

changed via the class options. For example, for A4 paper you can use the a4paper option:

```
\documentclass[a4paper]{letter}
```

Input

or you can use the geometry package [109]:

```
\documentclass{letter}
\usepackage[a4paper]{geometry}
```

Input

A document using the letter class may have one or more letter environments that contain the text of the letter:

```
\begin{letter}{⟨recipient's address⟩}
⟨body⟩
\end{letter}
```

Definition

Within ⟨*body*⟩, you can use:

```
\opening{⟨salutation text⟩}
```

Definition

for the letter greeting, and

`\closing{`⟨*closing text*⟩`}` Definition

for the closing text (such as "Yours Sincerely"). After `\closing` you can also use:

`\ps` ⟨*postscript text*⟩ Definition

for any postscripts (note that ⟨*postscript text*⟩ isn't an argument and you need to supply your own "PS" tag at the start of it)

`\cc{`⟨*text*⟩`}` Definition

to indicate a list of people to be cc'd, and

`\encl{`⟨*text*⟩`}` Definition

to specify a list of any enclosures.

Example 12. Writing a Simple Letter (letter class)
Here is a simple letter:

↑ Input

```
\documentclass[12pt]{letter}

\usepackage[a4paper]{geometry}
```

```
\usepackage[british]{babel}

\begin{document}

\begin{letter}{Mrs Mabel Canary\\24 The Street\\
Some Village\\Some Town\\Noshire\\AB1 2YZ}

  \opening{Dear Mrs Canary}

  This is an imaginary letter.

  This is the second paragraph of the letter.

  \closing{Yours sincerely}

  \ps PS: this is a postscript.

  \encl{Photocopy of something interesting\\
   Photocopy of something rather dull}

  \cc{Prof Important Person\\Dr Bor Ing}
\end{letter}
```

```
\end{document}
```

↓ Input

(I've used the babel package [7] with the `british` option to ensure that the date is displayed using the British format rather than the default US format.)

The resulting document is shown in Figure 3.1. You can download or view this example.

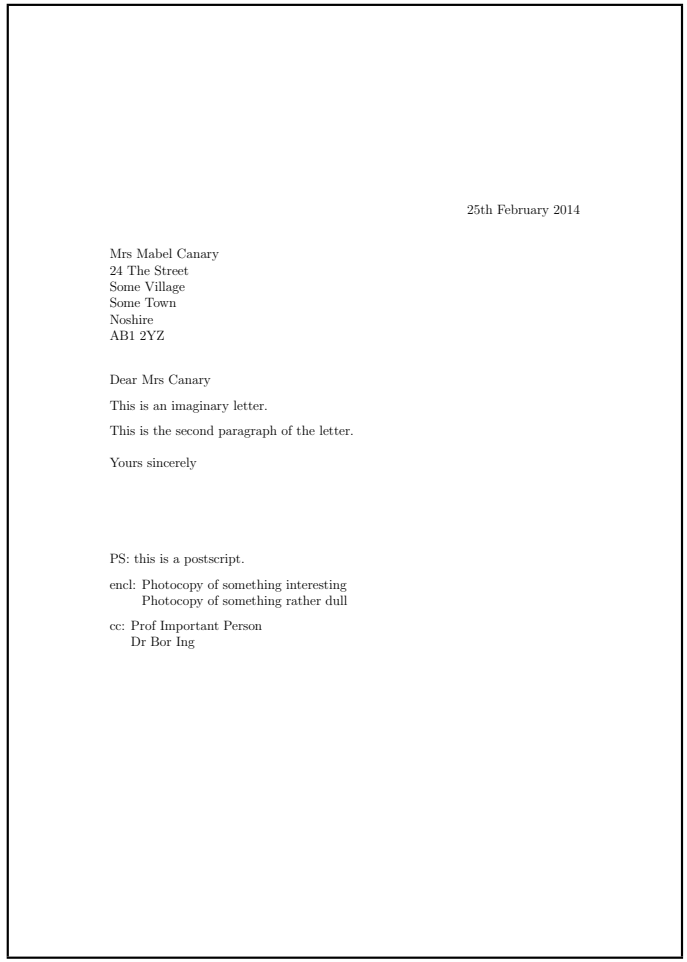
You can add information about the sender as well. This typically goes in the preamble. The sender's name is specified using:

`\name{`⟨*text*⟩`}` Definition

Additionally, you can specify the sender's name as it should appear after the closing text (supplied by `\closing`):

`\signature{`⟨*text*⟩`}` Definition

The sender's address is specified using:



25th February 2014

Mrs Mabel Canary
24 The Street
Some Village
Some Town
Noshire
AB1 2YZ

Dear Mrs Canary

This is an imaginary letter.

This is the second paragraph of the letter.

Yours sincerely

PS: this is a postscript.

encl: Photocopy of something interesting
Photocopy of something rather dull

cc: Prof Important Person
Dr Bor Ing

Figure 3.1 A Simple Letter Using the letter Class

\address{⟨*text*⟩} Definition

You may use \\ within ⟨*text*⟩ to separate the lines of the address. The sender's telephone number is specified using:

\telephone{⟨*text*⟩} Definition

Additional location information for the sender is specified using:

\location{⟨*text*⟩} Definition

By default, the location and telephone number information is placed on the footer of the first page if no sender address has been specified. If you want to specify the sender address as well, you need to use

```
\thispagestyle{firstpage}
```
Input

after \opening.

The letter class is very old and doesn't provide a means for specifying modern communication methods such as email, mobile phone numbers or web addresses.

Exercise 7. Writing a Letter (letter class)
Modify Example 12 to include sender details.

For the More Adventurous

Recall from §2.8 that you can fetch a single row of data from a database. Use one of the commands described in that section to fetch the recipient's details from the sample `people.csv` file or the `people` SQL table rather than explicitly typing them into the document. You can download or view a solution.

3.2 Writing a Letter Using the scrlttr2 Class

This section describes how to use the scrlttr2 KOMA-Script class [46] to write letters. (Don't be confused if you see a class called scrlettr. That's an older deprecated class.)

The scrlttr2 class defines the letter environment:

`\begin{letter}[`⟨*options*⟩`]{`⟨*addressee*⟩`}` Definition

where ⟨*addressee*⟩ is the recipient's postal details. Use `\\` to separate the lines of the address. As with the letter class described above, a document may have more than one letter environment, which will later come in useful when we look at mail merging in §3.5. Within the letter environment you must start the letter with

\opening{⟨*salutation*⟩} Definition

and close with

\closing{⟨*sign-off text*⟩} Definition

The ⟨*salutation*⟩ is the greeting at the start of the letter, such as `Dear Dr~Smith` and the ⟨*sign-off text*⟩ is the closing text, such as `Yours sincerely`.

After \closing{⟨*sign-off text*⟩} you can optionally use:

\ps ⟨*postscript text*⟩ Definition

(As with the letter class, the ⟨*postscript text*⟩ is not an argument of \ps, and you need to include "PS" in ⟨*postscript text*⟩ if you want it as it's not automatically generated.)

\encl{⟨*enclosures info*⟩} Definition

and

\cc{⟨*cc list*⟩} Definition

These three commands, if required, must go after \closing and before the end of the letter environment.

⚠ The scrlttr2 class is designed for use with the babel package [7], so remember to load it. (See Volume 1 [92, §5.8].)

Example 13. A Simple Letter (scrlttr2 class)
Here's a simple example letter:

↑ Input

```
\documentclass{scrlttr2}

\usepackage[british]{babel}

\begin{document}

\begin{letter}{Mrs Mabel Canary\\24 The Street\\Some Village\\
Some Town\\Noshire\\AB1 2YZ}
  \opening{Dear Mrs~Canary}

  This is an imaginary letter.

  This is the second paragraph of the letter.

  \closing{Yours sincerely}

  \ps PS: this is a postscript.
```

```
  \encl{Photocopy of something interesting\\
   Photocopy of something rather dull}

  \cc{Prof Important Person\\
   Dr Bor Ing}
\end{letter}

\end{document}
```

↓ Input

(You can download or view this document.) The resulting document is shown in Figure 3.2. Things to note:

- The first paragraph of the letter isn't indented, subsequent paragraphs are. If you prefer to have blank lines between paragraphs and no paragraph indentation you can use the `parskip=full` or `parskip=half` class options. For example:

  ```
  \documentclass[parskip=full]{scrlttr2}
  ```

  Input

- The date is automatically inserted. The date format is defined by the language dialect currently in use. Since I've loaded babel with the `british` option, the date is in the form ⟨*day*⟩ ⟨*month*⟩ ⟨*year*⟩.

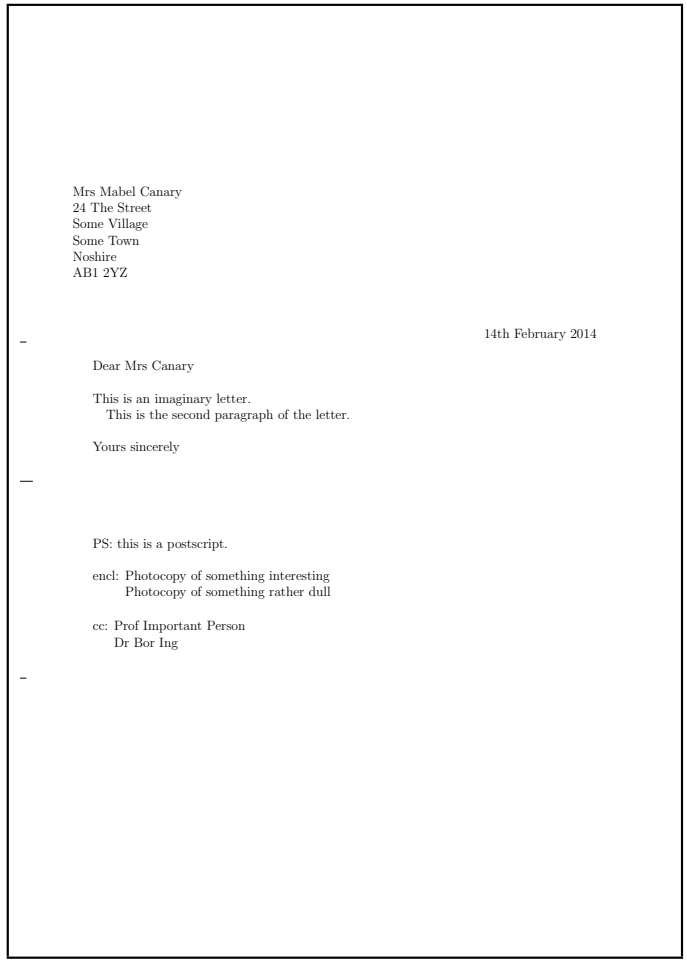


Figure 3.2 A Simple Letter Using the scrlttr2 Class

- There is space between the closing text and the postscript for you to sign your name.
- The page has horizontal marks on the left hand side. These are guides for folding or punching and are discussed below.
- This example assumes that you are going to print the letter on headed paper.

KOMA-Script has "variables" that represents document elements. Variable names don't start with a backslash. They are simple labels such as `yourref`. Variables have both content and a description. You can set the content, and optionally the description, using:

`\setkomavar{`⟨*name*⟩`}[`⟨*description*⟩`]{`⟨*content*⟩`}` Definition

where ⟨*name*⟩ is the variable's name, ⟨*description*⟩ is the variable's description and ⟨*content*⟩ is the variable's content. Alternatively, you can just set the description, without modifying the content, using:

`\setkomavar*{`⟨*name*⟩`}{`⟨*description*⟩`}` Definition

Example:
Suppose I want to add a "Your ref" line before the opening salutation. For example, supposing the recipient has requested the reference "ABC/123" in any correspondence, then I need to add

```
\setkomavar{yourref}{ABC/123}
```
Input

before the `\opening` command. This will now appear in the letter as

↑ Output

↓ Output

Here, the variable is `yourref` and the content is "ABC/123". The default description (in English) for this variable is "Your ref." This can be changed with the optional argument:

```
\setkomavar{yourref}[Your Reference:]{ABC/123}
```
Input

KOMA-Script also has "options" that correspond to the variables. The option name is the same as the variable name and can be set via:

\KOMAoption{⟨option⟩}{⟨value list⟩} Definition

or

\KOMAoptions{⟨option=value list⟩} Definition

For example, the subject option indicates how the subject text should be displayed. Possible values for this option are: afteropening, beforeopening, centered, left, right, titled, underlined or untitled. For example, to make the subject appear after the opening text you can use either

```
\KOMAoptions{subject=afteropening}
```
Input

or

```
\KOMAoption{subject}{afteropening}
```
Input

Some of these settings can be combined. For example:

```
% set the subject options:
\KOMAoption{subject}{afteropening,right,underlined,titled}
% set the subject contents:
\setkomavar{subject}{A sample letter}
```
Input

There are a lot of other variables, such as `invoice` for the invoice number and `customer` for the customer number. Common variables are listed in Table 3.1. Common options with some of their possible values are listed in Table 3.2. See the KOMA-Script documentation [46] for a complete list of all variables and options.

Options can also be set via the optional argument of the letter environment. For example:

```
\begin{letter}[subject=afteropening]{Mrs~Mabel Canary}
```
Input

or in the class options. For example:

```
\documentclass[subject=afteropening]{scrlttr2}
```
Input

The `foldmarks` option may have the single letter values combined. For example:

```
\KOMAoptions{foldmarks=vpmBT}
```
Input

or

```
\KOMAoption{foldmarks}{vpmBT}
```
Input

Table 3.1 Common scrlttr2 Variables

Variable	**Description**
customer	Customer number.
date	Letter date.
firstfoot	Footer for the first page of the letter.
nextfoot	Footer for subsequent pages of the letter.
firsthead	Header for the first page of the letter.
nexthead	Header for subsequent pages of the letter.
fromaddress	Sender's address (without sender's name).
fromemail	Sender's email.
fromfax	Sender's fax number.
fromlogo	Commands for inserting sender's logo.
frommobilephone	Sender's mobile phone number.
fromname	Sender's full name.
fromphone	Sender's telephone number.
fromurl	Sender's URL (e.g. home page).
invoice	Invoice number.
myref	Sender's reference.
location	Additional sender details.
signature	Signature beneath letter ending.
subject	Letter's subject.

Common scrlttr2 Variables (Continued)

Variable	Description
`title`	Letter's title.
`yourref`	Recipient's reference.

Table 3.2 Common scrlttr2 Options with Some of their Values

Option	Value	Description
`firsthead`	`true`	Display letter head.
	`false`	Don't display letter head.
`fromalign`	`center`	Centre return address.
	`left`	Left-justify return address.
	`right`	Right-justify return address.
`fromphone`	`true`	Include sender's phone number.
	`false`	Don't include sender's phone number.
`frommobilephone`	`true`	Include sender's mobile number.
	`false`	Don't include sender's mobile number.

Common scrlttr2 Options with Some of their Values (Continued)

Option	Value	Description
fromfax	true	Include sender's fax number.
	false	Don't include sender's fax number.
fromemail	true	Include sender's email.
	false	Don't include sender's email.
fromurl	true	Include sender's web address.
	false	Don't include sender's web address.
fromlogo	true	Include sender's logo.
	false	Don't include sender's logo.
addrfield	true	Print an address field including a return address, mode of dispatch and priority.
	false	Don't print address field.
backaddress	true	Print return address for window envelope.
	false	Don't print return address for window envelope.

Common scrlttr2 Options with Some of their Values (Continued)

Option	Value	Description
priority	false	Don't print priority field.
	economy	Use international priority B-Economy.
	priority	Use international priority A-Priority.
locfield	narrow	Narrow location field.
	wide	Wide location field.
numericaldate	true	Use numerical date.
	false	Use language-dependent date format.

Common scrlttr2 Options with Some of their Values (Continued)

Option	Value	Description
refline	dateleft	Place date left-most on the reference line.
	dateright	Place date right-most on the reference line.
	narrow	Restrict reference line to typearea.
	nodate	Don't place date on reference line.
	wide	The width of the reference line corresponds to the address and sender's additional details.

Common scrlttr2 Options with Some of their Values (Continued)

Option	Value	Description
subject	true	Print subject field.
	afteropening	Place subject field below letter opening.
	beforeopening	Place subject field above letter opening.
	centered	Centre subject field.
	left	Left-justify subject field.
	right	Right-justify subject field.
	title	Add title/description to subject field.
	untitled	Don't add title/description to subject field.
	underlined	Underline subject field (must be single-lined).
headsepline	true	Insert a separator line below the header.
	false	Don't insert a separator line below the header.

Common scrlttr2 Options with Some of their Values (Continued)

Option	Value	Description
footsepline	true	Insert a separator line above the footer.
	false	Don't insert a separator line above the footer.
pagenumber	foot	Page number in footer.
	head	Page number in header.
	false	No page number.
	center	Page number centred.
	left	Page number left-aligned.
	right	Page number right-aligned.

Common scrlttr2 Options with Some of their Values (Continued)

Option	Value	Description
foldmarks	false	Don't display fold marks.
	B	Activate upper horizontal foldmark on left paper edge.
	b	Deactivate upper horizontal foldmark on left paper edge.
	H	Activate all horizontal folding marks on left paper edge.
	h	Deactivate all horizontal folding marks on left paper edge.
	L	Activate left vertical foldmark on upper paper edge.
	l	Deactivate left vertical foldmark on upper paper edge.
	M	Activate middle horizontal foldmark on left paper edge.
	m	Deactivate middle horizontal foldmark on left paper edge.
	P	Activate punch or centre mark on left paper edge.

Common scrlttr2 Options with Some of their Values (Continued)

Option	Value	Description
	p	Deactivate punch or centre mark on left paper edge.
	T	Activate lower horizontal foldmark on left paper edge.
	t	Deactivate lower horizontal foldmark on left paper edge.
	V	Activate all vertical folding marks on upper paper edge.
	v	Deactivate all vertical folding marks on upper paper edge.

Recall from Volume 1 [92, §6] that the graphicx package [14] provides the command:

`\includegraphics[`⟨*options*⟩`]{`⟨*file*⟩`}` Definition

to include the image file called ⟨*file*⟩. (The extension may be omitted.)

By default, the logo isn't displayed, even if you set the `fromlogo` variable. If you want it displayed you need to activate it via:

```
\KOMAoption{fromlogo}{true}
```
Input

as well as set the fromlogo variable:

```
\setkomavar{fromlogo}{\includegraphics{mylogo}}
```
Input

Example 14. Writing a Letter: KOMA Settings

Remember that you can have multiple letters in the same document. If you want the content of a variable to be the same for all the letters in the document, put the \setkomavar or \setkomavar* command in the preamble. If the content of a variable depends on the recipient, then set it inside the letter environment (before \opening), as illustrated below:

```
\documentclass[12pt,parskip=full]{scrlttr2}

\usepackage[british]{babel}

\KOMAoption{subject}{afteropening,right,underlined,titled}

\KOMAoptions{foldmarks=vpmBT}
```
↑ Input

```
\setkomavar{signature}{Mr Big Head, Managing Director}
\setkomavar{subject}{A sample letter}

\begin{document}

\begin{letter}{Mrs Mabel Canary\\24 The Street\\Some Village\\
Some Town\\Noshire\\AB1 2YZ}
  \setkomavar{myref}{ABC/123}
  \setkomavar{invoice}{123456}
  \setkomavar{customer}{2}

  \opening{Dear Mrs~Canary}

  This is an imaginary letter.

  This is the second paragraph of the letter.

  \closing{Yours sincerely}

  \ps PS: this is a postscript.

  \encl{Photocopy of something interesting\\
```

```
   Photocopy of something rather dull}

  \cc{Prof Important Person\\Dr Bor Ing}
\end{letter}

\end{document}
```

Input

(You can download or view this example.) The resulting document is shown in Figure 3.3.

Exercise 8. Writing a Letter (scrlttr2 class)
For this exercise, modify the sample letter from Example 14 so that it includes a return address and logo. You can make up the sender's address or use your own. If you don't have an image file, you can download the sample logo dummy-logo.png. Alternatively you can use the example-image.pdf file included with the mwe package [82]. (You don't need to load mwe in order to use this image.) Remember you can scale the image using the optional argument of \includegraphics. Also, try changing some of the options, such as switching off all the fold and punch marks. You can download or view a solution.

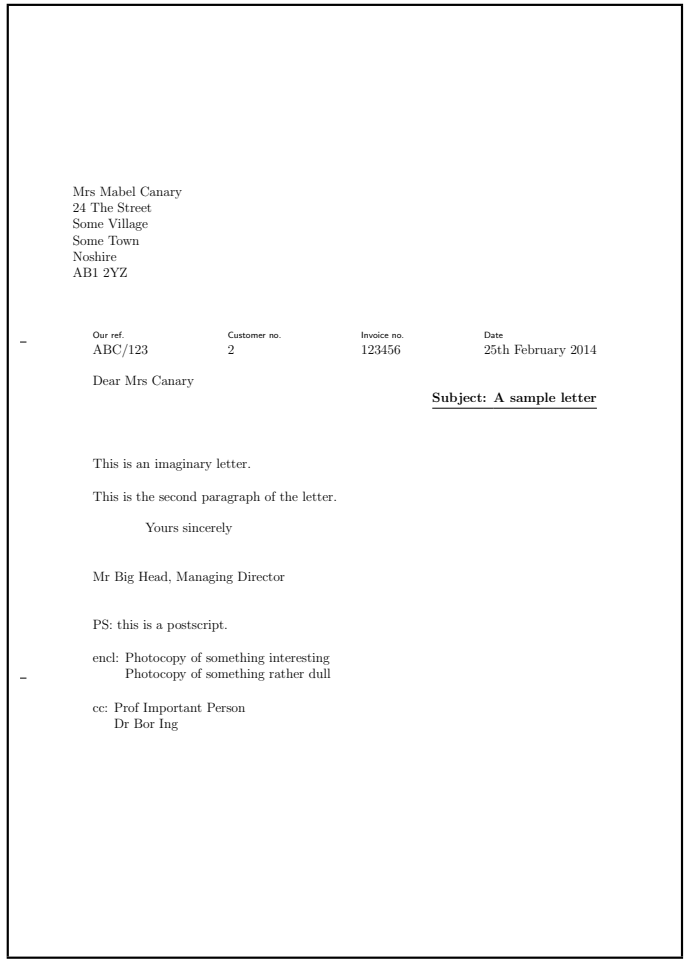


Figure 3.3 A Simple Letter Using KOMA-Script Variables

For the More Adventurous

Recall from §2.8 that you can fetch a single row of data from a database. Use one of the commands described in that section to fetch the recipient's details from the sample `people.csv` file or the `people` SQL table rather than explicitly typing them into the document. You can download or view a solution.

3.3 Writing a Letter Using the newlfm Class

The newlfm class [106] provides the newlfm environment

```
\begin{newlfm}
⟨text⟩
\end{newlfm}
```

Definition

where ⟨*text*⟩ is the body of the letter. The salutation text and recipient's name and address all need to be specified before this environment.

The date of the letter defaults to the current date (via `\today`) but can be set using:

`\dateset{`⟨*date*⟩`}`

Definition

The salutation text is specified via:

\greetto{⟨text⟩} Definition

For example

```
\greetto{Dear Mrs Canary}
```
Input

The closing text is specified via:

\closeline{⟨text⟩} Definition

For example

```
\closeline{Yours sincerely}
```
Input

The recipient's name is specified via:

\nameto{⟨name⟩} Definition

For example:

```
\nameto{Mrs Mabel Canary}
```
Input

The recipient's address is specified via:

\addrto{⟨address⟩} Definition

For example:

↑ Input

```
\addrto{24 The Street\\
Some Village\\
Some Town\\
Noshire\\
AB1 2YZ
}
```

↓ Input

The sender's name is specified via:

`\namefrom{`⟨*name*⟩`}` Definition

For example:

```
\namefrom{Mr Big Head}
```

Input

The sender's address is specified via:

`\addrfrom{`⟨*address*⟩`}` Definition

For example

```
\addrfrom{University of Somewhere\\
Some City\\
AB3 4YZ}
```

Input

Input

The sender's phone number is specified via:

\phonefrom{⟨*number*⟩}

Definition

For example

```
\phonefrom{0123456789}
```

Input

The sender's email address is specified via:

\emailfrom{⟨*address*⟩}

Definition

For example

```
\emailfrom{big.head@somewhere.ac.uk}
```

Input

The subject text is specified via:

\regarding{⟨text⟩} Definition

For example:

```
\regarding{sample letter}
```
Input

The postscript is specified via:

\psitem{⟨text⟩} Definition

For example:

```
\psitem{Don't forget to bring some cake!}
```
Input

The post-postscript is specified via:

\ppsitem{⟨text⟩} Definition

For example:

```
\ppsitem{And the ice cream!}
```
Input

The "CC" list is specified via:

\cclist{⟨text⟩} Definition

For example:

```
\cclist{Prof Important Person}
```
Input

The list of enclosures is specified via:

\encllist{⟨*text*⟩}
Definition

For example:

```
\encllist{Photograph of a hat}
```
Input

There are other commands as well. See the newlfm documentation [106] for further details.

The newlfm class comes with a number of predefined letter styles, which can be set via the class options: `busletter`, `busletternofrom`, `stdletter`, `stdletterfrom`. The "nofrom" styles don't display the sender's address. The business letter "busletter" styles use a different alignment to the standard "stdletter" styles.

Alternatively, the letter style can be set via:

\newlfmP{⟨*option list*⟩}
Definition

Other letter-related options that can be set using \newlfmP (or in the class option list) are listed in Table 3.3.

Table 3.3 Letter Options for the newlfm Class

Option	Description
noaddrfrom	Omit sender's address.
addrfromphone	Include sender's phone.
addrfromemail	Include sender's email.
addrfromfax	Include sender's fax.
printallfrom	Print all of the sender's details.
addrfromright	Right-align sender's block.
addrfromleft	Left-align sender's block.
printallto	Print all of recipient's details.
addrtoright	Right-align recipient's block.
addrtoleft	Left-align recipient's block.
addrtophone	Include recipient's phone.
addrtoemail	Include recipient's email.
addrtofax	Include recipient's fax.
dateright	Right-align date.
dateleft	Left-align date.
datecenter	Centre date.
dateyes	Include date.
dateno	Don't include date.

Table 3.3 Letter Options for the newlfm Class (Continued)

Option	Description
orderdatefromto	Display order: date, sender, recipient.
orderfromtodate	Display order: sender, recipient, date.
orderfromdateto	Display order: sender, date, recipient.
sigright	Right-align signature.
sigleft	Left-align signature.
sigcenter	Centre signature.

Example 15. A Simple Letter (newlfm class)
Here's a simple letter using the newlfm class:

↑ Input

```
\documentclass[stdletter]{newlfm}

\usepackage[british]{babel}

\newlfmP{orderfromtodate,sigcenter,addrfromphone,addrfromemail}
```

```
\nameto{Mrs Mabel Canary}
\addrto{24 The Street\\
Some Village\\Some Town\\
Noshire\\AB1 2YZ}

\namefrom{Mr Big Head}
\addrfrom{University of Somewhere\\Some City\\AB3 4YZ}
\emailfrom{big.head@somewhere.ac.uk}
\phonefrom{0123456789}

\regarding{A sample letter}

\greetto{Dear Mrs Canary}
\closeline{Yours sincerely}
\cclist{Prof Important Person\\Dr Bor Ing}
\encllist{Photocopy of something interesting\\
   Photocopy of something rather dull}
\psitem{this is a postscript}

\begin{document}
```

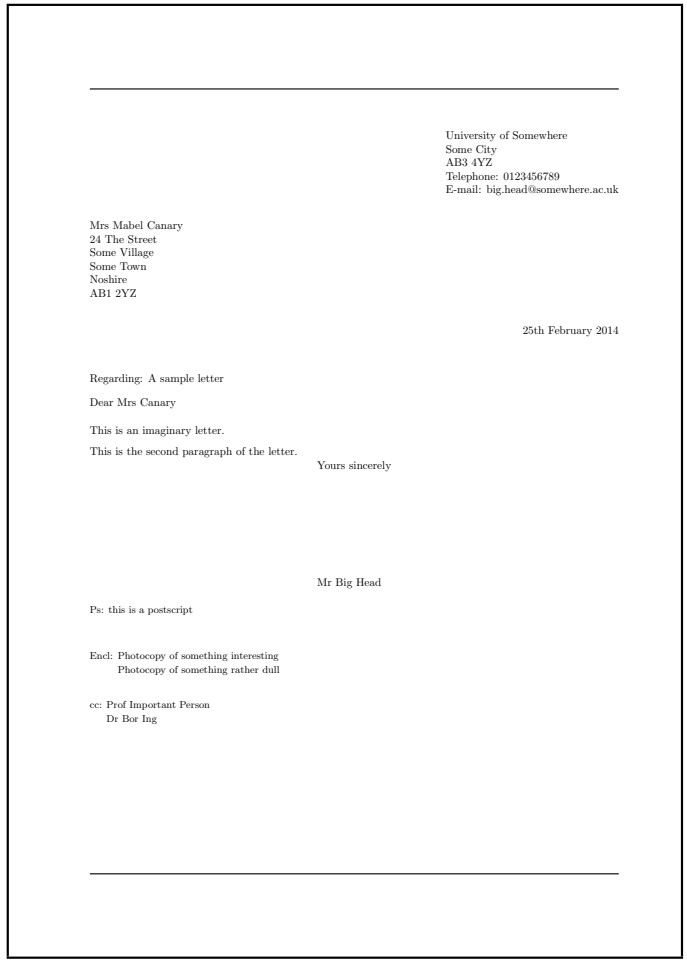
```
\begin{newlfm}
  This is an imaginary letter.

  This is the second paragraph of the letter.
\end{newlfm}

\end{document}
```

↓ Input

(You can download or view this example.) The resulting document is shown in Figure 3.4.

Exercise 9. Writing a Letter (newlfm class)
Create the document from Example 15 and try adjusting some of the options given in Table 3.3.

For the More Adventurous
As with Exercise 8, use one of the commands described in §2.8 to fetch the recipient's details from the sample `people.csv` file or the `people` SQL table rather than explicitly typing them into the document. You can download or view a solution.



University of Somewhere
Some City
AB3 4YZ
Telephone: 0123456789
E-mail: big.head@somewhere.ac.uk

Mrs Mabel Canary
24 The Street
Some Village
Some Town
Noshire
AB1 2YZ

25th February 2014

Regarding: A sample letter

Dear Mrs Canary

This is an imaginary letter.

This is the second paragraph of the letter.

Yours sincerely

Mr Big Head

Ps: this is a postscript

Encl: Photocopy of something interesting
Photocopy of something rather dull

cc: Prof Important Person
Dr Bor Ing

Figure 3.4 A Simple Letter Using the newlfm Class

3.4 Writing a Letter Using the isodoc Class

The isodoc class [21] differs from the classes described in the previous sections in that instead of encasing the letter within the body of an environment (such as letter or newlfm) the letter is created using a command:

`\letter[`⟨*recipient options*⟩`]{`⟨*contents*⟩`}` Definition

As with the other classes, a document may contain multiple letters. The optional argument ⟨*recipient options*⟩ is a key=value list of options to apply to this letter. The other argument ⟨*contents*⟩ contains the contents of the letter. General options can be set using:

`\setupdocument{`⟨*options*⟩`}` Definition

Again, ⟨*options*⟩ is a key=value list. There are a lot of options available, so this section will only cover common settings. For full details, see the isodoc user guide [21]

You don't need to use babel with isodoc. Instead you can set the language using the `language` key. As of the time of writing, available language options are: `en-GB` (default), `en-US`, `fr-FR`, `de-DE`, `nl-NL`, `nl-BE`, `it-IT`, `es-ES`, `ca-ES`, `nb-NO` and `sr-RS`. The hyphen is optional so, for example, `enGB` is the same as `en-GB`. For example, to switch to US English:

```
\setupdocument{language=en-US}
```
Input

Options that set information about the sender include:

`company`	Company name (or sender's name if a private document).
`logoaddress`	Sender's address. If omitted it will be constructed from the following:
`who`	Contact person's name.
`street`	Sender's street.
`city`	Sender's city.
`zip`	Sender's zip or postcode.
`countrycode`	Sender's country code.
`country`	Sender's country name.
`areacode`	Sender's area code.
`cityzip`	Place the zip code after the city.

`foreign` If this key is used, the country name will be added to the address, the zip code will be prefixed with the country code and the telephone numbers will be prefixed with the area code. (This isn't a boolean key, but if you want to generate multiple letters with a mixture of national and international addresses, you can switch this setting on and off using `\foreigntrue` and `\foreignfalse`.)

The above information is placed in the logo area. The options that govern the address window include:

`to` The recipient's address. You can use `\\` to break the lines.

`return` Include the return address in the address window.

`returnaddress` If the return address is too long for the address window, this key can be used to set a shorter version for the window.

Options that set the header information include:

`yourletter` If this letter is a reply to a letter from the recipient, this value is the date of the recipient's letter.

`yourref` The recipient's reference if this is a reply.

`ourref` The sender's reference.

`date` The date of this letter, which must be in the form ⟨*yyyy*⟩-⟨*mm*⟩-⟨*dd*⟩ or ⟨*yyyy*⟩⟨*mm*⟩⟨*dd*⟩. This will be converted into the format governed by the language setting (as specified by the `language` key described above). For example:

```
\setupdocument{date={2014-03-01}}
```
Input

indicates the first day of March, 2014.

`forcedate` May be used instead of the previous key to force the data to be in a specific format. For example:

```
\setupdocument{forcedate={Sat 1st March, 2014}}
```
Input

`subject` The subject of this letter.

Options that set the opening and closing information include:

`opening` The opening salutation.

`closing` The closing salutation.

`signature` The name of the sender as it should appear below the closing salutation.

`enclosures` Lists any enclosures accompanying the letter (may include \\ to start a newline).

`copyto` A "CC" list. Again this may include \\.

`autograph` This option governs the area between the closing text and the signature text. The value may be one of:

- **0** No space between the closing and signature text (default);
- **1** Leaves a space between the closing and signature text for a handwritten signature;
- **2–9** Inserts one of eight autograph images (see the isodoc manual [21] for further details).

Options that set the footer information include:

`footer` Enables the footer information.

`phoneprefix` Sets the phone prefix (defaults to `0`).

`phone` The sender's phone number (omit the phone prefix).

`cellphone`	The sender's mobile phone (omit the phone prefix).
`fax`	The sender's fax number (omit the phone prefix).
`email`	The sender's email address.
`website`	The sender's web address.

There are other options that govern the layout. See the isodoc documentation [21] for further details. There are also other options that are concerned with invoices. These are described in §4.1.

Example 16. A Simple Letter (isodoc class)
The letter from Example 14 can be rewritten using the isodoc class as follows:

↑ Input

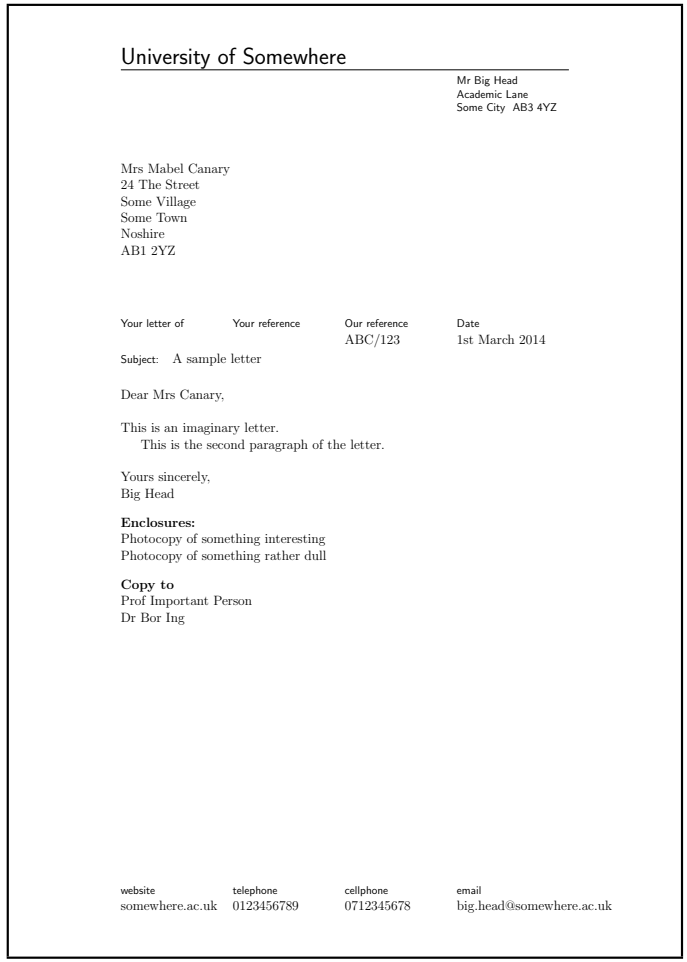
```
\documentclass[12pt]{isodoc}

\setupdocument
{%
  language={en-GB},%
  company={University of Somewhere},%
```

```
  who={Mr Big Head},%
  street={Academic Lane},%
  city={Some City},%
  zip={AB3 4YZ},%
  country={United Kingdom},%
  countrycode={GB},%
  areacode={44},%
  cityzip,%
  subject={A sample letter},%
  closing={Yours sincerely},%
  enclosures={Photocopy of something interesting\\
   Photocopy of something rather dull},%
  signature={Big Head},%
  copyto={Prof Important Person\\Dr Bor Ing},%
  footer,%
  phone={123456789},%
  cellphone={712345678},%
  email={big.head@somewhere.ac.uk},%
  website={somewhere.ac.uk},%
  date={2014-03-01}%
}
```

```
\begin{document}

\letter
[%
  opening={Dear Mrs Canary},%
  to={Mrs Mabel Canary\\%
  24 The Street\\%
  Some Village\\Some Town\\%
  Noshire\\AB1 2YZ},%
  ourref={ABC/123}%
]%
{%
  This is an imaginary letter.

  This is the second paragraph of the letter.
}

\end{document}
```

Input

(You can download or view this example.) The resulting document is shown in Figure 3.5. There's no space between the closing text and signature as I didn't use the autograph option.



University of Somewhere

Mr Big Head
Academic Lane
Some City AB3 4YZ

Mrs Mabel Canary
24 The Street
Some Village
Some Town
Noshire
AB1 2YZ

Your letter of	Your reference	Our reference	Date
		ABC/123	1st March 2014

Subject: A sample letter

Dear Mrs Canary,

This is an imaginary letter.
This is the second paragraph of the letter.

Yours sincerely,
Big Head

Enclosures:
Photocopy of something interesting
Photocopy of something rather dull

Copy to
Prof Important Person
Dr Bor Ing

website	telephone	cellphone	email
somewhere.ac.uk	0123456789	0712345678	big.head@somewhere.ac.uk

Figure 3.5 A Simple Letter Using the isodoc Class

3.5 Mail Merging

The scrlttr2 and newlfm classes both provide ways of creating a template letter for mail merging. However, in case you don't want to be fixed to a specific class, or you want to use the letter or isodoc class (or some other class not described in this book), this section looks at a more generic method of mail-merging using the datatool package.

Recall from §2.7.1 that you can iterate through a datatool database using `\DTLforeach*`. This technique can be used to create a letter for each person in the database, or you can apply filtering to only send to a subset of the database.

In fact, this is just a small modification of the "For the More Adventurous" sections of Exercise 7, Exercise 8 and Exercise 9. Any settings, such as the sender's details, that stay constant for all the letters can be set before `\DTLforeach*`. The letter or newlfm environment can go inside the body of `\DTLforeach*`.

Example 17. Mail Merging (letter class)
To send a letter to everyone listed in sample `people.csv` file:

↑ Input

```
\documentclass{letter}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\usepackage[a4paper]{geometry}
\usepackage[british]{babel}
\usepackage{datatool}

\DTLloaddb{people}{people.csv}
\DTLloaddb{countries}{country-codes.csv}

\name{Mr Big Head}
\signature{Big Head}
\location{Secret Lab of Experimental Stuff}
\address{University of Somewhere\\Some City\\AB3 4YZ}
\telephone{0123456789}

\begin{document}
```

```
\DTLforeach*{people}% data
{% assignments
  \Id=id,%
  \Surname=surname,%
  \Forenames=forenames,%
  \Title=title,%
  \AddressI=address1,%
  \AddressII=address2,%
  \Town=town,%
  \County=county,%
  \Postcode=postcode,%
  \CountryCode=country%
}
{%
 % fetch country name
 \xDTLassignfirstmatch{countries}{code}{\CountryCode}{\CountryName=name}

 \begin{letter}{% recipient's address
   \DTLifnullorempty{\Title}{}{\Title\␣}\Forenames\␣\Surname\\%
   \AddressI\\
   \DTLifnullorempty{\AddressII}{}{\AddressII\\}% optional line
   \Town\\
```

```
   \DTLifnullorempty{\County}{}{\County\\}% optional line
   \Postcode\\\CountryName}

 \opening{Dear \DTLifnullorempty{\Title}{\Forenames}{\Title} \Surname}

 \thispagestyle{firstpage}
 This is an imaginary letter.

 This is the second paragraph of the letter.

 \closing{Yours sincerely}

 \ps PS: this is a postscript.

 \encl{Photocopy of something interesting\\
  Photocopy of something rather dull}

 \cc{Prof Important Person\\Dr Bor Ing}
 \end{letter}
}

\end{document}
```

Input

This produces a six page document, where each page contains a letter to one of the six people in the database. You can download or view this document.

Example 18. Mail Merging (newlfm class)

To send a letter to everyone listed in sample people.csv file:

↑ Input

```
\documentclass[stdletter]{newlfm}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\usepackage[british]{babel}
\usepackage{datatool}

\DTLloaddb{people}{people.csv}
\DTLloaddb{countries}{country-codes.csv}

\newlfmP{orderfromtodate,sigcenter,addrfromphone,addrfromemail}
```

```
\namefrom{Mr Big Head}
\addrfrom{University of Somewhere\\Some City\\AB3 4YZ}
\emailfrom{big.head@somewhere.ac.uk}
\phonefrom{0123456789}

\regarding{A sample letter}

\begin{document}

\closeline{Yours sincerely}

\cclist{Prof Important Person\\Dr Bor Ing}

\encllist{Photocopy of something interesting\\
   Photocopy of something rather dull}

\psitem{this is a postscript}

\DTLforeach*{people}% data
{% assignments
  \Id=id,%
```

```
  \Surname=surname,%
  \Forenames=forenames,%
  \Title=title,%
  \AddressI=address1,%
  \AddressII=address2,%
  \Town=town,%
  \County=county,%
  \Postcode=postcode,%
  \CountryCode=country%
}
{%
 \xDTLassignfirstmatch{countries}{code}{\CountryCode}{\CountryName=name}

 \nameto{\DTLifnullorempty{\Title}{}{\Title\␣}\Forenames\␣\Surname}
 \addrto{%
   \AddressI\\
   \DTLifnullorempty{\AddressII}{}{\AddressII\\}% optional line
   \Town\\
   \DTLifnullorempty{\County}{}{\County\\}% optional line
   \Postcode\\\CountryName
 }
```

```
 \greetto{Dear \DTLifnullorempty{\Title}{\Forenames}{\Title} \Surname}

 \begin{newlfm}
   This is an imaginary letter.

   This is the second paragraph of the letter.
 \end{newlfm}
}

\end{document}
```

Input

As with the previous example, this produces a six page document, where each page contains a letter to one of the six people in the database. You can download or view this document.

Note

The newlfm environment tries to determine the total number of pages per letter. This is done using

```
\label{totpage}
```

Input

at the end of the letter. Since there are six letters, this causes five instances of

```
LaTeX Warning: Label `totpage' multiply defined.
```

However, since each letter is of the same length, these warnings can be ignored.

Exercise 10. Mail Merging

Adapt the more adventurous section of Exercise 8 so that a letter is generated for each person in the sample `people.csv` file who has the `subscribed` field set. If you prefer, you can use the `people` SQL table. (Recall the `\ifcsbool` command defined on page 212. Alternatively you can use the optional argument of `\DTLforeach*` with ifthen's `\equal` command.)

The recipient's country only needs to be included if it's different from the sender's country. For example, suppose the sender's address is in the United Kingdom, then the country name isn't required for the entries where the recipient's country code is "gb". Modify your code so that it doesn't include the country name in the recipient's address if it's the same as the sender's country. You can download or view a solution.

If you prefer, you can use one of the other letter classes, such as isodoc, described in §3.4. You can download or view a solution using the isodoc class.

3.6 Envelopes

The letter class defines a preamble-only command:

`\makelabels` Definition

which gathers the addresses of all the recipients and generates address labels at the end of the document. These labels can then be stuck onto the envelopes.

For example, the document from Example 17 had six pages, one page for each letter. If you add `\makelabels` to the preamble of that document, a seventh sheet will be generated with the address labels, shown in Figure 3.6.

This is fine if this matches the size of your label sheets, but it can't be easily adapted for other label sizes. There are some envelope-related packages listed on the letter topic but take care as some of them, such as envelope, were written for LaTeX2.09 and may not work as well with LaTeX 2ε or licensing issues may prevent them from being included in TeX Live.

Some of the other letter-like classes also provide envelope labelling facilities, but there are two packages that are both in the TeX Live and MiKTeX distributions: envlab and envbig. Another possibility is to use the ticket package, which is discussed in §10.2.

The user guide for envlab [110] can be obtained via:

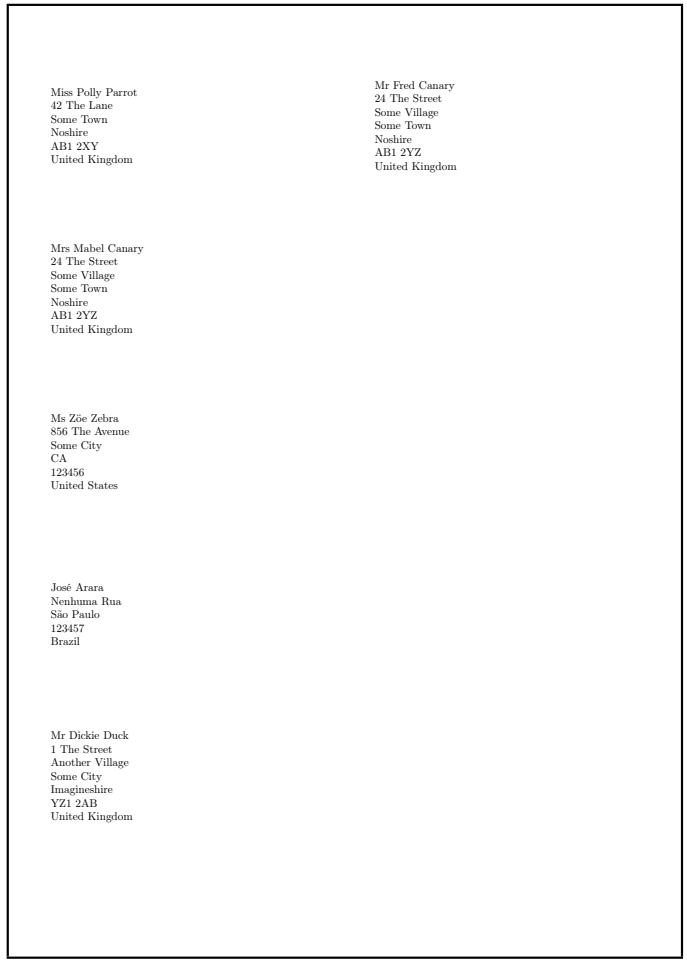


Figure 3.6 Address Labels

```
texdoc elguide
```
Shell

Note that `texdoc envlab` produces the documented source code, which you may find more complicated than the user guide. There's no proper manual for envbig; `texdoc envbig` just opens the README file. You need to open `envbig.sty` in your text editor and read the comments for an example of how to use the package. Therefore this book will just look at envlab. At the time of writing, the current version of envlab is 1.2 (1997-07-16).

The envlab package is designed for US postal layouts, but it's possible to define custom label sizes. This package redefines `\makelabels` but it's used in the same way as for the letter class.

The envlab package is configured for three different types of media: envelopes (one per page, optionally rotated), labels (without a return address) and big labels (with a return address). Envelopes are usually printed in landscape format. This rotation can be switched on or off using the `rotateenvelopes` or `norotateenvelopes` options.

The envelope layout can be set by the package options listed in Table 3.4 or a custom size can be set via:

`\SetEnvelope[`⟨*top margin*⟩`]{`⟨*width*⟩`}{`⟨*height*⟩`}`
Definition

where ⟨*top margin*⟩ is the height of the top margin, ⟨*width*⟩ is the envelope width and ⟨*height*⟩ is the envelope height.

Table 3.4 Envelope Options for the envlab Package

Option	Width	Height
businessenvelope	9.5 in	4.125 in
executiveenvelope	7.5 in	3.875 in
bookletenvelope	10.5 in	7.5 in
personalenvelope	6.5 in	3.625 in
c6envelope	162 mm	114 mm
c65envelope	224 mm	114 mm
c5envelope	229 mm	162 mm
dlenvelope	220 mm	110 mm

The standard label layout can be set by the package options listed in Table 3.5 or a custom size can be set via:

\SetLabel{⟨*width*⟩}{⟨*height*⟩}{⟨*top*⟩}{⟨*left*⟩}{⟨*sep*⟩}{⟨*columns*⟩}{⟨*rows*⟩}

Definition

where ⟨*width*⟩ is the total width from the left border of one label and the left border of the label in the next column, ⟨*height*⟩ is the total height from the top border of one label and the top border of the label in the row below, ⟨*top*⟩ and ⟨*left*⟩ are the distances between the edge of the paper and the label, ⟨*sep*⟩ is the horizontal distance between labels and ⟨*columns*⟩ and

⟨*rows*⟩ are the number of columns and rows of labels per page.

Table 3.5 Standard Label Options for the envlab Package

Option	Width	Height	Top	Left	Sep	Cols	Rows
avery5160label	2.75 in	1 in	0.5 in	0.19 in	0.12 in	3	10
avery5161label	4.19 in	1 in	0.5 in	0.16 in	0.19 in	2	10
avery5162label	4.19 in	1.33 in	0.83 in	0.16 in	0.19 in	2	7
avery5163label	4.19 in	2 in	0.5 in	0.16 in	0.19 in	2	5
avery5164label	4.19 in	3.33 in	0.5 in	0.16 in	0.19 in	2	3
avery5262label	110 mm	34 mm	21 mm	4 mm	5 mm	2	7
herma4625label	105 mm	42.3 mm	0 mm	5 mm	5 mm	2	7

The big label layout can be set by the package options listed in Table 3.6 or a custom size can be set via:

\SetBigLabel{⟨*width*⟩}{⟨*height*⟩}{⟨*top*⟩}{⟨*left*⟩}{⟨*sep*⟩}{⟨*columns*⟩}{⟨*rows*⟩} Definition

The arguments are the same as for \SetLabel.

If you have a partially used sheet, you can specify the starting label via:

\FirstLabel{⟨*row*⟩}{⟨*column*⟩} Definition

Table 3.6 Big Label Options for the envlab Package

Option	Width	Height	Top	Left	Sep	Cols	Rows
avery5163biglabel	4.19 in	2 in	0.5 in	0.16 in	0.19 in	2	5
avery5164biglabel	4.19 in	3.33 in	0.5 in	0.16 in	0.19 in	2	3

where ⟨*row*⟩ is the row index (starting from 1) and ⟨*column*⟩ is the column index (starting from 1). The labels are printed row by row.

The return address for the big envelopes is taken from the argument of \address but this can be changed by redefining:

\returnaddress

Definition

This can be changed to the textual address or it can use \includegraphics to use a company logo.

Example 19. Envelope Labels

This example creates a letter using the letter class but uses the envlab package to create a custom sized big label.

↑ Input

```
\documentclass{letter}
```

```
\usepackage[a4paper]{geometry}
\usepackage[british]{babel}

\usepackage{envlab}

\SetBigLabel{101mm}{139mm}{9mm}{3mm}{2mm}{2}{2}

\makelabels

\name{Mr Big Head}
\signature{Big Head}
\location{Secret Lab of Experimental Stuff}
\address{University of Somewhere\\Some City\\AB3 4YZ}
\telephone{0123456789}

\begin{document}

  \begin{letter}{Miss Polly Parrot\\42 The Lane\\Some Town\\AB1 2XY}

  \opening{Dear Miss Parrot}
```

```
  \thispagestyle{firstpage}
  This is an imaginary letter.

  This is the second paragraph of the letter.

  \closing{Yours sincerely}

  \ps PS: this is a postscript.

  \encl{Photocopy of something interesting\\
   Photocopy of something rather dull}

  \cc{Prof Important Person\\Dr Bor Ing}
  \end{letter}

\end{document}
```

↓ Input

This produces a document with two pages. The first contains the letter and the second contains the label. This second page is shown in Figure 3.7.

This produces a portrait label, but if you have a wide address it may look better rotated to make a landscape label. Unfortunately, the envlab options that govern rotation (`rotateenvelope` and `norotateenvelope`) don't apply

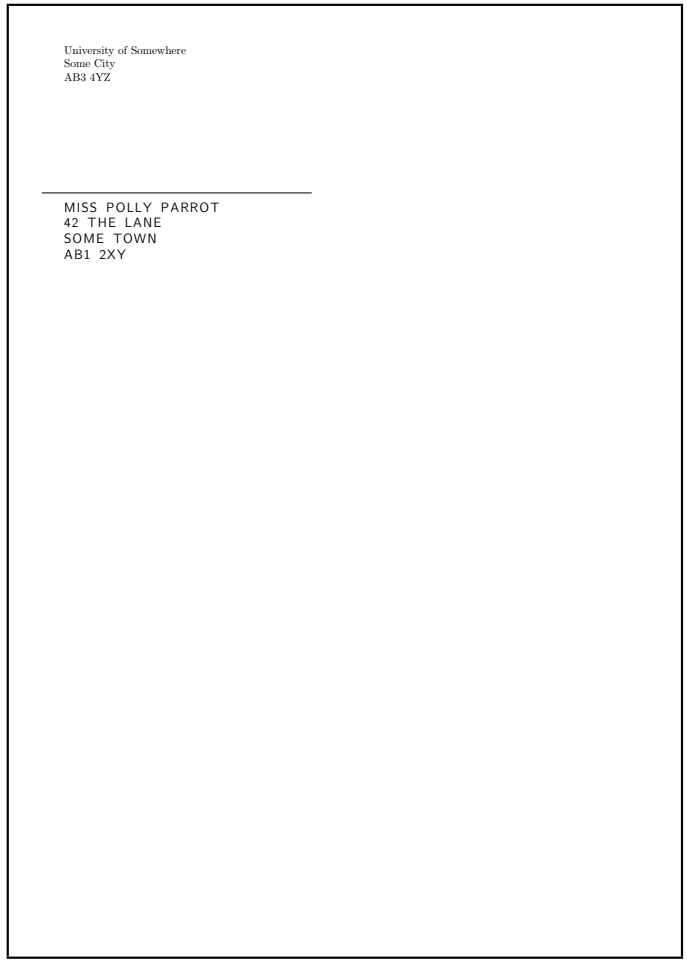


Figure 3.7 Custom Big Label

to labels.

These big labels are implicitly typeset using

\PrintBigLabel{⟨*from-address*⟩}{⟨*to-address*⟩} Definition

where ⟨*from-address*⟩ is the sender's address and ⟨*to-address*⟩ is the recipient's address. This command is defined as:

Input

```
\newcommand{\PrintBigLabel}[2]{%
 \begin{minipage}[t][\LabelHeight]{\LabelWidth}%
 \baselineskip=0pt%
 \lineskip=0pt%
 \parindent=0pt%
 \begin{center}%
 \PrintReturnAddress{#1}\\%
 \rule{\ToAddressWidth}{0.1pt}%
 \PrintAddress{#2}%
 \end{center}%
 \end{minipage}}
```

Input

This displays the label in a minipage with the dimensions given by \LabelWidth and \LabelHeight which have been set to the width and height of the printable label area. This can be redefined to provide your own custom label format. For example (recall \rotatebox provided by the graphicx package [14] described in Volume 1 [92, §6.1]):

Input

```
\renewcommand{\PrintBigLabel}[2]{%
   \begin{minipage}[t]{\LabelHeight}{\LabelWidth}
   \vfill
   \hspace*{1em}%
   \rotatebox[origin=l]{90}{\PrintReturnAddress{#1}}\hfill
   \rule{0.1pt}{\ToAddressWidth}\space
   \rotatebox[origin=l]{90}{\PrintAddress{#2}}%
   \hspace*{1em}%
   \vfill
   \end{minipage}%
}
```

Input

This now produces the label shown in Figure 3.8. You can further customize this redefinition if you like. For example, you may want the

recipient's address above the return address. You can download or view this example document.

⚠ Unfortunately the default capaddress package option doesn't work with extended characters unless they have been placed inside a group, which will cause a problem with some of the entries in the sample people.csv file or the people SQL table. This means that if we want to adapt Example 17 to use envlab, we have to switch off the capitalisation feature using the nocapaddress package option.

Example:

↑ Input

```
\documentclass{letter}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[nocapaddress]{envlab}

\makelabels
```

Figure 3.8 Custom Formatted Big Label

```
\begin{document}

  \begin{letter}{Ms Zöe Zebra\\856 The Avenue}

  \opening{Dear Miss Parrot}

  A sample letter.

  \closing{Yours sincerely}

  \end{letter}

\end{document}
```

Input

With the nocapaddress option, this code compiles without error. With the default capaddress option, the ö must be placed inside a group:

```
\begin{letter}{Ms Z{ö}e Zebra\\856 The Avenue}
```

Input

(For those of you who have used glossaries or mfirstuc, it stems from a similar issue. See the section "UTF-8" in the mfirstuc documentation [98].)

Alternatively use XeLaTeX instead of PDFLaTeX (the inputenc and fontenc packages need to be replaced by the fontspec [75] package):

↑ Input

```
% arara: xelatex
\documentclass{letter}

\usepackage{fontspec}
\usepackage{envlab}

\makelabels

\begin{document}

  \begin{letter}{Ms Zöe Zebra\\856 The Avenue}

  \opening{Dear Miss Parrot}

  A sample letter.

  \closing{Yours sincerely}

  \end{letter}
```

```
\end{document}
```

Input

Example 20. Mail Merging with letter and envlab
The above can be put together to form a complete document that contains the correspondence and the large mailing labels:

Input

```
\documentclass{letter}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\usepackage[a4paper]{geometry}
\usepackage[british]{babel}
\usepackage{datatool}
\usepackage{graphicx}

\usepackage[nocapaddress]{envlab}

\DTLloaddb{people}{people.csv}
\DTLloaddb{countries}{country-codes.csv}
```

```
\SetBigLabel{101mm}{139mm}{9mm}{3mm}{2mm}{2}{2}

\renewcommand{\PrintBigLabel}[2]{%
  \begin{minipage}[t][\LabelHeight]{\LabelWidth}%
    \vfill
    \hspace*{1em}%
    \rotatebox[origin=l]{90}{\PrintReturnAddress{#1}}\hfill
    \rule{0.1pt}{\ToAddressWidth}\space
    \rotatebox[origin=l]{90}{\PrintAddress{#2}}%
    \hspace*{1em}%
    \vfill
  \end{minipage}%
}

\makelabels

\name{Mr Big Head}
\signature{Big Head}
\location{Secret Lab of Experimental Stuff}
\address{University of Somewhere\\Some City\\AB3 4YZ}
\telephone{0123456789}
```

```
\begin{document}

\DTLforeach*{people}% data
{% assignments
  \Id=id,%
  \Surname=surname,%
  \Forenames=forenames,%
  \Title=title,%
  \AddressI=address1,%
  \AddressII=address2,%
  \Town=town,%
  \County=county,%
  \Postcode=postcode,%
  \CountryCode=country%
}
{%
 \xDTLassignfirstmatch{countries}{code}{\CountryCode}{\CountryName=name}

 \begin{letter}{\DTLifnullorempty{\Title}{}{\Title\␣}%
 \Forenames\␣\Surname\\\AddressI\\
 \DTLifnullorempty{\AddressII}{}{\AddressII\\}\Town\\
```

```
 \DTLifnullorempty{\County}{}{\County\\}\Postcode\\\CountryName}

 \opening{Dear \DTLifnullorempty{\Title}{\Forenames}{\Title} \Surname}

 \thispagestyle{firstpage}
 This is an imaginary letter.

 This is the second paragraph of the letter.

 \closing{Yours sincerely}

 \ps PS: this is a postscript.

 \encl{Photocopy of something interesting\\
  Photocopy of something rather dull}

 \cc{Prof Important Person\\Dr Bor Ing}
 \end{letter}
}

\end{document}
```

↓ Input

You can download or view this document.

The newlfm class automatically loads the envlab class. There's a class option `useenvlab` that's designed to activate the envlab functions but its use seems to be designed to work with newlfm's mechanism for storing addresses in an external file called `letrinfo.tex`. This doesn't fit in with the generic mail-merging functions discussed here, so instead we'll look at how to manually make address labels with envlab. This method can be applied to other classes or you can use this method if you just want to generate labels without a corresponding letter.

For the manual method, when you want to start typesetting the labels you need to use:

`\startlabels` Definition

Then you use

`\mlabel{`⟨*from-address*⟩`}{`⟨*to-address*⟩`}` Definition

for each label where ⟨*from-address*⟩ is the sender's address and ⟨*to-address*⟩ is the recipient's address.

Example 21. Mail Merging with newlfm, envlab and datatool

If you use the manual approach, you must generate the labels after you've finished typesetting the letters. This example illustrates this manual approach to generate letters and corresponding labels to everyone in the sample people.csv file database or the people SQL table who has the subscribed field set:

↑ Input

```
\documentclass[stdletter,nocapaddress,avery5164biglabel]{newlfm}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\usepackage[british]{babel}
\usepackage{datatool}

\newcommand{\ifcsbool}[3]{%
  \ifboolexpr
  {
    test{\ifdefstring{#1}{true}} or
    test{\ifdefstring#1{1}}
  }
```

```
  {#2}{#3}%
}

\DTLloaddb{people}{people.csv}
\DTLloaddb{countries}{country-codes.csv}

\newlfmP{orderfromtodate,sigcenter,addrfromphone,addrfromemail}

\namefrom{Mr Big Head}
\addrfrom{University of Somewhere\\Some City\\AB3 4YZ}
\emailfrom{big.head@somewhere.ac.uk}
\phonefrom{0123456789}

\regarding{A sample letter}

\closeline{Yours sincerely}

\cclist{Prof Important Person\\Dr Bor Ing}

\encllist{Photocopy of something interesting\\
   Photocopy of something rather dull}
```

```
\psitem{this is a postscript}

\begin{document}

\DTLforeach*{people}% data
{% assignments
  \Id=id,%
  \Surname=surname,%
  \Forenames=forenames,%
  \Title=title,%
  \AddressI=address1,%
  \AddressII=address2,%
  \Town=town,%
  \County=county,%
  \Postcode=postcode,%
  \CountryCode=country,%
  \Subscribed=subscribed%
}
{%
 \ifcsbool{\Subscribed}
 {%
  \xDTLassignfirstmatch{countries}{code}{\CountryCode}{\CountryName=name}
```

```
  \nameto{\DTLifnullorempty{\Title}{}{\Title\␣}\Forenames\␣\Surname}
  \addrto{%
   \AddressI\\
   \DTLifnullorempty{\AddressII}{}{\AddressII\\}\Town\\
   \DTLifnullorempty{\County}{}{\County\\}\Postcode\\\CountryName
  }

  \greetto{Dear \DTLifnullorempty{\Title}{\Forenames}{\Title} \Surname}

  \begin{newlfm}

  This is an imaginary letter.

  This is the second paragraph of the letter.

  \end{newlfm}
 }%
 {}% not subscribed
}

\startlabels
```

```
\DTLforeach*{people}% data
{% assignments
  \Id=id,%
  \Surname=surname,%
  \Forenames=forenames,%
  \Title=title,%
  \AddressI=address1,%
  \AddressII=address2,%
  \Town=town,%
  \County=county,%
  \Postcode=postcode,%
  \CountryCode=country,%
  \Subscribed=subscribed%
}
{%
 \ifcsbool{\Subscribed}
 {%
  \xDTLassignfirstmatch{countries}{code}{\CountryCode}{\CountryName=name}
  \mlabel
    {Mr Big Head\\University of Somewhere\\Some City AB3 4YZ}%
    {\DTLifnullorempty{\Title}{}{\Title\␣}\Forenames\␣\Surname\\%
```

```
  \AddressI\\
  \DTLifnullorempty{\AddressII}{}{\AddressII\\}\Town\\
  \DTLifnullorempty{\County}{}{\County\\}\Postcode\\\CountryName
  }
 }%
 {}% not subscribed
}
\end{document}
```

↓ Input

You can download or view this document.

Exercise 11. Mail Merging With Envelope Labels Using newlfm, envlab and datatool

There is some duplicate code in the previous example that's inefficient, especially if you have a large database. For this exercise, rewrite the document from Example 21 so that it only has one instance of each of the commands `\DTLforeach` and `\xDTLassignfirstmatch` and only one test for each member's subscribed status. Hint: recall the hook management described in §2.1.2. If you're feeling adventurous try to make the address on the labels upper case without using envlab's case-changing function. (Recall `\MakeUppercase` from Volume 2 [95, §5.1.1].)

You can download or view the solution.

4. Invoices

There are a number of bundles for typesetting invoices on CTAN (see the invoice topic) including the invoice package and the isodoc class. These are both available on MiKTEX and TEX Live and have English documentation available via `texdoc`.

The isodoc class has already been introduced in §3.4 as it can be used to create letters, so §4.1 describes how to use that class to create an invoice. However, it may be that you need to add an invoice to an existing document or need to use a particular document class, so §4.2 describes how to use the invoice package. Additionally, the invoice package converts currency (for foreign expenses) and computes your totals for you, whereas with isodoc you have to do the calculations yourself.

Finally, in case neither suit your requirements, §4.3 describes how to create your own custom invoice using the longtable and datatool packages.

4.1 Writing an Invoice Using the isodoc Class

The isodoc class [21] can be used to create either letters (see §3.4) or invoices. To generate an invoice you need to use:

`\invoice[`⟨*options*⟩`]{`⟨*contents*⟩`}` Definition

This is analogous to isodoc's `\letter` command discussed in §3.4. As with `\letter` a key=value list of options can be set using the optional argument ⟨*options*⟩ or using the command:

`\setupdocument{`⟨*options*⟩`}` Definition

In addition to those described in §3.4, there are also some options that relate to invoices. Some of these are described below. See the isodoc user guide [21] for details of the options not described in this book, which are omitted for brevity.

Some of the options that set payment information are listed below:

`term`	The payment term in days.
`currency`	Currency (default is euro).
`accountno`	Bank account number.

`routingno`	The bank's routing number (may be omitted).
`accountname`	Bank account name (may be omitted).
`iban`	Your International Bank Account Number (enter in lower case).
`bic`	Your Bank Identifier Code (enter in lower case).
`vatno`	Your VAT number.

The options that set the payment acceptance part:

`accept`	Show the acceptance data.
`acceptaccount`	Payer's bank account number.
`acceptaddress`	Payer's address (separate lines with `\\`).
`accepteuros`	Euros (or equivalent) part of the amount to be paid.
`acceptcents`	Cents (or equivalent) part of the amount to be paid.
`acceptdescription`	Description.

`acceptreference` Reference.

As with `\letter` you may have multiple `\invoice` commands within a document, but the ⟨*contents*⟩ part is more complicated. This argument will typically contain:

`\itable{`⟨*contents*⟩`}` Definition

This creates a two-column tabular-like environment with the given contents. You may use `&` and `\\` but the isodoc class provides some convenient commands to do this for you:

`\iitem{`⟨*item description*⟩`}{`⟨*amount*⟩`}` Definition

This puts ⟨*item description*⟩ in the first column and ⟨*amount*⟩ in the second column.

`\itotal[`⟨*tag*⟩`]{`⟨*amount*⟩`}` Definition

This adds a row for the total amount. The optional argument may be used to insert a tag, such as “Subtotal”.

In addition to `\itable`, you can also use

`\accountdata` Definition

To generate a table containing the account information needed to pay the invoice.

Example 22. An Invoice (isodoc class)
The above can be put together to create a simple invoice, listed below. Some of the options used in this example were introduced in §3.4.

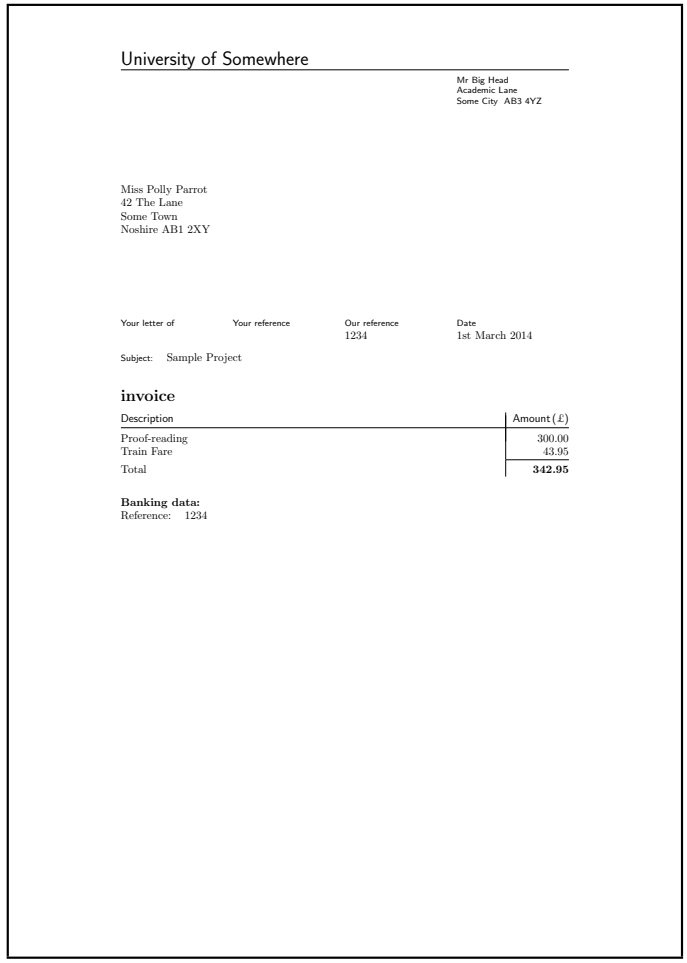
Input

```
\documentclass{isodoc}

\setupdocument
{
  language={en-GB},
  company={University of Somewhere},
  who={Mr Big Head},
  street={Academic Lane},
  city={Some City},
  zip={AB3 4YZ},
  country={United Kingdom},
  countrycode={GB},
  areacode={44},
  cityzip,
  date=2014-03-01,
  subject=Sample Project,
```

```
  currency={\pounds}
}

\begin{document}
\invoice
[
  ourref=1234,
  to={Miss Polly Parrot\\42 The Lane\\Some Town\\Noshire AB1 2XY}
]
{%
  \itable
  {%
    \iitem{Proof-reading}{300.00}
    \iitem{Train Fare}{43.95}
    \itotal{342.95}
  }
  \\[3ex]\accountdata
}
\end{document}
```

↓ Input

(You can download or view this example.) The resulting document is shown in Figure 4.1.



University of Somewhere

Mr Big Head
Academic Lane
Some City AB3 4YZ

Miss Polly Parrot
42 The Lane
Some Town
Noshire AB1 2XY

Your letter of	Your reference	Our reference	Date
		1234	1st March 2014

Subject: Sample Project

invoice

Description	Amount (£)
Proof-reading	300.00
Train Fare	43.95
Total	**342.95**

Banking data:
Reference: 1234

Figure 4.1 Invoice Using the isodoc Class

Exercise 12. Creating an Invoice for a Customer (isodoc class)
Create an invoice to be sent to José Arara at Nenhuma Rua, São Paulo, 123457, Brazil for 1 copy of the hardback book "'Duck and Goose': an allegory for modern times?" at 59.99, 20 copies of the paperback book "My Friend is a Duck" at 14.99 per copy, and 1 copy of the ebook "Annotated Notes on the 'Duck and Goose' chronicles" at 8.99. There is a promotional discount of 2.50 for this order. The cost of postage and packaging is 20.00. You will need to calculate the total for yourself.

You can download or view a solution.

For the More Adventurous
Instead of explicitly writing the customer and order information, fetch the values either from the sample CSV files (§2.5.1) or via two join statements on the tables in the `samples` SQL database (§2.5.3). This order can be identified by the row with the "id" field equal to 2 in the sample `ordergroups.csv` file or `ordergroups` SQL table. If you use the SQL database, you can perform the summations in the `SELECT` statement. If you use the CSV data, then it's more complicated. Although you can perform calculations using the commands described in §2.1.3, those commands can't be used within `\itable` as it uses the tabularx package [13] to layout the table which processes its contents multiple times which will throw the calculations out.

Instead you will need to employ the type of method described in Exercise 11 or perform two iterations over the data.

You can download or view a solution using the CSV files or download or view a solution using the SQL database.

4.2 Writing an Invoice Using the invoice Package

At the time of writing, the current version of invoice is 0.9 (dated 2011-10-01). This loads the fp package using `\input` rather than `\usepackage`, which has unfortunate side-effects as it indirectly loads `fp.sty` through `fp.tex` but `fp.tex` is intended for Plain TEX and messes with the definition of `\ProvidesPackage` and `\RequirePackage`. Until this is fixed, you need to make sure that you load invoice after all your other packages.

The invoice package [17] defines the invoice environment that's used to generate the invoice.

`\begin{invoice}{`⟨*base currency*⟩`}{`⟨*VAT*⟩`}` Definition

The first argument ⟨*base currency*⟩ is the currency name and the second argument ⟨*VAT*⟩ is the VAT percentage (without the percent sign). The ⟨*VAT*⟩ may be `0`, in which case the VAT entries are hidden from the invoice,

or 0.0, in which case the VAT entries are displayed but show zero-rated VAT.

⚠ Take care if you want to use a dollar sign. An error will occur if you use \$ as the base currency *unless* you also load the fontenc package with the T1 option. (Other options may also work, but the default OT1 font type fails.) This problem can also happen with other currency commands, such as \textdollar (textcomp package [58]) and \pounds, so where possible use fontenc with invoice. If for some reason you don't want to load fontenc, then you can use \string$ instead of \$ as a workaround. However, it's a good idea in general to use the fontenc package anyway. If you use fontenc remember to use inputenc as well.

[FAQ: Why bother with *inputenc* and *fontenc*?]

Within the invoice environment you can set the project title using:

\ProjectTitle{⟨*title*⟩} Definition

You must have at least one project title in your invoice environment.

After the project title you specify the fees using:

\Fee{⟨*description*⟩}{⟨*rate/unit*⟩}{⟨*count*⟩} Definition

the local expenses using:

\EBC{⟨*description*⟩}{⟨*amount*⟩} Definition

and the foreign expenses using:

\EFC{⟨description⟩}{⟨foreign currency⟩}{⟨amount⟩}{⟨conversion rate⟩}{⟨base currency result⟩} Definition

You may have multiple instances of these commands. Either the fees or the expenses may be omitted, but if both are present the fees must come first.

The fees must have a description (first argument), the rate per unit of work (second argument) and the number of units (third argument). For example, a fee for proof-reading a document at a cost of £150 per day for two days:

```
\Fee{Proof-reading}{150}{2}
```
Input

The local expenses must have a description (first argument) and the amount (second argument). For example, to claim the cost of a £43.95 train ticket:

```
\EBC{Train fare}{43.95}
```
Input

The foreign expenses must have a description (first argument), the name of the foreign currency (second argument), the cost in terms of the foreign currency unit (third argument), the conversion rate (fourth argument) and the result of the currency conversion (the fifth argument). One or other of the last two arguments may be empty.

For example, to charge for hotel accommodation at €300 with an exchange rate of 0.82:

```
\EFC{Hotel}{\texteuro}{300}{0.82}{}
```
Input

or to charge for hotel accommodation at €300 with a local currency value of 246.67:

```
\EFC{Hotel}{\texteuro}{300}{}{246.67}
```
Input

(Note that if you want to use \texteuro as in this example, you need to load the textcomp package [58].)

You can hide expenses that should contribute to the total but don't need to be itemized using:

```
\EBCi{⟨description⟩}{⟨amount⟩}
```
Input

for local expenses and

```
\EFCi{⟨description⟩}{⟨foreign currency⟩}{⟨amount⟩}{⟨conversion rate⟩}
{⟨base currency result⟩}
```
Input

for foreign expenses. The arguments are the same as for \EBC and \EFC. You can make a subtotal appear for all the hidden expenses using:

\STExpenses Definition

You may also specify a discount using:

\Discount{⟨*description*⟩}{⟨*amount*⟩} Definition

where ⟨*description*⟩ is a description about the discount and ⟨*amount*⟩ is the amount of the discount in terms of the base currency unit.

Example 23. An Invoice (invoice package)
The above can be put together to form a simple invoice:

Input

```
\begin{invoice}{\pounds}{20}
 \ProjectTitle{Sample Project}
 \Fee{Proof-reading}{150.0}{2}
 \EBC{Train fare}{43.95}
 \EFC{Hotel}{\texteuro}{300}{0.82}{}
\end{invoice}
```

Input

This produces the invoice shown in Figure 4.2. You can download or view this example.

Output

Sample Project

Activity		Rate/Unit	Count	Amount (£)
Proof-reading		150.0	2	300.00
VAT (20%)				60.00
Expense	Currency	Amount	Factor	£
Train fare	£			43.95
Hotel	€	300	0.82	246.00
Sum Fees				300.00
Sum VAT				60.00
Sum Expenses				289.95
Total				**649.95**

Figure 4.2 Sample Invoice (invoice package)

The invoice package provides some multilingual support so you can use it with babel [7]. If there is no support for your language, follow the instructions in the file `invoice.def` which is located in the same directory as `invoice.sty`. Alternatively, you can redefine the command names that generate the invoice tags if they don't suit your purpose. For example:

Input

```
\renewcommand{\Fees}{Products}
\renewcommand{\UnitRate}{Price}
\renewcommand{\Count}{Quantity}
\renewcommand{\Activity}{Product}
```

Input

Exercise 13. Creating an Invoice for a Customer (invoice package)

This exercise is like Exercise 12 except that now you need to create the document using the invoice package instead of the isodoc class (and you don't need to compute the total here as the invoice package does it for you).

Create an invoice to be sent to José Arara at Nenhuma Rua, São Paulo, 123457, Brazil for 1 copy of the hardback book "'Duck and Goose': an allegory for modern times?" at 59.99, 20 copies of the paperback book "My Friend is a Duck" at 14.99 per copy, and 1 copy of the ebook "Annotated

Notes on the 'Duck and Goose' chronicles" at 8.99. There is a promotional discount of 2.50 for this order. The cost of postage and packaging is 20.00.

You can choose the currency unit to suit your location, although some exchange rates might make these seem either very cheap or very expensive books, but don't worry about that. (Remember to use textcomp and fontenc packages for the currency symbol.) In the UK, physical books are zero-rated but ebooks are subject to the standard 20% VAT rate, so for simplicity assume that the ebook price includes VAT and just use 0 to hide VAT from the invoice.

(Remember that invoice is a package not a class file, so you need to choose an appropriate class to use with it. For example, you might want to use it with the letter class so you can print it on headed paper.) You can download or view a solution.

For the More Adventurous

Instead of explicitly writing the customer and order information, fetch the values either from the sample CSV files (§2.5.1) or via two join statements on the tables in the `samples` SQL database (§2.5.3). This order can be identified by the row with the "id" field equal to 2 in the sample `ordergroups.csv` file or `ordergroups` SQL table.

You can download or view a solution using the CSV files or download or view a solution using the SQL database.

4.3 Building Your Own Invoice using longtable and datatool

It may be that the layouts produced by the available classes or packages don't suit your requirements or perhaps you want to use isodoc but the \itable layout doesn't have enough columns for your needs. Since invoices typically involve aligning data in rows and columns, you can just use the tabular environment described in Volume 1 [92, §4.6] if it can fit within a single page. If the tabulated data exceeds a page, then you need to use a multi-paged tabular-like environment, such as the longtable environment provided by the longtable package [11].

The syntax for longtable is similar to tabular:

\begin{longtable}[⟨*horizontal alignment*⟩]{⟨*column specs*⟩} Definition

Unlike tabular the optional argument specifies the horizontal alignment instead of the vertical alignment since longtable isn't intended for in-line positioning. The horizontal alignment may be one of `l` (left), `c` (centre) or `r` (right). The default is `c`.

The ⟨*column specs*⟩ are the same as for tabular and the rows and columns are separated in the same way using \\ and &. For example:

Input

```
\begin{longtable}{lr}
Video & 8.99\\
CD & 9.11\\
DVD & 15.00\\
Total & 33.10
\end{longtable}
```

Input

produces:

↑ Output

Video	8.99
CD	9.11
DVD	15.00
Total	33.10

↓ Output

However, unlike tabular, you can also specify a caption as well as header and footer information. This is done at the start of the longtable environment:

```
\begin{longtable}{⟨column specs⟩}
\caption{⟨first page caption⟩}
\label{⟨table label⟩}\\
⟨code for first page header row⟩
\endfirsthead
\caption{⟨continuation caption⟩}
⟨code for the header row⟩
\endhead
```

⟨*code for last page footer row*⟩
\endlastfoot
⟨*code for the footer row*⟩
\endfoot
⟨*contents*⟩
\end{longtable}

You may omit any of the header, footer or caption information. The document will need at least two LaTeX runs to ensure the longtable environment correctly aligns the columns. The header and footer code will typically need to include & and \\ as per the column alignment specifications. You can use the starred version of \caption to suppress the numbering for example:

```
\caption*{Products (continued)}
```
Input

Example 24. Multi-Paged Tabulated Material

Recall from Example 4 that datatool's \DTLdisplaylongdb command could be used to display the contents of a large database over multiple pages. This command internally uses the longtable environment. We could instead explicitly use that environment to display the data from the sample country-codes.csv file or countries SQL table:

↑ Input

```
\begin{longtable}{cl}
 \bfseries Country Code & \bfseries Country Name\\
 \endhead
 \multicolumn{2}{r}{\emph{Continued on next page}}
 \endfoot
 \endlastfoot
 \DTLforeach*{countries}{\Code=code,\Name=name}{\Code & \Name\\}%
\end{longtable}
```

↓ Input

This sets the same header for all the pages and sets the footer to the text "Continued on next page". Since this isn't required for the final page, the last footer is set to empty. This produces a seven page table (without a caption). The beginning of the table looks like:

↑ Output

Country Code	Country Name
ad	Andorra
ae	United Arab Emirates
af	Afghanistan
ag	Antigua and Barbuda
ai	Anguilla

↓ Output

The end of the first page looks like:

↑ Output

bw	Botswana
by	Belarus
bz	Belize
ca	Canada

Continued on next page

↓ Output

The final page of the table is shown in Figure 4.3. You can download or view this example document.

Output

Country Code	Country Name
vg	Virgin Islands, British
vi	Virgin Islands, USA
vn	Viet Nam
vu	Vanuatu
wf	Wallis and Futuna
ws	Samoa
ye	Yemen
yt	Mayotte
za	South Africa
zm	Zambia
zw	Zimbabwe

Figure 4.3 Final Page of longtable Displaying Countries

Remember that you can use the booktabs package [24] if you want horizontal rules and you can use \DTLiflastrow to suppress the final \\ which would otherwise create unnecessary extra vertical space at the end of the table. For example:

Input

```
\begin{longtable}{cl}
 \bfseries Country Code & \bfseries Country Name\\
 \midrule
 \endhead
 \bottomrule
 \multicolumn{2}{r}{\emph{Continued on next page}}
 \endfoot
 \bottomrule
 \endlastfoot
 \DTLforeach*{countries}{\Code=code,\Name=name}%
 {\Code & \Name\DTLiflastrow{}{\\}}%
\end{longtable}
```

Input

Alternatively you can use \DTLiffirstrow:

↑ Input

```
\begin{longtable}{cl}
 \bfseries Country Code & \bfseries Country Name\\
 \midrule
 \endhead
 \bottomrule
 \multicolumn{2}{r}{\emph{Continued on next page}}
 \endfoot
 \bottomrule
 \endlastfoot
 \DTLforeach*{countries}{\Code=code,\Name=name}%
 {\DTLiffirstrow{}{\\}\Code & \Name}%
\end{longtable}
```

↓ Input

As with tabular, the column specifiers may include p{⟨*width*⟩} for a column with multilined cells. Recall from Volume 1 [92, §4.6] that the array package [57] can be used to insert a declaration before each cell in a given column via:

>{⟨*declaration*⟩} Definition

directly before the column specifier. This means that if you have a column for the description of an invoiced item, you can have ragged line wrapping, which looks better than the default fully-justified paragraphs in a narrow column context.

Example

Suppose my invoice needs four columns: the item description, the quantity ordered, the unit price and the quantity times price. The last three columns can just use the r specifier, but the first column may need a paragraph cell in the event of a long description:

Input

```
\begin{longtable}{>{\raggedright}p{0.3\linewidth}rrr}
\bfseries Item & \bfseries Quantity &
\bfseries Unit Price (\pounds) &
\bfseries Price (\pounds)\\
\midrule
\endhead
``\,`Duck and Goose': an
allegory for modern times?'' (hardback) &
1 & 59.99 & 59.99\\
```

```
``My Friend is a Duck'' (paperback) &
20 & 14.99 & 299.80\\
``Annotated Notes on the `Duck and
Goose' Chronicles'' (ebook) &
1 & 8.99 & 8.99\\
``The Adventures of Duck and Goose'' (hardback) & 1 & 18.99
& 18.99\\
\midrule
\multicolumn{3}{r}{\bfseries Sub-Total} & 368.78\\
\multicolumn{3}{r}{\bfseries Postage and Packaging} & 20.00\\
\multicolumn{3}{r}{\bfseries Promotional Discount} & $-2.50$\\
\midrule
\multicolumn{3}{r}{\bfseries Total} & 386.28
\end{longtable}
```

↓ Input

This produces:

↑ Output

Item	Quantity	Unit Price (£)	Price (£)
"'Duck and Goose': an allegory for modern times?" (hardback)	1	59.99	59.99
"My Friend is a Duck" (paperback)	20	14.99	299.80
"Annotated Notes on the 'Duck and Goose' Chronicles" (ebook)	1	8.99	8.99
"The Adventures of Duck and Goose" (hardback)	1	18.99	18.99
		Sub-Total	387.77
		Postage and Packaging	20.00
		Promotional Discount	−2.50
		Total	405.27

↓ Output

Exercise 14. Custom Invoice

Adapt Exercise 12 so that it uses longtable instead of `\itable`, and make it have separate columns for the quantity and unit price (as in the example above). In addition, let's now suppose the book prices exclude VAT. The physical books are zero-rated, but the ebooks are standard-rated at 20%. Add an extra column that indicates the VAT rating and add a row for the VAT after the subtotal. You can download or view a solution.

For the More Adventurous

As with the more adventurous part of Exercise 12, fetch the information from the sample CSV files or SQL database. You can download or view a solution for the CSV files or download or view a solution for the SQL data.

5. Curricula Vitæ (Résumés)

There are a number of classes and packages on CTAN for typesetting a curriculum vitæ (CV) or résumé. In fact, there's a surprisingly large list of them on the cv topic page. However, of the LaTeX options (as opposed to Plain TeX) some of them are old LaTeX2.09 styles and some have licence issues which prevent them from being included in TeX Live.[1] On the assumption that you're reading this chapter because you want to write a CV and are possibly pressed for time, I decided to describe only two of these options for brevity.

[FAQ: *Curriculum Vitae* (Résumé)]

I discounted the non-LaTeX options and the obsolete LaTeX2.09 styles, and additionally discounted those that aren't on TeX Live, since some readers may not want to fiddle around with manual installation. I also discounted bundles that don't have proper documentation (for example, just a README or sample file was provided). Due to my poor multilingual skills, I also discounted bundles that don't have English documentation.

[1] TeX Live is more restrictive than MiKTeX as it won't include any bundles that don't have an open source licence or don't provide source code for all the documentation. This means that there are some packages or classes that are available on MiKTeX but are not on TeX Live.

This still left me with more than two options to choose from, so my final selection was based on how easy I found it to create a working example from the documentation. The top two were: the currvita package and the europecv class. The former, currvita, is simple and easy to use. The latter, europecv, follows the European Union CV guidelines, although the Europass title and logo can be suppressed for non-EU users. If these don't suit you, or if you are fluent in French, German or Chinese, you may prefer to investigate the other choices listed on the cv topic page.

5.1 The currvita Package

The currvita package [71] is quite simple, and should work with most classes. For example, you could just use it with the base article class or you might want to include your CV in a letter, in which case you might want to use one of the letter-like classes.

The available package options are as follows:

`LabelsAligned` Produces more compact vertical spacing.

`TextAligned` Produces more generous vertical spacing. This is the default vertical spacing option.

`openbib` Produces an "open" format for the bibliography.

`ManyBibs` This option is provided for use with the bibunits [30] and multibib [31] packages and allows you to subdivide your publication list.

`NoDate` This option suppresses the date that by default is displayed at the bottom of the CV.

The date is set using:

`\date{`⟨*date*⟩`}` Definition

(As per `\maketitle`.) If you also want to specify a location next to the date, you can use:

`\cvplace{`⟨*location*⟩`}` Definition

The body of the CV is contained within the cv environment:

`\begin{cv}{`⟨*heading*⟩`}` Definition

where ⟨*heading*⟩ is the title text, such as "Résumé" or "Curriculum Vitae" (or "Curriculum Vitæ" if you prefer to use a ligature).

The contents of the CV are typically divided into sections containing lists. These sections can be typeset within the cv environment using the cvlist environment:

\begin{cvlist}{⟨section heading⟩} Definition

where ⟨*section heading*⟩ is the heading text for this list. Within the body of the cvlist environment, use the standard

\item[⟨*label*⟩] Definition

command to start each item.

Example 25. A Sample CV

The source code for this book loads the currvita package, so I can just use the cv environment within this document:

↑ Input

```
\date{10th March 2014}
\cvplace{My Office}
\begin{cv}{R\'esum\'e}
  \begin{cvlist}{Personal Information}
    \item[Name:]  Polly Parrot
    \item[Address:] 42 The Lane, Some Town,
      Noshire AB1 2XY, United Kingdom
    \item[Telephone:] 0123456789
    \item[Email:] polly.parrot@example.com
```

```
    \item[Nationality:] British
  \end{cvlist}
\end{cv}
```

↓ Input

This produces:

↑ Output

Résumé

Personal Information

Name:	Polly Parrot
Address:	42 The Lane, Some Town, Noshire AB1 2XY, United Kingdom
Telephone:	0123456789
Email:	polly.parrot@example.com
Nationality:	British

My Office, 10th March 2014

↓ Output

(You can download or view a complete sample document.)

Recall from Volume 2 [95, §5] that you can generate a list of citations using BibTEX or `biber`. In a CV it's likely that you will want to include a list of publications without citing them. In this case, instead of using

`\cite{`⟨*key list*⟩`}` Definition

you can use

`\nocite{`⟨*key list*⟩`}` Definition

to add the citations referenced in the comma-separated list ⟨*key list*⟩ without producing any text. Alternatively, you can add all entries defined in your `.bib` file using an asterisk:

```
\nocite{*}
```
Input

So if you want to include a list of your publications you can use `\nocite` with `\bibliography` and `\bibliographystyle` (as described in Volume 2 [95, §5.2]).

Exercise 15. Sample CV with Publications
Modify Example 25 so that it also includes a publications list. You can either use your own `.bib` file or you can use the test `xampl.bib` file that's included in TEX distributions.

You can download or view a solution.

5.2 The europecv Class

The europecv class [111] is designed for CVs that follow the common format defined by the European Commission in 2002, but it can also be used for people outside the European Union. By default, this class will display the Europass logo and title, so you will need to include the graphicx package if you want this setting. If you're outside the European Union, you can suppress the Europass logo and title using the class options `nologo` and `notitle`:

```
\documentclass[nologo,notitle]{europecv}
```
Input

If you are in the European Union then you also need to switch to narrow Helvetica using the class options `helvetica` and `narrow`:

```
\documentclass[helvetica,narrow]{europecv}
\usepackage{graphicx}
```

Input

Input

Or if you have Arial installed:

```
\documentclass[arial,narrow]{europecv}
\usepackage{graphicx}
```

Input

Input

The default paper size is US Letter, so if you want A4 paper, you need to use the a4paper option:

```
\documentclass[helvetica,narrow,a4paper]{europecv}
\usepackage{graphicx}
```

Input

Input

The body of the CV is placed in the europecv environment. So a minimal document that displays the Europass logo can be obtained with:

Input

```
\documentclass[helvetica,narrow,a4paper]{europecv}
\usepackage{graphicx}

\begin{document}
  \begin{europecv}
  \end{europecv}
\end{document}
```

Input

If you're not writing in English, you can set the language using the class option. This adjusts the predefined text used by europecv but doesn't load babel, so if you need babel (for example, you want to use the hyphenation patterns for your language) you have to load it yourself in the preamble. The europecv class automatically loads the inputenc package with the default input encoding is set to `utf8x`. If you are using a different input encoding you must set it in the class options. Remember that you also need to load the fontenc package.

Example:
If you are writing in French and using Latin 1 encoding:

Input

```
\documentclass[helvetica,narrow,latin1,french]{europecv}
\usepackage[T1]{fontenc}
\usepackage{babel}
```

Input

(babel will pick up the language option from the class option list.)

There are other class options as well, such as `totpages`, which will print the total number of pages on each page. See the europecv documentation [111] for further details.

5.2.1 Setting Personal Information

Your personal information is specified using commands that are akin to `\title` and `\author`. That is the commands just store information, so it's best to use them in the preamble.

`\ecvname{`⟨*name*⟩`}`

Definition

Specifies your name.

\ecvfootername{⟨*name*⟩} Definition

Specifies your name as it will appear in the footer. If omitted, it will be the same as set by \ecvname.

\ecvaddress{⟨*address*⟩} Definition

Specifies your address. The address will line-wrap, but if you want to force a line break you need to use \newline not \\.

\ecvtelephone[⟨*mobile*⟩]{⟨*telephone*⟩} Definition

Sets your telephone number and optionally mobile phone number.

\ecvfax{⟨*fax*⟩} Definition

Specifies your fax number.

\ecvemail{⟨*email address*⟩} Definition

Specifies your email address.

\ecvnationality{⟨*nationality*⟩} Definition

Specifies your nationality.

\ecvdateofbirth{⟨*date*⟩} Definition

Specifies your date of birth.

\ecvgender{⟨*gender*⟩} Definition

Specifies your gender.

\ecvpicture[⟨*options*⟩]{⟨*image filename*⟩} Definition

Specifies the name of the graphics file that contains an image of yourself. The optional argument are as used by \includegraphics. You can also include text before and after the image using:

\ecvbeforepicture{⟨*text*⟩} Definition

and

\ecvafterpicture{⟨*text*⟩} Definition

Within ⟨*text*⟩, you may add some vertical space using:

\ecvspace{⟨*height*⟩} Definition

to make some minor adjustments to the picture's position. This command can't be used outside the argument of \ecvbeforepicture or \ecvafterpicture.

Once you have specified all your personal details, as described above, you need to use:

`\ecvpersonalinfo[⟨vspace⟩]` Definition

within the `europecv` environment to display the information (analogous to `\maketitle`). The optional argument may be used to insert extra vertical space after the personal information section.

Example 26. Personal Information Section (europecv class)
This example uses the `me.pdf` sample file. You can change it as appropriate.

↑ Input

```
\documentclass[helvetica,narrow,a4paper]{europecv}

\usepackage[T1]{fontenc}
\usepackage{graphicx}

% Specify personal data:
\ecvname{Polly Parrot}
\ecvaddress{42 The Lane, Some Town, Noshire AB1 2XY,
United Kingdom}
\ecvtelephone[0712345678]{0123456789}
```

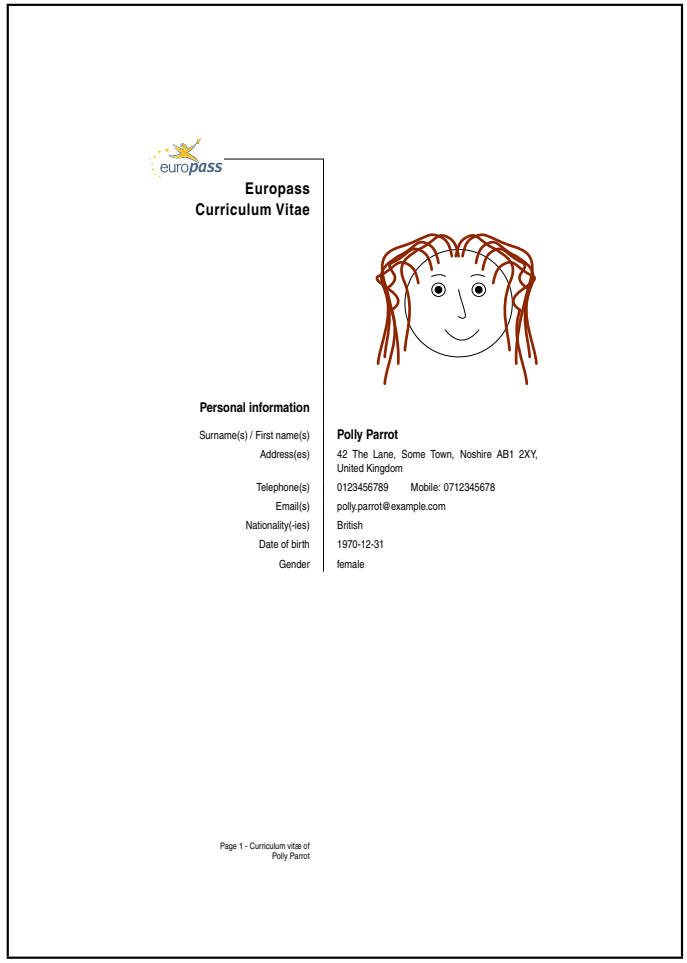
```
\ecvemail{polly.parrot@example.com}
\ecvnationality{British}
\ecvdateofbirth{1970-12-31}
\ecvgender{female}
\ecvpicture[width=2in]{me}% me.pdf image file

\begin{document}
  \begin{europecv}
  % display personal data:
  \ecvpersonalinfo
  \end{europecv}
\end{document}
```

↓ Input

(You can download or view this example.) The resulting document is shown in Figure 5.1.

5.2.2 Sections and Publication Lists

Sections in your CV are created using:



europass

Europass Curriculum Vitae

Personal information

Surname(s) / First name(s)	**Polly Parrot**
Address(es)	42 The Lane, Some Town, Noshire AB1 2XY, United Kingdom
Telephone(s)	0123456789 Mobile: 0712345678
Email(s)	polly.parrot@example.com
Nationality(-ies)	British
Date of birth	1970-12-31
Gender	female

Page 1 - Curriculum vitæ of Polly Parrot

Figure 5.1 Personal Information Section (europecv class)

`\ecvsection[`⟨*vspace*⟩`]{`⟨*title*⟩`}` Definition

where ⟨*title*⟩ is the section title and ⟨*vspace*⟩ is the height of the vertical space that can optionally be inserted after the title.

The text within the section is specified using:

`\ecvitem[`⟨*vspace*⟩`]{`⟨*left*⟩`}{`⟨*right*⟩`}` Definition

where ⟨*left*⟩ is the text to place on the left of the vertical rule and ⟨*right*⟩ is the text to place on the right of the vertical rule. The optional argument is again the height of the vertical space that can be inserted after the text.

Example 27. Curriculum Vitæ With Sections (europecv class)
This example adds to the code from Example 26 so that it includes a section listing professional positions.

↑ Input

```
\documentclass[helvetica,narrow,a4paper]{europecv}

\usepackage[T1]{fontenc}
\usepackage{graphicx}

% Specify personal data:
```

```
\ecvname{Polly Parrot}
\ecvaddress{42 The Lane, Some Town, Noshire AB1 2XY,
United Kingdom}
\ecvtelephone[0712345678]{0123456789}
\ecvemail{polly.parrot@example.com}
\ecvnationality{British}
\ecvdateofbirth{1970-12-31}
\ecvgender{female}
\ecvpicture[width=2in]{me}% me.pdf image file

\begin{document}
  \begin{europecv}
  % display personal data:
  \ecvpersonalinfo

   \ecvsection{Professional Positions}
   \ecvitem{1990--8}{Junior assistant at
   ``Wibblies Avian Emporium''.}
   \ecvitem{1998--Present}{Senior assistant at
    ``The International Society of Duck and Geese
Co-operation''.}
```

```
  \end{europecv}
\end{document}
```

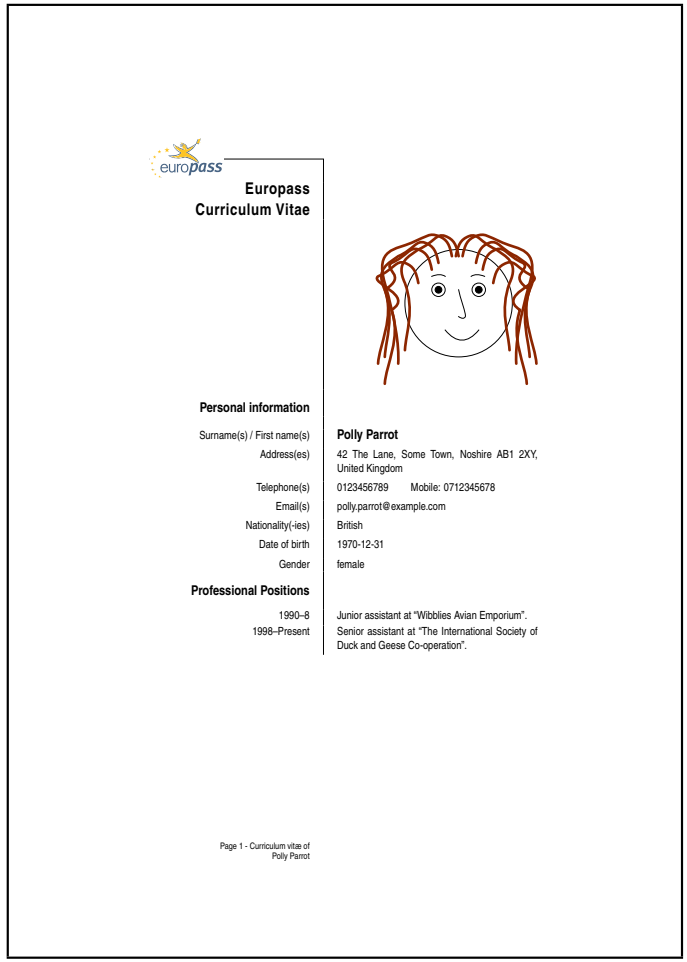
↓ Input

(You can download or view this document.) The resulting document is shown in Figure 5.2.

Bibliographies are more problematic as the contents of the europecv environment are set using a longtable environment (which is why you have to use \ecvsection rather than \section). This means that you can't just use \bibliography within the right argument of \ecvitem if the bibliography is likely to span a page break. You can, however, simply place the bibliography outside the europecv environment, as shown below:

↑ Input

```
\begin{document}
  \begin{europecv}
  % display personal data:
  \ecvpersonalinfo
  % start a new section:
  \ecvsection{Professional Positions}
  \ecvitem{1990--8}{Junior assistant at
```



europass

Europass Curriculum Vitae

Personal information

Surname(s) / First name(s)	**Polly Parrot**
Address(es)	42 The Lane, Some Town, Noshire AB1 2XY, United Kingdom
Telephone(s)	0123456789 Mobile: 0712345678
Email(s)	polly.parrot@example.com
Nationality(-ies)	British
Date of birth	1970-12-31
Gender	female

Professional Positions

1990–8	Junior assistant at "Wibblies Avian Emporium".
1998–Present	Senior assistant at "The International Society of Duck and Geese Co-operation".

Page 1 - Curriculum vitæ of Polly Parrot

Figure 5.2 Curriculum Vitæ Sections (europecv class)

```
  ``Wibblies Avian Emporium''.}
  \ecvitem{1998--Present}{Senior assistant at
   ``The International Society of Duck and Geese Co-operation''.}
  \end{europecv}

  \bibliographystyle{plain}
  \nocite{*}
  \bibliography{mypublications}
\end{document}
```

↓ Input

An alternative is to use the bibentry package[2] (part of the natbib bundle [19]), which provides:

\nobibliography{⟨*bib file*⟩} Definition

which is analogous to \bibliography except that it doesn't display the bibliography, and:

\bibentry{⟨*key*⟩} Definition

which displays the citation information for the reference identified by ⟨*key*⟩.

[2] The documentation for bibentry is inside the `bibentry.sty` file.

Example:

Input

```
\documentclass[helvetica,narrow,a4paper]{europecv}

\usepackage[T1]{fontenc}
\usepackage{graphicx}
\usepackage{bibentry}

% add personal data here

\begin{document}
  \bibliographystyle{plain}
  \nocite{*}
  \nobibliography{mypublications}
  \begin{europecv}
  % display personal data:
  \ecvpersonalinfo
  % start a new section:
  \ecvsection{Professional Positions}
  \ecvitem{1990--8}{Junior assistant at
```

```
  ``Wibblies Avian Emporium''.}
  \ecvitem{1998--Present}{Senior assistant at
   ``The International Society of Duck and Geese Co-operation''.}
  % publications section:
  \ecvsection{Publications}
  \ecvitem{}{\bibentry{mypub1}}
  \ecvitem{}{\bibentry{mypub2}}
  \end{europecv}
\end{document}
```

↓ Input

This assumes a file called `mypublications.bib` contains the bibliography database, including the citations with the labels `mypub1` and `mypub2`. Remember this requires a BibTEX run between LATEX runs (see Volume 2 [95, §5]).

Another possibility is to use the databib package (part of the datatool bundle). This has its own `.bst` BibTEX style that converts the bibliography data into one of datatool's internal databases. This is done using:

`\DTLloadbbl[`⟨*bbl*⟩`]{`⟨*db-name*⟩`}{`⟨*bib list*⟩`}` Definition

where ⟨*bbl*⟩ is the name of the `.bbl` file (defaults to `\jobname.bbl`), ⟨*db-name*⟩ is the name of the new database and ⟨*bib list*⟩ is the list of `.bib` files (without the `.bib` extension) where the bibliography data is stored.

As with \bibliography (see §5.1 and Volume 2 [95, §5.2]) you need to specify which citations you want included in the .bbl file either via:

\cite{⟨key list⟩} Definition

which also displays a reference in the text, or

\nocite{⟨key list⟩} Definition

which doesn't produce any text, but ensures that BibTEX includes the references in the .bbl file.

Example:

↑ Input

```
\nocite{*}
\DTLloadbbl{mypubdata}{myrefs}
```

↓ Input

This will create a datatool internal database called mypubdata that contains all the bibliographic data stored in the file myrefs.bib. (As with \bibliography, this requires a BibTEX run between LATEX runs to ensure the citations are up-to-date.)

This database can be iterated over using \DTLforeach (as described in §2.7.1). However, since many of the fields will be null depending on the entry type, it's easier to iterate over the entries using:

\DTLforeachbibentry[⟨*condition*⟩]{⟨*db-name*⟩}{⟨*body*⟩} Definition

This only makes local assignments, so it's no use in a tabular-like environment due to the scoping effect of & and \\. Instead you can use:

\gDTLforeachbibentry[⟨*condition*⟩]{⟨*db-name*⟩}{⟨*body*⟩} Definition

which makes global assignments.

As with \DTLforeach, there is also a starred version that performs a read-only iteration of the database. The optional argument is a conditional in the same format as the optional argument of \DTLforeach. There's no assignment list. Instead, you can access the citation key (as used by \cite and \bibitem) within ⟨*body*⟩ using:

\DBIBcitekey Definition

The entry type (for example, book) is stored in:

\DBIBentrytype Definition

(This will always be in lower case, regardless of the case used in the .bib file.) The remaining fields can be displayed using:

\DTLbibfield{⟨field name⟩} Definition

or they can be assigned to a control sequence ⟨cs⟩ using:

\DTLbibfieldlet{⟨cs⟩}{⟨field name⟩} Definition

In both cases, ⟨field name⟩ is the column label, but no check is performed to determine if the column exists, so the result may be a null value (see §2.9). Available field labels are: Address, Author, BookTitle, Chapter, Edition, Editor, HowPublished, Institution, Journal, Key, Month, Note, Number, Organization, Pages, Publisher, School, Series, Title, Type, Volume, Year, ISBN, DOI, PubMed, Abstract and Url. These labels are case-sensitive (independent of the case used in the .bib file).

You can determine if a field exists within the ⟨body⟩ part of \DTLforeachbibentry or \gDTLforeachbibentry using:

\DTLifbibfieldexists{⟨field label⟩}{⟨true part⟩}{⟨false part⟩} Definition

Since it's quite complicated working out which fields are relevant for which entry types, databib provides a convenient command that will format the entry in the current iteration according to its entry type:

\DTLformatbibentry Definition

By default, this only displays the fields that would typically be displayed using the standard plain bibliography style, so fields such as `Url` won't be displayed, even if they exist. This command also doesn't use `\bibitem` (recall Volume 1 [92, §5.6]). Since `\bibitem` internally uses `\item`, it's only appropriate in a list context, but it's possible it may be needed outside a list. Therefore databib provides:

`\DTLcustombibitem{`⟨*marker code*⟩`}{`⟨*ref text*⟩`}{`⟨*key*⟩`}` Definition

This is similar to `\bibitem[`⟨*label*⟩`]{`⟨*key*⟩`}`, except that it replaces `\item [`⟨*label*⟩`]` with ⟨*marker code*⟩ and sets the cross-reference text (that is, the reference text or number generated by `\cite{`key`}`) to ⟨*ref text*⟩. Unlike the other commands described above, `\DTLcustombibitem` may be used outside the ⟨*body*⟩ argument of both `\DTLforeachbibentry` and `\gDTLforeachbibentry`.

If you want to format a bibliographic entry outside of `\DTLforeachbibentry`/`\gDTLforeachbibentry` you can use:

`\DTLformatthisbibentry{`⟨*db-name*⟩`}{`⟨*cite key*⟩`}` Definition

where ⟨*db-name*⟩ is the label identifying the database and ⟨*cite key*⟩ is the label identifying the reference.

Example 28. Tabulating a Bibliography

Using the above commands, it's possible to display a bibliography within a longtable. Recall from Exercise 15 that TEX distributions come with an example file called `xampl.bib`. This has enough entries to test how well the code deals with page breaking, and so will be used in this example.

The references in this example will be numbered (rather than using an author and year system), so a counter is defined to keep track of the numbering. (See Volume 1 [92, §11].)

↑ Input

```
\documentclass{article}

\usepackage{longtable}
\usepackage{databib}

\newcounter{refcount}
\newcommand*{\refmark}{\refstepcounter{refcount}[\therefcount]}

\begin{document}
\nocite{*}
\DTLloadbbl{refdata}{xampl}
```

```
\section*{Publications}

\begin{longtable}{rp{0.5\textwidth}}
  \gDTLforeachbibentry{refdata}
  {%
   \DTLcustombibitem{\refmark}{\therefcount}{\DBIBcitekey} &
   \DTLformatbibentry\\
  }%
\end{longtable}
\end{document}
```

Input

This creates a four-paged document. The first page is shown in Figure 5.3. You can download or view this example document.

The data is unsorted, but remember that you can sort it using \DTLsort (see §2.4). For example, to sort in reverse chronological order:

```
\DTLsort*{Year=descending,Month=descending}{refdata}
```

Input

(The months should be specified using the BibTEX strings JAN, FEB etc, instead of explicitly writing the month names in order to sort the months in numerical order. You can redefine \DTLmonthname{⟨*number*⟩} to make the month names appear in a different language.)

Output

Publications

[1] L[eslie] A. Aamport. The gnats and gnus document preparation system. *G-Animal's Journal*, 1986.

[2] L[eslie] A. Aamport. The gnats and gnus document preparation system. *G-Animal's Journal*, 41(7):73+, July 1986. This is a full ARTICLE entry.

[3] L[eslie] A. Aamport. The gnats and gnus document preparation system. In *G-Animal's Journal* [4], pages 73+. This is a cross-referencing ARTICLE entry.

[4] *G-Animal's Journal*, 41(7), July 1986. The entire issue is devoted to gnats and gnus (this entry is a cross-referenced ARTICLE (journal)).

[5] Donald E. Knuth. *Fundamental Algorithms*, chapter 1.2. Addison-Wesley, 1973.

[6] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*, section 1.2, pages 10–119. Addison-Wesley, Reading, Massachusetts, second edition, 10 January 1973. This is a full INBOOK entry.

[7] Donald E. Knuth. *Fundamental Algorithms*, section 1.2. Volume 1 of *The Art of Computer Programming* [11], second edition, 1973. This is a cross-referencing INBOOK entry.

[8] Donald E. Knuth. *Seminumerical Algorithms*. Addison-Wesley, 1981.

[9] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, second edition, 10 January 1981. This is a full BOOK entry.

Figure 5.3 A Tabulated Bibliography (First Page)

However, the database will be empty on the first run, which means you'll get an error if you try to sort it. You can test if the database is empty using:

\DTLifdbempty{⟨*db-name*⟩}{⟨*true part*⟩}{⟨*false part*⟩} Definition

where ⟨*db-name*⟩ is the label identifying the database. For example:

↑ Input

```
\DTLifdbempty{refdata}
{}
{%
  \DTLsort*{Year=descending,Month=descending}{refdata}%
}
```

↓ Input

It's therefore possible to display your list of publications in the europecv environment, but it will be slow (since TEX is doing the sorting and it requires iterating over a database).

Example 29. List of Publications (europecv class)
This example again uses the sample `xampl.bib` file that comes with TEX distributions. You can replace `xampl` with the base name of your own personal `.bib` file.

↑ Input

```
\documentclass[helvetica,narrow,a4paper]{europecv}

\usepackage[T1]{fontenc}
\usepackage{graphicx}
\usepackage{databib}

% add personal data here

% citation marker code:

\newcounter{refcount}
\newcommand*{\refmark}{\refstepcounter{refcount}[\therefcount]}

\begin{document}
\nocite{*}
\DTLloadbbl{mypubdata}{xampl}
```

```
\DTLifdbempty{refdata}
{}
{%
  \DTLsort*{Year=descending,Month=descending}{mypubdata}
}
  \begin{europecv}
  % display personal data:
  \ecvpersonalinfo
  % start a new section
  \ecvsection{Professional Positions}
  \ecvitem{1990--8}{Junior assistant at
  ``Wibblies Avian Emporium''.}
  \ecvitem{1998--Present}{Senior assistant at
   ``The International Society of Duck and Geese Co-operation''.}
  % publications section
  \ecvsection{Publications}
  % iterate over bib data:
  \gDTLforeachbibentry*{mypubdata}%
  {%
    \ecvitem
    {% left column
      \DTLcustombibitem{\refmark}{\therefcount}{\DBIBcitekey}%
```

```
    }%
    {\DTLformatbibentry}% right column
  }%
  \end{europecv}
\end{document}
```

↓ Input

(You can download or view this document.)

You may not want to include all the citations defined in your .bib file. For example, you may only want to include your ten most recent publications or you may only want to include just your books or journal articles. This looks as though the solution can simply be obtained by filtering the rows of data, but unfortunately there's a problem: the optional argument of \gDTLforeachbibentry doesn't work within the longtable environment. Nor does \ifthenelse work within the final argument of \gDTLforeachbibentry when used inside longtable (although some of the etoolbox comparison commands, such as \ifnumless can).

When \gDTLforeachbibentry fails in a tabular-like environment, we can go back to the technique used in the solution to Exercise 11 and again employed in the more adventurous section of Exercise 12, where the commands described in §2.1.2 were used to first build a command that could subsequently be used in the problematic environment.

Exercise 16. List of Selected Publications (europecv class)

Bearing in mind the above note, modify Example 29 so that it just lists the ten most recent publications. You can keep track of the current row within `\DTLforeachbibentry` with the DTLbibrow counter. As with `\DTLforeach`, you can prematurely terminate the loop at the end of the current iteration by placing `\dtlbreak` anywhere within ⟨*body*⟩.

Hint: you will need to use the standalone `\DTLformatthisbibentry` command instead of `\DTLformatbibentry` and you can check if one integer value is less than another integer value using etoolbox's

`\ifnumless{`⟨*number 1*⟩`}{`⟨*number 2*⟩`}{`⟨*true part*⟩`}{`⟨*false part*⟩`}` Definition

If you want to compare the value of a counter, you need to use

`\value{`⟨*counter name*⟩`}` Definition

in ⟨*number 1*⟩ or ⟨*number 2*⟩.

You can download or view the solution to this part of the exercise.

For the More Adventurous

Instead of just the ten most recent publications, split the publication list into the three most recent articles, the three most recent books, the two most recent conference proceedings and the two most recent booklets.

You can download or view the solution to this part of the exercise.

5.2.3 Spoken Languages

The europecv class also has commands to produce the section on your spoken language skills. This section is started by identifying your mother tongue using:

`\ecvmothertongue[`⟨*vspace*⟩`]{`⟨*language*⟩`}` Definition

For example:

```
\ecvmothertongue{English}
```
Input

The language table header is typeset using:

`\ecvlanguageheader{`⟨*symbol*⟩`}` Definition

where ⟨*symbol*⟩ is a footnote symbol used in the table footer. Each row of the language table is then typeset using:

`\ecvlanguage[`⟨*vspace*⟩`]{`⟨*language*⟩`}{`⟨*l1*⟩`}{`⟨*l2*⟩`}{`⟨*l3*⟩`}{`⟨*l4*⟩`}{`⟨*l5*⟩`}` Definition

except for the last, which is typeset using:

\ecvlastlanguage[⟨vspace⟩]{⟨language⟩}{⟨l1⟩}{⟨l2⟩}{⟨l3⟩}{⟨l4⟩}{⟨l5⟩} Definition

where ⟨*vspace*⟩ again indicates any vertical space that should be inserted after the row, and ⟨*language*⟩ is the name of the language. The other five arguments ⟨*l1*⟩, ..., ⟨*l5*⟩ should be brief descriptions relating to:

⟨*l1*⟩ understanding (listening);

⟨*l2*⟩ understanding (reading);

⟨*l3*⟩ speaking (spoken interaction);

⟨*l4*⟩ speaking (spoken production);

⟨*l5*⟩ writing.

Each of these arguments should be in the form:

\ecvCEF{⟨*level*⟩}{⟨*descr*⟩} Definition

where ⟨*level*⟩ is the self-assessment level code and ⟨*descr*⟩ is a brief description. There are some convenient shortcuts described in Table 5.1.

After the last language row, the table footer is typeset using:

\ecvlanguagefooter[⟨*vspace*⟩]{⟨*symbol*⟩} Definition

where ⟨*symbol*⟩ should be the same as used in \ecvlanguageheader.

Table 5.1 Convenient Shortcut Commands for \ecvCEF

Shortcut	Expansion
\ecvAOne	\ecvCEF{A1}{basic user}
\ecvATwo	\ecvCEF{A2}{basic user}
\ecvBOne	\ecvCEF{B1}{independent user}
\ecvBTwo	\ecvCEF{B2}{independent user}
\ecvCOne	\ecvCEF{C1}{proficient user}
\ecvCTwo	\ecvCEF{C2}{proficient user}

Example

Input

```
\ecvmothertongue[10pt]{English}
\ecvlanguageheader{(*)}
\ecvlanguage{French}{\ecvCOne}{\ecvCTwo}{\ecvBTwo}{\ecvCOne}
{\ecvCTwo}
\ecvlastlanguage{German}{\ecvATwo}{\ecvATwo}{\ecvATwo}{\ecvATwo}
{\ecvATwo}
\ecvlanguagefooter{(*)}
```

Input

⚠ Unless you use either narrow Arial or condensed Helvetica, the language table may appear cramped. For this example to work, I used the following settings to adjust the left column width and the gap on either side of the vertical rule:

```
\ecvLeftColumnWidth{2cm}
\ecvColSep{4pt}
```

Input

and set the default font size to 10pt:

```
\documentclass[helvetica,narrow,a4paper,10pt]{europecv}
```

Input

This produces the output shown in Figure 5.4 unless you use the booktabs class option:

```
\documentclass[helvetica,narrow,a4paper,10pt,booktabs]{europecv}
```

Input

in which case it produces the output shown in Figure 5.5.

Output

Mother tongue(s)	English				
Self-assessment European level(*)	**Understanding**		**Speaking**		**Writing**
	Listening	Reading	Spoken interaction	Spoken production	
French	C1 Proficient user	C2 Proficient user	B2 Independent user	C1 Proficient user	C2 Proficient user
German	A2 Basic user	A2 Basic user	A2 Basic user	A2 Basic user	A2 Basic user

(*) *Common European Framework of Reference (CEF) level*

Figure 5.4 Example Language Table

Output

Mother tongue(s) *Self-assessment European level*(*)	**English**				
	Understanding		**Speaking**		**Writing**
	Listening	Reading	Spoken interaction	Spoken production	
French	C1 Proficient user	C2 Proficient user	B2 Independent user	C1 Proficient user	C2 Proficient user
German	A2 Basic user	A2 Basic user	A2 Basic user	A2 Basic user	A2 Basic user

(*) *Common European Framework of Reference (CEF) level*

Figure 5.5 Example Language Table (`booktabs` option)

6. Official Documents

This chapter covers topics related to official documents, including records (such as minutes or agendas of meetings), memos and press releases. The chapter also covers typesetting matters relating to confidential documents and legal documents, such as watermarks and redaction.

6.1 Memos

§3.3 introduced the newlfm class [106], which can be used for writing correspondence. This class can also be used for writing memos and press releases. (See the next section for press releases.) If you read the earlier section on the newlfm class, you may remember that you can set the style of the document through the class options. For example, the `stdletter` option sets the standard letter style. There are two styles relating to memos: `stdmemo` (standard memo) and `fullmemo` (full memo).

As before, you still use the newlfm environment. However, some of the commands described in §3.3 are ignored for the memo styles, such as

those that set the greeting, closing and signature. Again, options can be set using either the class option list or the command:

\newlfmP{⟨*options list*⟩} Definition

Memo options are listed in Table 6.1.

Don't be confused by the "Re:" line. The \regarding command described in §3.3 is for letter subject lines. (Prefixed with "Regarding:") In a memo, the text in the line prefixed with "Re:" is set using:

\re{⟨*text*⟩} Definition

⚠ Note that in the current version of newlfm (dated 2009-04-10) there are bugs in the memonofrom, memonoto and memonore options. The following lines are workarounds to implement these options:

↑ Input

```
\setboolean{@memo@e}{false}% memonofrom
\setboolean{@memo@g}{false}% memonoto
\setboolean{@memo@f}{false}% memonore
```

↓ Input

Table 6.1 Memo Options

Option	Description
memonofrom	Omit sender's block.
memoemailfrom	Include sender's email.
memoaddrfrom	Include recipient's address.
memophonefrom	Include sender's telephone number.
memofaxfrom	Include recipient's fax number.
memopagerfrom	Include recipient's pager number.
memonoto	Omit recipient's block.
memoemailto	Include recipient's email.
memoaddrto	Include recipient's address.
memophoneto	Include recipient's telephone number.
memofaxto	Include recipient's fax number.
memopagerto	Include recipient's pager number.
memodate	Set the date on the memo.
fullmemo	Use all optional items.
memonore	Omit the "Re:" line.

Example 30. Sample Memo (newlfm class)
Commands such as \nameto and \dateset were described earlier in §3.3. This example makes use of some of those commands, but sets the style to stdmemo to create a memo instead of a letter.

↑ Input

```
\documentclass[12pt]{newlfm}

\usepackage[british]{babel}

\newlfmP{stdmemo,memoemailfrom,memophonefrom,memoaddrfrom,memodate}

\dateset{11-03-2014}
\nameto{Mabel Canary}

\namefrom{Mr Big Head}
\emailfrom{big.head@somewhere.ac.uk}
\phonefrom{0123456789}
\addrfrom{Secret Lab of Experimental Stuff}

\re{Cricket Match}
```

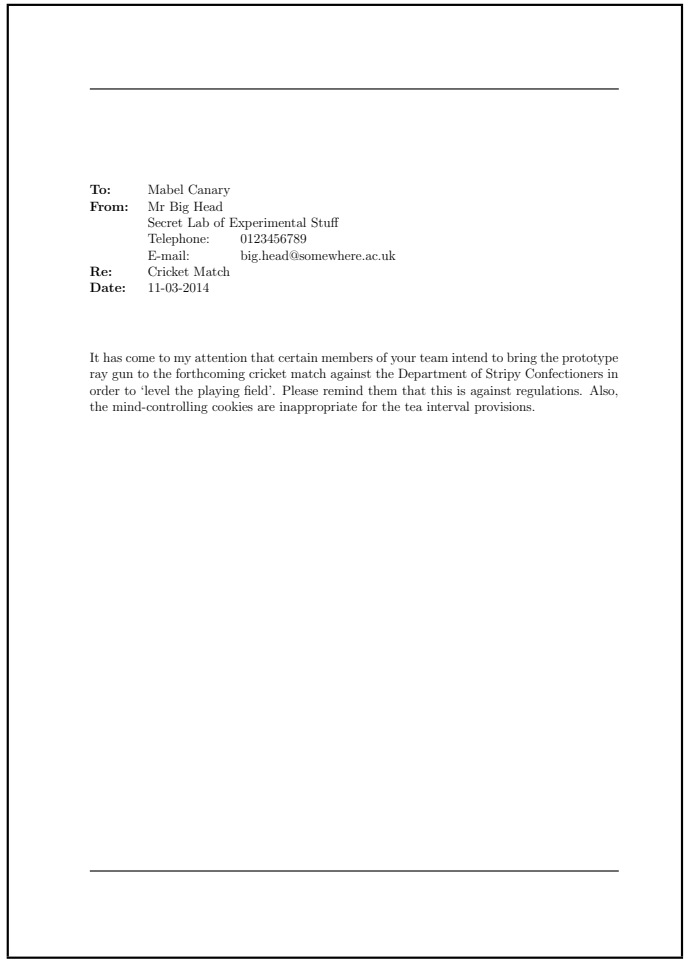
```
\begin{newlfm}
It has come to my attention that certain members of your team
intend to bring the prototype ray gun to the forthcoming cricket
match against the Department of Stripy Confectioners in order to
`level the playing field'. Please remind them that this is against
regulations.  Also, the mind-controlling cookies are inappropriate
for the tea interval provisions.
\end{newlfm}
\end{document}
```

↓ Input

(You can download or view this example.) The resulting document is shown in Figure 6.1.

6.2 Press Releases

The newlfm class can produce a press release using the `pressrelease` style. There are only two press release options: `dspace` (double-spaced, default) and `sspace` (single-spaced). This style has wide margins, a plain format and ends with three hashes (# # #). This is fairly standard for press releases, so don't try jazzing it up. As with the memo styles, some of the commands



To: Mabel Canary
From: Mr Big Head
Secret Lab of Experimental Stuff
Telephone: 0123456789
E-mail: big.head@somewhere.ac.uk
Re: Cricket Match
Date: 11-03-2014

It has come to my attention that certain members of your team intend to bring the prototype ray gun to the forthcoming cricket match against the Department of Stripy Confectioners in order to 'level the playing field'. Please remind them that this is against regulations. Also, the mind-controlling cookies are inappropriate for the tea interval provisions.

Figure 6.1 A Sample Memo (newlfm class)

described in §3.3 are ignored for the press release style, but some of the commands used by the memo styles are also ignored.

The subject of the press release is set using:

`\headline{`⟨*text*⟩`}` Definition

By default, the phrase "For Immediate Release" is placed at the start. This wording can be changed using

`\release{`⟨*text*⟩`}` Definition

Example 31. Press Release (newlfm class)

Note that the `\emailfrom` command is ignored by the `pressrelease` style, so it hasn't been included. The `\headline` command has been used to set the title and there is also a `\section` command available that produces an unnumbered section. The `\url` command is provided by the url package [5].

↑ Input

```
\documentclass{newlfm}

\usepackage{url}
\usepackage[british]{babel}
```

```
\newlfmP{pressrelease}

\namefrom{Mr Big Head}
\addrfrom{University of Somewhere\\Some City\\AB3 4YZ}
\phonefrom{0123456789}

\headline{Secret Experimental Lab Open Day}

\begin{document}

\begin{newlfm}
The University of Somewhere is throwing open the doors
of its renowned Secret Experimental Lab to the public
for the first time ever on 1st April 2014.

Mr Big Head, managing director of the lab, will be giving
conducted tours between 11:59 and 12:00. More details,
including car parking arrangements, are listed at
\url{www.somewhere.ac.uk/secretopenday}

\section{About the University of Somewhere}
```

```
The university is a non-existent academic institution
renowned for its cutting-edge imaginary research.

Website: \url{www.somewhere.ac.uk}
\end{newlfm}

\end{document}
```

↓ Input

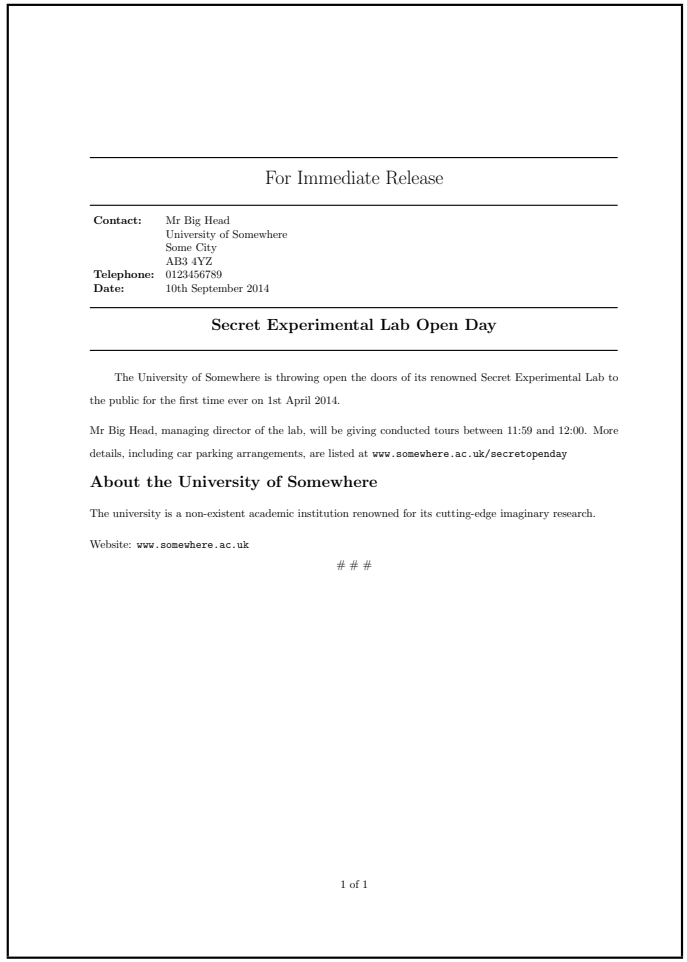
(You can download or view this example.) The resulting document is shown in Figure 6.2.

The newlfm class is quite difficult to adapt and it seems strange not to have an email option for a press release in this Internet age. Since this didn't suit my purposes and I couldn't find another class for press releases, I decided to write my own class called pressrelease [96], which I uploaded to CTAN. At the time of writing, the current version is 1.0 (dated 2014-09-10).

The pressrelease class has the following class options:

`10pt` Set the normal font size to 10pt.

`11pt` Set the normal font size to 11pt.



For Immediate Release

Contact: Mr Big Head
University of Somewhere
Some City
AB3 4YZ
Telephone: 0123456789
Date: 10th September 2014

Secret Experimental Lab Open Day

The University of Somewhere is throwing open the doors of its renowned Secret Experimental Lab to the public for the first time ever on 1st April 2014.

Mr Big Head, managing director of the lab, will be giving conducted tours between 11:59 and 12:00. More details, including car parking arrangements, are listed at `www.somewhere.ac.uk/secretopenday`

About the University of Somewhere

The university is a non-existent academic institution renowned for its cutting-edge imaginary research.

Website: `www.somewhere.ac.uk`

# # #

1 of 1

Figure 6.2 A Sample Press Release (newlfm class)

`12pt` Set the normal font size to 12pt.

`a4paper` Set the paper size to A4.

`letterpaper` Set the paper size to US Letter.

`symbols` Use symbols instead of textual tags.

As with newlfm there are commands to specify the contact information. The main headline text is specified using:

`\PRheadline{`⟨*text*⟩`}` Definition

and the sub-heading (if required) is specified using:

`\PRsubheadline{`⟨*text*⟩`}` Definition

The date of the press release is specified using the standard `\date` command.

The company logo (if required) is specified using:

`\PRlogo{`⟨*logo*⟩`}` Definition

where ⟨*logo*⟩ is the command or commands required to input or draw the logo. For example, if the logo is in the image file `company-logo.png`:

```
\PRlogo{\includegraphics{company-logo}}
```
Input

(Remember that you need to load the graphicx package if you want to use the \includegraphics command.)

The company's name is specified using:

\PRcompany{⟨*name*⟩}
Definition

The department's name is specified using:

\PRdepartment{⟨*name*⟩}
Definition

The company's location is specified using:

\PRlocation{⟨*location*⟩}
Definition

The company's address is specified using:

\PRaddress{⟨*address*⟩}
Definition

(You may use \\ to separate the lines in the address.)

The company's email address is specified using:

\PRemail{⟨*address*⟩}
Definition

The company's telephone number is specified using:

\PRphone{⟨number⟩} Definition

The company's contact mobile number is specified using:

\PRmobile{⟨number⟩} Definition

The company's fax number is specified using:

\PRfax{⟨number⟩} Definition

The company's website is specified using:

\PRurl{⟨website⟩} Definition

The company's opening hours are specified using:

\PRhours{⟨times⟩} Definition

For example:

```
\PRhours{Mon--Fri 9:00--17:00}
```
Input

As with titling commands, such as \title, the above commands just store the information and may be placed in the preamble. The actual contents of the press release should go in the pressrelease environment. Within this environment, you can use the about environment to add information about the company.

```
\begin{pressrelease}
⟨press release text⟩

\begin{about}                                          Definition
⟨information about the company⟩
\end{about}
\end{pressrelease}
```

Exercise 17. Press Release (pressrelease class)
Adapt Example 31 so that it uses the pressrelease class and add an email address to the contact details. You can download or view a solution.

6.3 Minutes

There are only three options for writing minutes listed on the meeting-admin topic page: the meetingmins class (which can also typeset agendas but has no multilingual support), the minutes package and the protocol class. The last one, protocol, only has German documentation, so it's not discussed here.

6.3.1 The meetingmins Class

The meetingmins class [6] uses the standard sectioning commands, such as `\section` and `\subsection`, but provides additional commands and environments useful in the creation of minutes or agendas. Note that this class doesn't provide multilingual support.

The default behaviour of this class is to assume you are creating minutes from a meeting. The document will be headed "Minutes for ⟨*date*⟩" and some of the commands described below will be ignored. You can change this format via one of the following class options:

`agenda` This indicates the document is an agenda. The heading will be "Agenda for ⟨*date*⟩".

`chair` This indicates the document is the chair's agenda. The heading will be "Chair's Agenda for ⟨*date*⟩" and there will be a list of members names with checkboxes for the chair to track attendance.

`notes` This sets the heading to "Notes for ⟨*date*⟩".

The date of the meeting is set using:

\setdate{⟨*date*⟩} Definition

The name of the committee is set using:

\setcommittee{⟨*name*⟩} Definition

The list of members is set using:

\setmembers{⟨*list*⟩} Definition

where ⟨*list*⟩ is a list of members. (The member list is only used by the `chair` class option.) The list of people present at the meeting is set using:

\setpresent{⟨*list*⟩} Definition

where again ⟨*list*⟩ is a list of names. (This list is ignored by the `chair` and agenda options.) Within ⟨*list*⟩, for both \setmembers and \setpresent, the chair's name should be set with:

\chair{⟨*name*⟩} Definition

Example:

↑ Input

```
\setdate{12th March 2014}
```

```
\setcommittee{Secret Lab of Experimental Stuff}
\setmembers{
  \chair{Mr Big Head},
  Prof Important Person,
  Dr Bor Ing
}
\setpresent{
  \chair{Mr Big Head},
  Dr Bor Ing
}
```

↓ Input

This sets information (analogous to \author and \title) but doesn't actually display any information, so these commands may be used in the preamble. As with \author and \title, you need to use

\maketitle

Definition

to make the title information appear.

Within the document environment, if you need numbered lists you can use the items environment

```
\begin{items}
\item ⟨item text⟩
...
\end{items}
```

Definition

within sections and the subitems environment

```
\begin{subitems}
\item ⟨item text⟩
...
\end{subitems}
```

Definition

within subsections and sub-subsections. As is usual for list environments, use \item at the start of each item. If there are items that should only appear in the chair's agenda or in the notes, you can instead use the hiddenitems environment

```
\begin{hiddenitems}
\item ⟨item text⟩
...
\end{hiddenitems}
```

Definition

within sections and the hiddensubitems environment

```
\begin{hiddensubitems}
\item ⟨item text⟩
...
\end{hiddensubitems}
```

Definition

within subsections and sub-subsections. There is also a similar hiddentext environment

```
\begin{hiddentext}
⟨text⟩
\end{hiddentext}
```

Definition

for non-list text. The contents ⟨*text*⟩ is only displayed if the `chair` or `notes` class option has been used.

The date and time of the next meeting can be displayed using:

`\nextmeeting{`⟨*date and time*⟩`}` Definition

This is ignored for the `agenda` option. Finally

`\priormins` Definition

is a convenient shortcut for "The minutes of the previous meeting were approved".

Example:

↑ Input

```
\section{Announcements}
\begin{hiddenitems}
 \item A~fire alarm drill is expected during the meeting.
\end{hiddenitems}

\section{Old Business}
\begin{items}
\item \priormins
\end{items}

\section{New Business}
\begin{items}
\item Discuss schedules for secret research.

\item Discuss schedules for new experimental stuff.
\end{items}

\nextmeeting{15th April 2014 at 15:00}
```

↓ Input

If the chair or notes class options are used the fire alarm announcement will be displayed, otherwise it will be skipped.

Exercise 18. Minutes and Agendas (meetingmins class)
Create a document using the meetingmins class that uses all the commands described in this section. Try using the different class options to see how they affect the document. (Remember that you can't mix the class options.)

You can download or view a solution.

6.3.2 The minutes Package

Unlike meetingmins, described above, minutes [52] is a package, so you can choose your document class. The package comes with alternative German commands that you can use instead of the English command names. For example, instead of using \subtitle you can use \untertitel. For brevity, this book only describes the English commands. See the minutes documentation for a list of the German equivalents and for the additional Dutch support. The default language is German, so if you are writing the minutes in English, you'll need to load babel [7] with the english option.

The meeting minutes are contained in the body of the Minutes environment

\begin{Minutes}{⟨title⟩} Definition

where ⟨title⟩ is the title of the minutes. You can have more than one Minutes environment within your document, for example, if you want a compilation of all the minutes for a particular group or committee.

Within the Minutes environment, you can set various information using the commands described below.

\subtitle{⟨title⟩} Definition

This sets the subtitle, if required.

\moderation{⟨name⟩} Definition

This sets the name of the meeting moderator (for example, the chair).

\minutetaker{⟨name⟩} Definition

This sets the name of the minute taker.

\participant{⟨names⟩} Definition

This sets the names of the people present at the meeting.

\guest{⟨*names*⟩} Definition

This sets the names of any guests present at the meeting.

\minutesdate{⟨*date*⟩} Definition

This sets the date of the meeting.

\starttime{⟨*time*⟩} Definition

This sets the starting time of the meeting.

\endtime{⟨*time*⟩} Definition

This sets the time the meeting ended.

\location{⟨*place*⟩} Definition

This sets the location of the meeting.

\cc{⟨*names*⟩} Definition

This sets the distribution list. The argument ⟨*names*⟩ is a list of names of people who should receive a copy of the minutes. To specify absentees, you can either use

\missingExcused{⟨*excused names*⟩} Definition

and

\missingNoExcuse{⟨*no-excuse names*⟩} Definition

or

\missing[⟨*excused names*⟩]{⟨*no-excuse names*⟩} Definition

where ⟨*excused names*⟩ is a list of names of missing people who provided an excuse and ⟨*no-excuse names*⟩ is a list of missing people who didn't provide an excuse.

The above commands all behave in an analogous way to \title and \author. Once they have been specified, you then need to use:

\maketitle Definition

Example 32. Sample Minutes (minutes package)
This example just sets up the title information for the minutes.

↑ Input

```
\documentclass{article}
```

```
\usepackage[english]{babel}
\usepackage{minutes}

\begin{document}

\begin{Minutes}{Secret Lab of Experimental Stuff}
 \subtitle{Annual General Meeting}
 \moderation{Mr Big Head}
 \minutetaker{Dr Bor Ing}
 \participant{Polly Parrot, Mabel Canary}
 \missing[Z\"oe Zebra, Jos\'e Arara]{Dickie Duck, Fred Canary}
 \guest{Prof Important Person}
 \minutesdate{12th March 2014}
 \starttime{15:00}
 \endtime{17:00}
 \location{University of Somewhere}
 \cc{Vice Chancellor}

 \maketitle
\end{Minutes}

\end{document}
```

Input

The result is shown in Figure 6.3. Since this document only contains the minutes from a single meeting, I haven't bothered to include an overall document title or table of contents. This causes warnings from the minitoc package [22] (which the minutes package loads). If you want to have a collection of minutes, you can add the title and contents at the start of the document:

Input

```
\begin{document}
\title{Minutes from the Secret Lab Meetings}
\author{Dr Bor Ing}
\maketitle
\tableofcontents
% add the Minutes environments here
```

Input

You can download or view this example.

The minutes can be subdivided into topics using:

Output

Secret Lab of Experimental Stuff

Annual General Meeting

Moderation Mr Big Head

Minutes taker Dr Bor Ing

Those present Polly Parrot, Mabel Canary

Absent (excused) Zöe Zebra, José Arara

Absent (not excused) Dickie Duck, Fred Canary

Guest Prof Important Person

Location of the meeting University of Somewhere

Date 12th March 2014 15:00–17:00

Distribution Vice Chancellor

Figure 6.3 Sample Minutes (Title Information Only)

\topic[⟨*toc text*⟩]{⟨*text*⟩} Definition

and subtopics using:

\subtopic[⟨*toc text*⟩]{⟨*text*⟩} Definition

where ⟨*text*⟩ is the topic or subtopic and ⟨*toc text*⟩ is alternative text for the table of contents. These commands are analogous to sectioning commands such as \section. The ⟨*toc text*⟩ (or ⟨*text*⟩ if the optional argument is omitted) appears in an overview section in the minutes and also appears in the overall document table of contents (if \tableofcontents has been used, as described above).

Tasks can be specified using the \task command which has a starred and unstarred version. The unstarred version has the syntax:

\task[⟨*footnote text*⟩]{⟨*name*⟩}[⟨*when*⟩]{⟨*text*⟩} Definition

The starred version has the syntax:

\task*[⟨*when*⟩]{⟨*text*⟩} Definition

Example:

↑ Input

```
\topic{Tasks}
\subtopic{New Experimental Stuff}
\task[done]{Mabel Canary}[tomorrow]{Proposal for a time machine}
\task[pending]{Polly Parrot}{Apply for a ray gun grant}

\subtopic{Kitchen}
\task*{Order a new coffee machine}
\task*[today]{Remove the mind-controlling cookies}
```

Input

As with the meetingmins class, the minutes package allows you to hide text. This is done either via the command:

`\secret{`⟨*secret text*⟩`}` Definition

or using the Secret environment

```
\begin{Secret}
⟨secret text⟩
\end{Secret}
```

Definition

The ⟨*secret text*⟩ will only be displayed if you use the `Secret` package option.

```
\usepackage[Secret]{minutes}
```

Input

Example:

Input

```
\begin{Secret}
 \task*{There will be a surprise lab inspection on Tuesday.}
\end{Secret}
```

Input

If the meeting discussed an opinion, this can be recorded using:

`\opinion{`⟨*main*⟩`}{`⟨*differing*⟩`}` Definition

where ⟨*main*⟩ is the main opinion held and ⟨*differing*⟩ is the differing opinion. The discussion can then be formatted using the Opinions environment:

```
\begin{Opinions}
\item[⟨name⟩] ⟨opinion⟩
...
\end{Opinions}
```

Definition

Example:

Input

```
\opinion
  {Keep the coffee break at 11.00am}% main
  {Move the coffee break to 10:30am}% differing

\begin{Opinions}
  \item[Mabel Canary] We should continue to have
   coffee at 11:00am.
  \item[Polly Parrot] We should move the coffee
   break to an earlier time.
\end{Opinions}
```

Input

Arguments can be formatted using the Argumentation environment.

```
\begin{Argumentation}
⟨items⟩
\end{Argumentation}
```

Definition

Within this environment, you can use the standard \item command for a comment or one of the following commands:

`\pro` ⟨*reason for*⟩ Definition

which itemizes a reason in favour of the argument,

`\Pro` ⟨*important reason for*⟩ Definition

which itemizes an important reason in favour of the argument,

`\contra` ⟨*reason against*⟩ Definition

which itemizes a reason against the argument,

`\Contra` ⟨*important reason against*⟩ Definition

which itemizes an important reason against the argument, and

`\result` ⟨*argument result*⟩ Definition

which itemizes the result of the argument.

Example:

↑ Input

```
\begin{Argumentation}
  \pro We've always had coffee at 11:00am. There's no need to
change it.
```

```
  \contra 11:00am is too long a wait for the caffeine addicts.
  \Pro Coffee at 10:30am would interfere with our clandestine
  experiments scheduled at that time.
  \item Prof Important Person said it would be better to have
tea instead of coffee.
  \result The coffee break will continue to be at 11:00am.
\end{Argumentation}
```

↓ Input

A single vote can be formatted using

\vote{⟨*description*⟩}{⟨*yes*⟩}{⟨*no*⟩}{⟨*abstain*⟩}[⟨*decision*⟩] Definition

where ⟨*description*⟩ is a brief description of the vote, ⟨*yes*⟩ is the number of "Yes" votes, ⟨*no*⟩ is the number of "No" votes and ⟨*abstain*⟩ is the number of abstainers. Optionally, a decision can be added. For example:

```
\vote{Maintain coffee at 11am}{2}{1}{1}
```

Input

indicates that there were two votes in favour of maintaining coffee at 11am, one vote against and one abstainer.

Multiple votes can be listed in the Vote environment:

```
\begin{Vote}
\vote{⟨description⟩}{⟨yes⟩}{⟨no⟩}{⟨abstain⟩}[⟨decision⟩]
...
\end{Vote}
```

Definition

Example:

↑ Input

```
\begin{Vote}
 \vote{Maintain coffee at 11am?}{2}{1}{1}
 \vote{Move clandestine experiments to after lunch?}{1}{3}{0}
\end{Vote}
```

↓ Input

Decisions are first declared using:

```
\decisiontheme{⟨theme⟩}{⟨title⟩}
```

Definition

This doesn't display anything in the document at this point, but it will be added to the list of decisions which can be displayed using:

```
\listofdecisions
```

Definition

This is like other `\listof...` commands, such as `\listoftables`. Each decision is then specified using:

\decision{⟨theme⟩}{⟨short description⟩}[⟨long description⟩] Definition

There is also a starred version which doesn't add the decision to the list of decisions:

\decision*{⟨short description⟩}[⟨long description⟩] Definition

Example:

↑ Input

```
\decisiontheme{Ray Guns}{Should we reverse the polarity of
ray guns?}
 \decision{Ray Guns}{The ray gun polarity doesn't need
modifying.}
 \decision*{We don't need to reverse the polarity.}
  [Reversing the polarity is generally considered to be a daft
idea.]
```

↓ Input

If necessary, the minutes can be signed at the end using

\signature{⟨*name*⟩} Definition

This should be placed before the end of the Minutes environment. Any additional information that doesn't belong to the minutes may be included in the Postscript environment

\begin{Postscript}
⟨*additional information*⟩ Definition
\end{Postscript}

or in the argument of

\postscript{⟨*additional information*⟩} Definition

For other commands not listed here, including how to alter the style, see the minutes documentation.

Exercise 19. Minutes (minutes package)
Extend the document in Example 32 to include topics, tasks, opinions, arguments, votes and decisions.

You can download or view a solution.

6.4 Confidentiality

This section covers some topics related to confidentiality. There are a number of options listed on the security topic page. This section will just be looking at redaction (§6.4.1) and watermarks (§6.4.2). See also §2.3 for advice on document security.

6.4.1 Redaction: The censor Package

The censor package [86] provides a way to black out redacted words or phrases, paragraphs, or boxes (such as included graphics or a tabular environment). Note that the code used for redaction affects line-breaking and paragraph justification, so don't expect the redacted version of the document to look as well as an unedited version.

Redacted words or phrases are tagged using:

`\censor{`⟨*text*⟩`}` Definition

Example:

↑ Input

```
Following the research group's unsuccessful attempt
```

```
to create \censor{mind-controlling} \censor{cookies},
they will now be working on a new \censor{ray gun}.
```

Input

produces:

Output

Following the research group's unsuccessful attempt to create [redacted], they will now be working on a new [redacted].

Output

Note that the argument of \censor is placed in a box so its contents can't be broken across a line, which is why I split up "mind-controlling" and "cookies". In fact, it may even be necessary to do:

```
\censor{mind}-\censor{controlling} \censor{cookies},
```

Input

A box can be redacted using:

\censorbox[⟨*declarations*⟩]{⟨*box*⟩}

Definition

where ⟨*box*⟩ is the box being redacted and ⟨*declarations*⟩ are any commands (such as font-changing declarations) that should be applied before the start of the box.

Example:

↑ Input

```
\begin{table}
 \caption{A Redacted Table}
 \label{tab:redacted}
 \centering
 \censorbox{%
   \begin{tabular}{lc}
   \bfseries Project & \bfseries Success Rate \\
   Mind-controlling Cookies & 2\%\\
   Telepathic Cakes & 1\%\\
   Exploding Chocolates & 25\%
   \end{tabular}%
 }
\end{table}
```

↓ Input

This produces Table 6.2.

A paragraph can be redacted using:

\blackout{⟨*text*⟩} Definition

Output

Table 6.2 A Redacted Table

Example:

Input

```
\blackout{Following the research group's unsuccessful attempt to
create mind-controlling cookies, they will now be working on
a new ray gun.}
```

Input

produces:

Output

Output

Note that there are restrictions on the use of \blackout: ⟨*text*⟩ can't end with "glue", such as a space or EOL character; periods aren't redacted; it can't be used across changes in scope, such as environment boundaries or across cells within a tabular-like environment. This command also shows spaces between words (unless the spaces are hidden within a token), but there is an alternative command that hides these spaces:

\xblackout{⟨*text*⟩} Definition

Example:

```
\xblackout{Following the research group's unsuccessful attempt to
create mind-controlling cookies, they will now be working on
a new ray gun.}
```

Input

Input

produces:

Output

Output

This command also suffers from drawbacks. For example, the redaction can cause lines to protrude into the left and right margins. See the censor documentation for further details.

The unredacted version of the document can be created by placing

\StopCensoring Definition

at the beginning of the document. You can also restart redaction with:

\RestartCensoring Definition

It's possible that you may need to work on your document at an insecure location. In which case, you won't want your sensitive information in your `.tex` file. The censor package provides:

\censor*{⟨*size*⟩} Definition

where ⟨*size*⟩ is the approximate width (in ex) of the redacted text, and:

\censorbox*[⟨*declarations*⟩]{⟨*width*⟩}{⟨*height*⟩}{⟨*depth*⟩} Definition

where ⟨*width*⟩ is the approximate width (in ex) of the box, ⟨*height*⟩ is the approximate height (in multiples of \baselineskip) and ⟨*depth*⟩ is the approximate depth (in multiples of \baselineskip). The optional argument should be used to specify any commands (such as font changing declara-

tions) that may effect the size of an ex or the value of \baselineskip. (For example, \small.) There's no starred version of \blackout or \xblackout.

Example:

Input

```
\newcommand*{\OldProject}{\censor*{4}-\censor*{11} \censor*{7}}
\newcommand*{\NewProject}{\censor*{3} \censor*{3}}
\newcommand*{\SuccessRates}{\censorbox*{40}{4}{0}}

Following the research group's unsuccessful attempt
to create \OldProject, they will now be working
on a new \NewProject. The success rate of previous projects is
shown in Table~\ref{tab:success}.

\begin{table}
 \caption{Project Success Rates}
 \label{tab:success}
 \centering
 \SuccessRates
\end{table}
```

Input

If the censoring is on, the redacted text will again be replaced by filled rectangles:

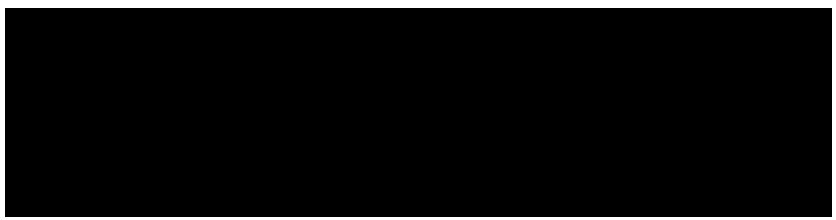
Output

Following the research group's unsuccessful attempt to create [redacted], they will now be working on a new [redacted]. The success rate of previous projects is shown in Table 6.3.

Output

Output

Table 6.3 Project Success Rates



However if censoring is switched off using `\StopCensoring`, the redacted text will now be replaced by an underlined blank space (in the case of `\censor*`) or an unfilled rectangle (in the case of `\censorbox*`):

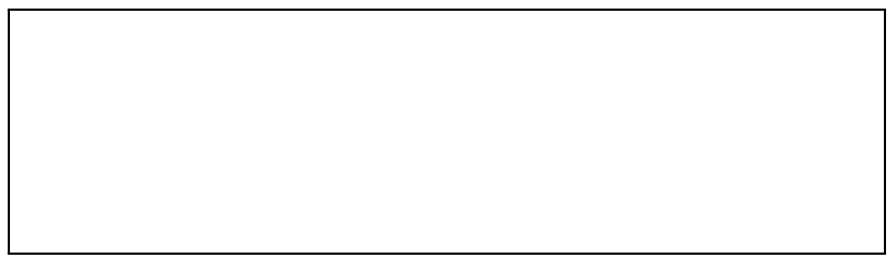
↑ Output

Following the research group's unsuccessful attempt to create ____-_____ _____, they will now be working on a new ___ ___. The success rate of previous projects is shown in Table 6.4.

↓ Output

Output

Table 6.4 Project Success Rates



In your secure environment, you can change the definitions of the macros for the redacted material:

↑ Input

```
\newcommand*{\OldProject}{%
 \censor{mind}-\censor{controlling} \censor{cookies}}
```

```
\newcommand*{\NewProject}{\censor{ray} \censor{gun}}
\newcommand*{\SuccessRates}{%
 \censorbox{%
   \begin{tabular}{lc}
   \bfseries Project & \bfseries Success Rate \\
   Mind-controlling Cookies & 2\%\\
   Telepathic Cakes & 1\%\\
   Exploding Chocolates & 25\%
   \end{tabular}%
 }%
}
```

↓ Input

There may be minor discrepancies in the formatting caused by approximate measurements.

It's possible you may not want to keep editing the definitions of commands like the example \OldProject, \NewProject and \SuccessRates, if you keep transferring your document between a secure and an insecure location. Instead, it's better to keep the real definitions of these commands (using the unstarred versions of \censor and \censorbox) in a separate file, and only input the file if it exists. Then, when you transfer your document to an insecure location, make sure you don't also transfer this sensitive file.

Example 33. Redaction

Suppose you have a file called `definitions.tex` that contains:

↑ Input

```
\newcommand*{\OldProject}{%
 \censor{mind}-\censor{controlling} \censor{cookies}}
\newcommand*{\NewProject}{\censor{ray} \censor{gun}}
\newcommand*{\SuccessRates}{%
 \censorbox{%
   \begin{tabular}{lc}
   \bfseries Project & \bfseries Success Rate \\
   Mind-controlling Cookies & 2\%\\
   Telepathic Cakes & 1\%\\
   Exploding Chocolates & 25\%
   \end{tabular}%
 }
}
```

↓ Input

This is your secret file that shouldn't leave your secure location. In your main document you can test for a file's existence, and only input it if it exists, using:

\InputIfFileExists{⟨*file*⟩}{⟨*true part*⟩}{⟨*false part*⟩} Definition

This tests if the file named ⟨*file*⟩ exists (the `.tex` extension may be omitted). If the file exists, this command does ⟨*true part*⟩ and then loads ⟨*file*⟩. If the file doesn't exist, this command just does ⟨*false part*⟩. So the main document can include this command to determine whether to use the starred or unstarred versions:

↑ Input

```
\documentclass[captions=tableheading]{scrartcl}

\usepackage{censor}

\InputIfFileExists{definitions}{}%
{%
  \newcommand*{\OldProject}{\censor*{4}-\censor*{11} \censor*{7}}
  \newcommand*{\NewProject}{\censor*{3} \censor*{3}}
  \newcommand*{\SuccessRates}{\censorbox*{40}{4}{0}}
}

\begin{document}
```

```
Following the research group's unsuccessful attempt
to create \OldProject, they will now be working
on a new \NewProject. The success rate of previous projects is
shown in Table~\ref{tab:success}.

\begin{table}
 \caption{Project Success Rates}
 \label{tab:success}
 \centering
 \SuccessRates
\end{table}

\end{document}
```

↓ Input

You can download or view this example.

6.4.2 Watermarks

[FAQ: 'Watermarks' on every page]

There are a number of packages that enable you to place some text (such as "CONFIDENTIAL" or "DRAFT") across the background of every page. In addition to these packages, the pdftk application described in §2.3, can

also be used for this purpose. CTAN has several topics that cover this area: watermark, background and decoration. This section will only discuss two packages: xwatermark (which extends the draftmark and watermark packages) and background. The background package is simpler to use. The xwatermark package is more flexible.

The background Package

The background package [55] provides a way of specifying a watermark using the tikz package (part of the pgf bundle [101]). By default, the package will use the word "Draft" as a watermark, but this can be changed. Options can either be set via the package option list or via:

`\backgroundsetup{`⟨*options*⟩`}` Definition

where ⟨*options*⟩ is a key=value list. The background package requires two LaTeX calls during the document build. Available options are:

`pages` This key specifies whether the watermark should appear on all or some of the pages. Available values are: `all` or `some`. If the value `some` is specified, you must use the command `\BgThisPage` on the pages where the watermark is required. If the value `all` is specified, you can use the command

\NoBgThisPage on the pages where you don't want a watermark. (Don't use this command in two-column mode.)

`firstpage` This is a boolean key. If `true`, the watermark only appears on the first page. (The default is `false`.)

`placement` This key specifies the placement of the watermark and may take one of the values: `center` (default), `top` or `bottom`.

`contents` This key specifies the material used for the watermark. The value may be just text (such as "Confidential") or an image included using \includegraphics. If the value of this key contains commands, the key may only be used within \backgroundsetup not as a package option.

`color` The watermark colour. (The background package automatically loads the xcolor package [40].)

`angle` This key specifies the angle of rotation.

`opacity` This key indicates the transparency. The value may be a number from 0 (full transparency) to 1 (no transparency). Note that PostScript doesn't support transparency, so if you use `latex`

and dvips, the watermark won't show any transparency in the PostScript file.

`scale` This key specifies the scale factor.

`position` This key can be used to adjust the position of the watermark. The value should be in TikZ coordinate form (see the pgf manual [101]).

`anchor` This key specifies the TikZ anchor for the watermark. (See the pgf manual [101].)

`hshift` This key specifies the horizontal shift.

`vshift` This key specifies the vertical shift.

Example 34. Watermarks (background package)
This example uses the lipsum package [32] to generate dummy text to pad out the document.

↑ Input

```
\documentclass[a4paper]{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usepackage{background}
\usepackage{lipsum}

\backgroundsetup{contents={CONFIDENTIAL}}

\begin{document}

\lipsum[1-55]

\end{document}
```

Input

The first page of this document is shown in Figure 6.4. You can download or view this example.

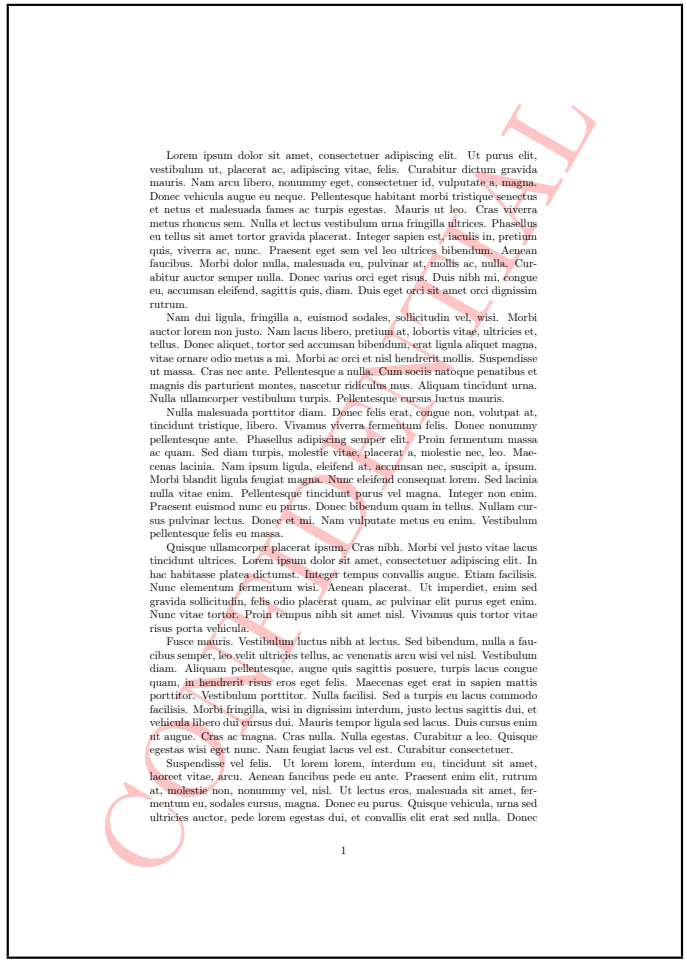


Figure 6.4 Sample Watermark (background package)

The xwatermark Package

The xwatermark package [59] provides a way of adding either text or graphics as watermarks in the background or foreground of selected pages. In most cases, more than one LaTeX run will be required on your document. You will also mostly need to load a colour package, such as xcolor [40]. The xwatermark package can also be used to tile marks across the page, but that option isn't covered here. See the xwatermark documentation for details.

There are a number of package options for xwatermark. Only a subset of them are listed here:

`printwatermark` This is a boolean key that determines whether or not the watermarks should be printed. The default value is `true`.

`picontoptext` This is a boolean key that determines whether the picture watermark should be placed on top of the text watermark where they occur on the same page. The default value is `true`.

`showpagecenter` This is a boolean key that determines whether the centre of the paper should be shown with a marker. The default value is `false`.

`disablegeometry` This is a boolean key. If `true`, this indicates that the page layout settings by the geometry package [109] should be disabled. The default value is `false`. (This option is less likely to be needed with newer versions of geometry.)

See the xwatermark package documentation for details of the other package options.

A new watermark is defined using:

`\newwatermark[`⟨*options*⟩`]{`⟨*mark*⟩`}` Definition

where ⟨*mark*⟩ is the watermark text. This may be empty if you want an image for the watermark. There is a starred variant of this command that puts the watermark in the foreground instead of the background. There is also a prime variant, but that's not covered here. See the xwatermark documentation [59] for further details.

The optional argument, ⟨*options*⟩, is a key=value list of options. For brevity, this section only covers a small subset of those options. See the xwatermark documentation for further details. The option list must contain the page or page set on which the watermark is to be displayed. The page specifier keys are as follows:

`allpages` This key doesn't have a value. It indicates that the watermark should be displayed on all pages.

`oddpages` This key doesn't have a value. It indicates that the watermark should be displayed on all odd pages.

`evenpages` This key doesn't have a value. It indicates that the watermark should be displayed on all even pages.

`firstpage` This key doesn't have a value. It indicates that the watermark should only be displayed on the first page.

`lastpage` This key doesn't have a value. It indicates that the watermark should only be displayed on the last page.

`page` The value of this key should be a page number to indicate that the page on which the watermark should be display. For example, `page=4` indicates that the watermark should be displayed on page 4.

`pages` The value of this key should be a range in the form ⟨*n*⟩-⟨*m*⟩ to indicate that the watermark should be displayed on pages ⟨*n*⟩ to ⟨*m*⟩, inclusive. For example, `pages=4-10` indicates that the watermark should be displayed on pages 4 to 10.

`pagex` The value of this key should be a comma-separated list of page numbers on which the watermark should be displayed. For

example, pagex={1,3,7} indicates that the watermark should be displayed on pages 1, 3 and 7.

The text for the watermark may also be specified in ⟨*options*⟩ using the textmark={⟨*mark*⟩} key. The watermark can be scaled, rotated or translated using the following keys:

scale The value of this key is the scaling factor.

angle The value of this key is the angle of rotation.

xpos The value of this key specifies the horizontal position of the watermark relative to the centre of the page.

ypos The value of this key specifies the vertical position of the watermark relative to the centre of the page.

The formatting of the watermark can be adjusted using the following keys:

textalign The value of this key indicates the horizontal alignment of the watermark and may be one of: center (default), left, right or justified.

fontfamily The value of this key is the name of the font family (as supplied to \fontfamily). For example, fontfamily=pbk indicates Bookman.

`fontseries` The value of this key is the name of the font series (as supplied to `\fontseries`). For example, `fontseries=b` indicates bold.

`fontsize` The value of this key is the font size. The default is 1 cm.

`textcolor` The value if this key is the colour of the watermark text.

If an image is required, the following keys specify the image file information:

`picfile` The value of this key is the image filename. This should be in the same form as the mandatory argument of `\includegraphics`, so remember to use forward slashes for the directory divider even if you are building your document on a Windows operating system.

`picfileext` The value of this key is the image file extension (such as `pdf` or `png`). Note that you have to use `picfileext` if you use `picfile`.

`picscale` The value of this key is the scaling factor to apply to the image.

`picangle` The value of this key is the angle of rotation to apply to the images.

There are other keys not covered here. See the xwatermark documentation [59] for further details.

Example 35. Watermarks (xwatermark package)
As with Example 34, this example uses the lipsum package [32] to generate dummy text to pad out the document.

↑ Input

```
\documentclass[a4paper]{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usepackage{xcolor}
\usepackage{xwatermark}
\usepackage{lipsum}

\newwatermark[%
  allpages,% show on all pages
  fontfamily=pbk,% use Bookman font family
  angle=55,% rotate by 55 degrees
  scale=2.75,% scale by 2.75
```

```
  xpos=-1cm,% shift by 1cm to the left
  ypos=1cm% shift up by 1cm
]{SAMPLE WATERMARK}

\begin{document}
\lipsum[1-55]

\end{document}
```

↓ Input

The first page of this document is shown in Figure 6.5. You can download or view this example.

6.5 Typesetting Legal Documents (Numbered Paragraphs)

The are a number of packages listed on the legal topic page for typesetting legal documents, however they are mostly for particular areas, such as German legal texts or US patent applications. Since they are too specific for a general guide, this section just looks at how to produce the paragraph

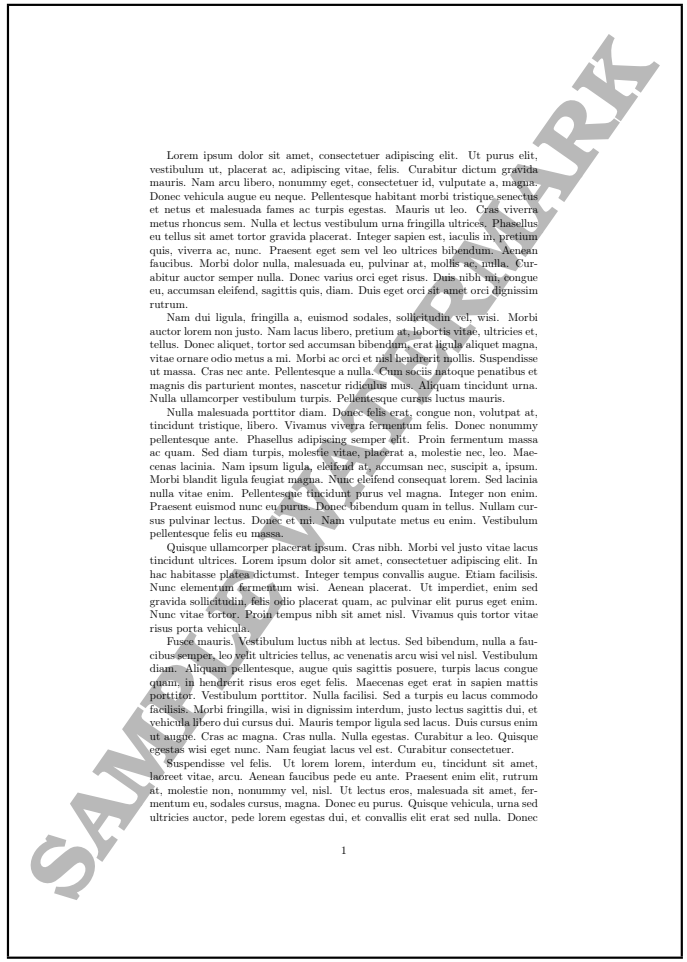


Figure 6.5 Sample Watermark (xwatermark package)

numbering styles used in many legal contexts, such as terms and conditions of sale. There are two main types of paragraph numbering styles: hierarchical and non-hierarchical. The first is covered in §6.5.1 and the second is covered in §6.5.2.

6.5.1 Hierarchical Paragraph Numbering

Suppose you want a document that looks something like the following:

↑ Output

1. Information About Us

    1.1. This website is run by the Secret Lab of Experimental Stuff ("We", "Our"). We operate from the University of Somewhere.

2. Our Products

    2.1. All Products shown on our site are subject to availability.

    2.2. You may only purchase our products if you are at least 18 years old.

3. Refunds

3.1. You are entitled to a refund unless:

3.1.1. you have eaten the mind-controlling cookies;

3.1.2. you have thrown the exploding chocolates;

3.1.3. you have used the ray gun as:

3.1.3.1. a table chock;

3.1.3.2. a weapon unless:

3.1.3.2.1. you have a ray gun permit;

3.1.3.2.2. you are an extraterrestrial.

↓ Output

The simplest way of achieving this is to use the enumerate environment (described in Volume 1 [92, §4.4.2]) and redefine the way the counters are displayed. By default, LaTeX allows up to four nested enumerate environments. Each level has a separate counter: enumi, enumii, enumiii and enumiv. Recall from Volume 1 [92, §11] that the displayed value of a counter is governed by the command:

\the⟨*counter*⟩ Definition

(For example, \theenumi.) The default formats for each level use: \arabic, \alph, \roman and \Alph. The format used if the items are cross-referenced using the \label/\ref mechanism prefixes \the⟨*counter*⟩ with:

\p@⟨*counter*⟩ Definition

For example, \p@enumiv\theenumiv. The prefix is ignored if \p@⟨*counter*⟩ doesn't exist (so just \the⟨*counter*⟩ is used).

The above \the⟨*counter*⟩ and \p@⟨*counter*⟩ are general counter-related commands. In addition, for the enumerate environment, there are also commands that determine the item label formats:

\label⟨*counter*⟩ Definition

(For example, \labelenumi for the first level.)

Example:
Suppose you want to have a lower case letter for the third level enumerate items:

```
\renewcommand*{\theenumiii}{\alph{enumiii}}
```
Input

and you want the label for the third level enumerate items to use parentheses:

```
\renewcommand*{\labelenumiii}{(\theenumiii)}
```
Input

but when you cross-reference a third level enumerate item, \ref should use the format ⟨*level1*⟩.⟨*level2*⟩.⟨*level3*⟩:

```
\renewcommand*{\p@enumiii}{\theenumi.\theenumii.}
```
Input

or, similarly setting \p@enumii:

```

```
Input

```
\renewcommand*{\p@enumii}{\theenumi.}
\renewcommand*{\p@enumiii}{\p@enumii\theenumii.}
```
Input

Example 36. Nested Enumeration

To reproduce the format shown at the start of this section, the standard \section command can be used for the first level ("Information About Us", "Our Products" and "Refunds"). The levels below this can be created using nested enumerate environments, where the counter formats are defined as:

```

```
Input

```
\renewcommand*{\theenumi}{\thesection.\arabic{enumi}}
\renewcommand*{\theenumii}{\theenumi.\arabic{enumii}}
\renewcommand*{\theenumiii}{\theenumii.\arabic{enumiii}}
\renewcommand*{\theenumiv}{\theenumiii.\arabic{enumiv}}
```

Input

and the labels are defined as:

Input

```
\renewcommand*{\labelenumi}{\theenumi.}
\renewcommand*{\labelenumii}{\theenumii.}
\renewcommand*{\labelenumiii}{\theenumiii.}
\renewcommand*{\labelenumiv}{\theenumiv.}
```

Input

Since all the counter formats (\theenumi, ..., \theenumiv) are now hierarchical, the internal commands used as a prefix by the cross-referencing mechanism need to be defined to do nothing:

Input

```
\renewcommand*{\p@enumi}{}
\renewcommand*{\p@enumii}{}
```

```
\renewcommand*{\p@enumiii}{}
\renewcommand*{\p@enumiv}{}
```

Input

The text can now be formatted as follows:

Input

```
\section{Information About Us}

\begin{enumerate}
 \item This website is run by the Secret Lab of Experimental Stuff
  (``We'', ``Our''). We operate from the University of Somewhere.
\end{enumerate}

\section{Our Products}

\begin{enumerate}
 \item All Products shown on our site are subject to availability.

 \item You may only purchase our products if you are at least
  18 years old.
\end{enumerate}
```

```
\section{Refunds}

\begin{enumerate}
 \item You are entitled to a refund unless:
 \begin{enumerate}
   \item you have eaten the mind-controlling cookies;
   \item you have thrown the exploding chocolates;
   \item you have used the ray gun as:
   \begin{enumerate}
     \item a table chock;
     \item a weapon unless:
     \begin{enumerate}
       \item \label{permit}you have a ray gun permit;
       \item you are an extraterrestrial.
     \end{enumerate}
   \end{enumerate}
 \end{enumerate}
\end{enumerate}
```

Input

I've labelled one of the items using \label, so I can reference it using:

```
See clause~\ref{permit}.
```
Input

which produces:

See clause 3.1.3.2.1.
Output

You can download or view a complete document.

Delving Deeper

If your document is complicated enough to require deeper levels than the default maximum of four, it's possible to extend the maximum enumerate depth. In order to do this, it's necessary to:

1. Modify the enumerate environment to allow more levels. This may additionally require modifying the underlying generic list environment that only allows a maximum of six nested list environments.

2. Define a new enum⟨n⟩ counter (using \newcounter) for each level ⟨n⟩, where ⟨n⟩ is the lower case Roman numeral representing the level index. (For example, enumv for the fifth level).

3. Define a new \labelenum⟨n⟩ command for the item label for each level ⟨n⟩.

4. Define a new \leftmargin⟨*n*⟩ length (using \newlength and \setlength) for the left margin for each level ⟨*n*⟩.

5. Define a new \@list⟨*n*⟩ command that sets up the margin for each level ⟨*n*⟩.

The counters and label commands are the easy part. For example, to allow a maximum of six levels, counters and label commands need to be defined for levels 5 (v) and 6 (vi):

Input

```
\newcounter{enumv}[enumiv]
\newcounter{enumvi}[enumv]
\newcommand*{\labelenumv}{\theenumv.}
\newcommand*{\labelenumvi}{\theenumvi.}
```

Input

(If necessary, you can also redefine the counter prefixes \p@⟨*counter*⟩.)

The left margin lengths are already provided up to six levels, but supposing you needed a seventh (vii), this would be done using:

```
\newlength\leftmarginvii
\setlength{\leftmarginvii}{15pt}
```

Input

Input

(Change the length as required.)

The \@list⟨*n*⟩ commands are more complicated. These commands need to set the length \leftmargin to the current level's left margin \leftmargin⟨*n*⟩ and set the \labelwidth length to \leftmargin⟨*n*⟩ less the value of \labelsep. Again, there are already up to six levels provided, but supposing you needed a seventh (vii), this would be done using:

```
\newcommand*{\@listvii}{%
  \setlength{\leftmargin}{\leftmarginvii}%
  \setlength{\labelwidth}{\leftmarginvii}%
  \addtolength{\labelwidth}{-\labelsep}%
}
```

Input

Input

Modifying the enumerate environment is slightly harder. This can be done using \renewenvironment (described in Volume 1 [92, §10.1]) however, in this case, only the beginning of the environment needs changing. The \begin{⟨*env-name*⟩} command works by (amongst other things) using the command \⟨*env-name*⟩ so \begin{enumerate} calls the command \enumerate, and it's this command that needs modifying. Recall from §2.1.1 that you can find out the definition of a command using either \show in your document or using the texdef script. If I run:

```
texdef -t latex enumerate
```

Shell

I get the response (tidied up a bit for legibility):

```
\enumerate:
macro:->\ifnum \@enumdepth >\thr@@
  \@toodeep
\else
  \advance\@enumdepth\@ne
  \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
  \expandafter\list\csname label\@enumctr\endcsname{%
     \usecounter\@enumctr
     \def\makelabel##1{\hss\llap{##1}}%
   }%
```

```
\fi
```

The key part here is the TEX conditional:

```
\ifnum \@enumdepth >\thr@@
```

This tests if the number stored in the \@enumdepth register is greater than \thr@@ (which is defined in the LATEX kernel to have the value 3). So the new definition of \enumerate needs to change \thr@@ to one less than the new maximum.

Since this code contains internal commands, the new definition should either be placed in a class or package or, if used in the document, be placed between \makeatletter and \makeatother. Therefore to set the maximum to six levels:

Input

```
\makeatletter

\renewcommand*{\enumerate}{%
  \ifnum \@enumdepth > 5
    \@toodeep
  \else
    \advance\@enumdepth\@ne
    \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
```

```
    \expandafter\list\csname label\@enumctr\endcsname{%
      \usecounter\@enumctr
      \def\makelabel##1{\hss\llap{##1}}%
    }%
  \fi
}

\makeatother
```

↓ Input

If you really do need more than six levels (although I hope you don't), you similarly need to modify `\list`, which by default tests if `\@listdepth` is greater than five.

Exercise 20. Extending the Maximum enumerate Depth
For this exercise, extend the maximum enumerate depth to 6 levels (as described above), so that you can create the following:

↑ Output

1. Information About Us

    1.1. This website is run by the Secret Lab of Experimental Stuff ("We", "Our"). We operate from the University of Somewhere.

2. Our Products

   2.1. All Products shown on our site are subject to availability.

   2.2. You may only purchase our products if you are at least 18 years old.

3. Refunds

   3.1. You are entitled to a refund unless:

      3.1.1. you have eaten the mind-controlling cookies;

      3.1.2. you have thrown the exploding chocolates;

      3.1.3. you have used the ray gun as:

         3.1.3.1. a table chock;

         3.1.3.2. a weapon unless:

            3.1.3.2.1. you have a ray gun permit;

            3.1.3.2.2. you are an extraterrestrial and:

               3.1.3.2.2.1. are not a resident of the planet Earth;

               3.1.3.2.2.2. have a licence under Galactic Treaty 1024, unless:

                  3.1.3.2.2.2.1. you come under Article 24, or

3.1.3.2.2.2.2. you live on Saturn.

↓ Output

The choice of document class is up to you. For example, you can use the article or scrartcl classes. (If you use scrartcl, the class option `numbers=endperiod` will display a full stop after the section numbers.) You can download or view a solution.

6.5.2 Non-Hierarchical Paragraph Numbering

Suppose now that instead of the hierarchical structure illustrated above, you simply want all your paragraphs numbered sequentially. TEX has a mechanism for specifying code that should be performed at the start of each paragraph:

`\everypar{`⟨*code*⟩`}` Definition

The numbering can be dealt with using a counter. For example, I could define a new counter called, say, para:

```
\newcounter{para}
```

Input

and I could define a command called, say, \numberedparagraph:

↑ Input

```
\newcommand*{\numberedparagraph}{%
  \refstepcounter{para}\thepara.\space
}
```

↓ Input

This command needs to go at the start of each paragraph, but it's rather tiresome to do this manually, so \everypar can be used instead. For example (using the lipsum package [32] to generate dummy text):

↑ Input

```
\everypar{\numberedparagraph}
\lipsum[1-3]
```

↓ Input

produces:

↑ Output

1. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2. Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

3. Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

↓ Output

Since the para counter is incremented using `\refstepcounter`, the paragraphs can be cross-referenced using the standard `\label`/`\ref` mechanism. If the paragraph numbering needs to be reset every page, you can specify the page counter as the "master" counter when you define the para counter:

[FAQ: Master and slave counters]

```
\newcounter{para}[page]
```

Input

⚠ In general you need to be careful about using page as a master counter, but in this case it's not a problem as this new para counter only gets incremented at the beginning of a paragraph so it doesn't conflict with TEX's output routine.

There is, however, a problem: some commands, such as the section commands, use \everypar to reset the paragraph behaviour. So, for example, a \chapter or \section command will override an earlier use of \everypar. It's also unlikely that you'll want the chapter and section headings to have a paragraph number. This last issue is easily dealt with by hooking into the sectioning commands using one of the etoolbox commands described in §2.1.2:

Input

```
\preto\chapter{\everypar{}}
\preto\section{\everypar{}}
\preto\subsection{\everypar{}}
\preto\subsubsection{\everypar{}}
\preto\paragraph{\everypar{}}
\preto\subparagraph{\everypar{}}
```

Input

The first issue, redoing \everypar{\numberedsection} after every sectioning command, is more complicated. Most classes use:

\@afterheading

Definition

within the definition of the sectioning commands. This command uses \everypar to suppress the indentation of the first paragraph following the

heading, and within the argument of \everypar there is another \everypar that resets the paragraph hook back to empty, which ensures that subsequent paragraphs are indented.

As in the previous section, either the \show command or the texdef script can be used to show the original definition of \@afterheading. For example, if I run:

```
texdef -t latex @afterheading
```

Shell

I get (reformatted for clarity):

```
\@afterheading:
macro:->\@nobreaktrue
 \everypar{%
   \if@nobreak
     \@nobreakfalse
     \clubpenalty\@M
     \if@afterindent
     \else
       {\setbox \z@ \lastbox }%
     \fi
   \else
     \clubpenalty\@clubpenalty
```

```
    \everypar{}%
  \fi
}
```

So \@afterheading can be redefined to use our new \numberedparagraph command:

↑ Input

```
\renewcommand{\@afterheading}{%
  \@nobreaktrue
  \everypar{%
    \if@nobreak
      \@nobreakfalse
      \clubpenalty\@M
      \if@afterindent
      \else
        {\setbox\z@\lastbox}%
      \fi
    \else
      \clubpenalty\@clubpenalty
      \everypar{\numberedparagraph}% <- modification
    \fi
```

```
    \numberedparagraph% <- modification
  }%
}
```

Input

Remember that this code uses internal commands, so either use \makeatletter and \makeatother or place the code in a package or class.

Exercise 21. Numbered Paragraphs
Modify the following document code so that the paragraphs are automatically numbered:

Input

```
\documentclass{article}

\usepackage{lipsum}% dummy text

\author{Some One}
\title{Numbered Paragraphs Example}

\begin{document}
\maketitle
```

```
\section{Sample Section}

\lipsum[1-4]

Some\label{sample} sample text.
\lipsum[5-10]

\subsection{Sample Subsection}

\lipsum[11-15]

\section{Another Section}

\lipsum[16-30]

\end{document}
```

↓ Input

Also, add a cross-reference to the paragraph labelled `sample`.
You can download or view a solution.

For the More Adventurous:
Modify the definition of `\numberedparagraph` so that it uses

`\marginpar[`⟨*left*⟩`]{`⟨*text*⟩`}` Definition

to put the paragraph numbers in the margins.

7. Dates and Times

There's a large list of packages related to dates and times on the calendar topic, timetable topic and date-time topic pages.

However, once you discount the plain TEX macros, the old LATEX2.09 styles, the packages that aren't included in TEX Live or MiKTEX, packages that don't have English documentation and packages that don't have proper documentation (just an example file or plain text file), then the list becomes much smaller and can be divided into those that provide commands to: compute dates; display calendars or timetables; display specific dates or times; or display the current time or the current date in a particular format (by redefining `\today`). Some of the packages cover more than one of these categories.

The datetime package, which can be used to display formatted dates and the current time, was described in Volume 1 [92, §4.2.1] and so is not covered here. Similarly the babel package [7], which also redefines `\today`, was described in Volume 1 [92, §5.8]. Since I started writing this book, I have replaced datetime with datetime2, which uses some of the code described in this chapter, so datetime2 is briefly introduced in §7.1.

⚠ In this book, "current time" and "current date" (or \today) indicate the time or date of the TEX run that created the resulting PDF (or DVI) document. If you actually want a real time clock in your document, you can use the tdclock package [69] however this will only work if your PDF viewer not only supports JavaScript (such as Adobe Reader) but also has JavaScript enabled.

Surprisingly none of the above CTAN date-related topics include the pgfcalendar package that comes with the pgf bundle [101]. This is a useful date utility package that can be used for computing dates by adding or subtracting days from a given date. It also has commands that display month and week day names, so it can be used for parsing and formatting dates. In addition, since the pgf package's main function is graphical, pgfcalendar can be used to display calendars.

This chapter is arranged as follows:

- §7.1 briefly introduces the datetime2 package.
- §7.2 describes the date utility commands of the pgfcalendar package that can: parse a date, or a date offset from another date, and convert it to a Julian day number; convert a Julian day number into an ISO-date; determine the week day from a Julian day number; test a date (for example, test if the date is on a particular week day, or is earlier or later than another date).

- §7.3 describes how to display a date using your own preferred format.
- §7.5 describes how to use the pgfcalendar package to display a calendar.
- §7.4 describes how to parse and display times.

7.1 The datetime2 Package

The datetime2 package has replaced the now obsolete datetime package. This section is just a brief introduction, see the user guide [100] for other commands and settings not described here.

The simplest use of this package is:

Input

```
\documentclass{article}
\usepackage{datetime2}
\begin{document}
This PDF was created on \today.
\end{document}
```

Input

This produces

This PDF was created on 2015-09-08.

Output

This is the default date format. If you want the full date, time and time zone, you can use

```
\DTMnow
```

Definition

instead of \today. This will display the date and time in the form

2015-09-08 11:17:58+01:00

Output

Note that XeLaTeX doesn't provide the time zone information so you will need to use PDFLaTeX or LuaLaTeX if you want this.

If you want to use a regional format, you can specify the region in the package option. For example:

Input

```
\documentclass{article}
\usepackage[en-GB]{datetime2}
\begin{document}
This PDF was created on \DTMnow.
\end{document}
```

Input

This produces:

This PDF was created on 8th September 2015 11:17am BST. Output

If you want to pick up the regional setting from babel you can use the useregional option:

Input

```
\documentclass[british]{article}
\usepackage{babel}
\usepackage[useregional]{datetime2}
\begin{document}
This PDF was created on \DTMnow.
\end{document}
```

Input

If you prefer a numeric regional format you can use useregional=numeric instead.

⚠ You must additionally install the appropriate language module, for example, datetime2-english [99] to enable the regional support. If you want to display the day of the week name, you can use the showdow package option, but this isn't available for all regions. You will need to check the documentation for the relevant module to find out if it's supported.

If you want to display a particular date (without the time) in the current style you can use

\DTMdate{⟨*date*⟩} Definition

where ⟨*date*⟩ is given in the format ⟨*yyyy*⟩-⟨*mm*⟩-⟨*dd*⟩. For example:

```
\DTMdate{2015-09-08}
```
Input

You can save a date and time for later use with:

\DTMsavetimestamp{⟨*name*⟩}{⟨*data*⟩} Definition

where ⟨*name*⟩ is a unique label that identifies this date and time, and ⟨*data*⟩ is in the format

⟨*YYYY*⟩-⟨*MM*⟩-⟨*DD*⟩T⟨*hh*⟩:⟨*mm*⟩:⟨*ss*⟩⟨*zone*⟩

where ⟨*YYYY*⟩ is the year, ⟨*MM*⟩ is the month number, ⟨*DD*⟩ is the day of the month, ⟨*hh*⟩ is the hour (24), ⟨*mm*⟩ is the minute value, ⟨*ss*⟩ is the second value and ⟨*zone*⟩ is the time zone, which may be either Z or in the format ⟨*TZh*⟩:⟨*TZm*⟩ where ⟨*TZh*⟩ is the hour offset and ⟨*TZm*⟩ is the minute offset. The argument ⟨*data*⟩ may also be a control sequence that expands (one level) to the required format.

Alternatively, you can just save the date with:

\DTMsavedate{⟨name⟩}{⟨date⟩} Definition

where the ⟨date⟩ is in the format ⟨YYYY⟩-⟨DD⟩-⟨MM⟩ and ⟨name⟩ is again a label.

A previously saved date and time can be displayed in the current style using:

\DTMuse{⟨name⟩} Definition

Just the date can be displayed using:

\DTMusedate{⟨name⟩} Definition

and just the time with:

\DTMusetime{⟨name⟩} Definition

In all cases, ⟨name⟩ is the label identifying the date or time stamp.

Example 37. Displaying Dates and Times (datetime2 package)
Here's an example that uses the en-GB style:

↑ Input

```
\documentclass{article}
```

```
\usepackage[en-GB]{datetime2}

\newcommand*{\DateStamp}{2015-11-28T20:13:04Z}
\DTMsavetimestamp{mydate}{\DateStamp}
\DTMsavetimestamp{mydate2}{2014-06-01T09:01:58+01:00}

\begin{document}
Now: \DTMnow.

Saved: \DTMuse{mydate}; \DTMuse{mydate2}.

\end{document}
```

↓ Input

This produces

↑ Output

Now: 8th September 2015 11:25am BST.
Saved: 28th November 2015 8:13pm GMT; 1st June 2014 9:01am BST.

↓ Output

You can download or view this document.

⚠ Be careful if you are using babel with the shorthands on as this may change the category code of characters such as : and - which will cause the date or time parsing in commands like \DTMsavetimestamp to fail. You may need to temporarily switch off the shorthands. See the babel manual [7] for further details.

7.2 The pgfcalendar Package Utility Commands

The pgfcalendar package may be used independently of the pgf package, but if used without pgf (or tikz) the pgfkeys package also needs to be loaded:

```
\usepackage{pgfkeys}
\usepackage{pgfcalendar}
```

Input

or

```
\usepackage{pgfkeys,pgfcalendar}
```

Input

In the command definitions below, ⟨*date*⟩ indicates a date specified using one of the following formats:

- A specific date:

  ⟨*year*⟩-⟨*month*⟩-⟨*day*⟩

- The last day of a particular month:

  ⟨*year*⟩-⟨*month*⟩-`last`

- An increment from a given date:

  ⟨*year*⟩-⟨*month*⟩-⟨*day*⟩+⟨*increment*⟩

  or

  ⟨*year*⟩-⟨*month*⟩-`last`+⟨*increment*⟩

Where ⟨*year*⟩ is the year (for example, `2014` or `\year` for the current year); ⟨*month*⟩ is the month number (for example, `6` for June or `\month` for the current month) and ⟨*day*⟩ is the day number (for example, `21` for the twenty-first of the month or `\day` for the current day). The ⟨*increment*⟩ (in days) may be either a positive or negative number. If negative the leading + is still required. For example, `2014-6-last+-4` means four days before the last day of June.

The pgfcalendar provides the commands:

\pgfcalendardatetojulian{⟨*date*⟩}{⟨*register*⟩} Definition

This converts a Gregorian date into the Julian day number and stores the result in ⟨*register*⟩, which must be a TEX register (*not* a LATEX counter).

Example:
(Recall \newcount from §2.1.3.)

↑ Input

```
\newcount\mycount
\pgfcalendardatetojulian{2014-03-18+2}{\mycount}
\the\mycount
```

↓ Input

produces:

2456737 Output

\pgfcalendarjuliantodate{⟨*Julian day*⟩}{⟨*year cs*⟩}{⟨*month cs*⟩}{⟨*day cs*⟩} Definition

This converts a Julian day number to an ISO-date and stores the resulting year, month and day-of-month numbers in the control sequences ⟨*year cs*⟩, ⟨*month cs*⟩ and ⟨*day cs*⟩.

Example:

```
\pgfcalendarjuliantodate{2456737}{\theyear}{\themonth}{\theday}
\theyear/\themonth/\theday
```

Input

Input

produces:

2014/03/20

Output

`\pgfcalendarjuliantoweekday{`⟨*Julian day*⟩`}{`⟨*register*⟩`}`

Definition

This converts a Julian day number to a week day number, where 0 indicates Monday, 1 indicates Tuesday, etc. The result is stored in the TEX register specified by ⟨*register*⟩.

Example:

```
\newcount\mycount
\pgfcalendarjuliantoweekday{2456737}{\mycount}
\the\mycount
```

Input

Input

produces:

3 Output

(which indicates Thursday).

\pgfcalendarifdate{⟨*date*⟩}{⟨*test*⟩}{⟨*true code*⟩}{⟨*false code*⟩} Definition

This tests the given date and does ⟨*true code*⟩ if the test succeeds otherwise it does ⟨*false code*⟩. The ⟨*test*⟩ may be one of the following key words:

`all`	Always yields true.
`Monday`	True if the date is a Monday.
`Tuesday`	True if the date is a Tuesday.
`Wednesday`	True if the date is a Wednesday.
`Thursday`	True if the date is a Thursday.
`Friday`	True if the date is a Friday.
`Saturday`	True if the date is a Saturday.
`Sunday`	True if the date is a Sunday.

`workday` True if the date occurs from Monday to Friday, inclusive.

`weekend` True if the date is a Saturday or Sunday.

Or the ⟨*test*⟩ may be a comparison against a reference, which may be an ISO date in the form ⟨*yyyy*⟩`-`⟨*mm*⟩`-`⟨*dd*⟩ (for example, `2014-03-20`) or with the year missing ⟨*mm*⟩`-`⟨*dd*⟩ (for example, `03-20`):

- `equals=`⟨*reference*⟩

  True if ⟨*date*⟩ is the same date as reference (where the year is specified in ⟨*reference*⟩) or has the same month and day parts as the reference (where the year is omitted from ⟨*reference*⟩).

- `at least=`⟨*reference*⟩

  True if ⟨*date*⟩ is equal to ⟨*reference*⟩ or is a later date than ⟨*reference*⟩. If the year is omitted from ⟨*reference*⟩, only the month and day in ⟨*date*⟩ form part of the test.

- `at most=`⟨*reference*⟩

  The reverse of the above.

- `between=`⟨*start reference*⟩ and ⟨*end reference*⟩

True if ⟨*date*⟩ lies between the two reference dates. If the year is omitted from the reference, only the month and day in ⟨*date*⟩ form part of the test.

- `day of month=`⟨*number*⟩

  True if the day of the month in ⟨*date*⟩ is equal to ⟨*number*⟩.

- `end of month=`⟨*number*⟩

  This is the reverse of the above in the sense that it's testing ⟨*number*⟩ against the day of the month counting backward from the end of the month. So `end of month=1` yields true if ⟨*date*⟩ is the last day of the month, and `end of month=2` yields true if ⟨*date*⟩ is the penultimate day of the month. If ⟨*number*⟩ is omitted, it's assumed to be 1.

Example:

Input

```
2014-03-20 is in the
\pgfcalendarifdate{2014-03-20}{at most=06-last}
{first}% test true
```

```
{second}% test false
\space half of the year.
```

Input

produces:

2014-03-20 is in the first half of the year.

Output

What happens if your date isn't in the form ⟨*yyyy*⟩-⟨*m*⟩-⟨*d*⟩? For example, it might be in the form ⟨*m*⟩/⟨*d*⟩/⟨*yyyy*⟩. Recall from §2.1.1 that the \def primitive can be used to define a macro that has a custom syntax. It's therefore possible to define a command that will parse this syntax and convert it:

```
\def\parsemdydate#1/#2/#3\endparsemdydate{#3-#1-#2}
```

Input

(The \endparsemdydate token is just an end placeholder, not a command that needs defining.)

Now

```
\parsemdydate 3/19/2014\endparsemdydate
```

Input

expands to 2014-3-19. Remember that if the date is stored in a macro, for example:

```
\newcommand*{\mydate}{3/19/2014}
```
Input

then you first need to expand the macro before it can be parsed by \parsemdydate. (Recall \expandafter from §2.7.2.)

```
\expandafter\parsemdydate\mydate\endparsemdydate
```
Input

Example 38. Calculating Ages

Remember that the sample people.csv file and people SQL table included a date of birth field (labelled dob). This example computes the ages of each person in that data. This is done by first computing the Julian day number for today. Then for each person in the database, the Julian day number is computed for that person's date of birth. This number is subtracted from the Julian day number for today. This gives the total number of days since that person was born. This number is then divided by 356 to give an approximate age in years. (Recall TEX's integer arithmetic described in §2.1.3.)

↑ Input

```
\documentclass[captions=tableheading]{scrartcl}

\usepackage[utf8]{inputenc}
```

```
\usepackage[T1]{fontenc}

\usepackage{pgfkeys,pgfcalendar}
\usepackage{datatool}

\DTLloaddb{people}{people.csv}

\newcount\julianday
\newcount\juliantoday
\newcount\age

% Compute the Julian day number for today:
\pgfcalendardatetojulian{\year-\month-\day}{\juliantoday}

\begin{document}
Ages as of \number\year-\number\month-\number\day\␣are
listed in Table~\ref{tab:ages}.

\begin{table}[htbp]
\caption{Ages}
\label{tab:ages}
\centering
```

```
\begin{tabular}{lc}
\bfseries Name & \bfseries Age%
\DTLforeach*{people}%
 {\Forenames=forenames,\Surname=surname,\DoB=dob}%
 {%
    \\\Forenames\␣\Surname &
    % Compute the Julian day number for the date of birth
    \pgfcalendardatetojulian{\DoB}{\julianday}%
    % Compute \age = (\juliantoday - \julianday)/365
    \age=\juliantoday
    \advance\age by -\julianday
    \divide\age by 365
    \number\age
 }%
\end{tabular}
\end{table}

\end{document}
```

Input

This produces Table 7.1 and the text:

Ages as of 2021-9-6 are listed in Table 7.1.

Output

Output

Table 7.1 Ages

Name	Age
Polly Parrot	50
Mabel Canary	53
Zöe Zebra	32
José Arara	30
Dickie Duck	68
Fred Canary	54

You can download or view this example. Remember that if your dates are in a different numerical format, for example, ⟨*m*⟩/⟨*d*⟩/⟨*yyyy*⟩, you need to convert them as described above. For example, replace

```
\pgfcalendardatetojulian{\DoB}{\julianday}
```

Input

with

↑ Input

```
\pgfcalendardatetojulian
  {\expandafter\parsemdydate\DoB\endparsemdydate}{\julianday}
```

↓ Input

where \parsemdydate is as described above.

7.3 Displaying a Date

In addition to the utility commands described above, the pgfcalendar package also provides commands to display month names and the day of week names.

\pgfcalendarweekdayname{⟨*week day number*⟩} Definition

This expands to a textual representation of the day of the week, where the numbering starts from 0 (Monday).

\pgfcalendarweekdayshortname{⟨*week day number*⟩} Definition

This expands to an abbreviated textual representation of the day of the week, ("Mon", "Tue", etc) where the numbering starts from 0 (Monday).

\pgfcalendarmonthname{⟨*month number*⟩} Definition

This expands to a textual representation of the month name.

\pgfcalendarmonthshortname{⟨*month number*⟩} Definition

This expands to an abbreviated textual representation of the month name ("Jan", "Feb", etc).

If you want the name in another language, you need to load the translator package[1] as well as babel [7]. For example:

```
\documentclass[french,german]{article}

\usepackage{babel}
\usepackage{translator}
\usepackage{pgfkeys,pgfcalendar}
```

↑ Input

↓ Input

(Only a limited number of languages are supported.)

Example:
Today's date can be formatted using the month name:

[1] provided with beamer [102]

```
\number\day~\pgfcalendarmonthname{\month} \number\year
```

Input

which produces:

6 September 2021

Output

Alternatively, you can define a custom command called, say, \datefmt that formats any date:

↑ Input

```
\newcommand*{\datefmt}[3]{%
  \number#3~\pgfcalendarmonthname{#2} \number#1%
}
```

↓ Input

This has the syntax:

\datefmt{⟨*year*⟩}{⟨*month*⟩}{⟨*day*⟩}

Definition

so now today's date can be formatted using:

```
\datefmt{\year}{\month}{\day}
```

Input

Alternatively, a specific date, for example, 2014-01-31 can be formatted using:

```
\datefmt{2014}{1}{31}
```

Input

which produces:

31 January 2014

Output

If you want to use an ordinal instead of a plain number for the day of the month (for example, 1st instead of 1), then you can use TEX's \ifcase conditional:

```
\ifcase ⟨number⟩
  ⟨case 0 code⟩%
\or
  ⟨case 1 code⟩%
\or
  ⟨case 2 code⟩%
\or
  ...
\else
  ⟨default code⟩%
\fi
```

Definition

This tests ⟨*number*⟩. If ⟨*number*⟩ equals 0, ⟨*case 0 code*⟩ is performed. If ⟨*number*⟩ equals 1, ⟨*case 1 code*⟩ is performed. If ⟨*number*⟩ equals 2, ⟨*case 2 code*⟩ is performed, etc. If none of the cases match, ⟨*default code*⟩

is performed. (The \else ⟨*default code*⟩ part may be omitted.) Since there are a maximum of 31 days in a month, 32 cases (including the unnecessary case 0) are needed:

Input

```
\newcommand*{\ord}[1]{%
  \number#1%
  \ifcase#1\or st\or nd\or rd\or th\or th\or th\or th\or
  th\or th\or th\or th\or th\or th\or th\or th\or th\or
  th\or th\or th\or th\or st\or nd\or rd\or th\or th\or
  th\or th\or th\or th\or th\fi
}
```

Input

Now

\ord{1}

Input

produces:

1st

Output

and

```
\ord{2}
```
Input

produces:

2nd
Output

etc. So the definition of \datefmt can now be defined as:

```
\newcommand*{\datefmt}[3]{%
  \ord{#3}~\pgfcalendarmonthname{#2} \number#1%
}
```
Input

If you like, you can redefine \today to use this format:

```
\renewcommand*{\today}{\datefmt{\year}{\month}{\day}}
```
Input

⚠ Take care if you use babel as it redefines \today every time the language changes (including at the beginning of the document environment). It does this via the commands \date⟨*language*⟩ that are invoked when the current language switches to ⟨*language*⟩. Each \date⟨*language*⟩ command redefines \today to use the format for ⟨*language*⟩. So if you want to redefine \today when you are using babel, you need to redefine \date⟨*language*⟩.

Example:
Suppose I'm using babel with the `british` option. In this case, the date is reset using `\datebritish` so I need to redefine it to use my own format instead:

Input

```
\renewcommand*{\datebritish}{%
  \renewcommand*{\today}{\datefmt{\year}{\month}{\day}}%
}
```

Input

Example 39. Custom Date Formatting
Suppose now you want to include the day of the week in your custom date format, or perhaps you want to be able to specify the date in the ISO numeric format, with possibly an increment, as with the first argument of `\pgfcalendardatetojulian`.

Recall from §7.2 that the week day can be obtained from a Julian day number using `\pgfcalendarjuliantoweekday` and the Julian day number can be obtained from a date using `\pgfcalendardatetojulian`. In this example, I'm going to define a new command called `\printdate` with the syntax:

\printdate{⟨*date*⟩} Definition

This will display the date in the form: ⟨*day name*⟩ ⟨*day of month number*⟩ ⟨*month name*⟩ ⟨*year*⟩. First, two new count registers need to be defined:

Input

```
\newcount\julianday
\newcount\dayofweek
```

Input

Next define the new command:

Input

```
\newcommand*{\printdate}[1]{%
  \pgfcalendardatetojulian{#1}{\julianday}%
  \pgfcalendarjuliantodate{\julianday}{\thisyear}{\thismonth}
{\thisday}%
  \pgfcalendarjuliantoweekday{\julianday}{\dayofweek}%
  % Now display the date:
  \datefmt[\dayofweek]{\thisyear}{\thismonth}{\thisday}%
}
```

Input

The actual date format, including the day of week, is dealt with by a new version of \datefmt that now has four arguments:

\datefmt[⟨*day of week*⟩]{⟨*yyyy*⟩}{⟨*m*⟩}{⟨*d*⟩} Definition

This command is defined as follows:

↑ Input

```
\newcommand*{\datefmt}[4][]{%
  \ifstrempty{#1}
  {}% day of week missing
  {%
    \pgfcalendarweekdayname{#1}\space
  }%
  \ord{#4}~\pgfcalendarmonthname{#3} \number#2%
}
```

↓ Input

This uses etoolbox's \ifstrempty command to omit the day of week name if the optional argument is absent. This means that you can still directly use, for example:

```
\datefmt{2014}{1}{31}
```
Input

and not worry about the week day.

Now this new \printdate command can be used in the document. For example:

```
\printdate{2014-05-last}
```
Input

produces:

Saturday 31st May 2014

Output

If you want \today to use the same format, then you can just redefine \today:

```
\renewcommand*{\today}{%
  \pgfcalendardatetojulian{\year-\month-\day}{\julianday}%
  \pgfcalendarjuliantoweekday{\julianday}{\dayofweek}%
  \datefmt[\dayofweek]{\year}{\month}{\day}%
}
```
↑ Input

↓ Input

Again, you need to put this in the definition of \date⟨*language*⟩ if you are using babel.

You can download or view a complete document.

If this is a format you are likely to use in multiple documents, you might want to define your own custom package called, say, mycustomdate. This requires creating a file called `mycustomdate.sty`, that contains the following:

↑ Input

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{mycustomdate}[2014/03/19 1.0 My custom date
format]

\RequirePackage{etoolbox}
\RequirePackage{pgfkeys,pgfcalendar}

% Command definitions for \printdate, \today, \datefmt
% \ord (if required) and \date⟨language⟩ (if required).

\endinput
```

↓ Input

This file should then be saved in your TEXMF path. For example, if you are using a Unix-like operating system, you can save it in, say, `~/texmf/tex/latex/mystuff/` (see Volume 1 [92, §A]).

If you're unfamiliar with writing packages, here's a brief explanation of the commands used above:

`\NeedsTeXFormat{`⟨*format*⟩`}[`⟨*version*⟩`]` Definition

This should be the first statement of any class or package and is used to identify the TEX format and, optionally, the version date. For a LATEX 2ε class or package, the ⟨*format*⟩ should be `LaTeX2e`. (Other formats may not define this command.) The version date, if present, must be in the numeric form ⟨*yyyy*⟩/⟨*mm*⟩/⟨*dd*⟩ (two digits are required for both the month and day numbers).

`\ProvidesPackage{`⟨*name*⟩`}[`⟨*version*⟩`]` Definition

This command identifies the package name and optionally a version. The ⟨*name*⟩ should match the filename (without the extension), so a package called `mycustomdate` should be in a file called `mycustomdate.sty`. The ⟨*version*⟩ should start with a numeric date in the form ⟨*yyyy*⟩/⟨*mm*⟩/⟨*dd*⟩ and may optionally be followed by a version number and a brief description.

\RequirePackage[⟨options⟩]{⟨name⟩}[⟨version⟩] Definition

This is analogous to \usepackage but is for use in a class or package. The final optional argument ⟨version⟩ indicates that the package must be at least that version. If an older version is installed a warning is issued.

\endinput Definition

This is a TEX primitive that instructs TEX to stop reading the current file. Anything following this command is skipped. (Some packages have their documentation in the .sty file after \endinput, but this practice has been deprecated in favour of providing the documentation as a PDF.)

Once you have added mycustomdate.sty to your TEX path, you can now load this package in your document via

\usepackage{mycustomdate} Input

Take care if you need to use babel. If this custom package includes code to redefine \date⟨language⟩ you will need to load babel first (and remember to load the translator package as well for the month and day of week names). Alternatively, you can check for the existence of \date⟨language⟩ at the start of the document environment, and redefine it if it exists:

↑ Input

```
\AtBeginDocument{%
  \ifdef{\datebritish}% check if \datebritish exists
  {% it does exist, so redefine it
    \renewcommand*{\datebritish}{%
      \renewcommand*{\today}{%
        \pgfcalendardatetojulian{\year-\month-\day}{\julianday}%
        \pgfcalendarjuliantoweekday{\julianday}{\dayofweek}%
        \datefmt[\dayofweek]{\year}{\month}{\day}%
      }%
    }%
    \datebritish
  }%
  {}% doesn't exist, do nothing
}
```

Input

Example 40. Custom Date Package

Putting the above together, here's a complete example that defines a package that uses a British date format:

Input

```
\NeedsTeXFormat{LaTeX2e}
```

```
\ProvidesPackage{mycustomdate}[2014/03/19 1.0 My custom date
format]

\RequirePackage{etoolbox}%
\RequirePackage{pgfkeys,pgfcalendar}

% Define an ordinal command:
\newcommand*{\ord}[1]{%
  \number#1%
  \ifcase#1\or st\or nd\or rd\or th\or th\or th\or th\or
  th\or th\or th\or th\or th\or th\or th\or th\or th\or
  th\or th\or th\or th\or st\or nd\or rd\or th\or th\or
  th\or th\or th\or th\or st\fi
}

% Define registers needed by \printdate:
\newcount\julianday
\newcount\dayofweek

% Define generic date format:
\newcommand*{\datefmt}[4][]{%
  \ifstrempty{#1}
```

```
  {}% day of week missing
  {%
    \pgfcalendarweekdayname{#1}\space
  }%
  \ord{#4}~\pgfcalendarmonthname{#3} \number#2%
}

% Define command to read ISO date and then use \datefmt
\newcommand*{\printdate}[1]{%
  \pgfcalendardatetojulian{#1}{\julianday}%
  \pgfcalendarjuliantodate{\julianday}{\thisyear}{\thismonth}{\thisday}%
  \pgfcalendarjuliantoweekday{\julianday}{\dayofweek}%
  % Now display the date:
  \datefmt[\dayofweek]{\thisyear}{\thismonth}{\thisday}%
}

% Redefine \today to use the same format:
\renewcommand*{\today}{%
  \pgfcalendardatetojulian{\year-\month-\day}{\julianday}%
  \pgfcalendarjuliantoweekday{\julianday}{\dayofweek}%
  \datefmt[\dayofweek]{\year}{\month}{\day}%
}
```

```
% Check if babel is used with the british option:
\AtBeginDocument{%
  \ifdef{\datebritish}% check if \datebritish exists
  {% it does exist, so redefine it
    \renewcommand*{\datebritish}{%
      \renewcommand*{\today}{%
        \pgfcalendardatetojulian{\year-\month-\day}{\julianday}%
        \pgfcalendarjuliantoweekday{\julianday}{\dayofweek}%
        \datefmt[\dayofweek]{\year}{\month}{\day}%
      }%
    }%
    \datebritish
  }%
  {}% doesn't exist, do nothing
}

\endinput
```

↓ Input

(You can download this package.)

And here's a document that uses this package:

↑ Input

```
\documentclass{article}

\usepackage{mycustomdate}

\begin{document}

Today: \today.
Tomorrow: \printdate{\year-\month-\day+1}.
Yesterday: \printdate{\year-\month-\day+-1}.

The first day of this month: \printdate{\year-\month-1}.
The last day of this month: \printdate{\year-\month-last}.

A specific date: \printdate{2014-3-20}.

A date without the day of week: \datefmt{2014}{3}{20}.

\end{document}
```

Input

You can download or view this document.

7.4 Parsing and Displaying Times

The pgfcalender doesn't provide any time-related utilities. TEX's \time primitive expands to the current time in terms of the number of minutes since midnight. The datetime package works out the current hour and minute by performing some arithmetic on the value of \time, but since \time is an integer number of minutes, there's no information about the number of seconds nor is there any information about the time zone. (The new datetime2 package uses the methods described in this section to determine the current time.) PDFTEX comes with a primitive[2] called

\pdfcreationdate Definition

that expands to D:⟨*YYYY*⟩⟨*MM*⟩⟨*DD*⟩⟨*hh*⟩⟨*mm*⟩⟨*ss*⟩⟨*time zone*⟩ where ⟨*YYYY*⟩ is the year (four digits), ⟨*MM*⟩ is the month number (two digits), ⟨*DD*⟩ is the day of the month (two digits), ⟨*hh*⟩ is the hour (two digits), ⟨*mm*⟩ is the number of minutes past the hour (two digits), ⟨*ss*⟩ is the number of seconds past the minute (two digits) and ⟨*time zone*⟩ is the time zone, which may be Z (for UTC+00) or +⟨*HH*⟩'⟨*mm*⟩' (for UTC+⟨*HH*⟩:⟨*mm*⟩) or -⟨*HH*⟩'⟨*mm*⟩' (for UTC−⟨*HH*⟩:⟨*mm*⟩).

The value of \pdfcreationdate is set at the start of the PDFTEX run.

[2] For further details about PDFTEX primitives see the PDFTEX documentation [105].

Example:

```
\pdfcreationdate
```
Input

produces:

D:20210906122340Z
Output

Recall the \parsemdydate command defined in §7.2 used \def to parse a date string. A similar method can be employed here, but unfortunately it's more complicated. At first glance it looks as though we can define a command in the form:

```
\def\parsepdfdatetime D:#1\endparsepdfdatetime{⟨code⟩}
```


However if we try this out (ignoring the argument for the time being):

```
\def\parsepdfdatetime D:#1\endparsepdfdatetime{}
\expandafter\parsepdfdatetime\pdfcreationdate\endparsepdfdatetime
```
↑ Input

↓ Input

we get an error:

```
! Use of \parsepdfdatetime doesn't match its definition.
<inserted text> D
                :20140319185833Z
l.10 \expandafter\parsepdfdatetime\pdfcreationdate
                                                \endparsepdfdatetime
```

The problem is the initial D as the following works fine:[3]

Input

```
\def\parsepdfdatetime#1:#2\endparsepdfdatetime{}
\expandafter\parsepdfdatetime\pdfcreationdate\endparsepdfdatetime
```

Input

The first argument (#1) will always be D and can be ignored. Since TEX only allows a maximum of nine arguments, this leaves eight arguments left, which can pick up the year (#2#3#4#5), month (#6#7) and day (#8#9) digits. This information is already available from TEX's \year, \month and \day primitives, however PDFTEX provides a similar primitive:

[3] It's the category code of the character "D" in the expansion of \pdfcreationdate that's the problem. If it's first changed to "other" (category code 12) before defining \parsepdfdatetime then the error won't occur.

\pdffilemoddate{⟨*filename*⟩} Definition

that expands to the modification date and time for the file given by ⟨*filename*⟩, and this uses the same format, so \parsepdfdatetime should still save the year, month and day information to be more generally useful.

In order to get around the nine argument maximum \parsepdfdatetime needs to call another command that will parse the remainder:

Input

```
\def\parsepdfdatetime#1:#2#3#4#5#6#7#8#9{%
  \def\theyear{#2#3#4#5}%
  \def\themonth{#6#7}%
  \def\theday{#8#9}%
  \parsepdftime
}
```

Input

Note that the end placeholder token \endparsepdfdatetime is no longer in the argument syntax of \parsepdfdatetime. It's now in the argument syntax of the new \parsepdftime command, which picks up the remaining time information:

Input

```
\def\parsepdftime#1#2#3#4#5#6#7\endparsepdfdatetime{%
  \def\thehour{#1#2}%
  \def\theminute{#3#4}%
  \def\thesecond{#5#6}%
  \def\thetimezone{#7}%
}
```

Input

The hour digits are now given by #1#2, the minute digits are #3#4 the second digits are #5#6 and the time zone information is in the final argument #7. This information has been stored in the new commands \thehour, \theminute, \thesecond and \thetimezone. If you want \thetimezone to be in the format ⟨*sign*⟩⟨*HH*⟩:⟨*mm*⟩ (where ⟨*sign*⟩ is either + or -) then replace:

```
\def\thetimezone{#7}%
```

Input

with

Input

```
\ifstrequal{#7}{Z}
```

```
  {%
    \def\thetimezone{+00:00}%
  }%
  {%
    \parsepdftimezone#7%
  }%
```

Input

where \parsepdftimezone is defined as:

Input

```
\def\parsepdftimezone#1'#2'{%
  \def\thetimezone{#1:#2}%
}
```

Input

(\ifstrequal is defined by etoolbox and tests if two strings are equal, but unlike \ifthenelse{\equal{⟨*string1*⟩}{⟨*string2*⟩}}{}{}, \ifstrequal doesn't perform any expansion on the strings.)

As with the \datefmt command defined in the previous section, we can also define an analogous command to format the time:

Input

```
\newcommand*{\timefmt}[4]{%
  #1:#2:#3#4%
}
```

Input

This has the syntax:

\timefmt{⟨*hour*⟩}{⟨*minutes*⟩}{⟨*seconds*⟩}{⟨*UTC offset*⟩} Definition

For example:

```
\timefmt{11}{03}{01}{+01:00}
```

Input

produces:

11:03:01+01:00 Output

Another possible definition is:

Input

```
\newcommand*{\timefmt}[4]{%
  #1:#2%
  \ifstrempty{#3}% test for empty 3rd argument
```

```
  {}% no seconds specified
  {:#3}% seconds
  \ifstrempty{#4}% test for empty 4th argument
  {}% no time zone
  {#4}%
}
```

Input

This uses etoolbox's \ifstrempty command to determine whether or not ⟨*seconds*⟩ or ⟨*UTC offset*⟩ have been specified. Alternatively, the time zone information can be dealt with by another command called, say, \timezonefmt:

Input

```
\newcommand*{\timefmt}[4]{%
  #1:#2%
  \ifstrempty{#3}% test for empty 3rd argument
  {}% no seconds specified
  {:#3}% seconds
  \timezonefmt{#4}% time zone
}
```

Input

Since it's possible that the time zone might not be fully expanded (for example, the argument might be \thetimezone), the new \timezonefmt first fully expands its argument before parsing it:

Input

```
\newcommand*{\timezonefmt}[1]{%
  \edef\thistimezone{#1}%
  \ifdefempty{\thistimezone}%
  {}% empty argument
  {%
    \expandafter\@timezonefmt\thistimezone\@endtimezonefmt
  }%
}
```

Input

Now the actual parsing is done by a new internal command with the syntax:

\@timezonefmt⟨*HH*⟩:⟨*mm*⟩\@endtimezonefmt Definition

Here's one possible definition of \@timezonefmt that just displays "Z" if both ⟨*HH*⟩ and ⟨*mm*⟩ are zero, otherwise it either does just ⟨*HH*⟩ if ⟨*mm*⟩ is zero or it does ⟨*HH*⟩:⟨*mm*⟩

Input

```
\def\@timezonefmt#1:#2\@endtimezonefmt{%
  \ifnum#2=0\relax
    \ifnum#1=0\relax
      Z%
    \else
      #1%
    \fi
  \else
    #1:#2%
  \fi
}
```

Input

Example:
Since \pdfcreationdate is set at the start of the LaTeX run, you only need to parse it once:

Input

```
\expandafter\parsepdfdatetime\pdfcreationdate\endparsepdfdatetime
\let\pdfhour\thehour
```

```
\let\pdfminute\theminute
\let\pdfsecond\thesecond
\let\pdftimezone\thetimezone
\newcommand*{\pdfnowtime}{%
  \timefmt{\pdfhour}{\pdfminute}{\pdfsecond}{\pdftimezone}}
```

Input

The time stamp for the LaTeX run can now be inserted into your document using this new \pdfnowtime command:

```
This PDF was created at \pdfnowtime.
```

Input

produces:

This PDF was created at 12:23:40Z.

Output

Example 41. Custom Date and Time Package

This example extends the custom package described in Example 40. I've added the LaTeX internal command:

\two@digits{⟨*number*⟩}

Definition

which ensures ⟨*number*⟩ has at least two digits. Take care when using commands that print numbers, such as \two@digits or \number, as you can

unexpectedly lose following spaces. It's for this reason that I've occasionally used \relax or \␣ (backslash space) in the code below.

This example package is now called mycustomdatetime so it needs to have the filename mycustomdatetime.sty and the package declaration should be modified accordingly:

↑ Input

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{mycustomdatetime}[2014/03/20 1.0 My custom date
and time format]
```

↓ Input

Remember the etoolbox package is required:

```
\RequirePackage{etoolbox}
```

Input

Now the user command

\timefmt{⟨*hh*⟩}{⟨*mm*⟩}{⟨*ss*⟩}{⟨*HH*⟩:⟨*SS*⟩}

Definition

is defined:

Input

```
\newcommand*{\timefmt}[4]{%
  \two@digits{#1}:\two@digits{#2}%
  \ifstrempty{#3}% test for empty 3rd argument
  {}% no seconds specified
  {:\two@digits{#3}}% seconds
  \timezonefmt{#4}% time zone
  \relax
}
```

Input

and its helper time zone formatting command:

`\timezonefmt{`⟨*HH*⟩`:`⟨*SS*⟩`}` Definition

is defined:

Input

```
\newcommand*{\timezonefmt}[1]{%
  \edef\thistimezone{#1}%
  \ifdefempty{\thistimezone}%
  {}% empty argument
```

```
  {%
    \expandafter\@timezonefmt\thistimezone\@endtimezonefmt
  }%
}
```

Input

along with its internal command:

Input

```
\def\@timezonefmt#1:#2\@endtimezonefmt{%
  \ifnum#2=0\relax
    \ifnum#1=0\relax
      Z%
    \else
      #1%
    \fi
  \else
    #1:#2%
  \fi
}
```

Input

If you want to use \two@digits in the time zone, you need to be careful with the plus or minus sign:

Input

```
\def\@timezonefmt#1:#2\@endtimezonefmt{%
  \ifnum #2=0\relax
    \ifnum #1=0\relax
      Z%
    \else
      \ifnum #1<0 $-$\two@digits{-#1}\else +\two@digits{#1}\fi
    \fi
  \else
    \ifnum #1<0 $-$\two@digits{-#1}\else +\two@digits{#1}\fi
    :\two@digits{#2}%
  \fi
}
```

Input

(The minus sign has been placed in math-mode using $-$ to ensure it's displayed as a real minus sign rather than as a hyphen. If for some reason you need to use \timefmt in math-mode, I suggest you put it in inside the argument of amsmath's \text command [1].)

Now for the commands that can parse the PDF date format:

↑ Input

```
\def\parsepdfdatetime#1:#2#3#4#5#6#7#8#9{%
  \def\theyear{#2#3#4#5}%
  \def\themonth{#6#7}%
  \def\theday{#8#9}%
  \parsepdftime
}

\def\parsepdftime#1#2#3#4#5#6#7\endparsepdfdatetime{%
  \def\thehour{#1#2}%
  \def\theminute{#3#4}%
  \def\thesecond{#5#6}%
  \ifstrequal{#7}{Z}
  {%
    \def\thetimezone{+00:00}%
  }%
  {%
    \parsepdftimezone#7%
  }%
}
```

```
\def\parsepdftimezone#1'#2'{%
  \def\thetimezone{#1:#2}%
}
```

Input

Provide a convenient way of displaying the document build time:

Input

```
\expandafter\parsepdfdatetime\pdfcreationdate\endparsepdfdatetime
\let\pdfhour\thehour
\let\pdfminute\theminute
\let\pdfsecond\thesecond
\let\pdftimezone\thetimezone

\newcommand*{\pdfnowtime}{%
  \timefmt{\pdfhour}{\pdfminute}{\pdfsecond}{\pdftimezone}}
```

Input

and for a complete date and time stamp:

```
\newcommand*{\pdfnow}{\today\␣\pdfnowtime}
```
Input

Alternatively if you want a numeric date independent of the definition of \today:

```
\newcommand*{\pdfnow}{%
 \year-\two@digits{\month}-\two@digits{\day}\space
 \pdfnowtime
}
```
Input

Similarly define a command with the syntax:

\filedate{⟨*filename*⟩}

Definition

that will display the time stamp of a file:

```
\newcommand*{\filedate}[1]{%
  \expandafter\parsepdfdatetime\pdffilemoddate{#1}\endparsepdfdatetime
  \datefmt{\theyear{\themonth}{\theday}\space
  \timefmt{\thehour}{\theminute}{\thesecond}{\thetimezone}}%
}
```
Input

Alternatively, if you want the day of week information to also be shown:

Input

```
\newcommand*{\filedate}[1]{%
  \expandafter\parsepdfdatetime\pdffilemoddate{#1}\endparsepdfdatetime
  \printdate{\theyear-\themonth-\theday}\␣
  \timefmt{\thehour}{\theminute}{\thesecond}{\thetimezone}%
}
```

Input

Or if you just want a numeric format regardless of the definition of \printdate:

Input

```
\newcommand*{\filedate}[1]{%
  \expandafter\parsepdfdatetime\pdffilemoddate{#1}\endparsepdfdatetime
  \theyear-\two@digits{\themonth}-\two@digits\theday\␣
  \timefmt{\thehour}{\theminute}{\thesecond}{\thetimezone}%
}
```

Input

The rest of the package code is as described in Example 40, including the definitions of \datefmt and \printdate. (You can download the complete package.) Here's an example document that uses this new package:

↑ Input

```
\documentclass{article}

\usepackage{mycustomdatetime}

\begin{document}

The file \jobname.tex was last modified on:
\filedate{\jobname.tex}.

The PDF was built by \TeX\␣on: \pdfnow.

Format a specific time (Zulu time):
\timefmt{8}{10}{35}{+0:00}.

Format a specific time (non-Zulu time):
\timefmt{8}{10}{35}{+1:00} or
\timefmt{8}{10}{35}{-4:30} or
```

```
\timefmt{8}{10}{35}{+5:45}.

Format a specific time without a time zone:
\timefmt{8}{10}{35}{}.

Format a specific time with a time zone but without seconds:
\timefmt{8}{10}{}{+00:00}.

Format a specific time without a time zone or seconds:
\timefmt{8}{10}{}{}.

\end{document}
```

↓ Input

You can download or view this document.

Exercise 22. Displaying Times

Add a command to the mycustomdatetime package described in Example 41 that has the syntax:

\printdatetime{⟨YYYY⟩-⟨MM⟩-⟨DD⟩ ⟨hh⟩:⟨mm⟩:⟨ss⟩⟨UTC offset⟩} Definition

This command should be equivalent to:

\printdate{⟨YYYY⟩-⟨MM⟩-⟨DD⟩} \timefmt{⟨hh⟩}{⟨mm⟩}{⟨ss⟩}{⟨UTC offset⟩} Input

Example usage:

↑ Input

```
\printdatetime{2014-03-25 01:23:15+00:00}
\printdatetime{2014-03-24 23:31:58+01:00}
\printdatetime{2014-03-24 16:28:56-06:00}
\printdatetime{2014-03-24 14:45:23+08:00}
\printdatetime{2014-03-25 03:12:04-04:30}
\printdatetime{2014-03-24 03:45:24-04:30}
\printdatetime{2014-03-24 21:20:24+05:45}
```

↓ Input

For the More Adventurous:
Suppose I now need all the times in a common time zone for easier comparison. Make a new command called, say, \printzuludatetime that has

the same syntax as \printdatetime but it converts the date and time to Zulu time (UTC+00:00) before displaying it.

You can download or view a solution. (The new datetime2 package [100] now comes with commands that perform a similar conversion in the accompanying datetime2-calc package.)

7.5 Displaying a Calendar

The pgfcalendar package provides the command:

\pgfcalendar{⟨*prefix*⟩}{⟨*start date*⟩}{⟨*end date*⟩}{⟨*code*⟩} Definition

This is a loop macro that iterates from ⟨*start date*⟩ to ⟨*end date*⟩ and performs ⟨*code*⟩ at each iteration. Within ⟨*code*⟩ you can access information about the current iteration using:

- \pgfcalendarcurrentjulian This is a TEX count register that holds the Julian day number for the current iteration;
- \pgfcalendarcurrentweekday The current week day index (0 for Monday, 1 for Tuesday, etc);
- \pgfcalendarcurrentyear The current year;

- \pgfcalendarcurrentmonth The current month (always two digits with a leading zero, if necessary);
- \pgfcalendarcurrentday The current day of the month.

In addition, within ⟨*code*⟩ you can also use:

- \pgfcalendarprefix The ⟨*prefix*⟩ parameter;
- \pgfcalendarbeginiso The ⟨*start date*⟩ in ISO format;
- \pgfcalendarbeginjulian The ⟨*start date*⟩ as a Julian day number;
- \pgfcalendarendiso The ⟨*end date*⟩ in ISO format;
- \pgfcalendarendjulian The ⟨*end date*⟩ as a Julian day number;
- \ifdate{⟨*tests*⟩}{⟨*true code*⟩}{⟨*false code*⟩}

  The same as using \pgfcalendarifdate for the current date.
- \pgfcalendarsuggestedname

  If ⟨*prefix*⟩ is empty, this expands to an empty string, otherwise it expands to ⟨*prefix*⟩-⟨*YYYY*⟩-⟨*MM*⟩-⟨*DD*⟩ so it can be used, for example, as a node name if the calendar is typeset using the tikz package's tikzpicture environment [101].

- \pgfcalendarshorthand{⟨*kind*⟩}{⟨*representation*⟩}

  This will expand to a representation of the current day, month, year or day of week, depending on whether ⟨*kind*⟩ is `d`, `m`, `y` or `w`. The ⟨*representation*⟩ may be one of:

  - `-` Numerical representation with no leading zeros;
  - `=` Numerical representation with a leading space for single digit numbers;
  - `0` Numerical representation with a leading zero for single digit numbers;
  - `t` Textual representation;
  - `.` Abbreviated textual representation.

  Typically you would use:

```
\let\%\pgfcalendarshorthand
```
Input

  before \pgfcalendar so that you can simply write, for example, \%wt instead of:

```
\pgfcalendarshorthand{w}{t}
```
Input

  but make sure you localise the effect of the \let by placing it inside a group or environment so that the normal behaviour of \% is

restored after the calendar has been typeset.

Examples:

1. To just display the day of the month from 2014-02-26 to 2014-03-15:

```
\pgfcalendar{}{2014-02-26}{2014-03-15}{%
\pgfcalendarcurrentday\␣}
```

Input

produces:

26 27 28 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15

Output

2. To display the date for each day from 2014-03-01 to 2014-03-04:

```
{% localise effect of \let
  \let\%\pgfcalendarshorthand
  \pgfcalendar{}{2014-03-01}{2014-03-04}{\%w.
\%d- \%mt \%y0\par}
}
```

↑ Input

↓ Input

produces:

↑ Output

Sat 1 March 2014

Sun 2 March 2014

Mon 3 March 2014

Tue 4 March 2014

↓ Output

The tikz package (part of the pgf bundle) provides a powerful and user-friendly way of drawing images. An in-depth discussion of the tikz package is beyond the scope of this book, but here's a very brief introduction to drawing nodes in a tikzpicture environment to help draw a simple calendar. For more detail about tikz, see the pgf user manual [101].

Within the tikzpicture environment, you can use

Definition

`\path[`⟨*path options*⟩`]` (⟨*position*⟩) node`[`⟨*node options*⟩`]` (⟨*node name*⟩) `{`⟨*text*⟩`}`;

to draw a node. Alternatively you can use

\node[⟨*node options*⟩] at (⟨*position*⟩) (⟨*node name*⟩) {⟨*text*⟩}; Definition

The full syntax is more complicated, but the (⟨*node name*⟩) is optional, as are the key=value lists ⟨*path options*⟩ and ⟨*node options*⟩. (Spaces before and after the commas and equal signs are ignored.) The full syntax of (⟨*position*⟩) is also quite complicated, but here I'll just use the (⟨x⟩,⟨y⟩) syntax.

Example:
(Don't forget to load the tikz package.)

↑ Input

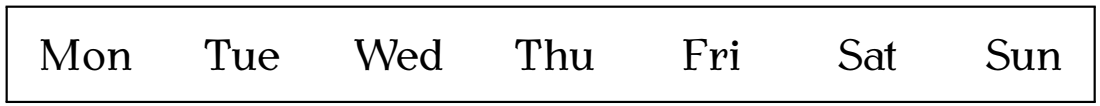
```
\fbox{%
 \begin{tikzpicture}
  \path (0,0) node {Mon};
  \path (1,0) node {Tue};
  \path (2,0) node {Wed};
  \path (3,0) node {Thu};
  \path (4,0) node {Fri};
  \path (5,0) node {Sat};
  \path (6,0) node {Sun};
 \end{tikzpicture}%
}
```

↓ Input

I've used the \fbox command (described in Volume 1 [92, §4.7.1]) to put a border around the picture. Fancier borders can be created using tikz commands within the tikzpicture environment.

The above code produces:

Output



It's possible to add a \pgfcalendar command to this environment and put the node drawing part in the ⟨*code*⟩ argument. Since \pgfcalendarcurrentweekday is an integer from 0 (Monday) to 6 (Sunday), it can be used for the ⟨x⟩ coordinate. Since tikz uses a right-handed coordinate system, the row below the ⟨y⟩ = 0 weekday name row displayed above needs to be negative. For example:

↑ Input

```
\fbox{%
 \begin{tikzpicture}
 % First row
  \path (0,0) node {Mon};
  \path (1,0) node {Tue};
```

```
  \path (2,0) node {Wed};
  \path (3,0) node {Thu};
  \path (4,0) node {Fri};
  \path (5,0) node {Sat};
  \path (6,0) node {Sun};
 % Second row
  \pgfcalendar{}{2014-03-01}{2014-03-02}{
   \path (\pgfcalendarcurrentweekday,-1)
     node {\pgfcalendarcurrentday};
  }
 \end{tikzpicture}%
}
```

↓ Input

This produces:

Mon	Tue	Wed	Thu	Fri	Sat	Sun
					01	02

Output

A counter is required if more than one week needs to be displayed. For example:

↑ Input

```
% Define a new count register:
\newcount\rowcount
% Initialise:
\mycount = 1\relax
% Draw the calendar:
\fbox{%
 \begin{tikzpicture}
  % header row
  \path (0,0) node {Mon};
  \path (1,0) node {Tue};
  \path (2,0) node {Wed};
  \path (3,0) node {Thu};
  \path (4,0) node {Fri};
  \path (5,0) node {Sat};
  \path (6,0) node {Sun};
  % Now iterate through the the month of March:
  \pgfcalendar{}{2014-03-01}{2014-03-31}
  {% Draw node for current day
   \path
     (\pgfcalendarcurrentweekday,-\rowcount) % coordinate
```

```
    node {\pgfcalendarcurrentday}; % node
   % Increment row count if today is a Sunday:
   \ifdate{Sunday}{\advance\rowcount by 1}{}
  }% end of loop
 \end{tikzpicture}%
}
```

Input

This now produces the image shown in Figure 7.1.

Nodes can have a border and background. These can be specified in the [⟨*node options*⟩]. For example:

```
\path (0,0) node[rectangle,draw] {Mon};
```

Input

will draw a rectangular border around the node while

```
\path (0,0) node[circle,fill=cyan] {Mon};
```

Input

will give the node a cyan circular background.

Example:
The above example can be modified to include borders and backgrounds:

Input

Output

Mon	Tue	Wed	Thu	Fri	Sat	Sun
					01	02
03	04	05	06	07	08	09
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Figure 7.1 Calendar (Days of March)

```
% Define a new count register:
\newcount\rowcount
% Initialise:
\mycount = 1\relax
% Draw the calendar:
\fbox{%
 \begin{tikzpicture}
  \path (0,0) node[circle,fill=yellow] {Mon};
  \path (1,0) node[circle,fill=yellow] {Tue};
  \path (2,0) node[circle,fill=yellow] {Wed};
  \path (3,0) node[circle,fill=yellow] {Thu};
  \path (4,0) node[circle,fill=yellow] {Fri};
  \path (5,0) node[circle,fill=cyan] {Sat};
  \path (6,0) node[circle,fill=cyan] {Sun};
  \pgfcalendar{}{2014-03-01}{2014-03-31}{
   % Draw node for current day
   \path (\pgfcalendarcurrentweekday,-\rowcount)
     node[rectangle,draw] {\pgfcalendarcurrentday};
   % Increment row count if today is a Sunday:
   \ifdate{Sunday}{\advance\rowcount by 1}{}
  }
 \end{tikzpicture}%
```

```
}
```

↓ Input

Instead of repeatedly using the same options it's possible to set them within a local scope using the scope environment:

↑ Input

```
% Define a new count register:
\newcount\rowcount
% Initialise:
\mycount = 1\relax
% Draw the calendar:
\fbox{%
 \begin{tikzpicture}
  \begin{scope}[every node/.style={circle,fill=yellow}]
    \path (0,0) node {Mon};
    \path (1,0) node {Tue};
    \path (2,0) node {Wed};
    \path (3,0) node {Thu};
    \path (4,0) node {Fri};
    \path (5,0) node[fill=cyan] {Sat};
    \path (6,0) node[fill=cyan] {Sun};
```

```
  \end{scope}
  \pgfcalendar{}{2014-03-01}{2014-03-31}{
   % Draw node for current day
   \path (\pgfcalendarcurrentweekday,-\rowcount)
     node[rectangle,draw] {\pgfcalendarcurrentday};
   % Increment row count if today is a Sunday:
   \ifdate{Sunday}{\advance\rowcount by 1}{}
  }
 \end{tikzpicture}%
}
```

Input

This produces the image shown in Figure 7.2. The nodes in the first row look a little uneven as the sizes vary according to the node contents. To neaten things up a bit, a minimum size can be imposed on the nodes:

Input

```
% Define a new count register:
\newcount\rowcount
% Initialise:
\mycount = 1\relax
% Draw the calendar:
```

Output

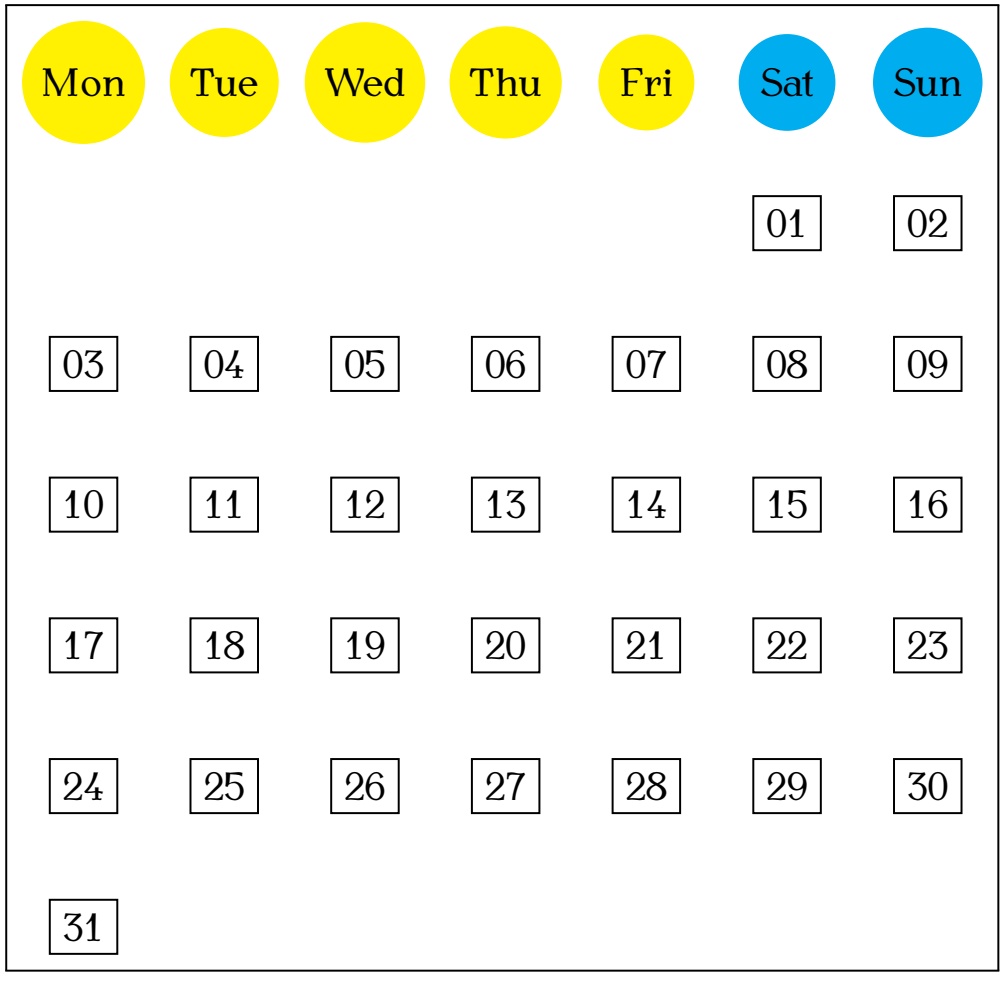


Figure 7.2 Calendar (Node Shapes Added)

```
\fbox{%
 \begin{tikzpicture}
  \begin{scope}[every node/.style={circle,fill=yellow,
minimum size=3em}]
    \path (0,0) node {Mon};
    \path (1,0) node {Tue};
    \path (2,0) node {Wed};
    \path (3,0) node {Thu};
    \path (4,0) node {Fri};
    \path (5,0) node[fill=cyan] {Sat};
    \path (6,0) node[fill=cyan] {Sun};
  \end{scope}
  \pgfcalendar{}{2014-03-01}{2014-03-31}{
   % Draw node for current day
   \path (\pgfcalendarcurrentweekday,-\rowcount)
     node[rectangle,draw] {\pgfcalendarcurrentday};
   % Increment row count if today is a Sunday:
   \ifdate{Sunday}{\advance\rowcount by 1}{}
  }
 \end{tikzpicture}%
}
```

↓ Input

This produces the image shown in Figure 7.3.

However now the nodes are bumping into each other, so they need to be moved apart. The default $\langle x \rangle$ and $\langle y \rangle$ coordinate units are 1 cm. This can be changed in the optional argument of the tikzpicture environment. For example:

↑ Input

```
% Define a new count register:
\newcount\rowcount
% Initialise:
\mycount = 1\relax
% Draw the calendar:
\fbox{%
 \begin{tikzpicture}[x=1.5cm,y=1.25cm]
  \begin{scope}[every node/.style={circle,fill=yellow,
minimum size=3em}]
    \path (0,0) node {Mon};
    \path (1,0) node {Tue};
    \path (2,0) node {Wed};
    \path (3,0) node {Thu};
    \path (4,0) node {Fri};
    \path (5,0) node[fill=cyan] {Sat};
```

Output

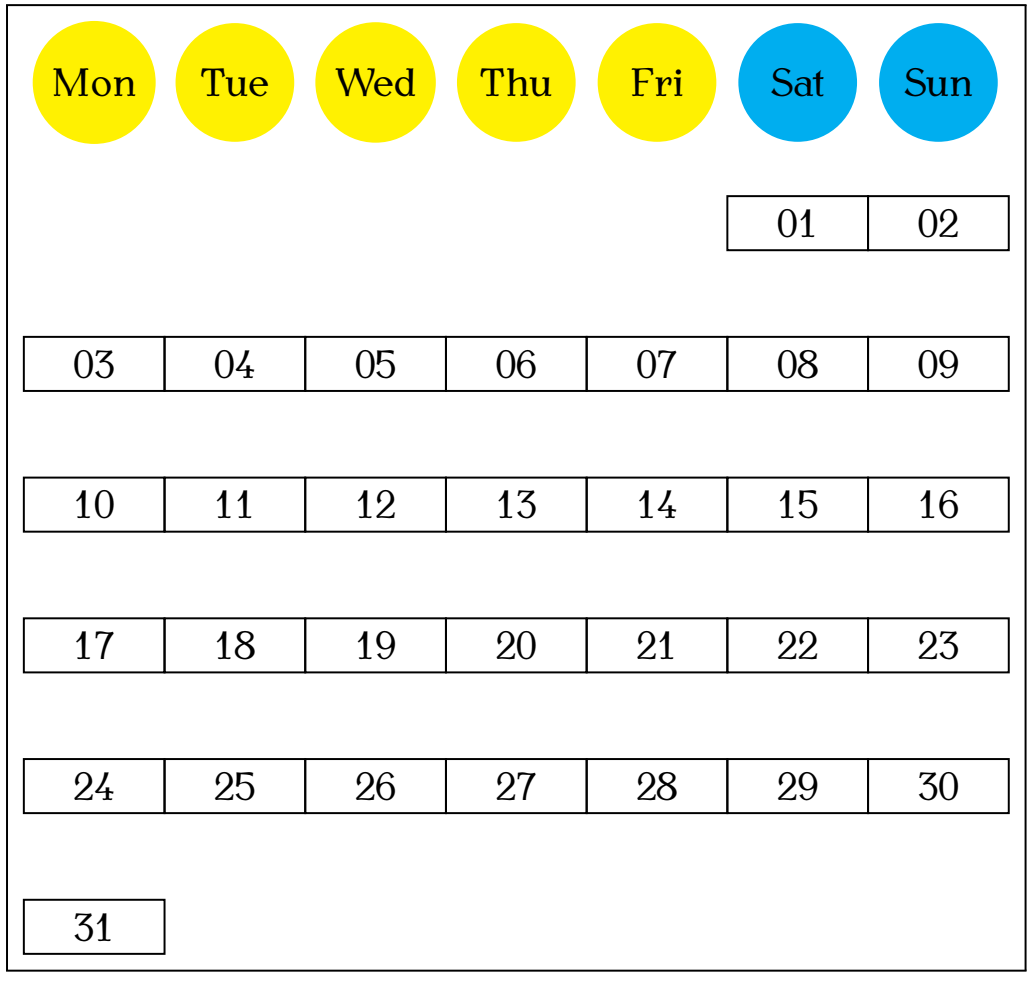


Figure 7.3 Calendar (Minimum Width Set on Nodes)

```
    \path (6,0) node[fill=cyan] {Sun};
  \end{scope}
  \pgfcalendar{}{2014-03-01}{2014-03-31}{
   % Draw node for current day
   \path (\pgfcalendarcurrentweekday,-\rowcount)
     node[rectangle,draw,minimum width=1cm]
{\pgfcalendarcurrentday};
   % Increment row count if today is a Sunday:
   \ifdate{Sunday}{\advance\rowcount by 1}{}
  }
 \end{tikzpicture}%
}
```

↓ Input

This produces the calendar shown in Figure 7.4.

The `circle` and `rectangle` shapes are always available, but there are other shapes as well that can be loaded via the relevant tikz library, which can be loaded in the preamble using:

`\usetikzlibrary{`⟨*name*⟩`}` Definition

where ⟨*name*⟩ is the library name. For example, there are some multipart shapes defined in the `shapes.multipart` library. In order to use these shapes, you not only need

Output

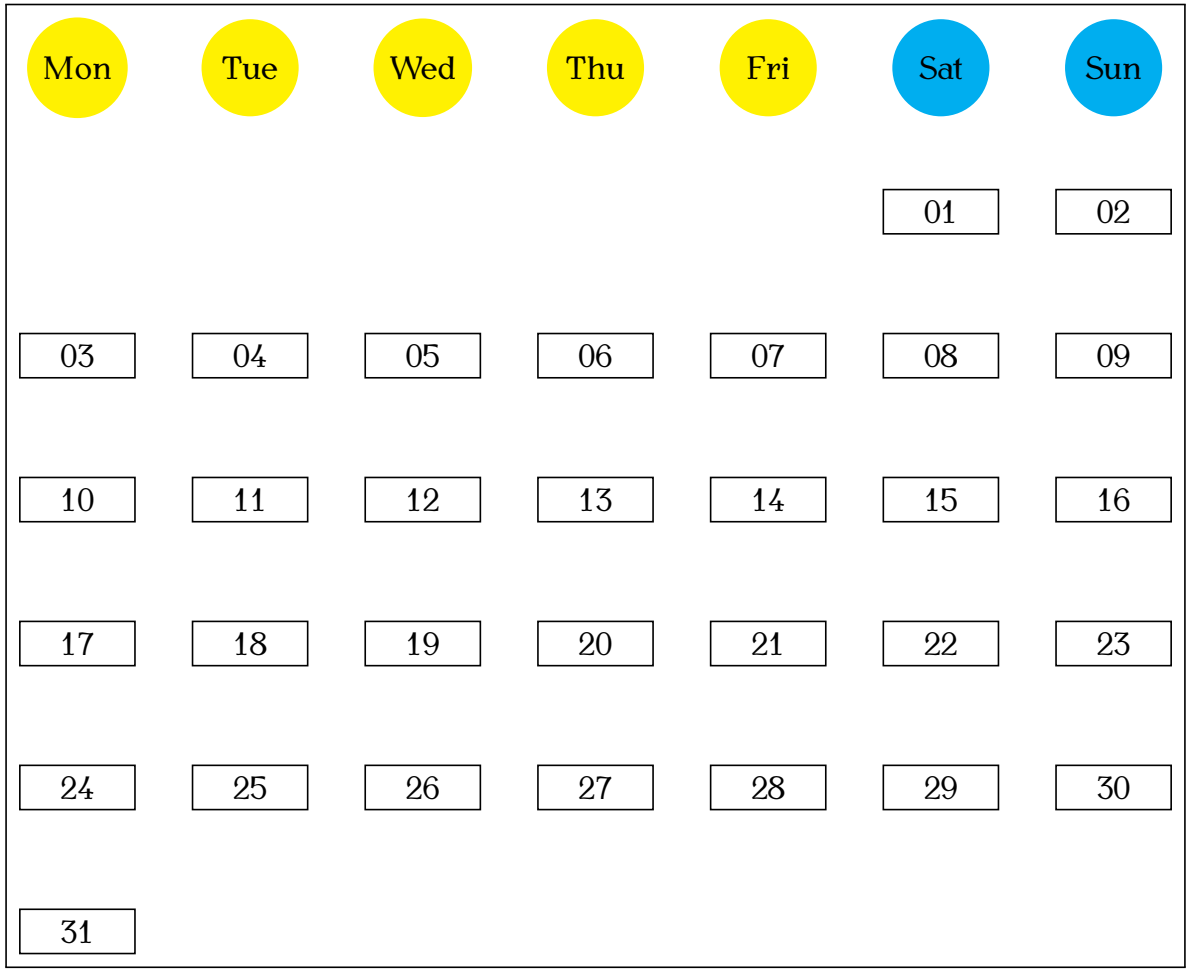


Figure 7.4 Calendar Using Circular and Rectangular Nodes (March 2014)

```
\usepackage{tikz}
```
Input

in the preamble but also

```
\usetikzlibrary{shapes.multipart}
```
Input

Example:
A rectangular split node with 2 splits can be created using:

Input

```
\begin{tikzpicture}
 \path (0,0)
 node[rectangle split,rectangle split parts=2,draw]
 {%
   Top
   \nodepart{two}
   Bottom
 };
\end{tikzpicture}
```

Input

The \nodepart{⟨part⟩} command moves from the current split part to the split part identified by ⟨part⟩. In the case of a rectangular split node, the second part is identified by the keyword two. The above code produces:

Output

With a vertical split node, such as in the above example, you can set a minimum width using the minimum width key, but you can't specify a minimum height. You can, however, specify a height for empty parts using the option rectangle split empty part height=⟨length⟩. For example:

Input

```
\begin{tikzpicture}
 \path (0,0)
 node
 [
   rectangle split,
   rectangle split parts=2,
   rectangle split empty part height=1cm,
   minimum width=2cm,
   draw
```

```
 ]
 {%
   Top
   \nodepart{two}
   % empty bottom part
 };
\end{tikzpicture}
```

↓ Input

This produces:

Output

You can specify fill colours for each part using the `rectangle split part fill={`⟨*colour list*⟩`}` option, where ⟨*colour list*⟩ is a comma-separated list of colours for each part, in order. For example:

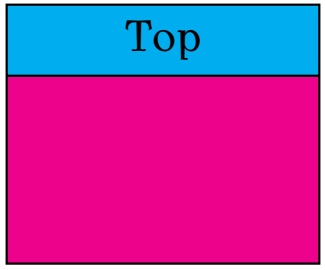
↑ Input

```
\begin{tikzpicture}
```

```
 \path (0,0)
 node
 [
   rectangle split,
   rectangle split parts=2,
   rectangle split empty part height=1cm,
   rectangle split part fill={cyan,magenta},
   minimum width=2cm,
   draw
 ]
 {%
   Top
   \nodepart{two}
   % empty bottom part
 };
\end{tikzpicture}
```

↓ Input

This produces:



Output

Since tikz loads the xcolor package [40], you can specify colours using the xcolor syntax. For example:

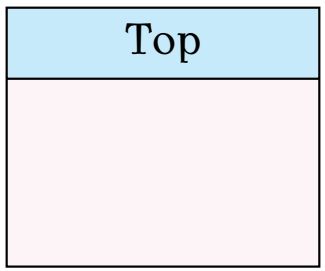
Input

```
\begin{tikzpicture}
 \path (0,0)
 node
 [
   rectangle split,
   rectangle split parts=2,
   rectangle split empty part height=1cm,
   rectangle split part fill={cyan!20,magenta!5},
   minimum width=2cm,
   draw
 ]
 {%
```

```
    Top
    \nodepart{two}
    % empty bottom part
  };
\end{tikzpicture}
```

↓ Input

This produces:



Output

Now the fill colour for the top part is 20% cyan tint and the fill colour for the bottom part is 5% magenta tint. This isn't a great colour scheme, but it's just used for illustrative purposes.

Example 42. Calendar for May 2014

The above can be put together to create a calendar for the month of May 2014:

↑ Input

```
\newcount\rowcount
\rowcount=1\relax

\fbox{%
\let\%\pgfcalendarshorthand
\begin{tikzpicture}[x=1.5cm,y=1.75cm]
  \begin{scope}
    [every node/.style={rectangle,fill=green!5,minimum width=1.4cm}]
  \path (0,0) node {Mon};
  \path (1,0) node {Tue};
  \path (2,0) node {Wed};
  \path (3,0) node {Thu};
  \path (4,0) node {Fri};
  \path (5,0) node {Sat};
  \path (6,0) node {Sun};
  \end{scope}
  \pgfcalendar{}{2014-05-01}{2014-05-31}
  {
   \path (\pgfcalendarcurrentweekday,-\rowcount)
    node
    [
      rectangle split,
```

```
      minimum width=1.4cm,
      rectangle split empty part height=1cm,
      rectangle split parts=2,
      rectangle split part fill={cyan!20,magenta!4},
      draw]
   {\%d-
    \nodepart{two}
   };
   \ifdate{Sunday}{\advance\rowcount by 1}{}
 }
\end{tikzpicture}%
}
```

↓ Input

This produces the calendar shown in Figure 7.5.

Suppose now I want to add some information to the calendar. For example, the two May bank holidays on the 5th and 26th of May. Additionally, suppose I also want a different colour background for weekends and bank holidays, for example, a light grey. The bank holiday information can be stored in control sequences whose names are in the format ⟨*prefix*⟩-⟨*YYYY*⟩-⟨*MM*⟩-⟨*DD*⟩, which is the format used by `\pgfcalendarsuggestedname`. Recall etoolbox's `\csdef` command described in §2.1.1. This can be used to define these control sequences:

Output

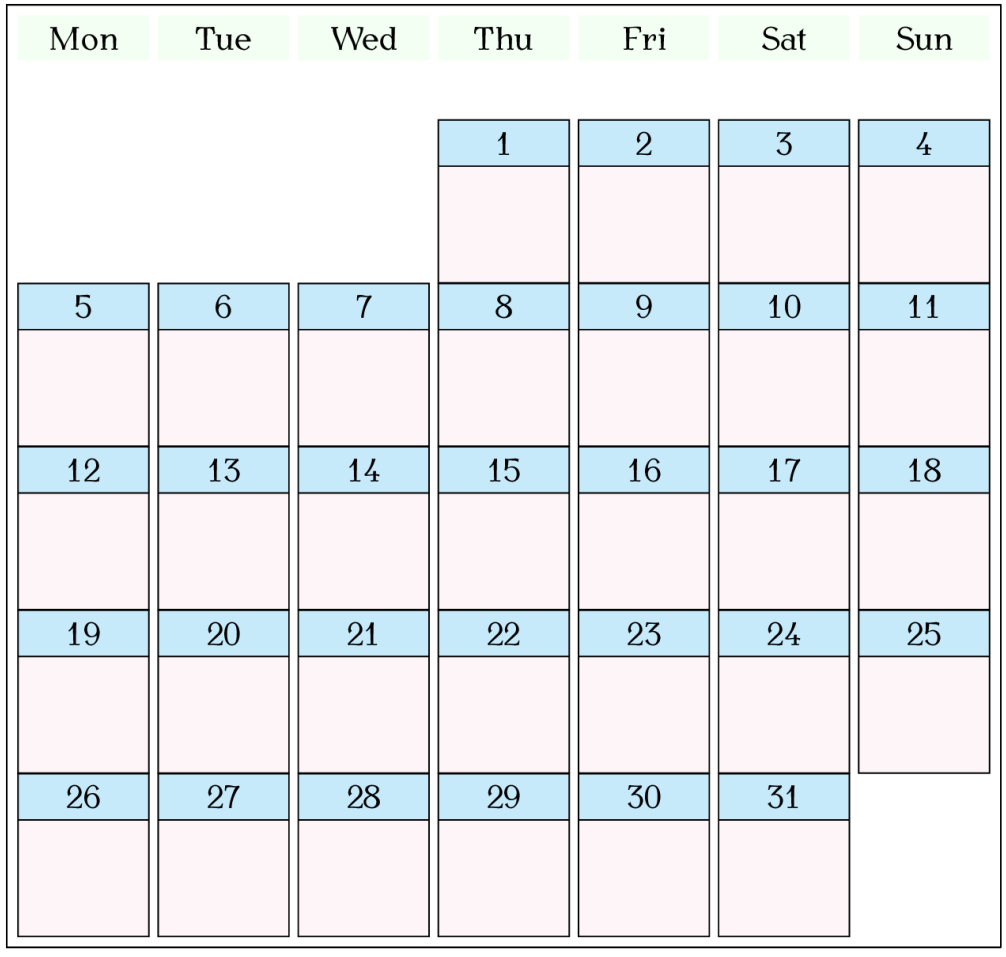


Figure 7.5 Calendar with Split Nodes (May 2014)

Input

```
\csdef{cal-2014-05-05}{Early May BH}
\csdef{cal-2014-05-26}{Spring BH}
```

Input

The \pgfcalendar command now needs cal as the ⟨*prefix*⟩ so that in the ⟨*code*⟩ part, the current day can be checked if it's a bank holiday using:

Input

```
\ifcsdef{\pgfcalendarsuggestedname}%
{%
  % Current day is a~bank holiday
}%
{%
  % Current day isn't a~bank holiday
}
```

Input

As before a TEX register called \rowcount is defined using:

```
\newcount\rowcount
```

Input

The actual code to generate the calendar is now:

↑ Input

```
\fbox{%
\rowcount=1\relax
\let\%\pgfcalendarshorthand
\begin{tikzpicture}[x=1.5cm,y=1.75cm]
  \begin{scope}
    [every node/.style={rectangle,fill=green!5,minimum width=1.4cm}]
  \path (0,0) node {Mon};
  \path (1,0) node {Tue};
  \path (2,0) node {Wed};
  \path (3,0) node {Thu};
  \path (4,0) node {Fri};
  \path (5,0) node {Sat};
  \path (6,0) node {Sun};
  \end{scope}
  \pgfcalendar{cal}{2014-05-01}{2014-05-31}{%
   \def\thebackground{magenta!4}%
   \ifcsdef{\pgfcalendarsuggestedname}%
```

```
    {%
      \def\thecontents{\csuse{\pgfcalendarsuggestedname}}%
      \def\thebackground{black!4}%
    }%
    {%
      \def\thecontents{\mbox{}}%
      \ifdate{weekend}{\def\thebackground{black!4}}{}%
    }%
    \path (\pgfcalendarcurrentweekday,-\rowcount)
     node
     [
       rectangle split,
       rectangle split parts=2,
       rectangle split part fill={cyan!20,\thebackground},
       draw]
    {\%d-
     \nodepart{two}%
     \parbox[t][1cm]{1.2cm}{\small\thecontents}%
    };
    \ifdate{Sunday}{\advance\rowcount by 1}{}%
  }%
\end{tikzpicture}%
```

```
}
```

↓ Input

This produces the calendar shown in Figure 7.6.

Suppose now you want to fill in the gaps at the beginning and end of the month. Recall the \foreach command mentioned in §2.7.2. This has the syntax:

\foreach ⟨*variables*⟩ [⟨*options*⟩] in {⟨*list*⟩}{⟨*body*⟩}

Definition

but it's cleverer than the other list macros described in that section as you can use ... within ⟨*list*⟩ if the list contents can be inferred from the beginning and end of the list. For example:

```
\foreach \x in {1,...,10} {\x\space}
```

Input

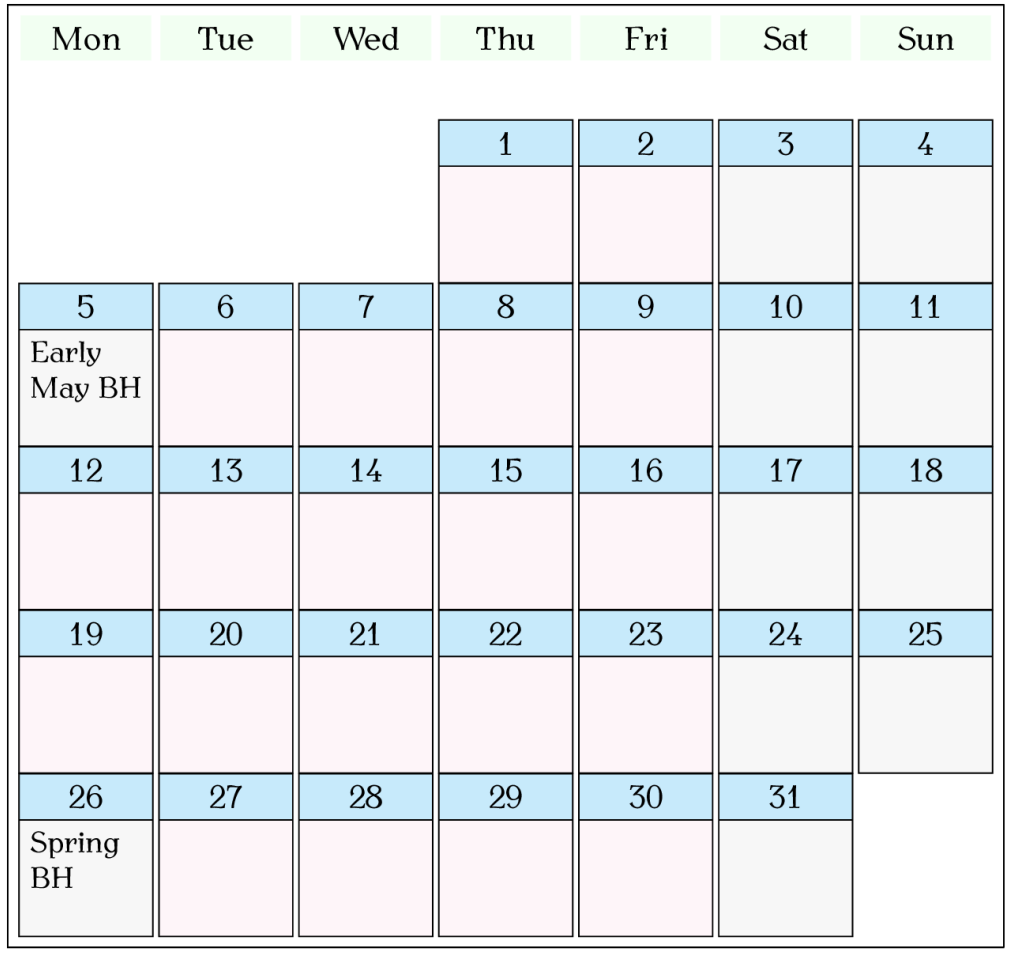
produces

1 2 3 4 5 6 7 8 9 10

Output

Therefore, within the ⟨*code*⟩ part of \pgfcalendar, you can test if the current day is the first day of the month (by testing that \pgfcalendarcurrentday is equal to 1) and use \foreach to fill in the last days of the previous month:

Output



Mon	Tue	Wed	Thu	Fri	Sat	Sun
			1	2	3	4
5 Early May BH	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26 Spring BH	27	28	29	30	31	

Figure 7.6 May 2014 with Bank Holidays

↑ Input

```
\ifnum\pgfcalendarcurrentday=1\relax
% Fill in days from previous month if this isn't a Monday
  \ifdate{Monday}{}
  {% Get last day of previous month
    \julianday = \pgfcalendarcurrentjulian\relax
    \advance\julianday by -\pgfcalendarcurrentweekday\relax
    \foreach \x in {0,...,\numexpr\pgfcalendarcurrentweekday-1}
    {
      \pgfcalendarjuliantodate{\julianday}{\theyear}{\themonth}{\theday}
      \path (\x,-1)
       node
       [
         rectangle split,
         rectangle split parts=2,
         draw]
      {\number\theday
       \nodepart{two}
       \parbox[t][1cm]{1.2cm}{\mbox{}}%
      };
      \global\advance\julianday by 1\relax
```

```
    }
  }
\fi
```

Input

This requires a new register:

```
\newcount\julianday
```

Input

The tikz package automatically loads the pgffor package, so the \foreach command will also be available if you use tikz. Note that \foreach uses a local scope for each iteration which is why \global is required when incrementing the \julianday register. The gap at the end of the final week can also be filled with the initial days of the next month, but as with \foreach, \pgfcalendar scopes each iteration, so the row register \rowcount will need to be incremented globally so that it can be used after the loop has completed. It's also useful to store the Julian day number and the week day number for the last day of the month so they can be accessed outside the loop. This saves the need to compute them again. So the last part of ⟨*code*⟩ needs to replace:

```
\ifdate{Sunday}{\advance\rowcount by 1}{}%
```

Input

with

Input

```
\ifdate{Sunday}{\global\advance\rowcount by 1}{}%
\xdef\lastjulianday{\number\pgfcalendarcurrentjulian}
\xdef\lastweekday{\number\pgfcalendarcurrentweekday}
```

Input

Now the remaining days of the last row can be completed outside the \pgfcalendar loop:

Input

```
\ifnum\lastweekday < 6\relax
   \julianday = \lastjulianday\relax
   \edef\lastweekday{\number\numexpr\lastweekday+1}
   \foreach \x in {\lastweekday,...,6}
   {
     \global\advance\julianday by 1\relax
     \pgfcalendarjuliantodate{\julianday}{\theyear}{\themonth}{\theday}
     \path (\x,-\rowcount)
      node
      [
        rectangle split,
```

```
        rectangle split parts=2,
        draw]
      {\number\theday
       \nodepart{two}
       \parbox[t]{1cm}{1.2cm}{\mbox{}}%
      };
   }
 \fi
```

Input

Note that you can also use \foreach to display the week day nodes:

Input

```
\foreach \x in {0,...,6}
   {\path (\x,0) node {\pgfcalendarweekdayshortname{\x}};}
```

Input

The complete code is:

Input

```
\newcount\rowcount
\newcount\julianday
```

```
\fbox{%
\rowcount=1\relax
\let\%\pgfcalendarshorthand
\begin{tikzpicture}[x=1.5cm,y=1.75cm]
  \begin{scope}
    [every node/.style={rectangle,fill=green!5,minimum width=1.4cm}]
    \foreach \x in {0,...,6}
     {\path (\x,0) node {\pgfcalendarweekdayshortname{\x}};}
  \end{scope}
  \pgfcalendar{cal}{2014-05-01}{2014-05-31}
  {% Is this the first day of the month?
    \ifnum\pgfcalendarcurrentday=1\relax
    % Fill in days from previous month if this isn't a Monday
    \ifdate{Monday}{}
    {% Get last day of previous month
     \julianday = \pgfcalendarcurrentjulian\relax
     \advance\julianday by -\pgfcalendarcurrentweekday\relax
     \foreach \x in {0,...,\numexpr\pgfcalendarcurrentweekday-1}
     {
      \pgfcalendarjuliantodate{\julianday}{\theyear}{\themonth}{\theday}
      \path (\x,-1)
        node
```

```
      [rectangle split,
       rectangle split parts=2,
       draw]
      {\number\theday
       \nodepart{two}
       \parbox[t][1cm]{1.2cm}{\mbox{}}%
      };
    \global\advance\julianday by 1\relax
   }
  }
  \fi
  \def\thebackground{magenta!4}%
  \ifcsdef{\pgfcalendarsuggestedname}%
  {%
    \def\thecontents{\csuse{\pgfcalendarsuggestedname}}%
    \def\thebackground{black!4}%
  }%
  {%
    \def\thecontents{\mbox{}}%
    \ifdate{weekend}{\def\thebackground{black!4}}{}%
  }%
  \path (\pgfcalendarcurrentweekday,-\rowcount)
```

```
    node
    [rectangle split,
     rectangle split parts=2,
     rectangle split part fill={cyan!20,\thebackground},
     draw]
   {\%d-
    \nodepart{two}%
    \parbox[t][1cm]{1.2cm}{\small\thecontents}%
   };
   \ifdate{Sunday}{\global\advance\rowcount by 1}{}%
   \xdef\lastjulianday{\number\pgfcalendarcurrentjulian}
   \xdef\lastweekday{\number\pgfcalendarcurrentweekday}
 }%
 \ifnum\lastweekday < 6\relax
  \julianday = \lastjulianday\relax
  \edef\lastweekday{\number\numexpr\lastweekday+1}
  \foreach \x in {\lastweekday,...,6}
  {
    \global\advance\julianday by 1\relax
    \pgfcalendarjuliantodate{\julianday}{\theyear}{\themonth}{\theday}
    \path (\x,-\rowcount)
     node
```

```
      [rectangle split,
       rectangle split parts=2,
       draw]
     {\number\theday
      \nodepart{two}
      \parbox[t]{1cm}{1.2cm}{\mbox{}}%
     };
   }
  \fi
\end{tikzpicture}%
}
```

↓ Input

The result is shown in Figure 7.7. You can download or view a complete document.

It's possible to create a general month calendar macro from the above. First a command that can be used to set information for a given date. This uses \appto so that information can be appended to a date.

↑ Input

Output

Mon	Tue	Wed	Thu	Fri	Sat	Sun
28	29	30	1	2	3	4
5 Early May BH	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26 Spring BH	27	28	29	30	31	1

Figure 7.7 May 2014 with Bank Holidays (including end of previous month and beginning of next month)

```
\newcommand*{\addevent}[2]{%
  \ifcsdef{cal-#1}
  {% already defined so append info
    \csappto{cal-#1}{\newline #2}%
  }%
  {% not defined
    \csdef{cal-#1}{#2}%
  }
}
```

↓ Input

This has the syntax:

\addevent{⟨*date*⟩}{⟨*information*⟩} Definition

Now the definition for the calendar month macro:

↑ Input

```
\newcommand*{\calendarmonth}[2]{%
  \fbox{%
  \rowcount=1\relax
  \let\%\pgfcalendarshorthand
  \begin{tikzpicture}[x=1.5cm,y=1.75cm]
```

```
% display the month name at the top
\path (3,1) node {\pgfcalendarmonthname{#2}};
\begin{scope}
  [every node/.style={rectangle,fill=green!5,minimum width=1.4cm}]
  \foreach \x in {0,...,6}
   {\path (\x,0) node {\pgfcalendarweekdayshortname{\x}};}
\end{scope}
\pgfcalendar{cal}{#1-#2-01}{#1-#2-last}
{%
  % Is this the first day of the month?
  \ifnum\pgfcalendarcurrentday=1\relax
  % Fill in days from previous month if this isn't a Monday
  \ifdate{Monday}{}
  {
    % Get last day of previous month
    \julianday = \pgfcalendarcurrentjulian\relax
    \advance\julianday by -\pgfcalendarcurrentweekday\relax
    \foreach \x in {0,...,\numexpr\pgfcalendarcurrentweekday-1}
    {
      \pgfcalendarjuliantodate
        {\julianday}{\theyear}{\themonth}{\theday}
      \path (\x,-1)
```

```
      node
      [
        rectangle split,
        rectangle split parts=2,
        draw]
      {\number\theday
       \nodepart{two}
       \parbox[t][1cm]{1.2cm}{\mbox{}}%
      };
      \global\advance\julianday by 1\relax
    }
  }
  \fi
  \def\thebackground{magenta!4}%
  \ifcsdef{\pgfcalendarsuggestedname}%
  {%
    \def\thecontents{\csuse{\pgfcalendarsuggestedname}}%
    \def\thebackground{black!4}%
  }%
  {%
    \def\thecontents{\mbox{}}%
    \ifdate{weekend}{\def\thebackground{black!4}}{}%
```

```
    }%
    \path (\pgfcalendarcurrentweekday,-\rowcount)
     node
     [
       rectangle split,
       rectangle split parts=2,
       rectangle split part fill={cyan!20,\thebackground},
       draw]
    {\%d-
     \nodepart{two}%
     \parbox[t][1cm]{1.2cm}{\small\thecontents}%
    };
    \ifdate{Sunday}{\global\advance\rowcount by 1}{}%
    \xdef\lastjulianday{\number\pgfcalendarcurrentjulian}
    \xdef\lastweekday{\number\pgfcalendarcurrentweekday}
  }%
  \ifnum\lastweekday < 6\relax
   \julianday = \lastjulianday\relax
   \edef\lastweekday{\number\numexpr\lastweekday+1}
   \foreach \x in {\lastweekday,...,6}
   {
     \global\advance\julianday by 1\relax
```

```
      \pgfcalendarjuliantodate{\julianday}{\theyear}{\themonth}{\theday}
      \path (\x,-\rowcount)
       node
       [
         rectangle split,
         rectangle split parts=2,
         draw]
      {\number\theday
       \nodepart{two}
       \parbox[t][1cm]{1.2cm}{\mbox{}}%
      };
    }
   \fi
  \end{tikzpicture}%
  }
}
```

↓ Input

The syntax for this macro is:

\calendarmonth{⟨*YYYY*⟩}{⟨*MM*⟩} Definition

Don't forget you also need to define the registers:

Input

```
\newcount\rowcount
\newcount\julianday
```

Input

Exercise 23. Calendar for 2014
Create a landscape document that has a calender month per page for 2014 (or the year of your choice). Read the tikz chapter of the pgf manual [101] to find ways of modifying the above code. You can download or view a solution.

8. Presentations (The beamer Class)

[FAQ: Producing presentations (including slides)]

There are a number of classes listed on the presentation topic page for presentations. For brevity, this book will only cover the beamer class [102]. At the time of writing, the beamer manual is over 200 pages long. This book is already larger than the previous two volumes combined, so this chapter will only look at how to create a basic document using beamer to help you get started.

A document that uses the beamer class typically consists of a series of frame environments. Each frame produces a slide, or possibly several slides if there are overlays. Here's a simple document that just creates a title slide:

↑ Input

```
\documentclass{beamer}

\title{Culinary Experimental Research}
\author{Mabel Canary}
```

```
\date{22nd March 2014}

\begin{document}
  \begin{frame}
    \maketitle
  \end{frame}
\end{document}
```

Input

The beamer class provides an optional argument to \title, \author and \date that isn't available with standard classes, such as article. The optional argument can be used to supply an abbreviated version which may be used in headlines or footlines. There are additional title page commands:

\subtitle[⟨*short title*⟩]{⟨*title*⟩} Definition

This specifies a subtitle.

\titlegraphic{⟨*graphic*⟩} Definition

This specifies graphics for the title page. Typically, ⟨*graphic*⟩ is code to load an image file.

\institute[⟨*short name*⟩]{⟨*name*⟩} Definition

This specifies the author's affiliation. If there are multiple authors from different institutes, the institutes should be separated by \and (in a similar manner to \author). Additionally, when there are multiple institutes, each institute should be prefixed by

\inst{⟨*text*⟩} Definition

This command should also be placed after the corresponding name (or names) in the argument of \author with matching ⟨*text*⟩.

Example:

↑ Input

```
\documentclass{beamer}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\title{Culinary Experimental Research}
\subtitle{Mind-Controlling Cookies and Exploding Chocolates}
\author[Canary and Zebra]{Mabel Canary\inst{1} \and
Zöe Zebra\inst{2}}
\date[Mar'14]{22nd March 2014}
```

```
\institute{\inst{1}Secret Lab of Experimental Stuff\\
  University of Somewhere\and
  \inst{2}Department of Stripy Confectioners\\
  College of Somewhere Else
}
\titlegraphic{\includegraphics[width=1in]{dummy-logo}}

\begin{document}
  \begin{frame}
    \maketitle
  \end{frame}
\end{document}
```

↓ Input

This uses the sample dummy-logo.png file available from the examples page. The resulting slide is shown in Figure 8.1. The footline (bottom right of the slide) provides navigation links.

Within the frame environment you can use:

\frametitle{⟨*title*⟩} Definition

to specify the title of the frame and

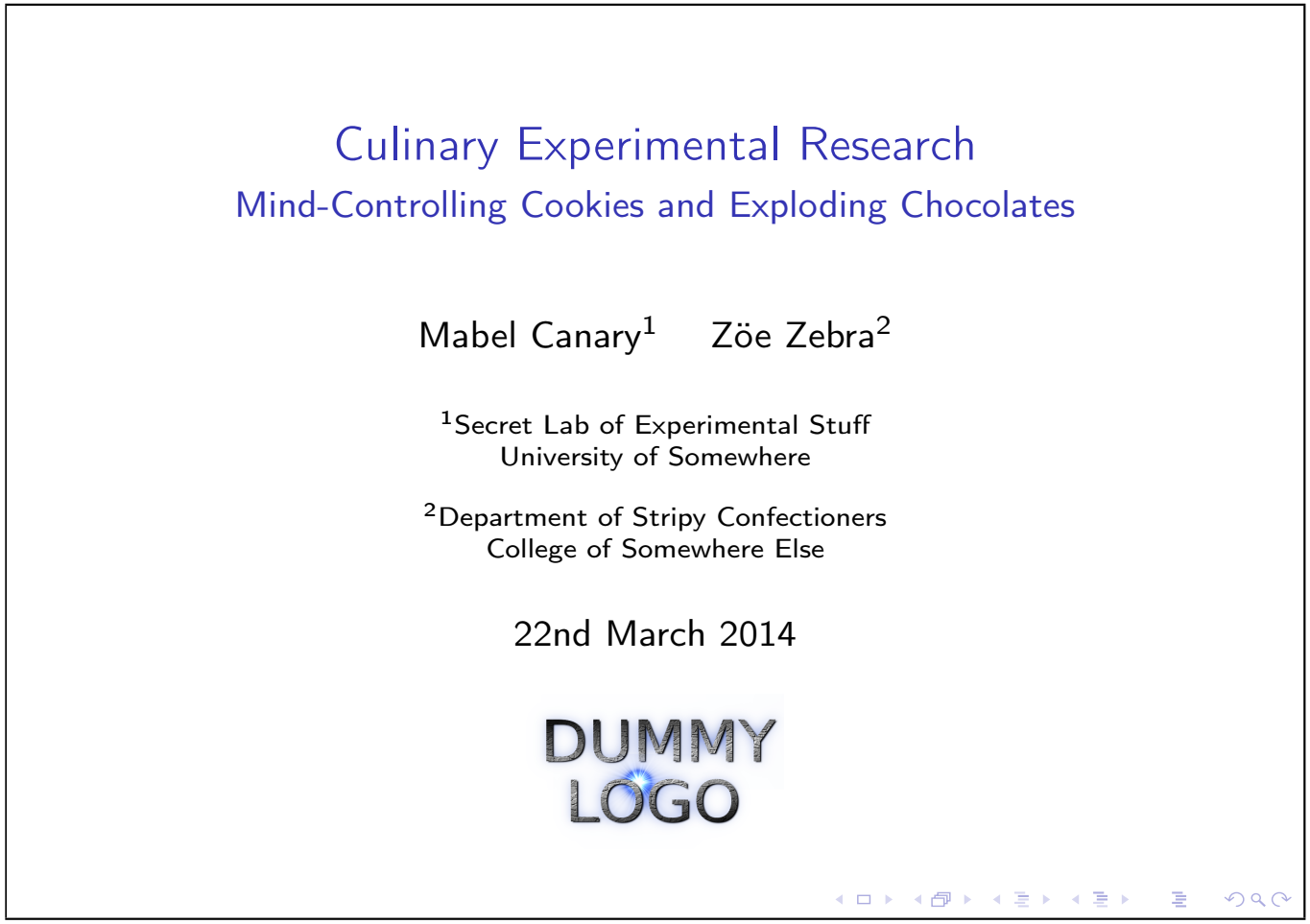


Figure 8.1 Title Frame

\framesubtitle{⟨*subtitle*⟩} Definition

to specify the subtitle.

Example:

↑ Input

```
\begin{frame}
  \frametitle{Mind-Controlling Cookies}
  \framesubtitle{Ingredients}

  \begin{itemize}
  \item Self-raising flour;
  \item Butter;
  \item Chocolate chips;
  \item Sugar obtained from secret genetically modified beet.
  \end{itemize}
\end{frame}
```

↓ Input

The resulting frame is shown in Figure 8.2.

⚠ The contents of the frame environment are fragile. If you want verbatim text in a frame (for example, using \verb or the verbatim or lstlisting

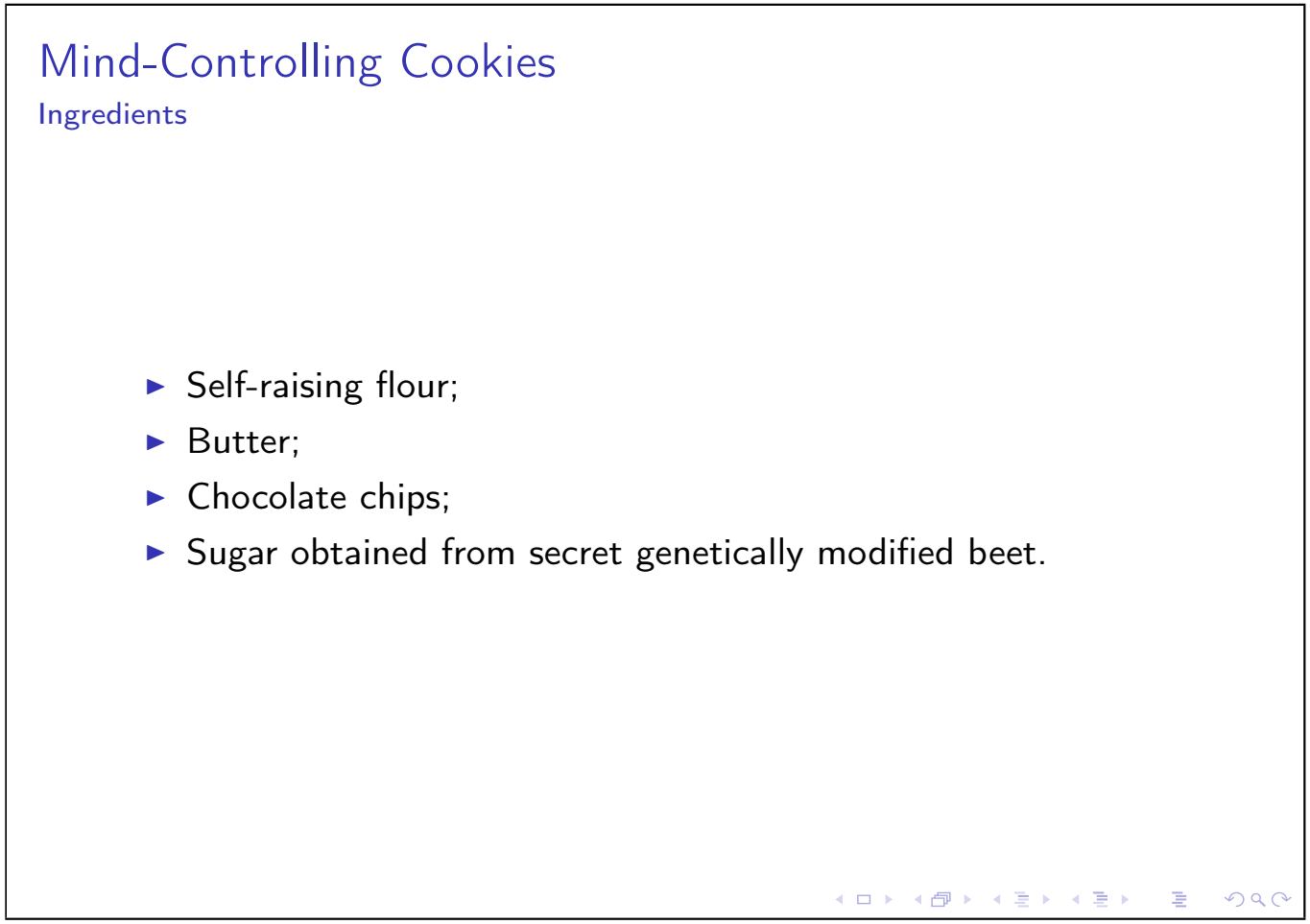


Figure 8.2 An Example Frame

environments) you must use the `fragile` option to the frame environment. For example:

Input

```
\begin{frame}[fragile]
\frametitle{Hello World!}
\begin{verbatim}
#!/usr/bin/perl
print "Hello World!\n";
1;
\end{verbatim}
\end{frame}
```

Input

The resulting frame is shown in Figure 8.3.

The standard sectioning commands may be used outside the frame environment. These will be added to the PDF bookmarks and you can add a table of contents frame by putting `\tableofcontents` inside a frame environment.

Example:

Figure 8.3 Verbatim in a Frame

↑ Input

```
\begin{frame}
  \tableofcontents
\end{frame}

\section{Experimental Research}
\subsection{Cookies}

\begin{frame}
  \frametitle{Mind-Controlling Cookies}
  \framesubtitle{Ingredients}

  \begin{itemize}
  \item Self-raising flour;
  \item Butter;
  \item Chocolate chips;
  \item Sugar obtained from secret genetically modified beet.
  \end{itemize}
\end{frame}
```

↓ Input

Areas of a frame can be divided into titled blocks using the block environment.

`\begin{block}{`⟨*title*⟩`}` Definition

Example:

↑ Input

```
\begin{frame}
  \frametitle{Mind-Controlling Cookies}
  \framesubtitle{Recipe}

  \begin{block}{Ingredients}
    \begin{itemize}
    \item Self-raising flour;
    \item Butter;
    \item Chocolate chips;
    \item Sugar obtained from secret genetically modified beet.
    \end{itemize}
  \end{block}
\end{frame}
```

↓ Input

The resulting frame is shown in Figure 8.4.

There are some predefined block environments, such as theorem and proof.

`\begin{theorem}[⟨title⟩]` Definition

If ⟨*title*⟩ is present, this is appended to the "Theorem" block title.

`\begin{proof}[⟨proof name⟩]` Definition

If ⟨*proof name*⟩ is present, it's used instead of "Proof" as the block title.

8.1 Overlays

Overlays allow you to uncover parts of a slide. For example, to uncover the items in the above itemize environment:

↑ Input

```
\begin{frame}
  \frametitle{Mind-Controlling Cookies}
  \framesubtitle{Recipe}

  \begin{block}{Ingredients}
```

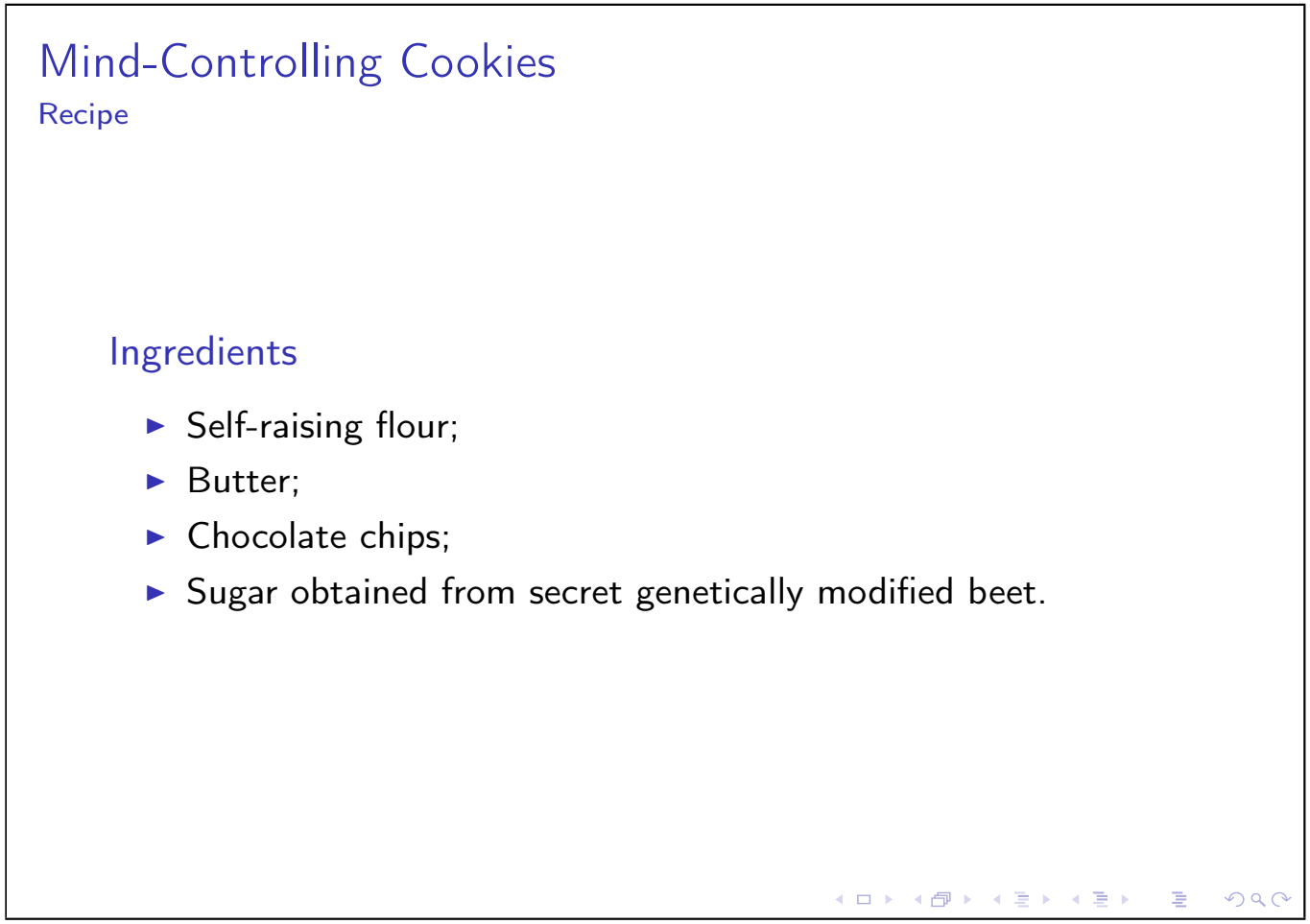


Figure 8.4 Block

```
    \begin{itemize}[<+->]
    \item Self-raising flour;
    \item Butter;
    \item Chocolate chips;
    \item Sugar obtained from secret genetically modified beet.
    \end{itemize}
  \end{block}
\end{frame}
```

Input

This creates five slides. The first just has one item (Figure 8.5), the second has two items (Figure 8.6), etc.

This completely hides the text until it's uncovered. If you prefer to show the text faintly before it's uncovered, you can use:

```
\setbeamercovered{transparent}
```

Input

This mixes 85% of the background colour with 15% of the text colour. Note that the effect varies according to the display device. Now the first slide of:

Input

```
\begin{frame}
```

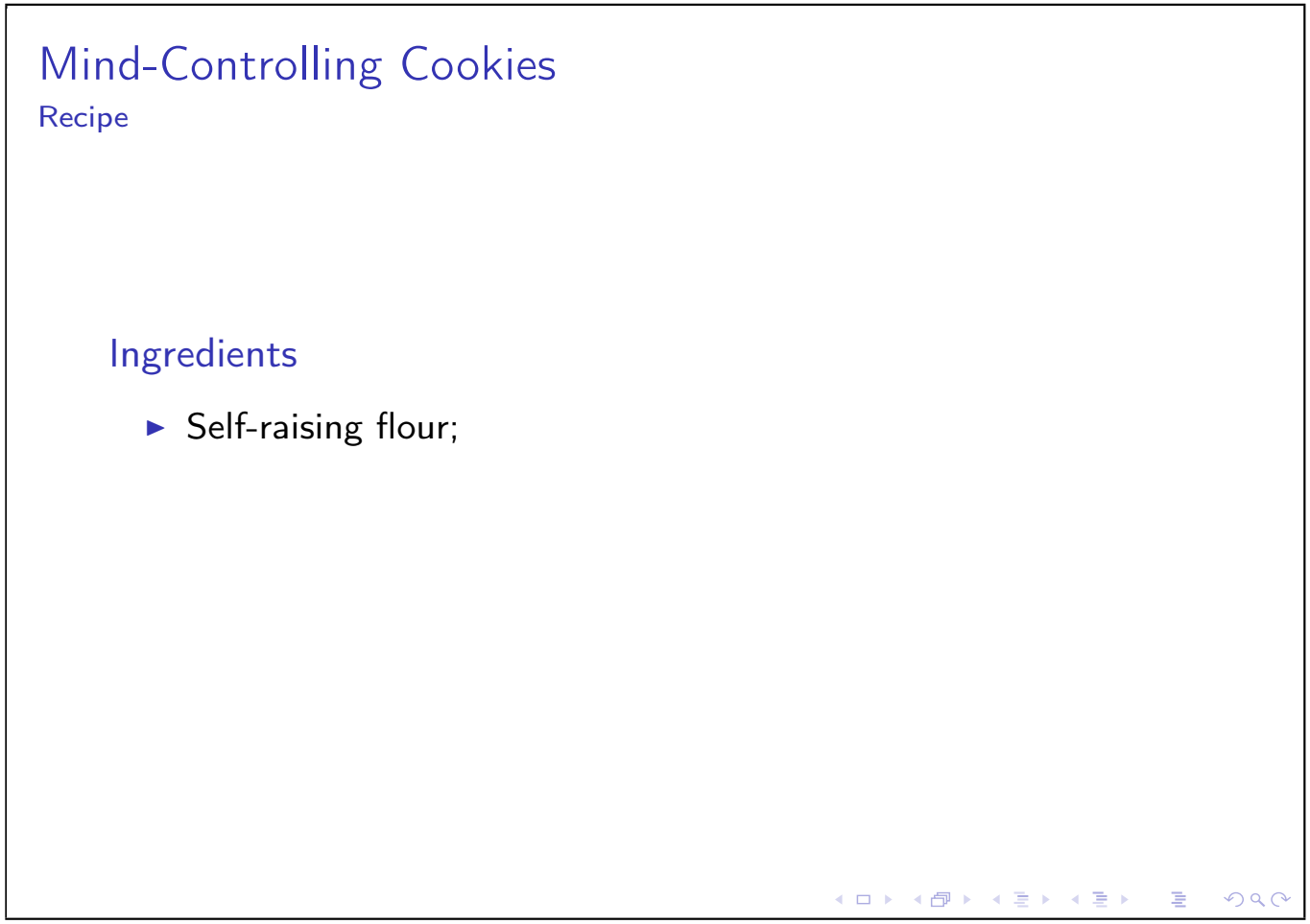


Figure 8.5 Overlays (First Slide)

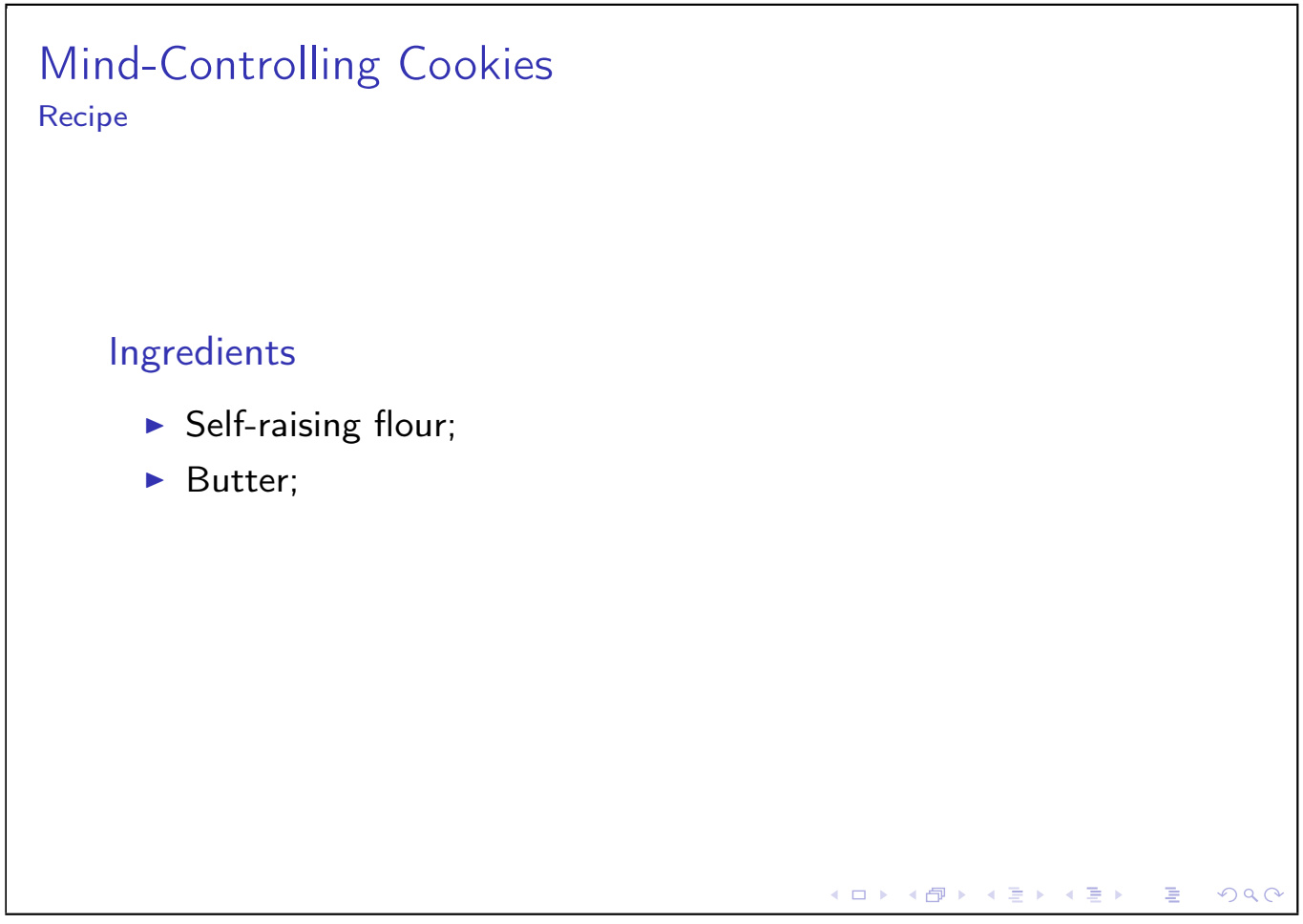


Figure 8.6 Overlays (Second Slide)

```
  \frametitle{Mind-Controlling Cookies}
  \framesubtitle{Recipe}

  \begin{block}{Ingredients}
    \begin{itemize}[<+->]
    \item Self-raising flour;
    \item Butter;
    \item Chocolate chips;
    \item Sugar obtained from secret genetically modified beet.
    \end{itemize}
  \end{block}
\end{frame}
```

↓ Input

faintly shows the second item onwards (see Figure 8.7).

Alternatively, you can specify the overlay information for each item.

Example:

↑ Input

```
\begin{frame}
  \frametitle{Mind-Controlling Cookies}
  \framesubtitle{Recipe}
```

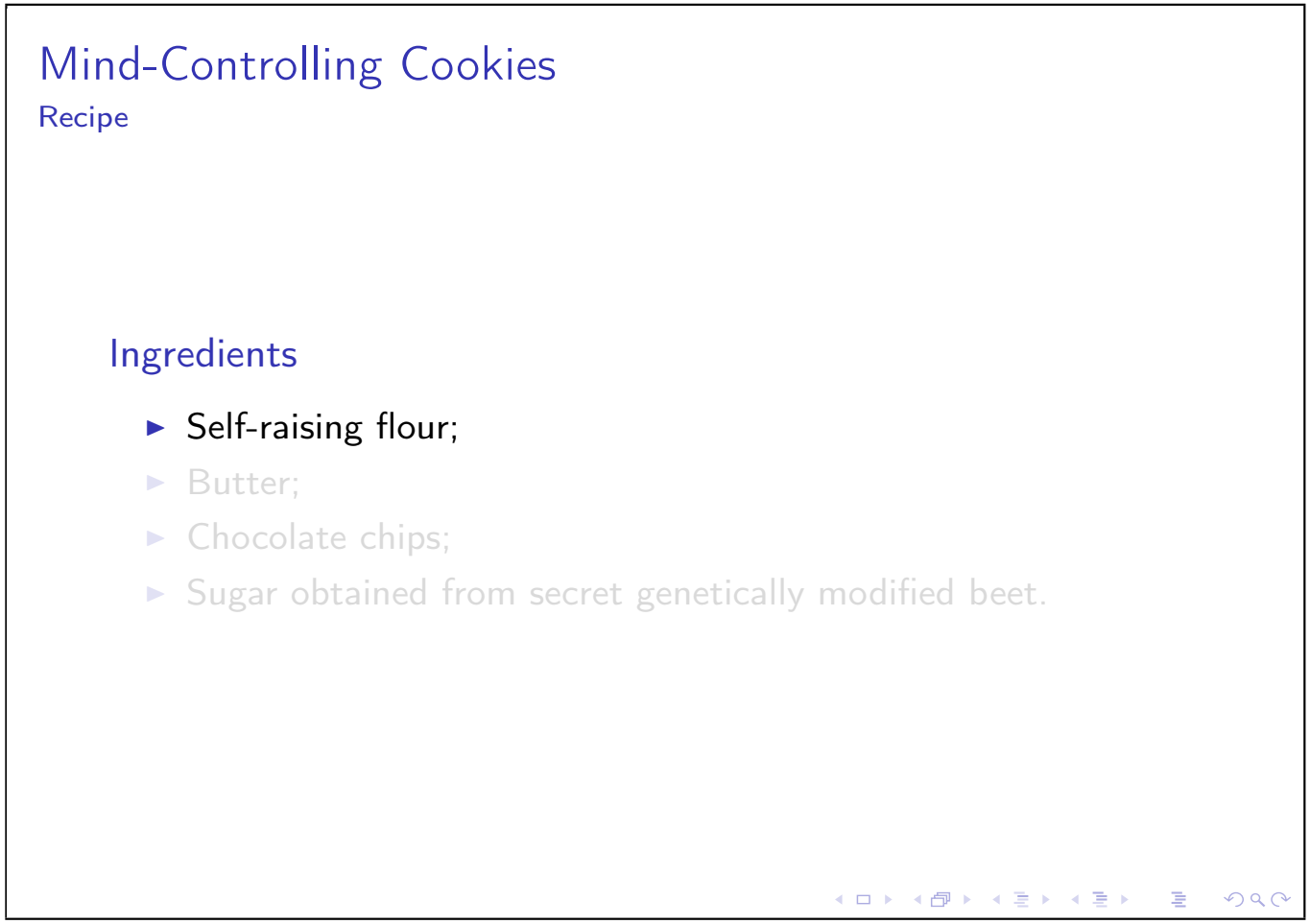


Figure 8.7 Overlays (`transparent` option)

```
  \begin{block}{Ingredients}
  \begin{itemize}
    \item<1-> Self-raising flour;
    \item<2-> Butter;
    \item<1-> Chocolate chips;
    \item<2-> Sugar obtained from secret genetically
modified beet.
    \end{itemize}
  \end{block}
\end{frame}
```

↓ Input

This shows the first and third items on the first slide and all items on the second slide. You can similarly apply overlays to beamer environments, such as block, or to standard environments, such as enumerate:

↑ Input

```
\begin{frame}
  \frametitle{Mind-Controlling Cookies}
  \framesubtitle{Recipe}
```

```
  \begin{block}{Ingredients}
    \begin{enumerate}[<+->]
    \item Self-raising flour;
    \item Butter;
    \item Chocolate chips;
    \item Sugar obtained from secret genetically modified beet.
    \end{enumerate}
  \end{block}
\end{frame}
```

↓ Input

The general syntax of the overlay specification is <⟨*start*⟩-⟨*end*⟩>, where ⟨*start*⟩ is the starting index and ⟨*end*⟩ is the end index. Parts of the specification can be omitted. For example, <2-> means slide 2 onwards whereas <2> means only on slide 2. This specification can also be applied to some common commands, such as

`\includegraphics`<⟨*overlay*⟩>[⟨*options*⟩]{⟨*image file*⟩}

Definition

For full syntax, see the beamer manual [102].

8.2 Themes

The appearance of the slides is governed by *themes*. There are five types of theme:

1. Presentation

   A presentation theme governs the whole appearance of the presentation. A presentation theme is chosen with:

   `\usetheme[`⟨*options*⟩`]{`⟨*name*⟩`}` Definition

2. Color

   A color theme governs the presentation's colour scheme. A color theme is chosen with:

   `\usecolortheme[`⟨*options*⟩`]{`⟨*name*⟩`}` Definition

3. Font

   A font theme governs the presentation's fonts or font attributes. A font theme is chosen with:

`\usefonttheme[`⟨*options*⟩`]{`⟨*name*⟩`}` Definition

4. Inner

   An inner theme governs the appearance of elements that are considered "inside" a frame. (For example, the appearance of theorems or list items.) An inner theme is chosen with:

   `\useinnertheme[`⟨*options*⟩`]{`⟨*name*⟩`}` Definition

5. Outer

   An outer theme governs the appearance of elements outside a frame. (For example, the headlines or footlines or whether there is a sidebar.) An outer theme is chosen with:

   `\useoutertheme[`⟨*options*⟩`]{`⟨*name*⟩`}` Definition

The examples above used the default presentation theme. There are a large number of themes to choose from. The rest of this section shows a selection of these themes. The selection is partly influenced by my own preferences, but partly by how well they appear in grey scale for the printed version of this book.

Example 43. Presentation Themes (Boadilla)

Input

```
\documentclass{beamer}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usetheme{Boadilla}

\title{Culinary Experimental Research}
\subtitle{Mind-Controlling Cookies and Exploding Chocolates}
\author[Canary and Zebra]{Mabel Canary\inst{1} \and
Zöe Zebra\inst{2}}
\date[Mar'14]{22nd March 2014}
\institute[SLES \& DSC]{\inst{1}Secret Lab of Experimental Stuff\\
  University of Somewhere\and
  \inst{2}Department of Stripy Confectioners\\
  College of Somewhere Else
}
\titlegraphic{\includegraphics[width=1in]{dummy-logo}}
```

```
\begin{document}
  \begin{frame}
    \maketitle
  \end{frame}
\end{document}

\section{Experimental Research}
\subsection{Cookies}

\begin{frame}
  \frametitle{Mind-Controlling Cookies}
  \framesubtitle{Recipe}

  \begin{block}{Ingredients}
  \begin{itemize}
    \item Self-raising flour;
    \item Butter;
    \item Chocolate chips;
    \item Sugar obtained from secret genetically modified beet.
    \end{itemize}
  \end{block}
```

```
\end{frame}
```

Input

This document uses the Boadilla presentation theme. The resulting slides are shown in Figures 8.8 and 8.9. This theme puts the author names, affiliations, title and date in the footline. You can download or view this document.

In the above example, if

```
\usetheme{Boadilla}
```

Input

is changed to

```
\usetheme{EastLansing}
```

Input

then the resulting slides have a green colour scheme with a headline (shown in Figures 8.10 and 8.11).

The Montpellier theme shows the sectioning information as a tree in the headline (see Figures 8.12 and 8.13) or there's the Goettingen theme that has a sidebar with a table of contents (see Figures 8.14 and 8.15).

You can mix a presentation theme with the other types of themes. For example, if I replace the line

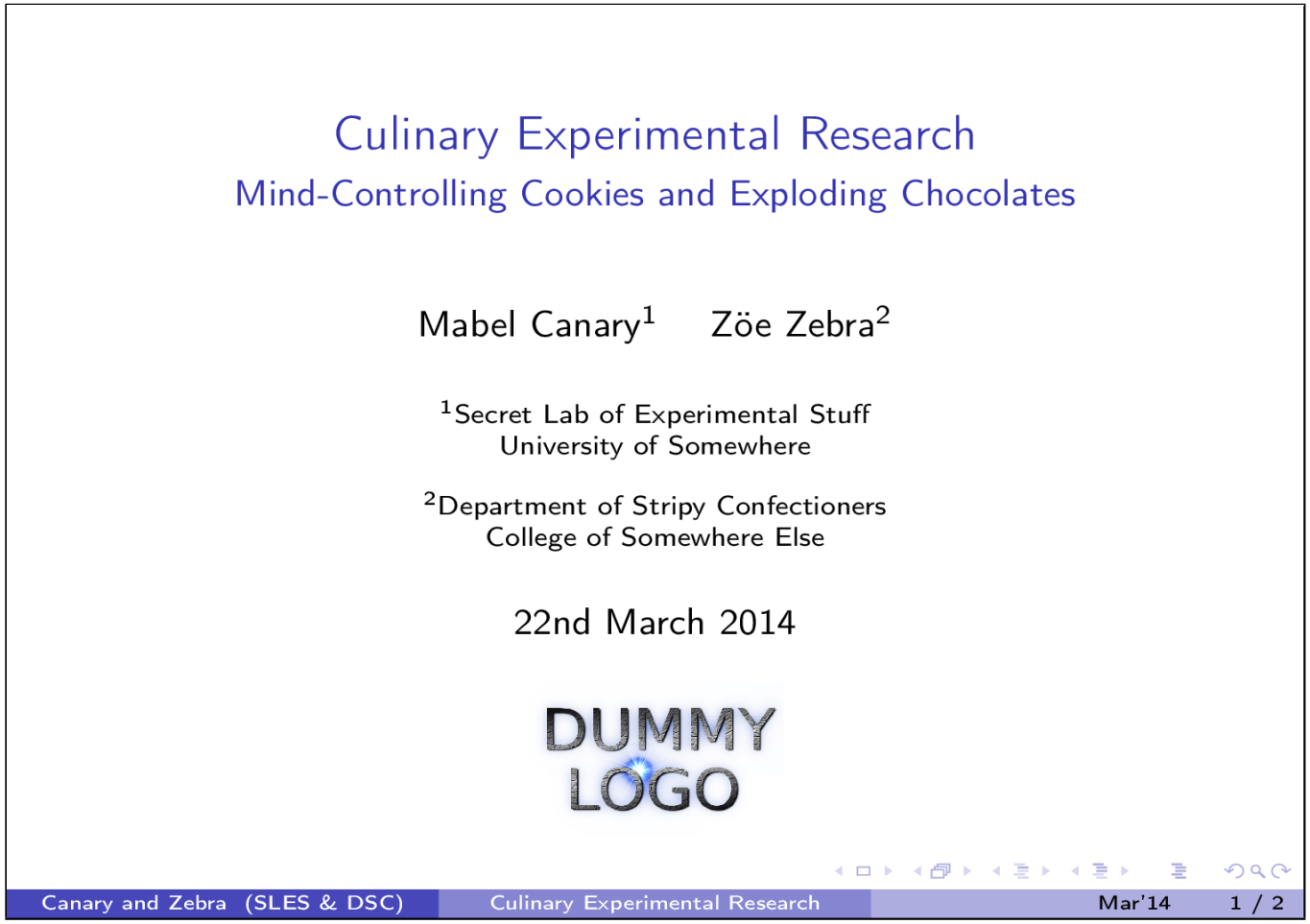


Figure 8.8 Boadilla Theme (Title Slide)

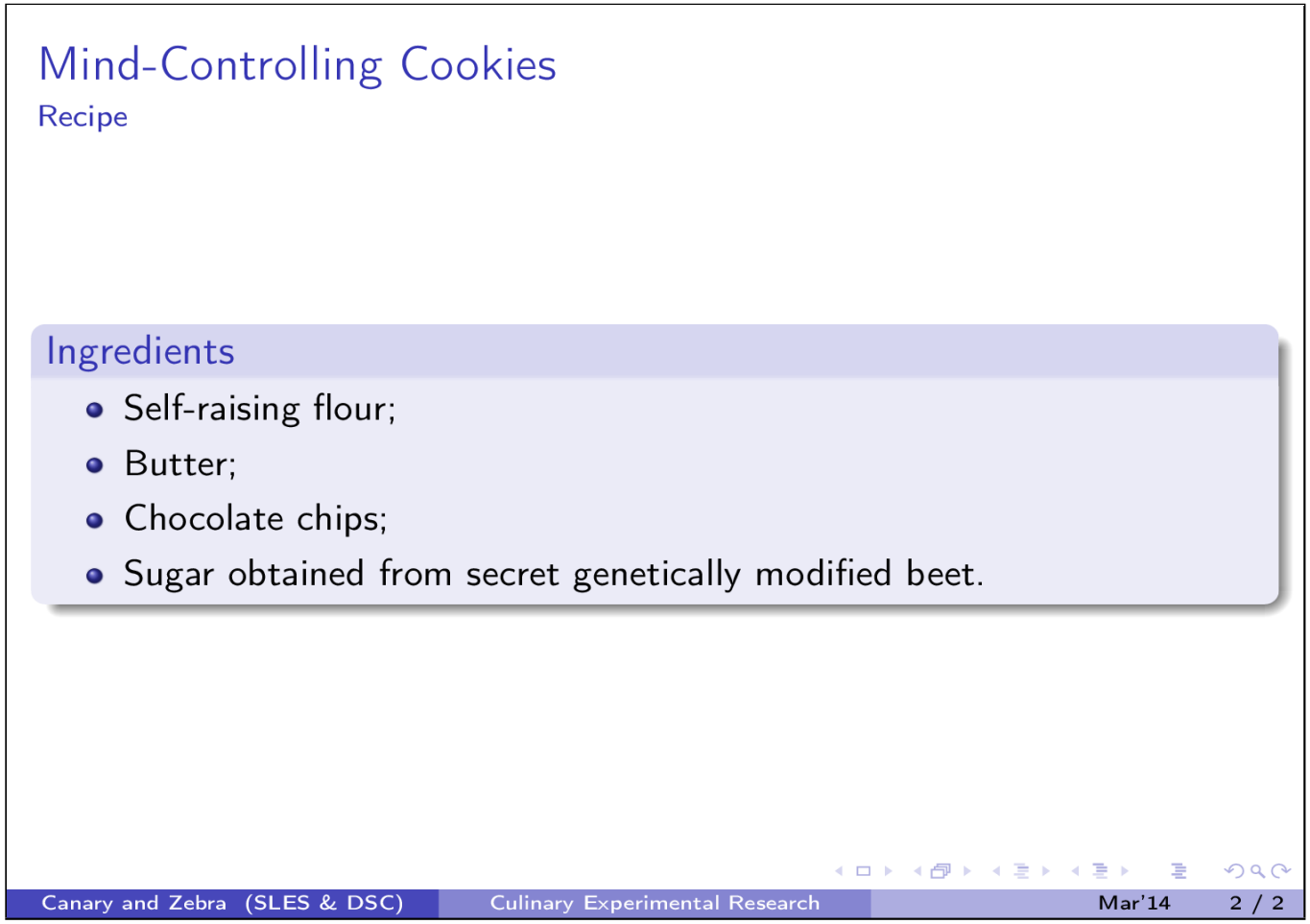


Figure 8.9 Boadilla Theme

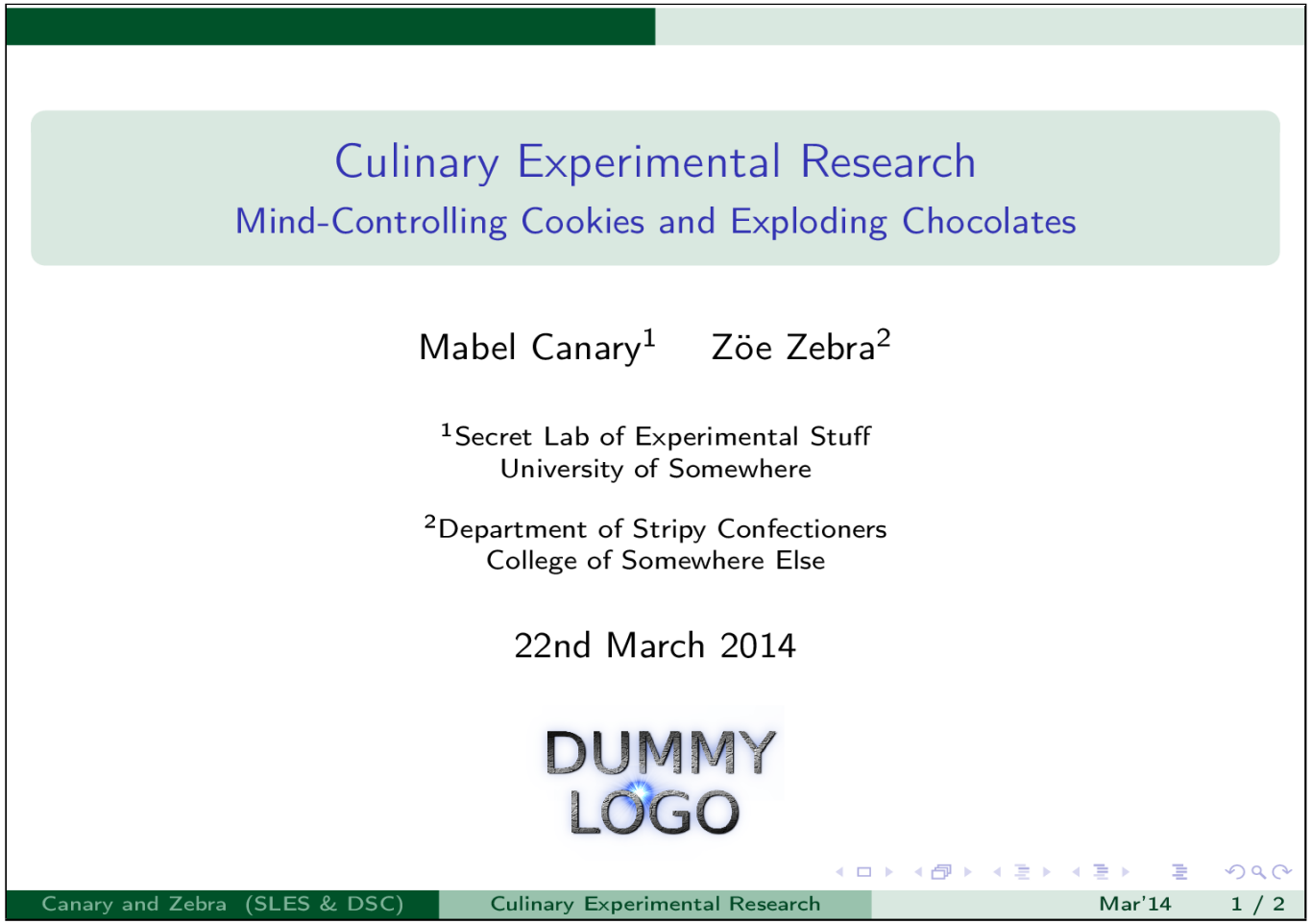


Figure 8.10 EastLansing Theme (Title Slide)

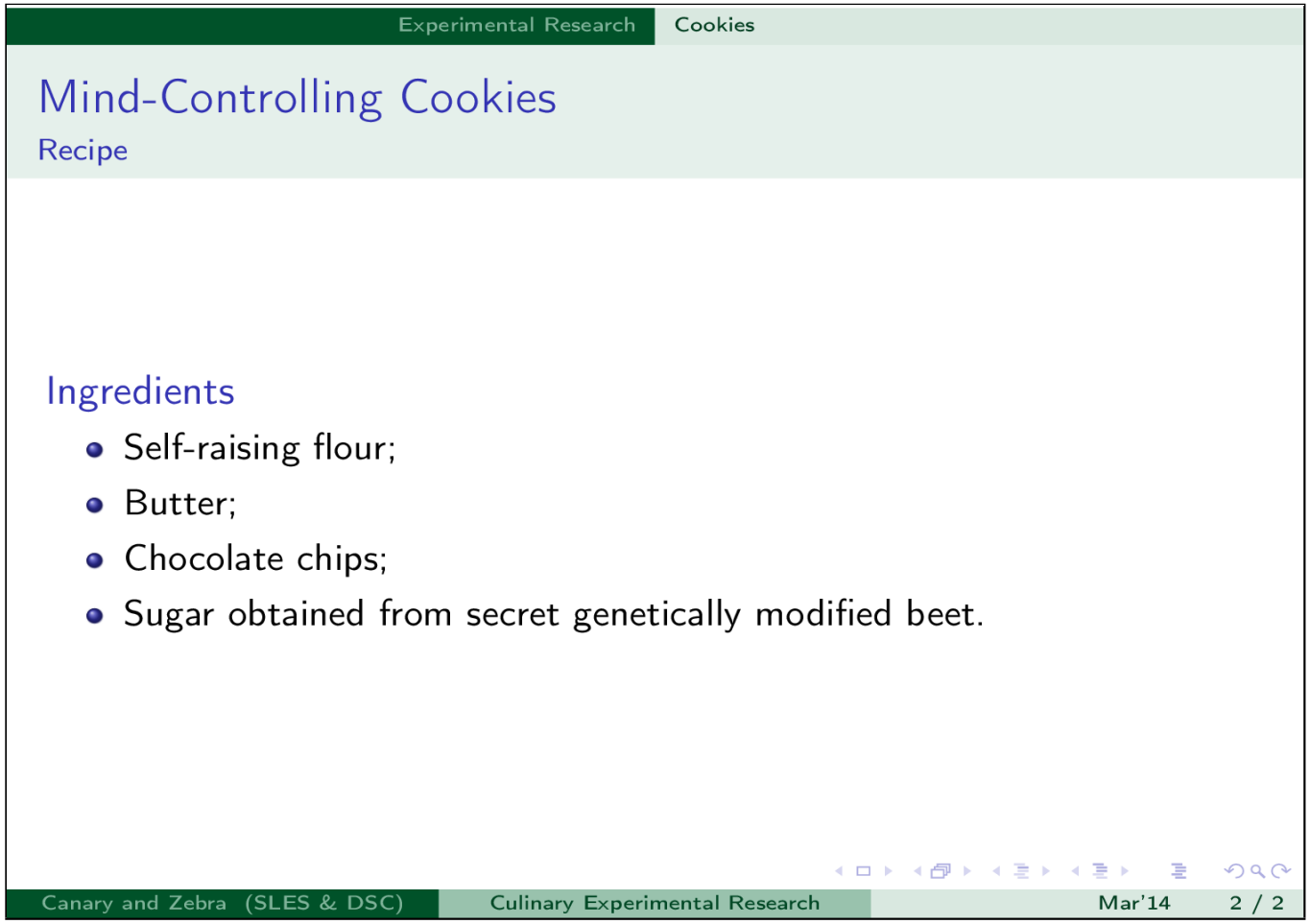


Figure 8.11 EastLansing Theme

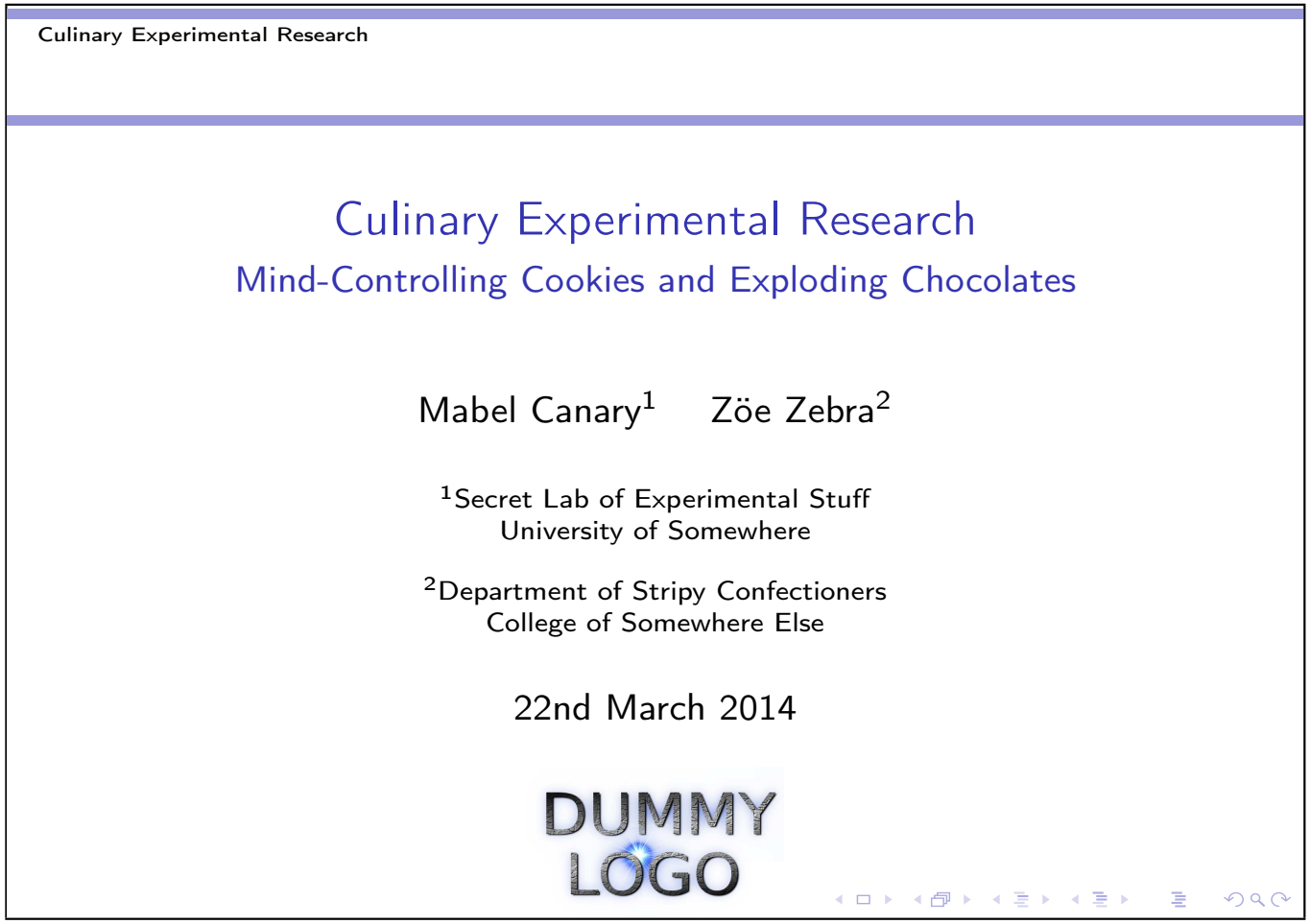


Figure 8.12 Montpellier Theme (Title Slide)

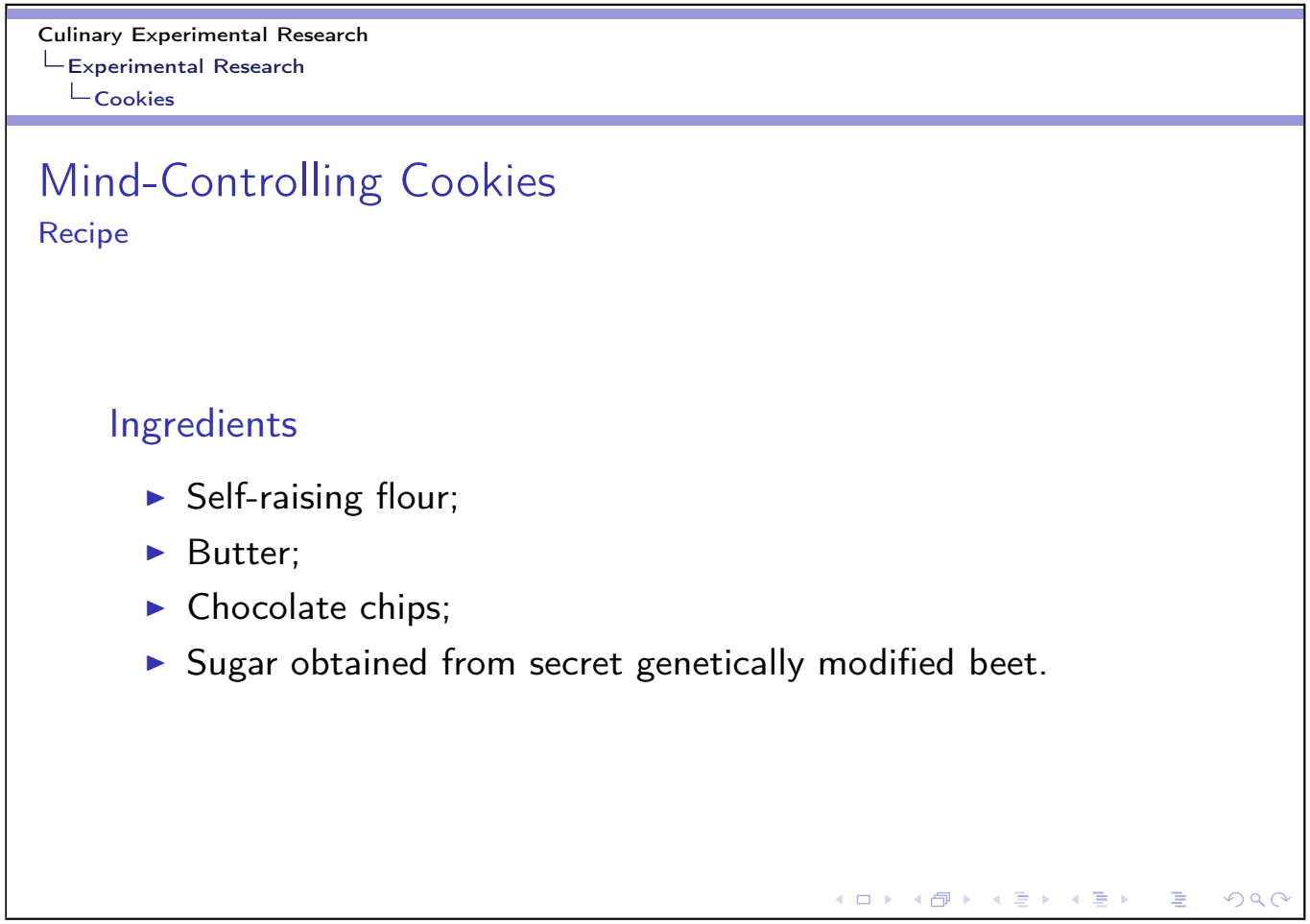


Figure 8.13 Montpellier Theme

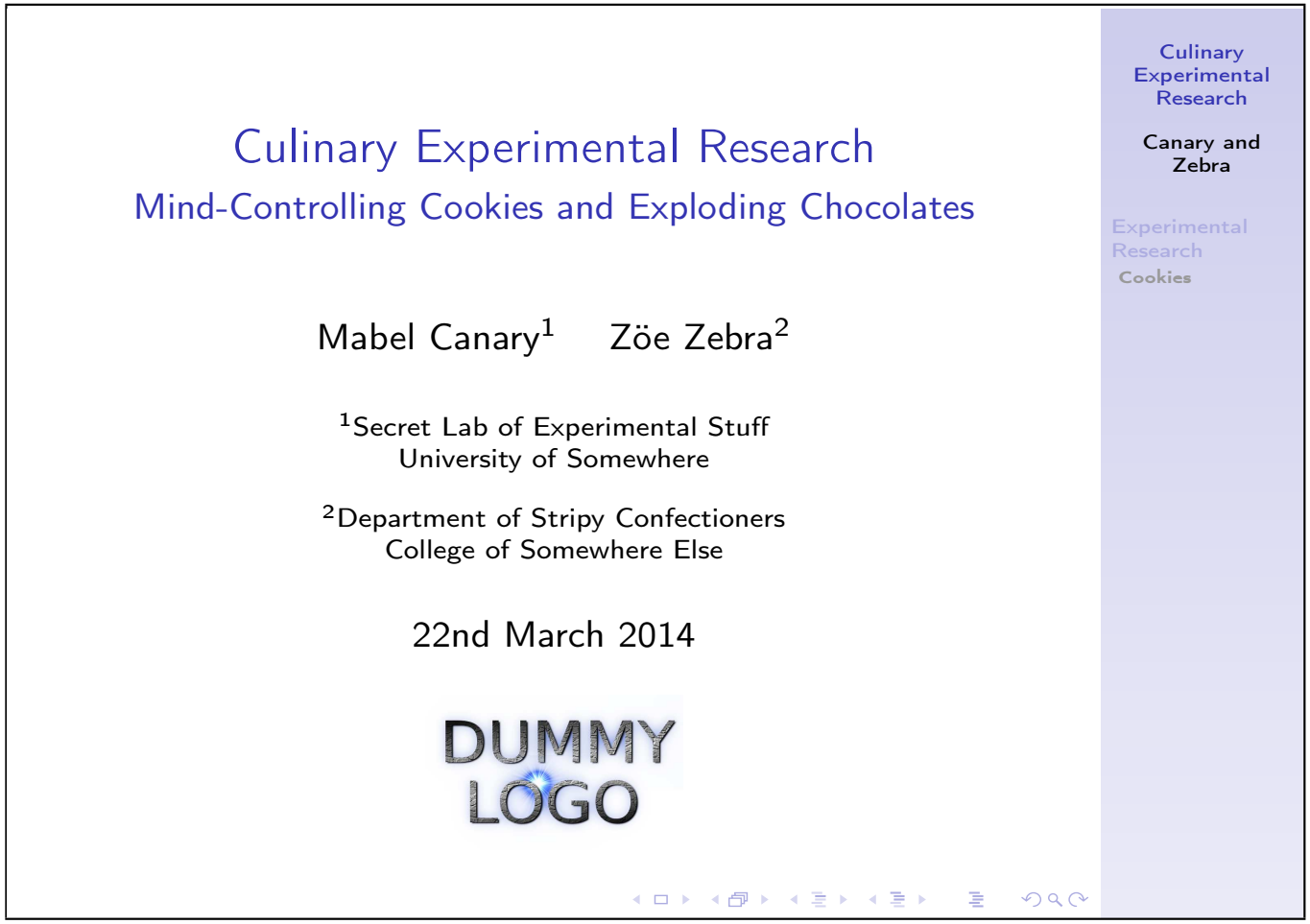


Figure 8.14 Goettingen Theme (Title Slide)

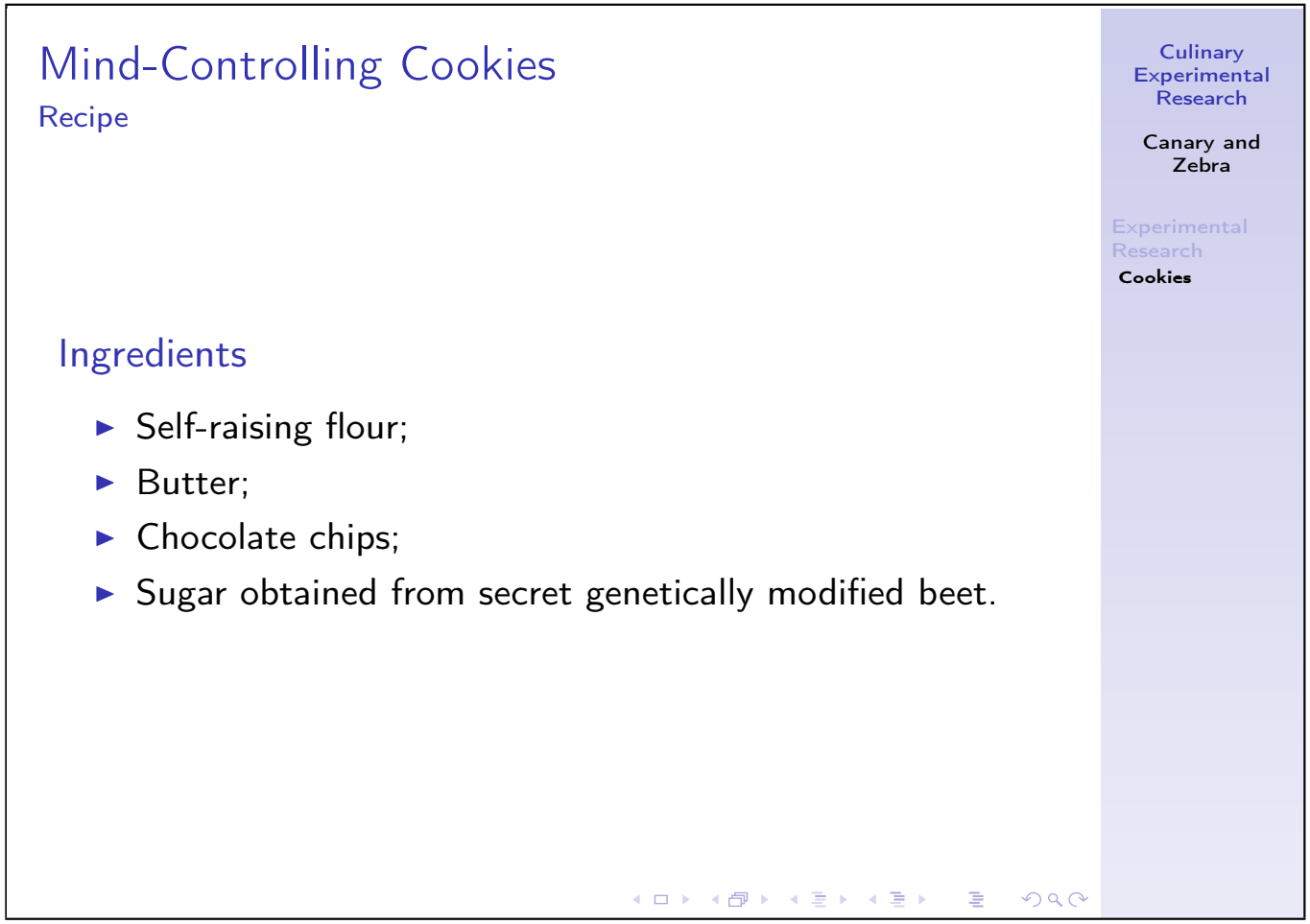


Figure 8.15 Goettingen Theme

```
\usetheme{Boadilla}
```

Input

from Example 43 with

```
\usetheme{Goettingen}
\useinnertheme[shadow]{rounded}
\usecolortheme{spruce}
```

↑ Input

↓ Input

then the result is as shown in Figures 8.16 and 8.17.

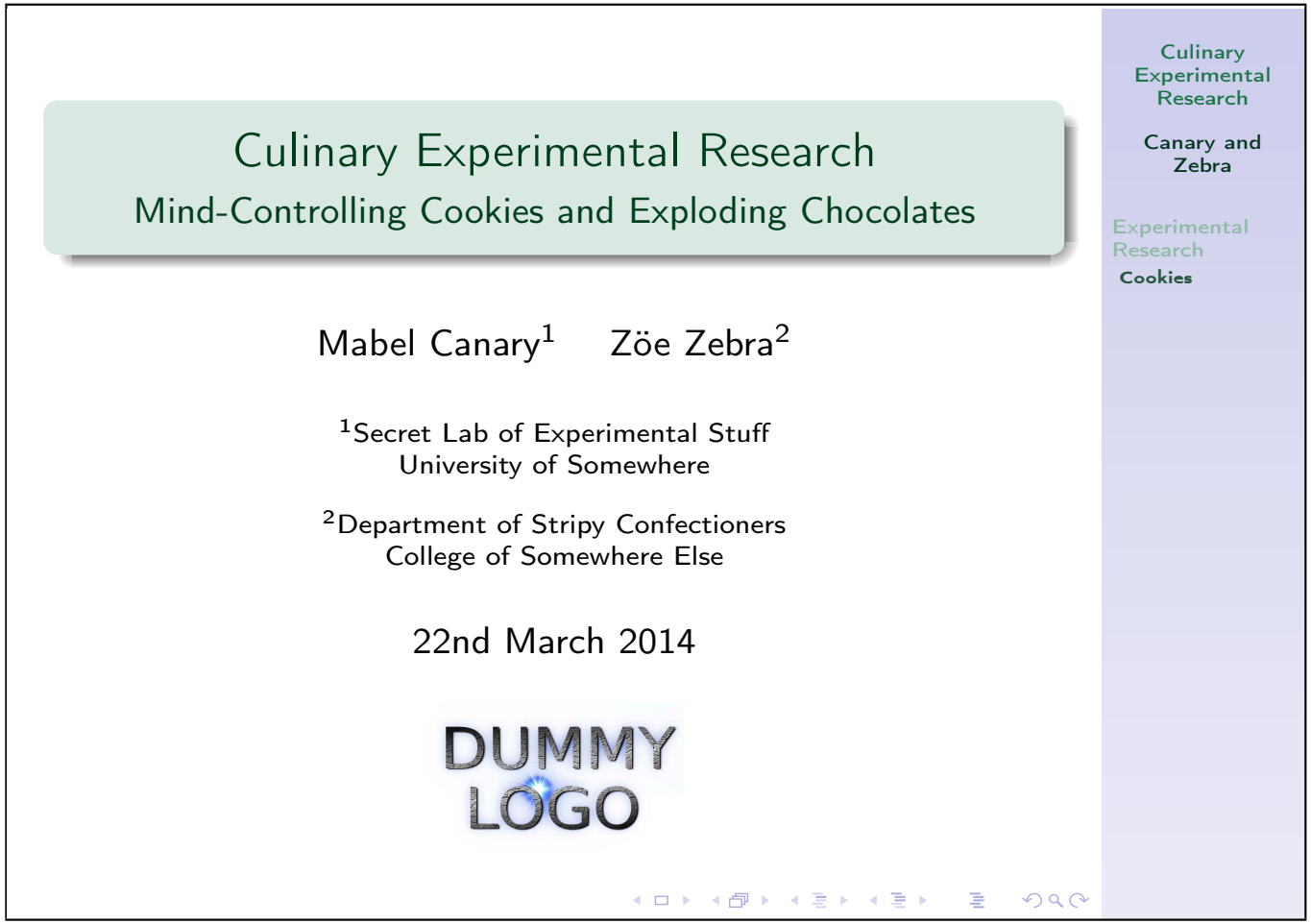


Figure 8.16 Goettingen Presentation Theme with rounded Inner Theme and spruce Color Theme (Title Slide)

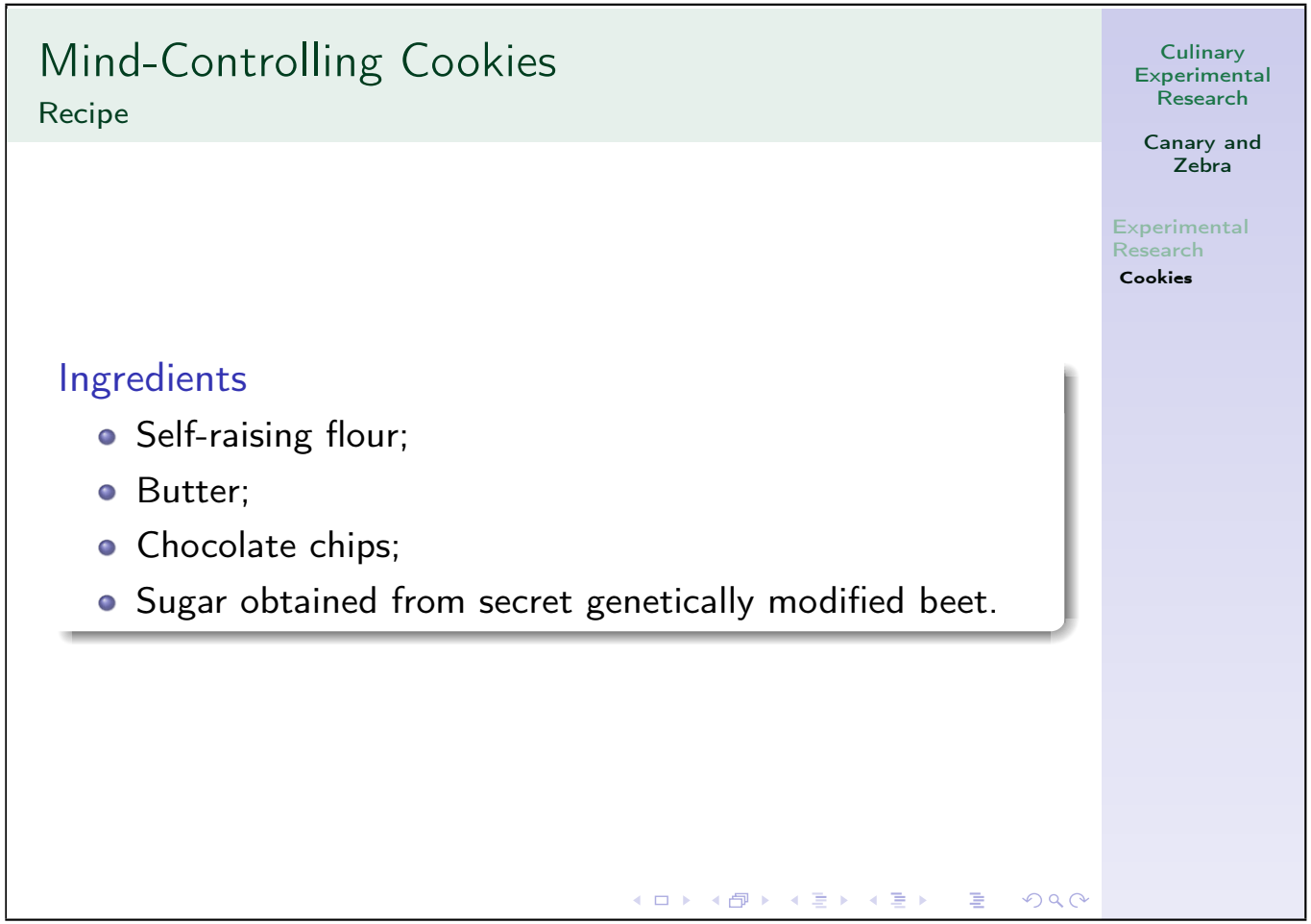


Figure 8.17 Goettingen Presentation Theme with rounded Inner Theme and spruce Color Theme

9. Assignments and Examinations

There are a number of classes or packages available on CTAN to help typeset assignment sheets or exam papers (see the exam topic).

The exam class (version 2.4, 2011-05-22) is comprehensive with its own list-like environments for enumerating questions and their parts, as well as environments for multiple choice questions. It also provides the possibility of assigning points to each question (or question part), creating solution sheets and grading tables. The exam class is described in §9.1, but take care as there is also an exam class provided by the exams bundle. The exam class described in this book is the one by Philip S. Hirschhorn and is in both the TeX Live and MiKTeX distributions. The exams bundle isn't in either of those distributions.

The exsheets package (version 0.17, 2014-10-15) also provides a means to create questions and their solutions. Unlike the exam class, the exsheets package allows you to divide the questions into classes and they can be printed selectively. Meta-data can also be assigned to the questions. The solutions may be either printed with their question or collected and printed together at a later point in the document. However, the package doesn't

support multiple choice style of questions (although you can use a package such as paralist to create inline numbered lists). The exsheets package is described in §9.2.

The probsoln package (version 3.04, 2012-08-23) provides a way to define problems and their solutions in a file (or multiple files). You can load all problems, a specific list of problems or a random selection. You may have one or more problem datasets and either print the solutions with the questions or print the solutions later on in the document. The probsoln package provides an inline numbered list environment. The probsoln package is described in §9.3.

The datatool package used in earlier chapters can also be used to store questions and their solutions. The `datatooltk` application can import the dataset files used by probsoln. You can add extra fields to specify, for example, the difficulty level or subject. §9.4 describes how you can use datatool to create a database of exam or assignment questions and their solutions.

Note that it's possible to use the exam class with either the probsoln package or the datatool package, but it can't be used with the exsheets package.


This chapter assumes that all your files are stored in a secure location that can't be accessed by curious students. If your operating system supports owner-only file permissions, I suggest you set them as appropriate (for example `chmod 600`), but don't rely on that as your only security measure (see also §2.3).

9.1 The exam Class

The exam class is a comprehensive class for typesetting examination papers. The user guide [35] is over 100 pages, so this section is only intended as a brief introduction. The exam class has the standard class options, such as `10pt`, `11pt` and `12pt`, but also provides two other options `addpoints` and `answers`, described below.

The `addpoints` class option enables the point-totalling commands. If this option is used, all points must be specified as integers. Half points may be indicated with the command:

`\half` Definition

For example, `2\half` instead of `2.5`.

In addition to the `addpoints` class option, you can also switch this mode on and off using the commands:

`\addpoints` Definition

(to switch it on) and

`\noaddpoints` Definition

(to switch it off).

The answers class option will display solutions. Again there are commands to switch this mode on and off:

\printanswers Definition

(to show the solutions) and

\noprintanswers Definition

(to hide the solutions).

All the exam questions should be contained within the single questions environment.

\begin{questions}
⟨*exam questions*⟩ Definition
\end{questions}

Within the questions environment, you start a new question using:

\question[⟨*points*⟩] Definition

where ⟨*points*⟩ is the number of points this question is worth. As mentioned above, if the addpoints option is on, ⟨*points*⟩ must an integer with optionally the \half command (or just the \half command on its own). The question is automatically numbered.

You can replace \question[⟨*points*⟩] with

\titledquestion{⟨*title*⟩}[⟨*points*⟩] Definition

if question should have a title. (The question number won't be included by default, but you can add it to the title using \thequestion in the ⟨*title*⟩ argument.)

The exam class user guide [35] suggests the following code if you want to provide space for the student to fill in their name if they are to write on the question sheet:

↑ Input

```
\begin{center}
 \fbox{\parbox{5.5in}{\centering
  Answer the questions in the spaces provided on the
  question sheets. If you run out of room for an answer,
  continue on the back of the page.%
 }}
\end{center}

\vspace{0.1in}

\makebox[\textwidth]{Name and section:\enspace\hrulefill}
```

```
\vspace{0.2in}

\makebox[\textwidth]{Instructor's name:\enspace\hrulefill}
```

Input

(Although you may prefer to use a fraction of \textwidth instead of the hardcoded 5.5in.)

Example:

Input

```
\documentclass[addpoints]{exam}

\begin{document}
\begin{center}
 \fbox{\parbox{0.8\textwidth}{\centering
  Answer the questions in the spaces provided on the
  question sheets. If you run out of room for an answer,
  continue on the back of the page.%
 }}
\end{center}

\vspace{0.1in}
```

```
\noindent\makebox[\textwidth]{Name and section:\enspace\hrulefill}

\vspace{0.2in}

\noindent\makebox[\textwidth]{Instructor's name:\enspace\hrulefill}
\begin{questions}
 \question[3\half] What ingredients can be found in
  mind-controlling cookies?

 \question[2\half] What are the health benefits of
  exploding chocolates?
\end{questions}
\end{document}
```

↓ Input

This produces:

↑ Output

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page.

Name and section: _____

Instructor's name: _____

1. (3½ points) What ingredients can be found in mind-controlling cookies?

2. (2½ points) What are the health benefits of exploding chocolates?

↓ Output

Questions may have parts, sub-parts or sub-sub-parts. Question parts should be enclosed in the parts environment:

```
\begin{parts}
⟨question parts⟩
\end{parts}
```

Definition

where each part is started with

```
\part[⟨points⟩]
```

Definition

Note that outside of the parts environment this command behaves as the standard \part sectioning command.

Sub-parts should be enclosed in the subparts environment:

\begin{subparts}
⟨*question sub-parts*⟩ Definition
\end{subparts}

where each sub-part is started with

\subpart[⟨*points*⟩] Definition

Sub-sub-parts should be enclosed in the subsubparts environment:

\begin{subsubparts}
⟨*question sub-sub-parts*⟩ Definition
\end{subsubparts}

where each sub-sub-part is started with

\subsubpart[⟨*points*⟩] Definition

As with \question, the optional argument indicates the number of points the part is worth, and should only contain digits or \half if the addpoints option is on.

Example:

↑ Input

```
\begin{questions}
```

```
\question Find the derivatives with respect to $x$ of the
following functions:
 \begin{parts}
 \part[\half] $y = x + 1$
 \part[1] $y = x^3 + 4x^2 - x + 3$
 \part[1\half] $y = \cos(x^2)$
 \end{parts}
\end{questions}
```

↓ Input

The result is:

↑ Output

1. Find the derivatives with respect to x of the following functions:
   (a) (½ point) $y = x + 1$
   (b) (1 point) $y = x^3 + 4x^2 - x + 3$
   (c) (1½ points) $y = \cos(x^2)$

↓ Output

If you want to specify questions worth bonus points you can use

\bonusquestion Definition

instead of \question,

\bonustitledquestion Definition

instead of \titledquestion,

\bonuspart Definition

instead of \part,

\bonussubpart Definition

instead of \subpart and

\bonussubsubpart Definition

instead of \subsubpart.

The default location of the points is after the question (or part) number. This can be changed to the margin using the declaration

\pointsinmargin Definition

which puts the points in the left margin, or the declaration

\pointsinrightmargin Definition

which puts the points in the right margin. If you prefer to place the points elsewhere, use the declaration

\pointsdroppedatright Definition

to switch off the automatic placement of the points and use

\droppoints Definition

at the end of the paragraph where you want the points displayed. If you use \pointsdroppedatright but don't use \droppoints the points won't be displayed. Note that \droppoints should only occur at the end of a paragraph or between paragraphs.

The number of questions, parts, sub-parts and sub-sub-parts can be referenced with the commands:

\numquestions Definition

(the number of questions),

\numparts Definition

(the number of parts),

\numsubparts Definition

(the number of sub-parts), and

\numsubsubparts Definition

(the number of sub-sub-parts).

If you have used the addpoints option, you can reference the total number of points using:

\numpoints Definition

For example:

↑ Input

```
This exam has \numquestions\␣questions worth a total
of \numpoints\␣points.
```

↓ Input

With the addpoints option, you can also display a grading table using:

\gradetable[⟨*orientation*⟩][⟨*index type*⟩] Definition

where ⟨*orientation*⟩ may be either h (horizontal) or v (vertical) and ⟨*index type*⟩ may be either questions or pages. The defaults are a vertical table

indexed by question. The grading table provides space for the student marks to be written in by hand. You will need at least two LaTeX runs (possibly three or four) to ensure the grading table is up-to-date.

If you want to centre the grading table, you can place it in the center environment, which will add extra vertical space at the start and end as well as centre its contents. (It's unlikely that you'll want the grading table in a floating environment as it typically appears in a fixed location.) The grading table is placed in a tabular environment, so it can't break across a page.

Any cover page material can be place in the coverpages environment.

```
\begin{coverpages}
⟨text⟩
\end{coverpages}
```

Definition

The cover pages use Roman numeral page numbers. The page counter is reset at the end of the environment. This environment must not contain the questions environment.

There are four environments for typesetting the solutions. The default behaviour is for the solutions to be hidden. To display the solutions, use the answers class option. Each environment takes an optional argument, which is the amount of space to be left for the students to write in their solution. If the optional argument is omitted, all four environments simply ignore their contents when the answers option hasn't been set.

\begin{solution}[⟨*length*⟩]
⟨*solution text*⟩ Definition
\end{solution}

If the solutions are hidden, this environment inserts ⟨*length*⟩ amount of blank space (as though you had used \vspace*{⟨*length*⟩}).

\begin{solutionorbox}[⟨*length*⟩]
⟨*solution text*⟩ Definition
\end{solutionorbox}

The solutionorbox environment is similar to the solution environment, but inserts an empty box of height ⟨*length*⟩ if the answers should be hidden.

\begin{solutionorlines}[⟨*length*⟩]
⟨*solution text*⟩ Definition
\end{solutionorlines}

The solutionorlines environment is similar to the solution environment, but inserts an area of height ⟨*length*⟩ with ruled lines if the answers should be hidden.

\begin{solutionordottedlines}[⟨*length*⟩]
⟨*solution text*⟩ Definition
\end{solutionordottedlines}

The solutionordottedlines environment is similar to solutionorlines but uses dotted lines.

Example:

Input

```
\begin{questions}
\question Find the derivatives with respect to $x$ of the
following functions:
 \begin{parts}
 \part[\half] $y = x + 1$

 \begin{solution}
 $y' = 1$
 \end{solution}

 \part[1] $y = x^3 + 4x^2 - x + 3$

 \begin{solution}
 $y' = 3x^2 + 8x - 1$
 \end{solution}
```

```
 \part[1\half] $y = \cos(x^2)$

 \begin{solution}
 $y' = -2x\sin(x^)$
 \end{solution}

 \end{parts}
\end{questions}
```

↓ Input

There are four environments provided for multiple choice questions. The first two label the choices and the other two print checkboxes in front of the choices for the student to tick. With these environments use:

\choice Definition

to start a new choice. To indicate the correct choice, use

\CorrectChoice Definition

instead of \choice.

\begin{choices}
⟨*list items*⟩ Definition
\end{choices}

The choices environment is a list environment with labelled choices as the items in the list.

```
\begin{oneparchoices}
⟨list items⟩
\end{oneparchoices}
```
Definition

The oneparchoices is an in-line list of labelled choices where there are no paragraph breaks unless explicitly inserted.

```
\begin{checkboxes}
⟨list items⟩
\end{checkboxes}
```
Definition

The checkboxes environment is a list environment with checkbox choices as the items in the list.

```
\begin{oneparcheckboxes}
⟨list items⟩
\end{oneparcheckboxes}
```
Definition

The oneparcheckboxes is an in-line list of checkbox choices where there are no paragraph breaks unless explicitly inserted.

Example:

↑ Input

```
\begin{questions}
  \question[1] Which of the following ingredients are used in
  mind-controlling cookies:
  \begin{choices}
    \choice arsenic
    \choice cyanide
    \choice curare
    \CorrectChoice secret genetically modified sugar beet
  \end{choices}
\end{questions}
```

↓ Input

This produces:

↑ Output

1. (1 point) Which of the following ingredients are used in mind-controlling cookies:

    A. arsenic

    B. cyanide

C. curare

D. secret genetically modified sugar beet

↓ Output

For more details about the exam class, including commands and options not mentioned here, see the exam class user guide [35].

Exercise 24. Creating an Exam Paper with the exam Class
Create an examination paper containing the questions (and solutions) in the above examples and add a grading table. Try experimenting with adding and removing the `answers` class option and try the variations of the solution environment (such as solutionorbox).

The exam class user guide [35] describes ways of adjusting the default settings. Try adding the command `\unframedsolutions` to the preamble and see how it changes the way the solutions are displayed or add `\bracketedpoints` to see how it affects the way the points are displayed. You can download or view a solution.

9.2 The exsheets Package

Unlike the exam class described in the previous section, exsheets is a package, so you need to find a suitable class to use with it. For the examples, I'm just going to use the article class, but you will probably find it easier to use a more flexible class, such as one of the KOMA-Script classes.

As with the exam class, the exsheets documentation [61] is quite long because there are so many options, so this section is just intended as an introduction. Options can be specified via the package option list, or in the optional argument to environments provided by exsheets, or via

`\SetupExSheets[`⟨*module*⟩`]{`⟨*option list*⟩`}` Definition

where the options listed in ⟨*option list*⟩ belong to the given module. If ⟨*module*⟩ is omitted, the module name is incorporated into the option list. For example, you can either do:

`\SetupExSheets[question]{`⟨*option*⟩`=`⟨*value*⟩`}` Input

or

`\SetupExSheets{question/`⟨*option*⟩`=`⟨*value*⟩`}` Input

where ⟨*option*⟩ is some option defined for the `question` module and ⟨*value*⟩ is the value being assigned to that option. There are a lot of options related

to the formatting of counters, question headings and subtitles, as well as options related to the table of contents.

Questions are written inside the question environment.

```
\begin{question}[⟨options⟩]{⟨points⟩}
⟨question text⟩
\end{question}
```

Definition

Note that the ⟨*points*⟩ argument is optional, despite the lack of square brackets. The other optional argument, ⟨*options*⟩, is the list of options. If ⟨*points*⟩ is omitted, no points will be associated with the question. The ⟨*points*⟩ may be in the form ⟨*p*⟩ or ⟨*p*⟩+⟨*b*⟩ or +⟨*b*⟩, where ⟨*p*⟩ is the number of points and ⟨*b*⟩ is the number of bonus points.

Examples:
A question with no points allocated:

↑ Input

```
\begin{question}
How many kilocalories are there in 100 grammes of exploding
chocolate?
\end{question}
```

↓ Input

A question worth five points:

Input

```
\begin{question}{5}
How many kilocalories are there in 100 grammes of exploding
chocolate?
\end{question}
```

Input

A question worth five points and one bonus point:

Input

```
\begin{question}{5+1}
How many kilocalories are there in 100 grammes of exploding
chocolate?
\end{question}
```

Input

The points for each question are added to the total marks. If you don't want the points added for a particular question, you need to put an exclamation mark ! before the points. (This prevents bonus points for the question.)

Example:
A question worth five points where the points aren't added to the running total:

Input

```
\begin{question}{!5}
How many kilocalories are there in 100 grammes of exploding
chocolate?
\end{question}
```

Input

There are a number of ⟨*key*⟩=⟨*value*⟩ options available for the question environment. For a complete list, see the exsheets documentation [61], but here are a few:

`type`=⟨*value*⟩ Determines the type of question. The value may be either `exam` or `exercise`. In the first case, the question number is preceded by "Question", in the second case by "Exercise".

`name`=⟨*name*⟩ Replaces the default "Exercise" or "Question" to ⟨*name*⟩.

`subtitle`=⟨*title*⟩ Adds a subtitle.

`class=`⟨*class*⟩	Assigns a class to the question.
`topic=`⟨*topic*⟩	Assigns a topic to the question.
`ID=`⟨*label*⟩	Assigns an ID to the question for later reference.
`print`	A boolean key. If true, the question is displayed. By default this is true.

The solution to a question should come after the question environment and should be placed inside the solution environment.

```
\begin{solution}[⟨options⟩]
⟨solution text⟩
\end{solution}
```

Definition

As with the question environment, the solution environment also allows the `print` boolean key in its ⟨*options*⟩, but by default it's false. The `name` key is also available in ⟨*options*⟩ and can be used to replace the default "Solution" text that precedes the solution number.

Example:

```
\begin{question}
What is the main drawback of ray guns?
\end{question}
\begin{solution}
Overheating.
\end{solution}
```

Input

Input

By default, the question will appear but the solution won't be displayed. The default result is:

Output

Exercise 1.
What is the main drawback of ray guns?

Output

If you want the solutions to appear where they are defined (that is, after their associated question), you can just add the following:

```
\SetupExSheets[solution]{print=true}
```

Input

or

```
\SetupExSheets{solution/print=true}
```
Input

However, if you want the solutions to appear later (for example, at the end of the document) you can use

```
\printsolutions[⟨settings⟩]
```
Definition

at the place where you want the solutions. Note that this command must only be used *after* all the solutions have been defined. The optional argument is a key=value list. Options include:

`chapter` If no value is specified, all solutions to the questions defined in the current chapter are listed. If a value is specified, it may be a comma-separated list or range of chapter numbers. For example, `chapter={1-7,10}` means chapters one to seven and chapter ten. Remember to use braces around the value if it contains commas.

`section` Analogous to the `chapter` option, but for sections. As above, the value may be omitted, in which case the current section is assumed, or may be a comma-separated list or range of values.

`byID` The value should be a comma-separated list of IDs identifying the questions whose solutions should be printed. (Recall the `ID` option that can be used when you define a question.)

The solutions are sorted automatically according to their order of definition. If you want to prevent this sorting (so that they are, instead, listed in the order specified in the value of `byID`) you can use the `sorted` boolean key in the `solution` module. The `sorted` key isn't required if the `byID` key isn't used.

Questions can be assigned to a "class", which could represent the difficulty level or similar attribute. As described earlier, this is done via the `class` key when you define a question. You can specify that only questions belonging to a certain class, or list of classes, should be included using the `use-classes` option. For example:

```
\SetupExSheets{use-classes={easy,medium,hard}}
```
Input

indicates to only use those questions that have been assigned to one of the classes: `easy`, `medium` or `hard`. Any questions that haven't been assigned to one of those classes will be discarded.

Similarly, questions can be assigned to a topic using the `topic` key in the optional argument of the question environment. The option `use-topics` is analogous to `use-classes`. There are other commands that allow you to assign properties to questions, but for brevity these are omitted here.

As with the exam class, you can also access the total points and, as before, you need at least two LaTeX runs to get an up-to-date value.

\pointsum Definition

Prints the total number of points (excluding bonus points). This command also has a starred version that omits the "unit", which is "P." by default.

\bonussum Definition

Prints the total number of bonus points. As with the previous command, this command also has a starred version that omits the unit.

\totalpoints Definition

Prints the total number of points (including bonus points). This command also has a starred version that omits the unit.

To typeset a number of points without adding it to the cumulative total use:

\points{⟨*number*⟩} Definition

where ⟨*number*⟩ is the number of points. As above, this command has a starred version that omits the unit.

There are some other related commands described in the exsheets manual [61], but for brevity aren't covered here. The "unit" can be changed via the `name` and `name-plural` keys in the `points` module. Recall from earlier, that the option can be set using:

```
\SetupExSheets[points]{name={point},name-plural={points}}
```
Input

or

```
\SetupExSheets{points/name={point},points/name-plural={points}}
```
Input

There are also options for the bonus point unit: `bonus-name` and `bonus-plural`. For example:

```
\SetupExSheets[points]{bonus-name={bonus point},
   bonus-plural={bonus points}}
```
Input

There are some other options related to the formatting of the points. See the exsheets manual for further details.

There is no provision for multiple choice questions, however you can use the inparaenum environment provided by the paralist package [77].

```
\begin{inparaenum}[⟨format⟩]
⟨list items⟩
\end{inparaenum}
```
Definition

This is analogous to the enumerate environment except that each item doesn't start a new paragraph (unless you explicitly insert a paragraph break).

The optional argument determines the counter format. The tokens A, a, I, i and 1 indicate the counter formats \Alph, \alph, \Roman, \roman and \arabic. (The paralist package also modifies the enumerate environment so that it takes an optional argument that changes the counter format in the same manner.)

Example:

↑ Input

```
\begin{question}
  Which of the following ingredients are used in
  mind-controlling cookies:
  \begin{inparaenum}[(A)]
    \item arsenic
    \item cyanide
    \item curare
    \item\label{correct-ingredient} secret genetically modified
      sugar beet
  \end{inparaenum}
\end{question}
\begin{solution}
Correct choice: \ref{correct-ingredient}.
```

```
\end{solution}
```

Input

You can put all your question and solution environments in an external file to form a databank. These solutions, either all or a subset, can then be included using:

`\includequestions[`⟨*options*⟩`]{`⟨*filenames*⟩`}` Definition

where ⟨*filenames*⟩ is a comma-separated list of filenames. Note that the exsheets documentation comes with a caveat that this command is experimental. The optional argument is a key=value list. Available options are:

`all` This is a boolean key. If true, all questions are selected.

`IDs` Only those solutions whose ID is contained in the list of IDs. Since this value contains commas, remember to enclose the list with braces.

`random` The value should be a number, ⟨*n*⟩, indicating that ⟨*n*⟩ questions should be randomly selected.

`exclude` Exclude any questions whose ID is contained in this list. Again, since the value contains commas, remember to enclose the list with braces. This option can be combined with the `random` option.

Exercise 25. Creating an Exam Paper with the exsheets Package
Try rewriting Exercise 24 using the exsheets package instead of the exam class. (You may not be able to automatically implement some features, such as the grading table.) You can download or view a solution.

9.3 The probsoln Package

The probsoln package [93] provides a means to define problems with their associated solution. These definitions may be placed in an external `.tex` file. You can then load all problems or a subset of problems (possibly randomly selected) into a dataset. You can have more than one dataset, each of which could, for example, represent a topic. In your document you can iterate through these datasets and display the problem, the solution or both. This means that you can gather the solutions together in another part of the document. Since probsoln is a package, you need to find an appropriate document class.

The probsoln package has the following options:

`answers`	Show the solutions.
`noanswers`	Hide the solutions (default).

`draft`	Display the problem label and dataset name when a problem is used.
`final`	Don't display the problem label and dataset name (default).
`usedefaultargs`	Make `\thisproblem` use the default arguments supplied with the problem definition.
`nousedefaultargs`	Make `\thisproblem` prompt for arguments (default).

The last two options will be described in more detail below. Remember that any options specified in `\documentclass` are also passed to packages, so if you use the `draft` class option, it will automatically enable probsoln's `draft` option, unless you have explicitly used probsoln's `final` option.

At the time of writing the current version of probsoln is version 3.04 (2012-08-23). Some of the features described here aren't available for earlier versions. Since defining the problems and their solutions requires either a command or an environment that gathers its contents, verbatim code requires special care. To allow for verbatim text, the probsoln package provides the `fragile` boolean key that can be used when defining a problem. If the majority of your problems require this option, you can set it using:

\setkeys{probsoln}{fragile} Input

(This command is provided by the keyval package [12], which is automatically loaded.)

In order to work with verbatim code, the probsoln package creates a temporary file that's used when the fragile option is set. The default name for this file is \jobname.vrb but if this conflicts with another package, you can change the extension by redefining

\ProbSolnFragileExt Definition

You can also change the basename by redefining

\ProbSolnFragileFile Definition

In addition to the answers and noanswers options, you can also show or suppress solutions using

\showanswers Definition

(to show the solutions) and

\hideanswers Definition

(to hide the solutions). These are declarations that can be scoped by placing them within a group.

You can check if the show solutions setting is on or off by testing the showanswers boolean flag. This can be done using:

\ifshowanswers ⟨*true part*⟩\else ⟨*false part*⟩\fi Definition

or you can use the \ifthenelse command provided by the ifthen package:

\ifthenelse{\boolean{showanswers}}{⟨*true part*⟩}{⟨*false part*⟩} Definition

or you can use the \ifbool command provided by the etoolbox package:

\ifbool{showanswers}{⟨*true part*⟩}{⟨*false part*⟩} Definition

For example:

```
Assignment 1\ifbool{showanswers}{ (Solution Sheet)}{}
```
Input

For longer text, you can use the environments onlyproblem

\begin{onlyproblem}[⟨*option*⟩]
⟨*text*⟩
\end{onlyproblem} Definition

to only display ⟨*text*⟩ if the solutions are suppressed, and onlysolution

\begin{onlysolution}[⟨*option*⟩]
⟨*text*⟩
\end{onlysolution} Definition

to only display ⟨*text*⟩ if the solutions are displayed. In both cases, the optional argument ⟨*option*⟩ may be the `fragile` boolean key, described above.

Example:

Input

```
\begin{onlyproblem}
What is the main drawback of ray guns?
\end{onlyproblem}
\begin{onlysolution}
Overheating.
\end{onlysolution}
```

Input

If the solutions are displayed, only the solution ("Overheating.") will be typeset, otherwise only the question ("What is the main drawback of ray guns?") will be typeset.

Note that spaces at the start of the environment are discarded but spaces at the end of the environment aren't. So the EOL character immediately before "What" is discarded (and similarly for the EOL character immediately before "Overheating") but the EOL character after the question mark is interpreted as a space (and similarly for the EOL character after the full

stop in the solution). If these spaces are unwanted, you can suppress them with the % comment character:

Input

```
\begin{onlyproblem}
What is the main drawback of ray guns?%
\end{onlyproblem}%
\begin{onlysolution}
Overheating.%
\end{onlysolution}
```

Input

If you want the question to always appear, regardless of the showanswers flag, then don't put it inside the onlyproblem environment:

Input

```
What is the main drawback of ray guns?
\begin{onlysolution}
Overheating.%
\end{onlysolution}
```

Input

You may prefer to put the solution inside the solution environment:

```
\begin{solution}⟨text⟩\end{solution}
```
Definition

which is equivalent to:

```
\par\noindent\textbf{\solutionname}: ⟨text⟩
```
Input

where \solutionname defaults to “Solution”.

Example:

```
What is the main drawback of ray guns?
\begin{onlysolution}
\begin{solution}
Overheating.
\end{solution}
\end{onlysolution}
```
↑ Input

↓ Input

The combination of the onlysolution and solution environments in this manner is analogous to the exam class’s solution environment. Since it’s possible that you may want to use the probsoln package with the exam class, if probsoln detects that the solution environment is already defined, it doesn’t try

defining its own solution environment. This means that if the above example was used with both the exam class and the probsoln package, there's a level of redundancy with the exam class performing a check for its own show/hide solution setting in its solution environment and the probsoln package performing a similar check for its showanswers flag in its onlysolution environment. Therefore, if you are using both exam and probsoln, I recommend you redefine the solution environment in a manner similar to probsoln's solution environment and just use probsoln's showanswers flag, as shown in Example 44.

The probsoln package provides an inline numbered list environment called textenum:

\begin{textenum}⟨*items*⟩\end{textenum} Definition

This uses the same counter and label as the enumerate environment would use at the current level. For example, if the textenum environment was used outside the enumerate environment, the enumi counter will be used with \labelenumi, but if textenum was used inside a single enumerate environment, the enumii will be used with \labelenumii.

As with the standard list environments, each item is started with \item but no paragraph break is inserted unless you explicitly add one (either via a blank line or \par). You can also use

\correctitem Definition

in place of \item to indicate a correct choice and

\incorrectitem Definition

in place of \item to indicate an incorrect choice. If the solutions are suppressed, these two commands behave the same as \item. When the solutions are displayed, the default behaviour of \correctitem is to put a frame around the item marker, and the default behaviour of \incorrectitem is to just display the marker (as per \item). You can change this by redefining

\correctitemformat{⟨*marker*⟩} Definition

which governs the format used by \correctitem, and

\incorrectitemformat{⟨*marker*⟩} Definition

which governs the format used by \incorrectitem.

Example 44. Using Both the exam Class and the probsoln Package
This example uses the exam class with the probsoln package. The exam class's solution environment is redefined to make it more like the probsoln package's solution environment. The \ignorespaces and \ignorespacesafterend commands (introduced in Volume 1 [92, §10]) suppress any spaces following the start and end of the environment that may be introduced by a spurious EOL

character. The \noindent command (also introduced in Volume 1 [92, §10]) suppresses the paragraph indentation.

With this redefinition of the solution environment, the exam class's \printanswers command no longer has an effect. Instead the probsoln package's \showanswers command should be used.

Since the textenum environment is an inline list, its items may be placed inside a tabular environment, as is done in the example document below.

↑ Input

```
\documentclass[addpoints]{exam}

\usepackage{probsoln}

\showanswers

\renewenvironment{solution}%
{\par\noindent\textbf{\solutionname}: \ignorespaces}%
{\ignorespacesafterend}

\renewcommand*{\theenumi}{\Alph{enumi}}

\begin{document}
```

```
\begin{center}\bfseries
Assignment~1\ifshowanswers\space (Solution Sheet)\fi
\end{center}

\begin{questions}
\question[1] What is the main drawback of ray guns?
\begin{onlysolution}
\begin{solution}
Overheating.
\end{solution}
\end{onlysolution}

\question[1] Which of the following is an ingredient of
mind-controlling cookies?
\begin{center}
 \begin{textenum}
  \begin{tabular}{ll}
  \incorrectitem arsenic &
  \incorrectitem cyanide \\
  \incorrectitem curare &
  \correctitem secret genetically modified sugar beet
  \end{tabular}
```

```
 \end{textenum}
\end{center}
\end{questions}

\end{document}
```

↓ Input

The resulting text is:

↑ Output

Assignment 1 (Solution Sheet)

1. (1 point) What is the main drawback of ray guns?

   Solution: Overheating.

2. (1 point) Which of the following is an ingredient of mind-controlling cookies?

   A. arsenic B. cyanide
   C. curare D. secret genetically modified sugar beet

↓ Output

You can download or view this example.

Thus far I haven't shown you anything that the exam class can't already do, so let's now look at how to define problems and their associated solutions for later use. A problem can be defined using the defproblem environment

```
\begin{defproblem}[⟨n⟩][⟨default args⟩]{⟨label⟩}[⟨option⟩]
⟨text⟩
\end{defproblem}
```

Definition

where ⟨*text*⟩ may include the onlyproblem or onlysolution environments (or a combination of both). The first optional argument ⟨*n*⟩ is the number of arguments this problem may take. Each argument may be referenced in ⟨*text*⟩ using the standard #1 etc method of referencing an argument. The second optional argument ⟨*default args*⟩ are the default arguments to use with this problem when it is automatically used by \thisproblem, described below. (The default argument is ignored when the problem is referenced with \useproblem, which must have the arguments explicitly added.)

If ⟨*n*⟩ is omitted, 0 is assumed and ⟨*default args*⟩ should also be omitted. The final optional argument ⟨*option*⟩ may be used to specify the `fragile` boolean key described earlier. If ⟨*text*⟩ contains any instances of the on-

lyproblem or onlysolution environments, they will inherit the `fragile` state from defproblem.

The only mandatory argument is ⟨*label*⟩, which is a label that uniquely identifies this problem so that it can later be referenced. As with all the other labelling systems described in this book, the label must not contain any special characters. The problem must be defined before use, typically either in the preamble or in an external `.tex` file.

Example:
Here's a simple example that doesn't require any arguments:

↑ Input

```
\begin{defproblem}{raygun}
 \begin{onlyproblem}
 What is the main drawback of ray guns?%
 \end{onlyproblem}%
 \begin{onlysolution}
 Overheating.%
 \end{onlysolution}
\end{defproblem}
```

↓ Input

Here's an example that requires one argument. This argument defaults to 2. Note that the braces { } are required for each argument.

Input

```
\begin{defproblem}[1][{2}]{diffsin}
 \begin{onlyproblem}
 Differentiate $f(x) = \sin(#1x)$.
 \end{onlyproblem}%
 \begin{onlysolution}
 $f'(x) = #1\cos(#1x)$
 \end{onlysolution}
\end{defproblem}
```

Input

In both of the above examples, I'm assuming that the solutions will be printed later in the document, separate from the question, so I haven't bothered to use the solution environment, since the "Solution:" tag is now redundant. If, on the other hand, I want a solution sheet that displays the solution with its associated question, then it's better to remove the onlyproblem environment and add the solution environment inside the onlysolution environment:

Input

```
\begin{defproblem}{raygun}
 What is the main drawback of ray guns?
 \begin{onlysolution}
  \begin{solution}
  Overheating.%
  \end{solution}
 \end{onlysolution}
\end{defproblem}
```

Input

Since it's quite cumbersome having to write so many \begin and \end commands, the probsoln package provides a convenient shortcut command:

\newproblem[⟨n⟩][⟨*default args*⟩]{⟨*label*⟩}{⟨*problem*⟩}{⟨*solution*⟩} Definition

This is equivalent to:

Input

```
\begin{defproblem}[⟨n⟩][⟨default args⟩]{⟨label⟩}%
 ⟨problem⟩%
 \begin{onlysolution}%
```

```
 \begin{solution}%
 ⟨solution⟩%
 \end{solution}%
 \end{onlysolution}%
\end{defproblem}
```

↓ Input

There is also a starred version:

\newproblem*[⟨n⟩][⟨default args⟩]{⟨label⟩}{⟨problem⟩} Definition

which is a shortcut for:

↑ Input

```
\begin{defproblem}[⟨n⟩][⟨default args⟩]{⟨label⟩}%
 ⟨problem⟩%
\end{defproblem}
```

↓ Input

Note that both versions of \newproblem don't support verbatim, so if your problem contains verbatim text you must use the defproblem environment (with the fragile key set).

Once you have defined your problems, you can the use them in your document. You can explicitly use a particular problem via the command:

\useproblem[⟨*dataset*⟩]{⟨*label*⟩}{⟨*arg*$_1$⟩}...{⟨*arg*$_N$⟩} Definition

where ⟨*label*⟩ is the label uniquely identifying the problem. If the problem was defined to have arguments, the arguments must then follow the label. The optional argument ⟨*dataset*⟩ indicates the dataset in which the problem is stored. If omitted the default dataset is assumed.

Example:
Given the definitions in the earlier example of the problems with the labels `raygun` and `diffsin`, these can now be used in the document:

↑ Input

```
\begin{enumerate}
\item \useproblem{raygun}

\item \useproblem{diffsin}{3}
\end{enumerate}
```

↓ Input

This is on the assumption that the problems were defined in the document preamble, which will automatically place them in the default dataset. Since the `diffsin` problem was defined to have one argument, that argument has to be provided.

If you define all your problems in external files, you can input each file using LaTeX's standard \input command, which will have the same effect as just defining all those problems within the document. Alternatively, you can load the problems into a particular dataset using one of the commands described below, where ⟨*dataset*⟩ is the name of the dataset and ⟨*filename*⟩ is the name of the file. If the dataset is omitted, the default dataset is assumed. Note that the dataset name mustn't contain any special characters.

\loadallproblems[⟨*dataset*⟩]{⟨*filename*⟩} Definition

This loads all the problems defined in the given ⟨*filename*⟩ and adds them to the ⟨*dataset*⟩ indicated in the optional argument.

\loadselectedproblems[⟨*dataset*⟩]{⟨*labels*⟩}{⟨*filename*⟩} Definition

This only defines the problems listed in the comma-separated list ⟨*labels*⟩. The other problems in ⟨*filename*⟩ are ignored.

\loadexceptproblems[⟨*dataset*⟩]{⟨*exception list*⟩}{⟨*filename*⟩} Definition

This is the reverse of \loadselectedproblems. It only defines the problems that aren't listed in the comma-separated list ⟨*exception list*⟩.

\loadrandomproblems[⟨dataset⟩]{⟨n⟩}{⟨filename⟩} Definition

This loads ⟨n⟩ randomly selected problems defined in ⟨filename⟩ and adds them to the given dataset.

\loadrandomexcept[⟨dataset⟩]{⟨n⟩}{⟨filename⟩}{⟨exception list⟩} Definition

Similar to the previous command but discounts the problems listed in ⟨exception list⟩ when making the random selection.

Once you have loaded your problems, using one of the above commands, you can iterate through a dataset using:

\foreachproblem[⟨dataset⟩]{⟨body⟩} Definition

This does ⟨body⟩ at each iteration. Within ⟨body⟩ you can use

\thisproblem Definition

to use the current problem and

\thisproblemlabel Definition

to access the current problem label. Unlike \useproblem, you don't supply the problem arguments when you use \thisproblem. If the problem requires one or more arguments and no default arguments were provided when the problem was defined or the usedefaultargs package option

wasn't used, then you will be prompted for the arguments, which requires LaTeX to be run in interactive mode. (Interactive mode is when LaTeX stops on encountering an error and prompts you for a response.)

Example:
To iterate through all problems in the default dataset:

Input

```
\begin{enumerate}
\foreachproblem{\item\thisproblem}
\end{enumerate}
```

Input

Assuming that you haven't switched on the solutions using \showanswers or the answers package option, this will just list all the problems. If you switch on the solutions, this will include the solutions but omit any text inside the onlyproblem environment.

There is also a similar command:

\foreachsolution[⟨*dataset*⟩]{⟨*body*⟩} Definition

which behaves like \foreachproblem but only iterates over those problems that contain an onlysolution environment. (You must have first used the problems earlier in the document.) Note that you still need to switch on

the show solution flag using \showanswers or the answers package option if you want the solutions displayed.

\foreachdataset{⟨*cs*⟩}{⟨*body*⟩} Definition

This iterates over all the defined datasets and does ⟨*body*⟩ at each iteration. Within ⟨*body*⟩ you can use the control sequence ⟨*cs*⟩ to access the current dataset name.

For example, to display all problems in all datasets:

Input

```
\begin{enumerate}
\foreachdataset{\thisdataset}%
{%
  \foreachproblem[\thisdataset]{\item\thisproblem}%
}
\end{enumerate}
```

Input

Example:
Suppose I have some calculus problems defined in a file called `calculus.tex` and I have some linear algebra problems defined in a file called `linearalgebra.tex`, then in my document preamble I could, say, load five

randomly selected calculus problems and four randomly selected linear algebra problems using:

Input

```
\loadrandomproblems[calculusproblems]{calculus}
\loadrandomproblems[linearalgebraproblems]{linearalgebra}
```

Input

(The .tex extension may be omitted.) This creates two datasets with the labels calculusproblems and linearalgebraproblems. In my document, I could simply iterate over all datasets using \foreachdataset as described above.

Input

```
\begin{enumerate}
\foreachdataset{\thisdataset}%
{%
  \foreachproblem[\thisdataset]{\item\thisproblem}%
}
\end{enumerate}
```

Input

Alternatively, I might want to divide the document into sections for each topic:

Input

```
\section{Calculus}
\begin{enumerate}
\foreachproblem[calculusproblems]{\item\thisproblem}
\end{enumerate}

\section{Linear Algebra}
\begin{enumerate}
\foreachproblem[linearalgebraproblems]{\item\thisproblem}
\end{enumerate}
```

Input

The seed used by the pseudo random number generator can be changed using:

`\PSNrandseed{⟨n⟩}` Definition

where ⟨n⟩ is a non-zero number to use as the seed. Random numbers can be generated using:

\PSNrandom{⟨register⟩}{⟨n⟩} Definition

which generates a random integer between 1 and ⟨n⟩ (inclusive) and stores it in the given count *register* (see §2.1.3) or

\random{⟨counter⟩}{⟨min⟩}{⟨max⟩} Definition

which generates a random integer between ⟨min⟩ and ⟨max⟩ (inclusive) and stores it in the LaTeX *counter* whose name is ⟨counter⟩. (See also §9.5 below.)

Example:
Recall the earlier `diffsin` problem. Suppose that instead of defining the problem to have an argument for the factor, the factor is randomly selected instead. First a new register needs to be defined:

```
\newcount\myrandarg
```
Input

Now the problem can be defined:

↑ Input

```
\begin{defproblem}{randdiffsin}
 \PSNrandom{\myrandarg}{10}%
 Differentiate $f(x) = \sin(\the\myrandarg x)$.
```

```
 \begin{onlysolution}
 $f'(x) = \the\myrandarg\cos(\the\myrandarg x)$
 \end{onlysolution}
\end{defproblem}
```

Input

If the random number seed is set to the current year:

```
\PSNrandseed\year
```

Input

then this problem will appear differently if the same document is rebuilt on different years.

⚠ If you want the solutions to appear in a different location to the questions (for example, at the end of the document) you'll need to store the randomly generated number to prevent a different number from being generated in the solution section. For example:

Input

```
\begin{defproblem}{randdiffsin}
 \ifundef\randdiffsinarg
 {%
  \PSNrandom{\myrandarg}{10}%
  \xdef\randdiffsinarg{\the\myrandarg}% globally store
```

```
 }
 {}%
 Differentiate $f(x) = \sin(\randdiffsinarg x)$.
 \begin{onlysolution}
 $f'(x) = \randdiffsinarg\cos(\randdiffsinarg x)$
 \end{onlysolution}
\end{defproblem}
```

↓ Input

(See also §9.5.)

You can iterate over ⟨*n*⟩ randomly selected items in a comma-separated list using:

`\doforrandN{`⟨*n*⟩`}{`⟨*cs*⟩`}{`⟨*list*⟩`}{`⟨*body*⟩`}` Definition

This performs ⟨*body*⟩ at each iteration where ⟨*cs*⟩ is a control sequence set to the currently selected item.

Example:
Suppose you have four files called `file1.tex`, `file2.tex`, `file3.tex` and `file4.tex` that all contain problem definitions, but you only want to load the problems defined in two of those files selected at random:

Input

```
\doforrandN
 {2}% randomly select 2 items from the list
 {\thisfile}% command in which to store the current item
 {file1,file2,file3,file4}% the list
 {\loadallproblems{\thisfile}}
```

Input

Example:
In this example, only one random selection is made and the selected item is saved for later use:

Input

```
\doforrandN{1}{\thisitem}{cow,duck,chicken}
  {\global\let\thesubject\thisitem}%
\doforrandN{1}{\thisitem}{road,field,river}
  {\global\let\theobject\thisitem}%
Why did the \thesubject\␣cross the \theobject?
\begin{onlysolution}
 \begin{solution}
```

```
 So that the \thesubject\␣could get to the other side of the
\theobject.
 \end{solution}
\end{onlysolution}
```

↓ Input

The previous warning also applies here if you intend to display the solutions in another part of the document. In this case, a similar approach can be used:

↑ Input

```
\begin{defproblem}{whycross}
 \ifdef{\thesubject}
 {}% already defined
 {%
  \doforrandN{1}{\thisitem}{cow,duck,chicken}
    {\global\let\thesubject\thisitem}%
  \doforrandN{1}{\thisitem}{road,field,river}
    {\global\let\theobject\thisitem}%
 }%
 \begin{onlyproblem}
  Why did the \thesubject\␣cross the \theobject?
```

```
 \end{onlyproblem}
 \begin{onlysolution}
 So that the \thesubject\␣could get to the other side of the
\theobject.
 \end{onlysolution}
\end{defproblem}
```

↓ Input

This uses the \ifdef command provided by the etoolbox package described in §2.1.1.

Note that the pgfmath package also provides pseudo-random commands, which you may prefer to use. Some of these are described in §9.5.

Example 45. Randomly Selecting Problems

In this example, I have a number of files containing problem definitions:

1. mth101.tex

   This file contains 10 easy differentiation problems in the form:

↑ Input

```
\begin{defproblem}{diff:sinx/x}
 \begin{onlyproblem}%
```

```
 $y = \frac{\sin x}{x}$.
 \end{onlyproblem}
 \begin{onlysolution}%
 \[\frac{dy}{dx} = \frac{\cos x}{x} - \frac{\sin x}{x^2}\]
 \end{onlysolution}%
\end{defproblem}
```

Input

You can download the complete file.

2. problems-1stprinciples.tex

   This file contains 5 differentiation from first principle problems in the form:

Input

```
\begin{defproblem}{dfp:cons}%
 \begin{onlyproblem}%
  Differentiate from first principles $f(x) = c$ where
$c$ is a constant.%
 \end{onlyproblem}%
```

```
 \begin{onlysolution}%
 \begin{align*}
 \frac{df}{dx} & =
\lim_{\Delta x\rightarrow 0}\frac{c-c}\Delta x\\
  & = \lim_{\Delta x\rightarrow 0}0\\
  & = 0
 \end{align*}%
 \end{onlysolution}%
\end{defproblem}
```

Input

You can download the complete file.

I can now write a document that randomly selects 3 problems from the first file and 1 problem from the second file. The questions are listed first and the solutions later:

Input

```
\documentclass{article}
\usepackage{amsmath}
\usepackage{probsoln}
```

```
\loadrandomproblems{3}{mth101}
\loadrandomproblems{1}{problems-1stprinciples}

\begin{document}
\section{Differentiation Problems}
\begin{enumerate}
\foreachproblem{\item\thisproblem}
\end{enumerate}

\section{Solutions}
\showanswers
\begin{enumerate}
\foreachsolution{\item\thisproblem}
\end{enumerate}
\end{document}
```

↓ Input

You can download or view this example document.

9.4 Using the datatool Package for Exams or Assignment Sheets

Since the datatool package and the datatooltk application have already been described in this book, it's worth mentioning that they can also be used to store a database of problems and their associated solutions. This can be done by creating a database with a label field, a question field and an answer field. Other fields can also be added to store, for example, the topic or level of difficulty.

If you already have a file containing probsoln problem definitions, datatooltk can convert it to a datatool database.[1] For example, the mth101.tex file from Example 45 can be imported either using the --probsoln command line option or the File→Import→Import probsoln file menu item in the GUI mode. Figure 9.1 shows the mth101.tex file imported into datatooltk. Since LaTeX is used to assist the conversion, the "pretty-printing" of the code has unfortunately been lost, but this won't affect the typeset output. (This also happens if you use \DTLsaverawdb or \DTLprotectedsaverawdb.)

The import process has created three fields: Label, Question and Answer. Extra fields can be added using the Edit→Column→Insert Column After menu item. For example, in Figure 9.2, I've added a new integer field called Level, where a value of 1 indicates easy, 2 indicates medium difficulty and

[1] You can't export back to the probsoln format.

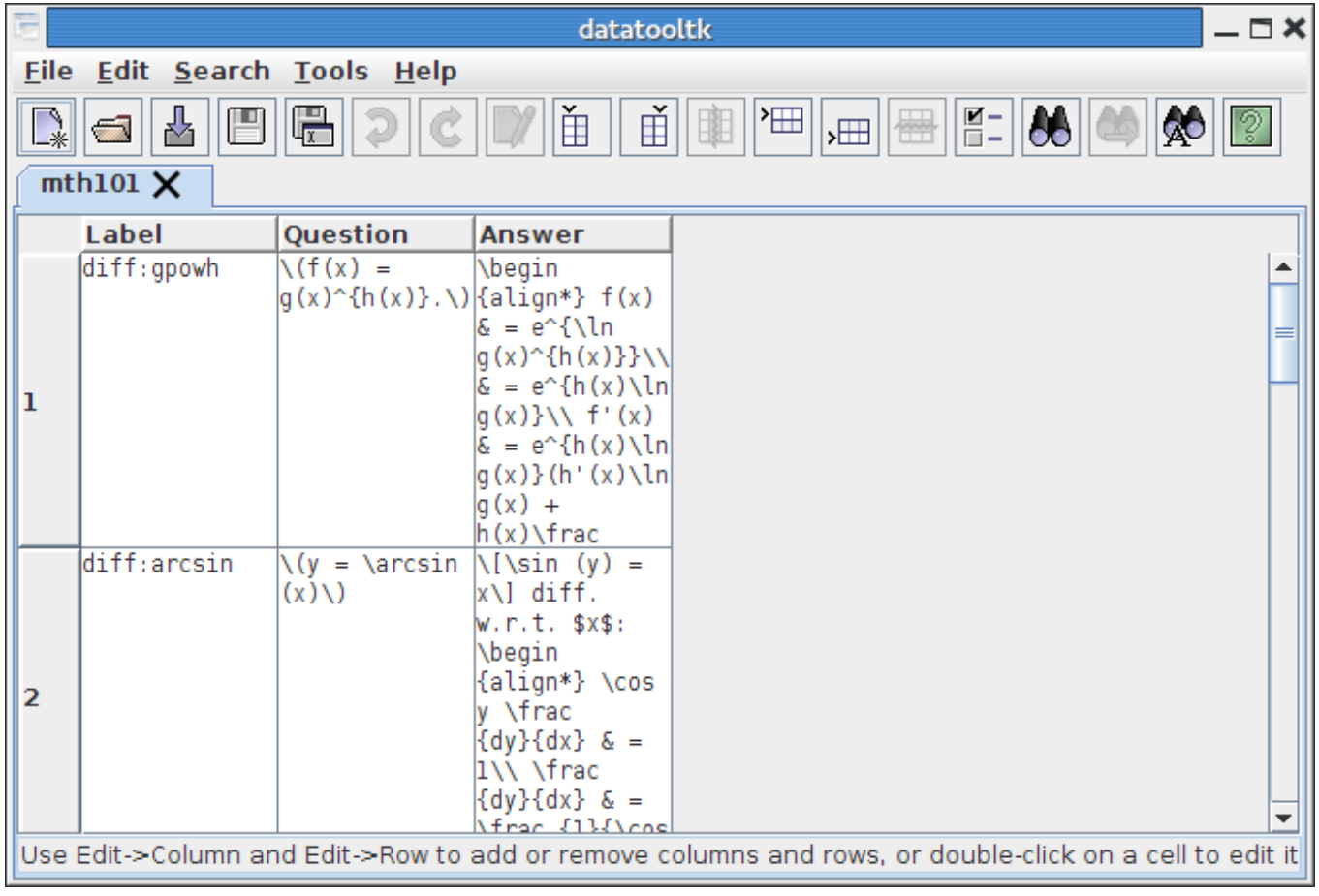


Figure 9.1 Importing a probsoln Dataset into `datatooltk`

3 indicates hard. This database can then be saved as, say, `mth101.dbtex` and loaded into a document using `\DTLloaddbtex`, as described in §2.2.2. You can add other columns as well, such as a topic.

Note that datatool has a drawback that probsoln doesn't have, and that is the lack of support for verbatim. You can, however, use `\lstinputlisting` (provided by the listings package [33], described in Volume 2 [95, §4.5]) or `\verbatiminput` (provided by the verbatim package [84]).

A new boolean variable can be defined using:

`\newboolean{`⟨*name*⟩`}` Definition

defined by the ifthen package, or

`\newbool{`⟨*name*⟩`}` Definition

defined by the etoolbox package, where ⟨*name*⟩ is the name of the variable. (Note that ⟨*name*⟩ is not a control sequence.) The state can be set using:

`\setboolean{`⟨*name*⟩`}{`⟨*state*⟩`}` Definition

defined by the ifthen package, or

`\setbool{`⟨*name*⟩`}{`⟨*state*⟩`}` Definition

defined by the etoolbox package, where ⟨*state*⟩ may be either `true` or `false`. With the etoolbox package, you can also use:

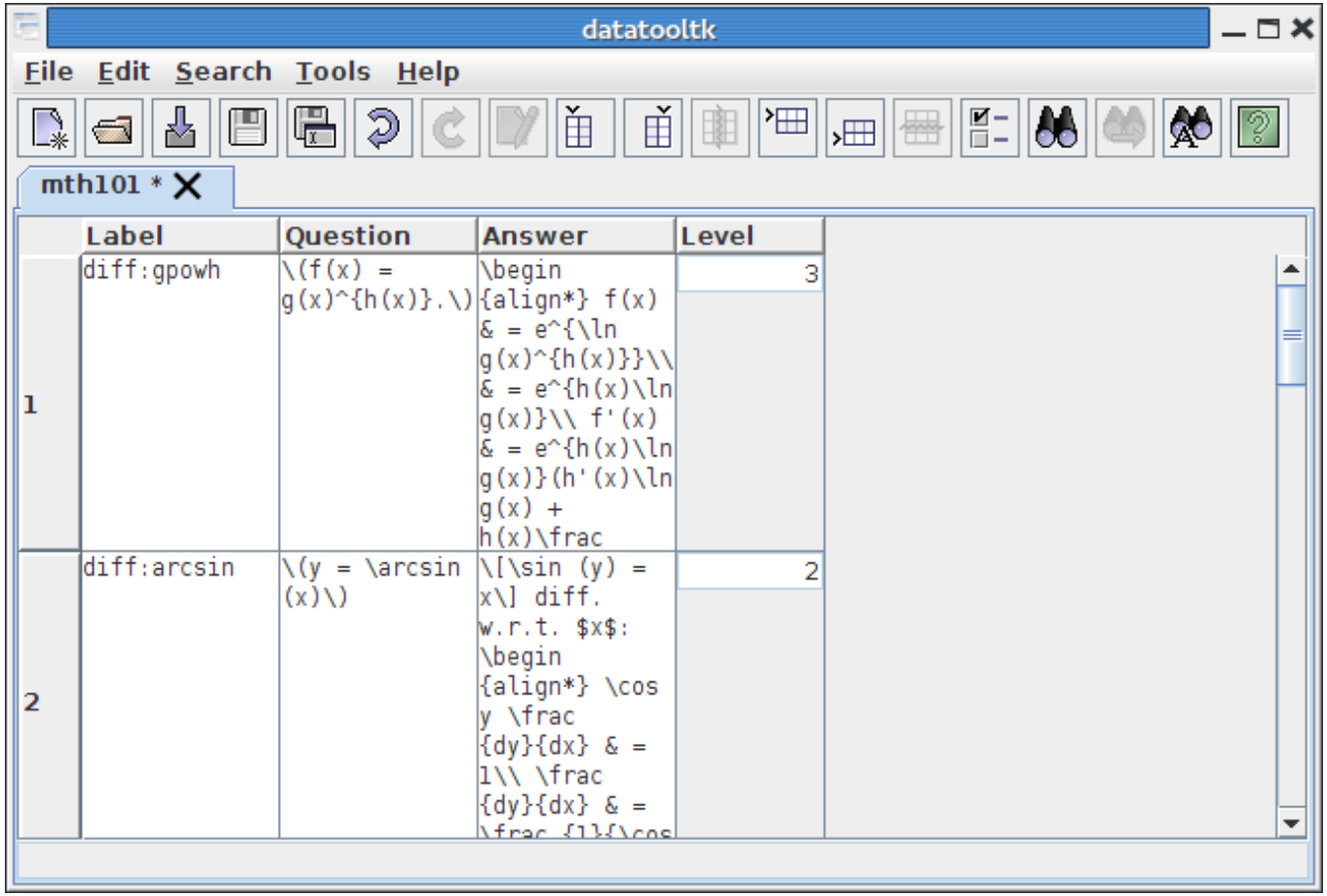


Figure 9.2 New Level Column Added

\boolfalse{⟨name⟩} Definition

to set the state to false or

\booltrue{⟨name⟩} Definition

to set the state to true.

The variable's state can be tested using:

\ifthenelse{\boolean{⟨name⟩}}{⟨true⟩}{⟨false⟩} Definition

defined by the ifthen package, or

\ifbool{⟨name⟩}{⟨true⟩}{⟨false⟩} Definition

defined by the etoolbox package.

Note that \newboolean and \newbool both use the same underlying TEX command to define a conditional so they have the same effect. The etoolbox \setbool can be prefixed with \global but ifthen's \setboolean can't.

It's therefore possible to define your own boolean flag that determines whether or not the solutions should be displayed.

Example 46. Creating a Problem Sheet using datatool

Returning to the database shown in Figure 9.2. Suppose that database is saved as mth101.dbtex. Now it can be loaded and iterated over to display

all the questions:

↑ Input

```
\documentclass{article}

\usepackage{etoolbox}
\usepackage{datatool}

\newbool{showanswers}
\booltrue{showanswers}

\DTLloaddbtex{\problemDB}{mth101.dbtex}

\begin{document}
\begin{center}\bfseries\Large
Assignment~1\ifbool{showanswers}{ (Solution Sheet)}{}
\end{center}

\begin{enumerate}
\DTLforeach*{\problemDB}
 {\Label=Label,\Question=Question,\Answer=Answer}%
 {%
```

```
   \item \Question
   \ifbool{showanswers}{\par\textbf{Solution: }\Answer}{}%
 }
\end{enumerate}

\end{document}
```

↓ Input

You can download or view this example document.

Alternatively, you could gather all the solutions at the end of the document:

↑ Input

```
\documentclass{article}

\usepackage{etoolbox}
\usepackage{datatool}

\newbool{showanswers}
\booltrue{showanswers}

\DTLloaddbtex{\problemDB}{mth101.dbtex}
```

```
\begin{document}
\begin{center}\bfseries\Large
Assignment~1
\end{center}

\begin{enumerate}
\DTLforeach*{\problemDB}
 {\Label=Label,\Question=Question}%
 {%
   \item \Question
 }
\end{enumerate}

\ifbool{showanswers}
{%
 \section{Solutions}

\begin{enumerate}
\DTLforeach*{\problemDB}
 {\Label=Label,\Answer=Answer}%
 {%
```

```
   \item \Answer
 }
\end{enumerate}

}{}
\end{document}
```

↓ Input

You can, of course, use the exam class or probsoln package with datatool. That way you don't need to define your own boolean variable.

It may, however, be that you only want a random selection of the questions from the database. While this could be done within the document using commands provided by the datatool package, it's more efficient to do this using datatooltk. That way, the random selection only needs to be done once per problem sheet (possibly repeated after any modifications to the database) which reduces the time taken for TEX to compile the document. The datatooltk has a number of command line options that can help with this:

- `--shuffle`

  Shuffle the rows in the database.

- `--seed` ⟨*number*⟩

  Set the random generator seed to ⟨*number*⟩.

- `--shuffle-iterations` ⟨*number*⟩

  Sets the number of iterations performed in the shuffle to ⟨*number*⟩

- `--truncate` ⟨*number*⟩

  Truncate the database to the first ⟨*number*⟩ rows. (This option is always performed after the shuffle option, regardless of the option order.)

- `--filter` ⟨*key*⟩ ⟨*operator*⟩ ⟨*value*⟩

  Adds a filter. This option may be used multiple times. Here ⟨*key*⟩ is the column label used by the filter. The ⟨*operator*⟩ may be one of: `eq` (equals), `ne` (does not equal), `le` (is less than or equal to), `lt` (is less than), `ge` (is greater than or equal to), `gt` (is greater than) or `regex` (matches the regular expression). In the last case, ⟨*value*⟩ should be a regular expression as used by `java.util.regex.Pattern`. In the other cases, ⟨*value*⟩ may be an integer, real number or string. If the datatype for the column identified by ⟨*key*⟩ is numerical and ⟨*value*⟩ is also numerical, then a numerical comparison is used, otherwise

a string comparison is used. For example, `--filter Level le 2` indicates that the filter should return a true value for any row where the value in the `Level` column is less than or equal to 2.

Filtering is always applied after shuffling and before truncating (if either of those options have been specified).

- `--filter-and`

  The default action in the event of multiple `--filter` options is to apply a logical "or". The `--filter-and` changes this behaviour to apply a logical "and" to all the filter results instead. For example, suppose the database also has a column labelled `Topic` and you want to select five easy questions from the topic "Algebra", then you need a logical "and":

```
datatooltk --in mth101.dbtex --shuffle --filter-and
--filter Level eq 1 --filter Topic eq Algebra
--truncate --output problems.dbtex
```

Shell

- `--filter-exclude`

When applying any filters, the `--filter-exclude` option will cause any matching rows to be excluded. (The default behaviour is to exclude non-matching rows.)

- `--merge` ⟨*col-label*⟩ ⟨*filename*⟩

  Merges the loaded database with the database in the file whose name is given by ⟨*filename*⟩. The merge is performed by merging each row in ⟨*filename*⟩ with the row in the database where the column given by the label ⟨*col-label*⟩ has the same value as the column with the same label in ⟨*filename*⟩. If no match is found, a new row is added.

With a combination of these options, it's possible to create a database file (called, say, `problems.dbtex`) that only contains a random subset of the complete database.

Examples:

1. Select five questions (of any level) at random:

```
datatooltk --in mth101.dbtex --shuffle --truncate 5 ↩
--output problems.dbtex
```

Shell

2. Select two level 1 questions at random:

```
datatooltk --in mth101.dbtex --shuffle
--filter Level eq 1 --truncate 5
--output problems.dbtex
```

Shell

3. Select four non-easy questions at random with the seed set to 2014:

```
datatooltk --in mth101.dbtex --shuffle --seed 2014
--filter Level ne 1 --truncate 4
--output problems.dbtex
```

Shell

The document from Example 46 just needs one line changed, and that's the line that loads the database:

```
\DTLloaddbtex{\problemDB}{problems.dbtex}
```

Input

Alternatively, if you want, say, four level 1 questions, two level 2 questions and one level 3 question, you can create three separate databases:

```
datatooltk --in mth101.dbtex --shuffle --filter Level eq 1
--truncate 4 --output problems1.dbtex
datatooltk --in mth101.dbtex --shuffle --filter Level eq 2
--truncate 2 --output problems2.dbtex
datatooltk --in mth101.dbtex --shuffle --filter Level eq 3
--truncate 1 --output problems3.dbtex
```

Shell

Now you need to load all three databases into your document:

Input

```
\DTLloaddbtex{\problemDBi}{problems1.dbtex}
\DTLloaddbtex{\problemDBii}{problems2.dbtex}
\DTLloaddbtex{\problemDBiii}{problems3.dbtex}
```

Input

and iterate over each of them:

Input

```
\begin{enumerate}
\DTLforeach*{\problemDBi}
 {\Label=Label,\Question=Question,\Answer=Answer}%
```

```
 {%
   \item \Question
   \ifbool{showanswers}{\par\textbf{Solution: }\Answer}{}%
 }
\DTLforeach*{\problemDBii}
 {\Label=Label,\Question=Question,\Answer=Answer}%
 {%
   \item \Question
   \ifbool{showanswers}{\par\textbf{Solution: }\Answer}{}%
 }
\DTLforeach*{\problemDBiii}
 {\Label=Label,\Question=Question,\Answer=Answer}%
 {%
   \item \Question
   \ifbool{showanswers}{\par\textbf{Solution: }\Answer}{}%
 }
\end{enumerate}
```

↓ Input

If you do intend to do this, I suggest you define a command to perform these iterations. For example:

↑ Input

```
\newcommand{\doquestions}[1]{%
  \DTLforeach*{#1}
   {\Label=Label,\Question=Question,\Answer=Answer}%
   {%
     \item \Question
     \ifbool{showanswers}{\par\textbf{Solution: }\Answer}{}%
  }%
}
```

Input

If the original database contains, say, two hundred problems, using datatooltk in this way can significantly speed up the document build. Each year you can run the datatooltk commands with a different random generator seed to produce a new assignment sheet or exam paper.

If you prefer to store your problems in a SQL database, you can perform the random selection with the SELECT statement. For example, if the problems are stored in a table called calculus within a database called mth101, then you can select, say, five questions at random using:

```
datatooltk --output problems.dbtex --sqluser username
--sqldb mth101 --sql "SELECT * FROM calculus ORDER BY RAND()
LIMIT 5"
```

Shell

What if you don't want to select any problems that appeared in the exam paper or assignment sheet in, say, the previous two years? You could add a year column to the original complete database, but this can be tiresome and prone to error if done manually. It could possibly be done by the LaTeX document, but this would require loading the entire database and saving it using `\DTLsaverawdb`, which means it's pointless using the `datatooltk` options described above and, as noted earlier, you'd lose any pretty-printing in the code.

Instead, I think it's more practical to keep a separate database containing just the problem labels and the year that problem was selected. This database can be updated by the document, but since any problems that haven't been used in the past two years can be discarded, this database is much smaller than the original database. Let's call this database file, say, `mth101-years.dbtex`. On the first year, this file won't exist. Recall from Example 33 the `\InputIfFileExists` command. If the file doesn't exist, a new database can be created using:

`\DTLnewdb{`⟨*db-name*⟩`}` Definition

where ⟨*db-name*⟩ is the database name.

Example:

↑ Input

```
\InputIfFileExists{mth101-years.dbtex}
{}% file exists
{\DTLnewdb{mth101-years}}% file doesn't exist
```

Input

While the main database is iterated over, each question label can be added to the mth101-years database with the current year. To add data, you first need to add a new row to the database using:

\DTLnewrow{⟨*db-name*⟩} Definition

and then you can add the entries for that row using:

\DTLnewdbentry{⟨*db-name*⟩}{⟨*col-label*⟩}{⟨*value*⟩} Definition

where ⟨*col-label*⟩ is the column label and ⟨*value*⟩ is the value for that column. By default, the value isn't expanded. To change this, you first need to use the command:

\dtlexpandnewvalue Definition

Example 47. Randomly Selecting Problems Not Used in the Past Two Years

(This exercise assumes that the current year is 2014.) Adapting the earlier code from Example 46:

↑ Input

```
\documentclass{article}

\usepackage{etoolbox}
\usepackage{datatool}

\newbool{showanswers}
\booltrue{showanswers}

\DTLloaddbtex{\problemDB}{mth101.dbtex}

\InputIfFileExists{mth101-years.dbtex}
{}% file exists
{\DTLnewdb{mth101-years}}% file doesn't exist

\begin{document}
\begin{center}\bfseries\Large
Assignment~1\ifbool{showanswers}{ (Solution Sheet)}{}
\end{center}

\dtlexpandnewvalue
```

```
\begin{enumerate}
\DTLforeach*{\problemDB}
 {\Label=Label,\Question=Question,\Answer=Answer}%
 {%
   \item \Question
   % add this label to the new database:
   \DTLnewrow{mth101-years}% add a new row
   \DTLnewdbentry{mth101-years}{Label}{\Label}%
   \DTLnewdbentry{mth101-years}{Year}{\number\year}%
   % print the solution if this is the answer sheet:
   \ifbool{showanswers}{\par\textbf{Solution: }\Answer}{}%
 }
\end{enumerate}
```

Input

At the end of the document, the database needs to be saved:

Input

```
\DTLsaverawdb{mth101-years}{mth101-years.dbtex}
\end{document}
```

Input

(You can download or view this document.)

The call to datatooltk can use the --merge command line option. For example, to randomly select five problems:

```
datatooltk --in mth101.dbtex --merge Label mth101-years.dbtex
--shuffle --filter-and --filter Year ne 2013
--filter Year ne 2012 --truncate 5 --output problems.dbtex
```

Shell

If the mth101 database doesn't need editing, this call only really needs to be done once a year. However, if you edit the database by removing, adding or swapping rows, you may end up with a different selection, and labels that are no longer selected will still be assigned to the current year. For example, suppose diff:arcsin was selected for this year, but then you add another problem to mth101.dbtex so that now diff:arcsin is no longer selected, but it's still listed in mth101-years.dbtex as having been selected this year. You can fix this using:

```
datatooltk --in mth101-years.dbtex --filter Year eq 2013
--filter Year eq 2012 --output mth101-years.dbtex
```

Shell

This also has the advantage of removing any problems from pre-2012, which trims down the database.

If you use make on a Unix-like system, the Makefile could look something like:

```
CURRYEAR:=$(shell date +%Y)
LASTYEAR:=$(shell expr $(CURRYEAR) - 1)
YEARBEFORE:=$(shell expr $(CURRYEAR) - 2)

assignmentsheet1.pdf     : assignmentsheet1.tex problems.dbtex
                         pdflatex assignmentsheet1

problems.dbtex  : mth101.dbtex
                datatooltk --in mth101.dbtex \
                --merge Label mth101-years.dbtex \
                --shuffle \
                --filter-and \
                --filter Year ne $(LASTYEAR) \
                --filter Year ne $(YEARBEFORE) \
                --truncate 5 \
                --output problems.dbtex

update          :
                datatooltk --in mth101-years.dbtex \
                --filter Year eq $(LASTYEAR) \
```

```
    --filter Year eq $(YEARBEFORE) \
    --output mth101-years.dbtex
```

Now, at the start of each year (or after altering the structure of the database in `mth101.dbtex`) you can use

```
make update
```
Shell

to trim `mth101-years.dbtex` to just the entries for the previous two years. (There's probably a more efficient way of writing this `Makefile`, but a discussion of the `make` utility is beyond the scope of this book. If you want to copy the above code, remember to use the TAB character in the appropriate places. Alternatively, you can download the file from the examples directory.)

Note that the `--merge` option will be ignored if the file to be merged doesn't exist. (Just a warning message will be displayed on the standard error stream.) This means that the `problems.dbtex` target will work on the first instance, even though the `mth101-years.dbtex` file doesn't exist.

Recall the `\marginpar` command from Exercise 21. This can be used to, say, display the number of points for a question in the margin. For example, if all questions are worth 20 points, then within the body of `\DTLforeach` the number of points can be inserted into the margin:

```
\item \marginpar{(20 points)}\Question
```
Input

Although it may be better to define a command called, say, \points to make it easier to customize. For example, in the preamble:

↑ Input

```
\newcommand*{\points}[1]{%
  \marginpar{(#1 points)}%
}
```

↓ Input

Then the body code of \DTLforeach can be simplified:

```
\item \points{20}\Question
```
Input

Now you just need to modify the definition of \points if you want to change the way the points are displayed. For example, if the argument of \points is always an integer, you could check for a single point and change "points" to "point":

↑ Input

```
\newcommand*{\points}[1]{%
```

```
  \marginpar{(#1
  \ifnum#1=1\relax
     point%
  \else
     points%
  \fi)}%
}
```

Input

If the argument may be a decimal number, the datatool package provides the command:

\dtlifnumeq{⟨*number 1*⟩}{⟨*number 2*⟩}{⟨*true*⟩}{⟨*false*⟩} Definition

which can be used with decimal numbers. For example:

Input

```
\newcommand*{\points}[1]{%
  \marginpar{(#1
  \dtlifnumeq{#1}{1}{point}{points})}%
}
```

Input

Perhaps the points should depend on the difficulty level. For example, 5 points for a level 1 question, 10 points for a level 2 question and 20 points for a level 3 question. The \ifcase command described in §7.3 can be used to check the level:

↑ Input

```
\item
   \ifcase\Level
   \or
     \points{5}%
   \or
     \points{10}%
   \or
     \points{20}%
   \fi
   \Question
```

↓ Input

Again, you can define a command that will simplify the document code:

↑ Input

```
\newcommand*{\PointsForLevel}[1]{%
```

```
    \ifcase#1
    \or
      \points{5}%
    \or
      \points{10}%
    \or
      \points{20}%
    \fi
}
```

↓ Input

Now the code in the loop is:

```
\item \PointsForLevel{\Level}\Question
```

Input

Exercise 26. Creating an Assignment Sheet with the datatool Package

The exercises directory that comes with this book has a database called `mth102.dbtex` (shown in Figure 9.3). You can download this file or create your own. This database is an amalgamation of the two databases from Example 45 with an extra column labelled "Topic". The topics are set to either "Basic" or "Theory". The questions taken from the `problems-1stprinciples` database have all been given a value of 3 for the level. Create an assign-

ment sheet (or exam paper) that has the questions randomly selected from the `mth102` database. There should be two Level 1 questions from the "Basics" topic, one Level 2 question from the "Basics" topic and one Level 3 question from the "Theory" topic. Each question should have the points displayed, using the above point allocation scheme.

For the More Adventurous:
Adjust the `\points` command so that it keeps a running total. This total should ideally occur at the start of the document, but as the value isn't known until the end of the document, the information needs to be written to the auxiliary (`.aux`) file. LaTeX provides the command:

`\protected@write{`⟨*output stream*⟩`}{`⟨*init code*⟩`}{`⟨*text*⟩`}` Definition

which will write ⟨*text*⟩ to the file identified by ⟨*output stream*⟩. The second argument, ⟨*init code*⟩, is provided for any initialisation that needs to be done prior to writing the text. The output stream for the document's auxiliary file is identified by the command `\@auxout`. You'll need to wrap the point total up in a command that can be used to reference the total at the start of the next run. Remember to use `\protect` in ⟨*text*⟩ to prevent expansion of this helper command.

You can download or view a solution to this exercise.

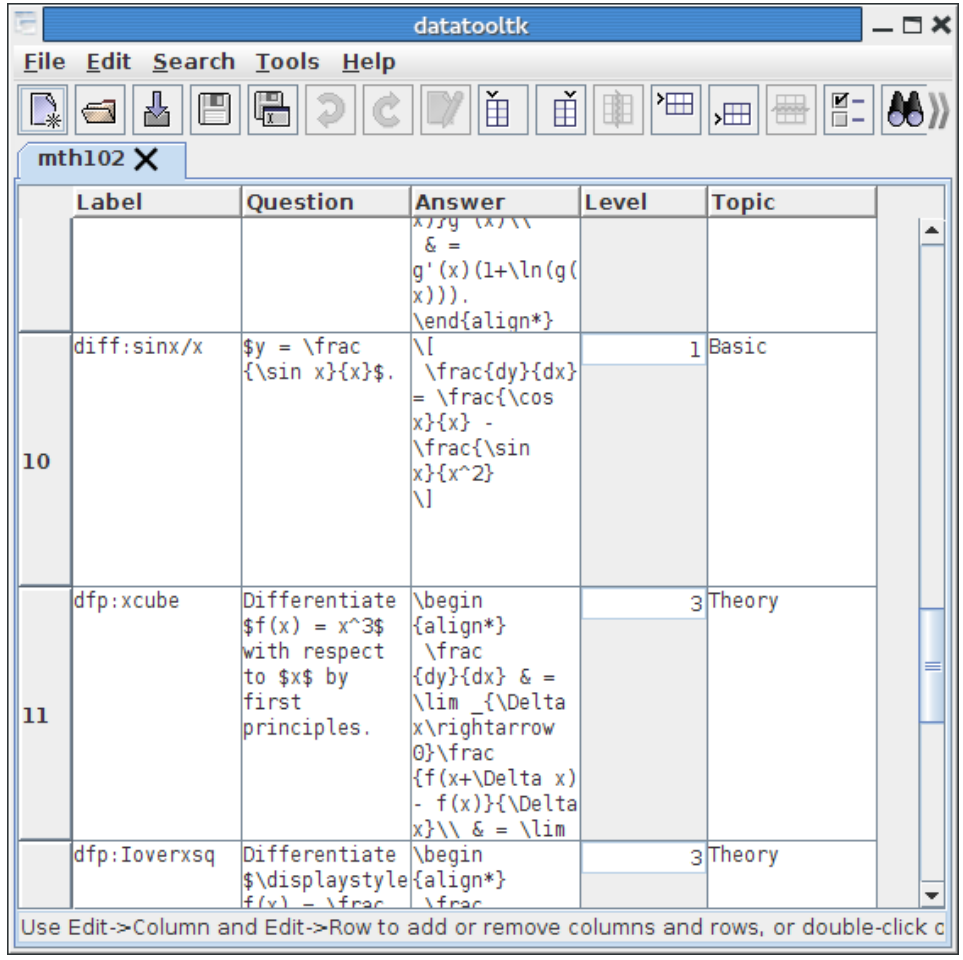


Figure 9.3 The `mth102` Database

9.5 Random Numbers

The previous sections have looked at randomly selecting problems from a database, but it may be that you want to generate questions that use random numbers (for example, as coefficients) to make a slightly different problem each year. In addition to the random number command \PSNrandom provided by probsoln, both the fp and pgfmath packages provide a way of randomly generating numbers. In the case of the fp package, you can generate a random number between 0 and 1 using

\FPrandom{⟨*cs*⟩} Definition

where ⟨*cs*⟩ is a control sequence in which to store the random number. The random number generator seed is set using

\FPseed=⟨*number*⟩ Definition

(\FPseed is a count register.) For example

```
\FPseed=\year
```
Input

will set the seed to the current year.

The pgfmath package provides

\pgfmathparse{⟨expression⟩} Definition

which parses the given mathematical expression and sets \pgfmathresult to the result. There are a number of functions that may be used within ⟨expression⟩ (see the pgf user guide [101] for further details) but the random generator functions are

rnd Definition

which generates a number between 0 and 1,

rand Definition

which generates a number between −1 and 1, and

random(⟨x⟩,⟨y⟩) Definition

which generates a random integer between ⟨x⟩ and ⟨y⟩, if both are present, or a random integer between 1 and ⟨x⟩ if only ⟨x⟩ is present:

random(⟨x⟩) Definition

or a random number between 0 and 1 if no arguments are present:

random() Definition

The random number seed can be set using:

\pgfmathsetseed{⟨n⟩} Definition

where ⟨n⟩ is an integer. For example:

↑ Input

```
Year: \the\year.
\pgfmathsetseed{\year}
\pgfmathparse{random(2,10)}
Random number: \pgfmathresult.
```

↓ Input

produces:

Year: 2021. Random number: 5. Output

Additionally, the pgfmath package also provides:

\pgfmathrandominteger{⟨cs⟩}{⟨minimum⟩}{⟨maximum⟩} Definition

which defines the control sequence ⟨cs⟩ to be a pseudo-randomly generated integer between ⟨minimum⟩ and ⟨maximum⟩ (inclusive). You can also define a list from which you want to randomly select an item. First you need to define the list using:

\pgfmathdeclarerandomlist{⟨*list name*⟩}{{⟨*item 1*⟩}{⟨*item 2*⟩}...} Definition

where ⟨*list name*⟩ is the name of the list and ⟨*item 1*⟩, ⟨*item 2*⟩ etc are the list items. (Note that this list isn't a comma-separated list. Each item is in braces like an argument.)

Once the list has been defined you can randomly select an item using:

\pgfmathrandomitem{⟨*cs*⟩}{⟨*list name*⟩} Definition

where ⟨*list name*⟩ identifies the list. The result can then be accessed using the supplied control sequence ⟨*cs*⟩.

Example:

↑ Input

```
% define list
\pgfmathdeclarerandomlist{projects}%
{% list items
 {ray-guns}% first item
 {mind-controlling cookies}% second item
 {exploding chocolates}% third item
 {telepathic cakes}% fourth item
}
```

```
% randomly select an item from the list
\pgfmathrandomitem{\thisproject}{projects}

What are the advantages and drawbacks of \thisproject?
```

↓ Input

⚠ As mentioned earlier, take care if you are using a mechanism that first displays questions and later (for example, at the end of the document) displays the solutions as this can cause a different randomly generated value in the solution. As before, I recommend that the question part globally defines a command that stores the randomly generated value which can later be accessed in the solution.

Example 48. Random Selection with pgfmath and probsoln
(Recall the commands \ifundef, \global and \let from §2.1.1.)

↑ Input

```
\documentclass{article}

\usepackage{pgfmath}
\usepackage{probsoln}

% set random seed
```

```
\pgfmathsetseed{\year}

\begin{defproblem}{easy.diff}%
  \ifundef\easydiffcoeff
  {%
    \pgfmathrandominteger{\easydiffcoeff}{2}{10}% random coefficient
    \global\let\easydiffcoeff\easydiffcoeff % make it global
  }
  {}% already been defined
  \begin{onlyproblem}
  % question
  Differentiate with respect to $x$:
  \[
    y = \sin(\easydiffcoeff x)
  \]
  \end{onlyproblem}
  \begin{onlysolution}
  % solution
  $ y' = \easydiffcoeff\cos(\easydiffcoeff x) $
  \end{onlysolution}
\end{defproblem}
```

```
\begin{document}

\section{Questions}

\begin{enumerate}
\foreachproblem{\item\thisproblem}
\end{enumerate}

\showanswers
\section{Solutions}
\begin{enumerate}
\foreachsolution{\item\thisproblem}
\end{enumerate}

\end{document}
```

Input

This produces (where the year is 2014) the result shown in Figure 9.4. You can download or view this example.

Output

1 Questions

1. Differentiate with respect to x:

$$y = \sin(8x)$$

2 Solutions

1. $y' = 8\cos(8x)$

Figure 9.4 Random Selection with pgfmath and probsoln

10. Business Cards, Flyers and Leaflets

There are only a few packages on CTAN for typesetting business cards listed under the file-card topic, and only one of them is for LaTeX 2ε (rather than the old LaTeX2.09) and that's bizcard [42]. At the time of writing, the current version is 1.1 dated 1999-09-04. It's in both MiKTeX and TeX Live, but is only suitable for the US 76.2 mm × 50.8 mm (2 in × 3.5 in) card size.

The labels topic provides some more options, but discounting Plain TeX and LaTeX2.09 options and also discounting packages that aren't in both MiKTeX and TeX Live, only three remain: jlabels [103] (for making letter-sized pages of labels), labels [67] (designed for sheets of Avery 5360 sticky labels, which could possibly be adapted for business cards) and ticket (for making labels, visiting-cards and pins). As far as I can tell, all these packages, with the exception of ticket, appear to be designed for letter paper.

The ticket package can be used with either letter or A4 paper (or any other size, if required). You just need to set up the paper margins, the size of each card, the distance between the cards and the number or rows and columns required to position the cards on the page. This package uses LaTeX's picture environment, which is an unsophisticated but platform-

independent drawing environment.

The layout topic has a much greater list. Some of these are general purpose packages for positioning text or graphics on the page. The only one that seems to be specifically designed for leaflets is the leaflet class (version 1.0e 2013-11-06, at the time of writing) which can be used to create z-fold leaflets. The general purpose positioning packages include the flowfram package, which has an optional helper GUI application called `flowframtk`.

There don't appear to be any classes or packages listed on CTAN that specifically deal with single-paged flyers, but that's hardly surprising given the lack of structure in such a document. A flyer would typically need a graphics environment, such as the picture environment, to insert the required text and graphics, although a very simple text-only flyer could be created using the article class with an empty page style.

It's sometimes necessary to include bar codes, such as QR codes, in leaflets or flyers. There are a number of bar code packages listed in the barcode topic. For brevity, only pst-barcode is discussed.

The rest of this chapter is arranged as follows:

§10.1 Describes how to use the picture environment (which is needed for the subsequent sections on using the ticket package and leaflet class).

§10.2 Describes how to use the ticket package to create business cards.

§10.3 Describes how to use the leaflet class.

§10.4 Describes how to use the pst-barcode class.

§10.5 Describes the flowfram package and the flowframtk application.

10.1 The picture Environment

[FAQ: Drawing with TEX]

The picture environment is defined in the LATEX kernel, so you don't need to load any packages to use it. However, it's rather restrictive as to what you can actually draw.

Definition

```
\begin{picture}(⟨width⟩,⟨height⟩)(⟨llx⟩,⟨lly⟩)
⟨picture commands⟩
\end{picture}
```

This has a different syntax to most of the standard environments as the arguments are placed in parentheses rather than curly braces. Another unusual aspect is that the second argument (⟨*llx*⟩,⟨*lly*⟩) is optional, even though it isn't delimited by square brackets.

The first argument (⟨*width*⟩,⟨*height*⟩) indicates the width and height of the picture, and the second argument indicates the co-ordinates of the lower left-hand corner (the origin, if omitted). In both cases, the values are considered to be in terms of the length \unitlength. The picture environment is in fact just another instance of a box (see Volume 1 [92, §4.7]) and

its contents should only consist of declarations and drawing or positioning commands.

The most commonly used of these commands is:

`\put(`⟨*x*⟩`,`⟨*y*⟩`){`⟨*object*⟩`}` Definition

which puts ⟨*object*⟩ at the co-ordinates specified by (⟨*x*⟩,⟨*y*⟩) (which are again in terms of `\unitlength`). The ⟨*object*⟩ may be text, included graphics (using the graphicx package's `\includegraphics` command), straight lines or arrows. A straight line is specified by:

`\line(`⟨*h*⟩`,`⟨*v*⟩`){`⟨*length*⟩`}` Definition

and an arrow (a straight line with an arrowhead) is specified by:

`\vector(`⟨*h*⟩`,`⟨*v*⟩`){`⟨*length*⟩`}` Definition

There are only a limited number of gradients available. In both cases, the gradient is specified via the horizontal ⟨*h*⟩ and vertical ⟨*v*⟩ displacements, where ⟨*h*⟩ and ⟨*v*⟩ are both integers without a common divisor. In the case of `\line`, ⟨*h*⟩ and ⟨*v*⟩ are restricted to values between −6 and +6, inclusive, whereas in the case of `\vector`, those arguments are restricted to values between −4 and +4, inclusive.

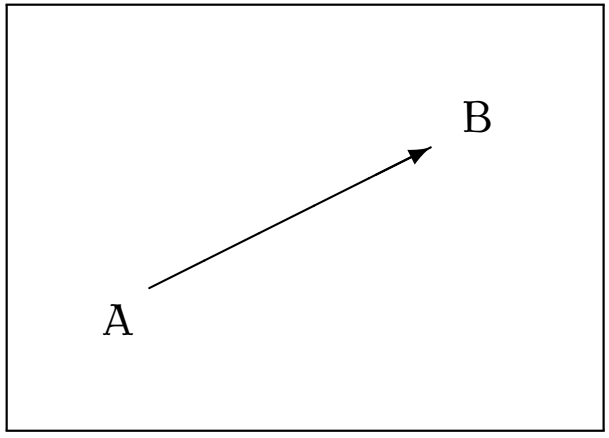
Example:
The following code first sets the unit length to 1 cm and then creates a picture that's 3 cm wide by 2.5 cm high. I've added a border around the picture using \fbox.

Input

```
\setlength{\unitlength}{1cm}% set the unit length
\fbox{%
  \begin{picture}(3.5,2.5)
    \put(0.5,0.5){A}
    \put(0.8,0.8){\vector(2,1){1.8}}
    \put(2.8,1.8){B}
  \end{picture}
}
```

Input

This produces:



Output

Slanted lines are drawn using a special font where the characters consist of small line segments. This is why there's a restriction on the available gradients.

You can also put circles or ovals in the picture using:

`\circle{`⟨*diameter*⟩`}`

Definition

to create a circle with the given diameter, or

`\oval(`⟨*w*⟩`,`⟨*h*⟩`)[`⟨*segment*⟩`]`

Definition

to create an oval whose width and height are given by ⟨*w*⟩ and ⟨*h*⟩. In both cases, the lengths are again specified in terms of `\unitlength`. The optional argument of `\oval` may be used if only a quarter or half oval is required, instead of the full oval. In the case of a half oval, ⟨*segment*⟩

should be a single letter identifying which half: `l` (left), `r` (right), `t` (top) or `b` (bottom). For a quarter oval, ⟨*segment*⟩ should be a two-letter combination, for example, `tr` indicates top right.

A filled circle is created using the starred form:

`\circle*{`⟨*diameter*⟩`}` Definition

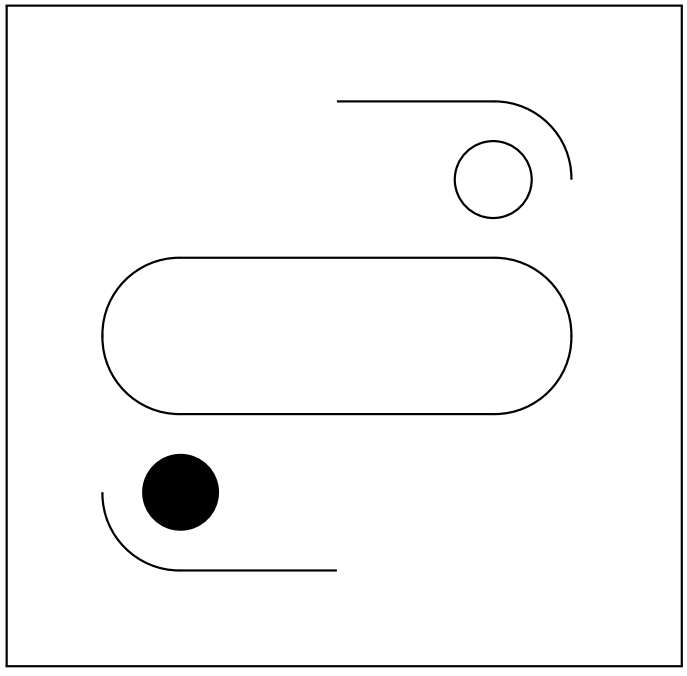
Example:

↑ Input

```
\setlength{\unitlength}{1cm}
\fbox{%
  \begin{picture}(4.0,4.0)
   \put(1,1){\circle*{0.5}}
   \put(2,1){\oval(3,1)[bl]}
   \put(2,3){\oval(3,1)[tr]}
   \put(2,2){\oval(3,1)}
   \put(3,3){\circle{0.5}}
  \end{picture}
}
```

↓ Input

This produces:



Output

\shortstack[⟨*align*⟩]{⟨*text*⟩}

Definition

The \shortstack command is similar to a single-column tabular environment, where the contents are given in ⟨*text*⟩. As with tabular, this command creates a box containing the tabulated data where the rows are separated using \\ but unlike tabular, the rows aren't evenly spaced. The optional argument ⟨*align*⟩ indicates the horizontal alignment of the column. This may be one of `c` (centre), `l` (left) or `r` (right). The default is `c`. The box's refer-

ence point is the lower-left corner. When \shortstack is used within the ⟨*object*⟩ argument of \put, the co-ordinates correspond to the reference point.

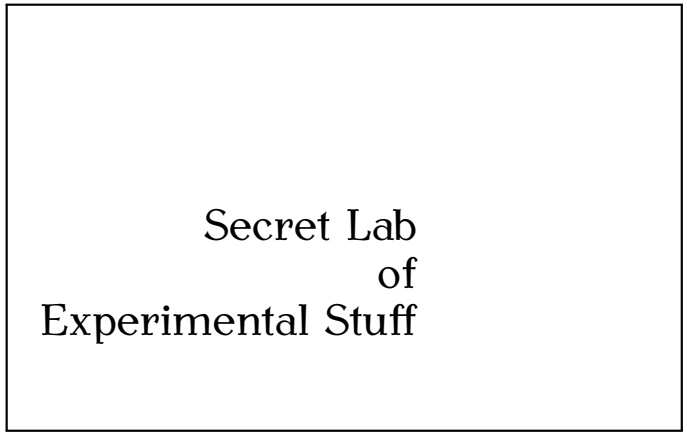
Example:

↑ Input

```
\setlength{\unitlength}{1cm}% set the unit length
\fbox{%
  \begin{picture}(4,2.5)
    \put(0.1,0.5){\shortstack[r]{Secret Lab\\
    of\\
    Experimental Stuff}}
  \end{picture}
}
```

↓ Input

This produces:



Output

You may remember \framebox and \makebox from Volume 1 [92, §4.7]. When used within the picture environment, these commands have different syntax.

\framebox(⟨w⟩,⟨h⟩)[⟨align⟩]{⟨text⟩}

Definition

and

\makebox(⟨w⟩,⟨h⟩)[⟨align⟩]{⟨text⟩}

Definition

In both cases the reference point is the lower-left corner of the box. The width and height of the box are given by ⟨w⟩ and ⟨h⟩ (again in terms of \unitlength). The optional argument ⟨align⟩ indicates the alignment of the text. The default is to centre the text both vertically and horizontally. To change this, the ⟨align⟩ argument may be one or two letters: l (left), r (right), t (top) and b (bottom).

There's another box similar to \framebox:

\dashbox{⟨*dash length*⟩}(⟨*w*⟩,⟨*h*⟩)[⟨*align*⟩]{⟨*text*⟩} Definition

This mostly has the same syntax as above, but produces a dashed frame. The additional argument ⟨*dash length*⟩ specifies the length of the dashes.

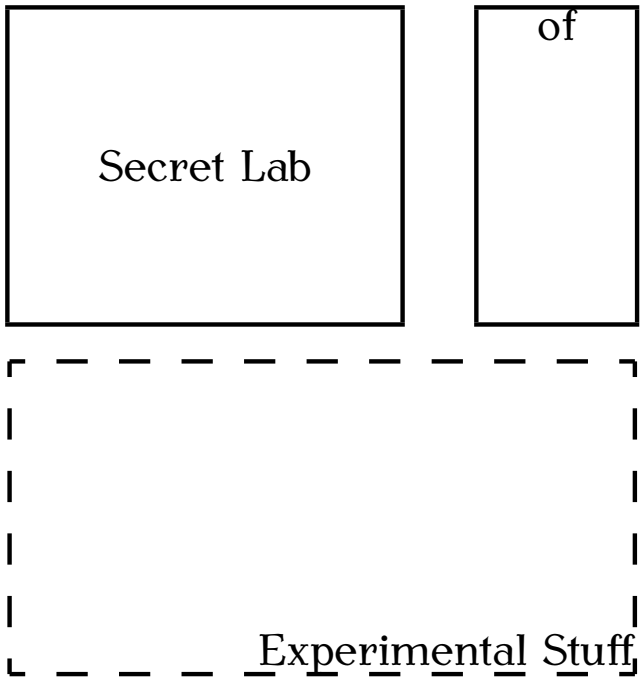
Example:

```
\setlength{\unitlength}{1cm}% set the unit length
\begin{picture}(4.0,4.25)
  \thicklines
  \put(0.0,2.25){\framebox(2.5,2){Secret Lab}}
  \put(3.0,2.25){\framebox(1,2)[t]{of}}
  \put(0.0,0.0){\dashbox{0.2}(3.75,2)[br]{Experimental Stuff}}
\end{picture}
```

Input

Input

This produces:



Output

There are two other commands provided for use in the picture environment. These aren't used with \put. As before, all lengths are in terms of \unitlength.

\qbezier[⟨*points*⟩](⟨*start x*⟩,⟨*start y*⟩)(⟨*control x*⟩,⟨*control y*⟩)(⟨*end x*⟩,⟨*end y*⟩)

Definition

This draws a quadratic Bézier curve with the given start, end and curvature control points. LaTeX draws the curve using multiple points. More points results in a smoother curve but a longer document build time. You can

use the optional argument to specify the number of points used to draw the curve.

\multiput(⟨x⟩,⟨y⟩)(⟨inc x⟩,⟨inc y⟩){⟨n⟩}{⟨object⟩} Definition

This puts ⟨n⟩ copies of ⟨object⟩, starting at position (⟨x⟩,⟨y⟩) and advancing the position by (⟨inc x⟩,⟨inc y⟩) each time.

Example 49. A Postcard

This example uses the picture environment to create a simple postcard advertising an event. The geometry package [109] is used to set a non-standard paper size (6 in wide by 4 in high with no margins). I also used the graphicx package to include the sample image chicken.png.

↑ Input

```
\documentclass[12pt]{article}

\usepackage[papersize={4in,6in},margin={0in,0in}]{geometry}
\usepackage{graphicx}

\pagestyle{empty}

\setlength{\unitlength}{1in}
```

```
\begin{document}
\centering
\begin{picture}(4,6)
 \put(0,1.25){\makebox(4,3.75){%
    \includegraphics[height=3.75in]{chicken}}}
 \put(0,5){\makebox(4,1){\large\bfseries
   Oh No! The Chickens Have Escaped!}}
 \put(0,0.5){\makebox(4,0.75){%
   \shortstack
   {%
   Written by award-winning author Dickie Duck\\
   Illustrated by internationally renown artist Jos\'e Arara
   }%
 }}
 \put(0,0){\makebox(4,0.5){\large\bfseries
   Book Launch 1st August 2014}}
\end{picture}

\end{document}
```

↓ Input

(You can download or view this document.) The resulting document is

shown in Figure 10.1.

10.2 The ticket Package

The ticket package [23] (version 0.4b, 2010-11-30, at the time of writing) can be used to make labels, visiting cards, pins and flash-cards. The ticket settings can be specified in a ticket definition file (with the extension `.tdf`). This file can then be specified when you load the ticket package:

`\usepackage[`⟨*tdf-file*⟩`,`⟨*other options*⟩`]{ticket}` Input

where ⟨*tdf-file*⟩ is the filename without the `.tdf` extension.

In the ticket definition file, you can set up the ticket dimensions. Since this package uses the picture environment, described above, you can adjust the unit of measurement by changing the value of `\unitlength` in the definition file. Within this file you can also specify the number and layout of tickets using:

`\ticketNumbers{`⟨*num cols*⟩`}{`⟨*num rows*⟩`}` Definition

where ⟨*num cols*⟩ and ⟨*num rows*⟩ are the number of tickets in the horizontal and vertical directions. The ticket dimensions are specified using:

Oh No! The Chickens Have Escaped!

Written by award-winning author Dickie Duck
Illustrated by internationally renown artist José Arara

Book Launch 1st August 2014

Figure 10.1 A Postcard

\ticketSize{⟨*width*⟩}{⟨*height*⟩} Definition

where the ⟨*width*⟩ and ⟨*height*⟩ are in terms of \unitlength. The distance between the tickets is specified using:

\ticketDistance{⟨*x-dist*⟩}{⟨*y-dist*⟩} Definition

where ⟨*x-dist*⟩ and ⟨*y-dist*⟩ are the horizontal and vertical distances in terms of \unitlength. Note that you need to set \unitlength before using the above dimension commands.

For single use only, you can just put these settings in your document after you load the ticket package (without specifying the ⟨*tdf-file*⟩ in the package options).

In the document you can set up the default ticket content by redefining:

\ticketdefault Definition

Since this is placed inside the picture environment, remember to use picture commands, such as \put. The default definition is:

```
\put ( 5, 5){Ticket....}
```
Input

Example

Suppose each ticket should have a logo (stored in the image file `logo.png`) and departmental information:

Input

```
\renewcommand*{\ticketdefault}{%
 \put (80,82) {\includegraphics[width=12mm]{logo}}
 \put (5,85) {\large\bfseries Secret Lab of Experimental Stuff}
}
```

Input

(Remember to load the graphicx package.)

The actual ticket is displayed using

`\ticket{`⟨*content*⟩`}` Definition

where ⟨*content*⟩ is additional content. You either need to have multiple `\ticket` commands for each ticket or place `\ticket` inside a loop. (Recall §2.7.)

Example

Suppose I want to display a ticket with a name on it:

Input

```
\ticket
{
```

```
  \put (49,30) {\makebox(0,0) {\Large\bfseries Polly Parrot}}
}
```

↓ Input

This will create a ticket with the default background (as given by \ticketdefault) and the name added to it.

The ticket package documentation suggests defining a wrapper command:

↑ Input

```
\newcommand*{\myticket}[1]{%
  \ticket
  {
    \put (49,30) {\makebox(0,0) {\Large\bfseries #1}}
  }%
}
```

↓ Input

Now you can do, for example,

↑ Input

```
\myticket{Polly Parrot}
\myticket{Mabel Canary}
```

↓ Input

The ticket package automatically sets the page style to `empty`. Package options are available to create marks or decorations around the tickets:

`crossmark`	Puts a cross at all four corners.
`circlemark`	Puts an unfilled circle at all four corners.
`emptycrossmark`	Like `crossmark` but only draws the parts of the marker that lie outside the ticket.
`cutmark`	Just adds cutmarks at the outer region.
`boxed`	Adds a frame around the ticket.

Example 50. Name Labels (ticket package)
This example creates a set of six name labels. I've used the geometry package to set up the paper margins and the lmodern package [38] to switch to the Latin Modern fonts. The graphicx package is required for the logo. I've used the sample logo `dummy-logo.png`, but you can replace this with another image if you like.

↑ Input

```
\documentclass[a4paper]{article}

% fonts and encodings
\usepackage{lmodern}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\usepackage[cutmark]{ticket}
\usepackage[margin=5mm]{geometry}
\usepackage{graphicx}

\setlength{\unitlength}{1mm}
\ticketNumbers{2}{3}
\ticketSize{98}{90}
\ticketDistance{4}{4}

\renewcommand*{\ticketdefault}{%
 \put (80,80) {\includegraphics[width=12mm]{dummy-logo}}
 \put (5,85) {\large\bfseries Secret Lab of Experimental Stuff}
 \put (5,75) {\large\scshape University of Somewhere}
```

```
 \put (45,30) {\makebox(0,0){\Large\itshape Culinary Experimental
Research}}
}

\newcommand*{\myticket}[1]{%
  \ticket
  {
    \put (45,50) {\makebox(0,0) {\Large\bfseries #1}}
  }%
}

\begin{document}
\myticket{Polly Parrot}
\myticket{Mabel Canary}
\myticket{Zöe Zebra}
\myticket{José Arara}
\myticket{Dickie Duck}
\myticket{Fred Canary}
\end{document}
```

↓ Input

(You can download or view this document.) The resulting document is shown in Figure 10.2.

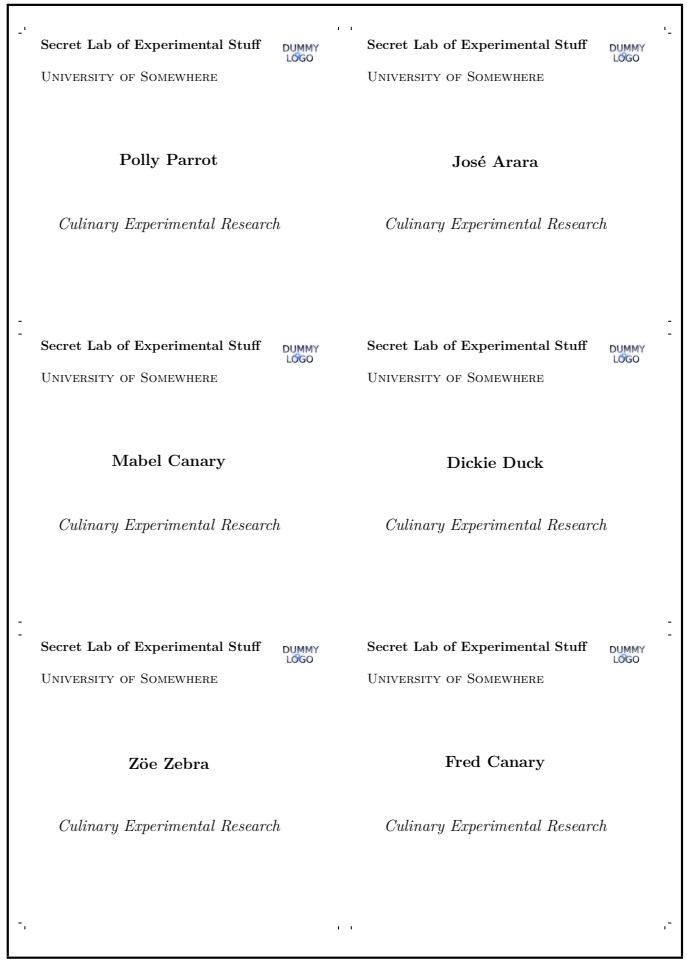


Figure 10.2 Name Labels (ticket package)

Exercise 27. Name Labels (Iteration)
Modify the document in Example 50 so that it uses an iteration method to display the tickets. You can use a comma-separated list with one of the etoolbox commands described in §2.7.2. (You can download or view a solution.) Alternatively load the names from the sample CSV or SQL data and iterate through the database, as described in §2.7.1. (You can download or view a solution for the CSV data.)

10.3 The leaflet Class

The leaflet class manual [63] is accessed using

```
texdoc leaflet-manual
```
Shell

If you just do

```
texdoc leaflet
```
Shell

you'll get the documented code instead.

The manual is formatted as a z-fold leaflet which illustrates the layout but makes on-screen reading difficult as the reverse sheet is upside-down.

The leaflet class loads the article class, but some changes are made to the defaults. For example, `\part` is not available and the other sectioning commands aren't numbered. You can use the `letterpaper` or `a4paper` class options to set the paper size, but other paper sizes may need to have the margins adjusted, which can be done using:

`\setmargins{`⟨*top*⟩`}{`⟨*bottom*⟩`}{`⟨*left*⟩`}{`⟨*right*⟩`}` Definition

Marginal notes and two-column typesetting are disabled and by default there are no page headers or footers. Paragraph indentation is set to zero and paragraphs are separated by vertical space.

Class options (in addition to the options provided by article) are:

`tumble` Print the back sheet upside-down (default).

`notumble` Don't print the back sheet upside-down.

`bothsides` Create both the front and back sheet (default).

`frontside` Only create the front sheet.

`backside` Only create the back sheet.

`foldmark` Print a fold mark (default).

`nofoldmark` Don't print a fold mark.

`combine` Combine three pages to a sheet (default). An error is issued if too much text is generated to fit onto the front and back sheets. (The surplus text is ignored.)

`nocombine` Don't combine multiple pages onto a single sheet.

`twopart` Creates a four-page leaflet (first part) and a two-page detachable form (second part).

`notwopart` Not a two-part leaflet (default).

You may find that ragged-right justification produces better results given the narrow page sizes (where a page is one-third of a sheet). The leaflet class provides the following commands:

`\CutLine{`⟨*page number*⟩`}` Definition

This command may only be used in the preamble and indicates that a cut line should be drawn to the left of the page given by ⟨*page number*⟩. The starred version just draws a dotted line. The unstarred version draws a dotted line with a pair of scissors.

\AddToBackground{⟨page number⟩}{⟨picture code⟩} Definition

Again this command may only be used in the preamble. This indicates that ⟨picture code⟩ should be added to the page given by ⟨page number⟩. With the starred version, the ⟨page number⟩ refers to the sheet number (1 for the front and 2 for the back sheet). The background is placed inside a picture environment, so the ⟨picture code⟩ may include any of the commands described in §10.1. Remember that the co-ordinates are in terms of \unitlength. If you want to provide a specific length that's independent of \unitlength, the leaflet class provides:

\LenToUnit{⟨length⟩} Definition

which may be used to specify a length.

The font declaration used for the sectioning commands is given by

\sectfont Definition

This may be redefined as required. The default value is \bfseries. The font declaration used for the item label in the description environment is given by

\descfont Definition

This may be redefined as required. The default value is \bfseries.

Example 51. Sample Leaflet

This example uses the starred version of \AddToBackground to place text across the first sheet and uses the unstarred version to place an oval on the first page. The origin is the lower left hand corner of the sheet/page. Rather than working out the co-ordinates in terms of \unitlength, I've used \LenToUnit with multiples of \paperwidth and \paperheight.

I've also changed the fonts used by the sectioning commands and the description item labels. Since the default Computer Modern fonts don't support bold smallcaps, I've used the Alegreya package [104] to switch to the Alegreya font, which does have bold smallcaps. (For other fonts, have a look at the Font Catalogue [48].) The wasysym package [41] provides the \Square command used for the tick boxes □.

Leaflets and flyers tend to be less structured than normal documents, so I've used some commands that typically shouldn't be used (or, at least, used only sparingly on the final copy) in article, report or book-like documents. These include

\newpage Definition

which forces a page break, without attempting to vertically justify the page,

\bigskip Definition

which inserts a vertical space,

\vfill

Definition

which inserts a vertical space that will expand to fit the available height. The paragraph justification can be made through an environment, such as center or flushright, which additionally inserts a vertical space above and below the environment, or the justification can be made through a declaration, such as \raggedright (recall Volume 1 [92, §2.12]).

↑ Input

```
\documentclass[a4paper,12pt,notumble]{leaflet}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{Alegreya}

\usepackage{wasysym}
\usepackage{xcolor}
\usepackage{graphicx}

\renewcommand*{\sectfont}{\scshape}
\renewcommand*{\descfont}{\bfseries\scshape}
```

```
\AddToBackground*{1}{%
 \put (0,0)
  {\rotatebox{33}{\resizebox{30cm}{!}{\color{lightgray}CLASSIFIED}}}%
}

\AddToBackground{1}{%
 \put
 (\LenToUnit{0.5\paperwidth},\LenToUnit{0.5\paperheight})
 {\oval(\LenToUnit{0.95\paperwidth},\LenToUnit{0.95\paperheight})}%
}

\CutLine{6}

\begin{document}
\begin{center}\bfseries\Huge
Culinary Experimental Research

\vfill

\normalsize
\begin{tabular}{c}
```

```
Secret Lab of Experimental Stuff\\
University of Somewhere
\end{tabular}

\vfill

\begin{tabular}{c}
Department of Stripy Confectioners\\
College of Somewhere Else
\end{tabular}

\vfill

\includegraphics[height=3cm]{dummy-logo}
\end{center}

\newpage
\raggedright

\section{Secret Lab of Experimental Stuff}

The Secret Lab of Experimental Stuff is a top-secret laboratory
```

```
whose existence is highly classified so don't tell anyone about it
or we'll get really cross with you.

The University of Somewhere denies all knowledge of the Secret Lab
of Experimental Stuff, except on Open Days where members of the
public may visit the facility and ask questions as long as they
consent to a memory wipe when they leave. The memory wipe is
harmless (well, we haven't really tested it properly, but no one's
complained so far) and your memory of the visit will be replaced by
a pleasant recollection of spending the day feeding the ducks in the
nearby pond.

\begin{flushright}
\includegraphics{mallard}
\end{flushright}

\section{Department of Stripy Confectioners}

% lots of text omitted

\newpage
\section{Query Form}
```

```
If you'd like to know more about the exciting collaboration between
the Secret Lab of Experimental Stuff and the Department of Stripy
Confectioners please fill in your details below and post this slip to:

\bigskip

\begin{tabular}{@{}l}
Miss Ingperson\\
Secret Lab of Experimental Stuff\\
University of Somewhere\\
Some City\\
AB3 4YZ
\end{tabular}

\bigskip

\Square\␣I would like to receive quarterly newsletters.

\Square\␣I agree to having my memory wiped.

\Square\␣Yes, I'd really like to feed the ducks.
```

```
\bigskip

\begin{tabular}{@{}lp{4cm}}
Name: & \dotfill \\
Address: & \dotfill\\
 & \dotfill \\
 & \dotfill \\
 & \dotfill \\
Postcode: & \dotfill\\
Country: & \dotfill\\
Telephone: & \dotfill\\
Mobile: & \dotfill\\
Email: & \dotfill
\end{tabular}

\newpage

\section{Research Team}

\begin{description}
\item[Administrator] Mr Big Head
```

```
\item[Assistant Administrator] Dr Bor Ing

\item[Project Co-ordinator] Mabel Canary

\item[Senior Scientists] Dickie Duck, Polly Parrot

\item[Research Assistants] Zöe Zebra, José Arara, Fred Canary

\end{description}

\section{Acknowledgements}

The Culinary Experimental Research team would like to thank the
following:

\begin{description}

\item[University of Somewhere] For something or other

\item[College of Somewhere Else] For providing bread crumbs
```

```
\item[The Ministry of Top Secret Stuff] For supporting the
project somehow.

\end{description}

\end{document}
```

↓ Input

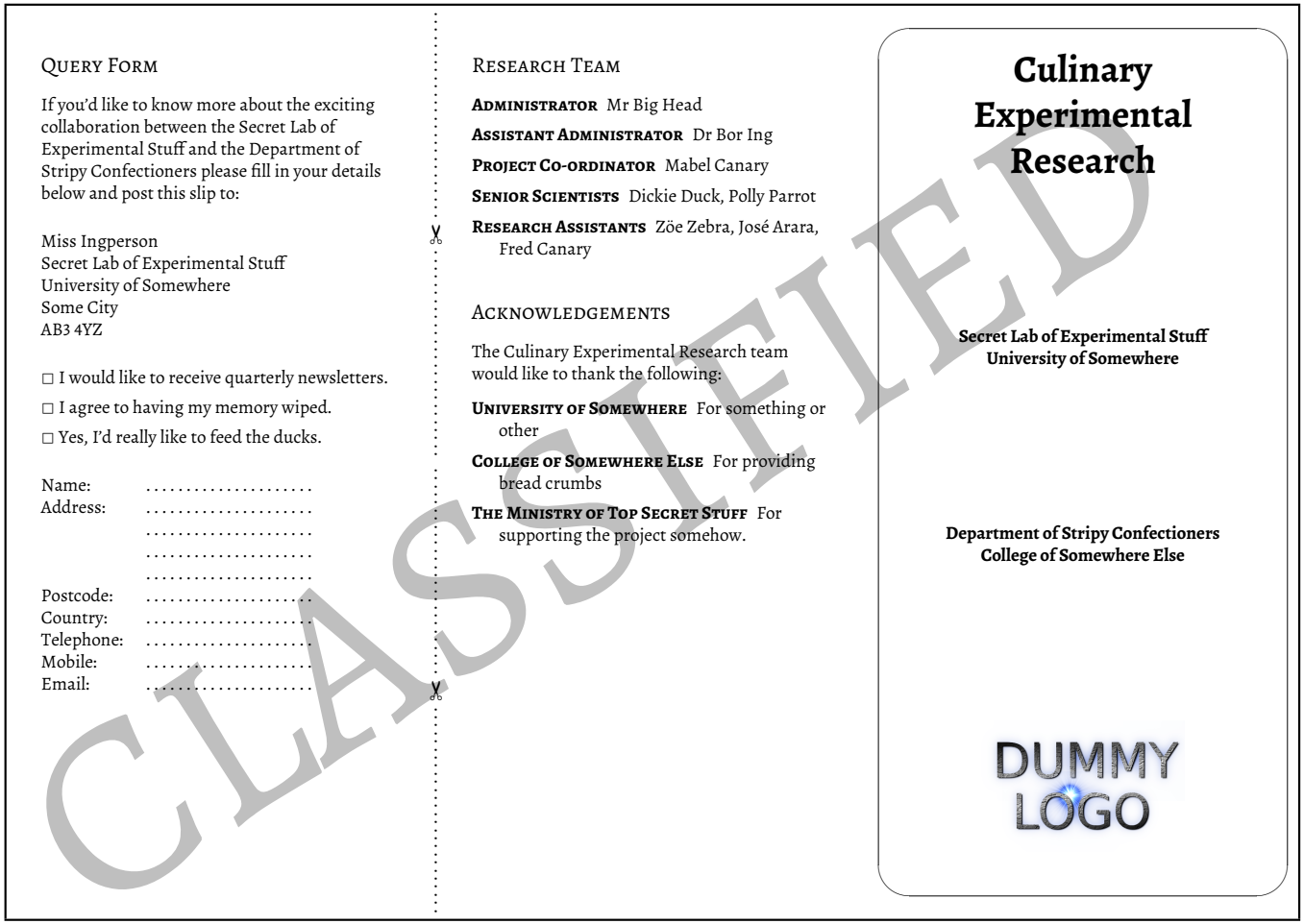
(You can download or view this document, and the sample image files dummy-logo.png, mallard.png and goose.png.)

The resulting document is shown in Figure 10.3 (first sheet) and Figure 10.4 (second sheet). The small marker line to the right of the first section heading on the second sheet is the folding mark. Try this example first without and then with the twopart class option.

Although the \AddToBackground hook adds code to the picture environment, if you prefer to use more advanced image drawing code, such as tikz [101] or pstricks [116], you can do so.

Example 52. Leaflet (with tikz)

The preamble code from Example 51 can be modified to use the tikz package. The new preamble is as follows:



Query Form

If you'd like to know more about the exciting collaboration between the Secret Lab of Experimental Stuff and the Department of Stripy Confectioners please fill in your details below and post this slip to:

Miss Ingperson
Secret Lab of Experimental Stuff
University of Somewhere
Some City
AB3 4YZ

☐ I would like to receive quarterly newsletters.

☐ I agree to having my memory wiped.

☐ Yes, I'd really like to feed the ducks.

Name: .....
Address: .....
.....
.....
.....
Postcode: .....
Country: .....
Telephone: .....
Mobile: .....
Email: .....

Research Team

Administrator Mr Big Head

Assistant Administrator Dr Bor Ing

Project Co-ordinator Mabel Canary

Senior Scientists Dickie Duck, Polly Parrot

Research Assistants Zöe Zebra, José Arara, Fred Canary

Acknowledgements

The Culinary Experimental Research team would like to thank the following:

University of Somewhere For something or other

College of Somewhere Else For providing bread crumbs

The Ministry of Top Secret Stuff For supporting the project somehow.

Culinary Experimental Research

Secret Lab of Experimental Stuff
University of Somewhere

Department of Stripy Confectioners
College of Somewhere Else

DUMMY LOGO

Figure 10.3 A Sample Leaflet (First Sheet)

Secret Lab of Experimental Stuff

The Secret Lab of Experimental Stuff is a top-secret laboratory whose existence is highly classified so don't tell anyone about it or we'll get really cross with you.

The University of Somewhere denies all knowledge of the Secret Lab of Experimental Stuff, except on Open Days where members of the public may visit the facility and ask questions as long as they consent to a memory wipe when they leave. The memory wipe is harmless (well, we haven't really tested it properly, but no one's complained so far) and your memory of the visit will be replaced by a pleasant recollection of spending the day feeding the ducks in the nearby pond.

Department of Stripy Confectioners

The Department of Stripy Confectioners is a department within the College of Somewhere Else. The department is internationally known for its cutting-edge research in the field of stripy confectionery, including humbugs and sticks of rock.

In a spirit of co-operation between the University of Somewhere and the College of Somewhere Else, a number of the department's faculty spend half their time feeding ducks in the pond near the University of Somewhere. The Department of Stripy Confectioners has received a grant of £1 million from the Ministry of Top Secret Stuff to purchase the bread crumbs provided by the University of Somewhere.

Mind-Controlling Cookies

Preliminary tests have shown that the mind-controlling properties of the mind-controlling cookies are somewhat disappointing, but critics have said that they taste very nice and could they have some more bread for the ducks.

Ingredients

- Self-raising flour;
- Butter;
- Chocolate chips;
- Sugar obtained from secret genetically modified beet.

Telepathic Cakes

Preliminary experiments on the telepathic cakes have revealed an unfortunate side-effect. The full results are in the senior scientist's head. Telepathy is required to view them. Please read the full terms and conditions before use.

Ingredients

The ingredients of the telepathic cakes may be obtained telepathically on consumption of the cake.

Exploding Chocolates

One of the junior research assistants suffered injuries during the initial development phase but, after a full and detailed investigation, the health and safety department observed that there are also geese present in the duck pond.

Figure 10.4 A Sample Leaflet (Second Sheet)

↑ Input

```
\documentclass[a4paper,12pt,notumble]{leaflet}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{Alegreya}

\usepackage{wasysym}
\usepackage[x11names]{xcolor}
\usepackage{graphicx}
\usepackage{tikz}
\usetikzlibrary{calc}

\renewcommand*{\sectfont}{\scshape}
\renewcommand*{\descfont}{\bfseries\scshape}

\AddToBackground{1}{%
 \put (0,0)
 {\begin{tikzpicture}
   \path[fill=Thistle1,draw=Thistle4,double=Orchid1,line width=2pt]
     (0,0) rectangle ($(\paperwidth, \paperheight)-(4pt,4pt)$);
```

```
  \end{tikzpicture}
 }%
}

\CutLine{6}
```

Input

The remainder of the document is as before. This uses the `calc` tikz library, which enables co-ordinate calculations using the `$` syntax. For example

```
($(\paperwidth, \paperheight)-(4pt,4pt)$)
```

Input

indicates the co-ordinate obtained by subtracting the point (4 pt, 4 pt) from the point (\paperwidth,\paperheight). The `calc` library must first be loaded using:

```
\usetikzlibrary{calc}
```

Input

The above example code also loads the xcolor package with the `x11names` option to enable the use of the X11 colour names, such as `Thistle1`. For further details about the syntax of the `\path` command, see the pgf/tikz user manual [101]. The first sheet is now as shown in Figure 10.5.

For other possible fancy frames or decorations, have a look at the decoration topic. There is also a non-CTAN tikz-based package called pgfornament available from http://altermundus.com/pages/tkz/ornament how-

ever this will require a manual installation since it's not included in the TEX distributions (recall Volume 1 [92, §A]).

10.4 The pst-barcode Package

The pst-barcode package [113] (version 0.12, 2013-10-26, at the time of writing) is a pstricks package for drawing twenty-nine different types of bar codes, including EAN-13 and QR codes. Since this is a pstricks package, it uses PostScript code, which means that it doesn't work directly with PDFLATEX unless you have the shell escape enabled and use a package such as pdftricks [66].

There are essentially two options if you want to generate a PDF file and you don't have the shell escape enabled:

1. Use `latex`, `dvips` and `ps2pdf` to obtain a PDF version of the document. For example, if your document is in the file `myDoc.tex` then you need to run the following commands:

```
latex myDoc
dvips -o myDoc.ps myDoc.dvi
ps2pdf myDoc.ps myDoc.pdf
```

Shell

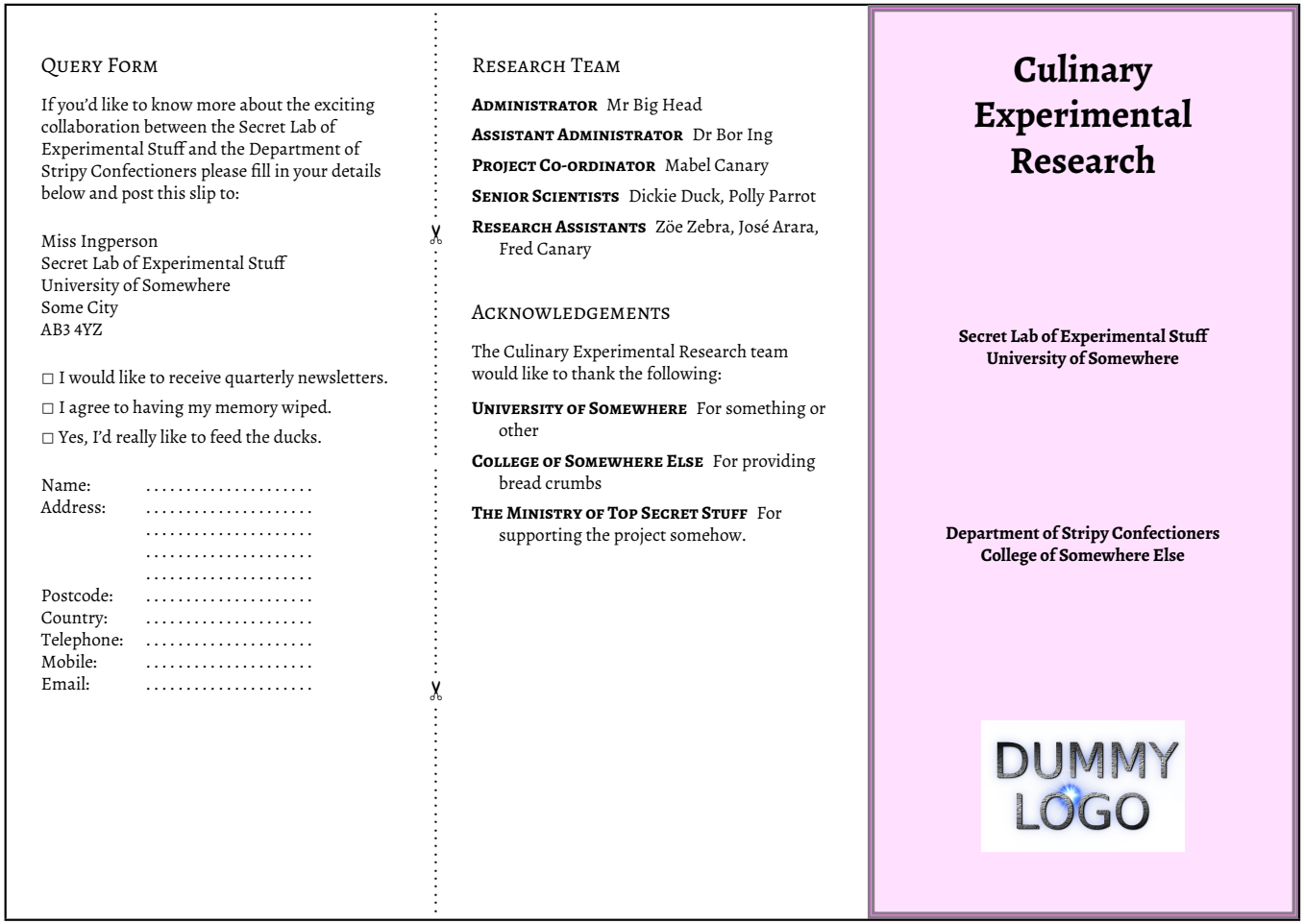


Figure 10.5 A Sample Leaflet Using tikz (First Sheet)

If you use a frontend, such as TeXworks, you need to find the appropriate buttons or menu options to run these commands. If you use arara, you need the following directives:

Input

```
% arara: latex
% arara: dvips
% arara: ps2pdf
```

Input

You can replace the two steps dvips and ps2pdf with a single call to dvipdfm.

2. Put the pstricks code in a standalone document, compile that document using latex, dvips and ps2pdf, as described above, and include the generated PDF file into the main document using \includegraphics.

The pst-pdf package [62] can be used to simplify the second option if you have multiple pstricks images in your document, but you still need the latex, dvips and ps2pdf invocations.

If I'm designing a flyer that requires a bar code, such as an advance information sheet with a QR code, I usually use the second option. Once I've generated the bar code, I rarely need to change it, as my modifications to the document usually concern the accompanying text rather than the bar code, so it's easiest to treat the bar code as an external graphics file.

The pst-barcode package provides the command:

`\psbarcode[`⟨*options*⟩`]{`⟨*text or filename*⟩`}{`⟨*PS options*⟩`}{`⟨*type*⟩`}` Definition

This generates a bar code with zero size, which means it typically needs to go inside an environment or command where you can specify the height and width. (Recall Volume 1 [92, §4.7].) Since pst-barcode automatically loads the pstricks package [116], you can use the pspicture environment:

`\begin{pspicture}[`⟨*baseline*⟩`](`⟨*llx*⟩`,`⟨*lly*⟩`)(`⟨*urx*⟩`,`⟨*ury*⟩`)`
⟨*picture commands*⟩
`\end{pspicture}` Definition

As with the picture environment, the co-ordinate arguments are specified using parentheses, but take care as the syntax of the pspicture environment is slightly different to that of the picture environment. In the case of pspicture, the first argument in parentheses (⟨*llx*⟩,⟨*lly*⟩) specifies the co-ordinates for the lower left corner, and second argument in parentheses (⟨*urx*⟩,⟨*ury*⟩) specifies the co-ordinates for the upper right corner of the picture's bound-

ing box. If (⟨*llx*⟩,⟨*lly*⟩) is omitted, the origin is assumed.

The optional argument ⟨*options*⟩ of \psbarcode is a key=value list. Available options include:

`file` This is a boolean key. This determines whether the argument ⟨*text or filename*⟩ is the bar code text (in the case of `file=false`) or the name of the file containing the bar code text (in the case of `file=true`). The default value for this option is `file=false`.

`transx` This specifies a horizontal shift to apply to the bar code. The default value is `transx=0`.

`transy` This specifies a vertical shift to apply to the bar code. The default value is `transy=0`.

`scalex` This specifies a horizontal scaling to apply to the bar code. The default value is `scalex=1`.

`scaley` This specifies a vertical scaling to apply to the bar code. The default value is `scaley=1`.

`rotate` This specifies the rotation (in degrees) to apply to the bar code. The default value is `rotate=0`.

The ⟨*PS options*⟩ argument are PostScript options separated by whitespace. Available options include:

`includetext`	This enables human readable text.
`font`	This sets the font, which must be a PostScript font. The default is `/Helvetica`.
`guardwhitespace`	This enables the display of whitespace guard marks.

The final argument ⟨*type*⟩ of `\psbarcode` indicates the type of bar code. For example, `ean13` for an EAN-13 bar code, `isbn` for an ISBN bar code, or `qrcode` for a QR code.

Example 53. ISBN Bar Code
The ISBN bar code is just a special form of EAN-13 bar code with a particular prefix. The data, provided in the ⟨*text or filename*⟩ argument, should contain 9 or 10 digits for ISBN-10, and 12 or 13 digits for ISBN-13. (In both cases, the digits separated appropriately with hyphens.) If only 9 (ISBN-10) or 12 (ISBN-13) digits are specified the ISBN check digit is calculated automatically.

The ISBN for the paperback version of this book is 978-1-909440-07-4 so I can create the ISBN bar code using:

```
\psbarcode{1-909440-07-4}{includetext guardwhitespace}{isbn}
```
Input

Here's a complete document containing the bar code:

```
\documentclass{article}

\usepackage{pst-barcode}

\begin{document}

\begin{pspicture}(-.4,-.2)(3.8,3)
\psbarcode{1-909440-07-4}{includetext guardwhitespace}{isbn}
\end{pspicture}

\end{document}
```
Input

How did I work out the co-ordinates for the bounding box? I put the picture inside the argument of \frame, which marks the picture's extent with a rectangle and then adjusted the co-ordinates until the picture fitted inside the frame. Like this:

Input

```
\frame{%
\begin{pspicture}(-.4,-.2)(3.8,3)
\psbarcode{1-909440-07-4}{includetext guardwhitespace}{isbn}
\end{pspicture}%
}
```

Input

Remember that this example document must be compiled with `latex` rather than `pdflatex`. If you want to turn this into an image that you can include in another document, change the document class to standalone [81] and remove all unnecessary blank lines:

Input

```
% arara: latex
% arara: dvips
% arara: ps2pdf
\documentclass{standalone}
\usepackage{pst-barcode}
\begin{document}
\begin{pspicture}(-.4,-.2)(3.8,3)
```

```
\psbarcode{1-909440-07-4}{includetext guardwhitespace}{isbn}
\end{pspicture}
\end{document}
```

Input

(You can download or view this document.)

If the file is called barcode-isbn.tex then you need to run:

```
latex barcode-isbn
dvips -o barcode-isbn.ps barcode-isbn.dvi
ps2pdf barcode-isbn.ps barcode-isbn.pdf
```

Shell

Alternatively, if you use arara and have included the arara directives shown above, you can just do:

```
arara barcode-isbn
```

Shell

This creates a PDF file called barcode-isbn.pdf that you can now include in another document using:

```
\includegraphics{barcode-isbn}
```

Input

This produces:

Output

Exercise 28. A QR Code

A QR code is obtained by setting ⟨*type*⟩ to `qrcode`. The data setting ⟨*text or filename*⟩ is typically a website address, so for this exercise, adapt the code from Example 53 so that it's now a QR code. Set the data to a website of your choice. If you can't think of one, you can use this book's home page: `http://www.dickimaw-books.com/latex/admin/`.

You can download or view a solution to this exercise.

10.5 The flowfram Package and the flowframtk Application

The LaTeX kernel provides a single-column mode through the use of \onecolumn (or the onecolumn class option) and two-column mode through the use of \twocolumn (or the twocolumn class option). The flowfram package provides a way to extend this so that you can have an arbitrary number of columns of arbitrary width and height placed in arbitrary locations.

The standard LaTeX declarations \onecolumn and \twocolumn automatically insert a page break before switching modes. One of the reasons for this page break is to ensure that there isn't a paragraph spanning different width columns as TeX's output routine doesn't set the line width until the end of the paragraph which would leave the tail end of the paragraph with the incorrect width at the start of the new column. The flowfram package doesn't automatically insert page breaks in this manner, but this inherent problem caused by the asynchronous behaviour of TeX's output routine is present if you have a paragraph spanning different width columns that have been defined using flowfram, so you need to take care.

These arbitrary columns are termed "flow frames" in the flowfram user manual [97], which is accessed using

```
texdoc ffuserguide
```

Shell

(If you just do

```
texdoc flowfram
```

Shell

you'll get the documented code instead.)

There are two other types of frames: "static frames" and "dynamic frames". These two types of frames need to have their contents set explicitly. With static frames, the contents are typeset in a box on being set. With dynamic frames, the contents are stored in a macro and retypeset each time the frame is drawn on a page. The frames are drawn onto the page in the following order: static, flow, dynamic. Within each category, the frames are drawn in order of definition. If frames overlap, their contents will overlap. (In other words, the text inside one frame doesn't attempt to avoid collision with the text inside another frame.)

Frames may have a border drawn around them. The default border is just a rectangle, but other borders may be used. The paragraph shape is unaffected by the shape of the border but it can be changed either using TEX's \parshape primitive or one of the commands provided by the shapepar package [4].

Flow frames can be defined using

\newflowframe[⟨*page list*⟩]{⟨*width*⟩}{⟨*height*⟩}{⟨*x*⟩}{⟨*y*⟩}[⟨*label*⟩] Definition

Static frames can be defined using

\newstaticframe[⟨page list⟩]{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}[⟨label⟩] Definition

Dynamic frames can be defined using

\newdynamicframe[⟨page list⟩]{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}[⟨label⟩] Definition

Each of these commands has a starred version that adds a rectangular border to the frame. The arguments are as follows:

⟨*page list*⟩ The list of pages on which this frame is visible. This may be one of the keywords: `all`, `none`, `odd` or `even`; or this may be a comma-separated list of page numbers or page ranges. Page ranges may be open-ended using <⟨*n*⟩ for pages less than page ⟨*n*⟩ or >⟨*n*⟩ for pages greater than page ⟨*n*⟩; or the ranges may be closed using ⟨*n*⟩-⟨*m*⟩ for pages between ⟨*n*⟩ and ⟨*m*⟩, inclusive. The page numbers referenced in the ⟨*page list*⟩ by default refer to the decimal value of the page counter. (For example, 1 means page 1 or page i or page I.) If the `pages=absolute` package option is used, then the page number refers to the absolute page number. If ⟨*page list*⟩ is omitted, `all` is assumed.

⟨*width*⟩ The width of the frame.

⟨*height*⟩ The height of the frame.

⟨x⟩ The x-coordinate of the bottom left-hand corner of the frame relative to the typeblock.

⟨y⟩ The y-coordinate of the bottom left-hand corner of the frame relative to the typeblock.

⟨*label*⟩ A label that uniquely identifies this frame.

The typeblock is the area where the text would be typeset if the document was in the regular one-column mode. Each frame can be referenced either by its label or by its index (starting from 1 for each frame type). For example, the first flow frame to be defined has an index equal to 1, and the first static frame to be defined also has an index equal to 1.

The contents of a static frame can be set using:

`\setstaticcontents{`⟨*id*⟩`}{`⟨*contents*⟩`}` Definition

where the frame contents are given by ⟨*contents*⟩. For the unstarred version, ⟨*id*⟩ is the frame's index. For the starred version, ⟨*id*⟩ is the label. Alternatively, the contents can be set using the staticcontents environment:

`\begin{staticcontents}{`⟨*id*⟩`}`
⟨*contents*⟩ Definition
`\end{staticcontents}`

As before there is a starred version where ⟨*id*⟩ is the frame's label rather than its index.

The contents of a dynamic frame can be set in a similar manner using:

`\setdynamiccontents{`⟨*id*⟩`}{`⟨*contents*⟩`}` Definition

or using the dynamiccontents environment:

`\begin{dynamiccontents}{`⟨*id*⟩`}`
⟨*contents*⟩ Definition
`\end{dynamiccontents}`

Unlike the static frames, you can also append text to a dynamic frame using:

`\appenddynamiccontents{`⟨*id*⟩`}{`⟨*text*⟩`}` Definition

As before, these all have starred versions where ⟨*id*⟩ is the frame's label.


Note that verbatim text isn't permitted in a dynamic frame, even with the environment version. Make sure that you set the contents before the frame is displayed on the page.

Example

Static frames are useful for background images, as shown below. This is a simple document that uses the lipsum package [32] to generate dummy text. This is a contrived example that shows what happens when you have

overlapping frames and what happens if you have a paragraph spanning flow frames of different widths. The document text is placed in the flow frames in the order of the frame definition, which is why the text starts on the right hand frame, as that was the first flow frame to be defined.

↑ Input

```
\documentclass[a4paper]{article}

\usepackage{lipsum}
\usepackage{graphicx}
\usepackage{flowfram}

\newflowframe
  {0.6\textwidth}% width
  {0.3\textheight}% height
  {0.4\textwidth}% x position
  {0.7\textheight}% y position

\newflowframe{0.6\textwidth}{0.5\textheight}{0pt}
{0.5\textheight}

\newflowframe*{\textwidth}{0.4\textheight}{0pt}{0pt}
```

```
\newstaticframe{2in}{2in}{0pt}{0pt}

\setstaticcontents{1}{\includegraphics[height=2in]{chicken}}

\begin{document}

\lipsum[1-4]

\end{document}
```

↓ Input

The rather unpleasant result is shown in Figure 10.6.

The issue caused by TEX's asynchronous output routine being unable to adjust the line width over a frame break can be seen by the shortened lines in the end part of the paragraph at the beginning of the lower frame. (The frame with the border.) The flowfram package notices the problem and issues a warning with a recommendation:

```
Package flowfram Warning: Moving to flow frame of unequal width,
(flowfram)                use of \framebreak advised, or text
might not appear
correctly (difference = 137.9979pt, tolerance = 2.0pt)
```

Figure 10.6 Overlapping Frames (flowfram package)

As with all manual interventions, the use of

```
\framebreak
```

Definition

before the first word of the new frame (between "magna." and "Nunc" in this example) should only be resorted to on the final version of the document, once all the text has been written.

Example 54. Advance Information Sheet

This example uses the flowfram package to create an advance information sheet for a book. The geometry package is used to set the margins. The page numbering is suppressed using the empty page style and the section numbering is suppressed by setting the secnumdepth counter to 0. (This just saves me from remembering to use the starred version of \section.) I've used the drm package [28] to illustrate a different font from those I've previously used, and I've used the pifont, which provides the dinglist environment that was mentioned in Volume 1 [92, §8.2]. Recall \dimexpr from §2.1.3, which is used here to calculate the position of the static frame.

↑ Input

```
\documentclass[12pt,a4paper]{article}

\usepackage[utf8]{inputenc}
```

```
\usepackage[T1]{fontenc}
\usepackage{drm}

\usepackage{pifont}
\usepackage[margin=0.5in]{geometry}
\usepackage{graphicx}
\usepackage{flowfram}

\pagestyle{empty}
\setcounter{secnumdepth}{0}

\newflowframe{0.6\textwidth}{\textheight}{0pt}{0pt}

\newdynamicframe{.3\textwidth}{\textheight}{.7\textwidth}{0pt}[sidepane]

\newstaticframe{2in}{2in}{\dimexpr(\textwidth-2in)}{0pt}[logo]

\setstaticcontents*{logo}{\includegraphics[width=2in]{dummy-logo}}

\begin{dynamiccontents*}{sidepane}
{\raggedright\bfseries\scshape\Large
Oh No! The Chickens Have Escaped
```

```
\par
}

\centering
\bigskip

\includegraphics[width=\linewidth]{chicken}

\bigskip

\begin{tabular}{@{}ll}
Genre:    & Children's Illustrated \\
          & Fiction\\
RRP:      & £5.99\\
Format:   & Paperback\\
Pages:    & 30\\
Pub Date: & 1st August 2014\\
ISBN:     & 978-x-xxxxxx-xx-x
\end{tabular}

\vfill
```

```
\includegraphics{barcode-qr}

\vfill

\end{dynamiccontents*}

\begin{document}\raggedright

\section{About the Book}

A fun illustrated children's story about some escaped chickens.
Fred and Mabel are looking after Granny's chickens for the day
but, oh no, they've escaped. Will Fred and Mabel find them all
before the chickens get into the road or get eaten by the hungry fox?

\section{About the Author}

Dickie Duck lives somewhere or other and won the best fowl book
award in 2014. He likes writing silly stories about ducks and
chickens.

\section{Keypoints}
```

```
\begin{dinglist}{118}
 \item A fun way of teaching children to count.

 \item Children will enjoy the repetition and rhyme.

 \item Features chickens doing stupid things.

 \item Completely fictitious book encourages children's
imaginations.
\end{dinglist}

\section{Marketing}

\begin{dinglist}{118}
 \item Written by award-winning author.

 \item Illustrated by world famous artist.

 \item Some other really interesting marketing information.
\end{dinglist}
```

```
\section{Contact}

\begin{tabular}{@{}l}
Dickie Duck\\
1 The Street\\
Another Village\\
Some City\\
Imagineshire\\
YZ1 2AB
\end{tabular}

\end{document}
```

↓ Input

You can download or view this example document. It uses the sample images chicken.png and dummy-logo.png. It also uses the barcode-qr.pdf file created in Exercise 28.

The resulting document is shown in Figure 10.7.

If you find it a bit awkward to work out the dimensions and locations of the frames, there's a helper GUI application called flowframtk, which provides a graphical means of defining the frames. As with datatooltk

About the Book

A fun illustrated children's story about some escaped chickens. Fred and Mabel are looking after Granny's chickens for the day but, oh no, they've escaped. Will Fred and Mabel find them all before the chickens get into the road or get eaten by the hungry fox?

About the Author

Dickie Duck lives somewhere or other and won the best fowl book award in 2014. He likes writing silly stories about ducks and chickens.

Keypoints

- A fun way of teaching children to count.
- Children will enjoy the repetition and rhyme.
- Features chickens doing stupid things.
- Completely fictitious book encourages children's imaginations.

Marketing

- Written by award-winning author.
- Illustrated by world famous artist.
- Some other really interesting marketing information.

Contact

Dickie Duck
1 The Street
Another Village
Some City
Imagineshire
YZ1 2AB

Oh No! The Chickens Have Escaped

Genre: Children's Illustrated Fiction
RRP: £5.99
Format: Paperback
Pages: 30
Pub Date: 1st August 2014
ISBN: 978-x-xxxxx-xx-x

DUMMY LOGO

Figure 10.7 Advance Information Sheet

and arara, this is a Java application so, if you want to use it, make sure you have an up-to-date version of the Java runtime environment installed on your computer.

To install flowframtk download the installer from flowframtk's home page. This is a .jar file. If your operating system knows how to run a .jar file, you should just be able to double-click on it, otherwise you can run it from the command line using:

```
java -jar flowframtk-0.7-installer.jar
```
Shell

(You may need to specify the full path to the .jar file. The version number 0.7 may also need to be changed if a new version has been produced since the time of writing this.)

Once flowframtk has been installed, it can be run either from your operating system's applications menu or from the command line using:

```
flowframtk
```
Shell

The main window is shown in Figure 10.8.

Example 55. Advance Information Sheet (with flowframtk)
To illustrate the use of flowframtk, the rest of this section will use flowframtk to create the advance information sheet from Example 54. The result

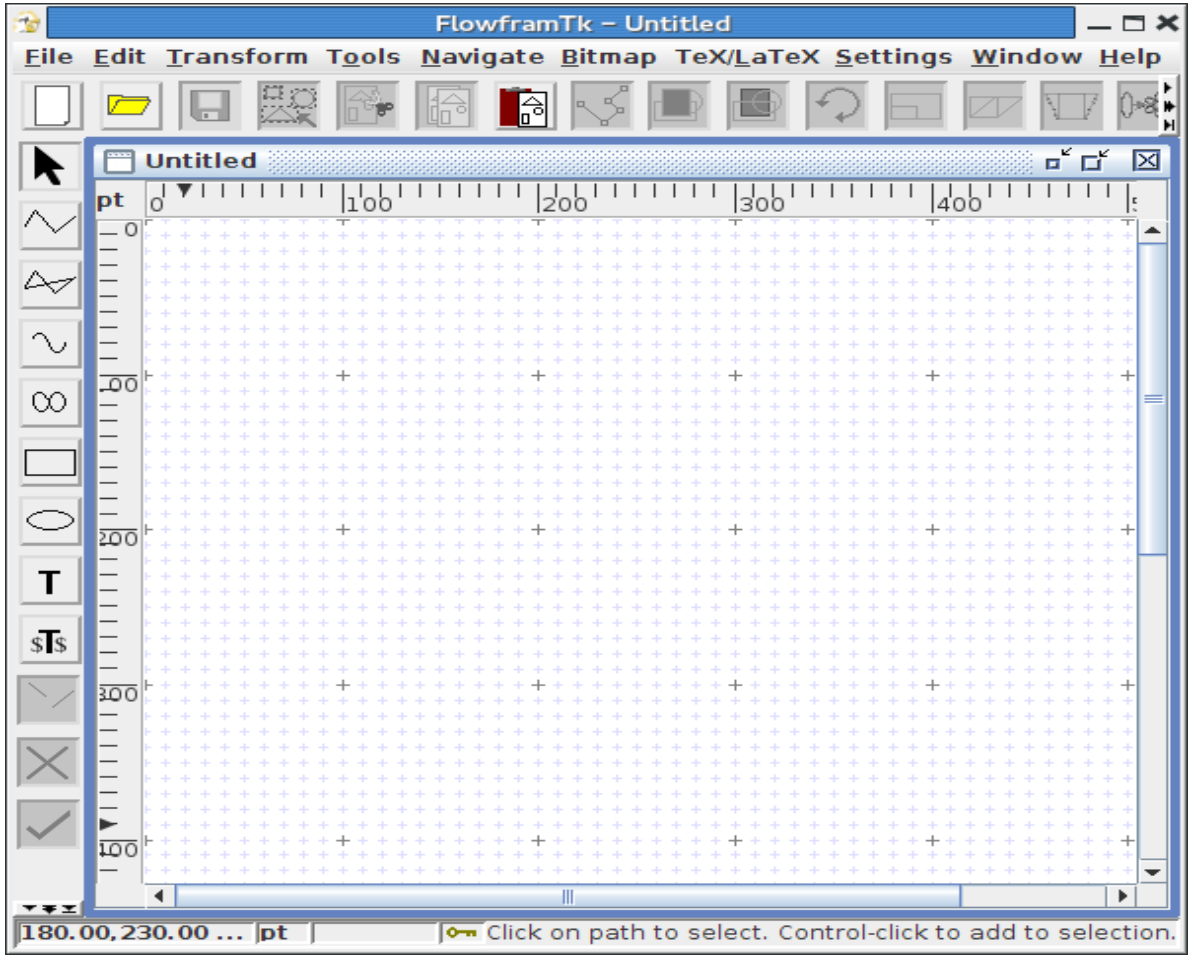


Figure 10.8 FlowframTk Main Window

will be slightly different as I'm going to add some extra features such as coloured backgrounds.

For this example, I prefer to work in metric measurements so, to reduce rounding errors, I'm first going to set the storage unit to millimetres and the grid to centimetres. To set the storage unit, go to Settings→Configure Image Settings. This opens the dialog box shown in Figure 10.9. Use the Storage Unit drop-down menu to change the unit to mm. Click on the green tick button to save this change and close the dialog window.

Next go to Settings→Grid→Grid Settings which will open the dialog window shown in Figure 10.10. Select the Rectangular tab (if not already selected) and change the major divisions to 1 cm and the sub-divisions to 2 or 4. Click on the green tick button to save this change and close the dialog window.

Next make sure the paper size is correctly set. I'm using A4 portrait paper, which can be selected through the Settings→Paper submenu. Now the TEX settings for the document need to be specified, so go to Settings →Configure TeX/LaTeX Settings which will open the dialog window shown in Figure 10.11.

Recall from Example 54 that I used the 12pt class option. I can specify that I want this by changing the Normal Font Size drop-down menu to 12. I'm going to use the default article class so I've left the Use default class radio button selected. If I want to use a different class, for example scrartcl, then

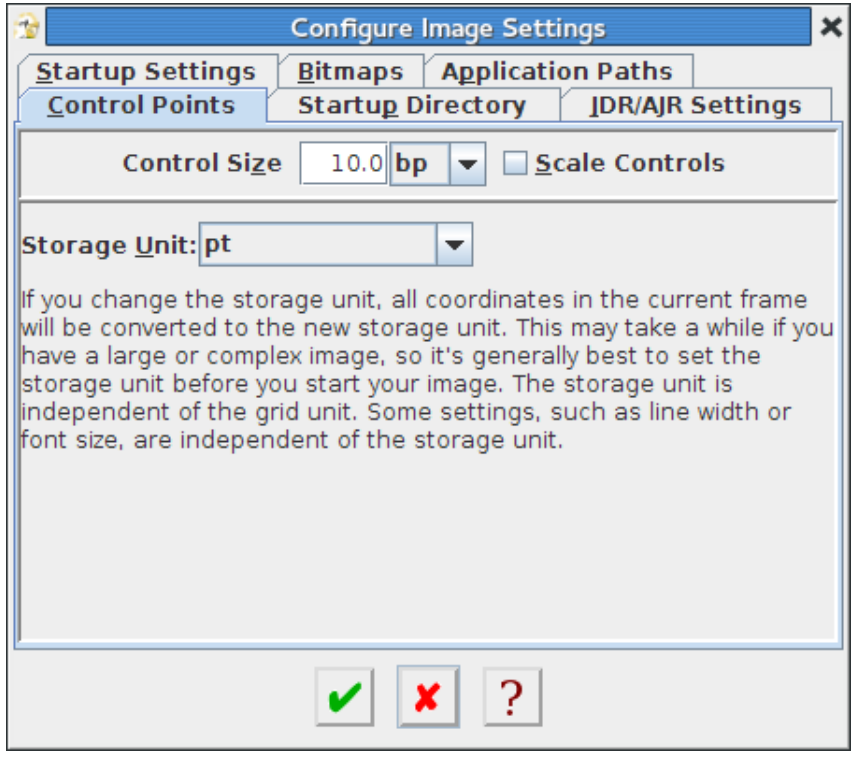


Figure 10.9 FlowframTk — Set the Storage Unit

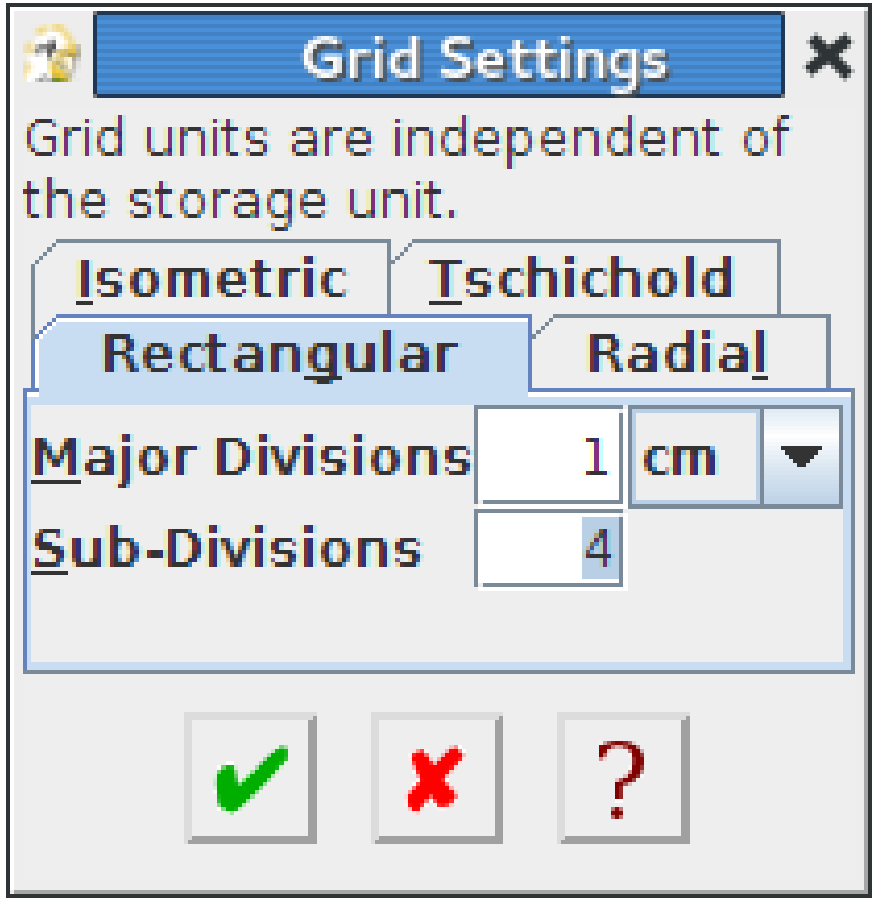


Figure 10.10 FlowframTk — Set the Grid

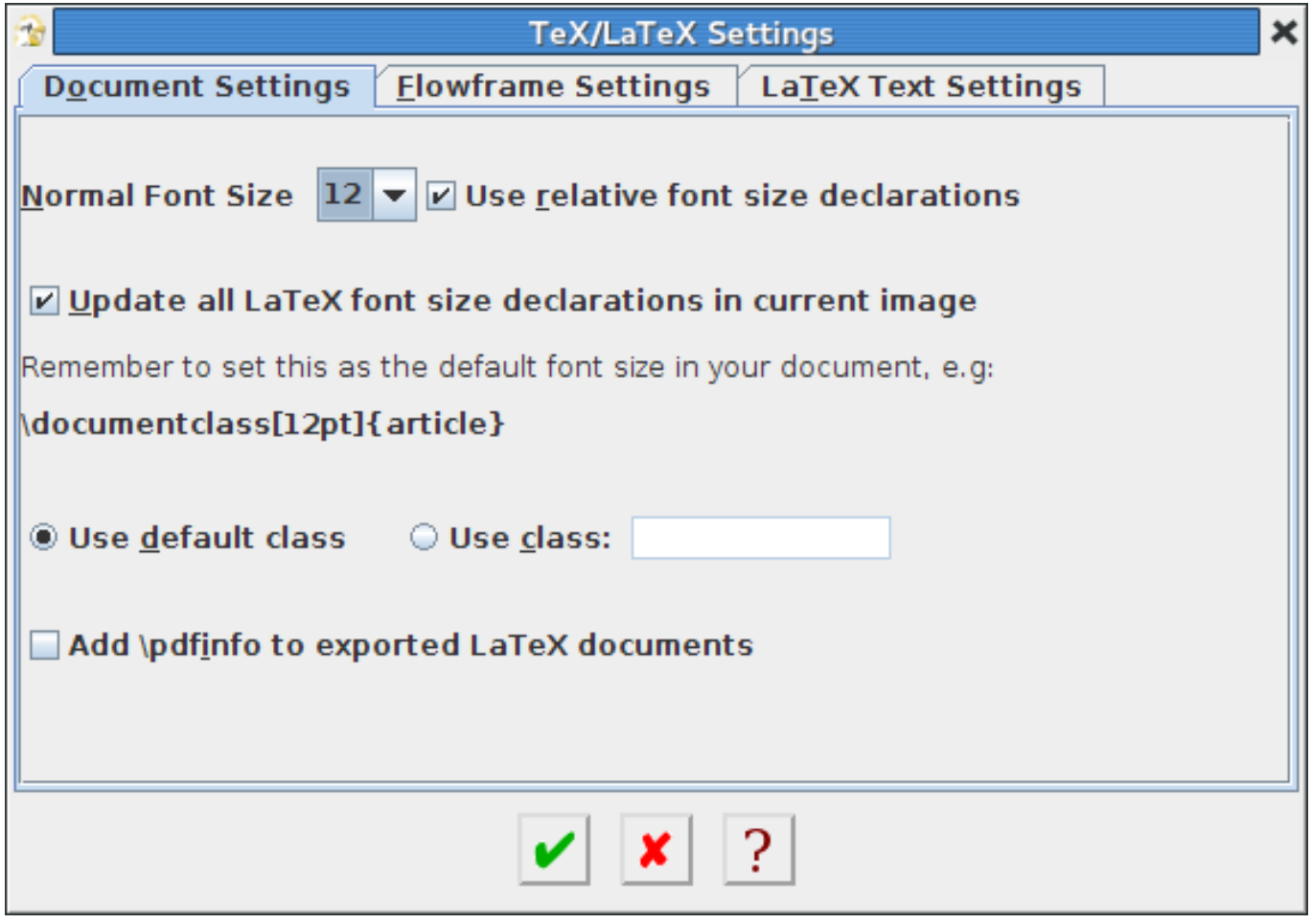


Figure 10.11 FlowframTk — Set the TEX/LATEX Settings

I would have to select the Use class radio button and type in the class name (without the extension) in the neighbouring field. Again, click on the green tick button to save the changes and close the window.

Next I need to specify the packages I want to use. This is done in the preamble editor, which is opened using the TeX/LaTeX→Preamble menu item. Note that I don't include the geometry or flowfram packages since these will automatically be added when I later use the export function. Here are the packages (as from the previous example) and I've also added the code to set the empty page style and switch off the section numbering, but I've deferred it using \AtBeginDocument:

↑ Input

```
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{drm}

\usepackage{pifont}
\usepackage{graphicx}

\AtBeginDocument{%
  \pagestyle{empty}%
  \setcounter{secnumdepth}{0}%
```

```
}
```

↓ Input

See Figure 10.12.

You can choose whether to show or hide the grid using the Settings→Grid sub-menu. I'm going to hide the grid but keep the rulers visible. Now I need to specify the margins. In Example 54 I had 0.5 in margins, but now I want to have a borderless document, so I'm going to set all the margins to zero. This is done using the TeX/LaTeX→Flow Frames→Set Typeblock menu item, which opens the dialog shown in Figure 10.13. Make sure all the margins are set to 0. (For this example, you don't need the other settings in this dialog window.) Although the margins default to 0, you must still click on the green tick button to save and close the dialog. A grey rectangle should now be displayed on the canvas with the word "typeblock" in the bottom left hand corner of the rectangle. (With zero margins, it may be difficult to see the rectangle.)

Now select the rectangle tool either using the Tools→Rectangle menu item or click on the rectangle button in the left tool bar. To draw a rectangle on the canvas, click where you want one corner and move the mouse to the location where the opposite corner should be and click there. For example, in Figure 10.14, I've created a rectangle with top left corner at (0 mm, 0 mm) and bottom right corner at (130 mm, 297 mm).

This rectangle will represent the main flow frame in my document, but

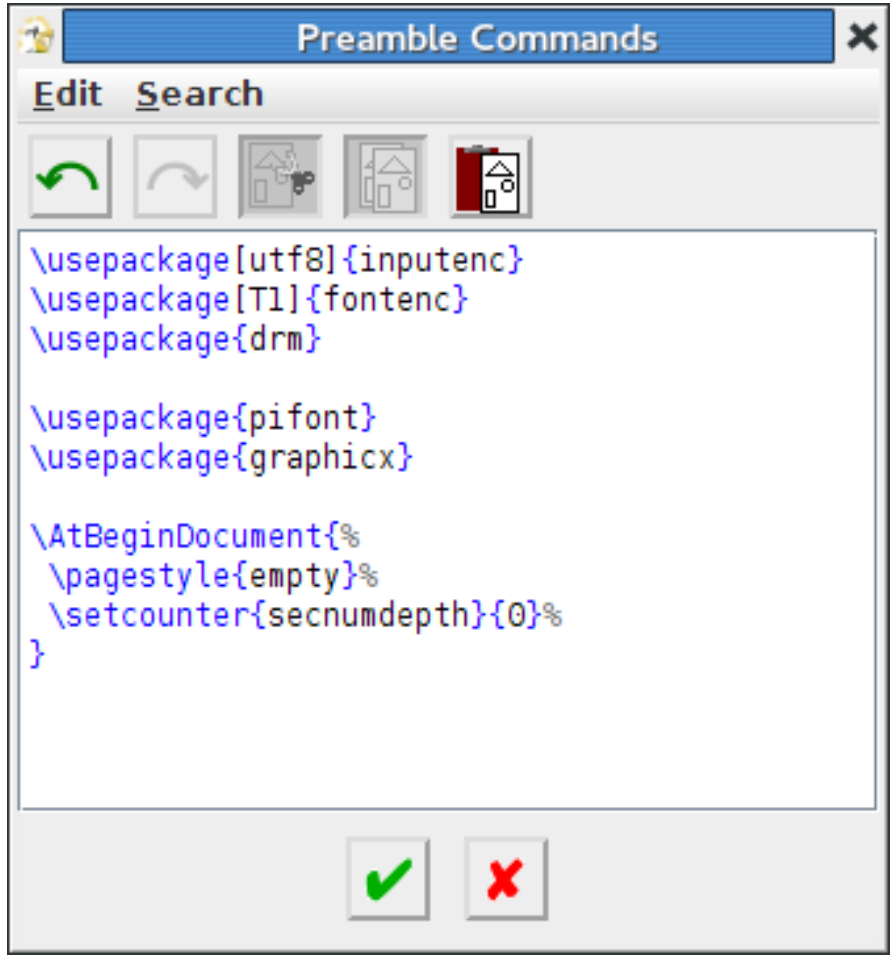


Figure 10.12 FlowframTk — Setting the Preamble

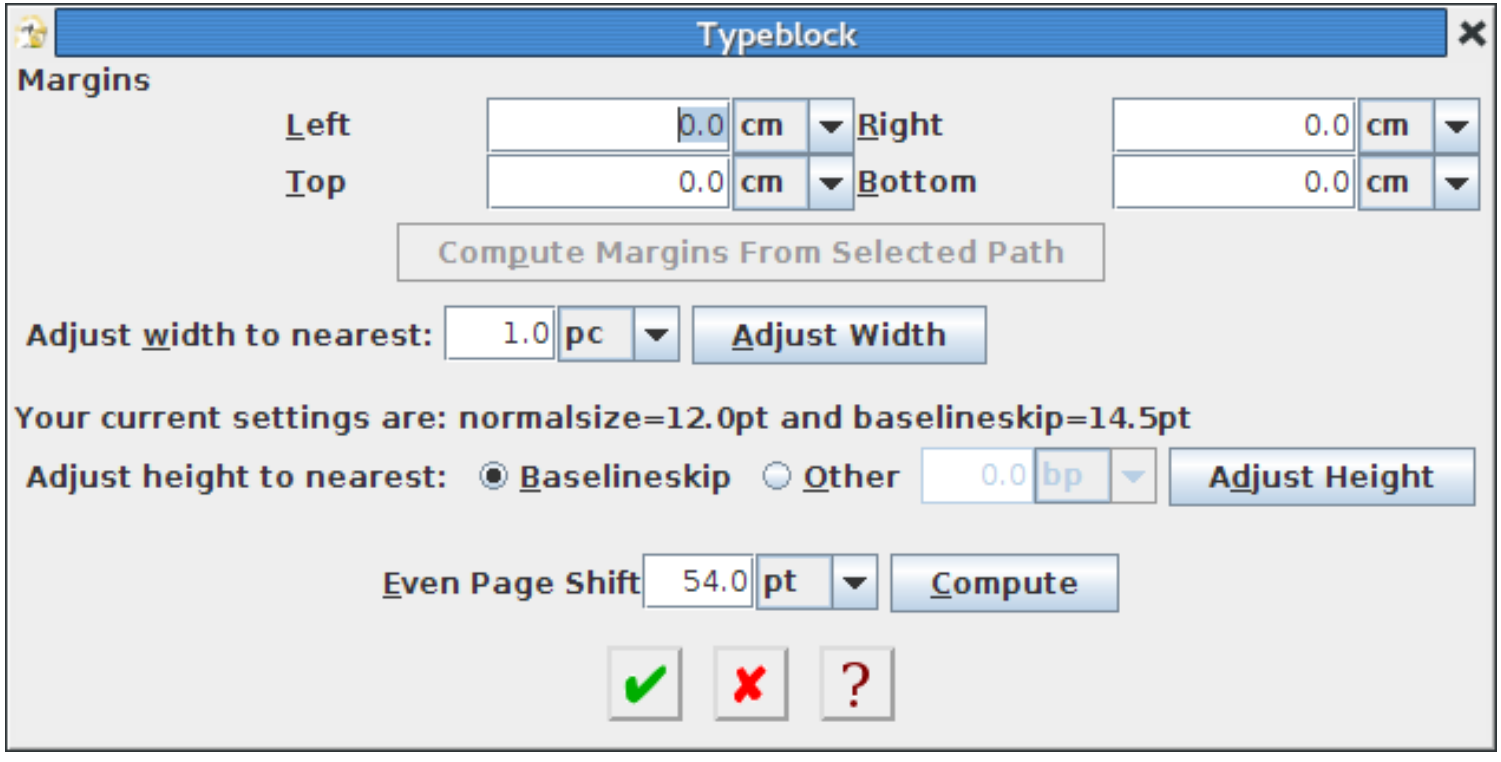


Figure 10.13 FlowframTk — Setting the Margins

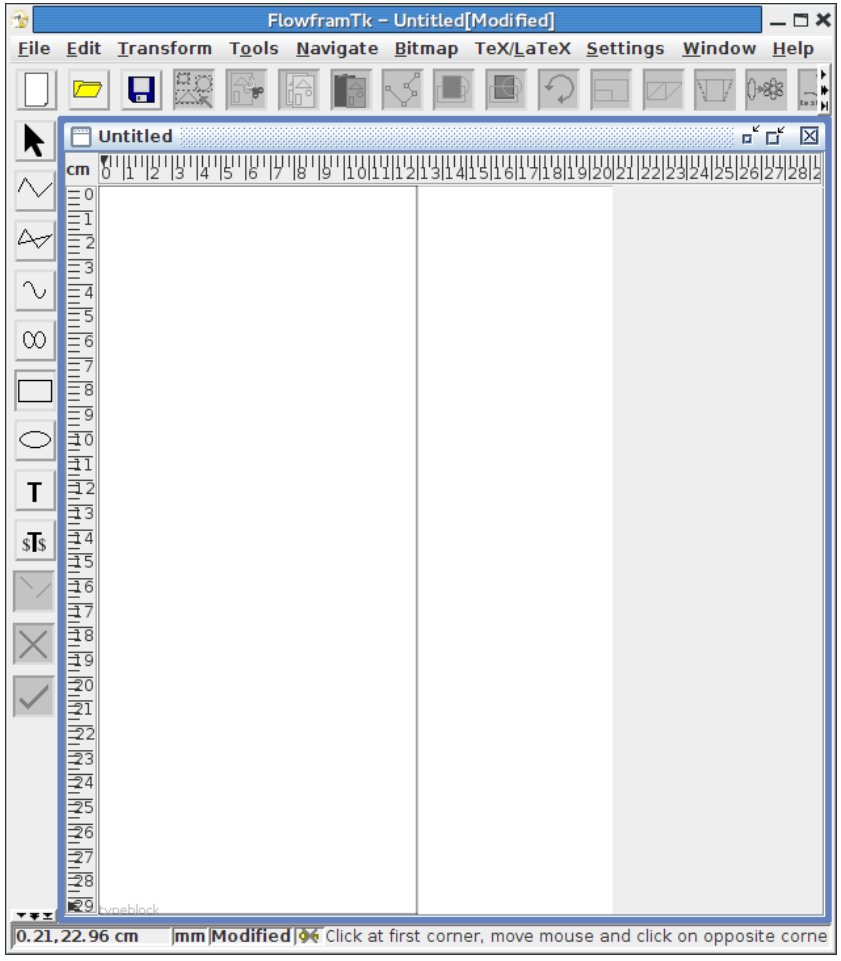


Figure 10.14 FlowframTk — Create a Rectangle

first I want to add a fill colour and remove the outline. To do this, I need to switch to the select tool (either using the Tools→Select menu item or by clicking on the button in the side bar with an arrow on it). Then I can click anywhere inside the rectangle to select it. When it's selected, a dashed red rectangle will appear around it. Then I can use the Edit→Fill Colour menu item to open the fill colour selector. In Figure 10.15, I have selected the Colour radio button and specified 20% cyan, 20% magenta, 0% yellow and 0% black. The rectangle's black outline can be removed by selecting the Edit→Path→Line Colour menu item, which will open the line colour selector. Click on the Transparent radio button to remove the outline.

This rectangle now needs to be identified as a flow frame. Make sure it's still selected and use the menu item TeX/LaTeX→Flow Frames→Set Frame to open the flow frame selector. Set the Type to Flow and this will enable the flow frame related options. Give the frame a label (for example, "main") and select the Border As Shown option. The margins can also be set to prevent the document text running against the border. I've chosen 5 mm for each margin, as shown in Figure 10.16. The even page options can be ignored since the result will be a single-paged document. Again click on the green tick to save the changes and close the dialog window. The selected rectangle in the main window should now have a pale grey rectangle inside it that shows the frame's margins (see Figure 10.17).

Now I want to create a static frame in the currently unfilled narrow

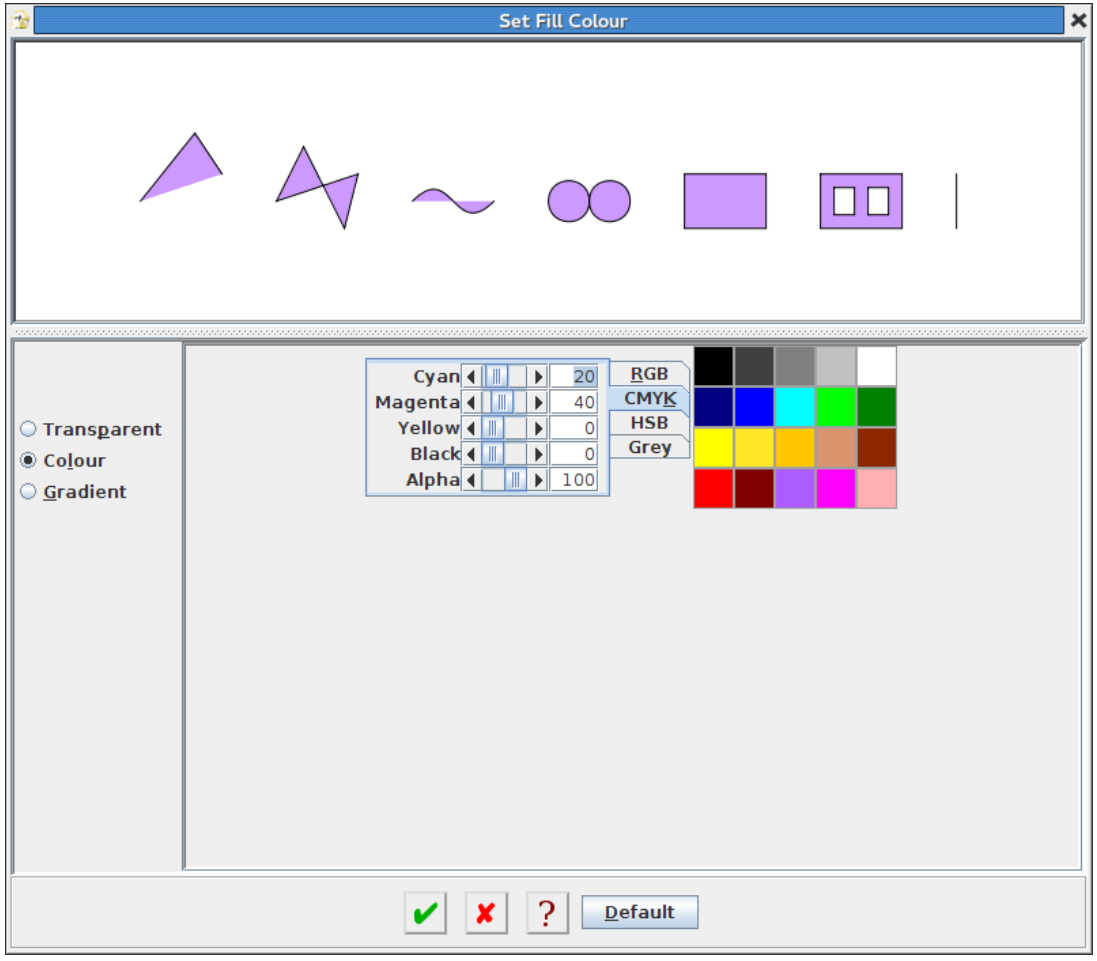


Figure 10.15 FlowframTk — Setting the Fill Colour

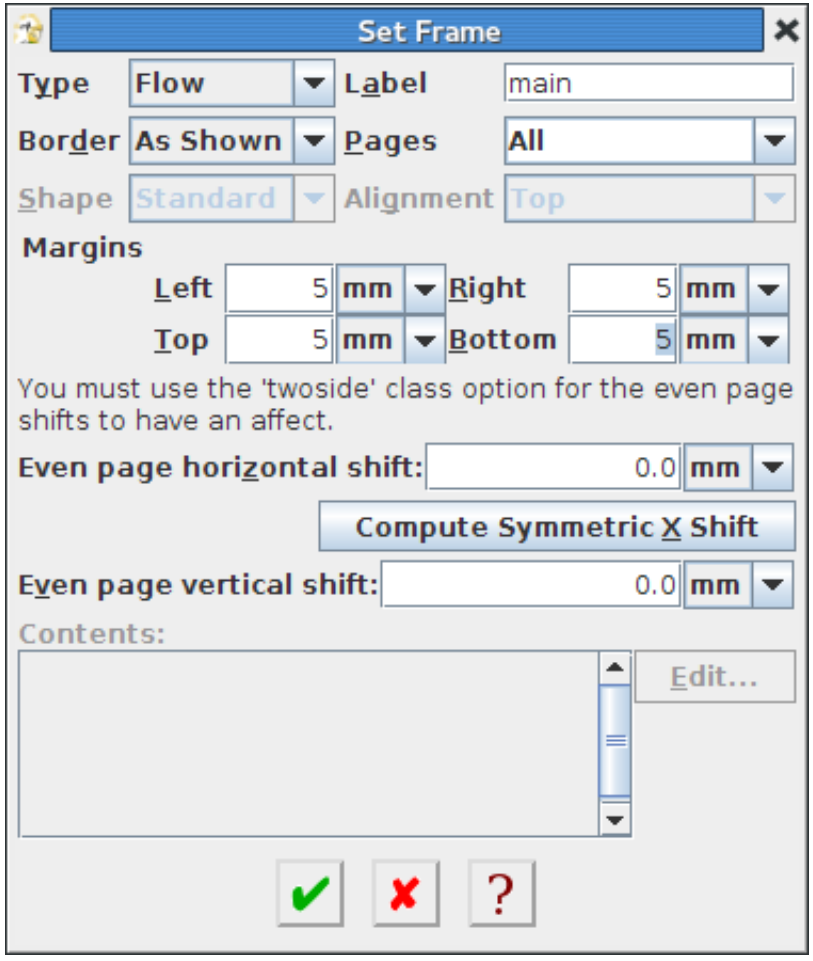


Figure 10.16 FlowframTk — Setting Flow Frame Data

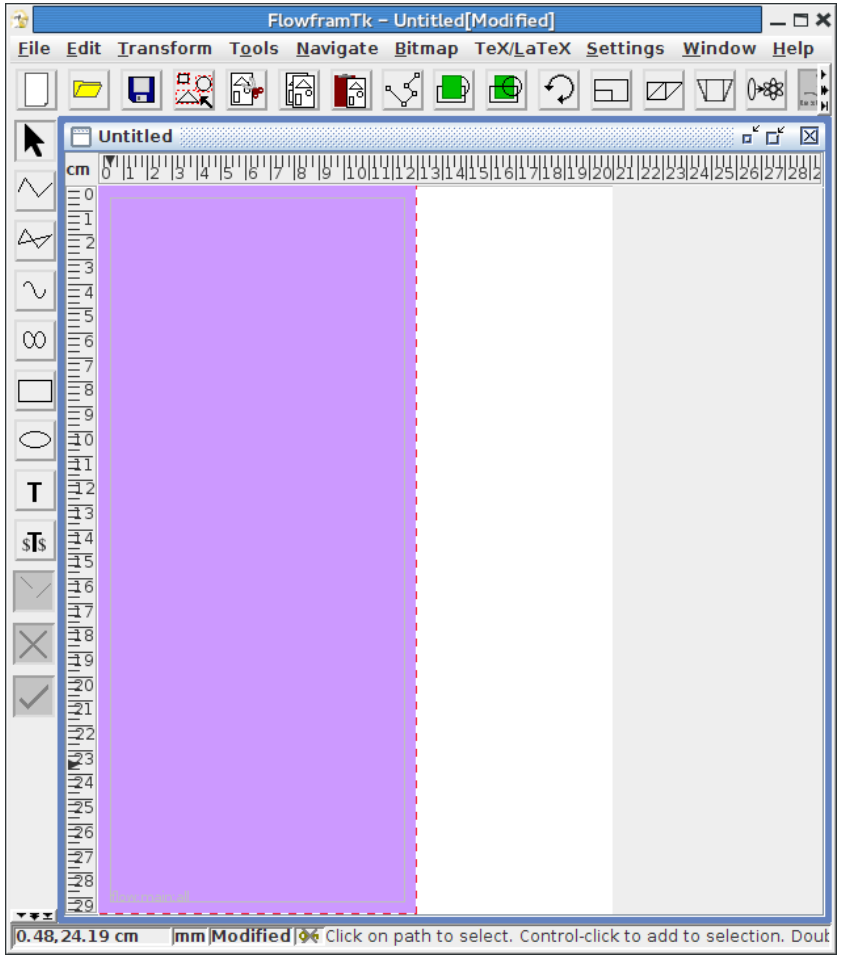


Figure 10.17 FlowframTk — Flow Frame Data Assigned

region on the right hand side of the page. This will just provide a background colour. I will later make some other frames on top of this one that will contain text and images. The process is much the same as for the previous frame. The rectangle tool is selected, a rectangle is drawn with opposing corners at (132 mm, 0 mm) and (210 mm, 297 mm). The fill colour is set to 0% cyan, 0% magenta, 50% yellow, 0% black, and the outline is set to transparent. However, the frame should now be a static frame, so once the rectangle has been created, select the menu item TeX/LaTeX→Flow Frames→Set Frame and set the Type to Static. This enables a different set of options to earlier, but for this frame the only extra information needed is the label. I've set this to "sidepane", as shown in Figure 10.18. Again, click on the green tick button to save the changes and close the dialog window.

It's a good idea at this point to save the image in case something goes wrong. There are two native file formats: `.jdr` (binary) and `.ajr` (ASCII). The binary version has greater precision but the ASCII version works better with version control systems (see §13.2). Since I use version control and I don't need double-precision for my co-ordinates, I'm going to use the ASCII version (`.ajr`). To save to a new file use the File→Save As menu item and select the appropriate file type (in my case, flowframtk ascii file (*.ajr)), as shown in Figure 10.19.

Now I need an area in which to put the book title (which appears on the top right of Figure 10.7). Again I need to use the rectangle tool to use as a

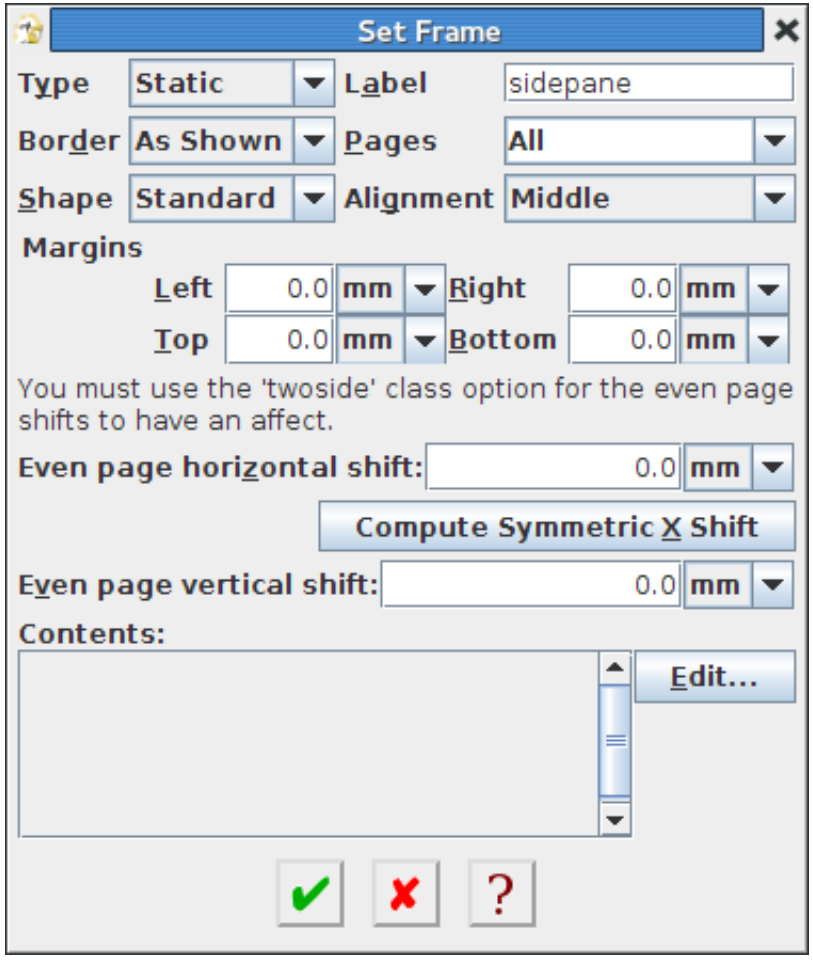


Figure 10.18 FlowframTk — Setting Static Frame Data

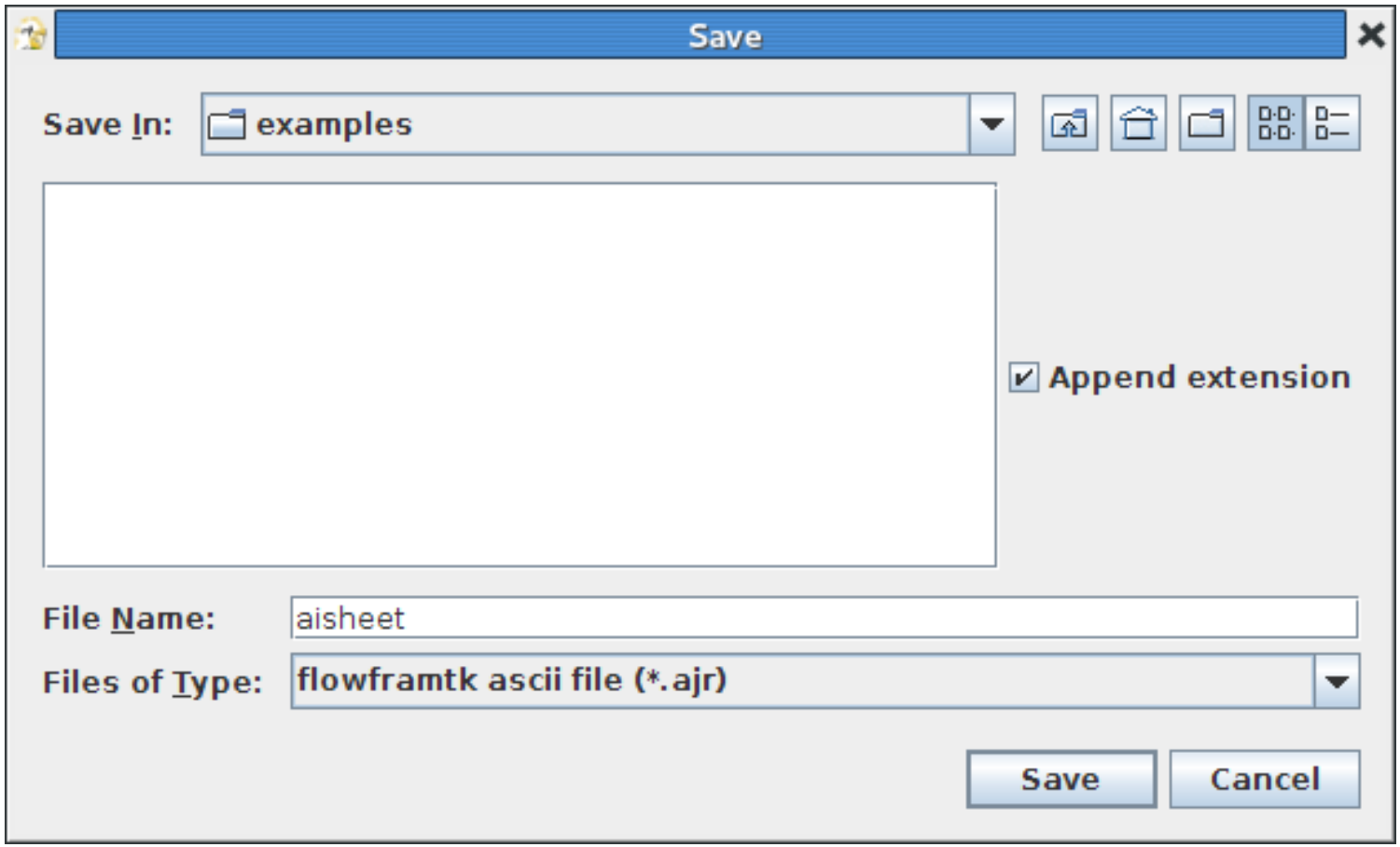


Figure 10.19 FlowframTk — Saving the Image

guide for my frame. My rectangle has opposing corners at (136 mm, 5 mm) and (206 mm, 62 mm). In this case I don't need to change the colours as I'm only using the rectangle as a guide to define a borderless dynamic frame. As before, I need to select this new rectangle and use the TeX/LaTeX→Flow Frames→Set Frame menu item to open the dialog box. Now I set the Type to Dynamic and the label to "title", but this time the Border option needs to be set to None as shown in Figure 10.20.

At the bottom of this dialog window is an area labelled Contents with a button labelled Edit next to it. This allows you to set the contents of a dynamic or static frame, just as you can do using commands such as \setdynamiccontents, described earlier. Click on this Edit button to add the frame contents, which is the LaTeX code from Example 54:

Input

```
\raggedright\bfseries\scshape\Large
Oh No! The Chickens Have Escaped
```

Input

As shown in Figure 10.21. Click on the green tick button to save these changes and return to the previous dialog window, and click on the green tick button there to save and close that window.

Links to bitmap images can be included in flowframtk images and they can also be used as frame backgrounds. Note that only the bitmap location

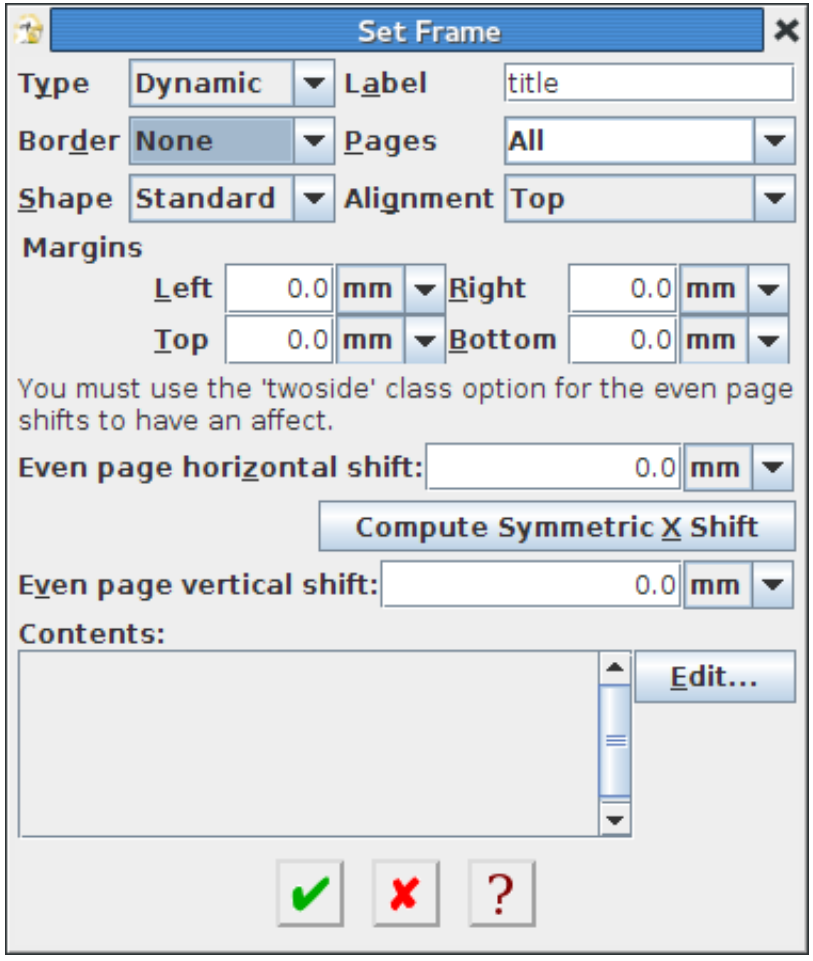


Figure 10.20 FlowframTk — Assigning Dynamic Frame Data

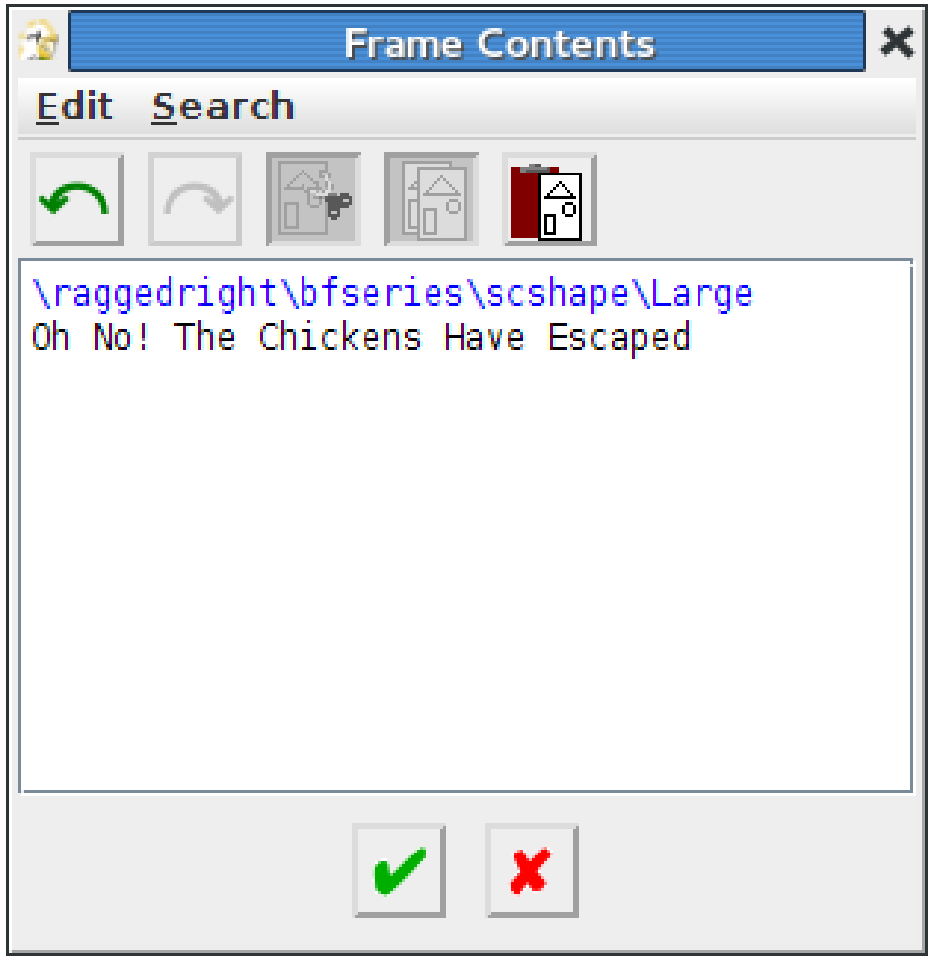


Figure 10.21 FlowframTk — Setting the Frame Contents

is saved in an .ajr or .jdr file. You can choose whether this location is an absolute path or a path relative to the .ajr/.jdr file. To do this, use the Settings→Configure Image Settings menu item to open the image settings dialog window (shown earlier in Figure 10.9). Select the Bitmaps tab and either check or uncheck the Use relative paths for bitmaps button. You can also use this panel to select which LaTeX command to use for inserting images. In Figure 10.22 I have set this to \includegraphics.

To insert a bitmap, make sure you are in select mode (Tools→Select) and use the menu item Bitmap→Insert Bitmap to open the bitmap selector, shown in Figure 10.23.

Select the required bitmap (I've chosen the sample image chicken.png) and click on the Open button. This will insert the bitmap into the image, but this particular image is far too big so it needs to be scaled. Note that, just like the transformation options in \includegraphics, this scaling doesn't modify the actual bitmap file. The newly inserted bitmap should already be selected. To scale it use the Transform→Scale menu item to open the scaling dialog window and set the scale factor, as shown in Figure 10.24. I've set the scale factor to 0.1.

The bitmap is now the correct size but is in the wrong position, as new bitmaps are always inserted with the top left corner at the origin. With the select tool set, you can use the mouse to drag the bitmap to the desired location, as shown in Figure 10.25.

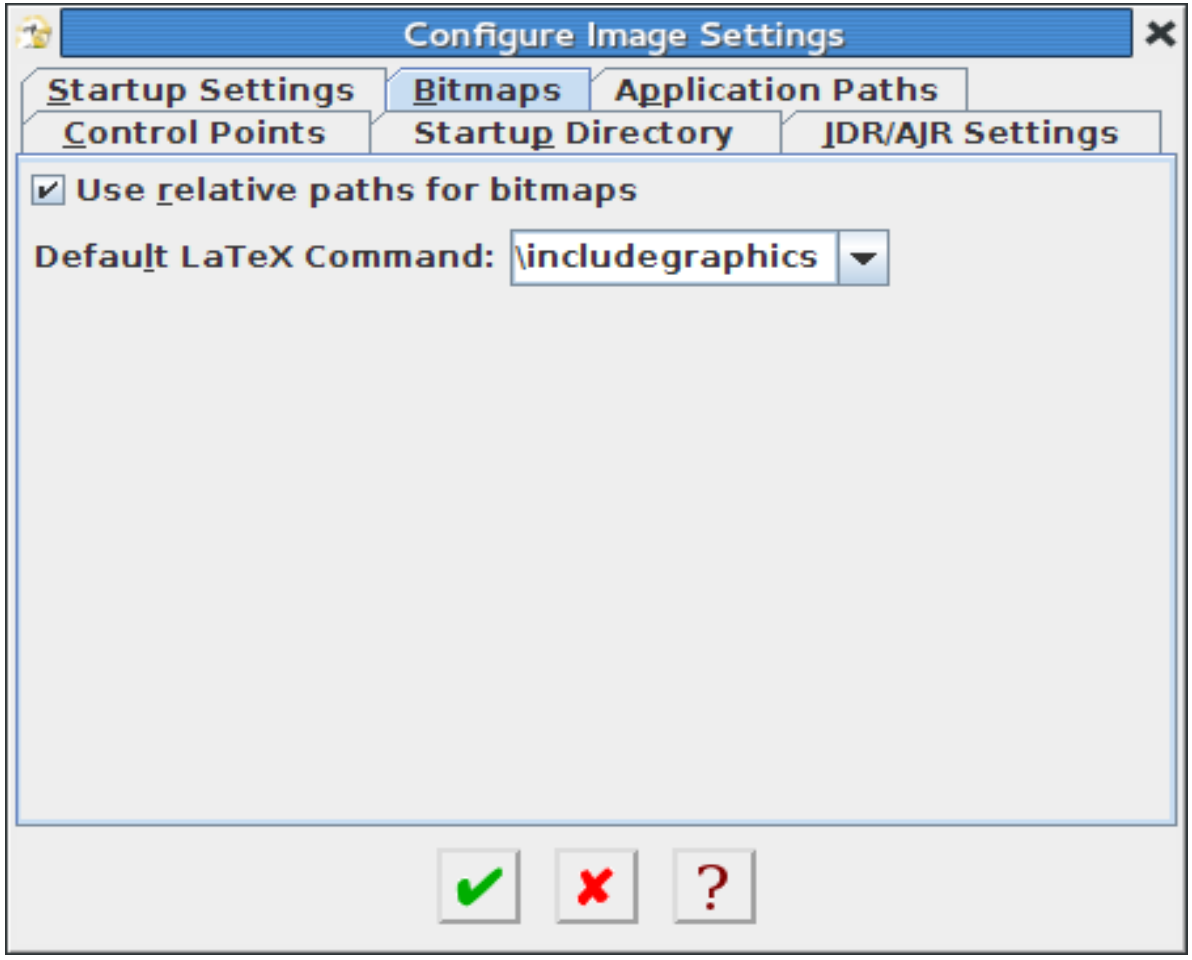


Figure 10.22 FlowframTk — Bitmap Options

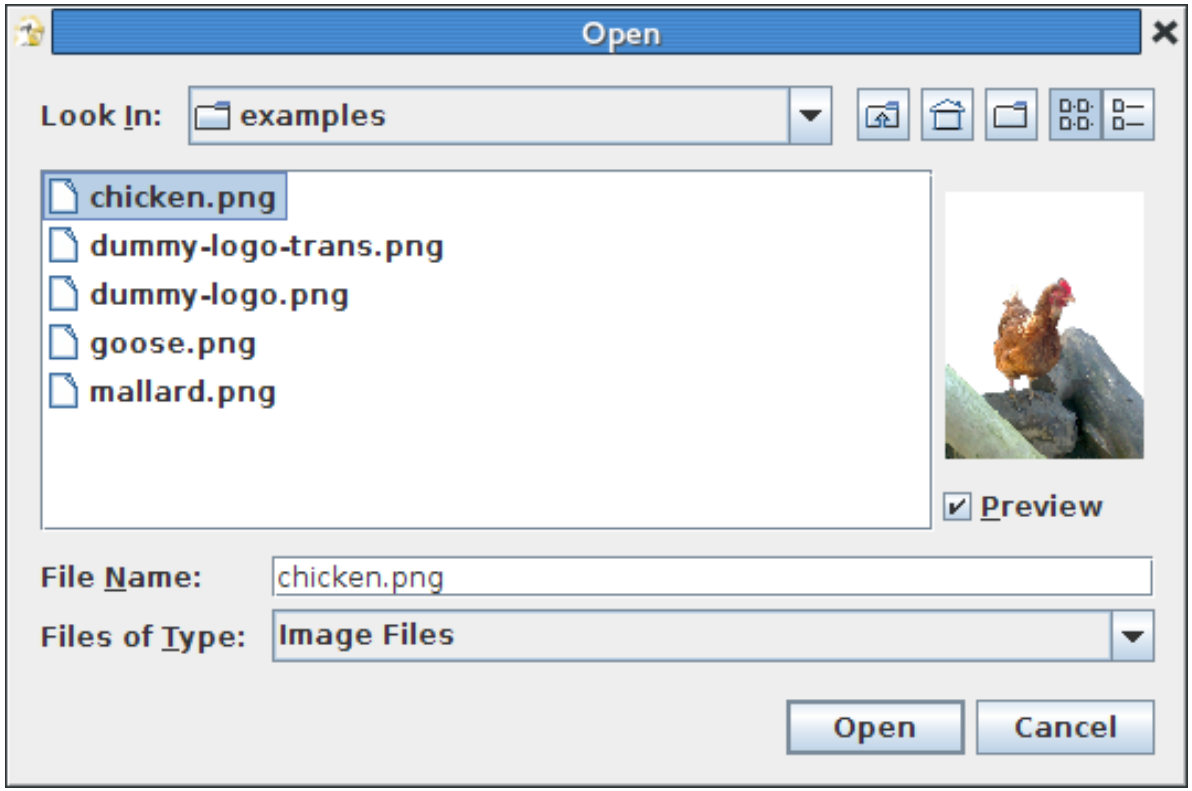


Figure 10.23 FlowframTk — Bitmap Selector

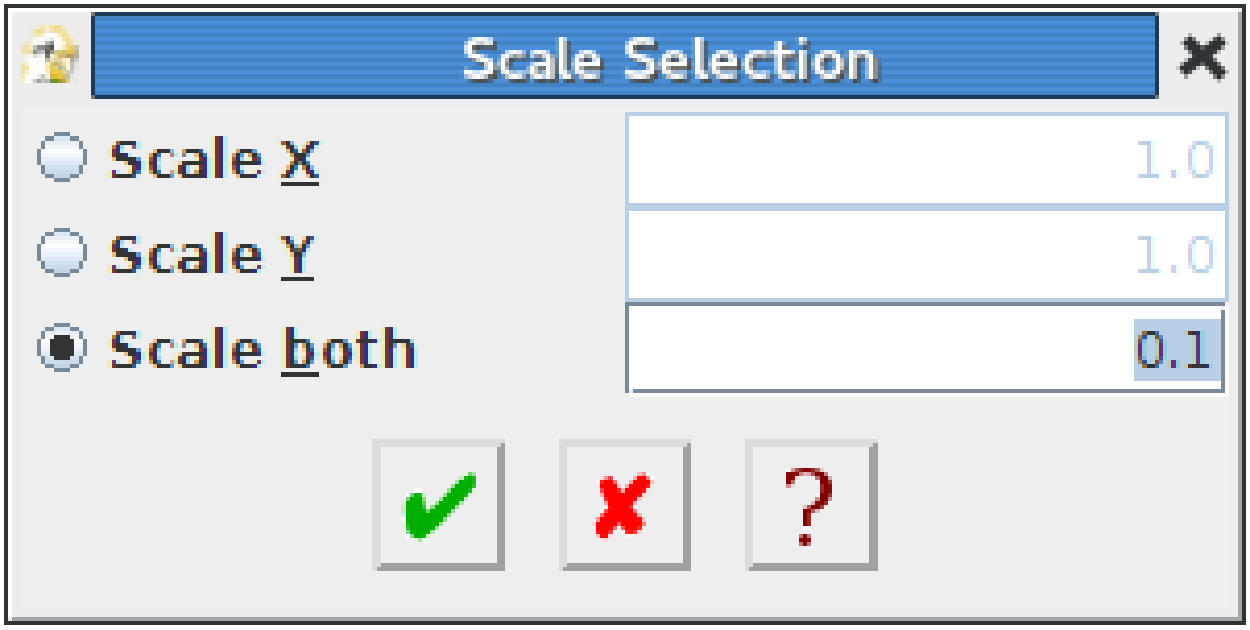


Figure 10.24 FlowframTk — Scaling

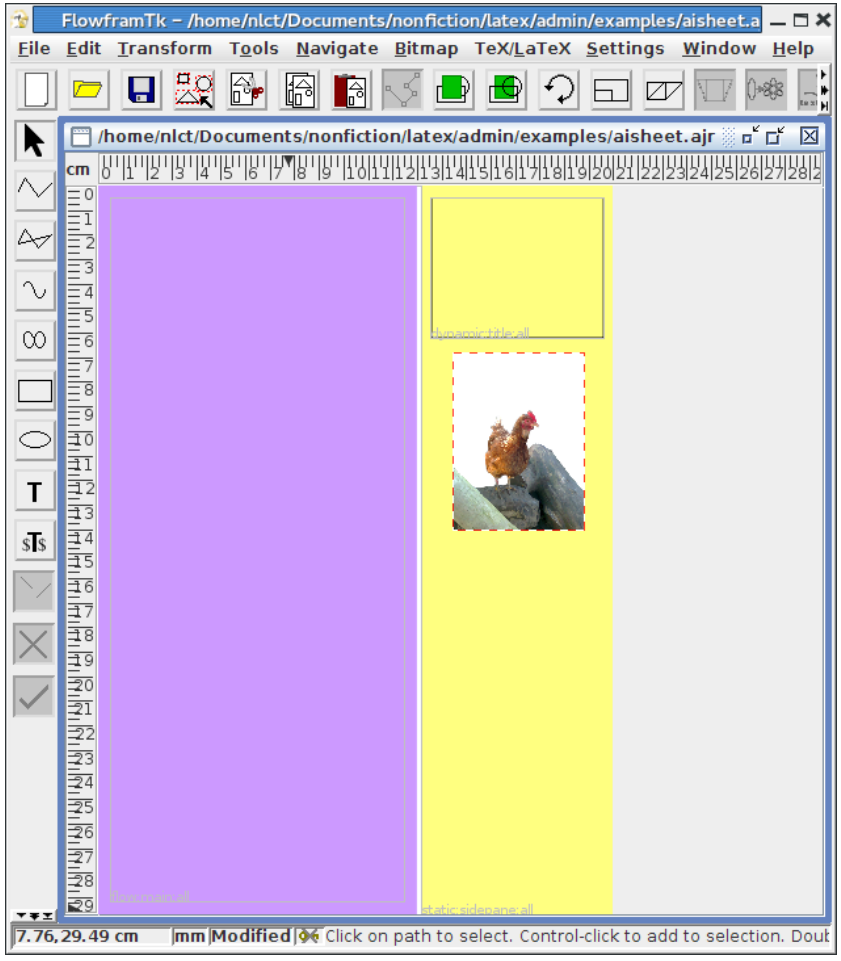


Figure 10.25 FlowframTk — Bitmap Moved to the Right

This bitmap can be set as the background to a static frame in a similar manner to the earlier static frame (labelled "sidepane"). So make sure the bitmap is selected, use the TeX/LaTeX→Flow Frames→Set Frame menu item to open the dialog box, and set the Type to Static, the label to, say, "titleimage" and make sure the Border is set to As Shown. I repeated this process for the sample image dummy-logo.png (with the scale set to 0.2) and labelled this frame "logo". The image so far, with both bitmaps, is as shown in Figure 10.26.

Now I just need another dynamic frame for the rest of the side panel information. Again this is done by creating a rectangle, selecting it and using the TeX/LaTeX→Flow Frames→Set Frame menu item to open the frame dialog window. Here I've set the Type to Dynamic, the label to "bookdata" and set Border to None. I've changed the Alignment option to Middle which will vertically balance the frame's contents. The contents can again be set by clicking on the Edit button, which will open the mini-LaTeX editor. The contents are as follows:

Input

```
\begin{tabular}{@{}ll}
Genre:    & Children's Illustrated \\
          & Fiction\\
RRP:      & £5.99\\
```

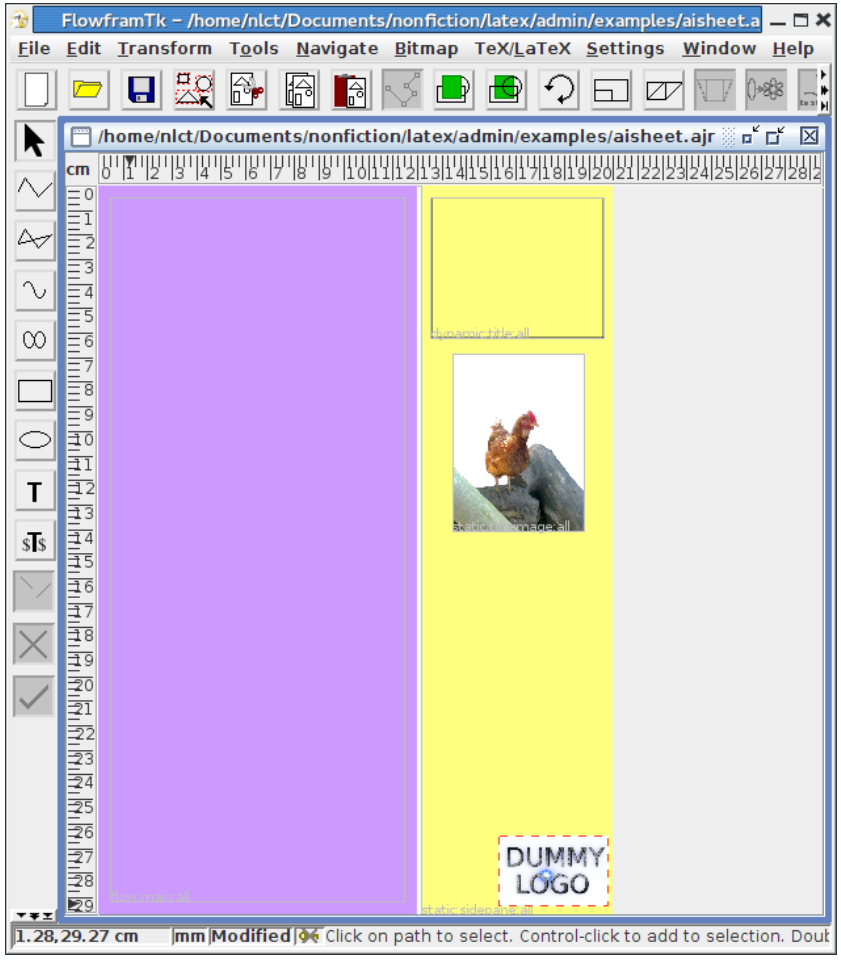


Figure 10.26 FlowframTk — Logo Added

```
Format:    & Paperback\\
Pages:     & 30\\
Pub Date: & 1st August 2014\\
ISBN:      & 978-x-xxxxxx-xx-x
\end{tabular}

\bigskip

\begin{center}
\colorbox{white}{\includegraphics{barcode-qr}}
\end{center}
```

↓ Input

(If you prefer, you could convert barcode-qr.pdf to a bitmap and insert it in a similar manner to the other bitmaps.) Click the green tick to save and return to the Set Frame dialog, shown in Figure 10.27.

The final image is as shown in Figure 10.28. Make sure you save it to an .ajr or .jdr before proceeding to the export function. The flowframtk application can only load its own native files. It can't load the files it exports, so if you need to make any modifications you'll need the .jdr/.ajr file.

Now that the image is complete with all the required flowfram data, you can export to a LaTeX class or package file using the File→Export menu item. I'm going to export to a class file, so I've set the file type filter to Class (*.cls)

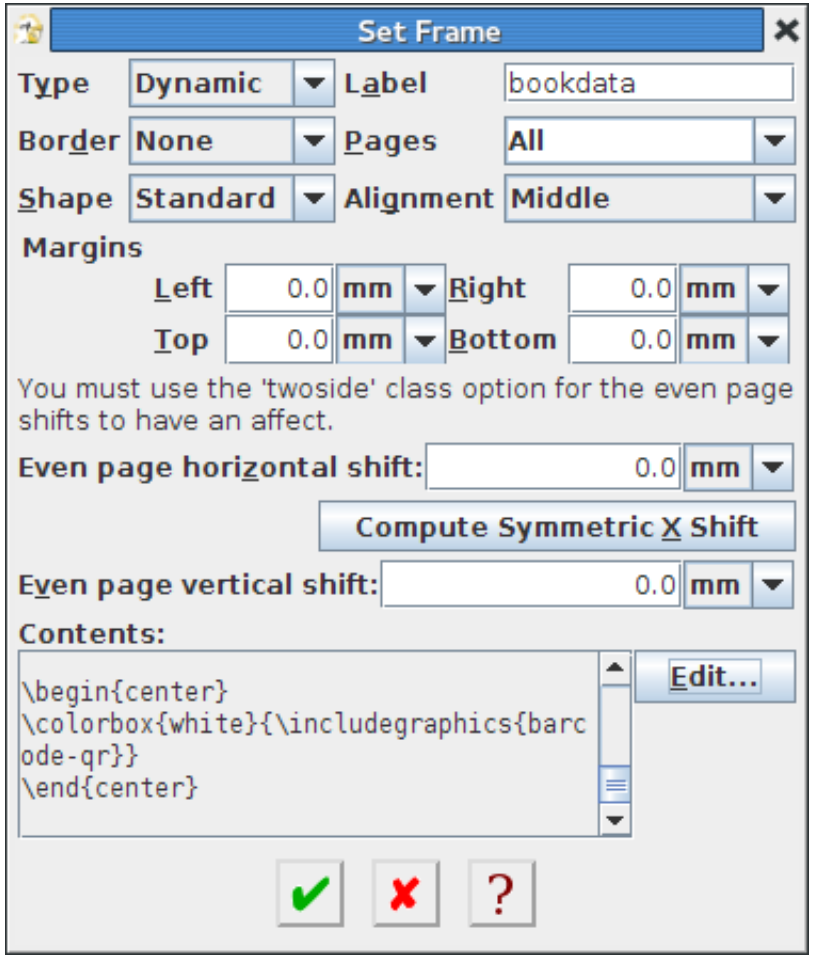


Figure 10.27 FlowframTk — Setting the bookdata Frame

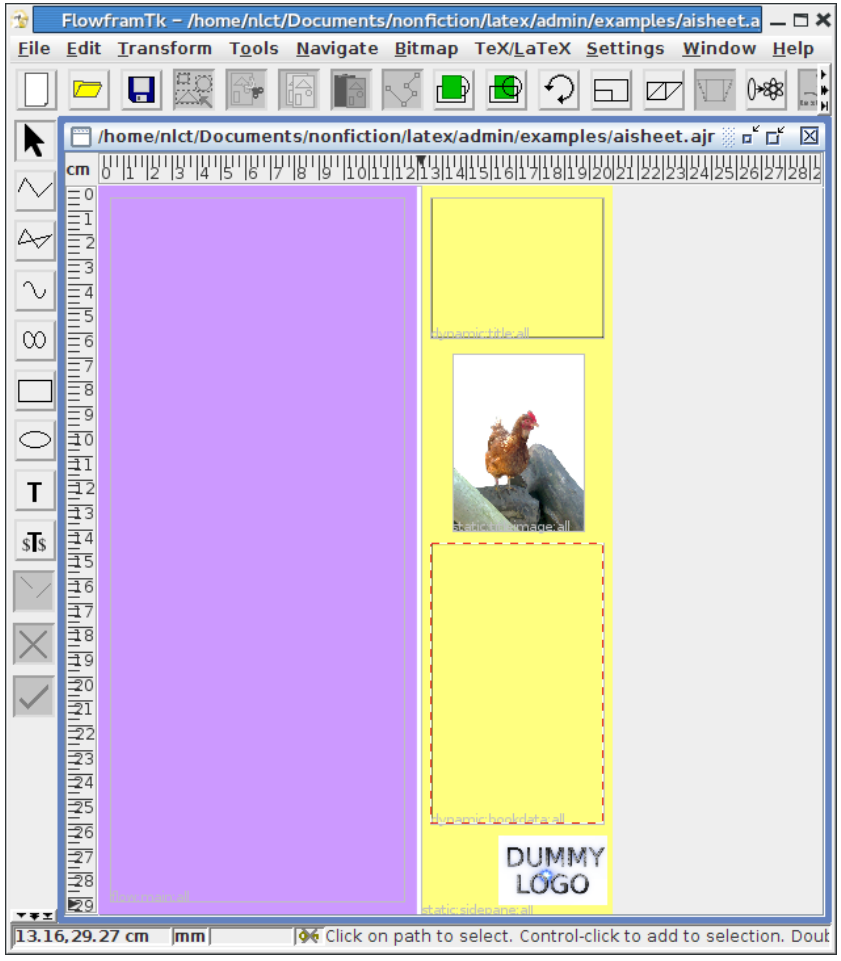


Figure 10.28 FlowframTk — Data Completed

as shown in Figure 10.29.

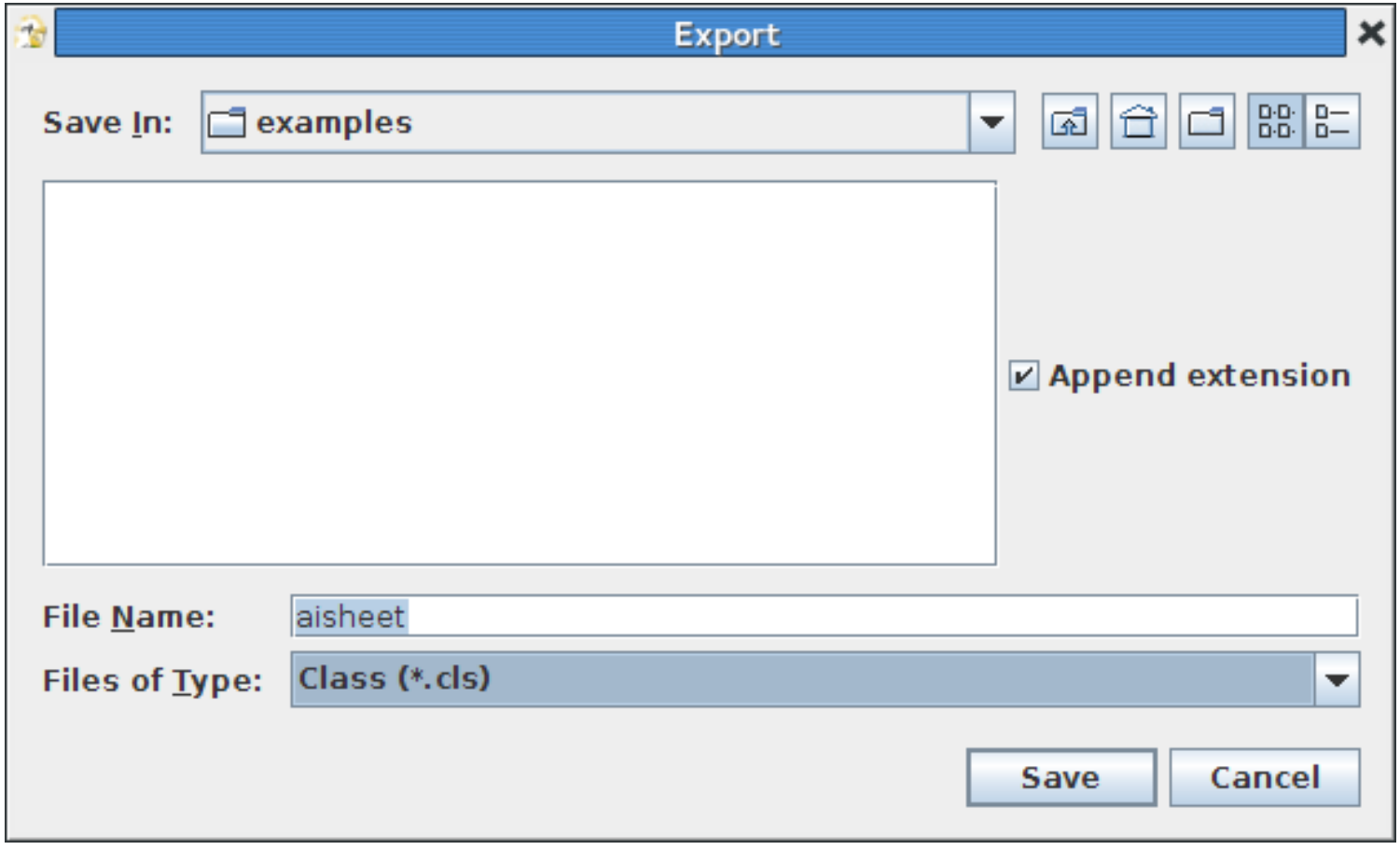


Figure 10.29 FlowframTk — Export to a Class File

My `examples` directory now contains the files: `aisheet.ajr`, `aisheet.cls` along with my original image files `chicken.png`, `barcode-qr.pdf` and `dummy-logo.png`. Now I just need to add a LaTeX document that uses this new aisheet class file:

↑ Input

```
\documentclass{aisheet}
\begin{document}\raggedright

\section{About the Book}

A fun illustrated children's story about some escaped chickens.
Fred and Mabel are looking after Granny's chickens for the day
but, oh no, they've escaped. Will Fred and Mabel find them all
before the chickens get into the road or get eaten by the hungry fox?

\section{About the Author}

Dickie Duck lives somewhere or other and won the best fowl book
award in 2014. He likes writing silly stories about ducks and
chickens.
```

```
\section{Keypoints}

\begin{dinglist}{118}
 \item A fun way of teaching children to count.

 \item Children will enjoy the repetition and rhyme.

 \item Features chickens doing stupid things.

 \item Completely fictitious book encourages children's
imaginations.
\end{dinglist}

\section{Marketing}

\begin{dinglist}{118}
 \item Written by award-winning author.

 \item Illustrated by world famous artist.

 \item Some other really interesting marketing information.
\end{dinglist}
```

```
\section{Contact}

\begin{tabular}{@{}l}
Dickie Duck\\
1 The Street\\
Another Village\\
Some City\\
Imagineshire\\
YZ1 2AB
\end{tabular}
\end{document}
```

Input

The resulting document is shown in Figure 10.30. You can download or view this example document.

About the Book

A fun illustrated children's story about some escaped chickens. Fred and Mabel are looking after Granny's chickens for the day but, oh no, they've escaped. Will Fred and Mabel find them all before the chickens get into the road or get eaten by the hungry fox?

About the Author

Dickie Duck lives somewhere or other and won the best fowl book award in 2014. He likes writing silly stories about ducks and chickens.

Keypoints

- A fun way of teaching children to count.
- Children will enjoy the repetition and rhyme.
- Features chickens doing stupid things.
- Completely fictitious book encourages children's imaginations.

Marketing

- Written by award-winning author.
- Illustrated by world famous artist.
- Some other really interesting marketing information.

Contact

Dickie Duck
1 The Street
Another Village
Some City
Imagineshire
YZ1 2AB

Oh No! The Chickens Have Escaped

Genre: Children's Illustrated Fiction
RRP: £5.99
Format: Paperback
Pages: 30
Pub Date: 1st August 2014
ISBN: 978-x-xxxxxx-xx-x

DUMMY LOGO

Figure 10.30 Advance Information Sheet (via flowframtk)

11. Forms

There aren't very many entries on CTAN that deal with forms. At the time of writing there are four entries listed in the form-fillin topic and only one of them, formular [114], is in both MiKTEX and TEX Live. If you are interested in writing proposals, there is also a proposal topic but again there aren't many entries in it. Some of the exam/assignment classes or packages (see §9 Assignments and Examinations) that have multiple choice or fill-in-the-blank options could also be used to create forms.

As with leaflets and flyers, forms don't really conform to standard typesetting styles. Small forms, such as the one for contact details in Example 51, can be created using tabular-like environments with `\hrulefill` or `\dotfill` for ruled or dotted line areas. There are font packages available that provide tick and cross symbols [64], such as pifont [83] and wasysym [41]. The decoration topic includes packages, such as framed [3] or mdframed [20], that can place frames around regions of text.

The code for the query form from Example 51 is reproduced in the document below. This just uses the standard article class instead of the leaflet class used in that example:

↑ Input

```
\documentclass{article}

\usepackage{wasysym}

\begin{document}
\section{Query Form}

If you'd like to know more about the exciting collaboration between
the Secret Lab of Experimental Stuff and the Department of Stripy
Confectioners please fill in your details below and post this slip to:

\bigskip

\begin{tabular}{@{}l}
Miss Ingperson\\
Secret Lab of Experimental Stuff\\
University of Somewhere\\
Some City\\
AB3 4YZ
\end{tabular}
```

```
\bigskip

\Square\␣I would like to receive quarterly newsletters.

\Square\␣I agree to having my memory wiped.

\Square\␣Yes, I'd really like to feed the ducks.

\bigskip

\begin{tabular}{@{}lp{4cm}}
Name: & \dotfill \\
Address: & \dotfill\\
 & \dotfill \\
 & \dotfill \\
 & \dotfill \\
Postcode: & \dotfill\\
Country: & \dotfill\\
Telephone: & \dotfill\\
Mobile: & \dotfill\\
Email: & \dotfill
```

```
\end{tabular}

\end{document}
```

Input

Both \dotfill and \hrulefill are *leaders* [45] and fill the available horizontal space. In the above example, I used the column identifier p{4cm} to create a column of width 4 cm, which gives \dotfill 4 cm of horizontal space to fill. If I'd just used the l left alignment column identifier then no dotted line would have appeared.

Example

Input

```
Some text\hrulefill Some more text.

Some text\dotfill Some more text.

\hrulefill Some text\dotfill Some more text.\hrulefill
```

Input

produces:

Output

Some text_____Some more text.
Some text . . . . . Some more text.
_____Some text . . . . . Some more text._____

Output

You can place a leader inside a fixed-width box. For example:

```
The \makebox[3em]{\hrulefill} sat on the
\makebox[3em]{\hrulefill}.
```

Input

which produces:

The _____ sat on the _____.

Output

A cut line can be produced with a combination of leaders and a pair of scissors symbol from a package such as pifont. For example:

Input

```
\par\noindent
\makebox[2em]{\dotfill}\ding{33}\dotfill\par
```

Input

which produces:

....✂..... Output

Note that this doesn't extend the cut line into the margins. To achieve that you need to use some negative length and a horizontal box with width given by \paperwidth. For example:

↑ Input

```
\par\noindent
\hspace*{-\dimexpr 1in+\hoffset+\oddsidemargin}%
\rlap{%
 \makebox[\paperwidth][l]{%
   \makebox[4em]{\dotfill}\ding{33}\dotfill
}}\par
```

↓ Input

For a two-sided document you will need to check if the current page is odd or even. For example:

↑ Input

```
\par\noindent
\ifodd\value{page}\relax
```

```
 \hspace*{-\dimexpr 1in+\hoffset+\oddsidemargin}%
\else
 \hspace*{-\dimexpr 1in+\hoffset+\evensidemargin}%
\fi
\rlap{%
 \makebox[\paperwidth][l]{%
   \makebox[4em]{\dotfill}\ding{33}\dotfill
}}\par
```

↓ Input

⚠ Be careful using this method of testing for an odd or even page, as it may not always work due to TEX's asynchronous output routine. If you are using one of the KOMA-Script classes, you can use KOMA-Script's \ifthispageodd command to determine if the current page is odd or even in a more robust manner.

[FAQ: Finding if you're on an odd or an even page]

You may find it easier to define a command that produces this with an optional argument to determine the distance between the start of the line and the scissor symbol. For example:

↑ Input

```
\newcommand{\cutline}[1][4em]{%
  \par\noindent
```

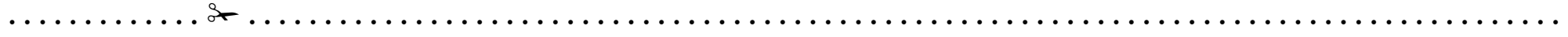
```
  \ifodd\value{page}\relax
   \hspace*{-\dimexpr 1in+\hoffset+\oddsidemargin}%
  \else
   \hspace*{-\dimexpr 1in+\hoffset+\evensidemargin}%
  \fi
  \rlap{%
   \makebox[\paperwidth][l]{%
     \makebox[#1]{\dotfill}\ding{33}\dotfill
  }}\par
}
```

Input

Now you can just use this command, for example:

```
\cutline[6em]
```

Input



Remember that if you want a hard copy this requires borderless printing otherwise a slim margin may still appear (as occurs in the paperback version of this book).

If you are using the flowfram package, you also need to take into account any additional offset caused by a frame that doesn't have its left edge flush against the left edge of the typeblock. In this case the definition of `\cutline` needs to be adjusted as follows:

Input

```
\newlength\frameoffset

\newcommand{\cutline}[1][4em]{%
  \par\noindent
  \ifodd\value{page}\relax
    \computeleftedgeodd{\frameoffset}%
    \getflowbounds{\value{thisframe}}%
  \else
    \computeleftedgeeven{\frameoffset}%
    \getflowevenbounds{\value{thisframe}}%
  \fi
  \addtolength{\frameoffset}{-\ffareax}%
  \hspace*{\frameoffset}%
  \rlap{%
   \makebox[\paperwidth][l]{%
     \makebox[#1]{\dotfill}\ding{33}\dotfill
  }}\par
}
```

Input

Exercise 29. Query Form
Reproduce the earlier query form from Example 51 as a single-paged document (for example, using article or scrartcl) with no page numbering and a cut line between the submission address and the actual form.

You can download or view a solution to this exercise.

11.1 Writing a Class File for a Form

The above may be suitable for a short form to be filled in by hand, but it may be that you want to produce a more complex form to be filled in by LaTeX users. In this case, it may be more appropriate to write a class file that provides commands to fill in the form data. This section describes how to do this and is developed from an article I wrote on the LaTeX Community Forum [91]. The next section will look at interactive form elements.

§7.3 briefly introduced package writing. There are similar commands for classes, and some of the package commands, such as `\RequirePackage`, may also be used in class files. As with packages, the class first identifies the TeX format using

`\NeedsTeXFormat{`⟨*format*⟩`}[`⟨*version*⟩`]` Definition

and then identifies the class using

`\ProvidesClass{`⟨*name*⟩`}[`⟨*version*⟩`]` Definition

This has the same syntax as `\ProvidesPackage` described in §7.3. The class code should be saved in a file called ⟨*name*⟩`.cls` and placed somewhere on TEX's path.

Many classes load a parent class, which saves defining many common elements, such as the sectioning commands or list environments. The parent class is loaded using:

`\LoadClass[`⟨*options*⟩`]{`⟨*name*⟩`}[`⟨*version*⟩`]` Definition

where ⟨*name*⟩ is the name of the parent class. The optional arguments are analogous to the optional arguments of `\RequirePackage`. Before you load a class, you can specify which options to pass to it using:

`\PassOptionsToClass{`⟨*option-list*⟩`}{`⟨*class-name*⟩`}` Definition

where ⟨*option-list*⟩ is a comma-separated list of options to pass to the class specified by ⟨*class-name*⟩. An option is defined using:

\DeclareOption{⟨option⟩}{⟨code⟩} Definition

where ⟨option⟩ is the option name and ⟨code⟩ is the code to perform for that option. The starred version of this command only has one argument:

\DeclareOption*{⟨code⟩} Definition

This indicates the code to perform for an unknown option. The option name can be referenced within ⟨code⟩ using

\CurrentOption Definition

Once all the options have been declared, they then need to be processed using:

\ProcessOptions Definition

Here's the code for a trivial class called simple-form:

↑ Input

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{simple-form}[2014/10/11]

\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}
```

```
\ProcessOptions

\LoadClass{article}

% class code

\endinput
```

↓ Input

This code needs to be saved in a file called `simple-form.cls`. At the moment this class doesn't provide anything in addition to the article class, but the new code will be added in the area between the `\LoadClass` line and the `\endinput` line.

In addition to `\Square`, which produces an empty square □, the wasysym package also defines

`\XBox` Definition

which produces a box with a cross in it ☒, and

`\CheckedBox` Definition

which produces a box with a tick in it ☑. These symbols will be useful for this new class, so the class code needs to load the wasysym package using

```
\RequirePackage{wasysym}
```
Input

Information for the form can be gathered using the same type of mechanism as \author, \title and \date. These work by having an internal command that stores the information and a user command that sets the internal command. For example, if the form requires a person's name, the internal command could be called, say, \@name which is initially empty

```
\newcommand*{\@name}{}
```
Input

and the user command could be called, say, \name which redefines the internal command:

```
\newcommand*{\name}[1]{%
  \renewcommand*{\@name}{#1}%
}
```
↑ Input

↓ Input

Then a command analogous to \maketitle is required to typeset the form. For example, this command could be called \makeform and it would use the internal commands to fill in the required areas. A trivial example would be:

```
\newcommand{\makeform}{%
  Name: \@name\␣Date: \@date
}
```

Input

Input

(\@date is the internal command used by \date and is initially defined as \today.)

This definition of \makeform has a problem when \name isn't used. If \@name is empty it won't take up any space. A better solution is to put \@name inside a horizontal box with a fixed width:

```
\newcommand{\makeform}{%
  Name: \makebox[6em][l]{\@name}\␣Date: \@date
}
```

Input

Input

This will leave a blank space if the name hasn't been set. If you prefer a lined space you could make the initial definition of \@name use \hrulefill

```
\newcommand*{\@name}{\hrulefill}
```
Input

Now a lined space will appear if \name hasn't been used, but the line won't be present if \name has been used. If you still want a line to appear even if \name has been used, then you could replace

```
\makebox[6em][l]{\@name}
```
Input

with

```
\makebox[6em][l]{\rlap{\@name}\hrulefill}
```
Input

If you have more than one blank area to fill in, then it's best to define a command to do this. For example:

↑ Input

```
\newcommand*{\form@fillin}[2]{%
  \makebox[#1][l]{\rlap{#2}\hrulefill}%
}
```

↓ Input

This has the syntax

\form@fillin{⟨*width*⟩}{⟨*text*⟩} Definition

so the trivial definition of \makeform can now look something like:

```
\newcommand{\makeform}{%
  Name: \form@fillin{6em}{\@name}\␣
  Date: \form@fillin{4em}{\@date}%
}
```

↑ Input

↓ Input

Variations of \form@fillin could include

```
\newcommand*{\form@fillin}[2]{%
  \makebox[#1][l]{\rlap{#2}\dotfill}%
}
```

↑ Input

↓ Input

which uses a dotted line instead or

```
\newcommand*{\form@fillin}[2]{%
  \makebox[#1][c]{%
  \hrulefill\makebox[0pt][c]{#2}\hrulefill}%
}
```

Input

Input

which centres the text within the ruled area or

```
\newcommand*{\form@fillin}[2]{%
  \makebox[#1][r]{\hrulefill\llap{#2}}%
}
```

Input

Input

which right-aligns the text within the ruled area.

Check boxes require a different interface, but there are various methods you can use. For example, for a gender check box you might want a command called, say, \male that ticks the "male" box and a command called, say, \female that ticks the "female" box. Alternatively you might prefer a command called, say, \gender that takes an argument which can

either be `male` or `female`. In both cases, internal commands are defined for each option that default to the unchecked case:

Input

```
\newcommand*{\gender@male}{\Square}
\newcommand*{\gender@female}{\Square}
```

Input

The user commands redefine these internal commands. In the first case:

Input

```
\newcommand*{\male}{%
  \renewcommand*{\gender@male}{\XBox}%
}
\newcommand*{\female}{%
  \renewcommand*{\gender@female}{\XBox}%
}
```

Input

In the second case:

Input

```
\newcommand*{\gender}[1]{%
  \ifcsdef{gender@#1}%
  {\csdef{gender@#1}{\XBox}}
  {% unknown option produces an error
    \ClassError{simple-form}{Unknown gender `#1'}
      {Options: `male', `female'}%
  }%
}
```

Input

This uses the etoolbox commands \ifcsdef and \csdef described in §2.1.1, and also uses

\ClassError{⟨*class-name*⟩}{⟨*error-message*⟩}{⟨*help-message*⟩} Definition

to display an error message. The first argument is the class name (simple-form in this case) and the second argument is the error message. The third argument provides a help message if the user types "h" in TEX's interactive mode.

What if I later decide to use \CheckedBox instead of \XBox? Alternatively, I might decide to use a radio button style. To help with code maintenance

it's better to define commands for the checked and unchecked status and use those commands for the form data. For example:

Input

```
\newcommand*{\form@unchecked}{\Square}
\newcommand*{\form@checked}{\XBox}

\newcommand*{\gender@male}{\form@unchecked}
\newcommand*{\gender@female}{\form@unchecked}

\newcommand*{\male}{%
  \renewcommand*{\gender@male}{\form@checked}%
}
\newcommand*{\female}{%
  \renewcommand*{\gender@female}{\form@checked}%
}
```

Input

Or

Input

```
\newcommand*{\gender}[1]{%
```

```
  \ifcsdef{gender@#1}%
  {\csdef{gender@#1}{\form@checked}}
  {%
    \ClassError{simple-form}{Unknown gender `#1'}
      {Options: `male', `female'}%
  }%
}
```

Input

Now there are only one or two lines to change if I want to use different symbols. For example, to use \CheckedBox instead of \XBox just requires one edit:

```
\newcommand*{\form@checked}{\CheckedBox}
```

Input

If you can't find a symbol that suits you, it's possible to combine symbols using a command such as \rlap. For example, to make round radio style buttons, you could use the ifsym package [44] with the geometry option and combine \BigCircle with \FilledSmallCircle.

Input

```
\newcommand*{\form@unchecked}{\BigCircle}
\newcommand*{\form@checked}{\rlap{\FilledSmallCircle}\BigCircle}
```

Input

These produce the symbols ○ and ◉.

⚠ Take care if you want to load both ifsym and wasysym as they have conflicting command names when ifsym is loaded with the geometry option. For example, both define \Square. If you want both packages, load ifsym without the geometry option and use \textifsymbol to access the symbols. For example:

Input

```
\newcommand*{\form@checked}{%
  \rlap{\textifsymbol[ifgeo]{117}}\textifsymbol[ifgeo]{37}}
\newcommand*{\form@unchecked}{\textifsymbol[ifgeo]{37}}
```

Input

Alternatively, if you want fancier buttons you can use picture drawing code. The following example creates on and off buttons using tikz with the shadings and shadows libraries:

Input

```
\RequirePackage[x11names]{xcolor}
\RequirePackage{tikz}
\usetikzlibrary{shadings}
\usetikzlibrary{shadows}
```

```
\newcommand*{\form@unchecked}{%
  \resizebox{!}{2ex}%
  {%
    \begin{tikzpicture}
    \path[fill=LightYellow4,circular glow] (0,0) circle(.5cm);
    \path[fill=LightYellow1,circular glow={fill=LightYellow3}]
      (0,0) circle(.35cm);
    \end{tikzpicture}%
  }%
}
\newcommand*{\form@checked}{%
  \resizebox{!}{2ex}%
  {%
    \begin{tikzpicture}
    \path[shade,inner color=LightYellow2,
          outer color=LightYellow4,
          circular glow] (0,0) circle(.5cm);
    \end{tikzpicture}%
  }%
}
```

↓ Input

This produces ○ and ◎.

Similarly, it's a good idea to provide a command to layout the check box or fill-in area and its accompanying text. For example:

```
\newcommand*{\form@layout@checkbox}[2]{#1 #2}
```
Input

This has the syntax:

\form@layout@checkbox{⟨*symbol*⟩}{⟨*text*⟩}

Definition

For example:

↑ Input

```
\form@layout@checkbox{\gender@male}{Male}
\form@layout@checkbox{\gender@female}{Female}
```

↓ Input

This means that if, say, you want to change all your check boxes so that the text is to the left of the check box symbol, then all you need to do is change the definition of \form@layout@checkbox. Similarly for the fill-in text fields:

```
\newcommand*{\form@layout@fillin}[3]{#3: \form@fillin{#1}{#2}}
```
Input

This has the syntax

\form@layout@fillin{⟨*width*⟩}{⟨*value*⟩}{⟨*text*⟩} Definition

For example:

```
\form@layout@fillin{6em}{\@name}{Name}
```
Input

Example 56. A Simple Form Class

Here's a simple form class with two fill-in areas (for the name and date) and two check boxes (for the gender). The article class is loaded with the options a4paper and 12pt as this example is simulating a form with specific paper size and font requirements.

The contents of the file simple-form.cls are as follows:

↑ Input

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{simple-form}[2014/10/11]

\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}

\ProcessOptions

\LoadClass[a4paper,12pt]{article}
```

```
\RequirePackage{etoolbox}
\RequirePackage{wasysym}

\newcommand*{\form@fillin}[2]{%
  \makebox[#1][l]{\rlap{#2}\hrulefill}%
}

\newcommand*{\form@checked}{\XBox}
\newcommand*{\form@unchecked}{\Square}

\newcommand*{\form@layout@checkbox}[2]{#1 #2}
\newcommand*{\form@layout@fillin}[3]{#3: \form@fillin{#1}{#2}}

\newcommand*\@name{}
\newcommand*{\name}[1]{\renewcommand*{\@name}{#1}}

\newcommand*{\gender@male}{\form@unchecked}
\newcommand*{\gender@female}{\form@unchecked}

\newcommand*{\gender}[1]{%
  \ifcsdef{gender@#1}%
```

```
  {\csdef{gender@#1}{\form@checked}}
  {%
    \ClassError{simple-form}{Unknown gender `#1'}%
    {Options: `male', `female'}%
  }%
}

\newcommand{\makeform}{%
  \form@layout@fillin{8em}{\@name}{Name}\qquad
  \form@layout@fillin{12em}{\@date}{Date}
  \par
  \bigskip
  \par
  \form@layout@checkbox{\gender@male}{Male}\qquad
  \form@layout@checkbox\gender@female{Female}
}

\endinput
```

↓ Input

An example document:

↑ Input

```
\documentclass{simple-form}

\name{Mabel Canary}
\gender{female}

\begin{document}

\makeform

\end{document}
```

↓ Input

The result is shown in Figure 11.1. You can download or view this example.

Output

Name: Mabel Canary Date: October 12, 2014

☐ Male ☒ Female

Figure 11.1 A Simple Form with Two Fill-In Areas and Two Check Boxes

The above example doesn't test if \gender has already been used, so it's possible for a user to do:

```
\gender{male}\gender{female}
```
Input

which would cause both boxes to be checked. If you want to prevent this from happening you could either produce an error message if the command is used more than once or make each subsequent use of the command reset the boxes before setting the new choice.

Here's a possible way of implementing the first case. It uses the \let assignment described in §2.1.1.

↑ Input

```
\newcommand*{\@gendererror}[1]{%
  \ClassError{simple-form}
  {\string\gender\space may only be used once}
  {}%
}

\newcommand*{\gender}[1]{%
  \let\gender\@gendererror
  \ifcsdef{gender@#1}%
  {\csdef{gender@#1}{\form@checked}}
```

```
  {%
    \ClassError{simple-form}{Unknown gender `#1'}%
    {Options: `male', `female'}%
  }%
}
```

↓ Input

This works as follows: the first time \gender is used, it redefines itself to have the same definition as \@gendererror, so the next time \gender is used, it's now equivalent to \@gendererror, which ignores its argument and produces an error message. (\string is a TEX primitive that converts the following control sequence into a list of characters, which provides an easy way of printing the control sequence in the transcript file or console.)

Here's a possible way of implementing the second case that defines a reset command:

↑ Input

```
\newcommand*{\@resetgender}{%
 \renewcommand*{\gender@male}{\form@unchecked}%
 \renewcommand*{\gender@female}{\form@unchecked}%
}
```

```
\newcommand*{\gender}[1]{%
  \@resetgender
  \ifcsdef{gender@#1}%
  {\csdef{gender@#1}{\form@checked}}
  {%
    \ClassError{simple-form}{Unknown gender `#1'}%
    {Options: `male', `female'}%
  }%
}
```

Input

The \male/\female version is simpler:

Input

```
\newcommand*{\male}{%
  \renewcommand*{\gender@female}{\form@unchecked}
  \renewcommand*{\gender@male}{\form@checked}
}
\newcommand*{\female}{%
  \renewcommand*{\gender@male}{\form@unchecked}
  \renewcommand*{\gender@female}{\form@checked}
}
```

Input

However the other method is neater for a large set of check boxes. If you do have many choices, you may find it easier to use a list-based approach. For example, suppose I want to produce the following:

Which project would you like to enrol on?

☐	Mind-Controlling Cookies	☐	Telepathic Cakes
☐	Exploding Chocolates	☐	Ray Gun

A convenient user command might be called, say, `\project` where the argument may be one of: `cookies`, `cakes`, `chocolates` or `raygun`. The internal commands are called `\project@`⟨*label*⟩ where ⟨*label*⟩ is the argument of `\project`. These commands can be reset using:

```
\csdef{project@⟨label⟩}{\form@unchecked}
```
Input

and set using

```
\csdef{project@⟨label⟩}{\form@checked}
```
Input

These can be wrapped up in two commands that each take the label as the argument:

Input

```
\newcommand*{\reset@project}[1]{%
  \csdef{project@#1}{\form@unchecked}%
}
\newcommand*{\set@project}[1]{%
  \ifcsdef{project@#1}
  {\csdef{project@#1}{\form@checked}}
  {%
    \ClassError{simple-form}{Unknown project `#1'}{}%
  }%
}
```

Input

It's also useful to provide a command to use the internal \project@⟨*label*⟩ command:

Input

```
\newcommand*{\use@project}[1]{%
  \ifcsdef{project@#1}{\csuse{project@#1}}{\form@unchecked}%
}
```

Input

This will produce an unchecked box if the label hasn't been defined, which means that the internal commands don't need to be initialised if the user wants a blank form to fill in by hand.

Here's a comma-separated list where each element contains two groups. The first is the label that will be used in the argument of `\project` and the second is the text to appear next to the check box in the form:

Input

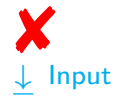
```
\newcommand*{\@projectlist}{%
  {cookies}{Mind-Controlling Cookies},%
  {cakes}{Telepathic Cakes},%
  {chocolates}{Exploding Chocolates},%
  {raygun}{Ray Gun}%
}
```

Input

Various list-iteration commands were discussed in §2.7.2, but in the examples from that section all of the lists had an element that could be used as a single argument to a command such as `\do`. However in this list each element needs to be treated as two arguments. For example, the command to reset the check boxes should iterate through this list but only grab the first group (the label) of each element.

Consider first:

```
\@for\this@element:=\@projectlist\do{%
  \reset@project\this@element
}
```



This won't work because it's equivalent to doing

```
\reset@project{{cookies}{Mind-Controlling Cookies}}
```


and so on. I could try using \expandafter described in §2.7.2:

```
\@for\this@element:=\@projectlist\do{%
  \expandafter\reset@project\this@element
}
```

This is an improvement as this is now equivalent to doing

```
\reset@project{cookies}{Mind-Controlling Cookies}
```

and so on. Now \reset@project picks up the label correctly, but the text after the label is left dangling and needs to be discarded. There are various ways to deal with this. The simplest solution is just to make \reset@project take two arguments and ignore the second argument:

Input

```
\newcommand*{\reset@project}[2]{%
  \csdef{project@#1}{\form@unchecked}%
}
```

Input

A more generic approach is to leave \reset@project with just one argument as before and use the LaTeX kernel command

\@firstoftwo{⟨*first*⟩}{⟨*second*⟩}

Definition

which does ⟨*first*⟩ and discards ⟨*second*⟩. This requires \expandafter to expand \this@element before applying \@firstoftwo:

```
\@for\this@element:=\@projectlist\do{%
  \reset@project{\expandafter\@firstoftwo\this@element}%
}
```

Input

Input

A similar method can be used to display the check boxes and their associated text within the form. There is an analogous LaTeX kernel command that grabs the second argument and discards the first:

\@secondoftwo{⟨*first*⟩}{⟨*second*⟩}

Definition

Here's a simple example that just displays the check boxes with their associated text without any tabulation:

```
\@for\this@element:=\@projectlist\do{%
  \use@project{\expandafter\@firstoftwo\this@element}% check box
  \space
  \expandafter\@secondoftwo\this@element
  \qquad
}
```

Input

Input

Or using the layout command \form@layout@checkbox defined earlier:

Input

```
\@for\this@element:=\@projectlist\do{%
  \form@layout@checkbox
    {\use@project{\expandafter\@firstoftwo\this@element}}% check box
    {\expandafter\@secondoftwo\this@element}% text
  \qquad
}
```

Input

This can be converted into a tabular environment but we need a way to track which column we're in. One way to do this is to define a register (recall §2.1.3).

Input

```
% initialise
\newcount\form@columncount
\form@columncount=1\relax
\def\form@precolumn{}%
% layout check boxes and text:
\begin{tabular}{ll}
```

```
\@for\this@element:=\@projectlist\do{%
  \global\let\this@element\this@element
  \form@precolumn
  \form@layout@checkbox
    {\use@project{\expandafter\@firstoftwo\this@element}}%
    {\expandafter\@secondoftwo\this@element}%
  \global\advance\form@columncount by 1\relax
  \ifnum\form@columncount>2\relax
    \global\form@columncount=1\relax
    \gdef\form@precolumn{\\}%
  \else
    \gdef\form@precolumn{&}%
  \fi
}%
\end{tabular}%
```

Input

(\global is required because of the local scoping effect of tabular cells.) This uses a similar method to those discussed in §2.7.5.

If you are likely to have more than one group of check boxes, then it makes more sense to create generic commands. First, we need generic versions of the above \reset@project, \set@project and \use@project where the first argument is the element label (such as cakes) and the

second argument is the block label (such as `project`):

Input

```
\newcommand*{\reset@element}[2]{%
  \csdef{#2@#1}{\form@unchecked}%
}

\newcommand*{\set@element}[2]{%
  \ifcsdef{#2@#1}%
  {\csdef{#2@#1}{\form@checked}}%
  {%
    \ClassError{simple-form}{Unknown #2 `#1'}{}%
  }%
}

\newcommand*{\use@element}[2]{%
  \ifcsdef{#2@#1}{\csuse{#2@#1}}{\form@unchecked}%
}
```

Input

So now instead of

`\reset@project{`⟨*label*⟩`}` Input

I need to use

`\reset@element{`⟨*label*⟩`}{project}` Input

and so on. It's also convenient to provide a command that can iterate over the `{`⟨*label*⟩`}{`⟨*text*⟩`}` list (such as `\@projectlist`) for the block:

↑ Input

```
\newcommand*{\for@block}[3]{%
  \ifcsdef{@#2list}%
  {%
    \expandafter\@for\expandafter
      #1\expandafter:\expandafter=\csname @#2list\endcsname\do{#3}%
  }%
  {%
    \ClassError{simple-form}{Unknown block `#2'}{}%
  }%
}
```

↓ Input

(The \expandafters are required because the list control sequence provided by \csname @#2list\endcsname needs to be expanded to the actual control sequence \@⟨*block-label*⟩list, for example \@projectlist, before \@for tries to iterate over it.) This has the syntax:

\for@block{⟨*cs*⟩}{⟨*block-label*⟩}{⟨*body*⟩} Definition

where ⟨*cs*⟩ is assigned to the {⟨*label*⟩}{⟨*text*⟩} element for the current iteration.

All elements within a block can be reset using \reset@block, which is defined as:

Input

```
\newcommand*{\reset@block}[1]{%
  \for@block{\this@element}{#1}%
  {%
    \reset@element{\expandafter\@firstoftwo\this@element}{#1}%
  }%
}
```

Input

This means that \project can now be defined as

Input

```
\newcommand*{\project}[1]{%
  \reset@block{project}%
  \set@element{#1}{project}%
}
```

Input

The generic two-column tabulated block of elements used by \makeform can be defined as follows:

Input

```
\newcount\form@columncount

\newcommand*{\form@block}[1]{%
  \def\form@precolumn{}%
  \form@columncount=1\relax
  \begin{tabular}{ll}
  \for@block{\this@element}{#1}%
  {%
    \global\let\this@element\this@element
    \form@precolumn
```

```
    \form@layout@checkbox
      {\use@element{\expandafter\@firstoftwo\this@element}{#1}}%
      {\expandafter\@secondoftwo\this@element}%
    \global\advance\form@columncount by 1\relax
    \ifnum\form@columncount>2\relax
      \global\form@columncount=1\relax
      \gdef\form@precolumn{\\}%
    \else
      \gdef\form@precolumn{&}%
    \fi
  }%
  \end{tabular}%
}
```

Input

This custom command has the syntax:

`\form@block{`⟨*block-label*⟩`}` Definition

So for the project example, this would just require

```
\form@block{project}
```

Input

This is hard-coded for two columns, but it would be more flexible to allow an arbitrary number of columns. For example if the command had the

syntax

\form@block{⟨*block-label*⟩}{⟨*columns*⟩} Definition

then the project block could be generated using

```
\form@block{project}{2}
```
Input

In this case, the hard-coded conditional in \form@block

```
\ifnum\form@columncount>2\relax
```
Input

can now have the total column count replaced with #2:

```
\ifnum\form@columncount>#2\relax
```
Input

However the column specifier argument for the tabular environment is a little more complicated as it now requires #2 lots of l (or whatever alignment specifier you want).

Recall TEX's \loop command from §2.7.4 and the hook management commands from §2.1.2. These can be used to generate the argument for the tabular environment:

```
% initialise
```
↑ Input

```
\def\form@columnargs{}%
\form@columncount=0\relax
% iterate #2 times
\loop
 \appto\form@columnargs{l}%
 \advance\form@columncount by 1\relax
\ifnum\form@columncount<#2
\repeat
```

Input

This will store the column specifiers in \form@columnargs which can now be used in the tabular environment argument:

```
\begin{tabular}{\form@columnargs}
```

Input

Therefore the new two-argument version of \form@block can be defined as:

```

```

Input

```
\newcount\form@columncount

\newcommand*{\form@block}[2]{%
  \def\form@columnargs{}%
```

```
\form@columncount=0\relax
\loop
 \appto\form@columnargs{l}%
 \advance\form@columncount by 1\relax
\ifnum\form@columncount<#2
\repeat
\def\form@precolumn{}%
\form@columncount=1\relax
\begin{tabular}{\form@columnargs}
\for@block\this@element{#1}%
{%
  \global\let\this@element\this@element
  \form@precolumn
  \form@layout@checkbox
    {\use@element{\expandafter\@firstoftwo\this@element}{#1}}%
    {\expandafter\@secondoftwo\this@element}%
  \global\advance\form@columncount by 1\relax
  \ifnum\form@columncount>#2\relax
    \global\form@columncount=1\relax
    \gdef\form@precolumn{\\}%
  \else
    \gdef\form@precolumn{&}%
```

```
    \fi
  }%
  \end{tabular}%
}
```

Input

The form check box elements are now much simpler to define:

Input

```
\newcommand*{\@genderlist}{{male}{Male},{female}{Female}}

\newcommand*{\gender}[1]{%
  \reset@block{gender}%
  \set@element{#1}{gender}%
}

\newcommand*{\@projectlist}{%
  {cookies}{Mind-Controlling Cookies},%
  {cakes}{Telepathic Cakes},%
  {chocolates}{Exploding Chocolates},%
  {raygun}{Ray Gun}%
}
```

```
\newcommand*{\project}[1]{%
  \reset@block{project}%
  \set@element{#1}{project}%
}
```

Input

If multiple selections are permitted, then the \reset@block command needs to be moved outside the user command definition to initialise all the elements. For example, if multiple projects may be selected:

Input

```
\reset@block{project}

\newcommand*{\project}[1]{%
  \set@element{#1}{project}%
}
```

Input

Exercise 30. Simple Form Class with Check Boxes
Adapt the class file simple-form.cls from Example 56 so that the form shown in Figure 11.2 can be created with the following document:

↑ Input

```
\documentclass{simple-form}

\name{Mabel Canary}
\date{2014-10-13}

\gender{female}
\project{cakes}
\icecream{vanilla}
\icecream{fudge}
\icecream{other}

\begin{document}

\makeform

\end{document}
```

↓ Input

For the More Adventurous

Add a fill-in area for the "Other" ice-cream option so that instead of the user writing:

```
\icecream{other}
```
Input

they can use a new command:

```
\othericecream{Neapolitan}
```
Input

which both checks the "Other" box and fills in the area, as shown in Figure 11.3. You can download or view a solution to this exercise.

If you have a large text area that needs to be filled in, you may prefer to use an environment to collect the information. For example, instead of creating a command to specify, say, a project description:

```
\projectdescription{⟨Several paragraphs of text⟩}
```
Input

which can be defined using

```
\newcommand{\@projectdescription}{}
\newcommand{\projectdescription}[1]{%
  \renewcommand{\@projectdescription}{#1}%
}
```
Input

Output

Name: Mabel Canary Date: 2014-10-13

☐ Male ☒ Female

Which project would you like to enrol on? (Tick one box.)

☐ Mind-Controlling Cookies ☒ Telepathic Cakes
☐ Exploding Chocolates ☐ Ray Gun

Which ice-cream flavours do you like? (Tick all that apply.)

☒ Vanilla	☐ Mint	☐ Toffee
☒ Fudge	☐ Guaraná	☐ Strawberry
☐ Raspberry Ripple	☐ Chilli	☒ Other

Figure 11.2 A Simple Form with Multiple Check Box Areas

Output

Name: Mabel Canary Date: 2014-10-13

☐ Male ☒ Female

Which project would you like to enrol on? (Tick one box.)

☐ Mind-Controlling Cookies ☒ Telepathic Cakes
☐ Exploding Chocolates ☐ Ray Gun

Which ice-cream flavours do you like? (Tick all that apply.)

☒ Vanilla	☐ Mint	☐ Toffee
☒ Fudge	☐ Guaraná	☐ Strawberry
☐ Raspberry Ripple	☐ Chilli	☒ Other: Neapolitan

Figure 11.3 A Simple Form with Multiple Check Box Areas and a Fill-In Other Area

you may prefer to have the user interface:

Input

```
\begin{ProjectDescription}
⟨Several paragraphs of text⟩
\end{ProjectDescription}
```

Input

This is more complicated to define, as you can't simply gather the contents of an environment when you use \newenvironment. There are a number of ways to achieve this.

If the environment contents are being gathered so that they can then be stored in a command definition, then the limitations applied to command definitions also apply to the environment contents. This includes the usual problems with verbatim code in a command argument.

[FAQ: Why doesn't verbatim work within...?]

The collect package [76] provides the collectinmacro environment:

Definition

```
\begin{collectinmacro}{⟨macro⟩}{⟨before⟩}{⟨after⟩}
⟨body⟩
\end{collectinmacro}
```

This defines the command ⟨*macro*⟩ to be ⟨*before*⟩⟨*body*⟩⟨*after*⟩. For example,

```
\begin{collectinmacro}{\mycommand}{Before. }{ After.}
Some text here.
\end{collectinmacro}
```
Input

This is equivalent to:

```
\newcommand{\mycommand}{Before. Some text here. After.}
```
Input

If you want to define an environment that uses this method, you can't use the environment form \begin{collectinmacro} and \end{collectinmacro}, but must instead use the commands \collectinmacro and \endcollectinmacro.

Example:

```
\newcommand{\@projectdescription}{}
\newenvironment{ProjectDescription}%
{\collectinmacro{@projectdescription}{}{}}%
{\endcollectinmacro}
```
Input

Another possibility is to use the amsmath package's

\collect@body⟨*cs*⟩ Definition

command. This gathers the contents of the current environment ⟨*body*⟩ and applies ⟨*cs*⟩{⟨*body*⟩}.

Example:

↑ Input

```
\newcommand{\@projectdescription}{}
\newcommand{\projectdescription}[1]{%
  \gdef\@projectdescription{#1}%
}
\newenvironment{ProjectDescription}%
{\collect@body\projectdescription}%
{}
```

↓ Input

This now means that the user can do either

↑ Input

```
\begin{ProjectDescription}
```

```
This will be an interesting project.
\end{ProjectDescription}
```

Input

or

```
\projectdescription{This will be an interesting project.}
```

Input

Note that I had to use \gdef instead of \renewcommand otherwise the change will be scoped by the encasing environment.

⚠ The \collect@body command uses short internal commands to gather the environment contents, which means that the environment can't contain paragraph breaks. If you want to allow paragraph breaks, you can use an analogous command provided by the environ package [74]:

\Collect@Body⟨*cs*⟩

Definition

Note that the unstarred version of \newcommand allows a paragraph break to be present within #1 so \projectdescription can be used by \Collect@Body in the following:

Input

```
\newcommand{\@projectdescription}{}
\newcommand{\projectdescription}[1]{%
```

```
  \gdef\@projectdescription{#1}%
}
\newenvironment{ProjectDescription}%
{\Collect@Body\projectdescription}%
{}
```

↓ Input

Example 57. A Simple Form Class (Gathering Environment Contents)

This example uses the \Collect@Body command from the environ package to allow the user to enter multi-paragraph data in a form. First the class file, sample-form.cls:

↑ Input

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{sample-form}[2014/11/03]

\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}

\ProcessOptions

\LoadClass[a4paper,12pt]{article}
```

```
\RequirePackage{environ}

\newcommand*{\form@fillin}[2]{%
  \makebox[#1][l]{\rlap{#2}\hrulefill}%
}

\newcommand*{\form@layout@fillin}[3]{#3: \form@fillin{#1}{#2}}

\newcommand*\@name{}
\newcommand*{\name}[1]{%
  \renewcommand*{\@name}{#1}%
}

\newcommand*\@projectdescription{}
\newcommand{\projectdescription}[1]{%
  \long\gdef\@projectdescription{#1}%
}

\newenvironment{ProjectDescription}%
{\Collect@Body\projectdescription}{}
```

```
\newcommand{\makeform}{%
  \section{Applicant Details}
  \form@layout@fillin{8em}{\@name}{Name}
  \section{Project Description}
  \@projectdescription
}

\endinput
```

↓ Input

Here's an example document:

↑ Input

```
\documentclass{sample-form}

\name{Mabel Canary}

\begin{ProjectDescription}
This project will be very interesting.

This is another paragraph.
\end{ProjectDescription}
```

```
\begin{document}
\makeform
\end{document}
```

Input

The result is shown in Figure 11.4. You can download or view this example document.

11.2 Electronic PDF Forms

The previous section just considered a PDF form that could be filled in using custom commands within the document. This section looks at creating a PDF form where a user can fill the form in using interactive buttons and text fields in a PDF viewer that supports forms, such as Adobe Reader.

The hyperref package [68] provides commands to generate an electronic PDF form. There's also the eforms package [90], which is part of the AcroTEX bundle, however this is only in MiKTEX and not in TEX Live. This section will look at using the hyperref package, since it's available in both MiKTEX and TEX Live.

Output

1 Applicant Details

Name: Mabel Canary

2 Project Description

This project will be very interesting.

This is another paragraph.

Figure 11.4 A Simple Form with a Text Area.

The commands that generate the interactive elements of the form, must all be placed inside the Form environment.

\begin{Form}[⟨*parameters*⟩]
⟨*form body*⟩ Definition
\end{Form}

The optional argument ⟨*parameters*⟩ is a key=value list of options if a "submit" button is included in the form. Available options are:

`action` The value should be the URL to process the form data.

`encoding` The encoding of the URL. The norm is PDF-encoding. The only valid value for this option is `html`.

`method` Values can be `post` or `get`.

The interactive elements can be created within the Form environment using any of the following commands:

\TextField[⟨*options*⟩]{⟨*label*⟩} Definition

to create a text field;

\CheckBox[⟨options⟩]{⟨label⟩} Definition

to create a check box;

\ChoiceMenu[⟨options⟩]{⟨label⟩}{⟨choices⟩} Definition

to create a selection of choices, such as a list menu, a popup menu, a combo menu or a group of radio buttons, where ⟨*choices*⟩ is a comma-separated list of labels for each available choice or a key=value list of ⟨*label*⟩=⟨*name*⟩ options;

\PushButton[⟨options⟩]{⟨label⟩} Definition

to create a push button;

\Submit[⟨options⟩]{⟨label⟩} Definition

to create a submit button and

\Reset[⟨options⟩]{⟨label⟩} Definition

to create a reset button.

In each case, the field has a textual label (given by ⟨*label*⟩) and a key=value list of options. There are a large number of options available. For the full list, see the hyperref manual [68]. A selection of common options follows:

`accesskey` Specifies the shortcut key to activate/focus an element. There is no default value.

`align` Alignment within a text field. Allow values: `0` (left-aligned), `1` (centred), `2` (right-aligned). The default value is `0`.

`combo` A boolean key to indicate if the choice list is a combo menu. The default value is `false`.

`default` The default value for a field.

`hidden` A boolean key to indicate if the field is hidden. The default value is `false`.

`menulength` The number of elements shown in a choice list. The default value is `4`.

`multiline` A boolean key to indicate if the text field is a multiline field.

`name` The name of the field (defaults to the label if omitted). Note that the label is the text that appears by the side of the field (or on the button, in the case of a push button) whereas the name identifies the field when referenced in the script that processes the form.

`password` A boolean key to indicate if the text field is a password field. The default value is `false`.

`popdown` A boolean key to indicate if the choice list is a popdown menu. The default value is `false`.

`radio` A boolean key to indicate if the choice list is a group of radio buttons. The default value is `false`.

`value` The initial value for the field.

Example 58. A Simple Electronic Form

This example form doesn't have a submit button. Here, the user just fills in the form using the interactive elements and either prints it out or saves it. (The ability to save the PDF file depends on the PDF viewer, but if you aren't able to save it you may be able to print it to another PDF file, using a "print to file" option in your printer dialog.)

↑ Input

```
\documentclass{article}

\usepackage{hyperref}
```

```
\begin{document}
\begin{Form}
\TextField{Name}\qquad \TextField{Date}

\ChoiceMenu[combo]{Gender}{Male,Female}

\ChoiceMenu[radio]{Project}{cookies,cakes,chocolates,raygun}

Which ice cream flavours do you like?

\CheckBox{vanilla}
\CheckBox{mint}
\CheckBox{toffee}
\CheckBox{fudge}
\CheckBox[name=guarana]{guaran\'a}
\CheckBox{strawberry}
\CheckBox{raspberry}
\CheckBox{chilli}
\CheckBox{other}

\end{Form}
```

```
\end{document}
```

Input

How the form elements are rendered depends on your PDF viewer. For example, Figure 11.5 shows this form displayed in Adobe Reader and Figure 11.6 shows the same file displayed in Google Chrome. For me, Google Chrome works best (except when it hangs) as there's no native 64 bit Linux version of Adobe Reader, which means I have to run Adobe Reader on Wine and some of the interactive elements cause it to crash. If you use another operating system, you may find that the Adobe PDF viewers, such as Adobe Reader, produce suitable results.

Unfortunately I can't find any other Linux-based PDF viewers that render this example correctly. Figure 11.7 shows the same PDF file viewed in Okular. This renders most of the interactive elements correctly, but fails on the group of radio buttons. Only the first radio button is correctly rendered as an interactive element. The other radio buttons appear as non-interactive open single quote marks. (These appear to be the decorative open quote mark ❛ from the ZapfDingbats font, `\ding{123}`.) A similar problem occurs with Evince and with the document viewer that comes with TeXworks. Other PDF viewers, such as Sumatra or the Linux version of Foxit (Figures 11.8 and 11.9), don't recognise any of the interactive elements (but Foxit on Windows does show the interactive elements, see Figure 11.10). Therefore, you will need to take care about your choice of

PDF viewer if you want to create an electronic PDF form (and the PDF viewer for any users of your form).

You may have noticed from the above example that each field's label (such as "Name" or "Project") is placed to the left of the interactive element (or elements, in the case of the radio group). This layout is governed by:

`\LayoutTextField{`⟨*label*⟩`}{`⟨*field*⟩`}` Definition

for text fields,

`\LayoutChoiceField{`⟨*label*⟩`}{`⟨*field*⟩`}` Definition

for choice fields, and

`\LayoutCheckField{`⟨*label*⟩`}{`⟨*field*⟩`}` Definition

for check boxes. These all default to ⟨*label*⟩ ⟨*field*⟩. Since the space is a regular breakable space, this allowed a line break to occur between the label "raspberry" and its associated check box (as can be seen, for example, in Figure 11.5). To prevent this, `\LayoutCheckField` can be redefined to use a non-breakable space:

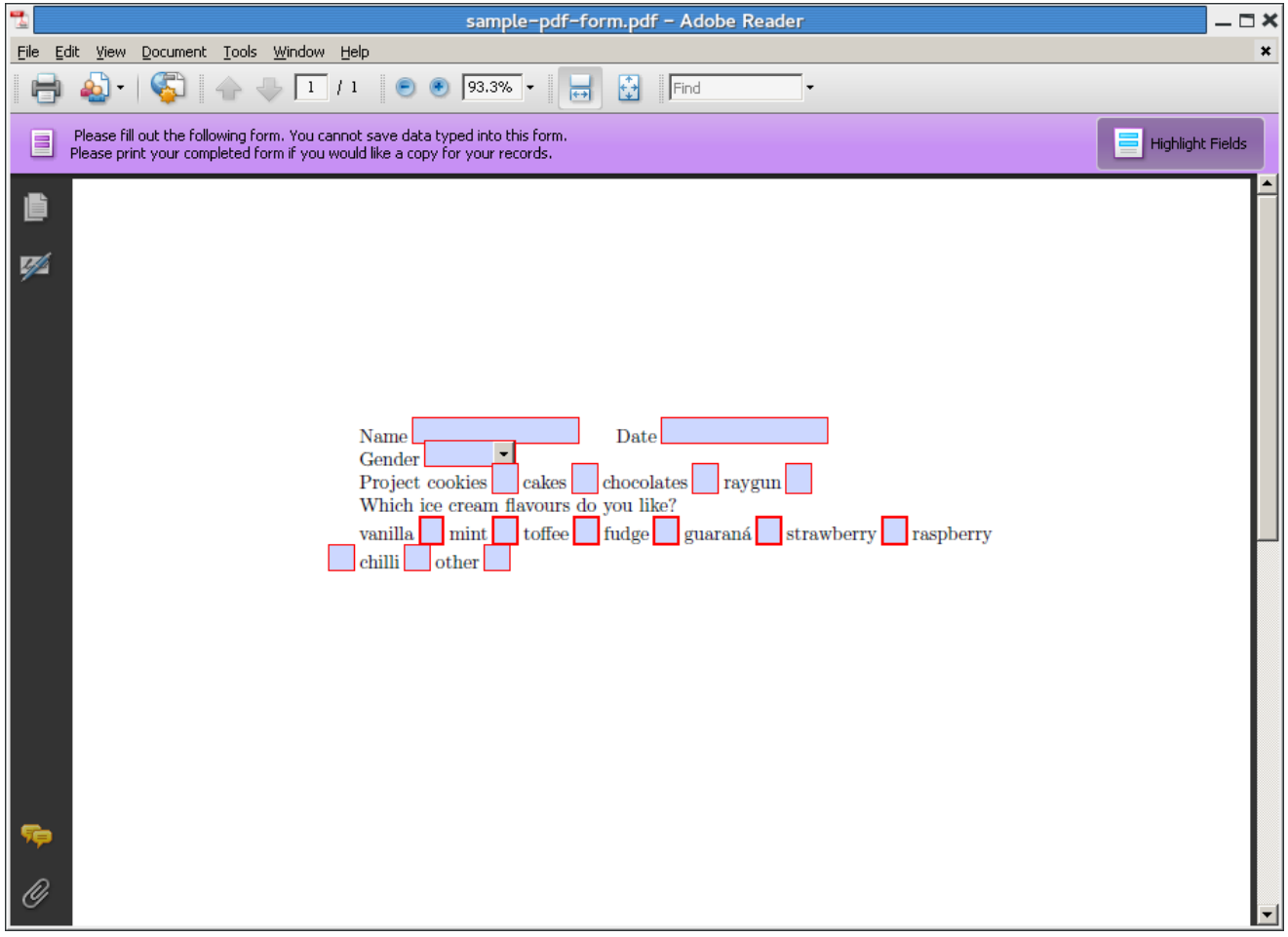


Figure 11.5 Example Interactive PDF Form Viewed in Adobe Reader

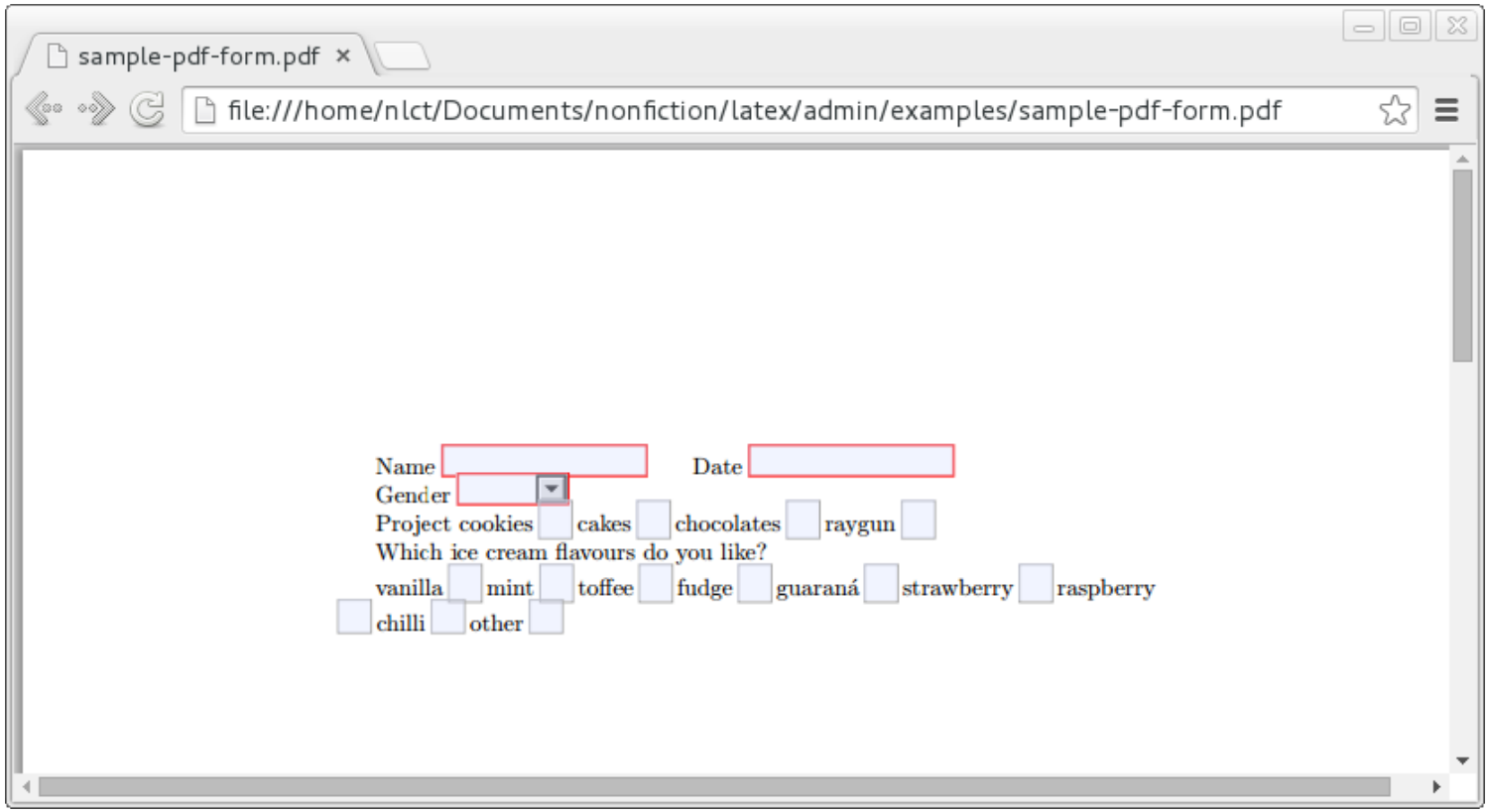


Figure 11.6 Example Interactive PDF Form Viewed in Google Chrome

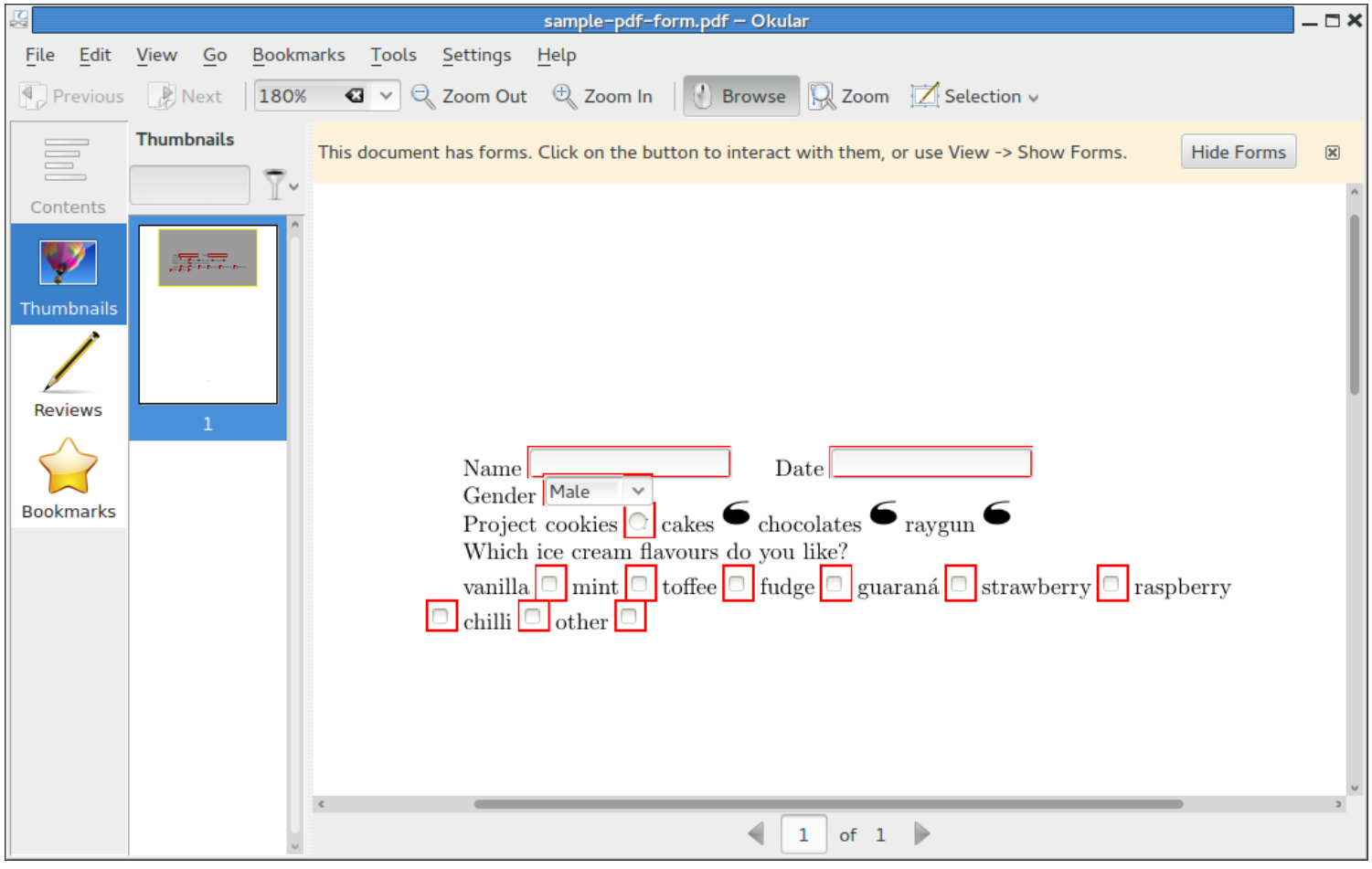


Figure 11.7 Example Interactive PDF Form Viewed in Okular

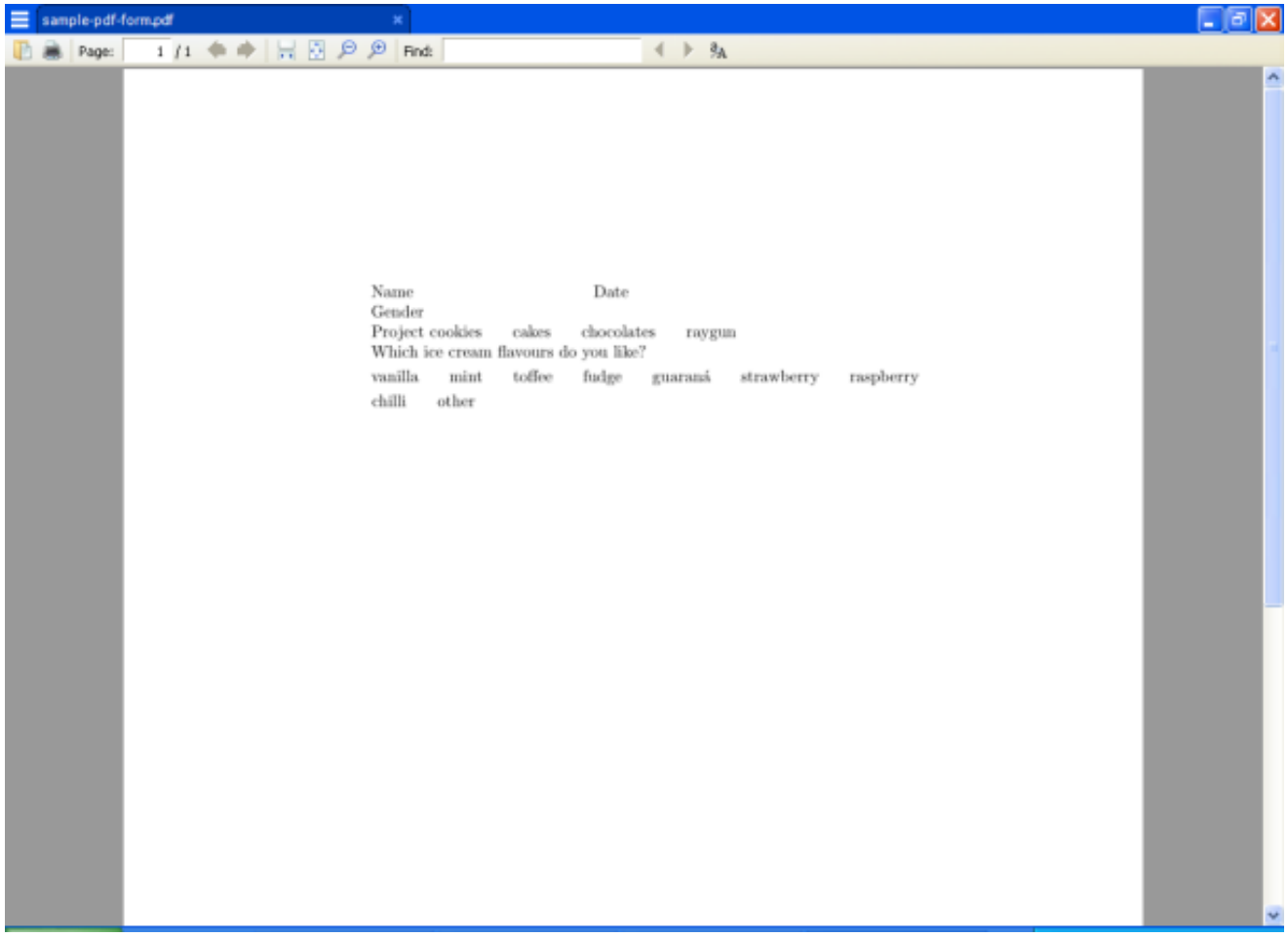


Figure 11.8 Example Interactive PDF Form Viewed in Sumatra

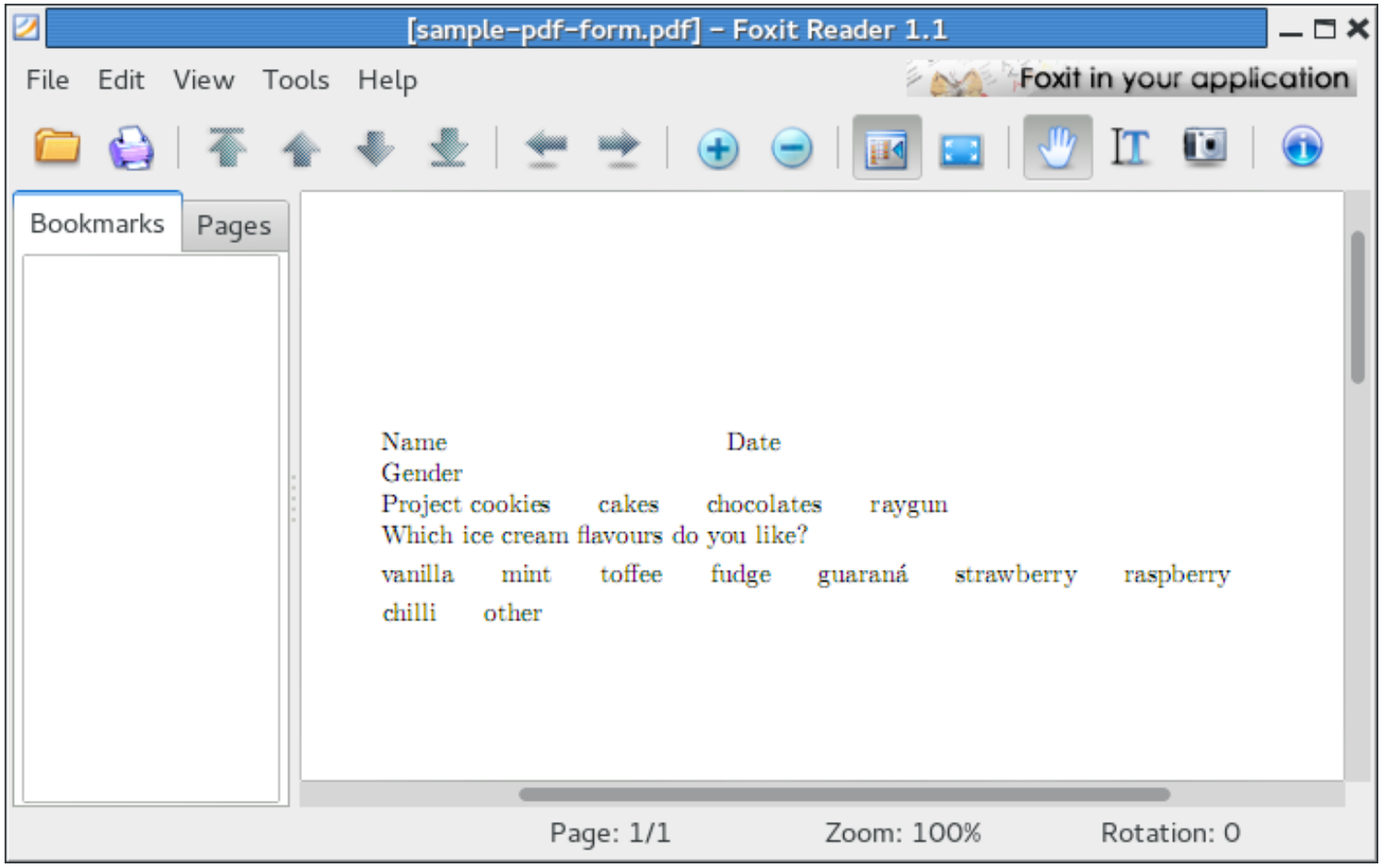


Figure 11.9 Example Interactive PDF Form Viewed in Foxit on Linux

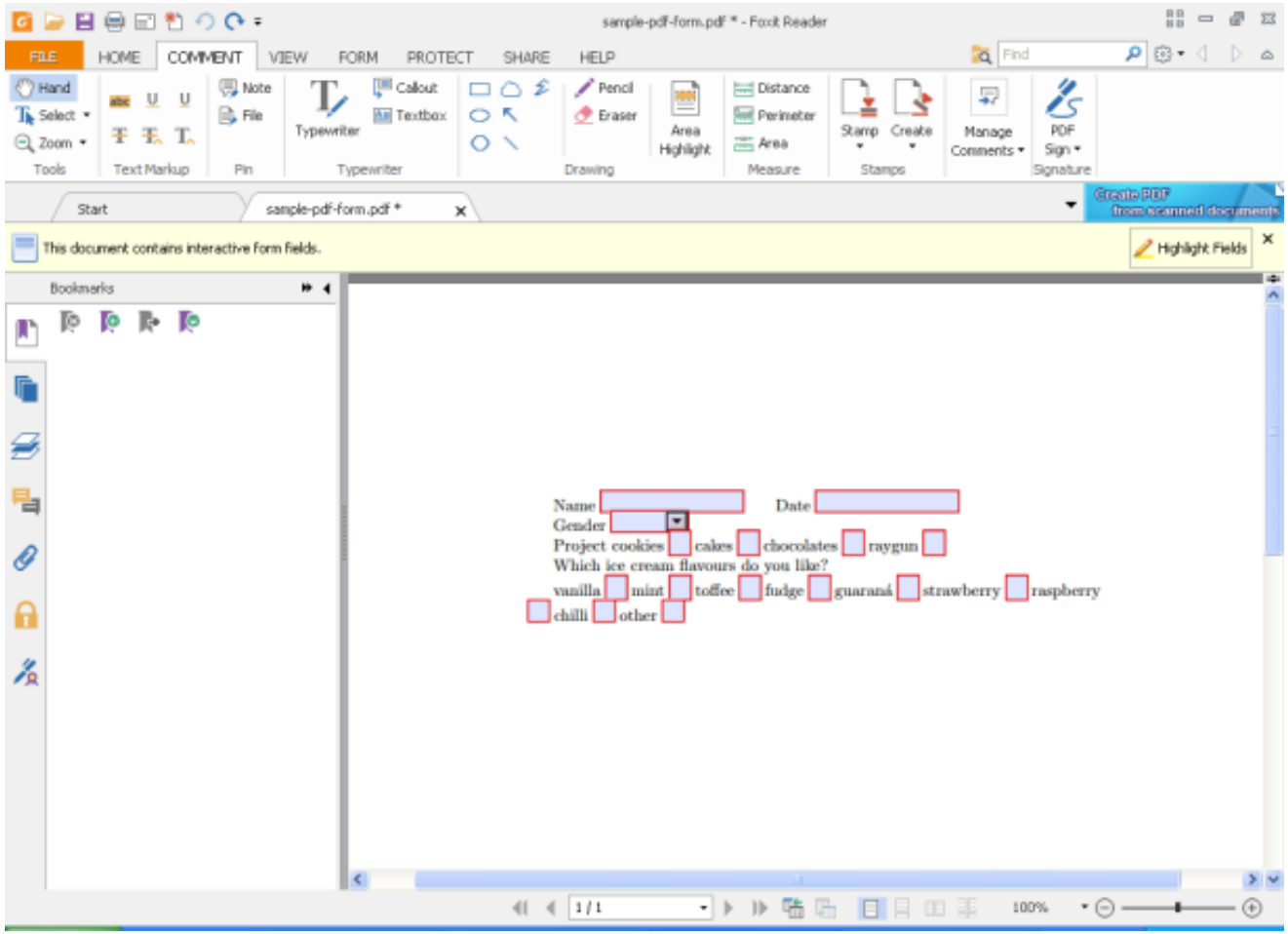


Figure 11.10 Example Interactive PDF Form Viewed in Foxit on Windows

```
\renewcommand*{\LayoutCheckField}[2]{#1~#2}
```
Input

Alternatively, if you also want the check box and label swapped round, so that the label is on the right:

```
\renewcommand*{\LayoutCheckField}[2]{#2~#1}
```
Input

Note that \LayoutChoiceField just controls the layout of the label for the list of choices and the choice list element or group of elements. In the case of a group of radio buttons, each radio button has a fixed layout with the radio button label first followed by a space and then the radio button. There's no user level macro for changing this layout.

The actual field display is given by:

\MakeRadioField{⟨*width*⟩}{⟨*height*⟩} Definition

for radio fields;

\MakeCheckField{⟨*width*⟩}{⟨*height*⟩} Definition

for check boxes;

\MakeTextField{⟨*width*⟩}{⟨*height*⟩} Definition

for text areas;

\MakeChoiceField{⟨*width*⟩}{⟨*height*⟩} Definition

for choice lists. These commands all default to creating a blank area of the given ⟨*width*⟩ and ⟨*height*⟩. (This is how, in Figure 11.9, the area taken up by the interactive elements appears as a blank space even though the fields aren't rendered. The actual rendering of the field is to some extent determined by the PDF viewer.)

The layout of the text on push buttons is determined by

\MakeButtonField{⟨*text*⟩} Definition

This defaults to just ⟨*text*⟩.

The default dimensions are given by the commands:

\DefaultHeightofSubmit Definition

for the default height of the submit button (`14pt`);

\DefaultWidthofSubmit Definition

for the default width of the submit button (`2cm`);

\DefaultHeightofReset Definition

for the default height of the reset button (`14pt`);

\DefaultWidthofReset Definition

for the default width of the reset button (2cm);

\DefaultHeightofCheckBox Definition

for the default height of check boxes (\baselineskip);

\DefaultWidthofCheckBox Definition

for the default width of check boxes (\baselineskip);

\DefaultHeightofChoiceMenu Definition

for the default height of choice boxes (\baselineskip);

\DefaultWidthofChoiceMenu Definition

for the default width of choice boxes (\baselineskip);

\DefaultHeightofText Definition

for the default height of single-lined text fields (\baselineskip);

\DefaultHeightofTextMultiline Definition

for the default height of multi-lined text fields (4\baselineskip);

\DefaultWidthofText Definition

for the default width of text fields (3cm). Note that these are all macros not lengths, so you need to use \renewcommand to change them. These defaults are used for fields that don't have the height or width options specified (in the optional argument of the field commands, such as \CheckBox).

Suppose now I want the check boxes from Example 58 to appear in a tabular layout, so that they appear more like those from Figure 11.2. A first attempt might look something like:

↑ Input

```
\renewcommand*{\LayoutCheckField}[2]{#2 #1}

Which ice cream flavours do you like? (Tick all that apply.)

\begin{center}
\begin{tabular}{lll}
\CheckBox{vanilla} &
\CheckBox{mint} &
\CheckBox{toffee}\\
\CheckBox{fudge} &
\CheckBox[name=guarana]{guaraná} &
```

```
\CheckBox{strawberry}\\
\CheckBox{raspberry}&
\CheckBox{chilli} &
\CheckBox{other}
\end{tabular}
\end{center}
```

↓ Input

However, this produces the form shown in Figure 11.11. The check boxes are far too narrow. Recall from above that the default width of the check boxes is given by \DefaultWidthofCheckBox, which is initialised to \baselineskip. One of the peculiarities of the tabular environment is that it temporarily sets the value of \baselineskip to 0 pt. This means that check boxes default to zero width when placed inside a tabular environment. Here's a second attempt that changes the defaults to depend on the font size instead:

↑ Input

```
\renewcommand*{\LayoutCheckField}[2]{#2 #1}
\renewcommand*{\DefaultWidthofCheckBox}{2ex}
\renewcommand*{\DefaultHeightofCheckBox}{2ex}
```

```
Which ice cream flavours do you like? (Tick all that apply.)

\begin{center}
\begin{tabular}{lll}
\CheckBox{vanilla} &
\CheckBox{mint} &
\CheckBox{toffee}\\
\CheckBox{fudge} &
\CheckBox[name=guarana]{guaraná} &
\CheckBox{strawberry}\\
\CheckBox{raspberry}&
\CheckBox{chilli} &
\CheckBox{other}
\end{tabular}
\end{center}
```

Input

This produces Figure 11.12, which has check boxes with a better width, but the heights are too large causing them to overlap. In fact, they are higher than the specified 2 ex given in the redefinition of \DefaultHeightofCheckBox. This seems to be caused by the tabular environment stretching the boxes to fill the available height, but this occurs outside of the box created by \MakeCheckField.

Here's a third attempt that explicitly sets the width and height within the definition of \LayoutCheckField using a \parbox:

Input

```
\renewcommand*{\LayoutCheckField}[2]{#2 #1}
\renewcommand*{\DefaultWidthofCheckBox}{2ex}
\renewcommand*{\DefaultHeightofCheckBox}{2ex}
\renewcommand*{\LayoutCheckField}[2]{%
 \parbox[\DefaultHeightofCheckBox]{\DefaultWidthofCheckBox}{#2}
#1}

Which ice cream flavours do you like? (Tick all that apply.)

\begin{center}
\begin{tabular}{lll}
\CheckBox{vanilla} &
\CheckBox{mint} &
\CheckBox{toffee}\\
\CheckBox{fudge} &
\CheckBox[name=guarana]{guaraná} &
\CheckBox{strawberry}\\
\CheckBox{raspberry}&
```

```
\CheckBox{chilli} &
\CheckBox{other}
\end{tabular}
\end{center}
```

Input

This produces the result shown in Figure 11.13, which now has square check boxes. Unfortunately this doesn't take into account the `height` or `width` options that may override the default sizes. In this case, that's not an issue as I want all the check boxes to be the same size. If you want larger check boxes in another area of the form, you can localise the effects of the above redefinition of `\LayoutCheckField` by scoping it. For example, by placing it inside the start of the `center` environment before the start of the `tabular` environment:

Input

```
\begin{center}
\renewcommand*{\LayoutCheckField}[2]{%
 \parbox[\DefaultHeightofCheckBox]{\DefaultWidthofCheckBox}{#2}
#1}%
\begin{tabular}{lll}
```

Input

The other possibility is to use the internal commands `\Fld@width` and `\Fld@height`, which store the width and height for the check box:

```
\parbox[\Fld@height]{\Fld@width{#2} #1}
```
Input

However, be careful about using internal commands that aren't part of the LaTeX kernel as they may change with future versions. Also remember that internal commands must be placed inside a class or package or should be enclosed inside `\makeatletter`...`\makeatother`.

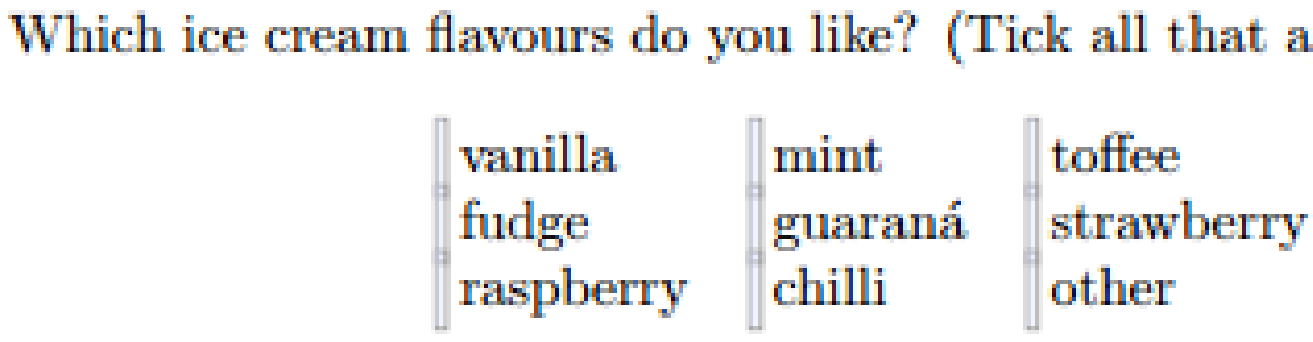


Figure 11.11 First Attempt at Laying Out Check Boxes in Rows and Columns

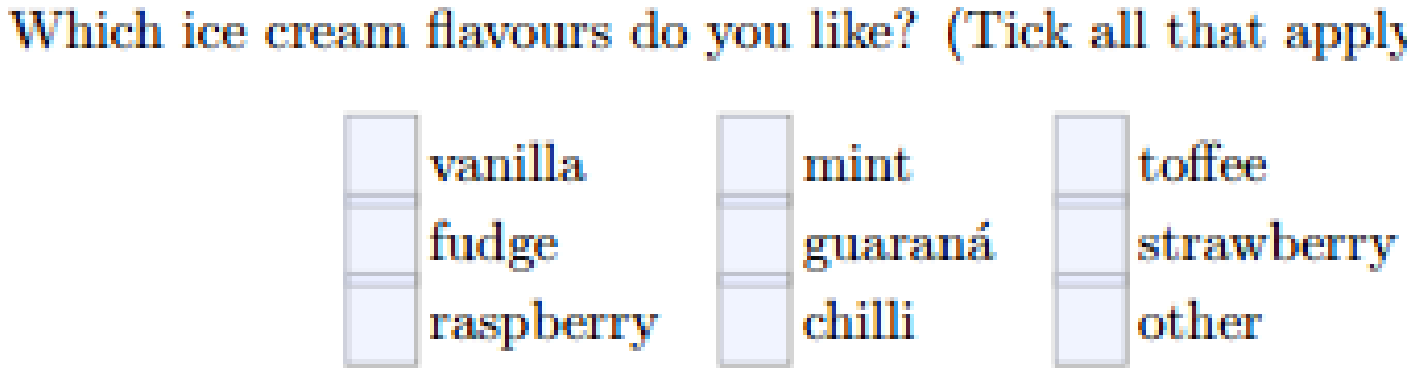


Figure 11.12 Second Attempt at Laying Out Check Boxes in Rows and Columns

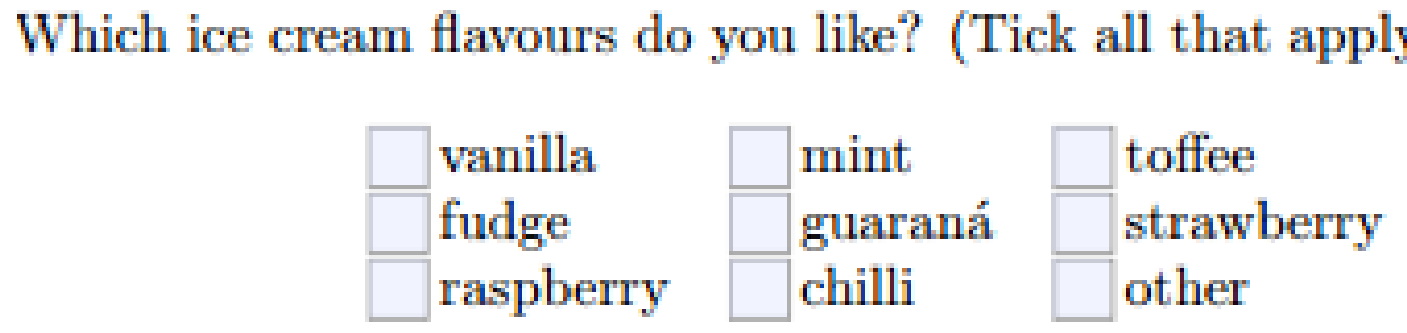


Figure 11.13 Third Attempt at Laying Out Check Boxes in Rows and Columns

12. Charts

Charts and diagrams can be produced in any graphical application that can export the image to a format that LaTeX can input. However it is also possible to write LaTeX code to generate the diagram. This has the advantage in that the fonts used in the diagram match those used in the rest of the document, but it's more complicated and can significantly slow the document build time.

This chapter describes LaTeX packages to generate various charts you may need in your administrative work. If you prefer to use a graphics application to generate a chart you can input the exported image using the graphicx package, as described in Volume 1 [92, §6], but make sure, if possible, that you export your image using a vector graphics format (such as PDF or EPS[1]) rather than a bitmap (such as PNG or JPEG).

There are many LaTeX packages available, ranging from general drawing packages, such as tikz or pstricks, to packages designed for specific types

[1] Note that the PDF and EPS file formats also support bitmaps so, if possible, check the settings on whatever application you use to create the image files to see if it uses a vector graphics format. If the image appears fuzzy when you magnify it, then it's most likely a bitmap.

of charts. See, for example, the diagram topic and sub-topics such as the diagram-block topic (block diagrams) and diagram-ctrl topic (control diagrams), as well as the genchart topic (bar- or pie-charts), planning topic (timelines and schedules) and gantt topic. There's also the pgf-tikz topic (for packages that use pgf/tikz) and the pstricks topic (for packages that use pstricks).

⚠ With the increase in computer graphics over the last couple of decades, there has been a corresponding rise in jazzed-up three-dimensional charts designed to impress the lay person. Such charts can be found from glossy brochures to company annual reports or news programs, but while these images may appear visually appealing, they distort the data and can produce a misleading impression. As a chartered mathematician I can't condone such deception, whether done by design or accident, so I'm not going to show you how to produce fancy effects.

⚠ Be careful if you have large numbers or you may get the "Dimension too large" TEX error. If you are dealing with very large values (in terms of magnitude), you may be better off using a custom data-handling tool to generate the image rather than trying to use TEX.

12.1 Flow Charts

The diagram-flow topic lists several packages for flow and similar diagrams, only two of which are available on both MiKTEX and TEX Live, and only one

of these is for flow charts, and that's the flowchart package, which requires the makeshape and tikz packages. Alternatively, you can just use the tikz package directly.

The tikz package [101] has already been briefly introduced in Sections 7.5, 10.3 and 11.1. Drawing a flow chart will also require the tikz libraries[2] `arrows.meta` and `shapes.geometric`, which can be loaded in the preamble using:

Input

```
\usetikzlibrary{arrows.meta}
\usetikzlibrary{shapes.geometric}
```

Input

The `positioning` library is also useful as it provides convenient ways of positioning nodes:

Input

```
\usetikzlibrary{positioning}
```

Recall from §7.5, that within the tikzpicture environment, you can use

[2] These are fairly new libraries, so you'll need an up-to-date version of pgf/tikz in order to use them.

\path[⟨path options⟩] (⟨position⟩) node[⟨node options⟩] (⟨node name⟩) {⟨text⟩}; Definition

to position a node or you can use the shortcut:

\node[⟨node options⟩] (⟨node name⟩) {⟨text⟩}; Definition

The position can be specified within [⟨node options⟩] using at=(⟨position⟩) or using the at keyword:

\node[⟨node options⟩] at (⟨position⟩) (⟨node name⟩) {⟨text⟩}; Definition

Alternatively, if you use the positioning library, the node can be placed relative to another node using one of the following ⟨key⟩=⟨value⟩ options:

above	Place this node above the location specified in the ⟨value⟩.
below	Place this node below the location specified in the ⟨value⟩.
left	Place this node to the left of the location specified in the ⟨value⟩.
right	Place this node to the right of the location specified in the ⟨value⟩.
above left	Place this node above left of the location specified in the ⟨value⟩.

`above right` Place this node above right of the location specified in the ⟨*value*⟩.

`below left` Place this node below left of the location specified in the ⟨*value*⟩.

`below right` Place this node below right of the location specified in the ⟨*value*⟩.

For each of these options, the ⟨*value*⟩ part may simply be in the form ⟨*shift*⟩ or in the form ⟨*shift*⟩ `of` ⟨*label*⟩, where shift may be a dimension (or an expression that evaluates to a dimension) or a number (in which case the unit is the tikz unit currently in use). If `of` ⟨*label*⟩ is present then the shift is relative to the node identified by ⟨*label*⟩. If the ⟨*shift*⟩ part is omitted, the default node distance is used. For further details, and for details of other placement options, see the pgf manual [101].

Example:

↑ Input

```
\begin{tikzpicture}
\node (start) {Ray-gun doesn't work};
\node[below=of start] (query) {Is it charged?};
```

```
\node[right=of query] (recharge) {Recharge battery};
\node[below=of query] (repair) {Repair ray-gun};
\end{tikzpicture}
```

Input

This produces the image shown in Figure 12.1.

Output

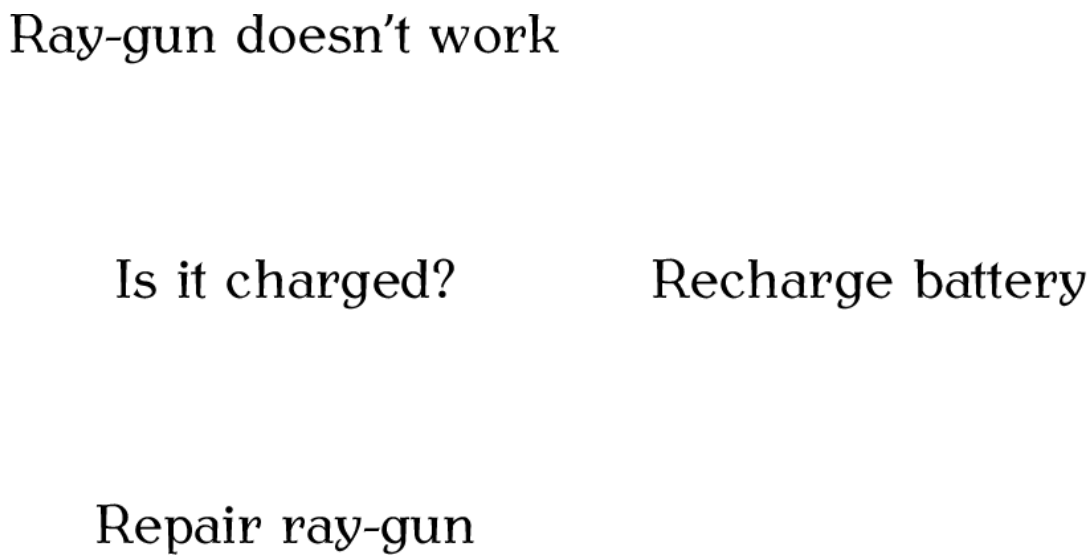


Figure 12.1 Nodes Positioned Relative to Each Other

At the moment this doesn't look much like a flow chart. The nodes all default to a rectangular shape, but the shape isn't visible unless you use

draw, for the outline, or fill, for the interior, within the ⟨*node options*⟩ specifications. In both cases, you can optionally supply a colour name. Since the xcolor package is automatically loaded by tikz, you can apply colour mixtures using the ! specification, such as red!50 to indicate 50% red (see the xcolor documentation [40] for further details).

Example:

↑ Input

```
\begin{tikzpicture}
\node[draw,fill=red!30] (start) {Ray-gun doesn't work};
\node[draw,fill=yellow,below=of start] (query) {Is it charged?};
\node[draw,fill=green!40,right=of query] (recharge)
{Recharge battery};
\node[draw,fill=green!40,below=of query] (repair)
{Repair ray-gun};
\end{tikzpicture}
```

↓ Input

This produces the image shown in Figure 12.2.

The rectangles can be given round corners using the rounded corners option. For example:

Output

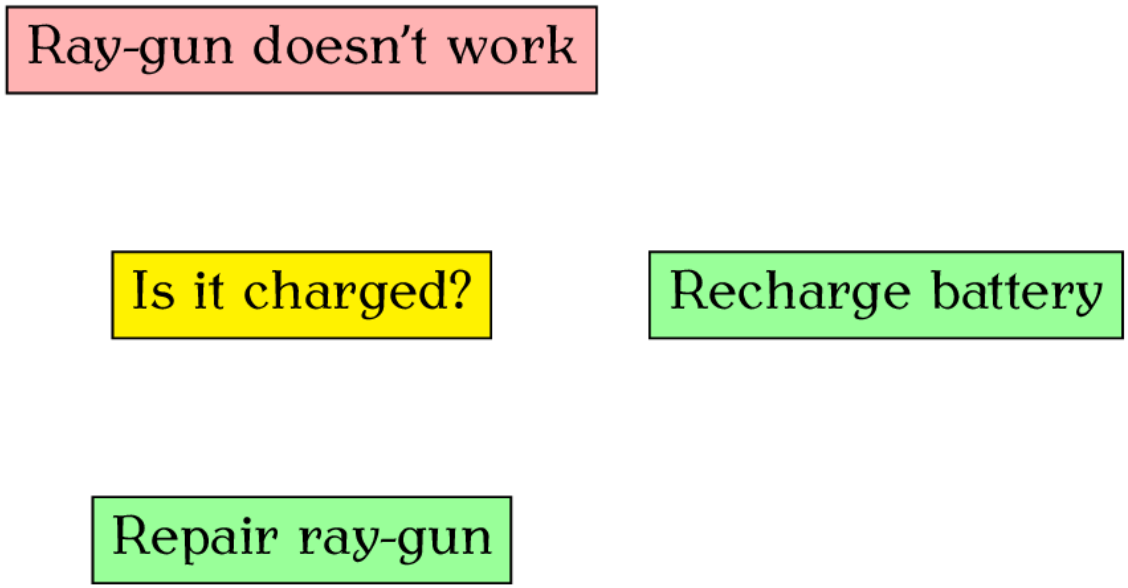


Figure 12.2 Node Shapes Drawn and Filled

```
\node[rounded corners,draw,fill=green!40,below=of query]
  (repair) {Repair ray-gun};
```

Input

Input

If you want to change the node shape, there are a number of shapes provided by various tikz libraries. For example, the diamond shape is provided by the shapes.geometric library. The shape name is given in the ⟨*node options*⟩. For example:

```
\node[diamond,draw,fill=yellow,below=of start] (query)
  {Is it charged?};
```

Input

Input

The default aspect ratio of the diamond width and height is 1. You can change this using the aspect option. For example:

```
\node[diamond,aspect=2,draw,fill=yellow,below=of start] (query)
  {Is it charged?};
```

Input

Input

A line can be drawn between two nodes using:

```
\draw[⟨options⟩] (⟨node1 label⟩) -- (⟨node2 label⟩);
```

Definition

For example:

```
\draw (start) -- (query);
```

Input

Arrow heads can be added to the start and end of the line using the option ⟨*start arrow*⟩-⟨*end arrow*⟩, where ⟨*start arrow*⟩ and ⟨*end arrow*⟩ indicate the type of arrow head. The simplest arrow types are given by < for an arrow head pointing to the start and > for an arrow head pointing to the end. The ⟨*start arrow*⟩ or ⟨*end arrow*⟩ may be omitted if no arrow head is needed at the start or end, respectively.

Example:

↑ Input

```
\begin{tikzpicture}
\node[rounded corners,draw,fill=red!30] (start)
 {Ray-gun doesn't work};
\node[diamond,aspect=2,draw,fill=yellow,below=of start] (query)
  {Is it charged?};
```

```
\node[draw,fill=green!40,rounded corners,right=of query]
  (recharge) {Recharge battery};
\node[draw,fill=green!40,rounded corners,below=of query]
  (repair) {Repair ray-gun};
% draw in arrows:
\draw[->] (start) -- (query);
\draw[->] (query) -- (recharge);
\draw[->] (query) -- (repair);
\end{tikzpicture}
```

Input

This produces the image shown in Figure 12.3.

A node can be added to a path. For example:

```
\draw[->] (query) -- (recharge) node[midway,above] {No};
```

Input

This places a node (with the text "No") above and midway along the line between the query and recharge nodes.

Example 59. Flow Chart

This example builds on the above. The arrows.meta library is loaded in order to use the Triangle[] arrow tip. This can be used by replacing the > arrow tip specifier in the optional argument to \draw. For example:

Output

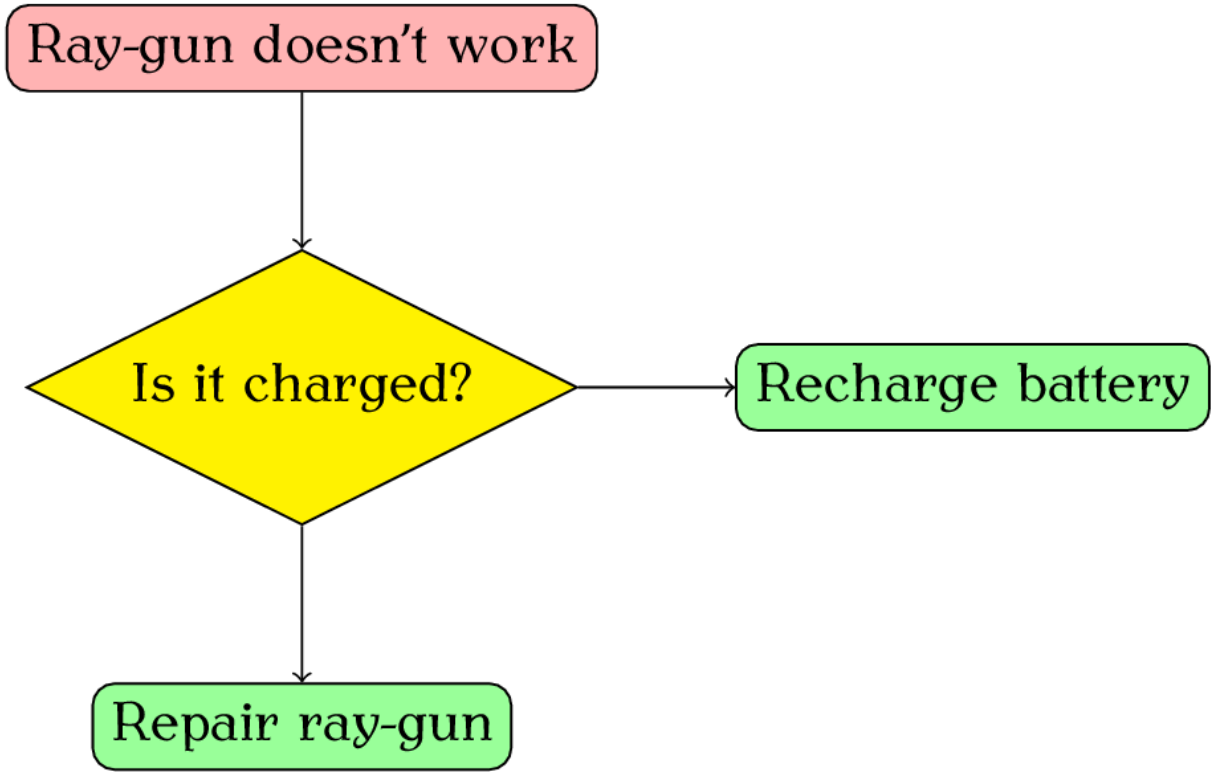


Figure 12.3 Nodes with Connecting Arrows

```
\draw[-{Triangle[]}] (start) -- (query);
```
Input

Alternatively, the > arrow tip specifier can be set to Triangle[] for the given scope. For example:

```
\begin{tikzpicture}[>={Triangle []}]
```
Input

This helps to ensure consistent arrow tips within the picture and means that you only need to edit one line if you decide to change the arrow tips (for example, from Triangle[] to Stealth[]).

Common node settings can be specified using every node/.style={⟨*node options*⟩} within the optional argument of the tikzpicture environment (to apply to all nodes within the environment) or the effect can be scoped using the scope environment (recall §7.5). This method can be used for the common settings for the recharge and repair nodes.

In addition, a thicker line width is set using the ultra thick option.

↑ Input

```
\documentclass{article}

\usepackage{tikz}[2013/12/13]% use at least version 3.0
\usetikzlibrary{arrows.meta}
\usetikzlibrary{shapes.geometric}
```

```
\usetikzlibrary{positioning}

\begin{document}

\begin{tikzpicture}[ultra thick,>={Triangle[]}]
\node[rounded corners,draw,fill=red!30] (start)
  {Ray-gun doesn't work};
\node[diamond,aspect=2,draw,fill=yellow,below=of start] (query)
  {Is it charged?};
\begin{scope}[every node/.style={draw,fill=green!40,
rounded corners}]
 \node[right=of query] (recharge) {Recharge battery};
 \node[below=of query] (repair) {Repair ray-gun};
\end{scope}
\draw[->] (start) -- (query);
\draw[->] (query) -- (recharge) node[midway,above] {No};
\draw[->] (query) -- (repair) node[midway,right] {Yes};
\end{tikzpicture}

\end{document}
```

↓ Input

Note that the last three lines of the tikzpicture environment above can

also have the arrow tips automatically added through the use of the scope environment:

Input

```
\begin{scope}[->]
 \draw (start) -- (query);
 \draw (query) -- (recharge) node[midway,above] {No};
 \draw (query) -- (repair) node[midway,right] {Yes};
\end{scope}
```

Input

This produces the image shown in Figure 12.4. You can download or view this example.

12.2 Pie Charts

At the time of writing, there are three pie chart packages listed on the genchart topic: pgf-pie (which uses pgf/tikz), piechart (shell and AWK scripts to generate pie-charts expressed as pstricks code) and piechartmp (which uses MetaPost). The pgf-pie package is available on MiKTEX but is not on TEX

Output

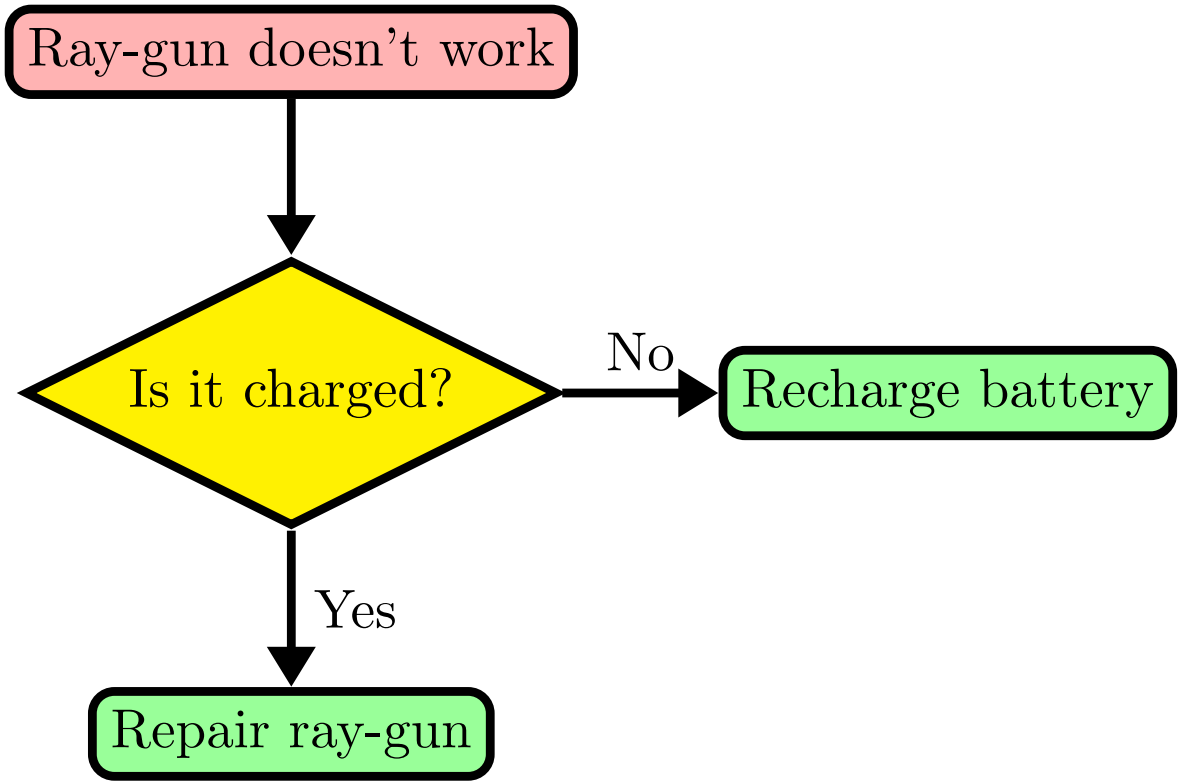


Figure 12.4 An Example Flow Chart

Live.[3] The piechart bundle isn't available on either MiKTEX or TEX Live, and only comes with a README file dated 1998. The piechartmp package is available on both MiKTEX and TEX Live, but requires writing MetaPost code which hasn't been covered in this series of books. In addition to these packages, the datatool bundle comes with the datapie package, which can be used to display pie charts from data stored in a datatool database.

Since the datatool package has already been introduced in this book, §12.2.1 discusses the datapie package. If you don't want the additional overhead that comes with the datatool package, §12.2.2 discusses the pgf-pie package, but TEX Live users will have to install the package manually.

12.2.1 The datapie Package

As mentioned above, the datapie package is part of the datatool bundle [94]. The datapie package automatically loads the datatool and tikz packages. The following package options are provided:

`color` Use colour (default).

`gray` Use greyscale.

[3] The licence is unknown and therefore has to be assumed to be non-free.

`rotateinner` Rotate the inner labels so that they are aligned with the pie chart radial axis.

`norotateinner` Don't rotate the inner labels (default).

`rotateouter` Rotate the outer labels so that they are aligned with the pie chart radial axis.

`norotateouter` Don't rotate the outer labels (default).

Once you have loaded the data (see §2.2), the numerical data within the database can be displayed as a pie chart using:

`\DTLpiechart[`⟨*condition*⟩`]{`⟨*settings*⟩`}{`⟨*db-name*⟩`}{`⟨*assign list*⟩`}` Definition

The optional argument ⟨*condition*⟩ is the same as that for `\DTLforeach`, ⟨*db-name*⟩ is the label uniquely identifying the database, and ⟨*assign list*⟩ is a comma-separated list of ⟨*cmd*⟩=⟨*col-label*⟩ pairs, the same as the penultimate argument of `\DTLforeach`. The remaining argument ⟨*settings*⟩ is a key=value list. The `variable` key must be present, but the remaining keys may be omitted.

`variable` The command to use (as specified in ⟨*assign list*⟩) that contains the data to be used to construct the pie chart. (Required.)

`start` The starting angle (degrees) of the first segment. The default is 0.

`radius` The radius of the pie chart. The default is 2 cm. This sets the length `\DTLradius`.

`innerratio` The distance from the centre of the pie chart to the point where the inner labels are placed is given by this value multiplied by the radius. This must come after `radius`, if the radius also needs to be set. The default is 0.5.

`inneroffset` The distance from the centre of the pie chart to the point where the inner labels are placed. This may be used instead of `innerratio`. If `inneroffset` is omitted, the `innerratio` is used.

`outerratio` The distance from the centre of the pie chart to the point where the outer labels are placed is given by this value multiplied by the radius. This must come after `radius`, if the radius also needs to be set. The default is 1.25.

`outeroffset` The distance from the centre of the pie chart to the point where the outer labels are placed. This may be used in-

stead of outerratio. If outeroffset is omitted, the outerratio is used.

cutawayratio — The distance from the centre of the pie chart to the point of cutaway segments is given by this value multiplied by the ratio. This must come after radius, if the radius also needs to be set. The default is 0.2.

cutawayoffset — The distance from the centre of the pie chart to the point of cutaway segments. This may be used instead of cutawayratio. If cutawayoffset is omitted, the cutawayratio value is used.

cutaway — The list of cutaway segments. This should be a comma-separated list of individual numbers, or number ranges (separated by a dash). For example, cutaway={1,3} will separate the first and third segments from the rest of the pie chart, whereas cutaway={1-3} will separate the first three segments. If omitted, the pie chart will be whole with no cutaway segments.

innerlabel — The inner label for the segments. The value may contain any of the commands assigned in ⟨*assign list*⟩. The default is the same as the value of the variable key.

`outerlabel` The outer label for the segments. The value may contain any of the commands assigned in ⟨*assign list*⟩. The default is empty.

`rotateinner` This is a boolean key. If `true`, the inner labels are rotated so that they are aligned with the pie chart radial axis.

`rotateouter` This is a boolean key. If `true`, the outer labels are rotated so that they are aligned with the pie chart radial axis.

The datapie package predefines colours for the first eight segments of the pie chart. If these don't suit your requirements, or if you have more than eight rows of data, you can set the colour for a given segment using:

`\DTLsetpiesegmentcolor{`⟨*n*⟩`}{`⟨*colour*⟩`}` Definition

where ⟨*n*⟩ is the segment index (starting from 1) and ⟨*colour*⟩ is the colour, as used in commands like `\color`.

There are two commands provided that can be used within the inner or outer labels:

`\DTLpievariable` Definition

This is set to the value of the `variable` key.

\DTLpiepercent

Definition

This command is set to the percentage value for the current segment. The value is rounded to ⟨*n*⟩ digits, where ⟨*n*⟩ is given by the counter DTLpieroundvar.

Example 60. An Example Pie Chart (datapie package)
Recall the sample booklist.csv file. Suppose, for some reason, I want to create a pie chart that displays the price for each book. The database contains 10 rows, so 10 segment colours need to be defined. A pie chart can be drawn as follows:

↑ Input

```
\documentclass{article}

\usepackage[x11names]{xcolor}
\usepackage{datapie}

\DTLloaddb{books}{booklist.csv}

% assign segment colours:

\DTLsetpiesegmentcolor{1}{Aquamarine1}
```

```
\DTLsetpiesegmentcolor{2}{Azure2}
\DTLsetpiesegmentcolor{3}{Burlywood3}
\DTLsetpiesegmentcolor{4}{CadetBlue2}
\DTLsetpiesegmentcolor{5}{Chartreuse3}
\DTLsetpiesegmentcolor{6}{Salmon1}
\DTLsetpiesegmentcolor{7}{DeepPink1}
\DTLsetpiesegmentcolor{8}{Goldenrod1}
\DTLsetpiesegmentcolor{9}{Honeydew1}
\DTLsetpiesegmentcolor{10}{Plum3}

\begin{document}

\DTLpiechart{%
 variable=\ThePrice,%
 innerratio=0.4,%
 innerlabel={\pounds\ThePrice},%
 rotateinner}%
 {books}% database name
 {\ThePrice=price}% assignment list

\end{document}
```

↓ Input

Output

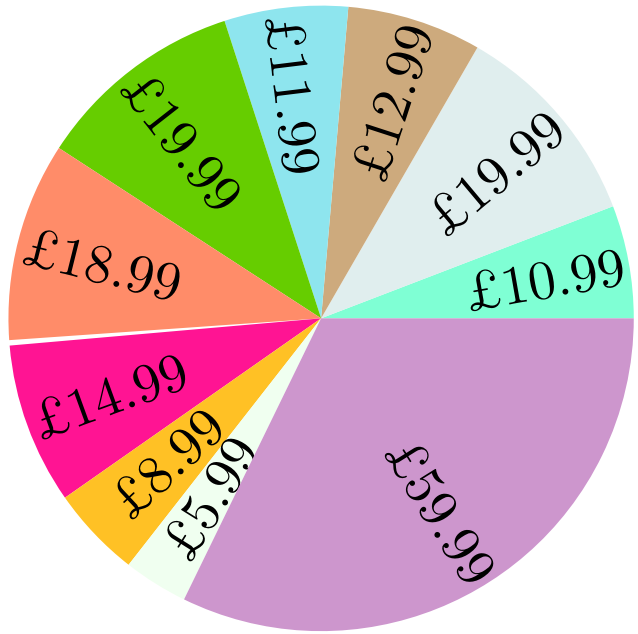


Figure 12.5 An Example Pie Chart (datapie package)

This produces the chart shown in Figure 12.5. However, this isn't particularly informative. The book titles could be added as an outer label, but as some of the titles are quite long, this would result in a rather messy chart. Instead, it would be neater to have a legend or key. The current text colour can be switched to the colour of a given segment using:

\DTLdopiesegmentcolor{⟨n⟩} Definition

where ⟨n⟩ is the segment index. This is a declaration that internally calls \color. Alternatively, you can use

\DTLdocurrentpiesegmentcolor Definition

which sets the colour for the current segment. This may be used inside a \DTLforeach loop:

↑ Input

```
\begin{tabular}{ll}
\DTLforeach*{books}{\TheTitle=title}%
{%
  \DTLiffirstrow{}{\\}%
  \DTLdocurrentpiesegmentcolor\rule{10pt}{10pt} & \TheTitle
}
\end{tabular}
```

↓ Input

Recall from Volume 1 [92, §4.7] that the tabular environment is a form of box. The pie chart created using \DTLpiechart is also a box, so the two can be placed beside each other, however you might need to adjust the vertical alignment. The complete document is as follows:

↑ Input

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\usepackage[x11names]{xcolor}
\usepackage{datapie}

\DTLloaddb{books}{booklist.csv}

% assign segment colours:

\DTLsetpiesegmentcolor{1}{Aquamarine1}
\DTLsetpiesegmentcolor{2}{Azure2}
```

```
\DTLsetpiesegmentcolor{3}{Burlywood3}
\DTLsetpiesegmentcolor{4}{CadetBlue2}
\DTLsetpiesegmentcolor{5}{Chartreuse3}
\DTLsetpiesegmentcolor{6}{Salmon1}
\DTLsetpiesegmentcolor{7}{DeepPink1}
\DTLsetpiesegmentcolor{8}{Goldenrod1}
\DTLsetpiesegmentcolor{9}{Honeydew1}
\DTLsetpiesegmentcolor{10}{Plum3}

\begin{document}

% pie chart:
\DTLpiechart{%
 variable=\ThePrice,%
 innerratio=0.4,%
 innerlabel={\pounds\ThePrice},%
 rotateinner}%
 {books}% database name
 {\ThePrice=price}% assignment list
\qquad% add some horizontal space
% legend:
\begin{tabular}[b]{ll}
```

```
\DTLforeach*{books}{\TheTitle=title}%
{%
  \DTLiffirstrow{}{\\}%
  \DTLdocurrentpiesegmentcolor\rule{10pt}{10pt} &
  \TheTitle
}
\end{tabular}

\end{document}
```

↓ Input

This produces the image shown in Figure 12.6. You can download or view this document.

Exercise 31. A Pie Chart (datapie package)
Recall the sample `orders.csv` file. Create a pie chart that displays the values in the `quantity` column. Alternatively you can use the `orders` SQL table.

For the More Adventurous (SQL)
A pie chart showing just the `quantity` column isn't particularly informative. It would be more interesting to have a pie chart of the total quantities

Output

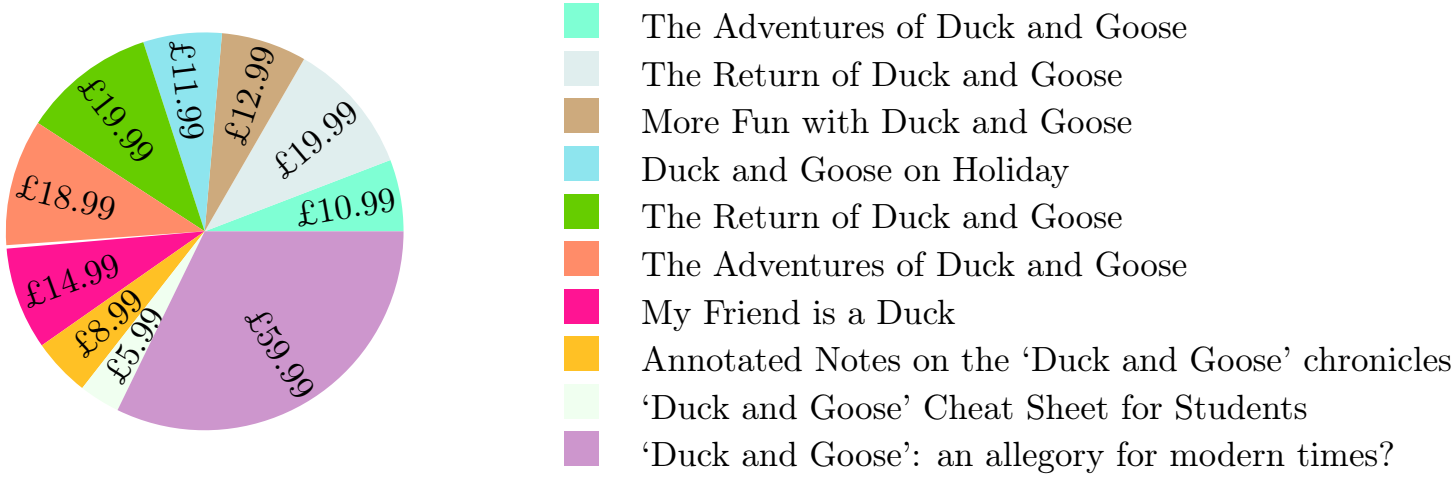


Figure 12.6 An Example Pie Chart with a Legend (datapie package)

ordered for each book title (rather than per order) accompanied by the title and format. This requires pulling data from multiple tables, which is far more efficiently done using SQL than using TEX. The SELECT statement is:

```
SELECT books.title AS booktitle, books.format AS bookformat,
SUM(orders.quantity) AS total FROM orders, books WHERE
orders.bookid=books.id GROUP BY orders.bookid
```

SQL

(If you want the data sorted in descending order of total quantities, you can append ORDER BY total DESC.)

Create a pie chart that shows the order totals for each book with a legend that shows the book title and format.

\DTLpiechart uses the tikzpicture environment and there are two hooks available to add additional picture drawing commands to that environment:

\DTLpieatbegintikz

Definition

This command is performed before the pie chart is drawn.

\DTLpieatendtikz

Definition

This command is performed after the pie chart is drawn.

Use one of these commands to add the legend as a node inside the tikzpicture environment. Hint: tikz allows radial coordinates specified in the

form (⟨*angle*⟩:⟨*radius*⟩). You can download or view the solution to this exercise.

12.2.2 The pgf-pie Package

The pgf-pie package [115] is available on MiKTEX, but not on TEX Live. For most of this book (and for the series, in general) I have chosen packages that are available on both distributions, but as there's so little choice in this topic, this section describes the pgf-pie package. If you are a TEX Live user, you will need to manually install the package as follows:

1. Download http://mirror.ctan.org/graphics/pgf/contrib/pgf-pie.zip
2. Unpack the `pgf-pie.zip` archive.
3. Copy `pgf-pie.sty` to somewhere on TEX's path.
4. Copy the manual `pgf-pie-manual.pdf` to somewhere on `texdoc`'s path.

For example, on Linux:

```
unzip pgf-pie.zip
mkdir -p ~/texmf/tex/latex/pgf-pie
cp pgf-pie/pgf-pie.sty ~/texmf/tex/latex/pgf-pie/
mkdir -p ~/texmf/doc/latex/pgf-pie
cp pgf-pie/pgf-pie-manual.pdf ~/texmf/doc/latex/pgf-pie/
```

Shell

You can use kpsewhich to check that the package has been successfully installed:

```
kpsewhich pgf-pie.sty
```

Shell

This should display the full path to `pgf-pie.sty` if the file is on TEX's path otherwise it displays nothing.

The pgf-pie package provides just one command:

`\pie[`⟨*options*⟩`]{`⟨*list*⟩`}`

Definition

which should be placed inside the tikzpicture environment. The ⟨*list*⟩ argument should be a comma-separated list of values in the form ⟨*number*⟩/⟨*text*⟩ where ⟨*number*⟩ is the value and ⟨*text*⟩ is the outer label for the segment.

The optional argument ⟨*options*⟩ is a key=value list where the following keys are available:

`pos` Sets the centre of the chart to the value {⟨*x*⟩,⟨*y*⟩}. The default is the origin.

`rotate` Rotates the chart by the given number of degrees.

`radius` Sets the radius of the chart. The default is 3.

`color` Sets the colours for each segment. The value should be a comma-separated list of colours corresponding to each segment or a single colour, which indicates the colour for the entire chart.

`explode` Offset the segments. The value may be a single number, in which case all segments are offset by that amount, or the value may be a comma-separated list of numbers where each value is the offset amount for the corresponding segment.

`sum` The sum of all the data. This can be calculated automatically if the `auto` option is set. The default is 100. If the `auto` option is off and the actual data sum is less that this value there will be a missing segment in the chart.

`auto`	A boolean key. If `true`, the sum of the data is calculated automatically.
`after number`	Indicates the text to place after the number shown in the segment. The default is `\%`.
`before number`	Indicates the text to place before the number shown in the segment.
`scale font`	A boolean key. If true, this scales the font used for the inner label according to the size of the segment, so large segments will have large inner labels and small segments will have small inner labels.
`text`	Indicates how to position the text (outer label). The value may be one of: `label` (place the text label outside the segment), `pin` (as `label` but also draws a line from the segment arc to the label), `inside` (place the text label inside the segment above the value) or `legend` (create a legend). The default is `label`.

There are some other options as well. See the manual [115] for further details.

Example 61. A Pie Chart (pgf-pie package)
This example has trivial labels A, ..., E and uses the optional argument to change some of the pie chart settings. Note that the pgf-pie package automatically loads tikz.

Input

```
\documentclass{article}

\usepackage{pgf-pie}

\begin{document}

\begin{tikzpicture}
\pie[explode={0,0.5,0,0.75,0},% offset segments 2 and 4
  color={yellow,cyan,green,pink,orange},% segment colour
  text=legend% create a legend
 ]% settings
 {5/A,10/B,15/C,30/D,40/E}% values and labels
\end{tikzpicture}

\end{document}
```

Input

This produces the chart shown in Figure 12.7.

12.3 Bar Charts

There are a number of bar chart packages listed on the genchart topic. Excluding the LaTeX2.09 entries, at the time of writing they are: bardiag [89], bchart [47] and pst-bar [72]. These three packages are available both on TeX Live and MiKTeX. Two of them, bardiag and pst-bar, use pstricks and the other, bchart, uses tikz. Additionally, the datatool bundle [94] provides the package databar, which can be used to display bar charts of data stored in datatool databases.

As mentioned in §10.4, the pstricks package uses PostScript code so it can't be used directly with PDFLaTeX, so this section will only cover packages that are driver-independent. Therefore §12.3.1 discusses the bchart package and, since this book has already introduced the datatool package, §12.3.2 discusses the databar package, which also uses the tikz package to draw the chart.

12.3.1 The bchart Package

The bchart package [47] provides the bchart environment

Output

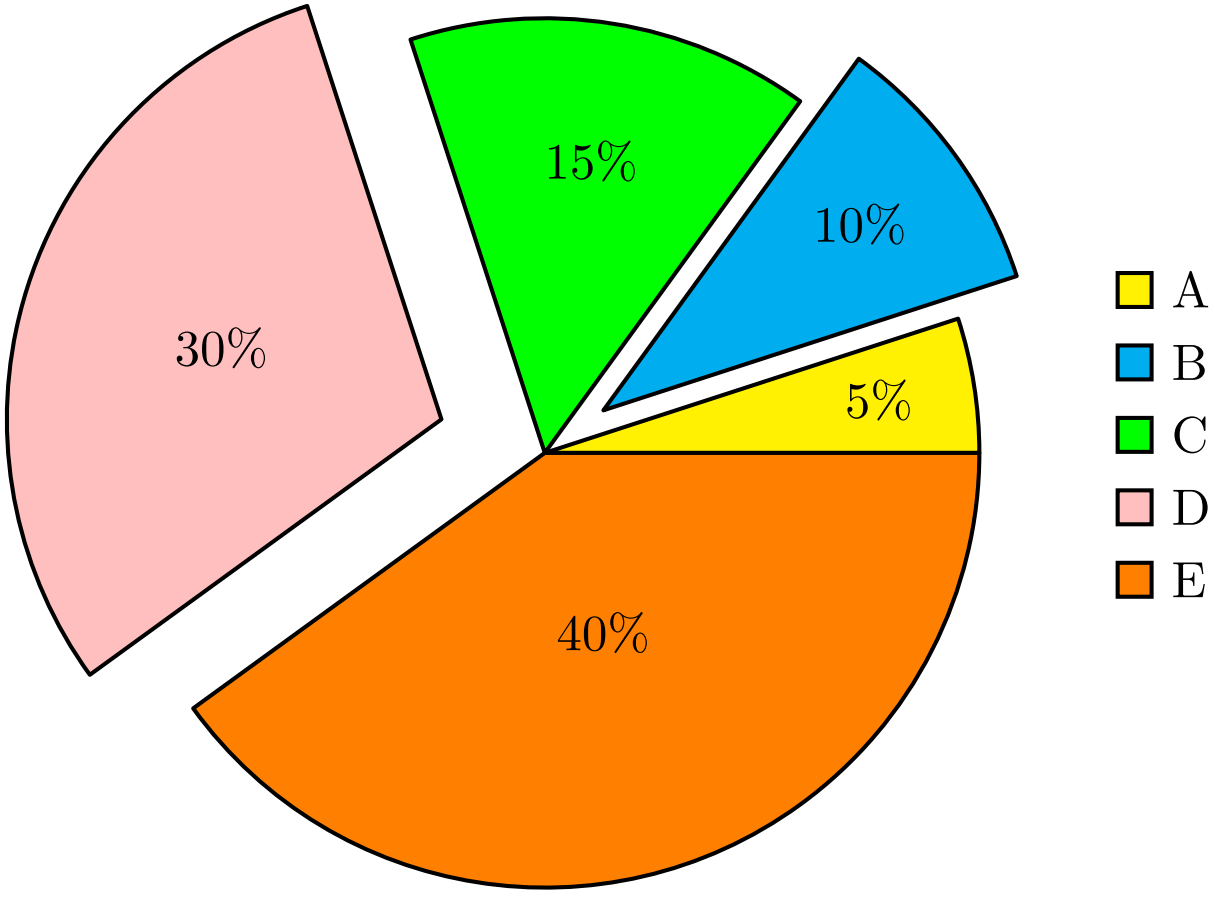


Figure 12.7 An Example Pie Chart (pgf-pie package)

\begin{bchart}[⟨*options*⟩]
⟨*bar drawing commands*⟩
\end{bchart}

Definition

The optional argument ⟨*options*⟩ is a key=value list where the following keys are available:

`max` The maximum x-axis value. (The default is 100.)

`min` The minimum x-axis value. (The default is 0.)

`step` The step size along the x-axis.

`steps` For irregular intervals or if rounding errors cause a problem for the `step` option, this option can be used instead. The value should be a comma-separated list of intervals along the x-axis.

`plain` Hides all the tick marks along the x-axis.

`unit` Specifies a unit to append to all the values displayed on the chart. The default is empty.

`width` Sets the width of the chart. The default is 8 cm. This setting doesn't change the height of the chart.

`scale` Scales both the width and height, maintaining the aspect ratio. (This doesn't change the text size.)

The font declaration used for the text displayed in the bar chart is given by:

`\bcfontstyle` Definition

This can be redefined using `\renewcommand` or set to empty to use the document font. The default definition is `\sf`, which is an obsolete font changing command and may cause issues with some classes such as the KOMA-Script classes. I suggest you redefine `\bcfontstyle` to use a modern declaration, such as `\sffamily`. For example:

```
\renewcommand*{\bcfontstyle}{\sffamily}
```
Input

Within the bchart environment, the bars are displayed using:

`\bcbar[`⟨*options*⟩`]{`⟨*number*⟩`}` Definition

where ⟨*number*⟩ is the bar's value. The optional argument ⟨*options*⟩ is a key=value list with the following keys:

`text` Sets the text displayed inside the bar (to the right of the *y*-axis).

`label` Sets the label displayed on the left of the *y*-axis.

`color` Sets the bar colour.

`plain` Hides the bar's value, which by default is displayed to the right of the bar.

`value` Sets the value displayed to the right of the bar. The default is the ⟨*number*⟩ in the mandatory argument of `\bcbar`.

You can insert vertical gaps between bars using:

`\bcskip[`⟨*options*⟩`]{`⟨*length*⟩`}` Definition

Within the scope of the `bchart` environment, the standard vertical skips `\smallskip`, `\medskip` and `\bigskip` are redefined in terms of `\bcskip` and may be used to insert small, medium or large gaps. As with `\bcskip`, these three commands also have an optional argument. This option only has one key available: `label`, which specifies a label.

A "free" label is placed to the left of the y-axis at the current location using:

`\bclabel{`⟨*text*⟩`}` Definition

where ⟨*text*⟩ is the label text. This doesn't add any gap or bar to the chart.

The x-axis can be labelled using:

\bcxlabel{⟨text⟩} Definition

Example:
Here's the data from the pie chart in Example 61 reproduced as a bar chart:

Input

```
\begin{bchart}[max=40,step=5]
\bcbar[label=A,color=yellow]{5}
\bcbar[label=B,color=cyan]{10}
\bcbar[label=C,color=green]{15}
\bcbar[label=D,color=pink]{30}
\bcbar[label=E,color=orange]{40}
\end{bchart}
```

Input

This produces the chart shown in Figure 12.8.

Exercise 32. A Bar Chart (bchart package)
Reproduce the bar chart shown in Figure 12.9. (The bar colours are the default setting.) Hint: if you actually try to enter values such as 75000 you are likely to get a "Dimension too large" error, so enter the numbers as one

Output

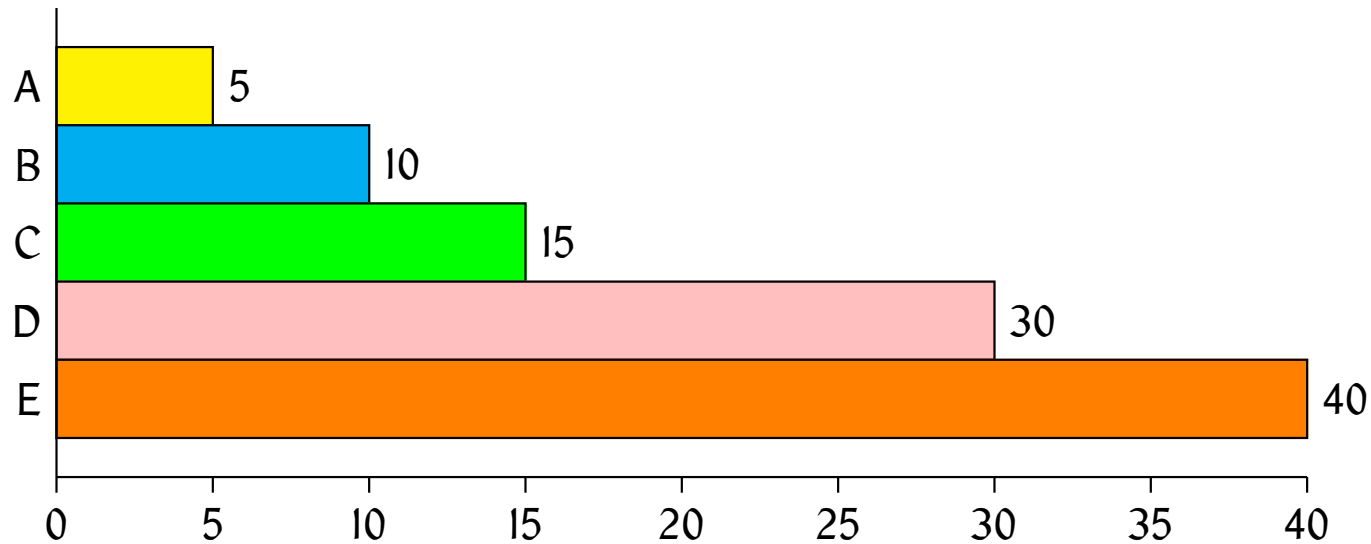


Figure 12.8 A Bar Chart (bchart package)

thousandth of the actual value and set the `unit` key so that the numbers are displayed as shown.

You can download or view the solution to this exercise.

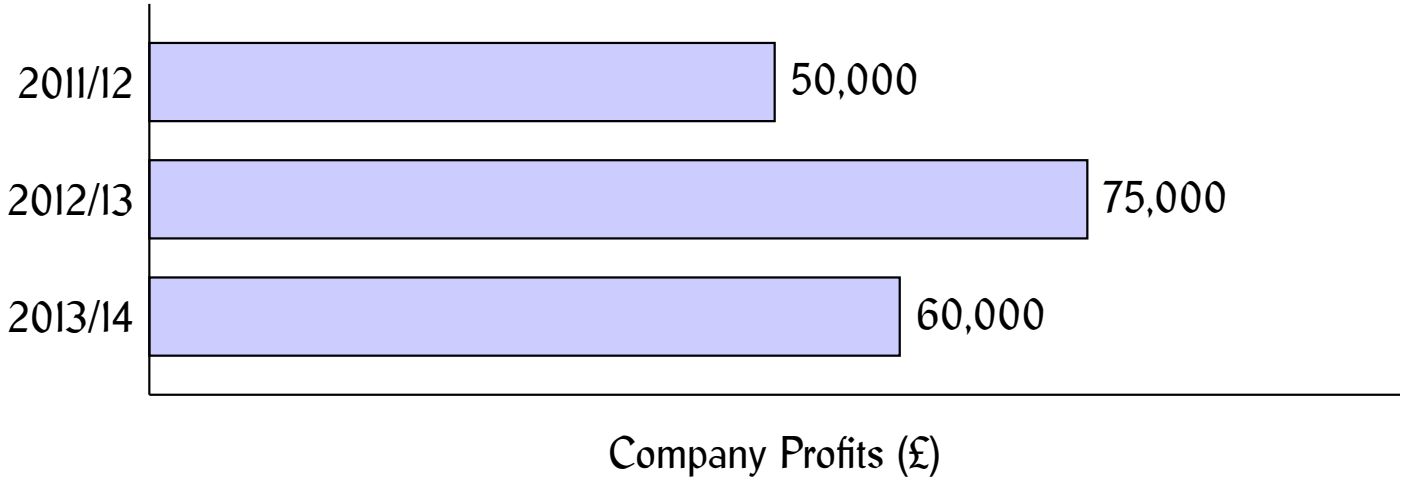


Figure 12.9 Bar Chart (bchart package) Exercise

12.3.2 The databar Package

The databar package is part of the datatool bundle [94] and can be used to generate bar charts from data stored in a datatool database. The bar charts

may be either vertical or horizontal. The default is vertical. The databar package has the following package options:

`color` Create coloured bar charts (default).

`gray` Create grey scale bar charts.

`vertical` Create vertical bar charts (default). The x-axis is the horizontal axis and the y-axis is the vertical axis.

`horizontal` Create horizontal bar charts. The x-axis is the vertical axis and the y-axis is the horizontal axis.

There are two commands provided to generate a bar chart:

`\DTLbarchart[`⟨*condition*⟩`]{`⟨*settings*⟩`}{`⟨*db-name*⟩`}{`⟨*assign list*⟩`}` Definition

and

`\DTLmultibarchart[`⟨*condition*⟩`]{`⟨*settings*⟩`}{`⟨*db-name*⟩`}{`⟨*assign list*⟩`}` Definition

The former generates a bar chart from a single column of data and the latter generates a bar chart with groups of bars representing multiple columns of data. The ⟨*condition*⟩, ⟨*db-name*⟩ and ⟨*assign list*⟩ are the same as for `\DTLforeach`. The ⟨*settings*⟩ argument is a key=value list where the following keys are available:

variable	This specifies the control sequence (which must be set in ⟨*assign list*⟩) that contains the value used to construct the bar chart. This key is required for \DTLbarchart and is unavailable for \DTLmultibarchart.
variables	This specifies a comma-separated list of control sequences (which must all be set in ⟨*assign list*⟩) that contain the values used to construct the bar chart. This key is required for \DTLmultibarchart and is unavailable for \DTLbarchart.
max	The maximum value on the *y*-axis. (A decimal number.)
length	The overall length of the *y*-axis. (A dimension.)
maxdepth	A zero or negative number (not a dimension) that specifies the maximum depth of the *y*-axis.
axes	This may take one of the following values: both (show both axes), x (only show the *x*-axis), y (only show the *y*-axis) or none (don't show either axes).

barlabel	Sets the lower bar label. When used with \DTLmultibarchart this indicates the group label.
multibarlabels	This value should be a comma-separated list of labels for each bar within a group for \DTLmultibarchart. This key is not available for \DTLbarchart.
upperbarlabel	The upper bar label. This key is not available for \DTLmultibarchart.
uppermultibarlabels	This value should be a comma-separated list of upper labels for each bar within a group for \DTLmultibarchart. This key is not available for \DTLbarchart.
yticpoints	A comma-separated list of tick locations for the y-axis. This setting overrides yticgap.
yticgap	Specifies the gap (a number not a dimension) between the y-tick marks.
yticlabels	A comma-separated list of tick labels for the y-axis.

ylabel	The y-axis label.
groupgap	The gap (a number in terms of the width of a bar) between groups when using \DTLmultibarchart. The default is 1, which indicates one bar width. This key is not available for \DTLbarchart.
verticalbars	A boolean key where true indicates a vertical bar chart and false indicates a horizontal bar chart.

In addition to the above settings, you can also change the appearance of the bar chart by changing any of the following before drawing the chart. Remember that you need to use \setlength to change the value of a length register. The y-tick labels are rounded to $\langle n \rangle$ digits after the decimal point, where $\langle n \rangle$ is given by the counter DTLbarroundvar.

\DTLbarchartlength Definition

This is a length register that stores the total length of the y-axis. The default is 3 in.

\DTLbarwidth Definition

This is a length register that stores the width of each bar. The default is 1 cm.

\DTLbarlabeloffset Definition

This is a length register that stores the distance from the x-axis to the lower bar label. The default is 10 pt.

\DTLsetbarcolor{⟨n⟩}{⟨colour⟩} Definition

This sets the ⟨n⟩th bar colour to ⟨colour⟩. Only the first eight bars have a colour defined by default. You need to use this command if you need more than eight bars or if you want to override the default colours.

\DTLdobarcolor{⟨n⟩} Definition

Sets the current text colour to the colour of the ⟨n⟩th bar.

\DTLbaroutlinecolor Definition

This macro expands to the colour of the bar outlines. This defaults to black. Use \renewcommand to change this value.

\DTLbaroutlinewidth Definition

A length register that stores the line width for the bar outlines. If it is set to 0 pt, the outline is not drawn. The default value is 0 pt.

Both \DTLbarchart and \DTLmultibarchart draw the chart inside a tikzpicture environment. You can redefine the following commands to insert code at the start or end of this environment:

\DTLbaratbegintikz Definition

for the hook at the start (after the unit vectors are set) and

\DTLbaratendtikz Definition

for the hook at the end.

There is also a hook for code to apply at every bar:

\DTLeverybarhook Definition

Within this book you can use \DTLstartpt (the start of the bar), \DTLmidpt (the mid point of the bar) and \DTLendpt (the end of the bar).

There are other commands as well that can be redefined to change the appearance of the bar chart. See the databar section of the datatool manual [94] for further details.

Example 62. An Example Bar Chart (databar package)

The pie chart from Example 60 can be reproduced as a bar chart as shown below. Since some of the book titles are quite long, the title has been placed

inside a \parbox to prevent the image from becoming overly tall and the bars are made wider by changing the value of \DTLbarwidth.

↑ Input

```
\documentclass{article}

\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\usepackage[x11names]{xcolor}
\usepackage{databar}

\DTLloaddb{books}{booklist.csv}

\DTLsetbarcolor{1}{Aquamarine1}
\DTLsetbarcolor{2}{Azure2}
\DTLsetbarcolor{3}{Burlywood3}
\DTLsetbarcolor{4}{CadetBlue2}
\DTLsetbarcolor{5}{Chartreuse3}
\DTLsetbarcolor{6}{Salmon1}
\DTLsetbarcolor{7}{DeepPink1}
\DTLsetbarcolor{8}{Goldenrod1}
```

```
\DTLsetbarcolor{9}{Honeydew1}
\DTLsetbarcolor{10}{Plum3}

\setlength{\DTLbaroutlinewidth}{1pt}
\setlength{\DTLbarwidth}{1.2cm}

\begin{document}

\setcounter{DTLbarroundvar}{2}

\DTLbarchart
 {variable=\ThePrice,% database column
  axes=both,% show both axes
  barlabel=\parbox{4cm}{\raggedright\TheTitle},% bar labels
  upperbarlabel={\pounds{ThePrice}},% upper bar labels
  yticgap=10,% gap between y tick marks
  ylabel={Price (\pounds)}% y-axis label
 }% settings
 {books}% database
 {\ThePrice=price,\TheTitle=title}% assignment list

\end{document}
```

Input

This produces the chart shown in Figure 12.10. You can download or view this example.

12.4 Gantt Charts

A Gantt chart (named after Henry Gantt) is a form of bar chart used to illustrate the work breakdown structure of a project. At the time of writing, the gantt topic has four entries, one of which is for ConTEXt. The remaining three are all LATEX packages available on both MiKTEX and TEX Live. The pgfgantt package [87] uses tikz, and the pst-gantt [26] and rtsched [53] packages use pstricks.

This section discusses the pgfgantt package, since it's driver-independent and uses the pgfcalendar package, which has already been introduced in §7.2. If you want to use the pgfgantt package, make sure you have an up-to-date version of the pgf package installed.

The pgfgantt package defines the ganttchart environment, which can be used to generate a Gantt chart.

\begin{ganttchart}[⟨*options*⟩]{⟨*start tss*⟩}{⟨*end tss*⟩}
⟨*entries*⟩
\end{ganttchart}

Definition

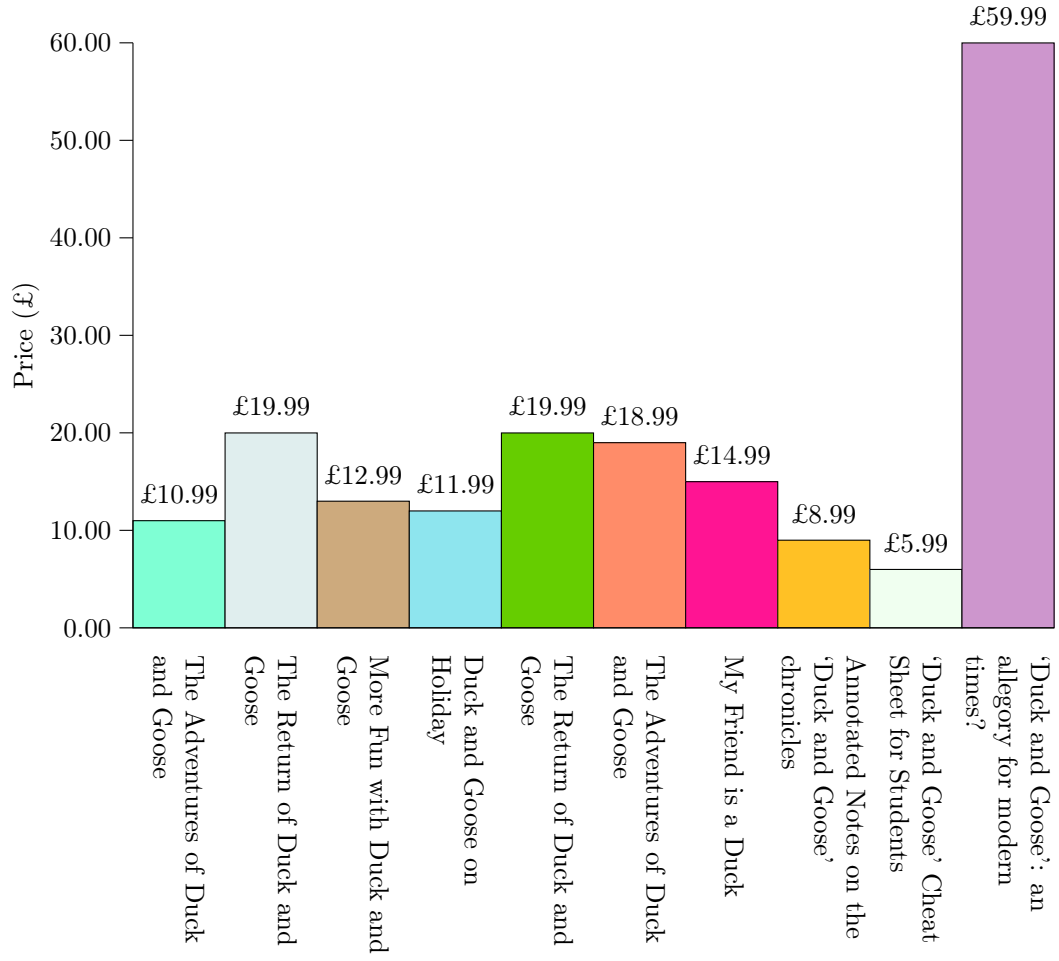


Figure 12.10 A Bar Chart (databar package)

The optional argument ⟨*options*⟩ is a key=value list. You can also use these keys in the optional argument of the commands, described below, provided for use within the ganttchart environment to apply the setting to just that element. Alternatively you can use

`\ganttset{`⟨*options*⟩`}` Definition

to apply the settings to the current scope.

For the mandatory ganttchart environment arguments, ⟨*start tss*⟩ is the starting time slot specifier and ⟨*end tss*⟩ is the end time slot specifier. The Gantt chart consists of several lines, which may contain title elements or chart elements. Each unit on the chart's x-axis represents a time slot.

Note that none of the elements start a new row. Instead you need to explicitly insert a line break using:

`\ganttnewline[`⟨*style*⟩`]` Definition

where ⟨*style*⟩ is a key=value list of options permitted by tikz's `\path` optional argument (such as `draw=red`). The default starts a new row without drawing anything.

A time slot specifier (tss) represents a time slot along the horizontal axis. The format used to specify a time slot is determined by the value of the `time slot format` key (which may be set in ⟨*options*⟩). This key may take one of the following values:

`simple`	Each time slot is specified as a positive integer, where `1` indicates the value of the `time slot format/start date` key (see below), `2` indicates the day after `time slot format/start date` etc.
`isodate`	Each time slot is specified using the ISO-standard format ⟨*yyyy*⟩-⟨*mm*⟩-⟨*dd*⟩. (The leading zero is optional for the month or day.)
`isodate-yearmonth`	As `isodate` but without the day -⟨*dd*⟩ part. The date is assumed to be the first day of the month.
`little-endian`	Each time slot is specified in ⟨*dd*⟩-⟨*mm*⟩-⟨*year*⟩ format. (You may also use a slash (`/`) or period (`.`) instead of a hyphen (`-`) for the separator.) A two-digit year will be completed according to the `base century` setting.
`middle-endian`	As `little-endian` but with the ⟨*dd*⟩ and ⟨*mm*⟩ swapped.
`big-endian`	As `little-endian` but with the order reversed (⟨*yyyy*⟩ first and ⟨*dd*⟩ last).

Other keys that may be used with ⟨*options*⟩ include:

`time slot format/base century`	Sets the century for the auto-completion of two-digit years. The default is 2000.
`time slot format/start date`	An ISO-standard date ⟨*yyyy*⟩-⟨*mm*⟩-⟨*dd*⟩ denoting the value of the time slot 1 when using the `simple` format. The default is `2000-01-01`, so the time slot 1 indicates 2000-01-01, the time slot 2 indicates 2000-01-02, etc.
`canvas/.style`	This sets the canvas style. The value should be an option list valid for the optional argument of a tikz node. (See §7.5 and §12.1.) The default is `{shape=rectangle,draw,fill=white}`, which creates a white canvas with a rectangular frame.
`newline shortcut`	This is a boolean key. If `true` this will allow you to use `\\` as a shortcut for `\ganttnewline`.
`compress calendar`	This is a boolean key. If `true`, one month

corresponds to one time slot, otherwise one day corresponds to one time slot.

There are many other options that govern the chart's formatting, see the pgfgantt manual [87] for further details.

Within the ganttchart environment, you can use the commands described below to create chart elements. In each case, ⟨*options*⟩ is as for the ganttchart environment.

`\ganttitle[`⟨*options*⟩`]{`⟨*text*⟩`}{`⟨*n*⟩`}` Definition

Draws a single title element that covers ⟨*n*⟩ time slots, with the given ⟨*text*⟩.

`\ganttitlelist[`⟨*options*⟩`]{`⟨*list*⟩`}{`⟨*n*⟩`}` Definition

This iterates over ⟨*list*⟩ and draws a title element that spans ⟨*n*⟩ time slots for each item in the list. The title text is given by the current list element. The list should be in the format accepted by `\foreach` (see §2.7.2).

`\ganttitlecalendar[`⟨*options*⟩`]{`⟨*calendar lines*⟩`}` Definition

This prints a title calendar that spans the whole chart. The starred form

`\ganttitlecalendar*[`⟨*options*⟩`]{`⟨*start tss*⟩`}{`⟨*end tss*⟩`}{`⟨*calendar lines*⟩`}` Definition

spans from ⟨*start tss*⟩ to ⟨*end tss*⟩. In both cases, ⟨*calendar lines*⟩ is a comma-separated list of line types:

`year` Print the year.

`month` Print the month number. This may optionally be followed by =⟨*format*⟩ where ⟨*format*⟩ may be one of: `name` (the full name) or `shortname` (abbreviated name).

`week` The week. This may optionally be followed by =⟨*number*⟩ where ⟨*number*⟩ is the number of the first week of the calendar.

`weekday` The week day number (starting from 0 for Monday). This may optionally be followed by =⟨*format*⟩ (analogous to `month`).

`day` The two-digit day of the month.

There are three chart elements, which can be created using the following:

`\ganttbar[`⟨*options*⟩`]{`⟨*text*⟩`}{`⟨*start tss*⟩`}{`⟨*end tss*⟩`}` Definition

This creates a bar indicating the duration of a task or subtask.

\ganttgroup[⟨options⟩]{⟨text⟩}{⟨start tss⟩}{⟨end tss⟩} Definition

This combines several subtasks into a single task. For the above two commands, ⟨*start tss*⟩ indicates the starting time slot and ⟨*end tss*⟩ indicates the end time slot for the task or task group.

\ganttmilestone[⟨options⟩]{⟨text⟩}{⟨tss⟩} Definition

This indicates that a milestone has been completed on the time slot given by ⟨*tss*⟩.

If you want to have links between each of these elements (that is, lines drawn from the end of one element to the start of the next element) you can use one of the following commands, analogous to the above three commands.

\ganttlinkedbar[⟨options⟩]{⟨text⟩}{⟨start tss⟩}{⟨end tss⟩} Definition

for an individual task or subtask,

\ganttlinkedgroup[⟨options⟩]{⟨text⟩}{⟨start tss⟩}{⟨end tss⟩} Definition

for a group, or

\ganttlinkedmilestone[⟨options⟩]{⟨text⟩}{⟨tss⟩} Definition

for a milestone.

Example 63. A Gantt Chart

This example is for a Gantt chart that spans a whole year. In this case a day as the time slot would produce a chart that's far too wide, so I've used the `compress calendar` option and I set the `x unit` to 8 mm. The pgfgantt manual recommends an x/y ratio of approximately $1:2$ so I've also set both `y unit title` and `y unit chart` to 16 mm. The `newline shortcut` option allows me to use `\\` instead of `\ganttnewline`.

↑ Input

```
\documentclass{article}

\usepackage{pgfgantt}

\begin{document}

\begin{ganttchart}
 [time slot format=isodate,%
  newline shortcut,%
  x unit=8mm,%
  y unit title=16mm,%
  y unit chart=16mm,%
```

```
  compress calendar%
  ]% options
 {2014-1-1}% start time slot
 {2014-12-31}% end time slot
\ganttitlecalendar{year,month=shortname}\\
\ganttgroup{Ray Gun Project}{2014-1-1}{2014-06-30}\\
\ganttbar{Design}{2014-1-1}{2014-04-15}\\
\ganttbar{Testing}{2014-04-01}{2014-06-30}\\
\ganttmilestone{Prototype Ready}{2014-07-01}\\
\ganttgroup{Telepathic Cakes}{2014-5-10}{2014-12-31}\\
\ganttbar{Development}{2014-5-10}{2014-12-31}
\end{ganttchart}

\end{document}
```

↓ Input

This produces the chart shown in Figure 12.11. You can download or view this example.

Output

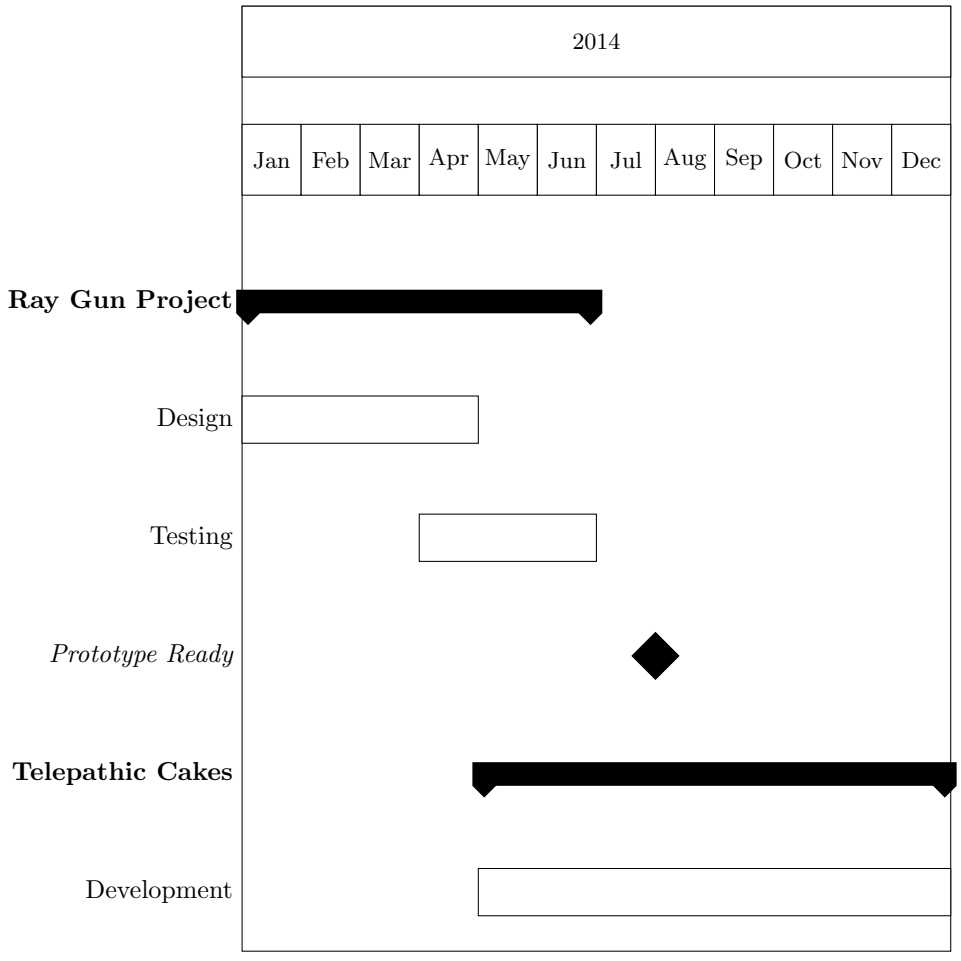


Figure 12.11 A Gantt Chart

12.5 Plots

If you have a large amount of data, you may want to consider using a mathematical tool (such as Matlab, Octave or GnuPlot) to generate your graphs as image files. If you use TEX, you may have excessively long document build times. Don't expect TEX to be able to compute logarithms or exponentials with the speed or precision of a custom-built numerical computing engine. Additionally, some packages can't parse scientific notation.

There are, however, some packages that use PostScript rather than TEX to perform the calculations (in which case you can't directly use PDFLATEX) and some of them rely on TEX's shell escape to call an application, such as GnuPlot, to generate the drawing code (in which case you need the shell escape enabled). If you have applications such as GnuPlot or Matlab installed or if you are happy to use the `latex+dvips+ps2pdf` route to generate your PDF files, there are a number of useful packages listed on the graphics-plot topic. In addition, pgf/tikz comes with an impressive mathematical engine provided by the pgfmath package. If you want to annotate your plot with text that matches your document format and have the patience to wait for your document to compile, you may be surprised by the images that can be produced using some of the packages available on CTAN.

Since the aim of this book is to be as useful to as many readers as possible, I'm again going to choose device-independent options that don't

require any additional applications. (This is, after all, a book on administrative work not high performance computing or advanced mathematics.) As with pie charts and bar charts, the datatool package also provides a package (dataplot) for drawing plots from data stored in a database. However, if you don't intend using that data anywhere else in the document, I suggest you use a more flexible package such as pgfplots, which is described below. The other advantage of pgfplots over dataplot is that pgfplots can parse scientific notation and can cope with larger values. The remainder of this section discusses the pgfplots package.

The pgfplots package [25] uses pgf/tikz so make sure you have an up-to-date version of the pgf package installed. The pgfplots manual [25] is 500 pages long at the time of writing, so this section is a very brief introduction. See the user manual for further details.

Options can be set using

`\pgfplotsset{`⟨*options*⟩`}` Definition

where ⟨*options*⟩ is a key=value list of options. The pgfplots manual recommends setting the `compat` key to benefit from recent features and to avoid possible changes if you recompile your document at a later date with a newer version. The log file will provide a suggested value.

Other common options include:

`width` Sets the width of the final picture. An empty values

	indicates the default width or rescale in proportion to the height (so that the aspect ratio is maintained if you have set the height).
`height`	As above but for the height.
`scale only axis`	This is a boolean key. If `true`, the above `width` or `height` settings apply to the size of the axes rather than the overall picture size (which may include axis labels or tick marks).
`xlabel`	The x-axis label.
`ylabel`	The y-axis label.
`title`	The plot title.
`major tick length`	The length of major tick marks.
`minor tick length`	The length of minor tick marks.
`tick align`	The value may be one of: `inside` (default), `outside` or `center`. This indicates the location of the tick marks relative to the axis line.

There are many different types of plots and axes, but this introduction will just consider normal two-dimensional Cartesian plots, which can be produced using the axis environment:

```
\begin{axis}[options]
⟨plot commands⟩
\end{axis}
```

Definition

This should be placed inside a tikzpicture environment. The contents of the axis environment may contain one or more instance of

`\addplot[`⟨*plot options*⟩`]` ⟨*plot specs*⟩;

Definition

There are a number of different ways of specifying the plot, such as:

`\addplot[`⟨*plot options*⟩`]` `coordinates {`⟨*co-ordinate list*⟩`}` ⟨*trailing path specs*⟩;

Definition

where ⟨*co-ordinate list*⟩ is a space-separated list of co-ordinates. A standard Cartesian co-ordinate can be specified in the form (⟨x⟩,⟨y⟩), for example, (0.5,3.1). The ⟨*trailing path specs*⟩ are optional and are as the path specifications used by tikz's `\path` and `\draw` commands.

`\addplot[`⟨*plot options*⟩`]` `table [column selection] {`⟨*file or inline table*⟩`}` ⟨*trailing path specs*⟩;

Definition

where ⟨*file or inline table*⟩ is either a filename or the data arranged in

the tabulated form of a space-separated data file. The ⟨*column selection*⟩ may contain the keys `x`=⟨*column key*⟩, to indicate the column containing the x values, and `y`=⟨*column key*⟩, to indicate the column containing the y values. The table data should include a header row at the start that may be used to reference the columns. If there is no header row, you can use `header=false` within ⟨*column selection*⟩ and use `x index`=⟨*index*⟩ and `y index`=⟨*index*⟩ instead of `x` and `y` to reference the required columns. The ⟨*index*⟩ should be an integer starting from 0 (the first column).

There are other `\addplot` specifications for evaluating expressions or using TEX's shell escape mechanism. See the pgfplots manual [25] for further details.

The ⟨*plot options*⟩ can be any of the path drawing options that are accepted by `\path` (or `\draw`), such as:

`mark` Sets the marker style. There are three standard markers: `*` (filled circle), `x` (cross) and `+` (plus). Additional markers are available with tikz's `plotmarks` library (which can be loaded using `\usetikzlibrary`).

`no markers` Overrides any `mark` value.

`mark repeat` The value of this key should be an integer ⟨n⟩, to indicate that only every ⟨n⟩th mark should be drawn.

`mark size`	The size of the markers.
`dashed`	Dashed line style.
`dotted`	Dotted line style.
`dashdotted`	Dash-dot line style.
`thin`	Thin line width.
`thick`	Thick line width.
`color`	The value should a colour name used for stroking and filling. You can omit the `color=` and just write the colour name.

See the pgf manual [101] for further details.

Example 64. A Sample Plot (pgfplots package)
Suppose instead of the bar chart in Exercise 32 I want a graph of company profits where the x-axis represents the year and the y-axis represents the profit. I could have a file containing that data called, say, `profits.dat` that contains:

```
year profit
2010 52000
2011 50000
2012 75000
2013 60000
```

In which case, I can plot the data using:

```
\addplot table {profits.dat};
```

Input

or I can just have the data inline as in the example document below:

↑ Input

```
\documentclass{article}
\usepackage{pgfplots}
\begin{document}

\begin{tikzpicture}
\begin{axis}
\addplot table
{
  year profit
  2010 52000
```

```
  2011 50000
  2012 75000
  2013 60000
};
\end{axis}
\end{tikzpicture}
\end{document}
```

Input

This produces the following message in the transcript:

```
Package pgfplots Warning: running in backwards compatibility mode
(unsuitable tick labels; missing features). Consider writing
\pgfplotsset{compat=1.11} into your preamble.
```

So I need to add

```
\pgfplotsset{compat=1.11}
```

Input

to the preamble. This produces the graph shown in Figure 12.12, which doesn't look right. The x-axis is far too cramped and doesn't need so many tick marks.

The positions of the x-axis tick marks can be changed using the `xtick` key. This may be set to `\empty` (generate automatically), the keyword `data`

Output

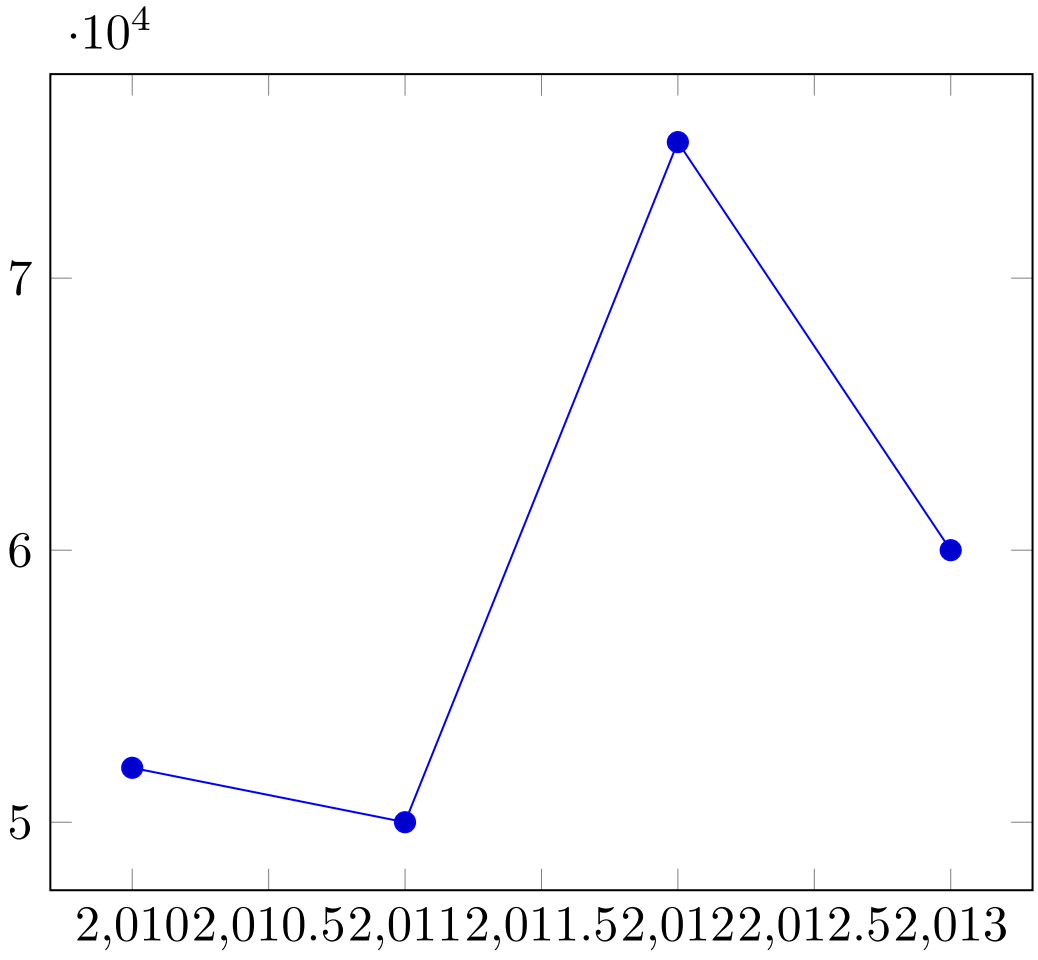


Figure 12.12 A Plot (First Attempt)

(use the co-ordinates provided by the first plot), or a comma-separated list of co-ordinates. (There are also analogous `ytick` and `ztick` keys.)

The tick labels can be changed using various keys. First is the `xticklabels` key which takes a comma-separated list of labels, where each label corresponds to a tick position. Alternatively, you can use `xticklabel` where the value is the code to produce the label. You can access information about the current tick using the following:

`\tick` Definition

This is the current element of the tick option.

`\ticknum` Definition

This command is only valid if the `x tick label as interval` option has been set (or `y tick label as interval` for the y-axis ticks) in which case it will contain the position of the next tick position.

The default label format is `\pgfmathprintnumber{\tick}`, which uses the number pretty-printing command provided by the pgfmath package. This is why the x-tick labels in Figure 12.12 have commas in them (for example, 2,010 rather than 2010). The comma can be removed by first using:

```
\pgfkeys{/pgf/number format/set thousands separator={}}
```
Input

Alternatively, if you just want to change the style for a particular axis:

Input

```
\pgfplotsset{x tick label style={
  /pgf/number format/set thousands separator={}
}}
```

Input

The modified document is shown below. I've also added a plot title using `title` and axes labels using `xlabel` and `ylabel`:

Input

```
\documentclass{article}

\usepackage{pgfplots}

\pgfplotsset{compat=1.11}

\begin{document}

\begin{tikzpicture}
\begin{axis}[xtick=data,% get x tick marks from data
```

```
 xlabel=Year,ylabel={Profits (\pounds)},% axes labels
 x tick label style={% change x tick label style
  /pgf/number format/set thousands separator={}%
 },
 title={Profits Since 2010},% plot title
 tick align=outside% ticks on the outside
 ]
\addplot table
{
  year profit
  2010 52000
  2011 50000
  2012 75000
  2013 60000
};
\end{axis}
\end{tikzpicture}

\end{document}
```

↓ Input

This produces the plot shown in Figure 12.13, which is much less cramped.

Output

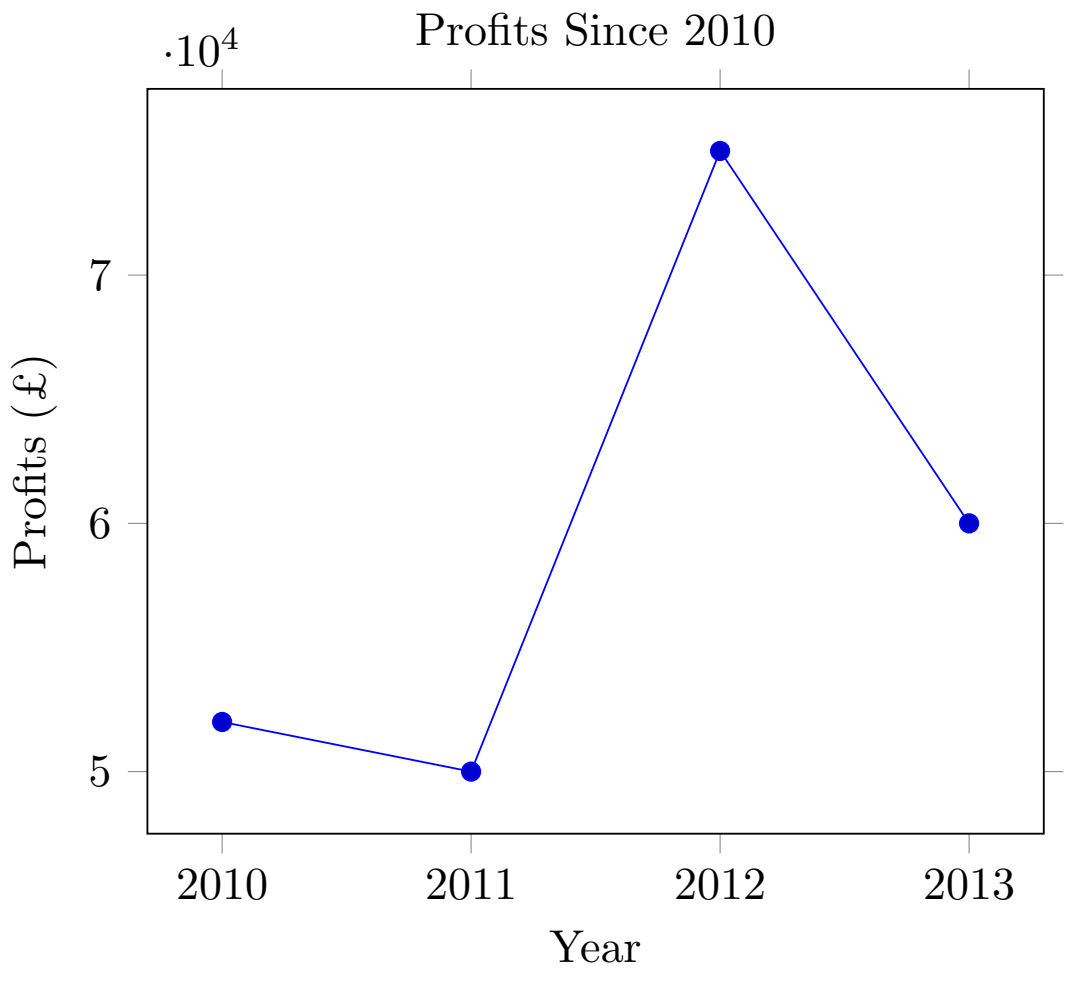


Figure 12.13 A Plot (Second Attempt)

In order to save space, the y-tick labels have been scaled by 10^4 but since, in this case, the y axis represents profits it would look better if this scaling wasn't applied. There are a number of different ways of changing the scaling (see the manual [25]) but to switch it off for all axes, you just need the option `scaled ticks=false`. It's also possible that you don't like the boxed axes, but this can be changed using the option `axis lines*=left`. These extra options produce the plot shown in Figure 12.14.

You can download or view this example where I have additionally loaded the `plotmarks` library and used the optional argument of `\addplot` to set the plot marks to filled diamonds (`mark=diamond*`) with the marker size set to 5 pt (`mark size=5pt`) and a thick cyan line stroke. (Try it for yourself before you download the example, as an additional exercise.)

Output

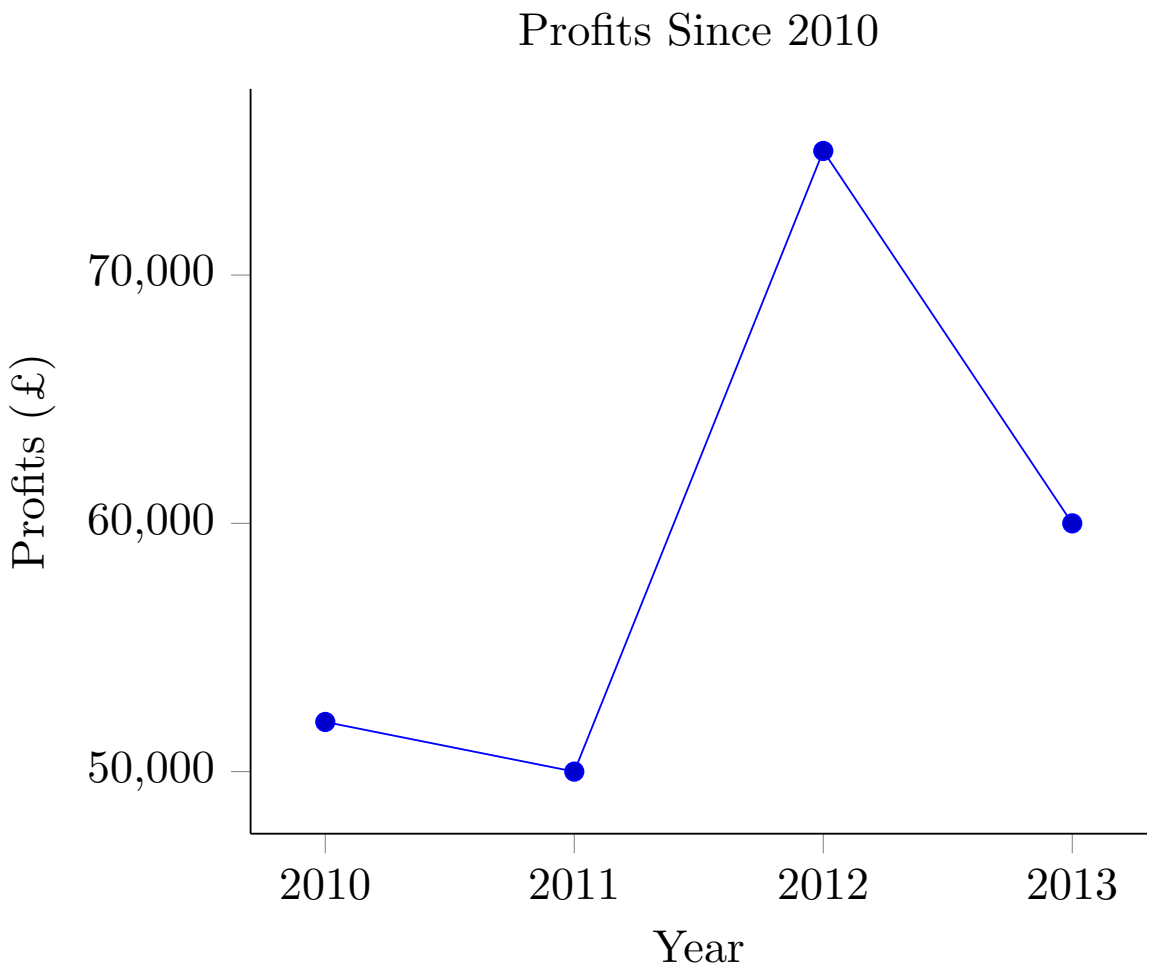


Figure 12.14 A Plot (Final Attempt)

13. Collaborating on Documents

There are a number of issues that can occur when multiple authors collaborate on the same document. For example, suppose Fred and Mabel are co-authors of a document, here are some problems they may encounter:

1. If both Fred and Mabel edit their own copies of the source code, how do they merge their edits? If Fred sends Mabel his updated source file, it could overwrite her changes. Similarly if Mabel sends Fred her updated source file, it could overwrite his changes.

2. Fred decides to use package foo, which is available on his MiKTEX distribution, but Mabel doesn't have foo on her TEX Live distribution.

3. Both Fred and Mabel want to use package baz but Mabel has a newer version and is using commands that aren't available in the older version that Fred has installed.

4. Fred and Mabel are working on a sensitive document. If they simply email each other the source code it could be intercepted or if

they transfer it on a portable device, such as a memory stick, it could get stolen.

5. Fred is the principle author and is in charge of writing the first draft. Once he has finished, he sends the document to Mabel who makes minor corrections, but Fred needs to know what modifications she has made.

6. Prof Important Person has decided he's also going to contribute to the document but insists on using his favourite word processor.

7. If Fred or Mabel make an inappropriate change to the document, they may want to roll back to an earlier version of the document to undo the modifications.

The more co-authors on the document, the more likely these problems will occur. The last case is an issue that may also need addressing for a single-authored document. This chapter describes ways of circumventing these problems, but the most convenient solution may not necessarily be the most appropriate solution if you have to take into account factors such as security or available resources or recalcitrant co-authors.

Here are some possible solutions to the above problems:

1. The easiest case is when Fred and Mabel are working on separate sections or chapters. They can then split up the source code and use `\input` (for sections) or `\include` (for complete chapters). Then Fred can just edit the files for his sections and Mabel can just edit the files for her sections. This way they won't be editing the same file at the same time.

   In the other case, where Fred and Mabel are both working on the same sections, then the most convenient solutions are a version control system (§13.2) or using an online LaTeX editor (§13.3).

2. Fred and Mabel agree to only use packages that are available on both MiKTeX and TeX Live, or Mabel manually installs the missing package, or they use an online LaTeX editor.

3. Fred and Mabel both update their TeX distributions before they start working on the document or they use an online LaTeX editor.

4. In this case, it depends on the level of security needed. Fred and Mabel may simply be able to use a password-protected version control system. Alternatively, they may need to store the document source code on a portable device that's locked in a safe when the document isn't being edited, and the authors will have to take it in turns to access the device in a non-networked secure environment.

5. Use LaTeX commands to markup the changes (§13.1) or use version control.

6. A non-LaTeX co-author can cause major headaches for a predominantly LaTeX team of authors. In some cases, gentle persuasion may help. If arguments about the quality of typesetting, the logic of document markup, the convenience of cross-referencing and automatically generated table of contents and similar lists don't have an effect, it might help to point out the security issues associated with Word files, such as the automatic hidden inclusion of personal data and revision information [27].

   If persuasion doesn't help, how much of a contribution will they be making to the document? If it's just a matter of a few minor corrections, then it might be possible for them just to mark their changes and for you to then incorporate those changes into the LaTeX source code. The best way of creating a Word document from the LaTeX source is to use `tex4ht` to create an OpenDocument Format (`.odt`) file which can then be converted to Word using an application such as LibreOffice. For example, if the LaTeX source is in a file called `myDoc.tex` then you can create `myDoc.odt` using:

```
mk4ht oolatex myDoc
```

Shell

7. Use a version control system.

Possible factors that may affect your choice:

Security There are different levels of document sensitivity, for example, information about a surprise event (we're organising Gran's eightieth birthday party, but don't let on); information about a new product that's patent-pending or about to be released with a big media splash; examination papers; information about your country's defence system including launch codes.

Locking up Gran's birthday invitations in a safe within a high-security complex complete with armed guards might be considered overkill. For medium level security, it may be sufficient to have a password-protected version control server located inside your firewall. Servers outside your firewall (including those used by online LaTeX editors) may be sufficient for low to medium security documents, but when you use a third-party system you have to put your trust in their security measures. When considering your options, you need to determine the detriment associated with any security breach. (See also §2.3.)

Cost Some of the solutions aren't free. This may impact your choice if you have a low budget.

Operating Systems While some solutions are cross-platform, others might not be. For example, the document build may include running an application or script that only works on a certain operating system. In this case, if possible, it may be better to switch to a cross-platform application (such as one that uses the Java virtual environment) or scripting language (such as Lua, Perl or Python). You may not like the cross-platform alternative, but at least all the authors can build the document.

One or more of the co-authors may have an ancient operating system that doesn't support the latest TEX distributions, or they may not have administrator privileges required to replace an obsolete TEX distribution that was installed a decade earlier. The online LATEX editors may help with some of these issues, but for security reasons they only allow a limited set of trusted applications (such as `makeindex` or `bibtex`) to run.

To a large extent this issue comes back to cost. The operating system may not be upgraded because the hardware is too old. The IT support may be underfunded and lack the resources for system-wide upgrades. In the case of additional applications to help with the document build, a cross-platform alternative may need to be purchased, if available, or a developer hired to create it.

Recalcitrant Authors Compromises have to be made on any project that involves more than one person. If one or more authors are determined to use a particular set-up or format or application that's incompatible with or inaccessible for the rest of the team then the chances are that the entire project will fail.

The remainder of this chapter is structured as follows:

- §13.1 discusses the changes package to markup changes within the document. This markup can be viewed in the PDF file or can be suppressed for the final version. The changes package can be used with LaTeX documents located on shared folders, under version control, or on an online server.

- §13.2 discusses how to access revision information for LaTeX documents under version control. Documents under version control can be temporarily locked to prevent conflicting edits, each author can use their favourite text editor to edit the document, and documents can be rolled back to earlier versions. However, all authors must make sure they all have the required class and packages installed.

- §13.3 discusses online LaTeX editors. Documents on an online LaTeX editor site can be edited by multiple authors, and the authors don't

need to worry about installing TEX locally, but they have to use the web interface and may not have access to all LATEX tools.

13.1 Change Markup

There are many entries listed on the editorial topic that allow you to mark changes in a document. Some of these are for LATEX2.09, Plain TEX, ConTEXt or LuaTEX, but even discounting these, there are still too many to describe, so I'm just going to describe the changes package [43], which is available on both TEX Live and MiKTEX. The changes package provides a way for the user to manually markup changes, such as additions, deletions or replacements.

The changes package provides an anonymous author that's used by default, but if you want to track the changes according to a particular author, you need to define each tracked author using:

`\definechangesauthor[`⟨*options*⟩`]{`⟨*id*⟩`}` Definition

where ⟨*options*⟩ is a key=value list of options and ⟨*id*⟩ is a label identifying the author. Available options:

`name` The author's name.

`color` The colour to use when marking up this author's changes. (Defaults to black.)

Example:

```
\definechangesauthor[name={Mabel Canary},color=cyan]{MC}
```
Input

To markup changes you can use:

`\added[`⟨*options*⟩`]{`⟨*text*⟩`}`
Definition

to indicate that ⟨*text*⟩ has been added,

`\deleted[`⟨*options*⟩`]{`⟨*text*⟩`}`
Definition

to indicate that ⟨*text*⟩ has been deleted, and

`\replaced[`⟨*options*⟩`]{`⟨*new text*⟩`}{`⟨*old text*⟩`}`
Definition

to indicate that ⟨*old text*⟩ has been replaced by ⟨*new text*⟩. Each of these commands has an optional argument that's a key=value list where the following keys are available:

`id` The author's ID (as provided in `\definechangesauthor`).

`remark` A remark about the change.

You can create a list or summary of the changes using

`\listofchanges[`⟨*options*⟩`]` Definition

The optional argument is again a key=value list, but there is currently only one key available: `style`, which may have the value `list` or `summary`. The default value is `list`. As with commands such as `\tableofcontents` and `\listoffigures`, this command requires two LaTeX runs to produce the list. The default extension for the summary of changes file is `.soc`, but this can be changed using

`\setsocextension{`⟨*extension*⟩`}` Definition

You can suppress the change markup using the `final` package option:

```
\usepackage[final]{changes}
```
Input

The `draft` option enables the markup. The other package options are ⟨*key*⟩=⟨*value*⟩ options, where the following keys are available:

- `markup`=⟨*value*⟩ sets the markup style. The ⟨*value*⟩ may be one of:

  `default` Colour markup for added text, strike out for deleted text.

  `underlined` Added text is underlined, deleted text is struck out.

`bfit`	Bold for added text and italic for deleted text.
`nocolor`	As `underlined` but without colour.

- `addedmarkup=`⟨*value*⟩ and `deletedmarkup=`⟨*value*⟩ set the markup style for added and deleted text. The ⟨*value*⟩ may be one of:

`none`	No markup.
`uline`	Underlined text (example).
`uuline`	Double underlined text (example).
`uwave`	Wavy underlined text (example).
`dashuline`	Dashed underlined text (example).
`dotuline`	Dotted underlined text (example).
`sout`	Struck out text (~~example~~).
`xout`	Crossed out text (*example*).
`bf`	Bold text.
`it`	Italic text.
`sl`	Slanted text.
`em`	Emphasized text.

- `authormarkup`=⟨*value*⟩ sets the style of the author identification. The ⟨*value*⟩ may be one of:

`superscript`	Superscript author's name or ID.
`subscript`	Subscript author's name or ID.
`brackets`	Author's name or ID in parentheses.
`footnote`	Author's name or ID in a footnote.
`none`	No author identification.

- `authormarkupposition`=⟨*value*⟩ sets which side of the change to place the author's ID. The ⟨*value*⟩ may be `right` or `left`.

- `authormarkuptext`=⟨*value*⟩ sets whether to use the author's ID or name in the change markup. The ⟨*value*⟩ may be either `id` (default) or `name`.

There are also commands to provide custom markup. See the changes documentation [43] for further details.

The changes package automatically loads the ulem [2] and xcolor packages. You can pass options to these packages using the `ulem` and `xcolor` keys. For example

```
\usepackage[ulem={normalem,normalbf}]{changes}
```
Input

The changes package also provides

```
\textsubscript{⟨text⟩}
```
Input

which is analogous to LaTeX's \textsuperscript kernel command.

⚠ You can't markup floats, such as figures or tables. Paragraph breaks within the markup text can also cause a problem. See the changes documentation [43] for further details.

Example 65. Recording Changes (changes package)
This example document has been edited by three authors: Mabel Canary, Fred Canary and Prof Important Person:

↑ Input

```
\documentclass[12pt]{article}

\usepackage[ulem={normalem,normalbf}]{changes}

\definechangesauthor[name={Mabel Canary},color=violet]{MC}
\definechangesauthor[name={Fred Canary},color=blue]{FC}
\definechangesauthor[name={Prof Important Person},color=teal]{IP}
```

```
\begin{document}
\section{About the Lab}

The Secret Lab of Experimental Stuff is a
\replaced[id=MC]{top-secret}{sinister} laboratory
whose existence is highly classified so don't tell
anyone about it \added[id=FC]{or we'll get really cross with you}.

The \added[id=IP]{world-renown} University of Somewhere denies all
knowledge of the Secret Lab of Experimental Stuff, except on Open
Days where members of the public may visit the facility and ask
questions as long as they consent to a memory wipe when they leave.
The memory wipe is \deleted[id=MC,remark={no it isn't}]{completely}
harmless and your memory of the visit will be replaced by a
\added[id=MC]{pleasant} recollection of spending the day feeding the
\replaced[id=FC,remark={what geese?}]{ducks}{geese}
\added[id=MC,remark={they're the weird-looking ducks}]{and geese} in
the nearby pond.

\end{document}
```

↓ Input

The changes are colour-coded according to the author who made the edit. Any remarks are added as footnotes, as shown in Figure 13.1. You can download or view this example.

13.2 Version Control

[FAQ: Version control using RCS, CVS or the like]

Version control (also known as revision or source control) is a system that tracks changes to documents and source code. This makes it possible to revert to an earlier version or restore files that are accidentally overwritten or deleted. Typically, the system requires a location where the version control database (repository) is stored (usually password-protected). The authors working on the document pull the latest version of the source code from the database, which creates their own local version of the document files. They can then modify these local files, commit their changes, and push the changed files back to the database.

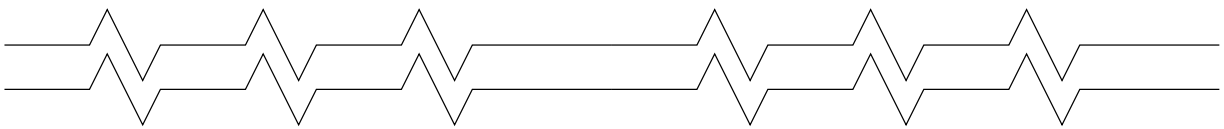
The version control system tracks these changes, who made the changes and when they were made. In the event that two or more authors have made conflicting changes to the same piece of text, the version control system will flag the conflict, and the authors need to decide whether to reject their change or overwrite the other author's change.

Output

1 About the Lab

The Secret Lab of Experimental Stuff is a top-secret~~sinister~~[MC] laboratory whose existence is highly classified so don't tell anyone about it or we'll get really cross with you[FC].

The world-renown[IP] University of Somewhere denies all knowledge of the Secret Lab of Experimental Stuff, except on Open Days where members of the public may visit the facility and ask questions as long as they consent to a memory wipe when they leave. The memory wipe is ~~completely~~[MC1] harmless and your memory of the visit will be replaced by a pleasant[MC] recollection of spending the day feeding the ducks~~geese~~[FC2] and geese[MC3] in the nearby pond.



[1]MC: no it isn't
[2]FC: what geese?
[3]MC: they're the weird-looking ducks

1

Figure 13.1 Recording Changes

For example, if Mabel edits section 1 while Fred edits section 2, when they commit their changes there won't be a conflict. The version control system is able to merge their changes so that the source code on the database contains both the new section 1 and the new section 2. Fred and Mabel can then pull this file to update their own local copy. If they decide they don't like the changes, they can revert to an earlier version of the document. If they accidentally delete their local copy, they can pull a fresh copy from the database.

There are a number of different version control systems, some of which are open source, such as CVS, Subversion (SVN) and Git. Once you have decided which system to use, you need to install the software on your computer and find a location for the database.

In the case of a single author who's only interested in using the version control system as a means of backup, then a local directory can be used. For example, I keep my version control database on an external hard drive. The contents of the external hard drive are periodically backed up to a secondary external device, as an additional precaution. This means that if my primary hard drive fails (which it has done) I can pull all my source code from the database on the external hard drive and I only lose the most recent changes that haven't been committed to the repository. If my external hard drive fails (which it has done) I can restore the repository from the secondary external device and commit my local files. In this case

I lose any of the revisions that haven't been backed up, but I still have the latest version of the document.

In the case of multiple authors, a server will be needed. The choice then comes down to a trade-off between document security and budget. There are free options available for public projects, but private projects usually require payment to a third party hosting service, or a secure server within your institution's firewall can be set up (assuming that all the authors are located within the institution). Some hosting services may be supported by advertisements. Bitbucket allows private hosting for up to 5 users under their free plan and GitHub provides private repositories under their paid plans [36].

Although Fred and Mabel can track all the document changes through the version control system, these changes aren't automatically added to the document and can't be viewed from the resulting PDF. This is often desired, but sometimes it's useful to include some of the revision information, particularly when a document is periodically updated and republished. In this latter case, there are some LaTeX packages that can do this, listed in the version-control topic.

For Git users, there's the gitinfo2 package [54] (which replaces the now deprecated gitinfo). For Subversion users, there are four packages available: svn [51], svninfo [9], svn-multi [80] and svn-prov [78]. This last package (svn-prov) is designed for package authors. These packages are all available on both

TEX Live and MiKTEX. There's also the vc package [34] that supports Git and Subversion, but this isn't on either TEX Live or MiKTEX and requires awk. For CVS users, there are packages such as rcs [85], rcsinfo [112] or rcs-multi [79], which are available both on TEX Live and MiKTEX.

⚠ Version control systems usually create hidden files or directories. If you need to bundle up your document and send it to, say, a journal or conference proceedings, make sure you exclude all the hidden files. For example:

```
zip -r ⟨name⟩.zip ⟨directory⟩ -x "*/\.*"
```

Shell

For Windows users, there's a command line zip program available in GNU On Windows (GOW).

Example 66. Accessing Subversion Revision Information

This book and the accompanying example documents and exercise solutions are in a Subversion repository. In order to insert version control information into one of these documents, I need to add a *keyword anchor*. This is just a bit of text in the form

```
$⟨KeywordName⟩$
```

Input

The ⟨*KeywordName*⟩ indicates the information you want to insert and (for Subversion) may be one of [65]:

`Date` The last time the file was known to have been changed in the repository.

`Revision` The last known revision in which the file was changed in the repository.

`Author` The last known user to change this file in the repository.

`HeadURL` The full URL to the latest version of the file in the repository.

`Id` A compressed combination of the other keywords.

`Header` Similar to `Id` but contains the full URL of the latest revision.

For example:

```
$Id$
```

Input

In addition to adding this text to the file under version control (called, for example, `svninfo-sample.tex`) you also need to tell Subversion to perform a keyword substitution whenever the file is committed. This is done using:

```
svn propset svn:keywords "Id" svninfo-sample.tex
```
Shell

Now, after I next commit the file `svninfo-sample.tex`, the keyword anchor `$Id$` will be changed to the revision information. For example:

```
$Id: svninfo-sample.tex 383 2014-12-12 19:23:03Z nlct $
```
Input

This will cause a problem when I build my document as LaTeX will interpret those dollar `$` symbols as the maths shift special character, so the information won't be typeset correctly. However, recall from §2.1.1 that you can use `\def` to define a command with a particular syntax. For example:

```
\def\svnInfo$#1$%
  % do something with #1
}
```
↑ Input

↓ Input

Now I can have

```
\svnInfo$Id: svninfo-sample.tex 383 2014-12-12 19:23:03Z nlct $
```
Input

and the `$` symbols are part of the syntax and aren't interpreted as the maths mode shift. The svninfo package provides a command called

\svnInfo␣$⟨*Id*⟩$␣ Definition

which is defined in a similar manner. Note that the trailing space is part of the syntax. This command parses the `Id` keyword anchor and gathers the information for later use in the document. By default this information is placed in the page footer.

For example, here's my original document called `svninfo-sample.tex`:

↑ Input

```
\documentclass{article}

\usepackage{svninfo}

\svnInfo $Id$

\author{Nicola Talbot}
\title{A Sample Document}

\begin{document}
\maketitle

This is a sample document.
```

```
\end{document}
```

Input

If I build this document using `pdflatex`, the resulting PDF file contains the footer

Rev: –revision–, December 15, 2014 1 *–sourcefile–*

Output

This is because there's currently no version control information in the file, just the keyword anchor `$Id$`. Next I need to add this new file to my Subversion repository:

```
svn add svninfo-sample.tex
```

Shell

(See the Subversion user manual [65] for information on setting up a repository.) Also, I need to indicate that this file contains the `Id` keyword anchor:

```
svn propset svn:keywords "Id" svninfo-sample.tex
```

Shell

Now I can commit these changes to the repository:

```
svn commit -m "Add svninfo example"
```

Shell

This not only commits the new file svninfo-sample.tex to the repository, but also modifies the file so that it now looks like:

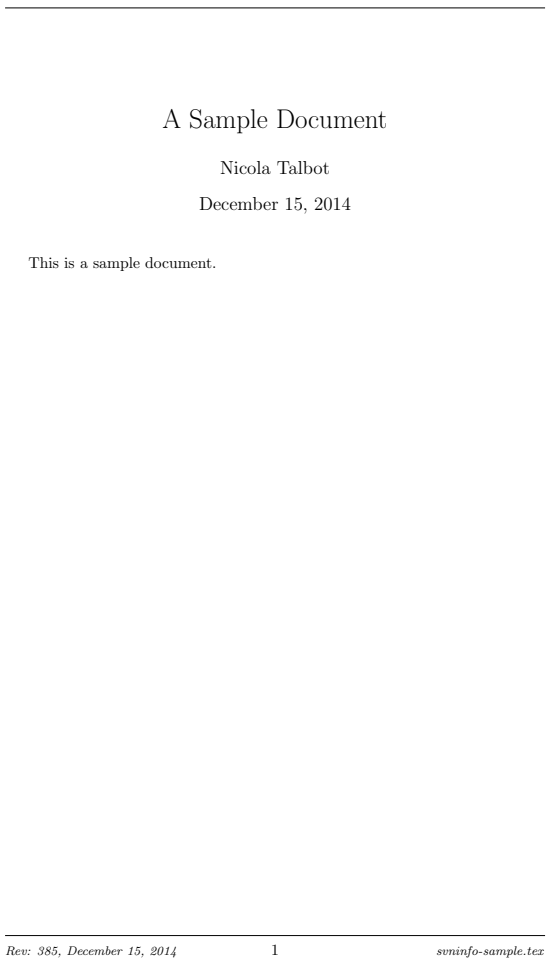
↑ Input

```
\documentclass{article}

\usepackage{svninfo}

\svnInfo $Id: svninfo-sample.tex 385 2014-12-15 11:45:51Z nlct $

\author{Nicola Talbot}
\title{A Sample Document}

\begin{document}
\maketitle

This is a sample document.
\end{document}
```

↓ Input

If I rebuild this document, the PDF file is now as shown in Figure 13.2. The footer now contains the revision number and the filename. You can download or view this example.

Output



A Sample Document

Nicola Talbot

December 15, 2014

This is a sample document.

Rev: 385, December 15, 2014 1 *svninfo-sample.tex*

Figure 13.2 Subversion Revision Information

In addition to \svnInfo, the svninfo package provides a way to gather information from a different keyword, such as Revision, using

\svnKeyword␣$⟨value⟩$␣ Definition

For example:

```
\svnKeyword $Author: nlct$
```
Input

Remember that you need to include these extra keywords in the svn:keywords property to ensure they are also expanded. For example, if you want to access the Author and Date rather than the Id, then you need to set the keywords using

```
svn propset svn:keywords "Date Author" svninfo-sample.tex
```
Shell

(Keywords are case-sensitive.) The footline can be removed using the nofancy package option

```
\usepackage[nofancy]{svninfo}
```
Input

The Subversion information can be accessed using any of the following commands:

\svnInfoFile Definition

for the name of the source file or --sourcefile-- if unknown;

\svnInfoRevision Definition

for the revision number of the checked out file or --revision-- if unknown;

\svnInfoDate Definition

for the date in the form ⟨*YYYY*⟩-⟨*MM*⟩-⟨*DD*⟩ when the file was checked out or the current date if unknown;

\svnInfoTime Definition

for the time when the file was checked out or --time-- if unknown;

\svnInfoOwner Definition

for user name of the file owner or --owner-- if unknown;

\svnInfoYear Definition

for the year when the file was checked out or the current year if unknown;

\svnInfoMonth Definition

for the month (in ⟨*MM*⟩ form) when the file was checked out or the current month if unknown;

\svnInfoDay Definition

for the day (in ⟨*DD*⟩ form) when the file was checked out or the current day if unknown;

\svnInfoLongDate Definition

for the date in the form of \today when the file was checked out or the current date if unknown.

A summary of the Subversion information can be obtained using

\svnId Definition

which uses the same format as the `Id` keyword anchor. In addition to the above commands, the svninfo package also provides commands for multi-file documents. Every file of the document must include either \svnInfo or \svnKeyword with the `Revision` keyword. Two LaTeX runs are required.

\svnInfoMinRevision Definition

displays the minimum revision number for all the files in the document, and

\svnInfoMaxRevision Definition

displays the maximum revision number for all the files in the document. (If unknown, the defaults are --minrevision-- and --maxrevision--.) The date from the latest Subversion revision (in the same format as \today) is obtained using:

\svnInfoMaxToday Definition

If you want the URL to the file in the repository, you need the HeadURL keyword

```
\svnKeyword $HeadURL:$
```
Input

and you can then use:

\svnInfoHeadURL Definition

to access this information.

13.3 Online LaTeX Editors

Online LaTeX editors allow you to store your source code on the provider's server and edit it using your web browser. The source code is compiled using the TeX distribution installed on the server. This means you don't need to worry about co-authors having a different TeX distribution (or even having a distribution at all). However, it means that you're limited to the classes and packages installed on the server and they may not be the most recent version.

The two most popular online LaTeX editors appear to be Overleaf (formerly WriteLaTeX) and ShareLaTeX. The information provided here (such as available features and pricing) may have changed since the time of writing this chapter.

ShareLaTeX There are four plans: Personal, Student, Collaborator, and Professional. The pricing and available features are listed in Table 13.1. The site supports the following languages: Italian, Japanese, Korean, Portuguese, Czech, Dutch, Chinese (Simplified), Norwegian, English, German, Danish, Russian, French, Swedish and Turkish.

To create a free personal account, just go to `https://www.sharelatex.com/register` and enter your email address and the password you want to use. To create a new project, go to `https://www.sharelatex.`

Table 13.1 ShareLaTeX Plans (as at December 2014)

		Personal	Student	Collaborator	Professional
GBP	per month	Free	£6	£12	£24
£	per year	Free	£60	£144	£288
EUR	per month	Free	€7	€14	€28
€	per year	Free	€70	€168	€336
USD	per month	Free	$8	$15	$30
$	per year	Free	$80	$180	$360
Maximum Number of Collaborators		1	6	10	Unlimited
Full Document History?		No	Yes	Yes	Yes
Sync to Dropbox?		No	Yes	Yes	Yes
Sync to GitHub?		No	Yes	Yes	Yes

com/project and click on the "New Project" button. This will drop-down a list of options, such as a blank project, upload a project, import from GitHub, or create a project from a template. There's a wide selection of templates:

- Bibliographies: BibTeX, thebibliography environment, natbib, biblatex, biblatex with split bibliographies, IEEE BibTeX style, and notes2bib.
- Books: includes Springer styles, Tufte style, classic thesis, Wiley styles, MIT styles and sffms (science fiction and fantasy manuscripts).
- Exams.
- Cover letters: moderncv styles and Illinois University.
- Other: includes business cards, pst-barcode, homework styles, various letter styles, brochures, flyers, posters, business reports, recipe, lab reports, assignment sheets, invoices, grant applications, and proposals.
- CV or Résumé: includes moderncv styles, europecv styles, classic, academic, professional, fancy, curve, curvita.
- Thesis: there are a lot of templates in this category, some are general and some are specific to particular institutes.

- Presentations: again there are a lot of templates, many of them using various beamer themes.
- Journals: many different journal styles.

Overleaf There are four plans available: Free, Personal+, Pro and Pro+. The pricing and available features are listed in Table 13.2. There are also tailored solutions for universities, publishers and enterprise. Students get all the features of the Pro account for half price.

When you create a new project with Overleaf, a unique identifier is generated that's used in the link to the project. The files that make up the project are private as long as that link remains private. It's unlikely that anyone will guess the link, but if it does get published or shared then the project can be accessed and edited. It's your responsibility to ensure you don't share the link with anyone who shouldn't access it. (Be careful of the link appearing in your browser history if you use a shared computer.) With the Pro plans you can create protected projects for added security. Overleaf uses Amazon S3 for secure data storage.

Table 13.2 Overleaf Plans (as at December 2014)

		Personal	**Personal+**	**Pro**	**Pro+**
GBP	per month	Free	£5	£6.50	£9.50
£	per year	Free	£48	£60	£90
EUR	per month	Free	€6.50	€7.50	€11.50
€	per year	Free	€60	€72	€108
USD	per month	Free	$9	$10	$15
$	per year	Free	$84	$96	$144
Storage space		up to 1GB	2GB	10GB	20GB
Files per project		60	120	240	500
Unlimited projects and collaborators		Yes	Yes	Yes	Yes
Save and restore version history		Yes	Yes	Yes	Yes
Tag, filter and clone projects		Yes	Yes	Yes	Yes
Quick save to Dropbox		No	Yes	Yes	Yes

Table 13.2 Overleaf Plans (Continued)

	Personal	**Personal+**	**Pro**	**Pro+**
Integrated spell-check	No	Yes	Yes	Yes
Editor themes	No	Yes	Yes	Yes
LaTeX auto-complete	No	Yes	Yes	Yes
Compare versions and see tracked changes	No	No	Yes	Yes
Access control on protected projects	No	No	Yes	Yes
Priority support	No	No	Yes	Yes
Full project history (coming soon)	No	No	Yes	Yes

Table 13.2 Overleaf Plans (Continued)

	Personal	Personal+	Pro	Pro+
PeerJ Free lifetime Publishing Plan	No	No	Basic	Enhanced

To create a free Overleaf account, just go to https://www.overleaf.com/signup, fill in your name and email and click on the create new account button. This takes you to the dashboard and sends you an email with a link to confirm your account registration and choose a password.

To create a new project, click on the "create new project" button in the dashboard. This pops up a window where you can select a template for your document. The available templates are divided into categories:

- Basic: blank paper, sample paper, presentation.
- Academic journals: IEEE Transactions, Springer LNCS, IEEE sponsored conferences and symposia, BioMed Central, MDPI, PeerJ, Elsevier, OSA Express journals, IEEE for Computer Science Journals, IOP journals, Optica, F1000Research, Language Science Press, Springer Journals, APS, Public Library

of Science, Advances in Optics and Photonics (AOP), Royal Society Open Science, Copernicus Publications.

- Bibliography: IEEE with BibTeX, `biber` and biblatex, natbib, Chicago citation style with biblatex.
- Book: Tufte, ePub/eBook, Language Science Press, fiction.
- Formal letter: includes newlfm, memo, professional formal letter, Aalto School of Business Letter, Carleton letter
- Homework assignment: a selection of templates for homework or lab reports.
- Newsletter: Tufte handout template, memo template, a newsletter template, a flowfram template.
- Poster: a selection of templates including a sciposter and a beamer template.
- Presentation: a selection of templates, most of them seem to use beamer with different themes.
- Project lab reports: a large selection of report-like templates.
- Résumé/CV: a selection of CV templates.
- Thesis: a large selection of thesis templates, many of them for specific universities.

One thing I noticed when trying out the templates was that once I had created a project by selecting a template, there was no way to permanently delete it. If you delete a project from the project list on the dashboard, the project is moved to trash and after 30 days, the project will be removed from the dashboard, but the project itself isn't deleted. This was a bit annoying as I only created the projects to see what the templates looked like. Only some of the templates actually included information in their description about the packages used within the template.

I tried both sites using Mozilla Firefox (v33.1) and Google Chrome (v39.0.2171.71) running under 64-bit Fedora 19. In addition, to test accessibility support for visually-impaired users, I tried both sites with the text-only browser Lynx (v2.8.8dev.15) and with the text-to-speech Jovie plugin for Konqueror (v4.11.5).

User Interface

Overleaf The interface was mostly fine on Firefox and Chrome. In general on the site, if the window isn't wide enough for the navigation bar, the navigation links are replaced by a button that drops-down a list of links so a horizontal scroll bar is never required by the browser. When editing a project in the source code pane, there were some issues with trying to copy selected text in the source code pane to

the clipboard. If I used the popup menu's "copy" item, I received a message "This action isn't yet available from the menu, but you can press Ctrl-C instead." This cleared the selection, and I had to reselect the text and use Ctrl-C. (Firefox's Edit→Copy menu item and Chrome's Copy button worked. The issue was just with the popup menu.)

An example document with Overleaf is shown in Figure 13.3. This has a split pane area with the source code on the left and a preview image on the right. The divider bar between the two panes can be moved to make one pane wider than the other. If the source code pane isn't wide enough for all the menu buttons, you can access the remaining buttons through the "More" drop-down menu.

The preview is automatically updated whenever the file is modified, which I found distracting, but the automatic compilation can be switched off by selecting the "Manual" button on the menu bar above the preview image. The only way of enlarging the preview image is to make the preview pane wider.

The source code can also be viewed as rich text. This gives a partial WYSIWYG feel, which may suit some users. To enable this feature, just click on the "Rich Text" button. The document shown in Figure 13.3 is reproduced in Figure 13.4 with this feature on.

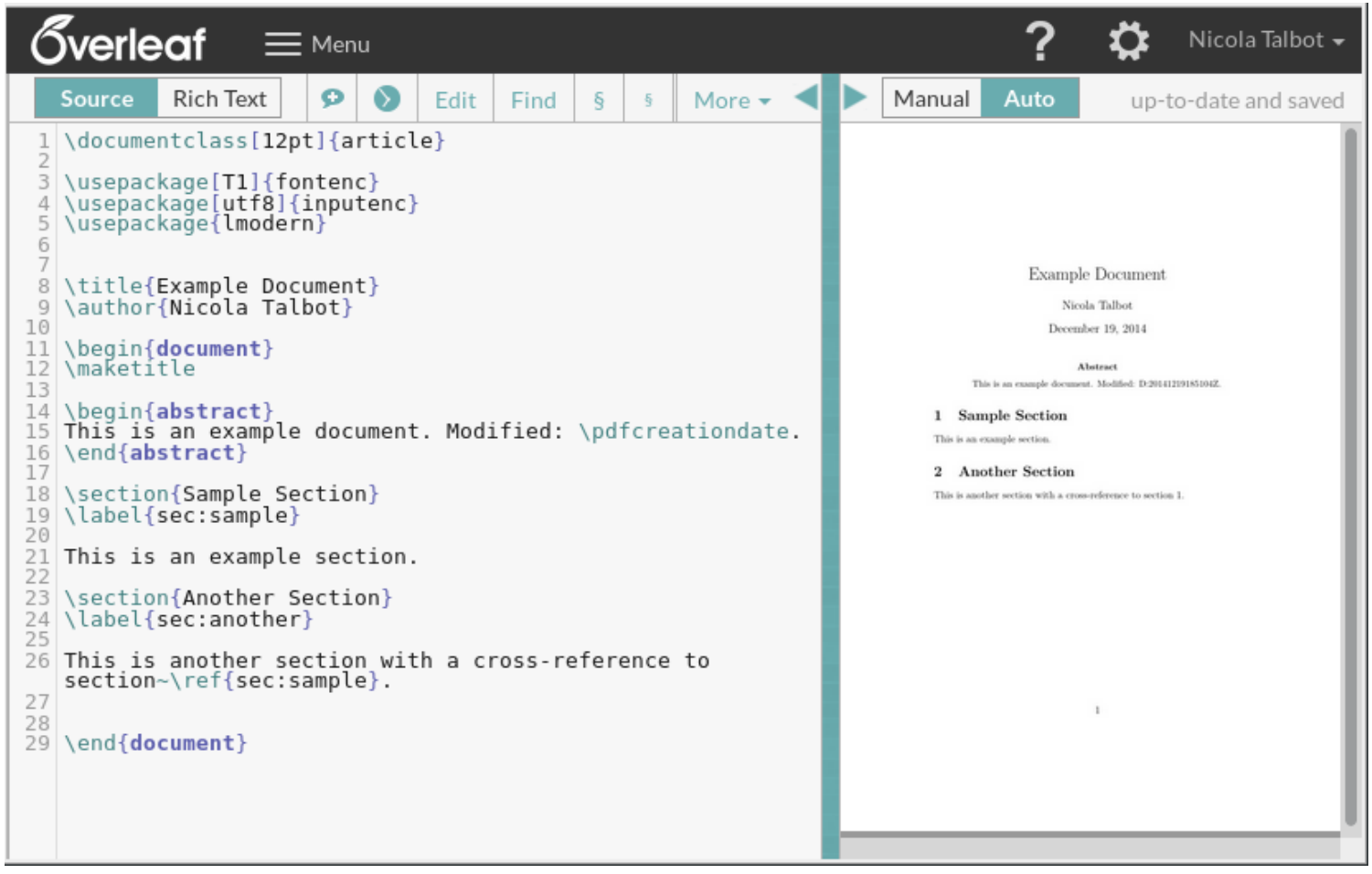


Figure 13.3 Overleaf (Mozilla Firefox)

To switch back to viewing the source code, click on the "Source" button. At the time of writing, the rich text function is still in beta.

Another interesting feature is that if I click on an area of the preview pane (for example, on the first section header) the corresponding line in the source code is highlighted. (There's a short delay between clicking and the source code line being found, so be patient, especially if you have a slow Internet connection.) I wasn't expecting SyncTeX support given that the preview window is an image, so I was pleasantly surprised by this.

The free version of Overleaf doesn't have spell-checking or the ability to change the editor themes, but it does have syntax highlighting.

The list of source files that make up a project can be viewed using the "Menu" button on the main Overleaf navigation bar at the top of the window. Select the "Project" item from this drop-down menu and a window will list the files (shown in Figure 13.5). To add new files, you can click on the "Add files..." button which drops-down a menu where you can choose various options, such as create a new file or folder, or upload from your computer or off-site storage, such as GoogleDocs or Dropbox.

You can use the file list to switch the editor to a different source

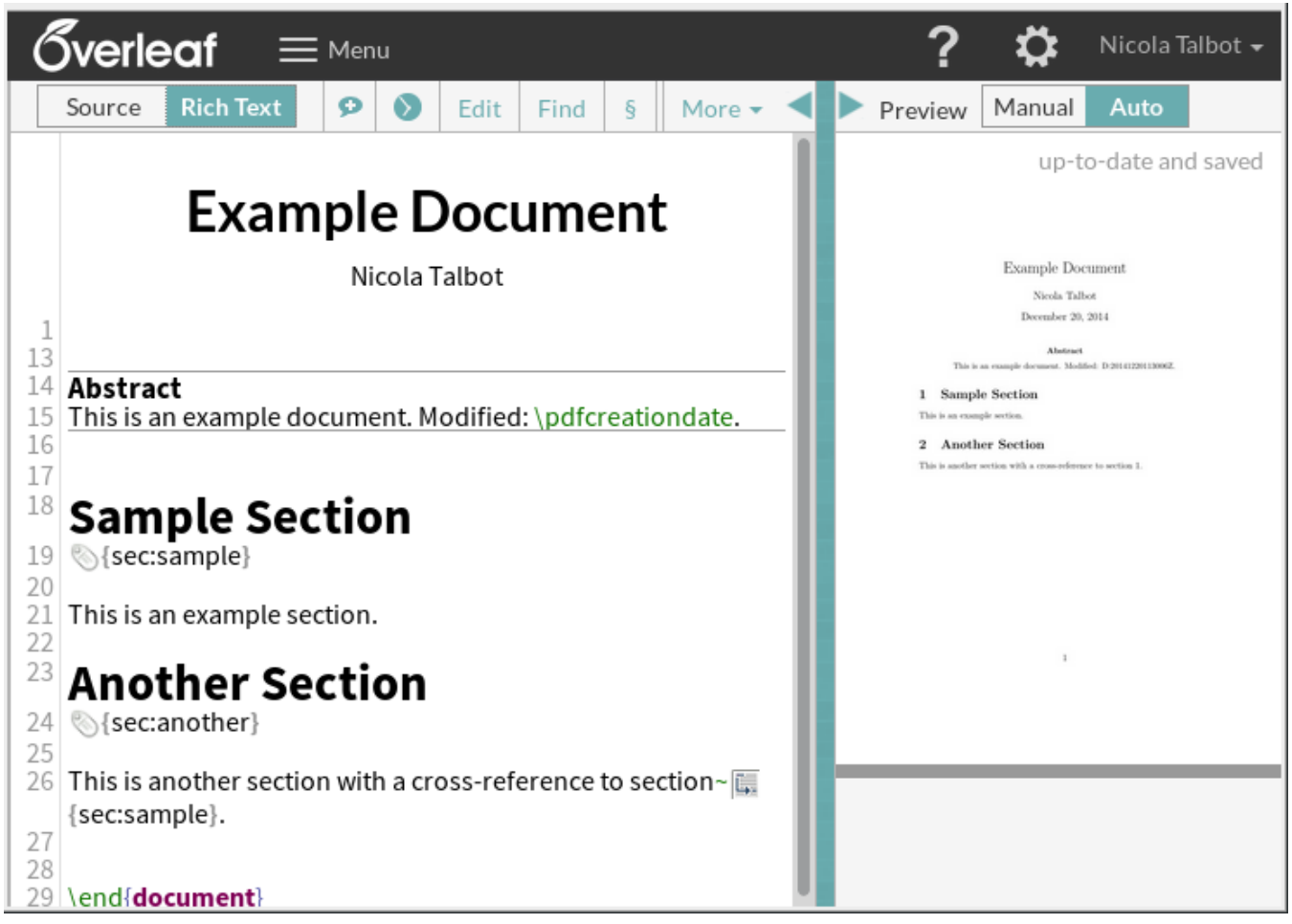


Figure 13.4 Overleaf — Rich Text Enabled

file or you can use the drop-down menu next to the filename to rename or delete the file or perform other operations.

There is also a settings button to the left of the account holder's name on the top navigation bar. The spell check is disabled because it's not available for the free plan, but it's possible to set the editor mode to Vim or Emacs, and the font size can be made smaller or larger.

ShareLaTeX I had no problems with the interface on Firefox or Chrome. As with Overleaf, if you resize the browser, the contents expand or contract with line wrapping in the source rather than introducing a horizontal scrollbar. As you type a command, a drop-down list of suggestions appears. You can click on an item in this list to complete the command name or to fill in the begin and end of an environment. Lines are numbered, but a widget appears next to a line number where an environment begins. You can click on this widget to collapse the environment, hiding its contents from view. This widget is also available for other commands, such as the sectioning commands. When you open a project, it's automatically compiled, but once you edit the source code, you need to manually recompile by clicking on the "Recompile" button.

The ShareLaTeX project window (Figure 13.6) has three panels: the

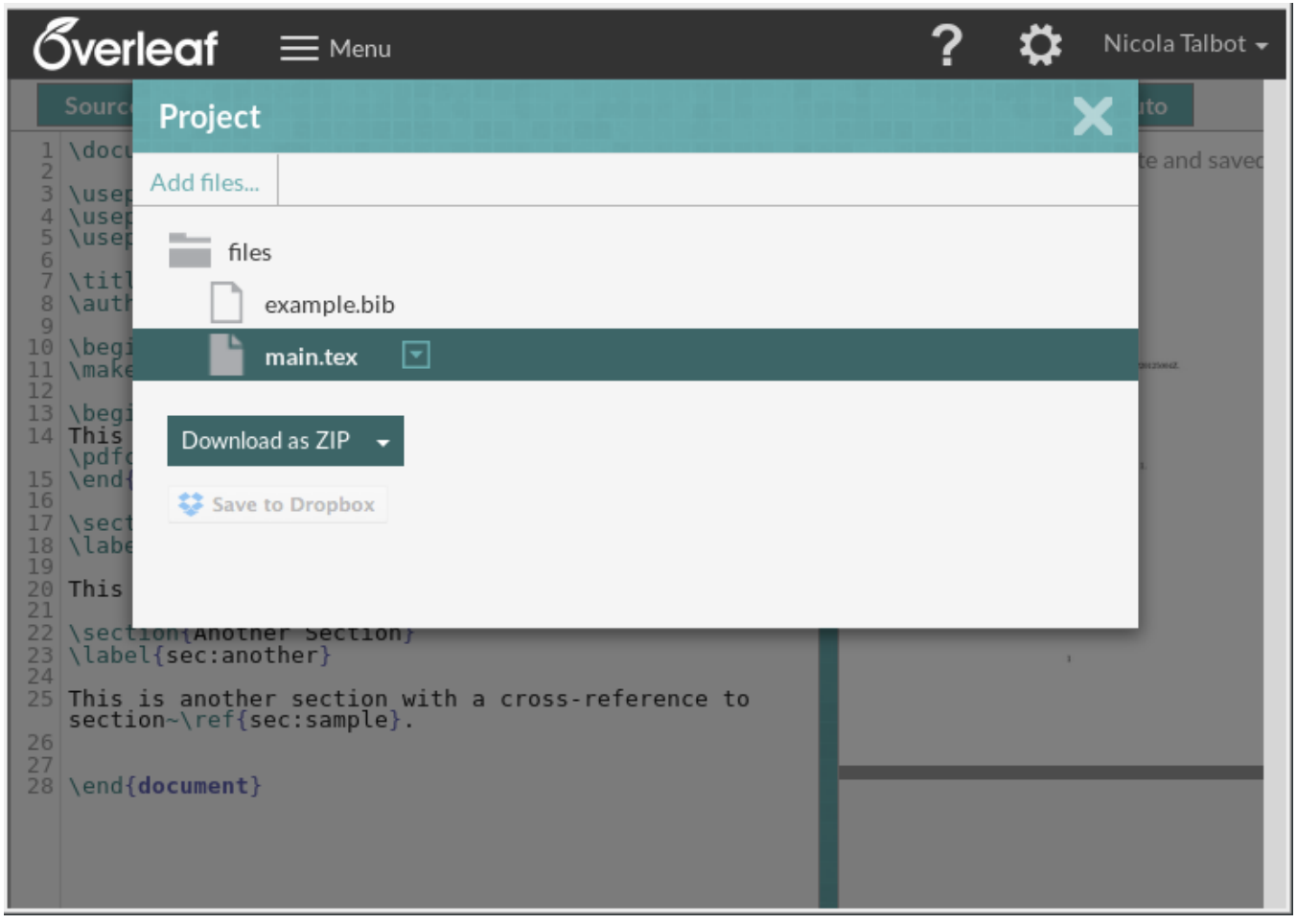


Figure 13.5 Overleaf — Project Files

file list panel, the source code panel, and the preview panel. The file list panel has buttons that allow you to create new files or folders, upload files, rename files or delete files. You can click on a filename in the list to switch the source code panel to that file. The preview panel has zoom buttons that are displayed when you move the mouse over the top of the preview image. Again SyncTeX is enabled. You can double-click on an area of the preview image to jump to the corresponding line of code in the middle panel or you can use the right and left arrows between the source code and preview panels.

The style of the source code panel can be changed using the menu widget (the left-most button above the file list panel) which pops up the panel shown in Figure 13.7. (The editor theme is in the lower part of the panel off the edge of the window but can be reached using the vertical scroll bar.) The configurable settings are: compiler (PDFLaTeX, LaTeX, XeLaTeX or LuaLaTeX), main document file, spell check language, auto-complete, theme (various available), key-bindings (none, Vim[1] or Emacs), font size, and PDF viewer (built-in or native). There are also hotkeys that you can use to navigate your way around the source code. These can be displayed

[1] The Vim key-bindings with the "terminal" editor theme provided an interface similar to my accustomed editing preferences, which I liked.

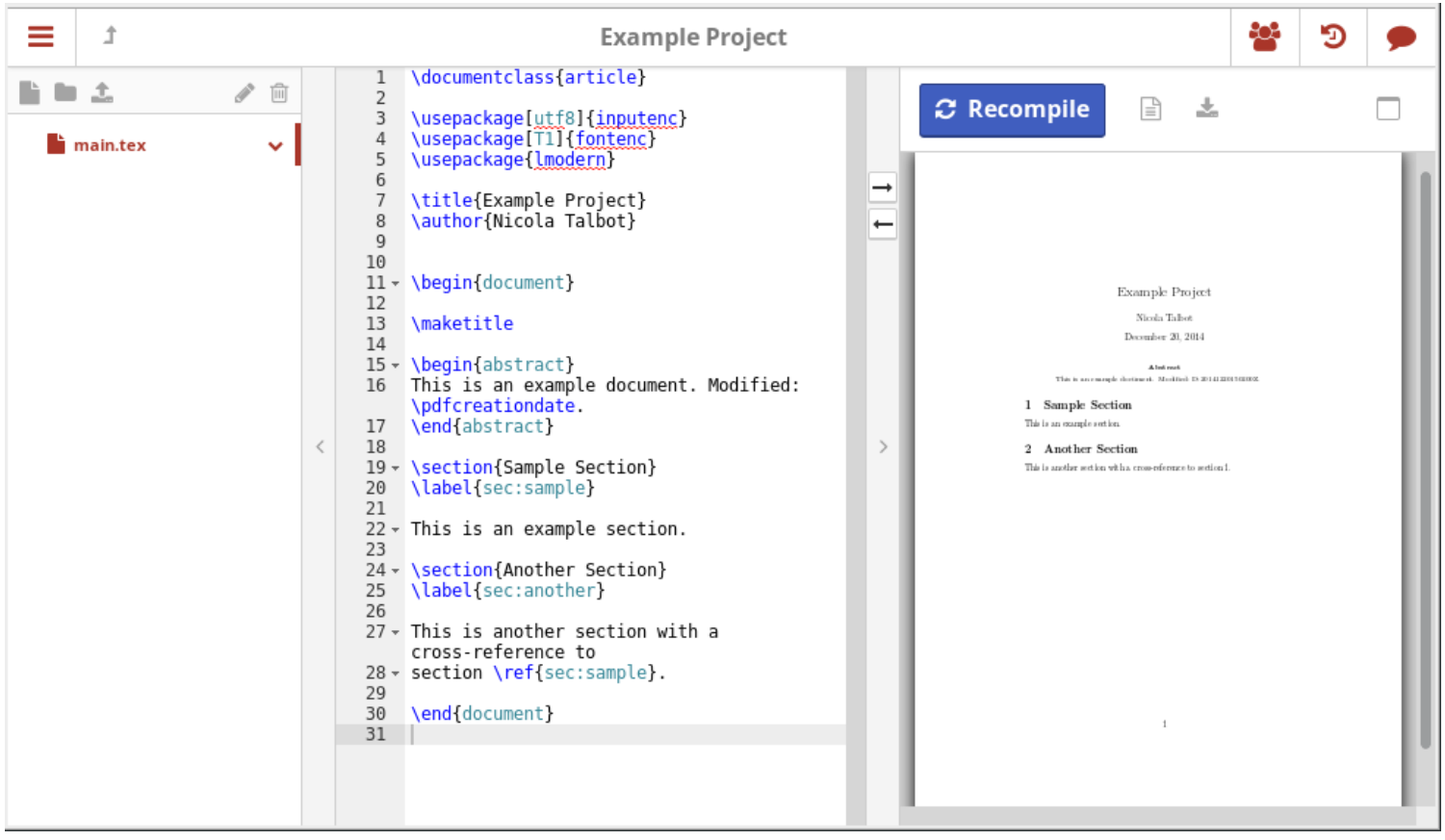


Figure 13.6 ShareLaTeX (Mozilla Firefox)

by clicking on the "Show Hotkeys" button below the settings list. To close the settings panel, just click on one of the other panels.

Error Handling

To test the error handling, I misspelt a command. (I removed the "f" from the command `\pdfcreationdate`.)

Overleaf The line where the error occurs is highlighted and a popup window displays the message immediately below the line (see Figure 13.8). Moving the cursor away from the error line will make the message disappear. Moving the cursor back to the error line will make the message reappear. At the right-hand end of the menu bar above the preview pane, there's a red drop-down menu labelled "compile error". There are two items: "go to first error/warning" and "open latex log". I couldn't find any way to open the log file when there weren't any errors or warnings. (There are some useful bits of information in log files that aren't error or warning messages, which I sometimes like to check even if the document appears okay.)

ShareLaTeX The line where the error occurs has a small red box with a white cross in it next to the line number. If you move the mouse

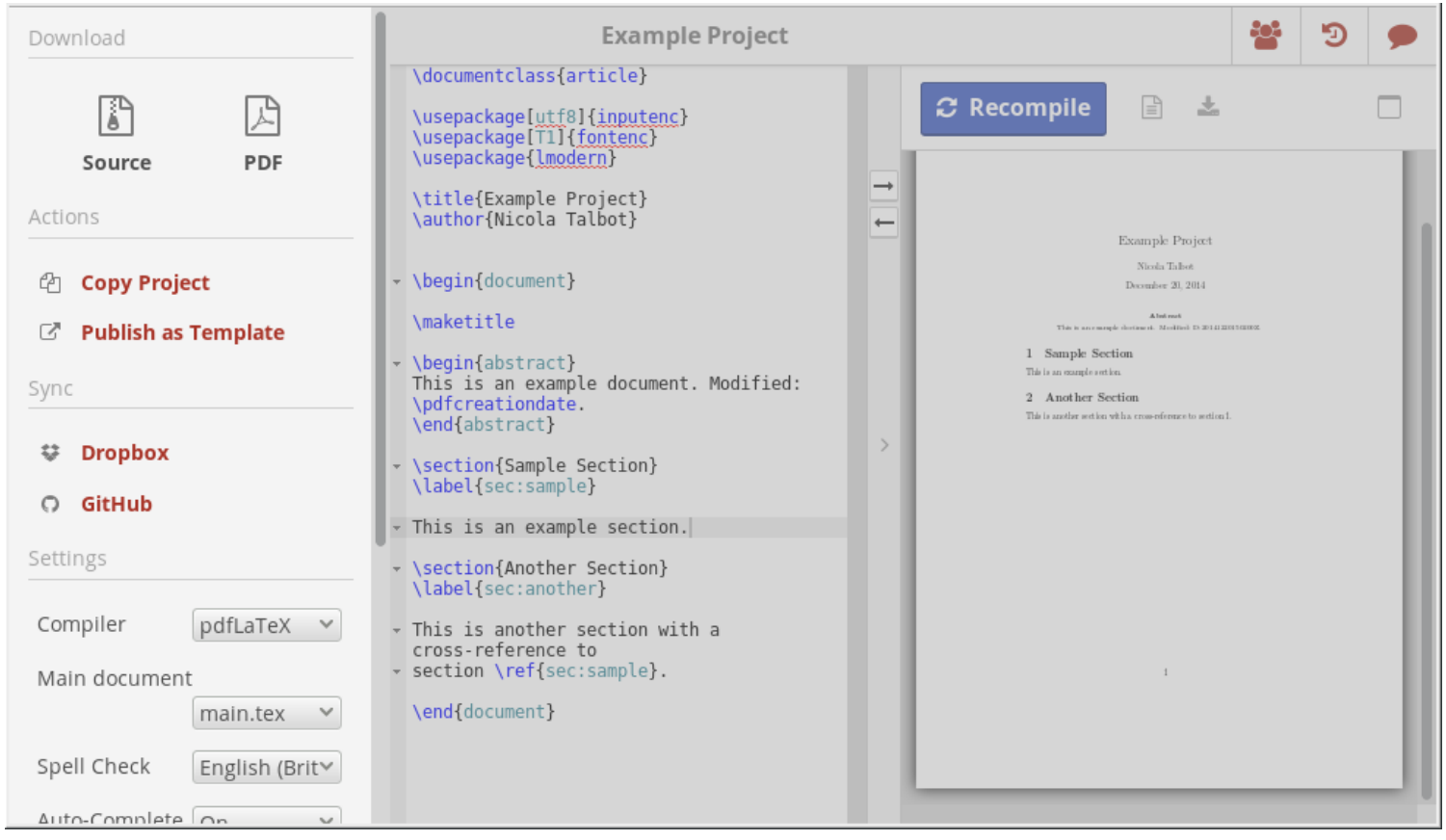


Figure 13.7 ShareLaTeX — Settings

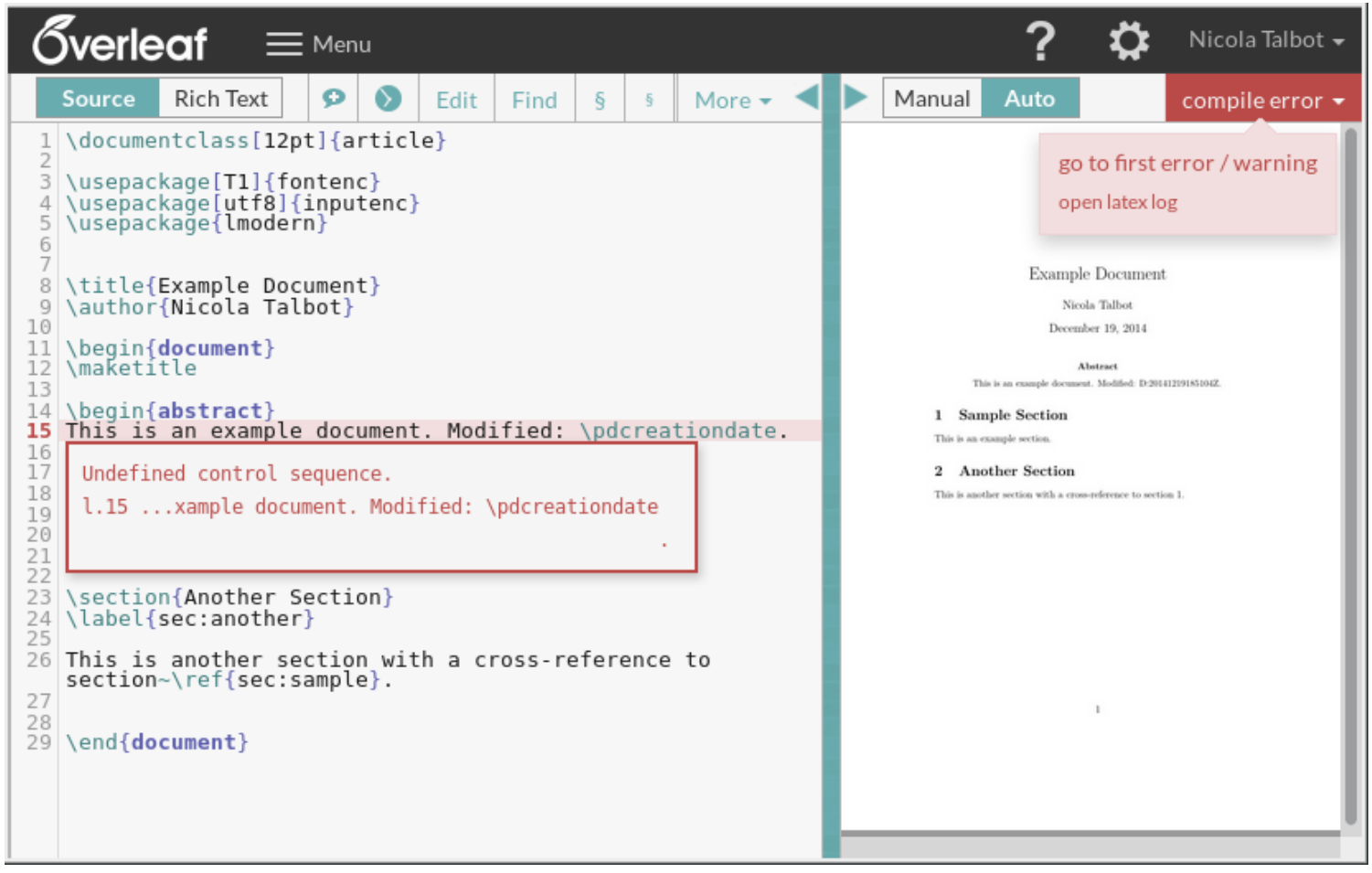


Figure 13.8 Overleaf — Error Handling

over this box, the error message will be displayed. You can also click on the "Log and output files" button above the preview window to display more details about the error (as shown in Figure 13.9). Below the detailed error message are buttons that allow you to view the full transcript, clear cached files and a drop-down menu that allows you to view the auxiliary files.

TeX Engine

Overleaf I was interested to discover that Overleaf seems to be able to determine what engine to use when compiling the document. The example document shown in Figure 13.3 uses `\pdfcreationdate`, which is a PDFTeX primitive. Inspecting the log file shows that PDFLaTeX is being used and the command successfully produces the date stamp. However, if I add the pstricks package to this document, I get an "Undefined control sequence" error for this command. Investigating the log file shows that XeLaTeX is being used (which doesn't provide `\pdfcreationdate`).

ShareLaTeX With ShareLaTeX you need to explicitly set the engine you want to use. You have a choice of PDFLaTeX, LaTeX, XeLaTeX and LuaLaTeX. The default is PDFLaTeX.

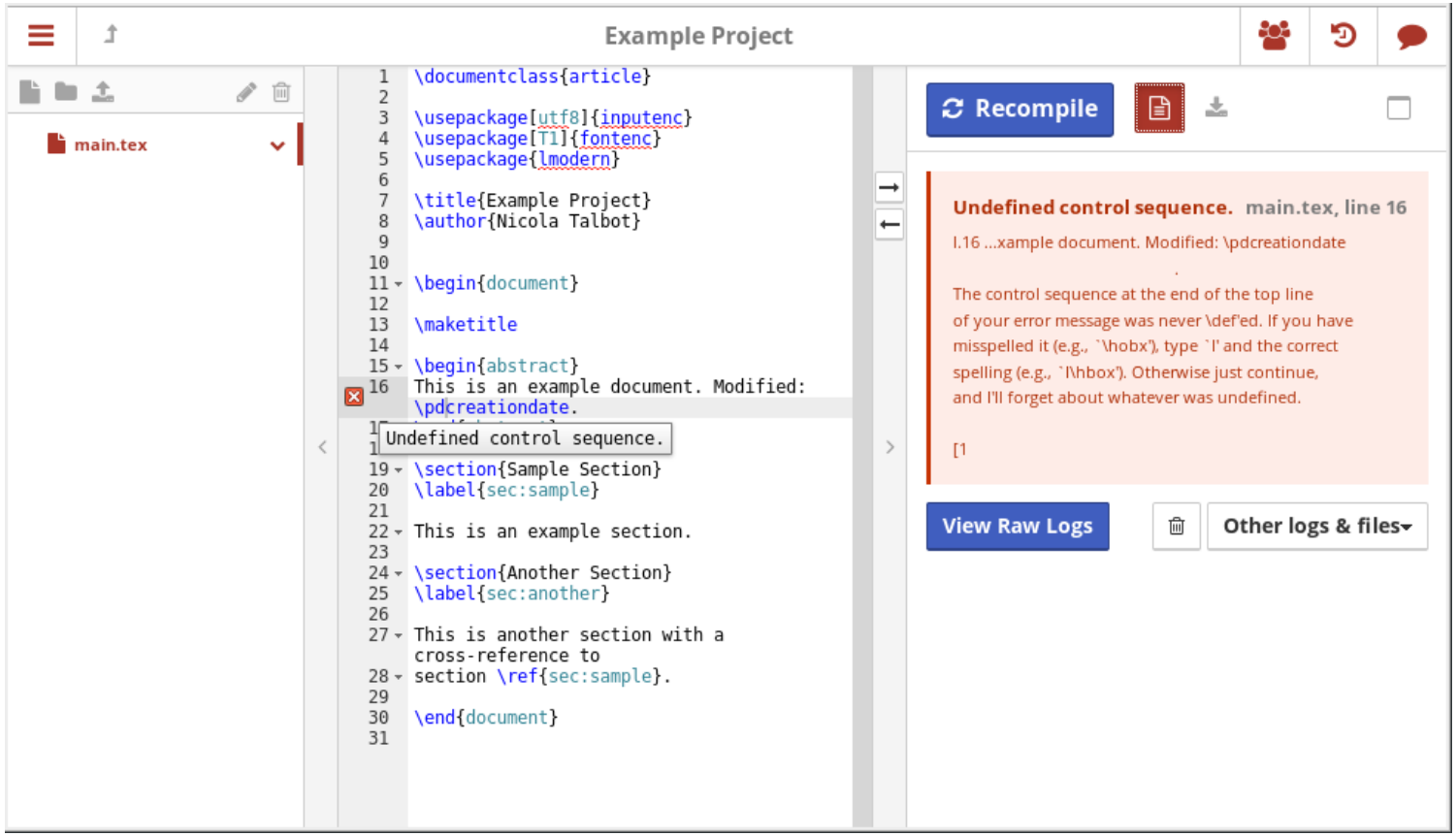


Figure 13.9 ShareLaTeX — Error Handling

Available Classes and Packages

I tried out the classes and packages described in this book. Unsurprisingly, packages that aren't on CTAN, such as pgfornament, also aren't available with the online editors nor are some packages that require external scripts, such as piechart and vc. Neither site had the non-TeX Live packages eforms, draftmark, pgf-pie and rtsched. Inspecting the generated log files showed that both sites use TeX Live on a Unix-like system.

In addition to the above, Overleaf didn't have drm or gitinfo2 installed (but it did have gitinfo2's predecessor, gitinfo) nor did it have the pressrelease class. (Both drm and pressrelease were released in September 2014, three months before the time of writing this, and gitinfo2 was new in May 2014.) ShareLaTeX had drm, gitinfo2 and pressrelease installed.

Therefore it seems that, at the time of writing, both Overleaf and ShareLaTeX have reasonably up-to-date TeX Live distributions, but ShareLaTeX's distribution is more up-to-date. This may, of course, have changed since I tested both sites.

External Applications

To test commonly-used external applications, I added an example bib file (called `example.bib`) that contained a sample entry:

```
@book{sample,
  title = "A Sample Book",
  author = "Ann Author",
  publisher = "A Publisher",
  year = 2014
}
```

I added this sample citation to my document and also added an index (with the makeidx package), and a glossary and list of acronyms (with the glossaries package).

Both Overleaf and ShareLaTeX correctly generated the bibliography, index, glossary and list of acronyms. I didn't need to specify any of the external applications, both sites automatically ran `bibtex` and `makeindex` on the appropriate files.

Interaction Between Collaborators

Overleaf I viewed my example source file in both Firefox and Chrome at the same time. As I edited the text in one window, the other window was automatically updated, so it seems that all collaborators should always be viewing the latest version of the code.

Each collaborator can add comments to the code using the "Add a comment" button (it looks like a speech bubble with a plus inside it).

I added a comment, and the source code pane was switched to rich text format, as shown in Figure 13.10. The comment can be hidden using the "Hide Comment" button on the top left of the comment box. When the comment is no longer needed, it can be closed using the "close" link at the bottom of the comment box or another author can reply to the comment using the "reply" link. If you switch back from the rich text mode to the regular source code mode, the comment is present using TeX's standard commenting mechanism with the % character, but it also includes the commenter's email address and the date the comment was made.

ShareLaTeX The free plan that I tried out didn't support multiple collaborators.

Accessibility

Given the complexity and visual nature of both sites, I didn't expect either site to work well with a simple text-only browser such as Lynx, but that and the Konqueror Jovie plugin were the only means I had to test accessibility for the visually impaired.

Overleaf The site doesn't appear to provide alternative text for images. For example, on the pricing and plans page the ticks and crosses that

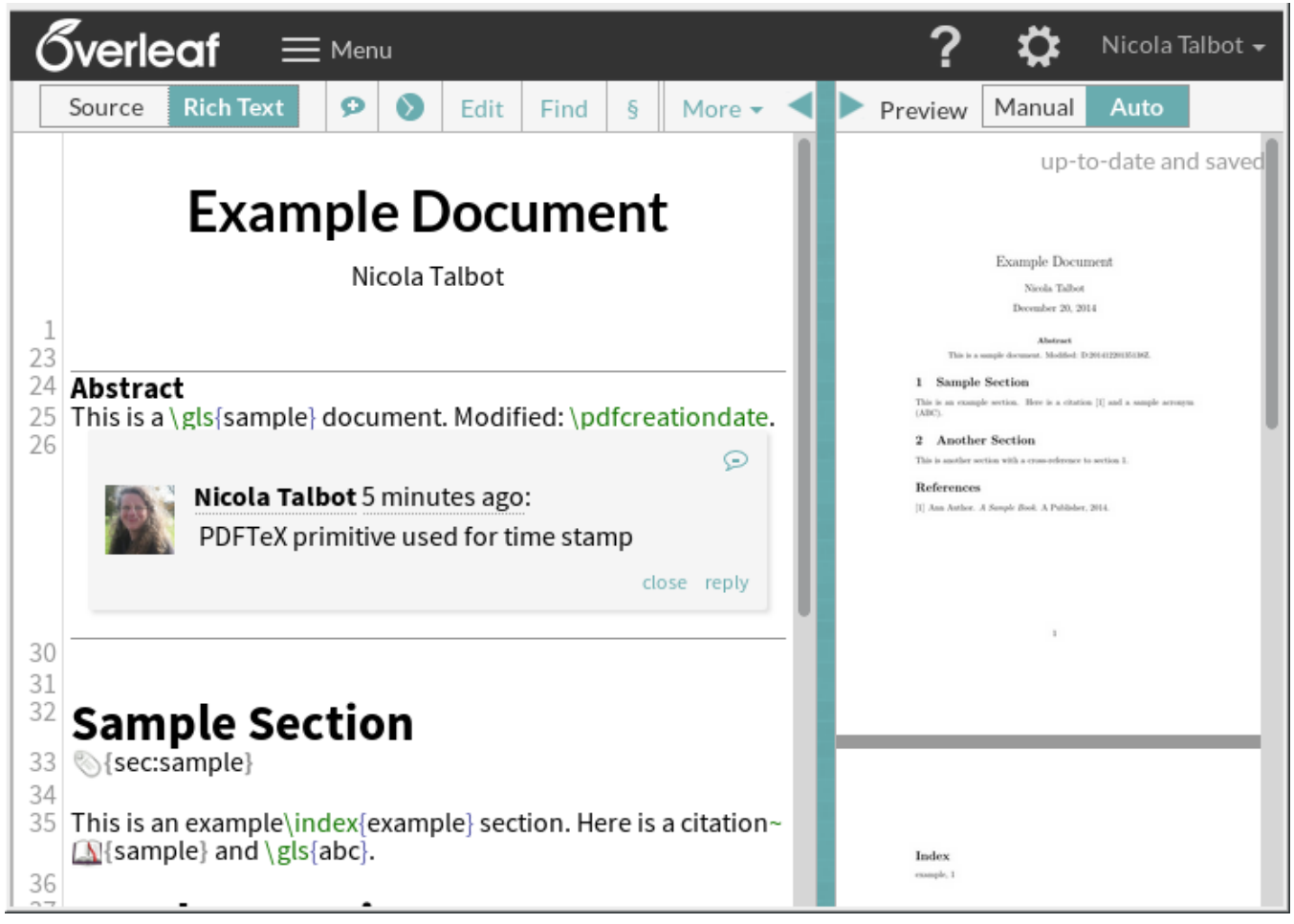


Figure 13.10 Overleaf — Comment Viewed as Rich Text

indicate which options are available with which plans are invisible to the text-to-speech synthesizer, which is useless for any users who can't see the images. On the other hand, text that isn't visually displayed on the page is read out, which is a bit confusing.

The project window doesn't render correctly in either Lynx or Konqueror and I wasn't able to view or edit the document source code. It's possible that a more sophisticated screen reader, such as Jaws, can produce better results with a different browser.

ShareLaTeX Again there didn't appear to be any alternative text for images. When viewed in Lynx, the prices weren't visible on the Plans and Pricing page, however they were read out by Konqueror's Jovie plugin. I couldn't view the project window in Konqueror. I just got an error claiming that the project had been modified by collaborators and that I should ask the project owner to upgrade the account. Again, it's possible that a more sophisticated speech reader can produce better results with a different browser.

Mobile Devices

I tried both sites on my Arnova 7GB tablet. This is a very basic mobile device and my rural broadband is a little iffy, so I didn't expect a fast

response from either site. I also borrowed my brother's iPad and my son's "it tablet" and tried both sites on those.

Both sites worked fine with the iPad and the it tablet, but they didn't work well with the Arnova tablet.

Overleaf I was able to login with the Arnova's default web browser and with Opera Mobile, but whenever I tried to access my list of projects from the dashboard, I was redirected to the site's home page.

ShareLaTeX The Arnova's default web browser and also the Opera Mobile browser were unable to render the email and password fields in ShareLaTeX's login page, so I was unable to test out the site as I was unable to login.

Summary

Both sites are impressive and have similar interfaces, but they are not browser-independent, only fully-functioning with the big well-known browsers (which is not really surprising given the complexity of the sites). If you have collaborators who have difficulty using complicated web interfaces (for example, if they are visually-impaired) you may want to consider using a version control repository instead so that they can use their preferred text editor and other tools that are accessible for them.

ShareLaTeX's paid plans have the ability to sync to GitHub, but remember that private GitHub repositories also require payment. Both sites have paid plans that also provide the ability to save to Dropbox. Overleaf's paid plans are cheaper than ShareLaTeX's.

If any of your collaborators are wary of LaTeX and are easily confused by the document commands, they may be swayed by Overleaf's rich text view if all they need to do is write text (as opposed to complicated equations or diagrams).

ShareLaTeX's free plan comes with a spell checker, multiple editor themes and auto-complete, but Overleaf's free plan doesn't. (The paid plans do.) On the other hand, ShareLaTeX's free plan doesn't allow multiple collaborators whereas Overleaf's free plan does.

Both sites displayed the time stamp in UTC+0, which also happens to be my current time zone (GMT) at the time of writing. This surprised me as I was expecting at least one of the sites to use a server in a different time zone. I can't tell if they are actually in the same time zone as me or if they can determine my location and set the time zone at the start of the document compilation.

Acknowledgements

Thank you to Joseph Wright for proof-reading this book and providing useful feedback. Thank you to my son, Cameron, for providing the Sumatra and Foxit on Windows screen shots, and for lending his tablet to try out the online editors. Thank you to my brother Stephen Talbot for lending his iPad to try out the online editors. Thank you to Paulo Cereda for being so patient with my bug reports and awkward feature requests for arara! Thank you to the package and class authors who responded to my bug reports. Thank you to Barbara Beeton for spotting errors and giving useful advice, and to everyone else who provided feedback during the creation of this book, and thank you to all those who keep the TEX community going, including the CTAN team, TUG and local TEX user groups, and all the class and package authors who share the fruits of their labours! And, of course, thank you to Donald Knuth for creating TEX, Leslie Lamport for creating LATEX, the LATEX team for taking over the development of LATEX, and the developers of PDFTEX, XETEX and LuaTEX.

If you're not already a member of TUG or a local TEX user group, please consider joining.

Bibliography

[1] American Mathematical Society (AMS). User's guide for the amsmath package (version 2.0), 2002. CTAN Mirror or `texdoc amsmath`.

[2] Donald Arseneau. The ulem package, 2011. CTAN Mirror or `texdoc ulem`.

[3] Donald Arseneau. The framed package, 2012. CTAN Mirror or `texdoc framed`.

[4] Donald Arseneau. shapepar.sty v2.2, 2013. CTAN Mirror or `texdoc shapepar`.

[5] Donald Arseneau. url.sty version 3.4, 2013. CTAN Mirror or `texdoc url`.

[6] Brian D. Beitzel. Formatting meeting minutes with the meetingmins LaTeX class, 2013. CTAN Mirror or `texdoc meetingmins`.

[7] Johannes L. Braams and Javier Bezos. babel version 3.9k, 2014. CTAN Mirror or `texdoc babel`.

[8] Klaus Dieter Braune and Richard Gussmann. Standard document class 'dinbrief', 2007. CTAN Mirror or `texdoc dinbrief`.

[9] Achim D. Brucker. The svninfo package version 0.7.4, 2010. CTAN Mirror or `texdoc svninfo`.

[10] David Carlisle. The ifthen package, 2001. `texdoc ifthen`.

[11] David Carlisle. The longtable package, 2004. CTAN Mirror or `texdoc longtable`.

[12] David Carlisle. The keyval package, 2014. `texdoc keyval`.

[13] David Carlisle. The tabularx package, 2014. `texdoc tabularx`.

[14] David Carlisle and The LaTeX3 Project. Packages in the 'graphics' bundle, 2014. CTAN Mirror or `texdoc graphics`.

[15] Florent Chervet. The etextools macros, 2010. CTAN Mirror or `texdoc etextools`.

[16] The comp.text.tex news group. http://groups.google.com/group/comp.text.tex.

[17] Oliver Corff. invoice 0.9: a package for writing invoices, 2011. CTAN Mirror or texdoc invoice.

[18] The comprehensive TEX archive network. http://ctan.org/.

[19] Patrick W. Daly. Natural sciences citations and references, 2010. CTAN Mirror or texdoc natbib.

[20] Marco Daniel and Elke Schubert. The mdframed package: auto-split frame environment, 2013. CTAN Mirror or texdoc mdframed.

[21] Wybo Dekker. The isodoc class for letters, invoices, and more, 2014. CTAN Mirror or texdoc isodoc.

[22] Jean-Pierre F. Drucbert. The minitoc package, 2008. CTAN Mirror or texdoc minitoc.

[23] Thomas Emmel. ticket.sty: making labels, visiting cards, pins and flash-cards with LATEX, 2010. CTAN Mirror or texdoc ticket.

[24] Simon Fear. Publication quality tables with LaTeX, 2005. CTAN Mirror or texdoc booktabs.

[25] Christian Feuersänger. Manual for package pgfplots: 2D/3D plots in LaTeX, version 1.11, 2014. CTAN Mirror or texdoc pgfplots.

[26] Denis Girou and Herbert Voß. pst-gantt v.0.22, 2013. CTAN Mirror or texdoc pst-gantt.

[27] Jeff Goldberg. MS-Word is *Not* a document exchange format. http://www.goldmark.org/netrants/no-word/attach.html, 2005. Accessed: 2014-11-10.

[28] Donald P. Goodman III. The drm font package, 2015. CTAN Mirror or texdoc drm.

[29] GNU on Windows. http://wiki.github.com/bmatzelle/gow/. Accessed: 2015-09-05.

[30] Thorsten Hansen. The bibunits package, 2004. CTAN Mirror or texdoc bibunits.

[31] Thorsten Hansen. The multibib package, 2008. CTAN Mirror or texdoc multibib.

[32] Patrick Happel. lipsum — access to 150 paragraphs of *Lorem ipsum* dummy text, 2014. CTAN Mirror or `texdoc lipsum`.

[33] Carsten Heinz, Brooks Moses, and Jobst Hoffmann. The listings package, 2014. CTAN Mirror or `texdoc listings`.

[34] Stephan Hennig. The vc bundle, 2008. CTAN Mirror or `texdoc vc-manual`.

[35] Philip S. Hirschhorn. Using the exam document class, 2011. CTAN Mirror or `texdoc exam`.

[36] Neil Chue Hong. Choosing a repository for your software project. http://software.ac.uk/resources/guides/choosing-repository-your-software-project, 2013. Accessed: 2014-12-12.

[37] Information Commissioner's Office. Data protection and freedom of information advice. http://ico.org.uk/. Accessed: 2014-09-24.

[38] Bogusław Jackowski and Janusz Marian Nowacki. The Latin Modern family of fonts, 2009. CTAN Mirror or `texdoc lmodern`.

[39] Alan Jeffrey and Frank Mittelbach. inputenc.sty v1.2b, 2014. CTAN Mirror or `texdoc inputenc`.

[40] Uwe Kern. Extending LaTeX's color facilities: the xcolor package, 2007. CTAN Mirror or `texdoc xcolor`.

[41] Axel Kielhorn. The wasysym macro package for LaTeX 2ε v2.0, 2003. CTAN Mirror or `texdoc wasysym`.

[42] Sebastian Marius Kirsch. bizcard: a LaTeX 2ε package for business/visiting/calling cards, 1999. CTAN Mirror or `texdoc bizcard`.

[43] Ekkart Kleinod. The changes-package: Manual change markup — version 2.0.3, 2014. CTAN Mirror or `texdoc changes.english`.

[44] Ingo Klöckl. Paket ifsym und die `if...`-fonts (German), 2001. CTAN Mirror or `texdoc ifsym`.

[45] Donald Ervin Knuth. *The TeXbook*. Addison-Wesley, 1986.

[46] Markus Kohm and Jens-Uwe Morawski. KOMA-Script a versatile LaTeX 2ε bundle, 2014. CTAN Mirror or `texdoc koma`.

[47] Tobias Kuhn. bchart: Simple bar charts in LaTeX, 2012. CTAN Mirror or `texdoc bchart`.

[48] The LaTeX font catalogue. http://www.tug.dk/FontCatalogue/.

[49] The LaTeX3 Project. The LaTeX3 interfaces, 2014. texdoc interface3.

[50] Philipp Lehman and Joseph Wright. The etoolbox package: an e-TeX toolbox for class and package authors, 2011. CTAN Mirror or texdoc etoolbox.

[51] Richard Lewis. The svn package, 2007. CTAN Mirror or texdoc svn.

[52] Knut Lickert. LaTeX 2ε for people in associations minutes.sty, 2009. CTAN Mirror or texdoc minutes.

[53] Giuseppe Lipari. The rtsched package for LaTeX (version 1.0), 2011. CTAN Mirror or texdoc rtsche.

[54] Brent Longborough. gitinfo2.sty: a package for accessing metadata from the git DVCS version 2.0.4, 2014. CTAN Mirror or texdoc gitinfo2.

[55] Gonzalo Medina. The background package, 2014. CTAN Mirror or texdoc background.

[56] Michael Mehlich. fp package, 1999. CTAN Mirror or texdoc fp.

[57] Frank Mittelbach and David Carlisle. A new implementation of LaTeX's tabular and array environment, 2014. CTAN Mirror or texdoc array.

[58] Frank Mittelbach, Robin Fairbairns, Werner Lemberg, and LaTeX3 Project Team. LaTeX font encodings, 2014. texdoc encguide.

[59] Ahmed Musa. The xwatermark package, 2012. CTAN Mirror or texdoc xwatermark.

[60] MySQL::market share. http://www.mysql.com/why-mysql/marketshare/.

[61] Clemens Niederberger. The ExSheets bundle, 2014. CTAN Mirror or texdoc exsheets.

[62] Rolf Niepraschk and Hubert Gäßlein. The pst-pdf package, 2008. CTAN Mirror or texdoc pst-pdf.

[63] Rolf Niepraschk, Walter Schmidt, and Hubert Gäßlein. The document class leaflet, 2013. CTAN Mirror or texdoc leaflet-manual.

[64] Scott Pakin. The comprehensive LaTeX symbol list, 2009. CTAN Mirror or texdoc symbols.

[65] C. Michael Pilato, Ben Collins-Sussman, and Brian W. Fitzpatrick. *Version Control with Subversion*. O'Reilly, 2008. http://svnbook.red-bean.com/.

[66] CV Radhakrishnan, CV Rajagopal, and Antoine Chambert-Loir. Trivial experiments with psTricks manipulation, 2003. CTAN Mirror or texdoc pdftricks.

[67] Sebastian Rahtz, Leonor Barroca, Grant Gustafson, and Julian Gilbey. A package for making sticky labels in LaTeX, 2003. texdoc labels.

[68] Sebastian Rahtz and Heiko Oberdiek. Hypertext marks in LaTeX: a manual for hyperref, 2012. CTAN Mirror or texdoc hyperref.

[69] Luis Rández and Juan I. Montijano. The ticking digital clock tdclock package v2.3, 2014. CTAN Mirror or texdoc tdclock.

[70] George Reese, Randy Jay Yarger, and Tim King. *Managing & Using MySQL*. O'Reilly, 2nd edition, 2002.

[71] Axel Reichert. currvita.sty, 1999. CTAN Mirror or texdoc currvita.

[72] Alan Ristow. pst-bar v.0.92: bar charts for pstricks, 2008. CTAN Mirror or texdoc pst-bar.

[73] R. M. Ritter. *Oxford Style Manual*. Oxford University Press, 2003.

[74] Will Robertson. A couple of things involving environments (environ v0.3), 2014. CTAN Mirror or `texdoc environ`.

[75] Will Robertson and Khaled Hosny. The fontspec package: Font selection for XeLaTeX and LuaLaTeX, 2014. CTAN Mirror or `texdoc fontspec`.

[76] Jonathan Sauer. The collect package, 2004. CTAN Mirror or `texdoc collect`.

[77] Bernd Schandl. paralist extended list environments, 2013. CTAN Mirror or `texdoc paralist`.

[78] Martin Scharrer. The svn-prov package: Use SVN Id keywords for package, class and file header (version 3.1862d), 2010. CTAN Mirror or `texdoc svn-prov`.

[79] Martin Scharrer. The rcs-multi package v0.1a, 2011. CTAN Mirror or `texdoc rcs-multi`.

[80] Martin Scharrer. The svn-multi package version 2.4d, 2011. CTAN Mirror or `texdoc svn-multi`.

[81] Martin Scharrer. The standalone package, 2012. CTAN Mirror or texdoc standalone.

[82] Martin Scharrer. The mwe package, 2012. CTAN Mirror or texdoc mwe.

[83] Walter Schmidt. Using common PostScript fonts with LaTeX, 2004. CTAN Mirror or texdoc psnfss.

[84] Rainer Schöpf, Bernd Raichle, and Chris Rowley. A new implementation of LaTeX's verbatim and verbatim* environments, 2001. CTAN Mirror or texdoc verbatim.

[85] Joachim Schrod. The rcs package, 1995. CTAN Mirror or texdoc rcs.

[86] Steven B. Segletes. The censor package, 2013. CTAN Mirror or texdoc censor.

[87] Wolfgang Skala. The pgfgantt package v4.0, 2013. CTAN Mirror or texdoc pgfgantt.

[88] Richard M. Smith. Microsoft Word bytes Tony Blair in the butt. http://www.computerbytesman.com/privacy/blair.htm, 2003. Accessed: 2014-11-10.

[89] R. Stepanyan. The bardiag package, 2003. CTAN Mirror or texdoc bardiag.

[90] D. P. Story. AcroTEX.Net eforms and insdljs documentation v2.0, 2013. CTAN Mirror or texdoc eforms.

[91] Nicola L. C. Talbot. Writing a LaTeX class file to produce a form. *The LaTeX Community's Know How Section*, November 2009. http://www.latex-community.org/know-how/latex/54-latex-document-classes/342-writing-a-latex-class-file-to-produce-a-form.

[92] Nicola L. C. Talbot. *LaTeX for Complete Novices*, volume 1 of *Dickimaw LaTeX Series*. Dickmaw Books, 2012. http://www.dickimaw-books.com/latex/novices/.

[93] Nicola L. C. Talbot. probsoln: creating problem sheets optionally with solutions, 2012. CTAN Mirror or texdoc probsoln.

[94] Nicola L. C. Talbot. User manual for datatool bundle, 2013. CTAN Mirror or texdoc datatool-user.

[95] Nicola L. C. Talbot. *Using LaTeX to Write a PhD Thesis*, volume 2 of *Dickimaw LaTeX Series*. Dickmaw Books, 2013. http://www.dickimaw-books.com/latex/thesis/.

[96] Nicola L. C. Talbot. pressrelease v1.0: typesetting press releases, 2014. CTAN Mirror or texdoc pressrelease.

[97] Nicola L. C. Talbot. Creating flow frames for posters, brochures or magazines using flowfram.sty version 1.16, 2014. CTAN Mirror or texdoc ffuserguide.

[98] Nicola L. C. Talbot. mfirstuc.sty v2.0: uppercasing first letter, 2015. CTAN Mirror or texdoc mfirstuc.

[99] Nicola L. C. Talbot. datetime2-english v1.01: English module for datetime2 package, 2015. CTAN Mirror or texdoc datetime2-english.

[100] Nicola L. C. Talbot. datetime2 v1.0: date and time formats, 2015. CTAN Mirror or texdoc datetime2.

[101] Till Tantau. The TikZ and PGF packages: Manual for version 3.0.0, 2013. CTAN Mirror or texdoc pgf.

[102] Till Tantau, Joseph Wright, and Vedran Miletić. The beamer class: user guide for version 3.33, 2013. CTAN Mirror or `texdoc beamer`.

[103] Jamal K. Tartir. LaTeX package jlabels, 2011. CTAN Mirror or `texdoc jlabels`.

[104] Bob Tennent. Alegreya fonts with LaTeX support, 2015. CTAN Mirror or `texdoc alegreya`.

[105] Hàn Thế Thành, Sebastian Rahtz, Hans Hagen, Hartmut Henkel, Pawel Jackowski, and Martin Schröder. The pdfTeX user manual, 2014. CTAN Mirror or `texdoc pdftex`.

[106] Paul A. Thompson. newlfm.cls: a new letter, fax, memo document class for LaTeX 2_ε, 2009. CTAN Mirror or `texdoc newlfm`.

[107] The TeX user group. `http://tug.org/`.

[108] UK list of TeX frequently asked questions (UK TeX FAQ). `http://www.tex.ac.uk/faq` or `texdoc faq`.

[109] Hideo Umeki. The geometry package v5.6, 2010. CTAN Mirror or `texdoc geometry`.

[110] Boris Veytsman. Printing envelopes and labels in LaTeX 2ε: envlab package user guide, 1997. CTAN Mirror or `texdoc elguide`.

[111] Nicola Vitacolonna. europecv: an unofficial class for european curricula, 2006. CTAN Mirror or `texdoc europecv`.

[112] Jürgen Vollmer. The rcsinfo package, 2005. CTAN Mirror or `texdoc rcsinfo`.

[113] Herbert Voß. pst-barcode: a PSTricks package for drawing barcodes, 2013. CTAN Mirror or `texdoc pst-barcode`.

[114] Hans-Christoph Wirth. formular.sty v1.0a, 2015. CTAN Mirror or `texdoc formular`.

[115] Yuan Xu. Drawing pie chart by using pgf-pie v0.2, 2012. CTAN Mirror or `texdoc pgf-pie`.

[116] Timothy Van Zandt. PSTricks PostScript macros for generic TeX, 2007. CTAN Mirror or `texdoc pst-user`.

Glossary

Active Character A character with category code 13. An active character is interpreted by TEX as a macro but it isn't escaped with a leading backslash. By default, there's only one active character ~ but some packages, such as babel, make other characters active. In particular the inputenc package makes all non-Latin or accented characters (such as é) active.

arara A Java application that automates the process of building a LATEX document. See also Volume 2 [95, §1.1.2]. URL: http://ctan.org/pkg/arara

Basic Latin Character One of the letters a, ..., z, A, ..., Z. (Part of the 7-bit ASCII set.) See also extended Latin character.

Boolean Key A key in a key=value list where the value may be either true or false. The value part may be omitted if it's true. For example, noheader=true and noheader both switch on the noheader setting.

Category Code The code assigned to a character that identifies its category. For example, the character "a" has the category code 11, which means it's a letter and can be used to form control words (see Volume 1 [92, §2.6]). The character "$" has the category code 3, which means it's the math-shift character. It's possible to change the category code of a character. For example, in `.cls` (class) or `.sty` (package) files, the "@" character has the category code 11 ("letter") so it can be used in control words (such as `\@for`) but outside of those files the "@" character has the catgory code 12 ("other"), so it can be used in the control symbol `\@` but not in any control words. For more details, see *The TEXbook* [45].

Comma-separated List A list where each item is separated by a comma. Leading or trailing spaces on either side of individual items in the list may or may not be ignored, depending on the context. If in doubt, err on the side of caution and remove spaces or suppress EOL terminators with the `%` comment character.

CSV Comma-separated variable.

CTAN The Comprehensive TEX Archive Network [18]. `http://mirror.ctan.org/`.

CV Curriculum vitæ (plural: curricula vitæ).

datatooltk A Java application that can be used to edit datatool databases. It can also be used to import data from CSV files, Excel spreadsheets, SQL databases and TEX files that can be loaded using probsoln's `\loadallproblems` command. This application can either be run in batch or GUI mode. Remember to add the `datatooltk/bin` directory to your system path if you want to invoke it from a command prompt. The way to do this varies according to your operating system. If you don't know how to do it, try doing a web search for "set path environment". URL: http://www.dickimaw-books.com/software/datatooltk/

datatooltk-gui A script that invokes datatooltk in GUI mode.

EOL End of line.

A character, or sequence of characters, signifying a line break (in the source code not in the resulting PDF) created by pressing the enter or return  key. The underlying character code is dependent on the operating system. For example, on Unix the EOL is indicated by the line feed (`\n` or `0x0A`) symbol (LF) whereas on Microsoft Windows the EOL is indicated by a combination of the carriage return (`\r` or `0x0D`) symbol (CR) and the LF symbol. The

category code for an EOL is 5, but usually TEX treats an EOL the same as the space symbol, unless it's immediately followed by another EOL, in which case a paragraph break created.

Extended Character A character that's outside of the 7-bit ASCII set. For example, an extended Latin character or any character from a non-Latin alphabet. Note that the numerical code representing a non-Latin character varies according to the file's input encoding. See also extended Latin character.

Extended Latin Character A character that's created by combining basic Latin characters to form ligatures (e.g. æ) or by applying diacritical marks to a basic Latin character or characters (e.g. á or ø). (See also extended character.) Note that the numerical code representing an extended Latin character varies according to the file's input encoding. For example, ï has codepoint `0x00EF` in the Unicode table, but has code 139 (`0x8B`) in the ASCII table.

`flowframtk` A Java application that can be used to construct frames for the flowfram package. URL: `http://www.dickimaw-books.com/software/flowframtk/`

GOW GNU On Windows [29]. `https://github.com/bmatzelle/gow`.

GUI Graphical user interface.

ICO Information Commissioner's Office [37]. http://ico.org.uk/.

Internal Command A command that contains an at character (@) in its name (such as \@for). These commands are intended for internal use in class or package files. You should avoid using them within your document, but if you really need to, you must first change the category code of the @ symbol to "letter" (via \makeatletter) and, after you've used the internal command, change the category code back to "other" (via \makeatother).

Java A language that can be deployed in a cross-platform environment, which means that as long as you have the Java runtime environment installed you can run a Java application regardless of the operating system used to develop the application. URL: https://java.com/

Key=value List A comma-separated list of ⟨*key*⟩=⟨*value*⟩ entries. Spaces on either side of the ⟨*key*⟩ and ⟨*value*⟩ are *usually* ignored. If ⟨*value*⟩ contains a comma (for example, the value is a list), the value must be enclosed in braces and unwanted spaces should be removed. (Recall from Volume 1 [92, §2] that unwanted space caused by the EOL character can be ignored using the % comment character.)

kpsewhich An application used for Kpathsea lookup and expansion.. URL: http://tug.org/texinfohtml/kpathsea.html [FAQ: Which tree to use]

make An application for automating the building of software or documents by specifying dependencies

Primitive A core command that isn't defined in terms of other commands. *The TEXbook* [45] describes primitives as low-level atomic operations that can't be decomposed into simpler functions. There are around 300 primitives provided by TEX. Other TEX formats, such as PDFTEX or XƎTEX provide additional primitives.

Shell Escape The ability to spawn processes during the document build. Since this is a security risk, it's usually disabled by default or a restricted version may be enabled that only allows a pre-set list of commands to be run. [FAQ: Spawning programs from (La)TeX: \write18]

Special Character A character that has a special meaning to TEX. The common special characters are: \ (the escape character, category code 0) { (group beginning, category code 1) } (group ending, category code 2) $ (math-shift, category code 3) & (alignment tab, category code 4) # (parameter, category code 6) ^ (superscript, category code 7) _ (subscript, category code 8) % (comment, category code 9)

and ~ (an active character). In some cases, the whitespace characters EOL (category code 5) and ␣ (category code 10) may also be considered special.

SQL Structured query language.

texdef A Perl script that displays the definition of (La)TEX commands. URL: http://ctan.org/pkg/texdef

Token Either a character (including special characters) or a control sequence.

TUG TEX User Group [107]. http://tug.org/.

UK FAQ UK List of TEX Frequently Asked Questions [108]. http://www.tex.ac.uk/faq.

UTF-8 Unicode Transformation Format — 8-bit.

This is an encoding that can represent every character in the Unicode character set (see also extended character and extended Latin character). If your source code contains Unicode characters, such as é (codepoint `0x00E9`) or ø (codepoint `0x00F8`), you must set both

your editor to UTF-8 and use the inputenc package with the `utf8` option in your document: `\usepackage[utf8]{inputenc}`. (Note that it's recommended that you also load fontenc if you use inputenc. See Volume 1 [92, §4.3.1].) XeLaTeX users should just load fontspec instead of inputenc and fontenc.

If you are using TeXworks, the status bar at the bottom of the window should show the encoding. You can change the editor encoding in TeXworks via the Edit→Preferences menu and select the "Editor" tab.

Whitespace A whitespace character is an invisible character that represents horizontal or vertical space in typography. TeX treats characters with a category code of 10 as a space. By default this includes the normal space character ␣ and the tab character  . The term "whitespace" may also include the EOL character.

Summary of Commands and Environments

Further information about the commands or environments summarised here may be obtained from: the package or class documentation via `texdoc` ⟨*name*⟩ (for class or package commands or environments); *The TEXbook* [45] (for TEX primitives); the LATEX 2ε source documentation via `texdoc source2e` (for LATEX Kernel commands or environments); the PDFTEX documentation via `texdoc pdftex` (for PDFTEX primitives); the ε-TEX documentation via `texdoc etex` (for ε-TEX primitives). If you use an up-to-date version of PDFLATEX you will be able to use all the primitives described here. Older versions of LATEX may not have some of the PDFTEX or ε-TEX primitives.

Symbols

!

Defined in: arara directive.

Logical not. [§1.2]

!

Defined in: graphicx package.

Used in `\resizebox` to maintain aspect ratio. [§10.3]

!

Defined in: xcolor package.

Used in colour specifications to indicate colour mixtures or percentages. [§12.1]

␣

A visual indication of a space in the code. When you type up the code, replace all instances of this symbol with a space via the space bar on your keyboard. [§1.0]

#⟨*digit*⟩

Defined in: LaTeX Kernel.

Replacement text for argument ⟨*digit*⟩. (See Volume 1 [92, §8].) [§2.1]

##⟨*digit*⟩

Defined in: LaTeX Kernel.

Replacement text for argument ⟨*digit*⟩ when the command definition is included in the definition of another command. [§2.7]

$

Defined in: LaTeX Kernel.

Switches in and out of in-line math mode. (See Volume 1 [92, §9.1].) [§2.2]

$

Defined in: tikz `calc` library.

When used with the tikz `calc` library this is used to indicate co-ordinate calculations. [§10.3]

%

Defined in: LaTeX Kernel.

Comment character used to ignore everything up to and including the EOL character in the source code. This is often used to suppress unwanted space caused by the EOL in source code. Sometimes comments are used to

provide information to applications that build your document, such as arara. See also Volume 1 [92, §2]. [§1.0]

`% arara:`

Instruction to arara indicating how to build the document. This is ignored if you are not using arara. With v4.0 long directives may have line breaks provided the continuation lines start with `% arara: -->` [§1.2]

&

Defined in: LaTeX Kernel.

Alignment tab. [§2.2]

&

Defined in: arara directive.

Non-short-circuit logical and. [§1.2]

&&

Defined in: arara directive.

Logical and. [§1.2]

'

Defined in: LaTeX Kernel.

Closing quote or apostrophe ’ symbol in text mode or prime symbol ′ in math mode. [§4.3]

''

Defined in: LaTeX Kernel.

Closing double quote ” symbol in text mode or double prime ″ in math mode. [§2.7]

--

Defined in: LaTeX Kernel.

En-dash – symbol. (Normally used for number ranges.) [§2.1]

--

Defined in: tikzpicture environment.

Used within the path specifications to indicate that a straight line should be drawn between two positions. [§12.1]

-->

Defined in: arara directive.

Continuation from previous line. [§1.2]

Defined in: LaTeX Kernel.

Em-dash — symbol. (Normally used to indicate omissions or interruptions or to highlight a parenthetical element.) [§2.9]

<

Defined in: tikzpicture environment.

Start arrow tip [§12.1]

>{⟨*decl*⟩}

Defined in: array package.

Used in tabular-like environment column specifiers before `l`, `r`, `c`, `p`, `m` or `b` to insert ⟨*decl*⟩ directly in front of the entry for that column. [§4.3]

>

Defined in: tikzpicture environment.

End arrow tip [§12.1]

@{⟨*text*⟩}

Defined in: LaTeX Kernel.

Used in the argument of tabular-like environments to specify text to insert between columns. Since ⟨*text*⟩ replaces the usual inter-column space, this may also

be used with an empty argument to simply suppress that space. [§10.3]

`\@`

Defined in: LaTeX Kernel.

Used when a sentence ends with a capital letter. This command should be placed after the letter and before the punctuation mark. [§3]

`\@afterheading`

Defined in: LaTeX Kernel.

Indicates a sectioning command has just been used, so the next paragraph should have its indentation suppressed. [§6.5]

`\@auxout`

Defined in: LaTeX Kernel.

Identifies the output stream for the document's auxiliary file. [§9.4]

`\@currenvir`

Defined in: LaTeX Kernel.

The name of the current environment. [§2.1]

`\@enumdepth`

Defined in: LaTeX Kernel.

TeX count register that keeps track of the current enumerate level. [§6.5]

`\@firstoftwo{`⟨*first*⟩`}{`⟨*second*⟩`}`

Defined in: LaTeX Kernel (internal command).

Just does the first argument and discards the second argument. See also `\@secondoftwo`. [§11.1]

`\@for`⟨*cs*⟩`:=`⟨*list*⟩`\do{`⟨*body*⟩`}`

Defined in: LaTeX Kernel (internal command).

Iterates through the comma-separated list and assigns the control sequence ⟨*cs*⟩ to the current item in the list so that it can be used as a placeholder in ⟨*body*⟩. The xfor package extends the functionality of this command, allowing you to terminate the loop at the end of the current iteration via `\@endfortrue`. [§2.7]

`\@ifundefined{`⟨*cs-name*⟩`}{`⟨*true-part*⟩`}{`⟨*false-part*⟩`}`

Defined in: LaTeX Kernel (internal command).

Determines if the control sequence given by ⟨*cs-name*⟩ (without the leading backslash) is undefined (or `\relax`). [§2.1]

`\@namedef{`⟨*cs-name*⟩`}`⟨*arg-syntax*⟩`{`⟨*definition*⟩`}`

Defined in: LaTeX Kernel (internal command).

Defines the control sequence given by ⟨*cs-name*⟩ (without the leading backslash). [§2.1]

`\@nameuse{`⟨*cs-name*⟩`}`

Defined in: LaTeX Kernel (internal command).

Uses the control sequence given by ⟨*cs-name*⟩ (without the leading backslash). [§2.1]

`\@secondoftwo{`⟨*first*⟩`}{`⟨*second*⟩`}`

Defined in: LaTeX Kernel (internal command).

Just does the second argument and discards the first argument. See also `\@firstoftwo`. [§11.1]

`\@toodeep`

Defined in: LaTeX Kernel.

Generates a "Too deeply nested" error. [§6.5]

`[`

Defined in: LaTeX Kernel.

Open delimiter of an optional argument. (See Volume 1 [92, §2.8.2].) [§1.0]

`\&`

Defined in: LaTeX Kernel.

Ampersand & symbol. [§2.2]

`\`

Defined in: LaTeX Kernel.

Escape character indicating a command. (See Volume 1 [92, §2.6].) [§3]

`\$`

Defined in: LaTeX Kernel.

Dollar $ symbol. [§2.2]

`\#`

Defined in: LaTeX Kernel.

Hash # symbol. [§2.2]

`\%`

Defined in: LaTeX Kernel.

Percent % symbol. [§2.2]

`\"{`⟨*c*⟩`}`

Defined in: LaTeX Kernel.

Umlaut over ⟨*c*⟩. Example: `\"{o}` produces ö. [§6.3]

`\,`

Defined in: LaTeX Kernel.

Thin space. [§4.3]

\[

Defined in: LaTeX Kernel.

Starts an unnumbered single-line of displayed maths. [§9.3]

\\

Defined in: LaTeX Kernel.

Starts a new row in environments that have a concept of rows rather than paragraphs (such as the tabular-style environments). This may have a starred version and/or an optional argument. [§2.7]

\'{⟨*c*⟩}

Defined in: LaTeX Kernel.

Acute accent over ⟨*c*⟩. Example: \'{o} produces ó. [§5.1]

\␣

Defined in: LaTeX Kernel.

(Backslash followed by space character.) Horizontal spacing command. [§2.7]

\]

Defined in: LaTeX Kernel.

Ends an unnumbered single-line of displayed maths. [§9.3]

_

Defined in: LaTeX Kernel.

Underscore _ symbol. [§2.2]

\{

Defined in: LaTeX Kernel.

Left brace { character. In math mode may be used as a delimiter. [§2.2]

\}

Defined in: LaTeX Kernel.

Right brace } character. In math mode may be used as a delimiter. [§2.2]

]

Defined in: LaTeX Kernel.

Closing delimiter of an optional argument. (See Volume 1 [92, §2.8.2].) [§1.0]

^{⟨*maths*⟩}

Defined in: LaTeX Kernel (Math Mode).

Displays its argument as a superscript. [§2.2]

_{⟨*maths*⟩}

Defined in: LaTeX Kernel (Math Mode).

Displays its argument as a subscript. [§2.2]

`

Defined in: LaTeX Kernel.

Open quote ‘ symbol. [§4.3]

``

Defined in: LaTeX Kernel.

Open double quote “ symbol. [§2.7]

{

Defined in: LaTeX Kernel.

Marks the beginning of a group. (See Volume 1 [92, §2.7].) [§1.0]

|

Defined in: arara directive.

Non-short-circuit logical or. [§1.2]

||

Defined in: arara directive.

Logical or. [§1.2]

}

Defined in: LaTeX Kernel.

Marks the end of a group. (See Volume 1 [92, §2.7].) [§1.0]

~

Defined in: LaTeX Kernel.

Unbreakable space. (See Volume 1 [92, §4.3].) [§1.2]

A

`\begin{about}`

Defined in: pressrelease class.

For use within the pressrelease environment, this environment contains information about the company. [§6.2]

`\accountdata`

Defined in: isodoc class.

Generates a table containing the account information needed to pay the invoice. [§4.1]

`\added[`⟨*options*⟩`]{`⟨*text*⟩`}`

Defined in: changes package.

Indicates that the given text has been added. [§13.1]

`\addplot[`⟨*path options*⟩`]` ⟨*plot specs*⟩;

Defined in: pgfplots package.

Adds a plot to the current image. [§12.5]

`\addpoints`

Defined in: exam class.

Enable the point-totalling commands. [§9.1]

`\address{`⟨*text*⟩`}`

Defined in: letter class.

Specifies the sender's address. [§3.1]

`\addrfrom{`⟨*address*⟩`}`

Defined in: newlfm class.

Specifies the sender's address. [§3.3]

`\addrto{`⟨*address*⟩`}`

Defined in: newlfm class.

Specifies the recipient's address. [§3.3]

`\AddToBackground{`⟨*page-number*⟩`}{`⟨*picture-code*⟩`}`

Defined in: leaflet class (preamble only).

Indicates picture code to place on the page given by ⟨*page-number*⟩. The starred version refers to the sheet number instead. [§10.3]

`\addtolength{`⟨*register*⟩`}{`⟨*dimension*⟩`}`

Defined in: LaTeX Kernel.

Adds ⟨*dimension*⟩ to the value of the given length register. [§6.5]

`\advance`⟨*register*⟩ by ⟨*value*⟩

Defined in: TeX primitive.

Increments the value stored in ⟨*register*⟩ by ⟨*value*⟩. The by keyword may be omitted. [§2.1]

`\Alph{`⟨*counter*⟩`}`

Defined in: LaTeX Kernel.

Displays counter value as an upper case letter. (A, B, C, ..., Z) [§6.5]

`\alph{`⟨*counter*⟩`}`

Defined in: LaTeX Kernel.

Displays counter value as a lower case letter. (a, b, c, ..., z) [§6.5]

\and

Defined in: LaTeX Kernel.

Used to separate authors in \author [§8.0]

\appenddynamiccontents{⟨*id*⟩}{⟨*text*⟩}

Defined in: flowfram package.

Appends the given text to the contents of a dynamic frame. [§10.5]

\appto⟨*cs*⟩{⟨*code*⟩}

Defined in: etoolbox package.

Appends ⟨*code*⟩ to the definition of the control sequence ⟨*cs*⟩. Use \gappto for a global assignment. [§2.1]

\arabic{⟨*counter*⟩}

Defined in: LaTeX Kernel.

Displays counter value as an Arabic number. (1, 2, 3, ...) [§6.5]

\begin{Argumentation}

Defined in: minutes package.

A list like environment to format an argument. [§6.3]

\AtBeginDocument{⟨*code*⟩}

Defined in: LaTeX Kernel.

Specifies code that should be performed at the beginning of the document environment. This command has a cumulative effect. [§7.3]

\author{⟨*name*⟩}

Defined in: Most classes that have the concept of a title page.

Specifies the document author (or authors). This command doesn't

display any text so may be used in the preamble, but if it's not in the preamble it must be placed before `\maketitle`. Some classes, such as beamer, provide an optional argument for this command. [§2.3]

`\begin{axis}[`⟨*options*⟩`]`

Defined in: pgfplots package.

Create a plot with normal Cartesian axes. [§12.5]

B

`\backgroundsetup{`⟨*options*⟩`}`

Defined in: background package.

Sets the background options. [§6.4]

`\baselineskip`

Defined in: LaTeX Kernel.

A TeX primitive that stores the minimum space from the bottom of one line to the bottom of the next line in a paragraph. This is recalculated whenever the font changes. [§3.6]

`\bcbar[`⟨*options*⟩`]{`⟨*value*⟩`}`

Defined in: bchart package.

For use within the bchart environment, this draws a bar with the given value. [§12.3]

`\bcfontstyle`

Defined in: bchart package.

The font declaration used for the bar chart labels. [§12.3]

`\begin{bchart}[`⟨*options*⟩`]`

Defined in: bchart package.

Creates a bar chart. [§12.3]

`\bclabel{`⟨*text*⟩`}`

Defined in: bchart package.

For use within the bchart environment, this inserts a "free" label at the current position within the chart. [§12.3]

`\bcskip{`⟨*length*⟩`}`

Defined in: bchart package.

For use within the bchart environment, this inserts a gap of the given length. [§12.3]

`\bcxlabel{`⟨*text*⟩`}`

Defined in: bchart package.

For use within the bchart environment, this sets the x-axis label. [§12.3]

`\begin{`⟨*env-name*⟩`}[`⟨*env-option*⟩`]{`⟨*env-arg-1*⟩`}...{`⟨*env-arg-n*⟩`}`

Defined in: LaTeX Kernel.

Starts an environment. (Must have a matching `\end`. See Volume 1 [92, §2.15].) [§1.2]

`\bfseries`

Defined in: LaTeX Kernel.

Switches to the bold weight in the current font family. See Volume 1 [92, §4.5.1]. [§2.7]

`\BgThisPage`

Defined in: background package.

Indicates the background should be displayed on the current page. (For use with the `some` pages option.) [§6.4]

`\bibentry{`⟨*key*⟩`}`

Defined in: bibentry package.

Prints the bibliographic entry for citation ⟨*key*⟩. [§5.2]

`\bibitem[`⟨*tag*⟩`]{`⟨*key*⟩`}`

Defined in: LaTeX Kernel.

Indicates the start of a new reference in the bibliography. May only be used inside the contents of thebibliography environment [§5.2]

`\bibliography{`⟨*bib-list*⟩`}`

Defined in: LaTeX Kernel.

Inputs the `.bbl` file (if it exists) and identifies the name(s) of the bibliography database files where the citations are defined. [§1.2]

`\bibliographystyle{`⟨*style-name*⟩`}`

Defined in: LaTeX Kernel.

Specifies the bibliography style to be used by bibtex. [§1.2]

`\BigCircle`

Defined in: ifsym package with `geometry` option.

Produces a large open circle. [§11.1]

`\bigskip`

Defined in: LaTeX Kernel.

Inserts a large vertical space. The size is given by the length `\bigskipamount`. [§10.3]

`\blackout{`⟨*text*⟩`}`

Defined in: censor package.

Redacts ⟨*text*⟩, which can consist of one or more paragraphs. [§6.4]

`\begin{block}{`⟨*title*⟩`}`

Defined in: beamer class.

Puts its contents in a titled block. [§8.0]

`\bonuspart[`⟨*points*⟩`]`

Defined in: exam class.

Like `\part` but indicates the points are bonus marks. [§9.1]

`\bonusquestion[`⟨*points*⟩`]`

Defined in: exam class.

Like `\question` but indicates the points are bonus marks. [§9.1]

`\bonussubpart[`⟨*points*⟩`]`

Defined in: exam class.

Like `\subpart` but indicates the points are bonus marks. [§9.1]

`\bonussubsubpart[`⟨*points*⟩`]`

Defined in: exam class.

Like `\subsubpart` but indicates the points are bonus marks. [§9.1]

`\bonussum`

Defined in: exsheets package.

Displays the total number of bonus points. (Requires two LaTeX runs to ensure it's up to date.) The starred version omits the unit. [§9.2]

`\bonustitledquestion{`⟨*title*⟩`}[`⟨*points*⟩`]`

Defined in: exam class.

Like `\titledquestion` but indicates the points are bonus marks. [§9.1]

`\boolean{`⟨*name*⟩`}`

Defined in: ifthen package.

May be used in the first argument of `\ifthenelse` to test the state of the named boolean variable. [§9.3]

`\boolfalse{`⟨*name*⟩`}`

Defined in: etoolbox package.

Sets the given boolean variable's state to `false`. [§9.4]

`\booltrue{`⟨*name*⟩`}`

Defined in: etoolbox package.

Sets the given boolean variable's state to `true`. [§9.4]

`\bottomrule[`⟨*wd*⟩`]`

Defined in: booktabs package.

Horizontal rule for the bottom of a tabular environment. [§2.6]

`\bracketedpoints`

Defined in: exam class.

Changes the points format to use square brackets instead of the default parentheses. [§9.1]

`\breakforeach`

Defined in: pgffor package.

When used inside the ⟨*body*⟩ part of `\foreach`, the loop will be terminated after the completion of the current iteration. [§3]

C

`\caption[`⟨*short-caption*⟩`]`
`{`⟨*caption-text*⟩`}`

Defined in: LaTeX Kernel.

Inserts the caption for a float such as a figure or table. [§2.6]

`\cc{`⟨*cc info*⟩`}`

Defined in: Classes that define the letter environment.

Used to indicate the additional recipients of the letter. [§3.1]

`\cc{`⟨*names*⟩`}`

Defined in: minutes package.

Specifies the distribution list. [§6.3]

`\cclist{`⟨*text*⟩`}`

Defined in: newlfm class.

Sets distribution list. [§3.3]

`\censor{`⟨*text*⟩`}`

Defined in: censor package.

Redacts ⟨*text*⟩ by replacing the text with a filled black rectangle of the same size. [§6.4]

`\censor*{`⟨*size*⟩`}`

Defined in: censor package.

Alternative to `\censor` when the sensitive text should be omitted from the document source. [§6.4]

`\censorbox[`⟨*declarations*⟩`]`
`{`⟨*contents*⟩`}`

Defined in: censor package.

Redacts ⟨*contents*⟩ (a box, such as a tabular environment) by replacing the contents with a filled black rectangle of the same size. [§6.4]

`\censorbox*[`⟨*declarations*⟩`]`
`{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*depth*⟩`}`

Defined in: censor package.

Alternative to `\censorbox` when the sensitive text should be omitted from the document source. [§6.4]

`\begin{center}`

Defined in: LaTeX Kernel.

Centres its contents and places a small vertical gap above and below the environment. This gap can interfere with the vertical spacing in figure or table environments, so the use of center within a float is considered inappropriate. It can, however, be used outside a float where the gap may be useful to separate its

contents from the previous and following paragraphs. [§3.0]

`\centering`

Defined in: LaTeX Kernel.

Switches the paragraph alignment to centred. (See Volume 1 [92, §2.12].) [§2.6]

`\chair{`⟨*name*⟩`}`

Defined in: meetingmins class.

Sets the name of the chair (for use within `\setmembers` and `\setpresent`). [§6.3]

changed(⟨*ref*⟩)

Defined in: arara directive.

Evaluates to true if the given file has changed. The argument ⟨*ref*⟩ may either be a string "⟨*extension*⟩" which indicates the file extension or a file reference `toFile("filename")`. [§1.2]

`\chapter[`⟨*short-title*⟩`]{`⟨*title*⟩`}`

Defined in: Book-style classes (such as scrbook or scrreprt) that have the concept of chapters.

Inserts a chapter heading. [§6.5]

`\CheckBox[`⟨*options*⟩`]{`⟨*label*⟩`}`

Defined in: hyperref package.

A check box (for use within the Form environment.) [§11.2]

`\begin{checkboxes}`

Defined in: exam class.

A list environment with checkbox choices as the items in the list. Items are specified via `\choice`. [§9.1]

`\CheckedBox`

Defined in: wasysym package.

Produces a square with a tick in it ☑. [§11.1]

`\choice`

Defined in: exam class.

For use in one of the choices or checkboxes environments, this commands starts a new choice. Use `\CorrectChoice` instead of `\choice` to indicate the correct choice. [§9.1]

`\ChoiceMenu[`⟨*options*⟩`]{`⟨*label*⟩`}{`⟨*choices*⟩`}`

Defined in: hyperref package.

A list menu, popup menu, combo menu or set of radio buttons (for use within the Form environment.) [§11.2]

`\begin{choices}`

Defined in: exam class.

A list environment with labelled choices as the items in the list. Items are specified via `\choice`. [§9.1]

`\circle{`⟨*diameter*⟩`}`

Defined in: LaTeX Kernel.

For use in the argument of `\put`, this command draws a circle with the given diameter (specified in terms of `\unitlength`. The starred version fills the circle. [§10.1]

`\cite[`⟨*text*⟩`]{`⟨*key list*⟩`}`

Defined in: LaTeX Kernel.

Inserts the citation markers of each reference identified in the key list. A second run is required to ensure the reference is correct.

Some packages redefine `\cite` to have two optional arguments. (See Volume 2 [95, §5].) [§1.2]

`\ClassError{`⟨*class-name*⟩`}{`⟨*error-message*⟩`}{`⟨*help-message*⟩`}`

Defined in: LaTeX Kernel.

Displays an error message for the given class. [§11.1]

`\clearpage`

Defined in: LaTeX Kernel.

Inserts a page break and processes any unprocessed floats. [§3]

`\closeline{`⟨*text*⟩`}`

Defined in: newlfm class.

The closing text. [§3.3]

`\closing{`⟨*sign-off text*⟩`}`

Defined in: Classes that define the letter environment.

Typesets the signing off text at the end of the letter. [§3.1]

`\Collect@Body`⟨*cs*⟩

Defined in: environ package.

As amsmath's `\collect@body` but allows paragraph breaks within the environment. [§11.1]

`\collect@body`⟨*cs*⟩

Defined in: amsmath package.

Collects the contents of the current environment and passes it to the command ⟨*cs*⟩ which should take an argument. The environment contents may not contain any paragraph breaks. [§11.1]

`\begin{collectinmacro}`
`{`⟨*macro*⟩`}{`⟨*before*⟩`}{`⟨*after*⟩`}`

Defined in: collect package.

Collects the body of the environment and stores ⟨*before*⟩⟨*body*⟩⟨*after*⟩ in the given macro. [§11.1]

`\color[`⟨*model*⟩`]{`⟨*specs*⟩`}`

Defined in: color and xcolor packages.

A declaration that switches the current foreground colour to the given specification. [§10.3]

`\colorbox[`⟨*model*⟩`]{`⟨*specs*⟩`}`
`{`⟨*text*⟩`}`

Defined in: color package.

Produces a box containing ⟨*text*⟩ with the background given by ⟨*specs*⟩ for the given colour model. [§10.5]

`\computeleftedgeeven{`⟨*cs*⟩`}`

Defined in: flowfram package.

Gets the co-ordinate of the left edge of the even numbered pages relative to the typeblock and stores it in the supplied control sequence. [§11.0]

`\computeleftedgeodd{`⟨*cs*⟩`}`

Defined in: flowfram package.

Gets the co-ordinate of the left edge of the odd numbered pages relative to the typeblock and stores it in the supplied control sequence. [§11.0]

`\Contra`

Defined in: minutes package.

For use within the Argumentation environment, this indicates the start of an important item against the argument. [§6.3]

`\contra`

Defined in: minutes package.

For use within the Argumentation environment, this indicates the start of an item against the argument. [§6.3]

`\CorrectChoice`

Defined in: exam class.

Used instead of `\choice` to indicate the correct choice. [§9.1]

`\correctitem`

Defined in: probsoln package.

For use within the textenum, this command may be used in place of `\item` to indicate a correct choice. If the solutions aren't displayed this command behaves the same as `\item`. [§9.3]

`\correctitemformat{`⟨*marker*⟩`}`

Defined in: probsoln package.

The format used by `\correctitem`. [§9.3]

`\cos`

Defined in: LaTeX Kernel (Math Mode).

Typesets cos function name. [§9.1]

`\begin{coverpages}`

Defined in: exam class.

Contains material to go before the start of the exam. [§9.1]

`\csappto{`⟨*cs-name*⟩`}{`⟨*code*⟩`}`

Defined in: etoolbox package.

As \appto but requires the name (without the leading backslash) of the control sequence. Use \csgappto for a global assignment. [§2.1]

\csdef{⟨*cs-name*⟩}⟨*arg-syntax*⟩{⟨*definition*⟩}

Defined in: etoolbox package.

This is analogous to \def except that the name of the control sequence (without the initial backslash) is supplied. [§2.1]

\cseappto{⟨*cs-name*⟩}{⟨*code*⟩}

Defined in: etoolbox package.

As \eappto but requires the name (without the leading backslash) of the control sequence. Use \csxappto for a global assignment. [§2.1]

\csedef{⟨*cs-name*⟩}⟨*arg-syntax*⟩{⟨*definition*⟩}

Defined in: etoolbox package.

This is analogous to \edef except that the name of the control sequence (without the initial backslash) is supplied. [§2.1]

\csepreto{⟨*cs-name*⟩}{⟨*code*⟩}

Defined in: etoolbox package.

As \epreto but requires the name (without the leading backslash) of the control sequence. Use \csxpreto for a global assignment. [§2.1]

\csgappto{⟨*cs-name*⟩}{⟨*code*⟩}

Defined in: etoolbox package.

As \gappto but requires the name (without the leading backslash) of the control sequence. [§2.1]

`\csgdef{`⟨*cs-name*⟩`}`⟨*arg-syntax*⟩ `{`⟨*definition*⟩`}`

Defined in: etoolbox package.

This is analogous to `\gdef` except that the name of the control sequence (without the initial backslash) is supplied. [§2.1]

`\csgpreto{`⟨*cs-name*⟩`}{`⟨*code*⟩`}`

Defined in: etoolbox package.

Global version of `\cspreto`. [§2.1]

`\cslet{`⟨*cs-name*⟩`}{`⟨*cs*⟩`}`

Defined in: etoolbox package.

Analogous to `\let` except that the name of the control sequence (without the initial backslash) is supplied for the first argument. [§2.1]

`\csletcs{`⟨*new cs-name*⟩`}{`⟨*org cs-name*⟩`}`

Defined in: etoolbox package.

Analogous to `\let` except that the names of the control sequences (without the initial backslash) are supplied. [§2.1]

`\csname` ⟨*cs-name*⟩`\endcsname`

Defined in: TEX primitive.

Expands to the control sequence whose name (without the leading backslash) is given by ⟨*cs-name*⟩. If the control sequence isn't already defined, TEX will first define it to `\relax` before using it. [§2.1]

`\cspreto{`⟨*cs-name*⟩`}{`⟨*code*⟩`}`

Defined in: etoolbox package.

As `\preto` but requires the name (without the leading backslash) of

the control sequence. Use \csgpreto for a global assignment. [§2.1]

\csuse{⟨*cs-name*⟩}

Defined in: etoolbox.

Executes the control sequence whose name (without the leading backslash) is given by ⟨*cs-name*⟩. If the command doesn't exist, this expands to an empty string. [§2.1]

\csvloop[⟨*auxiliary-commands*⟩] {⟨*list*⟩}

Defined in: etextools package.

Iterates through the comma-separated list using a handler macro provided in the optional argument. If none is provided, the command \do is used. The list may explicitly be a comma-separated list or it may be a macro that expands to a list, unless the starred form is used. There are other variants to this command not described here. See the etextools documentation for further details. [§2.7]

\csxappto{⟨*cs-name*⟩}{⟨*code*⟩}

Defined in: etoolbox package.

As \xappto but requires the name (without the leading backslash) of the control sequence. [§2.1]

\csxdef{⟨*cs-name*⟩}⟨*arg-syntax*⟩ {⟨*definition*⟩}

Defined in: etoolbox package.

This is analogous to \xdef except that the name of the control sequence (without the initial backslash) is supplied. [§2.1]

`\csxpreto{`⟨*cs-name*⟩`}{`⟨*code*⟩`}`

Defined in: etoolbox package.

Global version of `\csepreto`. [§2.1]

`\CurrentOption`

Defined in: LaTeX Kernel.

Refers to the current option when used in the argument of `\DeclareOption*` [§11.1]

`\CutLine{`⟨*page-number*⟩`}`

Defined in: leaflet class (preamble only).

Indicates that a cut line should be drawn to the left of the given page number. The starred version only draws a dotted line. The unstarred version draws a dotted line and a pair of scissors. [§10.3]

`\begin{cv}{`⟨*heading*⟩`}`

Defined in: currvita package.

The body of the CV. [§5.1]

`\begin{cvlist}{`⟨*heading*⟩`}`

Defined in: currvita package.

A headed list-like structure for use within the cv environment. [§5.1]

`\cvplace{`⟨*location*⟩`}`

Defined in: currvita package.

Specifies the location where the CV was written. [§5.1]

D

`\dashbox{`⟨*dash-length*⟩`}(`⟨*w*⟩`,`⟨*h*⟩`)[`⟨*align*⟩`]{`⟨*text*⟩`}`

Defined in: picture environment.

Similar to `\framebox` but produces a dashed frame. [§10.1]

`\date{`⟨*text*⟩`}`

Defined in: Most classes that have the concept of a title page.

Specifies the document date. This command doesn't display any text so may be used in the preamble, but if it's not in the preamble it must be placed before \maketitle. If omitted, most classes assume the current date. Some classes, such as beamer, provide an optional argument for this command. [§5.1]

\dateset{⟨*date*⟩}

Defined in: newlfm class.

Sets the date. [§3.3]

\day

Defined in: TEX primitive.

The current day of the month. [§7.2]

\DBIBcitekey

Defined in: databib package.

For use with \DTLforeachbibentry or \gDTLforeachbibentry, this expands to the name of the current cite key. [§5.2]

\DBIBentrytype

Defined in: databib package.

For use with \DTLforeachbibentry or \gDTLforeachbibentry, this expands to the name of the current entry type (for example, book). [§5.2]

\decision{⟨*theme*⟩}{⟨*short-description*⟩}[⟨*long-description*⟩]

Defined in: minutes package.

Specifies a decision, which is added to the list of decisions. [§6.3]

\decision*{⟨*short-description*⟩}[⟨*long-description*⟩]

Defined in: minutes package.

Specifies a decision, which isn't added to the list of decisions. [§6.3]

`\decisiontheme{`⟨*theme*⟩`}{`⟨*title*⟩`}`

Defined in: minutes package.

Defines a decision theme which will be added to the list of decisions. [§6.3]

`\DeclareOption{`⟨*option*⟩`}{`⟨*code*⟩`}`

Defined in: LaTeX Kernel.

Declares an option for a class or package. [§11.1]

`\DeclareOption*{`⟨*code*⟩`}`

Defined in: LaTeX Kernel.

Declares the code for an unknown option. You can refer to the option within ⟨*code*⟩ using `\CurrentOption`. [§11.1]

`\def`⟨*cs*⟩⟨*arg-syntax*⟩`{`⟨*definition*⟩`}`

Defined in: TeX primitive.

This locally defines the command ⟨*cs*⟩ that has the given syntax. Use `\gdef` for global definitions. [§2.1]

`\DefaultHeightofCheckBox`

Defined in: hyperref package.

The default height of check boxes. [§11.2]

`\DefaultHeightofChoiceMenu`

Defined in: hyperref package.

The default height of choice boxes. [§11.2]

`\DefaultHeightofReset`

Defined in: hyperref package.

The default height of the reset button. [§11.2]

`\DefaultHeightofSubmit`

Defined in: hyperref package.

The default height of the submit button. [§11.2]

`\DefaultHeightofText`

Defined in: hyperref package.

The default height of single-lined text fields. [§11.2]

`\DefaultHeightofTextMultiline`

Defined in: hyperref package.

The default height of multi-lined text fields. [§11.2]

`\DefaultWidthofCheckBox`

Defined in: hyperref package.

The default width of check boxes. [§11.2]

`\DefaultWidthofChoiceMenu`

Defined in: hyperref package.

The default width of choice boxes. [§11.2]

`\DefaultWidthofReset`

Defined in: hyperref package.

The default width of the reset button. [§11.2]

`\DefaultWidthofSubmit`

Defined in: hyperref package.

The default width of the submit button. [§11.2]

`\DefaultWidthofText`

Defined in: hyperref package.

The default width of text fields. [§11.2]

`\definechangesauthor[`⟨*options*⟩`]{`⟨*id*⟩`}`

Defined in: changes package.

Defines a tracked author. [§13.1]

`\begin{defproblem}[`⟨*n*⟩`][`⟨*default-args*⟩`]{`⟨*label*⟩`}[`⟨*option*⟩`]`

Defined in: probsoln package.

Defines a new problem. (The contents may include the onlysolution environment for the solution.) [§9.3]

`\deleted[`⟨*options*⟩`]{`⟨*text*⟩`}`

Defined in: changes package.

Indicates that the given text has been deleted. [§13.1]

`\Delta`

Defined in: LaTeX Kernel (Math Mode).

Greek upper case delta Δ. [§9.3]

`\descfont`

Defined in: leaflet class.

The font declaration used by the item labels in the description environment. [§10.3]

`\begin{description}`

Defined in: Most class files.

Labelled list. [§10.3]

`\dimexpr`⟨*dimension expression*⟩

Defined in: ε-TeX primitive.

Expands to the value given by the dimension expression. [§2.1]

`\ding{`⟨*n*⟩`}`

Defined in: pifont package.

Inserts PostScript ZapfDingbats character with code ⟨*n*⟩, which must be an integer. [§2.9]

`\begin{dinglist}{`⟨*number*⟩`}`

Defined in: pifont package.

A list where the item marker is given by character ⟨*number*⟩ in the Zapf Dingbats font. [§10.5]

`\Discount{`⟨*description*⟩`}{`⟨*amount*⟩`}`

Defined in: invoice package.

For use within the invoice environment, this command is used to specify a discount. [§4.2]

`\divide`⟨*register*⟩ by ⟨*value*⟩

Defined in: TEX primitive.

Divides the value stored in ⟨*register*⟩ by ⟨*value*⟩. The by keyword may be omitted. [§2.1]

`\do{`⟨*item*⟩`}`

Defined in: etoolbox package.

Handler macro used by commands like `\docsvlist` and `\csvloop`. The argument is the item in the current iteration of the list. [§2.7]

`\docsvlist{`⟨*item1,item2,...*⟩`}`

Defined in: etoolbox package.

Loops over the given comma-separated list and executes the command `\do` for every item in the list, using the item as the argument of `\do`. It's up to the user to define `\do` as appropriate. [§2.7]

`\begin{document}`

Defined in: LATEX Kernel.

The body of the document. [§6.3]

`\documentclass[`⟨*option-list*⟩`]`

{⟨*class-name*⟩}

Defined in: LaTeX Kernel.

Loads the document class file, which sets up the type of document you wish to write. (See Volume 1 [92, §4].) [§1.0]

\doforrandN{⟨*n*⟩}{⟨*cs*⟩}{⟨*list*⟩} {⟨*body*⟩}

Defined in: probsoln package.

Iterates over a randomly selected ⟨*n*⟩ items in a comma-separated list. At each iteration, ⟨*cs*⟩ is defined to be the currently selected item and ⟨*body*⟩ is performed. [§9.3]

\dolistcsloop{⟨*list-csname*⟩}

Defined in: etoolbox package.

Similar to \dolistloop except the name (without the preceding backslash) of the list control sequence is used. [§2.7]

\dolistloop{⟨*list-cs*⟩}

Defined in: etoolbox package.

Iterates over all items in the list macro ⟨*list-cs*⟩ and performs \do{⟨*item*⟩} for each item. [§2.7]

\dotfill

Defined in: LaTeX Kernel.

Fills the remaining space with a dotted line. [§10.3]

\draw ⟨*specification*⟩;

Defined in: tikz package.

For use within the tikzpicture environment. Draws the path with the given specification using the current path stroking operations. [§12.1]

`\droppoints`

Defined in: exam class.

Prints the question points. This command should only occur at the end of a paragraph or between paragraphs. (Used with `\pointsdroppedatright`.) [§9.1]

`\DTLabs{`⟨*cs*⟩`}{`⟨*number*⟩`}`

Defined in: datatool package.

Computes the absolute value of ⟨*number*⟩ and stores the result in ⟨*cs*⟩, which must be a control sequence. The number should be formatted according to the current locale (set via `\DTLsetnumberchars`) and may optionally be prefixed with a known currency identifier. See the datatool documentation for further details. [§2.1]

`\dtlabs{`⟨*cs*⟩`}{`⟨*number*⟩`}`

Defined in: datatool package.

Computes the absolute value of ⟨*number*⟩ and stores the result in ⟨*cs*⟩, which must be a control sequence. The number must be a plain decimal number (a full stop as a decimal point, no number grouping and no currency prefix). [§2.1]

`\DTLadd{`⟨*cs*⟩`}{`⟨*number1*⟩`}{`⟨*number2*⟩`}`

Defined in: datatool package.

Adds ⟨*number1*⟩ to ⟨*number2*⟩ and stores the result in ⟨*cs*⟩, which must be a control sequence. The number should be formatted according to the current locale (set via `\DTLsetnumberchars`) and may optionally be prefixed with a

known currency identifier. See the datatool documentation for further details. [§2.1]

`\dtladd{`⟨*cs*⟩`}{`⟨*number1*⟩`}{`⟨*number2*⟩`}`

Defined in: datatool package.

Adds ⟨*number1*⟩ to ⟨*number2*⟩ and stores the result in ⟨*cs*⟩, which must be a control sequence. The number must be a plain decimal number (a full stop as a decimal point, no number grouping and no currency prefix). [§2.1]

`\DTLassign{`⟨*db-name*⟩`}{`⟨*row-idx*⟩`}{`⟨*assign-list*⟩`}`

Defined in: datatool package.

Applies the assignment list, which has the same format as for `\DTLforeach` and `\DTLforeach*`, to the given row. Row indexes start from 1. [§2.8]

`\DTLassignfirstmatch{`⟨*db-name*⟩`}{`⟨*col-label*⟩`}{`⟨*value*⟩`}{`⟨*assign-list*⟩`}`

Defined in: datatool package.

Finds the first row in the database ⟨*db-name*⟩ where entry in the column identified by the label ⟨*col-label*⟩ matches ⟨*value*⟩ and applies the assignment list, which has the same format as for `\DTLforeach` and `\DTLforeach*`. Note that no expansion is performed on ⟨*value*⟩. See also `\xDTLassignfirstmatch`. [§2.8]

`\DTLbaratbegintikz`

Defined in: databar package.

Hook used by `\DTLbarchart` or `\DTLmultibarchart` at the start of

the tikzpicture environment. [§12.3]

\DTLbaratendtikz

Defined in: databar package.

Hook used by \DTLbarchart or \DTLmultibarchart at the start of the tikzpicture environment. [§12.3]

\DTLbarchart[⟨*condition*⟩] {⟨*settings*⟩}{⟨*db-name*⟩}{⟨*assign-list*⟩}

Defined in: databar package.

Generates a bar chart from a column of the datatool database called ⟨*db-name*⟩. [§12.3]

\DTLbarchartlength

Defined in: databar package.

A length register storing the total length of the y-axis. [§12.3]

\DTLbarlabeloffset

Defined in: databar package.

A length register storing the offset between the x-axis and the lower bar label. [§12.3]

\DTLbaroutlinecolor

Defined in: databar package.

A macro that expands to the colour used to draw the bar outlines. [§12.3]

\DTLbaroutlinewidth

Defined in: databar package.

A length register that stores the width of the bar outline. A zero width indicates that the outline shouldn't be drawn. [§12.3]

\DTLbarwidth

Defined in: databar package.

A length register storing the width of each bar. [§12.3]

`\DTLbibfield{`⟨*field-name*⟩`}`

Defined in: databib package.

(For use within the final argument of `\DTLforeachbibentry` or `\gDTLforeachbibentry`.) This command displays the value of entry in the column whose label is given by ⟨*field-name*⟩. [§5.2]

`\DTLbibfieldlet{`⟨*cs*⟩`}{`⟨*field-name*⟩`}`

Defined in: databib package.

(For use within the final argument of `\DTLforeachbibentry` or `\gDTLforeachbibentry`.) This command assigns the value of entry in the column whose label is given by ⟨*field-name*⟩ to the control sequence ⟨*cs*⟩. [§5.2]

`\dtlbreak`

Defined in: datatool package.

When used in the body of commands like `\DTLforeach`, this signifies that the loop should be terminated at the end of the current iteration. [§2.7]

`\dtlcompare{`⟨*register*⟩`}{`⟨*text1*⟩`}{`⟨*text2*⟩`}`

Defined in: datatool package.

A case-sensitive comparison handler for use with `\dtlsort`. [§2.4]

`\DTLcustombibitem{`⟨*marker-code*⟩`}{`⟨*ref-text*⟩`}{`⟨*key*⟩`}`

Defined in: databib package.

This command is analogous to `\bibitem[`⟨*label*⟩`]{`⟨*key*⟩`}` except it

uses ⟨*marker code*⟩ instead of `\item[`⟨*label*⟩`]`. [§5.2]

`\dtldefaultkey`

Defined in: datatool package.

The prefix to use when generating a default column label. [§2.2]

`\dtldisplayafterhead`

Defined in: datatool package.

Hook after the header row in `\DTLdisplaydb` and `\DTLdisplaylongdb`. [§2.6]

`\DTLdisplaydb[`⟨*omit-list*⟩`]{`⟨*db-name*⟩`}`

Defined in: datatool package.

Displays the database identified by ⟨*db-name*⟩ in a tabular environment. [§2.2]

`\dtldisplayendtab`

Defined in: datatool package.

Hook used at the end of `\DTLdisplaydb` and `\DTLdisplaylongdb`. [§2.6]

`\DTLdisplaylongdb[`⟨*options*⟩`]{`⟨*db-name*⟩`}`

Defined in: datatool package.

Displays the database identified by ⟨*db-name*⟩ in a longtable environment. [§2.6]

`\dtldisplaystarttab`

Defined in: datatool package.

Hook used at the start of `\DTLdisplaydb` and `\DTLdisplaylongdb`. [§2.6]

`\DTLdiv{`⟨*cs*⟩`}{`⟨*number1*⟩`}{`⟨*number2*⟩`}`

Defined in: datatool package.

Computes ⟨*number1*⟩ divided by ⟨*number2*⟩ and stores the result in ⟨*cs*⟩, which must be a control sequence. The number should be formatted according to the current locale (set via `\DTLsetnumberchars`) and may optionally be prefixed with a known currency identifier. See the datatool documentation for further details. [§2.1]

`\dtldiv{`⟨*cs*⟩`}{`⟨*number1*⟩`}{`⟨*number2*⟩`}`

Defined in: datatool package.

Computes ⟨*number1*⟩ divided by ⟨*number2*⟩ and stores the result in ⟨*cs*⟩, which must be a control sequence. The number must be a plain decimal number (a full stop as a decimal point, no number grouping and no currency prefix). [§2.1]

`\DTLdobarcolor{`⟨*n*⟩`}`

Defined in: databar package.

Sets the current text colour to the colour of the ⟨*n*⟩th bar. [§12.3]

`\DTLdocurrentpiesegmentcolor`

Defined in: datapie package.

As `\DTLdopiesegmentcolor` but for the current segment. [§12.2]

`\DTLdopiesegmentcolor{`⟨*n*⟩`}`

Defined in: datapie package.

Switches the current text colour to that of the ⟨*n*⟩th segment. [§12.2]

`\DTLendpt`

Defined in: databar package.

For use within the definition of `\DTLeverybarhook`, this can be

used to reference the end of the bar. [§12.3]

`\DTLeverybarhook`

Defined in: databar package.

Hook used by `\DTLbarchart` or `\DTLmultibarchart` at each bar. [§12.3]

`\dtlexpandnewvalue`

Defined in: datatool package.

Switches on the value expansion when using `\DTLnewdbentry` [§9.4]

`\DTLforeach[`⟨*condition*⟩`]{`⟨*db-name*⟩`}{`⟨*assign-list*⟩`}{`⟨*body*⟩`}`

Defined in: datatool package.

Iterates through each row of the database identified by ⟨*db-name*⟩ and does ⟨*body*⟩ if ⟨*condition*⟩ is met for that row. [§2.6]

`\DTLforeach*[`⟨*condition*⟩`]{`⟨*db-name*⟩`}{`⟨*assign-list*⟩`}{`⟨*body*⟩`}`

Defined in: datatool package.

A read-only version of `\DTLforeach`. If no modifications need to be made to the database, this is the better version to use as it's quicker. (How much quicker depends on the size of the database.) [§2.7]

`\DTLforeachbibentry [`⟨*condition*⟩`]{`⟨*db-name*⟩`} {`⟨*body*⟩`}`

Defined in: databib package.

Iterates through the database (loaded by `\DTLloadbbl`) called ⟨*db-name*⟩, and performs ⟨*body*⟩ on each row where ⟨*condition*⟩ is met. The starred version is read-only. [§5.2]

\DTLformatbibentry

Defined in: databib package.

(For use within the final argument of \DTLforeachbibentry or \gDTLforeachbibentry.) This command displays the current row according to the format for its given entry type. [§5.2]

\DTLformatthisbibentry{⟨*db-name*⟩}{⟨*key*⟩}

Defined in: databib package.

Similar to \DTLformatbibentry but can be used outside \DTLforeachbibentry/ \gDTLforeachbibentry to format the referenced identified by ⟨*key*⟩ in the database ⟨*db-name*⟩. [§5.2]

\DTLgabs{⟨*cs*⟩}{⟨*number*⟩}

Defined in: datatool package.

Global version of \DTLabs [§2.1]

\DTLgadd{⟨*cs*⟩}{⟨*number1*⟩} {⟨*number2*⟩}

Defined in: datatool package.

Global version of \DTLadd [§2.1]

\DTLgdiv{⟨*cs*⟩}{⟨*number1*⟩} {⟨*number2*⟩}

Defined in: datatool package.

Global version of \DTLdiv [§2.1]

\DTLgmul{⟨*cs*⟩}{⟨*number1*⟩} {⟨*number2*⟩}

Defined in: datatool package.

Global version of \DTLmul [§2.1]

\DTLgneg{⟨*cs*⟩}{⟨*number*⟩}

Defined in: datatool package.

Global version of \DTLneg [§2.1]

`\DTLground{`⟨*cs*⟩`}{`⟨*number*⟩`}{`⟨*num-digits*⟩`}`

Defined in: datatool package.

Global version of `\DTLround` [§2.1]

`\DTLgsub{`⟨*cs*⟩`}{`⟨*number1*⟩`}{`⟨*number2*⟩`}`

Defined in: datatool package.

Global version of `\DTLsub` [§2.1]

`\dtlicompare{`⟨*register*⟩`}{`⟨*text1*⟩`}{`⟨*text2*⟩`}`

Defined in: datatool package.

A case-insensitive comparison handler for use with `\dtlsort`. [§2.4]

`\DTLifbibfieldexists{`⟨*field-name*⟩`}{`⟨*true-part*⟩`}{`⟨*false-part*⟩`}`

Defined in: databib package.

(For use within the final argument of `\DTLforeachbibentry` or `\gDTLforeachbibentry`.) Does ⟨*true-part*⟩ if the entry in the column whose label is given by ⟨*field-name*⟩ is non-null. Otherwise does ⟨*false-part*⟩. [§5.2]

`\DTLifdbempty{`⟨*db-name*⟩`}{`⟨*true-part*⟩`}{`⟨*false-part*⟩`}`

Defined in: datatool package.

Checks if the datatool internal database called ⟨*db-name*⟩ is empty. [§5.2]

`\DTLiffirstrow{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: datatool package.

Provided for use in the ⟨*body*⟩ part of `\DTLforeach` this does ⟨*true*⟩ if it's on the iteration is on the first row otherwise it does ⟨*false*⟩. [§2.7]

`\DTLifinlist{`⟨*item*⟩`}{`⟨*list*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: datatool package.

Checks if the given item is in the given comma-separated list. A one-level expansion is performed on ⟨*list*⟩ but not on ⟨*item*⟩. [§2.7]

`\DTLiflastrow{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: datatool package.

Provided for use in the ⟨*body*⟩ part of `\DTLforeach` this does ⟨*true*⟩ if it's on the iteration is on the last row otherwise it does ⟨*false*⟩. [§2.7]

`\DTLifnull{`⟨*cs*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: datatool package.

If the control sequence ⟨*cs*⟩ is null (as defined by datatool) this does ⟨*true*⟩ otherwise this does ⟨*false*⟩. [§2.9]

`\DTLifnullorempty{`⟨*cs*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: datatool package.

This is a combination of `\DTLifnull` and `\ifdefempty`. If the control sequence ⟨*cs*⟩ is null (as defined by datatool) or empty this does ⟨*true*⟩ otherwise this does ⟨*false*⟩. [§2.9]

`\dtlifnumeq{`⟨*number1*⟩`}{`⟨*number2*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: datatool-base package.

Checks if ⟨*number1*⟩ equals ⟨*number2*⟩ and does ⟨*true*⟩ if true, otherwise does ⟨*false*⟩. The numbers may be integers or decimals [§9.4]

`\dtllastloadeddb`

Defined in: datatool package.

When you load a `.dbtex` file, this command is set to the label associated with the data in that file. [§2.2]

`\dtlletterindexcompare{`⟨*register*⟩`}{`⟨*text1*⟩`}{`⟨*text2*⟩`}`

Defined in: datatool package.

A letter-ordering comparison handler for use with `\dtlsort`. [§2.4]

`\DTLloadbbl[`⟨*bbl*⟩`]{`⟨*db-name*⟩`}{`⟨*bib-list*⟩`}`

Defined in: databib package.

Loads bibliography data from the `.bbl` file given by ⟨*bbl*⟩ and stores it in the datatool internal database called ⟨*db-name*⟩. This also sets the bibliography style to `databib.bst` and identifies the list of `.bib` files where the bibliography data is defined. [§5.2]

`\DTLloaddb[`⟨*options*⟩`]{`⟨*db-name*⟩`}{`⟨*filename*⟩`}`

Defined in: datatool package.

Loads the data given in the CSV file ⟨*filename*⟩ and stores it in a datatool database called ⟨*db-name*⟩. [§2.2]

`\DTLloaddbtex{`⟨*cs*⟩`}{`⟨*filename*⟩`}`

Defined in: datatool package.

Inputs the and assigns the database name to the command ⟨*cs*⟩ [§2.2]

`\DTLloadrawdb[`⟨*options*⟩`]{`⟨*db-name*⟩`}{`⟨*filename*⟩`}`

Defined in: datatool package.

As \DTLloaddb but maps nine of the ten special characters to LaTeX commands that display the relevent symbol. [§2.2]

\DTLmaketabspace

Defined in: datatool package.

Changes the category code of the tab separator to 10 (space). [§2.2]

\DTLmidpt

Defined in: databar package.

For use within the definition of \DTLeverybarhook, this can be used to reference the mid point of the bar. [§12.3]

\DTLmonthname{⟨*number*⟩}

Defined in: databib package.

Displays the month name for the month given by ⟨*number*⟩. [§5.2]

\DTLmul{⟨*cs*⟩}{⟨*number1*⟩}{⟨*number2*⟩}

Defined in: datatool package.

Computes ⟨*number1*⟩ multiplied by ⟨*number2*⟩ and stores the result in ⟨*cs*⟩, which must be a control sequence. The number should be formatted according to the current locale (set via \DTLsetnumberchars) and may optionally be prefixed with a known currency identifier. See the datatool documentation for further details. [§2.1]

\dtlmul{⟨*cs*⟩}{⟨*number1*⟩}{⟨*number2*⟩}

Defined in: datatool package.

Computes ⟨*number1*⟩ multiplied by ⟨*number2*⟩ and stores the result in ⟨*cs*⟩, which must be a

control sequence. The number must be a plain decimal number (a full stop as a decimal point, no number grouping and no currency prefix). [§2.1]

`\DTLmultibarchart[`⟨*condition*⟩`]{`⟨*settings*⟩`}{`⟨*db-name*⟩`}{`⟨*assign-list*⟩`}`

Defined in: databar package.

Generates a bar chart with grouped data from columns of the datatool database called ⟨*db-name*⟩. [§12.3]

`\DTLneg{`⟨*cs*⟩`}{`⟨*number*⟩`}`

Defined in: datatool package.

Negates ⟨*number*⟩ and stores the result in ⟨*cs*⟩, which must be a control sequence. The number should be formatted according to the current locale (set via `\DTLsetnumberchars`) and may optionally be prefixed with a known currency identifier. See the datatool documentation for further details. [§2.1]

`\dtlneg{`⟨*cs*⟩`}{`⟨*number*⟩`}`

Defined in: datatool package.

Negates ⟨*number*⟩ and stores the result in ⟨*cs*⟩, which must be a control sequence. The number must be a plain decimal number (a full stop as a decimal point, no number grouping and no currency prefix). [§2.1]

`\DTLnewdb{`⟨*db-name*⟩`}`

Defined in: datatool package.

Creates a new database called ⟨*db-name*⟩. [§9.4]

`\DTLnewdbentry{`⟨*db-name*⟩`}{`⟨*col-label*⟩`}{`⟨*value*⟩`}`

Defined in: datatool package.

Adds an entry to the current row of the named database. [§9.4]

`\DTLnewrow{`⟨*db-name*⟩`}`

Defined in: datatool package.

Adds a new row to the database. This new row becomes the current row when adding new entries. [§9.4]

`\DTLnumitemsinlist{`⟨*list*⟩`}{`⟨*cs*⟩`}`

Defined in: datatool package.

Counts the number of non-empty items in the given comma-separated list and stores the result in ⟨*cs*⟩. A one-level expansion is performed on ⟨*list*⟩. [§2.7]

`\DTLpieatbegintikz`

Defined in: datapie package.

Hook performed at the start of `\DTLpiechart`. [§12.2]

`\DTLpieatendtikz`

Defined in: datapie package.

Hook performed at the end of `\DTLpiechart`. [§12.2]

`\DTLpiechart[`⟨*condition*⟩`]{`⟨*settings*⟩`}{`⟨*db-name*⟩`}{`⟨*assign-list*⟩`}`

Defined in: datapie package.

Creates a pie chart from the data given in the database ⟨*db-name*⟩. [§12.2]

`\DTLpiepercent`

Defined in: datapie package.

May be used in the inner or outer label to access the percentage value of the pie chart `variable`. [§12.2]

`\DTLpievariable`

Defined in: datapie package.

May be used in the inner or outer label to access the value of the pie chart `variable`. [§12.2]

`\DTLprotectedsaverawdb{`⟨*db-name*⟩`}{`⟨*filename*⟩`}`

Defined in: datatool package.

Like `\DTLsaverawdb` but works with databases that contain fragile commands. [§9.4]

`\DTLradius`

Defined in: datapie package.

The pie chart radius [§12.2]

`\DTLrawmap{`⟨*string*⟩`}{`⟨*replacement*⟩`}`

Defined in: datatool package.

Defines extra mappings for `\DTLloadrawdb`. [§2.2]

`\DTLround{`⟨*cs*⟩`}{`⟨*number*⟩`}{`⟨*num-digits*⟩`}`

Defined in: datatool package.

Rounds ⟨*number*⟩ to ⟨*num-digits*⟩ decimal places and stores the result in ⟨*cs*⟩, which must be a control sequence. The number should be formatted according to the current locale (set via `\DTLsetnumberchars`) and may optionally be prefixed with a known currency identifier. See the datatool documentation for further details. [§2.1]

`\dtlround{`⟨*cs*⟩`}{`⟨*number*⟩`}{`⟨*num-digits*⟩`}`

Defined in: datatool package.

Rounds ⟨*number*⟩ to ⟨*num-digits*⟩ decimal places and stores the result in ⟨*cs*⟩, which must be a control sequence. The number must be a plain decimal number (a full stop as a decimal point, no number grouping and no currency prefix). [§2.1]

`\DTLsaverawdb{`⟨*db-name*⟩`}{`⟨*filename*⟩`}`

Defined in: datatool package.

Saves the database in the format that can be loaded by `\DTLloaddbtex` and the `datatooltk` application. If the database contains any fragile commands, use `\DTLprotectedsaverawdb` instead. [§9.4]

`\DTLsetbarcolor{`⟨*n*⟩`}{`⟨*colour*⟩`}`

Defined in: databar package.

Sets the ⟨*n*⟩th bar colour to ⟨*colour*⟩. [§12.3]

`\DTLsetdelimiter{`⟨*character*⟩`}`

Defined in: datatool package.

Specifies the delimiter character for CSV files. [§2.2]

`\DTLsetheader{`⟨*db-name*⟩`}{`⟨*col-label*⟩`}{`⟨*header*⟩`}`

Defined in: datatool package.

Assigns a header for the column identified by ⟨*col-label*⟩ in the database labelled ⟨*db-name*⟩. [§2.2]

`\DTLsetnumberchars{`⟨*number group character*⟩`}{`⟨*decimal character*⟩`}`

Defined in: datatool package.

Sets the number group and decimal character for real numbers. See the datatool package for further details. [§2.1]

`\DTLsetpiesegmentcolor{`⟨*n*⟩`}{`⟨*colour*⟩`}`

Defined in: datapie package.

Sets the colour for the ⟨*n*⟩th pie chart segment. [§12.2]

`\DTLsetseparator{`⟨*character*⟩`}`

Defined in: datatool package.

Specifies the separator character for CSV files. [§2.2]

`\DTLsettabseparator`

Defined in: datatool package.

Sets the separator character for CSV files to the tab character. [§2.2]

`\DTLsort[`⟨*replacement*⟩`]{`⟨*criteria*⟩`}{`⟨*db-name*⟩`}`

Defined in: datatool package.

A shortcut for `\dtlsort` where the comparison handler is `\dtlcompare` (case-sensitive). [§2.4]

`\dtlsort[`⟨*replacement*⟩`]{`⟨*criteria*⟩`}{`⟨*db-name*⟩`}{`⟨*handler*⟩`}`

Defined in: datatool package.

Sorts the data identified by ⟨*db-name*⟩ according to the columns listed in the ⟨*criteria*⟩ using the given comparison handler control sequence. [§2.4]

`\DTLsort*[`⟨*replacement*⟩`]{`⟨*criteria*⟩`}{`⟨*db-name*⟩`}`

Defined in: datatool package.

A shortcut for `\dtlsort` where the comparison handler is

\dtlicompare (case-insensitive). [§2.4]

\DTLstartpt

Defined in: databar package.

For use within the definition of \DTLeverybarhook, this can be used to reference the start of the bar. [§12.3]

\DTLsub{⟨*cs*⟩}{⟨*number1*⟩}{⟨*number2*⟩}

Defined in: datatool package.

Computes ⟨*number1*⟩ minus ⟨*number2*⟩ and stores the result in ⟨*cs*⟩, which must be a control sequence. The number should be formatted according to the current locale (set via \DTLsetnumberchars) and may optionally be prefixed with a known currency identifier. See the datatool documentation for further details. [§2.1]

\dtlsub{⟨*cs*⟩}{⟨*number1*⟩}{⟨*number2*⟩}

Defined in: datatool package.

Computes ⟨*number1*⟩ minus ⟨*number2*⟩ and stores the result in ⟨*cs*⟩, which must be a control sequence. The number must be a plain decimal number (a full stop as a decimal point, no number grouping and no currency prefix). [§2.1]

\dtlwordindexcompare{⟨*register*⟩}{⟨*text1*⟩}{⟨*text2*⟩}

Defined in: datatool package.

A word-ordering comparison handler for use with \dtlsort. [§2.4]

`\DTMdate{`⟨*date*⟩`}`

Defined in: datetime2 package.

Displays the given date according to the current date format. [§7.1]

`\DTMnow`

Defined in: datetime2 package.

Inserts into the output file the date and time when the LaTeX application created it from the source code. [§7.1]

`\DTMsavedate{`⟨*name*⟩`}{`⟨*date*⟩`}`

Defined in: datetime2 package.

Stores the given date. [§7.1]

`\DTMsavetimestamp{`⟨*name*⟩`}`
`{`⟨*data*⟩`}`

Defined in: datetime2 package.

Stores the given date and time data. [§7.1]

`\DTMuse{`⟨*name*⟩`}`

Defined in: datetime2 package.

Displays the previously saved date and time stamp. [§7.1]

`\DTMusedate{`⟨*name*⟩`}`

Defined in: datetime2 package.

Displays the previously saved date. [§7.1]

`\DTMusetime{`⟨*name*⟩`}`

Defined in: datetime2 package.

Displays the previously saved time. [§7.1]

`\begin{dynamiccontents}{`⟨*id*⟩`}`

Defined in: flowfram package.

Sets the contents of a dynamic frame. (Verbatim not allowed.) [§10.5]

E

`\eappto`⟨*cs*⟩`{`⟨*code*⟩`}`

Defined in: etoolbox package.

Similar to `\appto` but expands ⟨*code*⟩. Use `\xappto` for a global assignment. [§2.1]

`\EBC{`⟨*description*⟩`}{`⟨*amount*⟩`}`

Defined in: invoice package.

For use within the invoice environment, this command is used to specify a local expense. [§4.2]

`\EBCi{`⟨*description*⟩`}{`⟨*amount*⟩`}`

Defined in: invoice package.

For use within the invoice environment, this command is like `\EBC` but although the amount is added to the total expense it's not itemized. [§4.2]

`\ecvaddress{`⟨*address*⟩`}`

Defined in: europecv class.

Specifies your address. [§5.2]

`\ecvafterpicture{`⟨*text*⟩`}`

Defined in: europecv class.

Specifies text to include after insert your personal image. You can use `\ecvspace` within ⟨*text*⟩. [§5.2]

`\ecvAOne`

Defined in: europecv class.

Shortcut for `\ecvCEF{A1}{basic user}`. [§5.2]

`\ecvATwo`

Defined in: europecv class.

Shortcut for `\ecvCEF{A2}{basic user}`. [§5.2]

\ecvbeforepicture{⟨*text*⟩}

Defined in: europecv class.

Specifies text to include before insert your personal image. You can use \ecvspace within ⟨*text*⟩. [§5.2]

\ecvBOne

Defined in: europecv class.

Shortcut for \ecvCEF{B1}{intermediate user}. [§5.2]

\ecvBTwo

Defined in: europecv class.

Shortcut for \ecvCEF{B2}{intermediate user}. [§5.2]

\ecvCEF{⟨*level*⟩}{⟨*descr*⟩}

Defined in: europecv class.

For use in the ⟨*l1*⟩, ..., ⟨*l5*⟩ arguments of \ecvlanguage or \ecvlastlanguage. [§5.2]

\ecvColSep{⟨*width*⟩}

Defined in: europecv class.

Sets the column separation. [§5.2]

\ecvCOne

Defined in: europecv class.

Shortcut for \ecvCEF{C1}{proficient user}. [§5.2]

\ecvCTwo

Defined in: europecv class.

Shortcut for \ecvCEF{C2}{proficient user}. [§5.2]

\ecvdateofbirth{⟨*date*⟩}

Defined in: europecv class.

Specifies your date of birth. [§5.2]

`\ecvemail{`⟨*address*⟩`}`

Defined in: europecv class.

Specifies your email address. [§5.2]

`\ecvfax{`⟨*fax number*⟩`}`

Defined in: europecv class.

Specifies your fax number. [§5.2]

`\ecvfootername{`⟨*name*⟩`}`

Defined in: europecv class.

Specifies your name as it will appear in the footer. If omitted, the name will be taken from that specified by `\ecvname`. [§5.2]

`\ecvgender{`⟨*gender*⟩`}`

Defined in: europecv class.

Specifies your gender. [§5.2]

`\ecvitem[`⟨*vspace*⟩`]{`⟨*left text*⟩`}{`⟨*right text*⟩`}`

Defined in: europecv class.

Adds ⟨*left text*⟩ to the left of the vertical rule and ⟨*right text*⟩ to the right of the vertical rule. [§5.2]

`\ecvlanguage[`⟨*vspace*⟩`]{`⟨*language*⟩`}{`⟨*l1*⟩`}{`⟨*l2*⟩`}{`⟨*l3*⟩`}{`⟨*l4*⟩`}{`⟨*l5*⟩`}`

Defined in: europecv class.

Typesets a row in the language table. Each of the ⟨*l1*⟩, ..., ⟨*l5*⟩ arguments should be in the form `\ecvCEF{`⟨*level*⟩`}{`⟨*desc*⟩`}`. Use `\ecvlastlanguage` for the last row. [§5.2]

`\ecvlanguagefooter[`⟨*vspace*⟩`]{`⟨*symbol*⟩`}`

Defined in: europecv class.

Typesets the footer of the language table and identifies the symbol to use as a footnote symbol, which should be the same as that used in `\ecvlanguageheader`. [§5.2]

`\ecvlanguageheader{`⟨*symbol*⟩`}`

Defined in: europecv class.

Typesets the header of the language table and identifies the symbol to use as a footnote symbol. [§5.2]

`\ecvlastlanguage[`⟨*vspace*⟩`]{`⟨*language*⟩`}{`⟨*l1*⟩`}{`⟨*l2*⟩`}{`⟨*l3*⟩`}{`⟨*l4*⟩`}{`⟨*l5*⟩`}`

Defined in: europecv class.

Typesets the last row in the language table. Each of the ⟨*l1*⟩, ..., ⟨*l5*⟩ arguments should be in the form `\ecvCEF{`⟨*level*⟩`}{`⟨*desc*⟩`}`. [§5.2]

`\ecvLeftColumnWidth{`⟨*width*⟩`}`

Defined in: europecv class.

Sets the width of the left column. [§5.2]

`\ecvmothertongue[`⟨*vspace*⟩`]{`⟨*language*⟩`}`

Defined in: europecv class.

Starts the spoken language section and identifies your mother tongue. [§5.2]

`\ecvname{`⟨*name*⟩`}`

Defined in: europecv class.

Specifies your name. [§5.2]

`\ecvnationality{`⟨*nationality*⟩`}`

Defined in: europecv class.

Specifies your nationality. [§5.2]

`\ecvpersonalinfo[`⟨*vspace*⟩`]`

Defined in: europecv class.

Typesets your personal details. [§5.2]

`\ecvpicture[`⟨*options*⟩`]{`⟨*image filename*⟩`}`

Defined in: europecv class.

Specifies the name of the file showing an image of yourself. [§5.2]

`\ecvsection[`⟨*vspace*⟩`]{`⟨*title*⟩`}`

Defined in: europecv class.

Creates a section heading. [§5.2]

`\ecvspace{`⟨*height*⟩`}`

Defined in: europecv class.

May only be used within the argument of `\ecvbeforepicture` or `\ecvafterpicture` to insert some vertical space. [§5.2]

`\ecvtelephone[`⟨*mobile*⟩`]{`⟨*telephone*⟩`}`

Defined in: europecv class.

Specifies your telephone number and optionally your mobile phone number. [§5.2]

`\edef`⟨*cs*⟩⟨*arg-syntax*⟩`{`⟨*definition*⟩`}`

Defined in: TEX primitive.

This locally defines the command ⟨*cs*⟩ to the full expansion of ⟨*definition*⟩. Use `\xdef` for global definitions. [§2.1]

`\EFC{`⟨*description*⟩`}{`⟨*foreign currency*⟩`}{`⟨*amount*⟩`}`

{⟨*conversion rate*⟩}{⟨*base currency result*⟩}

Defined in: invoice package.

For use within the invoice environment, this command is used to specify a foreign expense. [§4.2]

\EFCi{⟨*description*⟩}{⟨*foreign currency*⟩}{⟨*amount*⟩} {⟨*conversion rate*⟩}{⟨*base currency result*⟩}

Defined in: invoice package.

For use within the invoice environment, this command is like \EFC but although the amount is added to the total expense it's not itemized. [§4.2]

\emailfrom{⟨*text*⟩}

Defined in: newlfm class.

Specifies the sender's email address. [§3.3]

\emph{⟨*text*⟩}

Defined in: LaTeX Kernel.

Toggles the upright and italic/slanted rendering of ⟨*text*⟩. (See Volume 1 [92, §4.5.1].) [§2.1]

\empty

Defined in: LaTeX Kernel.

Does nothing. [§12.5]

\encl{⟨*enclosures info*⟩}

Defined in: Classes that define the letter environment.

Used to indicate any enclosures accompanying the letter. [§3.1]

\encllist{⟨*text*⟩}

Defined in: newlfm class.

Specifies the list of enclosures. [§3.3]

`\end{`*⟨env-name⟩*`}`

Defined in: LaTeX Kernel.

Ends an environment. (Must have a matching `\begin`. See Volume 1 [92, §2.15].) [§1.2]

`\endfirsthead`

Defined in: longtable package.

Marks the end of the header code for the first page of the longtable environment. [§4.3]

`\endfoot`

Defined in: longtable package.

Marks the end of the footer code for the longtable environment. [§4.3]

`\endhead`

Defined in: longtable package.

Marks the end of the header code for the longtable environment. [§4.3]

`\endinput`

Defined in: TeX primitive.

Stops reading the current file. Anything in the current file occurring after this command is skipped. [§7.3]

`\endlastfoot`

Defined in: longtable package.

Marks the end of the footer code for the last page of the longtable environment. [§4.3]

`\endtime{`*⟨time⟩*`}`

Defined in: minutes package.

Specifies the end time of the meeting. [§6.3]

`\enspace`

Defined in: LaTeX Kernel.

Horizontal spacing command (half as wide as `\quad`). [§9.1]

`\begin{enumerate}`

Defined in: LaTeX Kernel.

Ordered list. Some packages, such as paralist, modify this environment to provide an optional argument that allows you to adjust the counter format. [§6.5]

`\epreto`⟨*cs*⟩`{`⟨*code*⟩`}`

Defined in: etoolbox package.

Similar to `\preto` but expands ⟨*code*⟩. Use `\xpreto` for a global assignment. [§2.1]

`\equal{`⟨*text1*⟩`}{`⟨*text2*⟩`}`

Defined in: ifthen package.

A test that can be used within the condition of `\ifthenelse` to determine if ⟨*text1*⟩ is the same as ⟨*text2*⟩. (Both arguments are expanded.) [§2.7]

`\begin{europecv}`

Defined in: europecv class.

The body of the CV. [§5.2]

`\evensidemargin`

Defined in: LaTeX Kernel.

A length containing the horizontal offset for even pages. See `\hoffset`. [§11.0]

`\everypar{`⟨*code*⟩`}`

Defined in: TeX primitive.

Indicates code to be performed at the start of every paragraph. [§6.5]

`exists`(⟨*ref*⟩)

Defined in: `arara` directive.

Evaluates to true if the given file doesn't exist. The argument ⟨*ref*⟩ may either be a string "⟨*extension*⟩" which indicates the file extension or a file reference `toFile`("⟨*filename*⟩"). [§1.2]

`\expandafter`⟨*token 1*⟩⟨*token 2*⟩

Defined in: TEX primitive.

Equivalent to ⟨*token 1*⟩ expansion of ⟨*token 2*⟩. [§2.7]

`\expandonce`⟨*cs*⟩

Defined in: etoolbox package.

Only permits one level of expansion of the command ⟨*cs*⟩. [§2.1]

F

`\fbox{`⟨*text*⟩`}`

Defined in: LATEX Kernel.

Puts a frame around its contents, prohibiting a line break in the contents. [§7.5]

`\Fee{`⟨*description*⟩`}{`⟨*rate/unit*⟩`}{`⟨*count*⟩`}`

Defined in: invoice package.

For use within the invoice environment, this command is used to specify a fee. [§4.2]

`\begin{figure}[`⟨*placement*⟩`]`

Defined in: Most classes except for those designed for correspondence or similarly restrictive documents.

Floats the contents to the nearest location according to the

preferred placement options, if possible. Within the environment, \caption may be used one or more times, as required. (See Volume 1 [92, §7.1].) [§2.1]

\FilledSmallCircle

Defined in: ifsym package with geometry option.

Produces a small filled circle. [§11.1]

\FirstLabel{⟨*row*⟩}{⟨*column*⟩}

Defined in: envlab package.

Sets the starting label on a partially used sheet. [§3.6]

\begin{flushright}

Defined in: LaTeX Kernel.

Aligns its contents flush-right and places a small vertical gap above and below the environment. [§10.3]

\fontfamily{⟨*name*⟩}

Defined in: LaTeX Kernel.

Sets the name of the current font family. (The change won't take effect until the next \selectfont.) For example, in text mode, \rmfamily is equivalent to \fontfamily{cmr}\selectfont (unless it's been modified by a font package.) [§6.4]

\fontseries{⟨*weight*⟩}

Defined in: LaTeX Kernel.

Sets the name of the current font series. (The change won't take effect until the next \selectfont.) For example, in text mode, \bfseries is equivalent to \fontseries{b}\selectfont (unless it's been modified by a font package.) [§6.4]

`\footnote[`⟨*number*⟩`]{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Inserts a footnote. [§2.1]

`\forcsvlist{`⟨*handler-cs*⟩`}`
`{`⟨*item1,item2,...*⟩`}`

Defined in: etoolbox package.

This is like `\docsvlist` except that instead of using `\do` it uses ⟨*handler-cs*⟩. [§2.7]

`\foreach`⟨*variables*⟩`[`⟨*options*⟩`]` in `{`⟨*list*⟩`}{`⟨*body*⟩`}`

Defined in: pgffor package.

Iterates through the ⟨*list*⟩ and assigns ⟨*variables*⟩ which can be used in ⟨*body*⟩. The full syntax is quite complicated, so read the pgf manual for further details. The `\breakforeach` command can be used to prematurely terminate the loop after the current iteration. [§2.7]

`\foreachdataset{`⟨*cs*⟩`}{`⟨*body*⟩`}`

Defined in: probsoln package.

Iterates over all defined datasets and performs ⟨*body*⟩ at each iteration. [§9.3]

`\foreachproblem[`⟨*dataset*⟩`]`
`{`⟨*body*⟩`}`

Defined in: probsoln package.

Iterates through the given dataset and does ⟨*body*⟩ at each iteration. Within ⟨*body*⟩, `\thisproblem` may be used to display the current problem and `\thisproblemlabel` may be used to access the current problem label. [§9.3]

`\foreachsolution[`⟨*dataset*⟩`]`

`{`⟨*body*⟩`}`

Defined in: probsoln package.

Similar to `\foreachproblem` but only iterates through problems that contain the onlysolution environment. [§9.3]

`\forlistcsloop{`⟨*handler-cs*⟩`}{`⟨*list-csname*⟩`}`

Defined in: etoolbox package.

Similar to `\forlistloop` except the list control sequence name (without the leading backslash) is used. [§2.7]

`\forlistloop{`⟨*handler-cs*⟩`}{`⟨*list-cs*⟩`}`

Defined in: etoolbox package.

Similar to `\dolistloop` except it uses ⟨*handler-cs*⟩ instead of `\do` at each iteration. [§2.7]

`\begin{Form}[`⟨*parameters*⟩`]`

Defined in: hyperref package.

Environment containing interactive form elements. [§11.2]

`found(`⟨*ref*⟩`,` ⟨*expression*⟩`)`

Defined in: `arara` directive.

Evaluates to true if the given regular expression is found in the given file. The argument ⟨*ref*⟩ may either be a string "⟨*extension*⟩" which indicates the file extension or a file reference `toFile("`⟨*filename*⟩`")`. [§1.2]

`\FPrandom{`⟨*cs*⟩`}`

Defined in: fp package.

Generates a random number between 0 and 1 and assigns the result to the given control sequence ⟨*cs*⟩. [§9.5]

`\FPseed`

Defined in: fp package.

A count register that stores the random generator seed. [§9.5]

`\frac{`⟨*numerator*⟩`}{`⟨*denominator*⟩`}`

Defined in: LaTeX Kernel (Math Mode).

Displays a fraction. [§9.3]

`\begin{frame}`

Defined in: beamer class.

Creates a slide (or possibly multiple slides if the frame contains overlays) [§8.0]

`\frame{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Puts a rectangular frame around ⟨*text*⟩. Similar to `\fbox` but doesn't insert a gap between ⟨*text*⟩ and the frame. The beamer class redefines this command for its frame environment. [§10.4]

`\framebox[`⟨*width*⟩`][`⟨*align*⟩`]{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Puts a frame around its contents, prohibiting a line break in the contents. [§10.1]

`\framebox(`⟨*w*⟩`,`⟨*h*⟩`)[`⟨*align*⟩`]{`⟨*text*⟩`}`

Defined in: picture environment.

Unlike the ordinary `\framebox` command, this version doesn't add any space between the frame and the text. [§10.1]

`\framebreak`

Defined in: flowfram package.

Inserted at the point of the frame break when a paragraph spans two flow frames of unequal widths. [§10.5]

`\framesubtitle{`⟨*subtitle*⟩`}`

Defined in: beamer class.

Subtitle for a frame. (For use within the frame environment.) [§8.0]

`\frametitle{`⟨*title*⟩`}`

Defined in: beamer class.

Title for a frame. (For use within the frame environment. [§8.0]

G

`\ganttbar[`⟨*options*⟩`]{`⟨*text*⟩`}{`⟨*start-tss*⟩`}{`⟨*end-tss*⟩`}`

Defined in: pgfgantt package.

For use within the ganttchart environment, this draws a bar representing a task or subtask. [§12.4]

`\begin{ganttchart}[`⟨*options*⟩`]{`⟨*start*⟩`}{`⟨*end*⟩`}`

Defined in: pgfgantt.

Creates a Gantt chart. [§12.4]

`\ganttgroup[`⟨*options*⟩`]{`⟨*text*⟩`}{`⟨*start-tss*⟩`}{`⟨*end-tss*⟩`}`

Defined in: pgfgantt package.

For use within the ganttchart environment, this draws a group bar. [§12.4]

`\ganttlinkedbar[`⟨*options*⟩`]{`⟨*text*⟩`}{`⟨*start-tss*⟩`}{`⟨*end-tss*⟩`}`

Defined in: pgfgantt package.

For use within the ganttchart environment, this draws a linked bar representing a task or subtask. [§12.4]

`\ganttlinkedgroup[`⟨*options*⟩`]{`⟨*text*⟩`}{`⟨*start-tss*⟩`}{`⟨*end-tss*⟩`}`

Defined in: pgfgantt package.

For use within the ganttchart environment, this draws a linked group bar. [§12.4]

`\ganttlinkedmilestone[`⟨*options*⟩`]{`⟨*text*⟩`}{`⟨*tss*⟩`}`

Defined in: pgfgantt package.

For use within the ganttchart environment, this draws a linked milestone marker. [§12.4]

`\ganttmilestone[`⟨*options*⟩`]{`⟨*text*⟩`}{`⟨*tss*⟩`}`

Defined in: pgfgantt package.

For use within the ganttchart environment, this draws a milestone marker. [§12.4]

`\ganttnewline[`⟨*options*⟩`]`

Defined in: pgfgantt package.

For use within the ganttchart environment, this starts a new row. [§12.4]

`\ganttset{`⟨*options*⟩`}`

Defined in: pgfgantt.

Sets the options governing the Gantt chart style. [§12.4]

`\ganttitle[`⟨*options*⟩`]{`⟨*text*⟩`}{`⟨*n*⟩`}`

Defined in: pgfgantt package.

For use within the ganttchart environment, this draws a single title element. [§12.4]

`\ganttitlecalendar[`⟨*options*⟩`]{`⟨*calendar-lines*⟩`}`

Defined in: pgfgantt package.

For use within the ganttchart environment, this draws a title calendar that spans the whole chart. [§12.4]

`\ganttitlecalendar*[`⟨*options*⟩`]{`⟨*start-tss*⟩`}{`⟨*end-tss*⟩`}{`⟨*calendar lines*⟩`}`

Defined in: pgfgantt package.

For use within the ganttchart environment, this draws a title calendar that spans the chart from ⟨*start-tss*⟩ to ⟨*end-tss*⟩. [§12.4]

`\ganttitlelist[`⟨*options*⟩`]{`⟨*list*⟩`}{`⟨*n*⟩`}`

Defined in: pgfgantt package.

For use within the ganttchart environment, this iterates over ⟨*list*⟩ and draws a title element spanning ⟨*n*⟩ time slots. [§12.4]

`\gappto`⟨*cs*⟩`{`⟨*code*⟩`}`

Defined in: etoolbox package.

Global version of `\appto`. [§2.1]

`\gdef`⟨*cs*⟩⟨*arg-syntax*⟩`{`⟨*definition*⟩`}`

Defined in: TEX primitive.

As `\def` but the definition is global. [§2.1]

`\gDTLforeachbibentry[`⟨*condition*⟩`]{`⟨*db-name*⟩`}{`⟨*body*⟩`}`

Defined in: databib package.

Global version of `\DTLforeachbibentry`. The starred version is read-only. [§5.2]

`\getflowbounds{`⟨*id*⟩`}`

Defined in: flowfram package.

Gets the location and size of the given flow frame. The results are stored in the lengths `\ffareax`, `\ffareay`, `\ffareawidth` and `\ffareaheight`. [§11.0]

`\getflowevenbounds{`⟨*id*⟩`}`

Defined in: flowfram package.

Gets the location and size of the given flow frame for even pages. The results are stored in the lengths `\ffareax`, `\ffareay`, `\ffareawidth` and `\ffareaheight`. [§11.0]

`\global`⟨*assignment*⟩

Defined in: TEX primitive.

An assignment prefix that indicates the following assignment shouldn't be confined to the current scope. [§2.1]

`\gpreto`⟨*cs*⟩`{`⟨*code*⟩`}`

Defined in: etoolbox package.

Global version of `\preto`. [§2.1]

`\gradetable[`⟨*orientation*⟩`]`
`[`⟨*index-type*⟩`]`

Defined in: exam class.

Displays the grading table. [§9.1]

`\greetto{`⟨*text*⟩`}`

Defined in: newlfm class.

The salutation text. [§3.3]

`\guest{`⟨*names*⟩`}`

Defined in: minutes package.

Specifies the names of any guests present. [§6.3]

H

`\half`

Defined in: exam class.

Indicates a half point [§9.1]

`\headline{`⟨*text*⟩`}`

Defined in: newlfm class.

Specifies the subject of a press release. [§6.2]

`\hfill`

Defined in: LaTeX Kernel.

Inserts a horizontal space that will expand to fit the available width. [§3.6]

`\begin{hiddenitems}`

Defined in: meetingmins class.

Similar to the items environment but is only displayed if the agenda class option is used. [§6.3]

`\begin{hiddensubitems}`

Defined in: meetingmins class.

Similar to the subitems environment but is only displayed if the agenda class option is used. [§6.3]

`\begin{hiddentext}`

Defined in: meetingmins class.

This environment only displays its contents if the agenda class option is used. [§6.3]

`\hideanswers`

Defined in: probsoln package.

Hides the solutions (from that point onwards). [§9.3]

`\hoffset`

Defined in: LaTeX Kernel.

A length containing the horizontal offset. The left margin of a page is computed from `\hoffset` plus 1 in plus

`\oddsidemargin`/`\evensidemargin`. [§11.0]

`\hrulefill`

Defined in: LaTeX Kernel.

Fills the remaining space with a line. [§9.1]

`\hspace{`⟨*length*⟩`}`

Defined in: LaTeX Kernel.

Inserts a horizontal gap of the given width. The unstarred version doesn't create a space if it occurs at the beginning or end of a paragraph. The starred version always creates a space. [§3.6]

`\Huge`

Defined in: Most document classes.

Switches to extra-huge sized text. [§10.3]

`\hypersetup{`⟨*key-val list*⟩`}`

Defined in: hyperref package.

Set up options. [§2.3]

I

`if` ⟨*condition*⟩

Defined in: `arara` directive.

Only run the application if ⟨*condition*⟩ is true. [§1.2]

`\ifbool{`⟨*name*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: etoolbox package.

Expands to ⟨*true*⟩ if the boolean flag ⟨*name*⟩ is true, and to ⟨*false*⟩ otherwise. [§9.3]

`\ifboolexpr{`⟨*expression*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: etoolbox package.

Evaluates the ⟨*expression*⟩ and executes ⟨*true*⟩ if true, and ⟨*false*⟩ otherwise. The syntax for the expression is described in the etoolbox manual. [§2.9]

`\ifcase`⟨*number*⟩ ⟨*case0 code*⟩`\or` ⟨*case1 code*⟩`\or` ⟨*case2 code*⟩`\or` ...`\else` ⟨*default code*⟩`\fi`

Defined in: TEX primitive.

If ⟨*number*⟩ equals 0, performs ⟨*case0 code*⟩, if ⟨*number*⟩ equals 1, performs ⟨*case1 code*⟩, if ⟨*number*⟩ equals 2, performs ⟨*case2 code*⟩, etc. If none of the cases match, ⟨*default code*⟩ is performed. The `\else` ⟨*default code*⟩ part may be omitted. If ⟨*case 0 code*⟩ starts with a number, insert `\relax` before it to prevent TEX from scanning it as part of ⟨*number*⟩. [§7.3]

`\ifcsdef{`⟨*cs-name*⟩`}{`⟨*true-part*⟩`}{`⟨*false-part*⟩`}`

Defined in: etoolbox package.

Checks if the control sequence with the name ⟨*cs-name*⟩ exists. [§2.1]

`\ifcsundef{`⟨*cs-name*⟩`}{`⟨*true-part*⟩`}{`⟨*false-part*⟩`}`

Defined in: etoolbox package.

Checks if the control sequence whose name is given by ⟨*cs name*⟩ doesn't exist or is defined as `\relax`. [§2.1]

`\ifdate{`⟨*tests*⟩`}{`⟨*true-part*⟩`}{`⟨*false-part*⟩`}`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. The same as

`\pgfcalendarifdate` for the current date. [§7.5]

`\ifdef{`⟨*cs*⟩`}{`⟨*true-part*⟩`}{`⟨*false-part*⟩`}`

Defined in: etoolbox package.

Checks if the control sequence ⟨*cs*⟩ exists. [§2.1]

`\ifdefempty{`⟨*cs*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: etoolbox package.

If the control sequence ⟨*cs*⟩ is empty this does ⟨*true*⟩ otherwise it does ⟨*false*⟩. [§2.9]

`\ifdefstring{`⟨*cs*⟩`}{`⟨*string*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: etoolbox package.

If the control sequence ⟨*cs*⟩ was defined to be ⟨*string*⟩ this does ⟨*true*⟩ otherwise it does ⟨*false*⟩. (No expansion is performed.) [§2.9]

`\ifinlist{`⟨*item*⟩`}{`⟨*list-cs*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: etoolbox package.

Checks if ⟨*item*⟩ is included in one of etoolbox's internal list macros ⟨*list-cs*⟩ and does ⟨*true*⟩ if true, otherwise it does ⟨*false*⟩. No expansion is performed on ⟨*item*⟩. See also `\xifinlist` [§2.7]

`\ifinlistcs{`⟨*item*⟩`}{`⟨*list-csname*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Defined in: etoolbox package.

As `\ifinlist` but the control sequence name is supplied (without the backslash). [§2.7]

`\ifnum`⟨*num1*⟩ ⟨*comp*⟩ ⟨*num2*⟩

⟨*true-part*⟩`\else` ⟨*false-part*⟩`\fi`

Defined in: TEX primitive.

Compares ⟨*num1*⟩ to ⟨*num2*⟩. The comparison ⟨*comp*⟩ may be one of: = (equality), < (less than) or > (greater than). The `\else` ⟨*false-part*⟩ may be omitted. If ⟨*true-part*⟩ starts with a number, insert `\relax` before it to prevent TEX from scanning it as part of ⟨*num2*⟩. [§2.7]

`\ifnumless{`⟨*number1*⟩`}{`⟨*number2*⟩`}{`⟨*true-part*⟩`}{`⟨*false-part*⟩`}`

Defined in: etoolbox package.

Checks if ⟨*number1*⟩ is less than ⟨*number2*⟩. Both values should be integers. [§5.2]

`\ifodd`⟨*number*⟩ ⟨*odd code*⟩`\else` ⟨*even code*⟩`\fi`

Defined in: TEX primitive.

Tests if ⟨*number*⟩ is odd. If true this does ⟨*odd code*⟩ otherwise it does ⟨*even code*⟩. The `\else` ⟨*even code*⟩ may be omitted. [§11.0]

`\ifshowanswers` ⟨*true-part*⟩`\else` ⟨*false-part*⟩`\fi`

Defined in: probsoln package.

Tests if the solutions are displayed. [§9.3]

`\ifstrempty{`⟨*string*⟩`}{`⟨*true-part*⟩`}{`⟨*false-part*⟩`}`

Defined in: etoolbox package.

Tests if ⟨*string*⟩ is empty. (No expansion is performed on ⟨*string*⟩.) [§7.3]

\ifstrequal{⟨*string1*⟩}{⟨*string2*⟩}{⟨*true-part*⟩}{⟨*false-part*⟩}

Defined in: etoolbox package.

Tests if ⟨*string1*⟩ is the same as ⟨*string2*⟩. (No expansion is performed on ⟨*string1*⟩ or ⟨*string2*⟩.) [§7.4]

\ifthenelse{⟨*condition*⟩}{⟨*true*⟩}{⟨*false*⟩}

Defined in: ifthen package.

If the condition is met, does ⟨*true*⟩ otherwise does ⟨*false*⟩. The ⟨*condition*⟩ must follow the syntax defined by the ifthen package. The datatool package provides additional commands that may be used in ⟨*condition*⟩. Note that in general, it's better to use the conditionals provided by the etoolbox package, but the optional argument of \DTLforeach (and the starred version) requires the same format as the first argument of \ifthenelse. [§2.7]

\ifthispageodd{⟨*odd code*⟩}{⟨*even code*⟩}

Defined in: KOMA-Script classes.

Determines if the current page is odd or even. [§11.0]

\ifundef{⟨*cs*⟩}{⟨*true-part*⟩}{⟨*false-part*⟩}

Defined in: etoolbox package.

Checks if the control sequence ⟨*cs*⟩ doesn't exist or is defined as \relax. [§2.1]

\ignorespaces

Defined in: TEX primitive.

Used in begin environment code to suppress any spaces occurring at

the start of the environment (see also \ignorespacesafterend). [§9.3]

\ignorespacesafterend

Defined in: LaTeX Kernel.

Used in end environment code to suppress any spaces following the end of the environment. [§9.3]

\iitem{⟨*description*⟩}{⟨*amount*⟩}

Defined in: isodoc class.

Generates a row of data for use within \itable. [§4.1]

\include{⟨*filename*⟩}

Defined in: LaTeX Kernel.

Issues a \clearpage, creates an associated auxiliary file, inputs ⟨*filename*⟩ and issues another \clearpage. (See also \input.) [§13.0]

\includegraphics[⟨*option-list*⟩] {⟨*filename*⟩}

Defined in: graphicx package.

Inserts a graphics file into the document. Permitted file types depend on the output format. (PostScript (PS) and Encapsulated PostScript (EPS) for the DVI format. PDF, JPG and PNG for the PDF format (also EPS if the TeX distribution permits on-the-fly epstopdf conversion). [§3.2]

\includequestions[⟨*options*⟩] {⟨*filenames*⟩}

Defined in: exsheets package.

Includes questions defined in the named files. [§9.2]

\incorrectitem

Defined in: probsoln package.

For use within the textenum, this command may be used in place of \item to indicate an incorrect choice. If the solutions aren't displayed this command behaves the same as \item. [§9.3]

\incorrectitemformat{⟨*marker*⟩}

Defined in: probsoln package.

The format used by \incorrectitem. [§9.3]

\begin{inparaenum}[⟨*format*⟩]

Defined in: paralist package.

An inline numbered list. (Similar to the enumerate environment, but the items don't start a new paragraph, unless you explicitly insert a paragraph break.) [§9.2]

\input{⟨*filename*⟩}

Defined in: LaTeX Kernel.

Reads in the contents of ⟨*filename*⟩. [§2.2]

\InputIfFileExists{⟨*file*⟩}{⟨*true-part*⟩}{⟨*false-part*⟩}

Defined in: LaTeX Kernel.

If the given file exists, this does ⟨*true-part*⟩ and then loads the file. Otherwise it does ⟨*false-part*⟩. [§6.4]

\inst{⟨*text*⟩}

Defined in: beamer class.

Used to prefix each institute listed in \institute when there are multiple institutes.
A corresponding \inst{⟨*number*⟩} should be placed after the relevant author name within \author. [§8.0]

`\institute{`⟨*text*⟩`}`

Defined in: Various classes or packages that have the concept of an institute in the title page.

Specifies the author's institution for use with `\maketitle` in a similar manner to `\title`. Some classes, such as beamer, also provide an optional argument for this command. [§8.0]

`\begin{invoice}{`⟨*base-currency*⟩`}{`⟨*VAT*⟩`}`

Defined in: invoice package.

The body of the invoice. [§4.2]

`\invoice[`⟨*options*⟩`]{`⟨*contents*⟩`}`

Defined in: isodoc class.

Creates an invoice. [§4.1]

`\itable{`⟨*contents*⟩`}`

Defined in: isodoc class.

Generates the table used within `\invoice`. [§4.1]

`\item[`⟨*marker*⟩`]`

Defined in: LaTeX Kernel.

Specifies the start of an item in a list. (Only allowed inside one of the list making environments, such as enumerate.) [§5.1]

`\begin{itemize}`

Defined in: LaTeX Kernel.

Unordered list. [§8.1]

`\begin{items}`

Defined in: meetingmins class.

A numbered list for use within a section. Use the subitems environment for lists within subsections and sub-subsections. [§6.3]

`\itotal[`⟨*tag*⟩`]{`⟨*amount*⟩`}`

Defined in: isodoc class.

Generates a row of for the total amount for use within `\itable`. [§4.1]

`\itshape`

Defined in: LaTeX Kernel.

Switches to the italic form of the current font family, if it exists. See Volume 1 [92, §4.5.1]. [§10.2]

J

`\jobname`

Defined in: TeX primitive.

The current job name. This is usually the base name (without the `.tex` extension) of the main `.tex` file, but can be changed using TeX's `-jobname` switch. [§5.2]

K

`\KOMAoption{`⟨*option*⟩`}{`⟨*value list*⟩`}`

Defined in: KOMA-Script classes.

Allows you to specify a list of values to the given multi-valued KOMA-Script option. [§3.2]

`\KOMAoptions{`⟨*option list*⟩`}`

Defined in: KOMA-Script classes.

Allows you to set one or more of the KOMA-Script options. [§3.2]

L

`\label{`⟨*string*⟩`}`

Defined in: LaTeX Kernel.

Assigns a unique textual label linked to the most recently incremented cross-referencing counter in the current scope. (See Volume 1 [92, §5.5].) [§1.2]

`\labelenumi`

Defined in: LaTeX Kernel.

Used by `\item` in the first level enumerate environment to display the label. [§6.5]

`\labelenumii`

Defined in: LaTeX Kernel.

Used by `\item` in the second level enumerate environment to display the label. [§6.5]

`\labelenumiii`

Defined in: LaTeX Kernel.

Used by `\item` in the third level enumerate environment to display the label. [§6.5]

`\labelenumiv`

Defined in: LaTeX Kernel.

Used by `\item` in the fourth level enumerate environment to display the label. [§6.5]

`\LabelHeight`

Defined in: envlab package.

A length register used to store the label height. [§3.6]

`\labelsep`

Defined in: LaTeX Kernel.

Length register used by list environments to store the size of the horizontal gap between the item marker and the following text. [§6.5]

`\LabelWidth`

Defined in: envlab package.

A length register used to store the label width. [§3.6]

`\labelwidth`

Defined in: LaTeX Kernel.

Length register used by list environments to store the label width. The value is typically set in the ⟨*list declarations*⟩ argument of the list environment. [§6.5]

`\Large`

Defined in: Most document classes.

Switches to extra-large sized text. [§2.1]

`\begin{Large}`

Defined in: Most document classes.

Sets its body in an extra-large sized font. [§2.1]

`\large`

Defined in: Most document classes.

Switches to large sized text. [§10.1]

`\LayoutCheckField{`⟨*label*⟩`}{`⟨*field*⟩`}`

Defined in: hyperref package.

Lays out the label and associated check box. This just defaults to ⟨*label*⟩ ⟨*field*⟩. [§11.2]

`\LayoutChoiceField{`⟨*label*⟩`}{`⟨*field*⟩`}`

Defined in: hyperref package.

Lays out the label and associated choice field. This just defaults to ⟨*label*⟩ ⟨*field*⟩. [§11.2]

`\LayoutTextField{`⟨*label*⟩`}{`⟨*field*⟩`}`

Defined in: hyperref package.

Lays out the label and associated text field. This just defaults to ⟨*label*⟩ ⟨*field*⟩. [§11.2]

`\leftmargin`

Defined in: LaTeX Kernel.

Length register used by list environments to store the left margin width. The value is typically set in the ⟨*list declarations*⟩ argument of the list environment. [§6.5]

`\LenToUnit{`⟨*length*⟩`}`

Defined in: leaflet class.

May be used to specify a length instead of a value that's in terms of `\unitlength` for use within the picture environment. [§10.3]

`\let`⟨*new cs*⟩⟨*org cs*⟩

Defined in: TeX primitive.

Gives ⟨*new cs*⟩ the same meaning as ⟨*org cs*⟩. For example if I define: `\newcommand{\mycmd}{stuff}` and then `\let\mynewcmd\mycmd` is like doing `\newcommand{\mynewcmd}{stuff}`, but no existence check is performed. If I then redefine `\mycmd`, `\mynewcmd` still retains the old definition of `\mycmd`. Since there's no check to see if the new command is already defined, care is required to prevent accidentally overwriting a preexisting command. (The arguments ⟨*new cs*⟩ and ⟨*org cs*⟩ don't actually have to be control sequences. They can be any tokens, but that's beyond the scope of this book.) [§2.1]

`\letcs{`⟨*cs*⟩`}{`⟨*cs-name*⟩`}`

Defined in: etoolbox package.

Analogous to `\let` except that the name of the control sequence (without the initial backslash) is supplied for the second argument. [§2.1]

`\begin{letter}{`⟨*addressee*⟩`}`

Defined in: Some classes used for writing letters, such as scrlttr2.

Typesets its contents as correspondence. The KOMA-Script scrlttr2 class also allows an optional argument before the mandatory argument. [§3.1]

`\letter[`⟨*recipient-options*⟩`]`
`{`⟨*contents*⟩`}`

Defined in: isodoc class.

Creates a letter. [§3.4]

`\lim`

Defined in: LaTeX Kernel (Math Mode).

Typesets lim function name (may have limits via `_` or `^`). [§9.3]

`\line(`⟨*h*⟩`,`⟨*v*⟩`){`⟨*length*⟩`}`

Defined in: LaTeX Kernel.

For use within the argument of `\put`, this draws a straight line of the given length whose horizontal and vertical extent (gradient vector) is given by (⟨*h*⟩,⟨*v*⟩). [§10.1]

`\lineskip`

Defined in: LaTeX Kernel.

A TeX primitive that stores the interline glue. [§3.6]

`\linewidth`

Defined in: LaTeX Kernel.

A length containing the desired current line width. This is usually the width of the typeblock, but inside a minipage or `\parbox` it will be the width the box. Note that the actual contents of the line may fall short of the line width (underfull hbox) or extend beyond it (overfull hbox). [§4.3]

`\lipsum[`⟨*selection*⟩`]`

Defined in: lipsum package.

Produces dummy text for testing purposes. The optional argument specifies which paragraphs to display. (There are 250 predefined paragraphs). By default this command displays the first seven paragraphs, but this may be changed via the optional argument, which may be either a single number or a range. The starred version suppresses the paragraph breaks. [§6.4]

`\begin{list}{`⟨*label*⟩`}{`⟨*list declarations*⟩`}`

Defined in: LaTeX Kernel.

A generic list environment [§6.5]

`\listadd{`⟨*list-cs*⟩`}{`⟨*item*⟩`}`

Defined in: etoolbox package.

Appends the given ⟨*item*⟩ to the list control sequence ⟨*list-cs*⟩. A blank item is not added. No expansion is performed on the item. [§2.7]

`\listcsadd{`⟨*list-csname*⟩`}{`⟨*item*⟩`}`

Defined in: etoolbox package.

Similar to \listadd except the name (without the preceding backslash) of the list control sequence is used. [§2.7]

\listcseadd{⟨*list-csname*⟩}{⟨*item*⟩}

Defined in: etoolbox package.

Similar to \listeadd except the name (without the preceding backslash) of the list control sequence is used. [§2.7]

\listcsgadd{⟨*list-csname*⟩}{⟨*item*⟩}

Defined in: etoolbox package.

As \listcsadd but the assignment is global. [§2.7]

\listcsxadd{⟨*list-csname*⟩}{⟨*item*⟩}

Defined in: etoolbox package.

As \listcseadd but the assignment is global. [§2.7]

\listeadd{⟨*list-cs*⟩}{⟨*item*⟩}

Defined in: etoolbox package.

Appends the expansion of ⟨*item*⟩ to the list control sequence ⟨*list-cs*⟩. A blank item is not added. [§2.7]

\listgadd{⟨*list-cs*⟩}{⟨*item*⟩}

Defined in: etoolbox package.

As \listadd but the assignment is global. [§2.7]

\listofchanges[⟨*options*⟩]

Defined in: changes package.

Print a list or summary of the changes. At least two LaTeX runs are required to make the list appear. [§13.1]

`\listofdecisions`

Defined in: minutes package.

Displays a list of decisions. [§6.3]

`\listoffigures`

Defined in: Most classes that have the concept of document structure.

Inserts the list of figures. A second (possibly third) run is required to ensure the page numbering is correct. [§13.1]

`\listoftables`

Defined in: Most classes that have the concept of document structure.

Inserts the list of tables. A second (possibly third) run is required to ensure the page numbering is correct. [§6.3]

`\listxadd{`⟨*list-cs*⟩`}{`⟨*item*⟩`}`

Defined in: etoolbox package.

As `\listeadd` but the assignment is global. [§2.7]

`\llap{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Places ⟨*text*⟩ to the left of the reference point without taking up any space. [§11.1]

`\loadallproblems[`⟨*dataset*⟩`]`
`{`⟨*filename*⟩`}`

Defined in: probsoln package.

Loads all problems defined in ⟨*filename*⟩ and appends them to the specified data set in the order in which they are defined in the file. [§9.3]

`\LoadClass[`⟨*options*⟩`]{`⟨*name*⟩`}`
`[`⟨*version*⟩`]`

Defined in: LaTeX Kernel.

Used in class files to load another class. [§11.1]

`\loadexceptproblems[`⟨*dataset*⟩`]{`⟨*exception-list*⟩`}{`⟨*filename*⟩`}`

Defined in: probsoln package.

Loads the problems defined in the given filename except those listed in ⟨*exception list*⟩ and adds them to the given dataset. [§9.3]

`\loadrandomexcept[`⟨*dataset*⟩`]{`⟨*n*⟩`}{`⟨*filename*⟩`}{`⟨*exception-list*⟩`}`

Defined in: probsoln package.

Loads ⟨*n*⟩ randomly selected problems defined in the given filename excluding those listed in ⟨*exception list*⟩ and adds them to the given dataset. [§9.3]

`\loadrandomproblems[`⟨*dataset*⟩`]{`⟨*n*⟩`}{`⟨*filename*⟩`}`

Defined in: probsoln package.

Loads ⟨*n*⟩ randomly selected problems defined in the given filename and adds them to the given dataset. [§9.3]

`\loadselectedproblems[`⟨*dataset*⟩`]{`⟨*labels*⟩`}{`⟨*filename*⟩`}`

Defined in: probsoln package.

Loads the listed problems defined in the given filename and adds them to the given dataset. [§9.3]

`\location{`⟨*text*⟩`}`

Defined in: letter class.

Specifies the sender's additional location information. [§3.1]

`\location{`⟨*place*⟩`}`

Defined in: minutes package.

Specifies the location of the meeting. [§6.3]

`\long`⟨*assignment*⟩

Defined in: TEX primitive.

An assignment prefix that indicates the following assignment (such as `\def`) should define a long command. [§2.1]

`\begin{longtable}[`⟨*horizontal alignment*⟩`]{`⟨*column specifiers*⟩`}`

Defined in: longtable package.

Like a combination of the table and tabular environments, except it can span multiple pages. [§2.6]

`\loop`⟨*code*⟩`\if... \repeat`

Defined in: LATEX Kernel.

Repeats code while the given condition is true. [§2.7]

`\lstinputlisting{`⟨*options*⟩`}{`⟨*filename*⟩`}`

Defined in: listings package.

Reads in ⟨*filename*⟩ and typesets the contents as displayed code. [§9.4]

`\begin{lstlisting}[`⟨*options*⟩`]`

Defined in: listings package.

Typesets the contents of the environment as displayed code. [§8.0]

M

`\makeatletter`

Defined in: LATEX Kernel.

Changes the category code of the at character (@) to "letter" so that it can be used in control sequence names. See also `\makeatother`. [§2.1]

`\makeatother`

Defined in: LaTeX Kernel.

Changes the category code of the at character (@) to "other". This means that it can no longer be used in control sequence names. See also `\makeatletter`. [§2.1]

`\makebox[`⟨*width*⟩`][`⟨*align*⟩`]{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Puts ⟨*text*⟩ in a box, prohibiting a line break in the contents. [§9.1]

`\makebox(`⟨*w*⟩`,`⟨*h*⟩`)[`⟨*align*⟩`]{`⟨*text*⟩`}`

Defined in: picture environment.

Creates a box picture object without a frame. [§10.1]

`\MakeButtonField{`⟨*text*⟩`}`

Defined in: hyperref package.

The format for push button labels. [§11.2]

`\MakeCheckField{`⟨*width*⟩`}{`⟨*height*⟩`}`

Defined in: hyperref package.

The display for check fields. [§11.2]

`\MakeChoiceField{`⟨*width*⟩`}{`⟨*height*⟩`}`

Defined in: hyperref package.

The display for choice fields. [§11.2]

`\makelabels`

Defined in: letter class and envlab package (preamble only).

Switches on the label-generating function. [§3.6]

`\MakeRadioField{`⟨*width*⟩`}{`⟨*height*⟩`}`

Defined in: hyperref package.

The display for radio fields. [§11.2]

`\MakeTextField{`⟨*width*⟩`}{`⟨*height*⟩`}`

Defined in: hyperref package.

The display for text fields. [§11.2]

`\maketitle`

Defined in: Most classes that have the concept of a title page.

Generates the title page (or title block). This command is usually placed at the beginning of the document environment. [§5.1]

`\MakeUppercase{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Converts its argument to upper case. [§3.6]

`\marginpar[`⟨*left*⟩`]{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Puts ⟨*text*⟩ in the margin. If the document provides left and right margins (for example, a two-sided document) ⟨*left*⟩ indicates the text to use if the margin is on the left and ⟨*text*⟩ indicates the text to use if the margin is on the right. [§6.5]

`\mbox{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Ensures that the given text doesn't contain a line break. [§7.5]

`\medskip`

Defined in: LaTeX Kernel.

Inserts a medium-sized vertical space. The size is given by the length `\medskipamount`. [§12.3]

`\midrule[`⟨*wd*⟩`]`

Defined in: booktabs package.

Horizontal rule to go below headings row of a tabular environment. [§2.6]

`\begin{minipage}[`⟨*pos*⟩`]`
`[`⟨*height*⟩`]{`⟨*width*⟩`}`

Defined in: LaTeX Kernel.

Makes a box with line-wrapped contents. (See also `\parbox`.) [§3.6]

`\begin{Minutes}{`⟨*title*⟩`}`

Defined in: minutes package.

Contains the minutes from a meeting. [§6.3]

`\minutesdate{`⟨*date*⟩`}`

Defined in: minutes package.

Specifies the date of the meeting. [§6.3]

`\minutetaker{`⟨*name*⟩`}`

Defined in: minutes package.

Specifies the name of the minute taker. [§6.3]

`missing(`⟨*ref*⟩`)`

Defined in: `arara` directive.

Evaluates to true if the given file doesn't exist. The argument ⟨*ref*⟩ may either be a string "⟨*extension*⟩" which indicates the file extension or a file reference `toFile("`⟨*filename*⟩`")`. [§1.2]

`\missing[`⟨*excused-names*⟩`]{`⟨*no-excuse*

names⟩}

Defined in: minutes package.

List of absentees. [§6.3]

`\missingExcused`{⟨*names*⟩}

Defined in: minutes package.

List of absentees who gave an excuse. [§6.3]

`\missingNoExcuse`{⟨*names*⟩}

Defined in: minutes package.

List of absentees who didn't give an excuse. [§6.3]

`\mlabel`{⟨*from-address*⟩}{⟨*to-address*⟩}

Defined in: envlab package.

Manually creates an address label. [§3.6]

`\moderation`{⟨*name*⟩}

Defined in: minutes package.

Specifies the name of the meeting moderator. [§6.3]

`\month`

Defined in: TEX primitive.

The current month number. [§7.2]

`\multicolumn`{⟨*cols-spanned*⟩}{⟨*col-specifier*⟩}{⟨*text*⟩}

Defined in: LATEX Kernel.

Spans multiple columns in a tabular-style environment. See Volume 1 [92, §4.6.2]. [§2.7]

`\multiply`⟨*register*⟩ by ⟨*value*⟩

Defined in: TEX primitive.

Multiplies the value stored in ⟨*register*⟩ by ⟨*value*⟩. The by keyword may be omitted. [§2.1]

`\multiput(`⟨*x*⟩`,`⟨*y*⟩`)(`⟨*inc-x*⟩`,`⟨*inc-y*⟩`){`⟨*n*⟩`}{`⟨*object*⟩`}`

Defined in: LaTeX Kernel.

For use in the picture environment, this puts ⟨*n*⟩ copies of ⟨*object*⟩, starting at position (⟨*x*⟩,⟨*y*⟩) and advancing the position by (⟨*inc-x*⟩,⟨*inc-y*⟩) each time. [§10.1]

N

`\name{`⟨*text*⟩`}`

Defined in: letter class.

Specifies the sender's name. [§3.1]

`\namefrom{`⟨*name*⟩`}`

Defined in: newlfm class.

Specifies the sender's name. [§3.3]

`\nameto{`⟨*name*⟩`}`

Defined in: newlfm class.

Specifies the recipient's name. [§3.3]

`\NeedsTeXFormat{`⟨*format*⟩`}[`⟨*version*⟩`]`

Defined in: LaTeX Kernel.

This should be the first statement of any class or package identifying the required TeX format. For a LaTeX 2ε class or package this should be `LaTeX2e`. (Other formats may not define this command.) [§7.3]

`\newbool{`⟨*name*⟩`}`

Defined in: etoolbox package.

Defines a new boolean flag called ⟨*name*⟩. [§9.4]

`\newboolean{`⟨*name*⟩`}`

Defined in: ifthen package.

Defines a new boolean variable. [§9.4]

`\newcommand{`⟨*cmd*⟩`}[`⟨*n-args*⟩`][`⟨*default*⟩`]{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Defines a new command. (See Volume 1 [92, §8].) [§2.1]

`\newcount`⟨*cs*⟩

Defined in: TeX primitive.

Defines a new count register. (Note to be confused with the LaTeX `\newcounter` command.) [§2.1]

`\newcounter{`⟨*counter*⟩`}[`⟨*outer-counter*⟩`]`

Defined in: LaTeX Kernel.

Defines a new counter (see Volume 1 [92, §11]). [§2.1]

`\newdynamicframe[`⟨*page-list*⟩`]{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}[`⟨*label*⟩`]`

Defined in: flowfram package.

Defines a new dynamic frame. The starred version adds a rectangular border to the frame. [§10.5]

`\newenvironment{`⟨*env-name*⟩`}[`⟨*n-args*⟩`][`⟨*default*⟩`]{`⟨*begin-code*⟩`}{`⟨*end-code*⟩`}`

Defined in: LaTeX Kernel.

Defines a new environment. [§11.1]

`\newflowframe[`⟨*page-list*⟩`]{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}[`⟨*label*⟩`]`

Defined in: flowfram package.

Defines a new flow frame. The starred version adds a rectangular border to the frame. [§10.5]

`\newglossaryentry{⟨label⟩}{⟨key-val list⟩}`

Defined in: glossaries package.

Defines a new glossary entry or term. [§2.1]

`\newlength⟨length cs⟩`

Defined in: LaTeX Kernel.

Defines a new length register called ⟨*length cs*⟩. [§6.5]

`\begin{newlfm}`

Defined in: newlfm class.

The body of the letter, fax or memo. [§3.3]

`\newlfmP{⟨option list⟩}`

Defined in: newlfm class.

Sets the given options. [§3.3]

`\newline`

Defined in: LaTeX Kernel.

Forces a line break. [§5.2]

`\newpage`

Defined in: LaTeX Kernel.

Forces a page break leaving a ragged bottom. [§10.3]

`\newproblem[⟨n⟩][⟨default-args⟩]{⟨label⟩}{⟨problem⟩}{⟨solution⟩}`

Defined in: probsoln package.

A shortcut that defines a problem with an associated solution using the defproblem, onlysolution and solution environments. [§9.3]

`\newproblem*[⟨n⟩][⟨default-args⟩]{⟨label⟩}{⟨problem⟩}`

Defined in: probsoln package.

A shortcut that defines a problem using the defproblem environment. [§9.3]

`\newstaticframe[`⟨*page-list*⟩`]{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}[`⟨*label*⟩`]`

Defined in: flowfram package.

Defines a new static frame. The starred version adds a rectangular border to the frame. [§10.5]

`\newwatermark[`⟨*options*⟩`]{`⟨*mark*⟩`}`

Defined in: xwatermark package.

Specifies a watermark. [§6.4]

`\nextmeeting{`⟨*date and time*⟩`}`

Defined in: meetingmins class.

Displays the next meeting date and time. [§6.3]

`\noaddpoints`

Defined in: exam class.

Disable the point-totalling commands. [§9.1]

`\NoBgThisPage`

Defined in: background package.

Indicates the background shouldn't be displayed on the current page. (For use with the `all` pages option.) [§6.4]

`\nobibliography{`⟨*bib-list*⟩`}`

Defined in: bibentry package.

Analogous to `\bibliography`, this command writes the information to the `.aux` file that's required by BibTEX but doesn't display anything in the document. Individual bibliography entries can

then be displayed using \bibentry. [§5.2]

\nocite{⟨*key list*⟩}

Defined in: LaTeX Kernel.

Like \cite except that it doesn't produce any text. The ⟨*key list*⟩ may be just an asterisk * to indicated that all citations in the .bib file (or files) should be included in the bibliography. [§5.1]

\node[⟨*options*⟩] (⟨*label*⟩) {⟨*text*⟩};

Defined in: tikz package.

Node specification, for use within the tikzpicture environment. [§7.5]

\nodepart{⟨*part*⟩}

Defined in: tikz package.

For use within the contents of a split node, this moves from the current split to the split identified by ⟨*part*⟩. [§7.5]

\noexpand⟨*token*⟩

Defined in: TeX primitive.

Prevents ⟨*token*⟩ from being expanded in an expandable context. [§2.1]

\noindent

Defined in: LaTeX Kernel.

Suppress the indentation that would usually occur at the start of the next paragraph. [§9.1]

\noprintanswers

Defined in: exam class.

Disable (don't show) the solutions. [§9.1]

\normalsize

Defined in: LaTeX Kernel.

Switches to normal sized text. [§10.3]

`\number`⟨*num*⟩

Defined in: TEX primitive.

Displays the decimal value of ⟨*num*⟩. (Any redundant leading zeros are stripped.) [§2.1]

`\numexpr`⟨*integer expression*⟩

Defined in: ε-TEX primitive.

Expands to the value given by the integer expression. [§2.1]

`\numparts`

Defined in: exam class.

The number of parts. [§9.1]

`\numpoints`

Defined in: exam class.

The total number of points. [§9.1]

`\numquestions`

Defined in: exam class.

The number of questions. [§9.1]

`\numsubparts`

Defined in: exam class.

The number of sub-parts. [§9.1]

`\numsubsubparts`

Defined in: exam class.

The number of sub-sub-parts. [§9.1]

O

`\oddsidemargin`

Defined in: LATEX Kernel.

A length containing the horizontal offset for odd pages. See `\hoffset`. [§11.0]

`\onecolumn`

Defined in: LaTeX Kernel.

Issues a page break and switches to one column mode. [§10.5]

`\begin{oneparcheckboxes}`

Defined in: exam class.

Checkbox choices are listed inline. Items are specified via `\choice`. [§9.1]

`\begin{oneparchoices}`

Defined in: exam class.

Labelled choices are listed inline. Items are specified via `\choice`. [§9.1]

`\begin{onlyproblem}[`⟨*option*⟩`]`

Defined in: probsoln package.

Only displays its contents if the `showanswers` boolean flag is off. [§9.3]

`\begin{onlysolution}[`⟨*option*⟩`]`

Defined in: probsoln package.

Only displays its contents if the `showanswers` boolean flag is on. [§9.3]

`\opening{`⟨*salutation*⟩`}`

Defined in: Classes that define the letter environment.

Typesets the salutation at the start of the letter. [§3.1]

`\opinion{`⟨*main*⟩`}{`⟨*differing*⟩`}`

Defined in: minutes package.

Indicates a discussion on an opinion. [§6.3]

`\begin{Opinions}`

Defined in: minutes package.

A list like environment to format opinions. [§6.3]

`\oval(`⟨*w*⟩`,`⟨*h*⟩`)[`⟨*segment*⟩`]`

Defined in: LaTeX Kernel.

For use in the argument of `\put`, this command draws an oval of the given width and height. The optional argument indicates to only draw a quarter or half oval. [§10.1]

P

`\p@enumi`

Defined in: LaTeX Kernel.

Prefixed used when cross-referencing the enumi counter. [§6.5]

`\p@enumii`

Defined in: LaTeX Kernel.

Prefixed used when cross-referencing the enumii counter. [§6.5]

`\p@enumiii`

Defined in: LaTeX Kernel.

Prefixed used when cross-referencing the enumiii counter. [§6.5]

`\p@enumiv`

Defined in: LaTeX Kernel.

Prefixed used when cross-referencing the enumiv counter. [§6.5]

`\pagestyle{`⟨*style*⟩`}`

Defined in: LaTeX Kernel.

Sets the style of the headers and footers. [§10.1]

`\paperheight`

Defined in: LaTeX Kernel.

A length containing the total height of the page. See also `\textheight`. [§10.3]

`\paperwidth`

Defined in: LaTeX Kernel.

A length containing the total width of the page. See also `\textwidth`. [§10.3]

`\par`

Defined in: TeX primitive (context-dependent).

Insert a paragraph break. Usually just a blank line in the source code is used instead of `\par`. This command is typically only used within command or environment definitions or when it's important to remind authors that a paragraph break is required at a certain point where a blank line may appear accidental or may be overlooked. (For example, in `{\centering Some centred text.\par}` it's a reminder that there must be a paragraph break before the closing `}` to ensure the `\centering` declaration has an effect.) [§1.0]

`\paragraph[`⟨*short-title*⟩`]{`⟨*title*⟩`}`

Defined in: Most classes that have the concept of document structure.

Inserts a subsubsubsection header. Most classes default to an unnumbered running header for this sectional unit. [§6.5]

`\parbox[`⟨*pos*⟩`][`⟨*height*⟩`]{`⟨*width*⟩`}{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Makes a box with line-wrapped contents. (More restrictive than minipage.) [§7.5]

`\parindent`

Defined in: LaTeX Kernel.

A length register that stores the indentation at the start of paragraphs. [§2.1]

`\parshape=`⟨*n*⟩ ⟨i_1⟩ ⟨l_1⟩ ... ⟨i_n⟩ ⟨l_n⟩

Defined in: TeX primitive.

Creates a shaped paragraph. [§10.5]

`\part[`⟨*points*⟩`]`

Defined in: exam class.

Inside the parts environment, this command starts a new numbered question part, with optionally the number of points the part is worth. Outside the parts environment this command works as a standard sectioning command. [§9.1]

`\part[`⟨*short-title*⟩`]{`⟨*title*⟩`}`

Defined in: Most classes that have the concept of document structure.

Inserts a part sectional unit. [§9.1]

`\participant{`⟨*names*⟩`}`

Defined in: minutes package.

Specifies the names of the people present. [§6.3]

`\begin{parts}`

Defined in: exam class.

Contains all the question parts. [§9.1]

`\PassOptionsToClass{`⟨*option-list*⟩`}{`⟨*class*⟩`}`

Defined in: LaTeX Kernel.

Passes the given options to the given class. (Must be used before the class is loaded.) [§11.1]

`\path` ⟨*specification*⟩;

Defined in: tikz package.

For use within the tikzpicture environment. [§7.5]

`\pdfcreationdate`

Defined in: PDFTEX primitive.

The date and time at the start of the current TEX run. This expands to `D:`⟨*YYYY*⟩⟨*MM*⟩⟨*DD*⟩⟨*hh*⟩⟨*mm*⟩⟨*ss*⟩⟨*time zone*⟩. This primitive is also available with LuaTEX but not XETEX. [§7.4]

`\pdffilemoddate{`⟨*filename*⟩`}`

Defined in: PDFTEX primitive.

The modification date and time of the file ⟨*filename*⟩. This expands to `D:`⟨*YYYY*⟩⟨*MM*⟩⟨*DD*⟩⟨*hh*⟩⟨*mm*⟩⟨*ss*⟩⟨*time zone*⟩. Unlike `\pdfcreationdate`, this primitive isn't provided by LuaTEX. [§7.4]

`\pgfcalendar{`⟨*prefix*⟩`}{`⟨*start-date*⟩`}{`⟨*end-date*⟩`}{`⟨*code*⟩`}`

Defined in: pgfcalendar package.

A for loop that iterates from ⟨*start-date*⟩ to ⟨*end-date*⟩ and performs ⟨*code*⟩ at each iteration. [§7.5]

`\pgfcalendarbeginiso`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. The ⟨*start-date*⟩ parameter in ISO format. [§7.5]

`\pgfcalendarbeginjulian`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. The ⟨*start-date*⟩

parameter converted to a Julian day number. [§7.5]

`\pgfcalendarcurrentday`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. This expands to the day of the month for the current iteration. [§7.5]

`\pgfcalendarcurrentjulian`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. This is a TEX count register that holds the Julian day number for the current iteration. [§7.5]

`\pgfcalendarcurrentmonth`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. This expands to the month for the current iteration. [§7.5]

`\pgfcalendarcurrentweekday`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. This expands to the week day number for the current iteration (0 for Monday, 1 for Tuesday, etc). [§7.5]

`\pgfcalendarcurrentyear`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. This expands to the year for the current iteration. [§7.5]

`\pgfcalendardatetojulian {`⟨*date*⟩`}{`⟨*register*⟩`}`

Defined in: pgfcalendar package.

Converts the specified date into a Julian day number and stores the result in the given register. [§7.2]

`\pgfcalendarendiso`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. The ⟨*end-date*⟩ parameter in ISO format. [§7.5]

`\pgfcalendarendjulian`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. The ⟨*end-date*⟩ parameter converted to a Julian day number. [§7.5]

`\pgfcalendarifdate{`⟨*date*⟩`}{`⟨*test*⟩`}{`⟨*true-part*⟩`}{`⟨*false-part*⟩`}`

Defined in: pgfcalendar package.

Tests the specified date and does ⟨*true-part*⟩ if the test succeeds otherwise it does ⟨*false-part*⟩. [§7.2]

`\pgfcalendarjuliantodate{`⟨*Julian-day*⟩`}{`⟨*year-cs*⟩`}{`⟨*month-cs*⟩`}{`⟨*day-cs*⟩`}`

Defined in: pgfcalendar package.

Converts a Julian day number into an ISO-date. The resulting numbers are stored in the ⟨*year-cs*⟩, ⟨*month-cs*⟩ and ⟨*day-cs*⟩ control sequences. [§7.2]

`\pgfcalendarjuliantoweekday{`⟨*Julian-day*⟩`}{`⟨*register*⟩`}`

Defined in: pgfcalendar package.

Converts a Julian day number into a week day number (0 for Monday, 1 for Tuesday, etc). The

resulting number is stored in the given register. [§7.2]

`\pgfcalendarmonthname{`⟨*month-number*⟩`}`

Defined in: pgfcalendar package.

Expands to a textual representation of the month. [§7.3]

`\pgfcalendarmonthshortname {`⟨*month-number*⟩`}`

Defined in: pgfcalendar package.

Expands to an abbreviated textual representation of the month. [§7.3]

`\pgfcalendarprefix`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. The ⟨*prefix*⟩ parameter. [§7.5]

`\pgfcalendarshorthand{`⟨*kind*⟩`} {`⟨*representation*⟩`}`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. Expands to a representation of the current day, month, year or day of week. [§7.5]

`\pgfcalendarsuggestedname`

Defined in: pgfcalendar package.

For use within the ⟨*code*⟩ part of `\pgfcalendar`. If the ⟨*prefix*⟩ parameter is empty, this command expands to empty otherwise it expands to ⟨*prefix*⟩-⟨*YYYY*⟩-⟨*MM*⟩-⟨*DD*⟩, where ⟨*YYYY*⟩ is the current year, ⟨*MM*⟩ is the current month and ⟨*DD*⟩ is the current day of the month. [§7.5]

`\pgfcalendarweekdayname{`⟨*week day number*⟩`}`

Defined in: pgfcalendar package.

Expands to a textual representation of the day of week. [§7.3]

`\pgfcalendarweekdayshortname {`⟨*week day number*⟩`}`

Defined in: pgfcalendar package.

Expands to an abbreviated textual representation of the day of week. [§7.3]

`\pgfkeys{`⟨*options*⟩`}`

Defined in: pgfkeys package.

Sets the options related to the pgf package and any related packages that use pgf's option interface. [§12.5]

`\pgfmathdeclarerandomlist{`⟨*list-name*⟩`}{{`⟨*item1*⟩`}{`⟨*item2*⟩`}...}`

Defined in: pgfmath package.

Defines a list that can have items randomly selected from it (using `\pgfmathrandomitem`). [§9.5]

`\pgfmathparse{`⟨*expression*⟩`}`

Defined in: pgfmath package.

Parses the given mathematical expression and stores the result in `\pgfmathresult`. [§9.5]

`\pgfmathprintnumber{`⟨*number*⟩`}`

Defined in: pgfmath package.

Pretty-prints the given number. [§12.5]

`\pgfmathrandominteger{`⟨*cs*⟩`}` `{`⟨*minimum*⟩`}{`⟨*maximum*⟩`}`

Defined in: pgfmath package.

Defines the control sequence ⟨*cs*⟩ to be a pseudo-randomly generated number between ⟨*minimum*⟩ and ⟨*maximum*⟩ (inclusive). [§9.5]

`\pgfmathrandomitem{`⟨*cs*⟩`}{`⟨*list-name*⟩`}`

Defined in: pgfmath package.

Randomly select an item from the given list, which should have previously been declared using `\pgfmathdeclarerandomlist`. [§9.5]

`\pgfmathsetseed{`⟨*n*⟩`}`

Defined in: pgfmath package.

Sets the seed for the random number generator. [§9.5]

`\pgfplotsset{`⟨*options*⟩`}`

Defined in: pgfplots package.

Sets the options governing plot styles. [§12.5]

`\phonefrom{`⟨*text*⟩`}`

Defined in: newlfm class.

Specifies the sender's phone number. [§3.3]

`\begin{picture}`
`(`⟨*width*⟩`,`⟨*height*⟩`)(`⟨*llx*⟩`,`⟨*lly*⟩`)`

Defined in: LaTeX Kernel.

An environment that produces a box with the given dimensions (specified in terms of `\unitlength`). The second argument ⟨*llx,lly*⟩ is optional and specifies the co-ordinates of the lower left corner, (0,0) if omitted. The contents of the environment should consist of commands such as `\put` that puts text or lines in the box. [§10.0]

`\pie[`⟨*options*⟩`]{`⟨*list*⟩`}`

Defined in: pgf-pie package.

For use within the tikzpicture environment. Draws a pie chart. [§12.2]

`\points{`⟨*n*⟩`}`

Defined in: exsheets package.

Displays the number of points. The starred version omits the unit. [§9.2]

`\pointsdroppedatright`

Defined in: exam class.

Switches off the automatic point placement. (Used with `\droppoints`.) [§9.1]

`\pointsinmargin`

Defined in: exam class.

Puts the points in the left margin. [§9.1]

`\pointsinrightmargin`

Defined in: exam class.

Puts the points in the left margin. [§9.1]

`\pointsum`

Defined in: exsheets package.

Displays the total number of points, excluding bonus points. (Requires two LaTeX runs to ensure it's up to date.) The starred version omits the unit. [§9.2]

`\begin{Postscript}`

Defined in: minutes package.

Displays additional information that doesn't form part of the minutes. [§6.3]

`\postscript{`⟨*text*⟩`}`

Defined in: minutes package.

Displays additional information that doesn't form part of the minutes. [§6.3]

`\pounds`

Defined in: LaTeX Kernel.

Pound £ symbol. This robust command may be used in math or text mode. [§2.2]

`\ppsitem{`⟨*text*⟩`}`

Defined in: newlfm class.

Specifies the PPS line. [§3.3]

`\PRaddress{`⟨*address*⟩`}`

Defined in: pressrelease class.

Specifies the company's address. [§6.2]

`\PRcompany{`⟨*name*⟩`}`

Defined in: pressrelease class.

Specifies the company's name. [§6.2]

`\PRdepartment{`⟨*name*⟩`}`

Defined in: pressrelease class.

Specifies the department's name. [§6.2]

`\PRemail{`⟨*address*⟩`}`

Defined in: pressrelease class.

Specifies the company's email address. [§6.2]

`\begin{pressrelease}`

Defined in: pressrelease class.

Creates a press release. [§6.2]

`\preto`⟨*cs*⟩`{`⟨*code*⟩`}`

Defined in: etoolbox package.

Prepends ⟨*code*⟩ to the definition of the control sequence ⟨*cs*⟩. Use `\gpreto` for a global assignment. [§2.1]

`\PRfax{`⟨*number*⟩`}`

Defined in: pressrelease class.

Specifies the company's fax number. [§6.2]

`\PRheadline{`⟨*text*⟩`}`

Defined in: pressrelease class.

Specifies the headline. [§6.2]

`\PRhours{`⟨*times*⟩`}`

Defined in: pressrelease class.

Specifies the company's opening hours. [§6.2]

`\PrintAddress{`⟨*address*⟩`}`

Defined in: envlab package.

Used internally to typeset the recipient's address. [§3.6]

`\printanswers`

Defined in: exam class.

Enable (show) the solutions. [§9.1]

`\PrintBigLabel{`⟨*from-address*⟩`}{`⟨*to-address*⟩`}`

Defined in: envlab package.

Used internally to typeset the big labels. [§3.6]

`\PrintReturnAddress{`⟨*address*⟩`}`

Defined in: envlab package.

Used internally to typeset the sender's address. [§3.6]

`\printsolutions[`⟨*settings*⟩`]`

Defined in: exsheets package.

Prints all the solutions. [§9.2]

`\priormins`

Defined in: meetingmins class.

Displays the text "The minutes of the previous meeting were approved". [§6.3]

`\PRlocation{`⟨*location*⟩`}`

Defined in: pressrelease class.

Specifies the company's location. [§6.2]

`\PRlogo{`⟨*code*⟩`}`

Defined in: pressrelease class.

Specifies the code to use to produce the logo. Typically, this will usually just involve the `\includegraphics` command. [§6.2]

`\PRmobile{`⟨*number*⟩`}`

Defined in: pressrelease class.

Specifies the company's mobile phone number. [§6.2]

`\Pro`

Defined in: minutes package.

For use within the Argumentation environment, this indicates the start of an important item in favour of the argument. [§6.3]

`\pro`

Defined in: minutes package.

For use within the Argumentation environment, this indicates the start of an item in favour of the argument. [§6.3]

`\ProbSolnFragileExt`

Defined in: probsoln package.

Stores the extension of the temporary file used by probsoln to

work with verbatim code. (Defaults to `vrb`.) [§9.3]

`\ProbSolnFragileFile`

Defined in: probsoln package.

Stores the basename of the temporary file used by probsoln to work with verbatim code. (Defaults to `\jobname`.) [§9.3]

`\ProcessOptions`

Defined in: LaTeX Kernel.

Process all the declare options. [§11.1]

`\ProjectTitle{`⟨*title*⟩`}`

Defined in: invoice package.

For use within the invoice environment, this command is used to specify the project title. [§4.2]

`\begin{proof}[`⟨*proof name*⟩`]`

Defined in: beamer class.

An environment for typesetting proofs. [§8.0]

`\protect`⟨*command*⟩

Defined in: LaTeX Kernel.

Used in a moving argument to prevent a fragile command from expanding. [§9.4]

`\protected@csedef{`⟨*cs-name*⟩`}`⟨*arg-syntax*⟩`{`⟨*definition*⟩`}`

Defined in: etoolbox package.

Similar to `\protected@edef` but the name of the control sequence (without the leading backslash) is used. [§2.1]

`\protected@csxdef{`⟨*cs-name*⟩`}`⟨*arg-syntax*⟩`{`⟨*definition*⟩`}`

Defined in: etoolbox package.

Similar to `\protected@xdef` but the name of the control sequence (without the leading backslash) is used. [§2.1]

`\protected@edef`⟨*cs*⟩⟨*arg-syntax*⟩`{`⟨*definition*⟩`}`

Defined in: LaTeX Kernel.

Similar to `\edef` but a protected expansion is performed on ⟨*definition*⟩. [§2.1]

`\protected@write{`⟨*output-stream*⟩`}{`⟨*init-code*⟩`}{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Writes ⟨*text*⟩ to the file identified by ⟨*output stream*⟩. [§9.4]

`\protected@xdef`⟨*cs*⟩⟨*arg-syntax*⟩`{`⟨*definition*⟩`}`

Defined in: LaTeX Kernel.

Similar to `\xdef` but a protected expansion is performed on ⟨*definition*⟩. [§2.1]

`\providecommand{`⟨*cmd*⟩`}[`⟨*n-args*⟩`][`⟨*default*⟩`]{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Defines the command only if it doesn't already exist. [§2.1]

`\ProvidesClass{`⟨*name*⟩`}[`⟨*version*⟩`]`

Defined in: LaTeX Kernel.

Identifies the class name and optionally the version date. [§11.1]

`\ProvidesPackage{`⟨*name*⟩`}[`⟨*version*⟩`]`

Defined in: LaTeX Kernel.

Identifies the package name and optionally the version date. [§4.2]

`\PRphone{`⟨*number*⟩`}`

Defined in: pressrelease class.

Specifies the company's phone number. [§6.2]

`\PRsubheadline{`⟨*text*⟩`}`

Defined in: pressrelease class.

Specifies the sub-headline. [§6.2]

`\PRurl{`⟨*website*⟩`}`

Defined in: pressrelease class.

Specifies the company's website. [§6.2]

`\ps`

Defined in: Classes that define the letter environment.

Indicates the start of the postscript. [§3.1]

`\psbarcode[`⟨*options*⟩`]{`⟨*text or filename*⟩`}{`⟨*PS options*⟩`}{`⟨*type*⟩`}`

Defined in: pst-barcode package.

Generates a bar code. [§10.4]

`\psitem{`⟨*text*⟩`}`

Defined in: newlfm class.

Specifies the PS line. [§3.3]

`\PSNrandom{`⟨*register*⟩`}{`⟨*n*⟩`}`

Defined in: probsoln package.

Randomly generates an integer between 1 and ⟨*n*⟩ (inclusive) and stores the result in the given register. [§9.3]

`\PSNrandseed{`⟨*n*⟩`}`

Defined in: probsoln package.

Sets the random number generator seed. [§9.3]

`\begin{pspicture}[`⟨*baseline*⟩`]`
`(`⟨*llx*⟩`,`⟨*lly*⟩`)(`⟨*urx*⟩`,`⟨*ury*⟩`)`

Defined in: pstricks package.

Environment for drawing vector graphics. [§10.4]

`\PushButton[`⟨*options*⟩`]{`⟨*label*⟩`}`

Defined in: hyperref package.

A push button that performs some action (for use within the Form environment.) [§11.2]

`\put(`⟨*x*⟩`,`⟨*y*⟩`){`⟨*object*⟩`}`

Defined in: LaTeX Kernel.

For use inside the picture environment, this puts ⟨*object*⟩ at the given co-ordinates (which are in terms of `\unitlength`). [§10.1]

Q

`\qbezier[`⟨*points*⟩`](`⟨*start-x*⟩`,`⟨*start-y*⟩`)(`⟨*control-x*⟩`,`⟨*control-y*⟩`)(`⟨*end-x*⟩`,`⟨*end-y*⟩`)`

Defined in: LaTeX Kernel.

For use in the picture, this draws a quadratic Bézier curve with the given start, end and curvature control points. [§10.1]

`\qquad`

Defined in: LaTeX Kernel.

Horizontal spacing command (twice as wide as `\quad`). [§11.1]

`\quad`

Defined in: LaTeX Kernel.

Horizontal spacing command equal to the current font's em value. [§3]

`\begin{question}[`⟨*options*⟩`]{`⟨*points*⟩`}`

Defined in: exsheets package.

Creates a new question. Both arguments are optional! [§9.2]

`\question[`⟨*points*⟩`]`

Defined in: exam class.

Starts a new numbered question, with optionally the number of points the question is worth. [§9.1]

`\begin{questions}`

Defined in: exam class.

Contains all the question in the exam. [§9.1]

R

`\raggedright`

Defined in: LaTeX Kernel.

Ragged-right paragraph justification. [§4.3]

`\random{`⟨*counter*⟩`}{`⟨*min*⟩`}{`⟨*max*⟩`}`

Defined in: probsoln package.

Randomly generates an integer between ⟨*min*⟩ and ⟨*max*⟩ (inclusive) and stores the result in the given counter. [§9.3]

`\re{`⟨*text*⟩`}`

Defined in: newlfm class.

Specifies the "re" part of a memo. [§6.1]

`\ref{`⟨*string*⟩`}`

Defined in: LaTeX Kernel.

References the value of the counter linked to the given label. A second (possibly third) run of LaTeX is required to ensure the cross-references are up-to-date. (See Volume 1 [92, §5.5].) [§1.2]

`\refstepcounter{`⟨*counter*⟩`}`

Defined in: LaTeX Kernel.

Increments the value of the given counter by one and allows the counter to be cross-referenced using `\ref` and `\label`. [§2.1]

`\regarding{`⟨*text*⟩`}`

Defined in: newlfm class.

Specifies subject of letter. [§3.3]

`\relax`

Defined in: TeX primitive.

Unexpandable nothing. [§2.1]

`\release{`⟨*text*⟩`}`

Defined in: newlfm class.

Specifies the release information. [§6.2]

`\renewcommand{`⟨*cmd*⟩`}[`⟨*n-args*⟩`][`⟨*default*⟩`]{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Redefines an existing command. (See Volume 1 [92, §8.2].) [§2.1]

`\renewenvironment{`⟨*env-name*⟩`}[`⟨*n-args*⟩`][`⟨*default*⟩`]{`⟨*begin-code*⟩`}{`⟨*end-code*⟩`}`

Defined in: LaTeX Kernel.

Redefines an existing environment. [§6.5]

`\replaced[`⟨*options*⟩`]{`⟨*new-text*⟩`}{`⟨*old-text*⟩`}`

Defined in: changes package.

Indicates that ⟨*old-text*⟩ has been replaced with ⟨*new text*⟩. [§13.1]

`\RequirePackage[`⟨*options*⟩`]{`⟨*name*⟩`}[`⟨*version*⟩`]`

Defined in: LaTeX Kernel.

Analogous to \usepackage but for use in class or package files. [§4.2]

\Reset[⟨*options*⟩]{⟨*label*⟩}

Defined in: hyperref package.

A reset button that resets the form (for use within the Form environment.) [§11.2]

\resizebox{⟨*h-length*⟩}{⟨*v-length*⟩}{⟨*text*⟩}

Defined in: graphicx package.

Scales the specified contents to the given dimensions. [§10.3]

\RestartCensoring

Defined in: censor package.

Switches on the censoring. [§6.4]

\result

Defined in: minutes package.

For use within the Argumentation environment, this indicates the start of an item indicating the result of the argument. [§6.3]

\returnaddress

Defined in: envlab package.

The return address. Defaults to the address given by \address [§3.6]

\rightarrow

Defined in: LaTeX Kernel (Math Mode).

Right arrow →. [§9.3]

\rlap{⟨*text*⟩}

Defined in: LaTeX Kernel.

Places ⟨*text*⟩ to the right of the reference point without taking up any space. [§11.0]

`\rmfamily`

Defined in: LaTeX Kernel.

Switches to the predefined serif font. (Defaults to Computer Modern Roman.) [§3]

`\Roman{`⟨*counter*⟩`}`

Defined in: LaTeX Kernel.

Displays counter value as an upper case Roman number. (I, II, III, ...) [§9.2]

`\roman{`⟨*counter*⟩`}`

Defined in: LaTeX Kernel.

Displays counter value as a lower case Roman number. (i, ii, iii, ...) [§6.5]

`\romannumeral`⟨*number*⟩

Defined in: TeX primitive.

Expands ⟨*number*⟩ to a lower case Roman numeral. [§6.5]

`\rotatebox[`⟨*option-list*⟩`]`
`{`⟨*angle*⟩`}{`⟨*text*⟩`}`

Defined in: graphicx package.

Rotates the given contents by the given angle. [§3.6]

`\rule[`⟨*raise*⟩`]{`⟨*width*⟩`}`
`{`⟨*height*⟩`}`

Defined in: LaTeX Kernel.

Draws a rule (filled rectangle) with the given width and height. If the optional argument is present, the rule is raised by that amount. [§3.6]

S

`\sb{`⟨*maths*⟩`}`

Defined in: LaTeX Kernel (Math Mode).

Displays its argument as a subscript. [§3]

`\begin{scope}[⟨options⟩]`

Defined in: tikz package.

For use within the tikzpicture environment, this provides a way of scoping options. [§7.5]

`\scshape`

Defined in: LaTeX Kernel.

Switches to the small-caps form of the current font family, if it exists. See Volume 1 [92, §4.5.1]. [§10.2]

`\begin{Secret}`

Defined in: minutes package.

The environment contents will only be displayed if the package option `Secret` is used. [§6.3]

`\secret{⟨secret text⟩}`

Defined in: minutes package.

The ⟨*secret text*⟩ will only be displayed if the package option `Secret` is used. [§6.3]

`\sectfont`

Defined in: leaflet class.

The font declaration used by the sectioning commands. [§10.3]

`\section[⟨short-title⟩]{⟨title⟩}`

Defined in: Most classes that have the concept of document structure.

Inserts a section header. [§1.2]

`\selectfont`

Defined in: LaTeX Kernel.

Used to select the current font after the font attributes have been

set via commands like \fontfamily. [§3]

\setbeamercovered{⟨*options*⟩}

Defined in: beamer class.

Sets how covered material should appear. [§8.1]

\SetBigLabel{⟨*width*⟩}{⟨*height*⟩}{⟨*top*⟩}{⟨*left*⟩}{⟨*sep*⟩}{⟨*columns*⟩}{⟨*rows*⟩}

Defined in: envlab package.

Sets the custom dimensions for the big labels. [§3.6]

\setbool{⟨*name*⟩}{⟨*state*⟩}

Defined in: etoolbox package.

Sets the state (`true` or `false`) of a boolean variable. (Local effect only.) A new boolean variable can be defined using \newbool and can be tested in the first argument of \ifbool. [§9.4]

\setboolean{⟨*name*⟩}{⟨*state*⟩}

Defined in: ifthen package.

Sets the state (`true` or `false`) of a boolean variable. (Local effect only.) A new boolean variable can be defined using \newboolean and can be tested in the first argument of \ifthenelse with \boolean. [§6.1]

\setcommittee{⟨*committee name*⟩}

Defined in: meetingmins class.

Sets the committee name. [§6.3]

\setcounter{⟨*counter*⟩}{⟨*number*⟩}

Defined in: LaTeX Kernel.

Sets the value of a counter. [§10.5]

`\setdate{`⟨*date*⟩`}`

Defined in: meetingmins class.

Sets the date of the meeting. [§6.3]

`\setdynamiccontents{`⟨*id*⟩`}`
`{`⟨*contents*⟩`}`

Defined in: flowfram package.

Sets the contents of a dynamic frame. [§10.5]

`\SetEnvelope[`⟨*top-margin*⟩`]`
`{`⟨*width*⟩`}{`⟨*height*⟩`}`

Defined in: envlab package.

Sets the custom envelope dimensions. [§3.6]

`\setkeys{`⟨*family*⟩`}{`⟨*options*⟩`}`

Defined in: keyval package.

Sets the options provided by the given option family [§9.3]

`\setkomavar{`⟨*name*⟩`}`
`[`⟨*description*⟩`]{`⟨*content*⟩`}`

Defined in: KOMA-Script classes.

Sets the content and optionally the description of a KOMA-Script variable. [§3.2]

`\setkomavar*{`⟨*name*⟩`}`
`{`⟨*description*⟩`}`

Defined in: KOMA-Script classes.

Sets the description of a KOMA-Script variable. [§3.2]

`\SetLabel{`⟨*width*⟩`}{`⟨*height*⟩`}`
`{`⟨*top*⟩`}{`⟨*left*⟩`}{`⟨*sep*⟩`}{`⟨*columns*⟩`}`
`{`⟨*rows*⟩`}`

Defined in: envlab package.

Sets the custom label dimensions. [§3.6]

`\setlength{`⟨*register*⟩`}{`⟨*dimension*⟩`}`

Defined in: LaTeX Kernel.

Sets the value of a length register. [§6.5]

`\setmargins{`⟨*top*⟩`}{`⟨*bottom*⟩`}{`⟨*left*⟩`}{`⟨*right*⟩`}`

Defined in: leaflet class.

Used to specify the page margins. [§10.3]

`\setmembers{`⟨*list*⟩`}`

Defined in: meetingmins class.

Sets the list of members. The chair should be indicated within the list with the `\chair` command. [§6.3]

`\setpresent{`⟨*list*⟩`}`

Defined in: meetingmins class.

Sets the list of people present. The chair should be indicated within the list with the `\chair` command. [§6.3]

`\setsocextension{`⟨*extension*⟩`}`

Defined in: changes package.

Sets the extension of the list or summary of changes file. [§13.1]

`\setstaticcontents{`⟨*id*⟩`}{`⟨*contents*⟩`}`

Defined in: flowfram package.

Sets the contents of a static frame. [§10.5]

`\setupdocument{`⟨*options*⟩`}`

Defined in: isodoc class.

Sets up the general document options [§3.4]

`\SetupExSheets[`⟨*module*⟩`]{`⟨*options*⟩`}`

Defined in: exsheets package.

Sets options for the exsheet package. [§9.2]

`\sffamily`

Defined in: LaTeX Kernel.

Switches to the predefined sans-serif font. (Defaults to Computer Modern Sans.) [§12.3]

`\shortstack[`⟨*align*⟩`]{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Produces a box with a single column of text with the reference point at the lower-left corner. Similar to using a single column tabular environment. [§10.1]

`\show`⟨*token*⟩

Defined in: TeX primitive.

Interrupts the document compilation and writes the definition of ⟨*token*⟩ to the transcript. [§2.1]

`\showanswers`

Defined in: probsoln package.

Shows the solutions (from that point onwards). [§9.3]

`\signature{`⟨*text*⟩`}`

Defined in: letter class; some other letter-like classes; minutes package.

Specifies the sender's (or author's) name to go after the closing text. [§3.1]

`\sin`

Defined in: LaTeX Kernel (Math Mode).

Typesets sin function name. [§9.1]

`\small`

Defined in: Most document classes.

Switches to small sized text. [§6.4]

`\smallskip`

Defined in: LaTeX Kernel.

Inserts a small vertical space. The size is given by the length `\smallskipamount`. [§12.3]

`\begin{solution}[`⟨*options*⟩`]`

Defined in: exsheets package.

Contains the solution to the preceding question. [§9.2]

`\begin{solution}`

Defined in: probsoln package.

Starts a new paragraph (without indentation), typesets `\solutionname` in bold followed by a colon space and then the environment contents. [§9.3]

`\begin{solution}[`⟨*length*⟩`]`

Defined in: exam class.

Typesets its contents if the `answers` option is set otherwise it hides its contents, optionally replacing the contents with space for the student to fill in their answer. [§9.1]

`\solutionname`

Defined in: probsoln package.

The solution title text used by the solution environment. [§9.3]

`\begin{solutionorbox}[`⟨*length*⟩`]`

Defined in: exam class.

Typesets its contents if the `answers` option is set otherwise it

hides its contents, optionally replacing the contents with an empty box for the student to fill in their answer. [§9.1]

`\begin{solutionordottedlines}` `[⟨length⟩]`

Defined in: exam class.

Typesets its contents if the `answers` option is set otherwise it hides its contents, optionally replacing the contents with an area containing dotted lines for the student to fill in their answer. [§9.1]

`\begin{solutionorlines}` `[⟨length⟩]`

Defined in: exam class.

Typesets its contents if the `answers` option is set otherwise it hides its contents, optionally replacing the contents with a lined area for the student to fill in their answer. [§9.1]

`\sp{`⟨*maths*⟩`}`

Defined in: LaTeX Kernel (Math Mode).

Displays its argument as a superscript. [§3]

`\space`

Defined in: LaTeX Kernel.

A space character. Sometimes used instead of typing a space character when the author wants to emphasize that there's supposed to be a space there. For example, if the space occurs at the end of a line of code, using `\space` is a visual indication that the author hasn't simply forgotten to discard an unwanted EOL character. [§3.6]

`\Square`

Defined in: wasysym package or ifsym with the `geometry` option.

Produces a square symbol □. [§10.3]

`\startlabels`

Defined in: envlab package.

Starts the manual generation of labels (which should be made using `\mlabel`). [§3.6]

`\starttime{`⟨*time*⟩`}`

Defined in: minutes package.

Specifies the start time of the meeting. [§6.3]

`\begin{staticcontents}{`⟨*id*⟩`}`

Defined in: flowfram package.

Sets the contents of a static frame. [§10.5]

`\STExpenses`

Defined in: invoice package.

For use within the invoice environment, this command is used to make a subtotal appear for all the expenses hidden via `\EBCi` or `\EFCi`. [§4.2]

`\StopCensoring`

Defined in: censor package.

Switches off the redaction to produce an uncensored version of the document. [§6.4]

`\string`⟨*cs*⟩

Defined in: TEX primitive.

Converts the given control sequence into the list of characters that makes up that control sequence name (including the initial backslash) where each

character in the list has category code 12 (“other”) [§11.1]

`\begin{subitems}`

Defined in: meetingmins class.

A numbered list for use within subsections and sub-subsections. Use the items environment for lists within sections. [§6.3]

`\Submit[`⟨*options*⟩`]{`⟨*label*⟩`}`

Defined in: hyperref package.

A submit button that sends the data to the URL provided by the form’s action (for use within the Form environment.) [§11.2]

`\subparagraph[`⟨*short-title*⟩`]{`⟨*title*⟩`}`

Defined in: Most classes that have the concept of document structure.

Inserts a subsubsubsubsection header. Most classes default to an unnumbered running header for this sectional unit. [§6.5]

`\subpart[`⟨*points*⟩`]`

Defined in: exam class.

Starts a new numbered question sub-part, with optionally the number of points the sub-part is worth. [§9.1]

`\begin{subparts}`

Defined in: exam class.

Contains all the question sub-parts. [§9.1]

`\subsection[`⟨*short-title*⟩`]{`⟨*title*⟩`}`

Defined in: Most classes that have the concept of document structure.

Inserts a subsection header. [§6.3]

`\subsubpart[`⟨*points*⟩`]`

Defined in: exam class.

Starts a new numbered question sub-sub-part, with optionally the number of points the sub-sub-part is worth. [§9.1]

`\begin{subsubparts}`

Defined in: exam class.

Contains all the question sub-sub-parts. [§9.1]

`\subsubsection[`⟨*short-title*⟩`]{`⟨*title*⟩`}`

Defined in: Most classes that have the concept of document structure.

Inserts a subsubsection header. [§6.5]

`\subtitle{`⟨*title*⟩`}`

Defined in: Various classes or packages that have the concept of a subtitle.

Specifies the subtitle. Usually for use with `\maketitle` in a similar manner to `\title`. Some classes, such as beamer, also provide an optional argument for this command. [§6.3]

`\subtopic[`⟨*toc title*⟩`]{`⟨*title*⟩`}`

Defined in: minutes package.

Starts a new subtopic. [§6.3]

`\svnId`

Defined in: svninfo package.

Prints a summary of the Subversion information in the same form as the `Id` keyword anchor. [§13.2]

`\svnInfo␣$`⟨*Id*⟩`$␣`

Defined in: svninfo package.

Parses the Subversion `Id` keyword anchor. [§13.2]

`\svnInfoDate`

Defined in: svninfo package.

Prints the date in the form ⟨*YYYY*⟩-⟨*MM*⟩-⟨*DD*⟩ when the file was checked out or the current date if unknown. [§13.2]

`\svnInfoDay`

Defined in: svninfo package.

Prints the day (in ⟨*DD*⟩ form) when the file was checked out or the current day if unknown. [§13.2]

`\svnInfoFile`

Defined in: svninfo package.

Prints the name of the source file or `--sourcefile--` if unknown. [§13.2]

`\svnInfoHeadURL`

Defined in: svninfo package.

Prints the information obtained from the `HeadURL` keyword. [§13.2]

`\svnInfoLongDate`

Defined in: svninfo package.

Prints date in the form of `\today` when the file was checked out or the current date if unknown. [§13.2]

`\svnInfoMaxRevision`

Defined in: svninfo package.

Prints the maximum revision number for all the files in the document or `--maxrevision--` if unknown. [§13.2]

`\svnInfoMaxToday`

Defined in: svninfo package.

Prints date in the form of `\today` from the latest Subversion revision. [§13.2]

`\svnInfoMinRevision`

Defined in: svninfo package.

Prints the minimum revision number for all the files in the document or `--minrevision--` if unknown. [§13.2]

`\svnInfoMonth`

Defined in: svninfo package.

Prints the month (in ⟨*MM*⟩ form) when the file was checked out or the current month if unknown. [§13.2]

`\svnInfoOwner`

Defined in: svninfo package.

Prints the user name of the file owner or `--owner--` if unknown. [§13.2]

`\svnInfoRevision`

Defined in: svninfo package.

Prints the revision number of the checked out file or `--revision--` if unknown. [§13.2]

`\svnInfoTime`

Defined in: svninfo package.

Prints the time when the file was checked out or `--time--` if unknown. [§13.2]

`\svnInfoYear`

Defined in: svninfo package.

Prints the year when the file was checked out or the current year if unknown. [§13.2]

`\svnKeyword␣$⟨keyword⟩$␣`

Defined in: svninfo package.

Parses the Subversion keyword anchor. [§13.2]

T

`\begin{table}[⟨placement⟩]`

Defined in: Most classes except for those designed for correspondence or similarly restrictive documents.

Floats the contents to the nearest location according to the preferred placement options, if possible. Within the environment, `\caption` may be used one or more times, as required. (See Volume 1 [92, §7.2].) [§2.6]

`\tableofcontents`

Defined in: Most classes that have the concept of document structure.

Inserts the table of contents. A second (possibly third) run is required to ensure the page numbering is correct. [§6.3]

`\begin{tabular}[⟨v-pos⟩]{⟨col-specs⟩}`

Defined in: LaTeX Kernel (Text Mode).

Environment for lining things up in rows and columns. [§2.6]

`\task[⟨footnote-text⟩]{⟨name⟩}[⟨when⟩]{⟨text⟩}`

Defined in: minutes package.

Specifies a task. [§6.3]

`\task*[⟨when⟩]{⟨text⟩}`

Defined in: minutes package.

Specifies a task. [§6.3]

`\telephone{`⟨*text*⟩`}`

Defined in: letter class.

Specifies the sender's telephone number. [§3.1]

`\TeX`

Defined in: LaTeX Kernel.

Typesets the TeX logo. [§7.4]

`\text{`⟨*text*⟩`}`

Defined in: amsmath package (Math Mode).

Displays its argument in the normal text font (as opposed to the current maths font). [§7.4]

`\textasciicircum`

Defined in: LaTeX Kernel.

Circumflex ^ symbol. [§2.2]

`\textasciitilde`

Defined in: LaTeX Kernel.

Tilde ~ symbol. (If you are typing an URL, use the url package, which provides `\url{`⟨*address*⟩`}` that allows you to directly type ~ in the address.) [§2.2]

`\textbf{`⟨*text*⟩`}`

Defined in: LaTeX Kernel.

Renders ⟨*text*⟩ with a bold weight in the current font family, if it exists. (See Volume 1 [92, §4.5.1].) [§1.0]

`\textcolor[`⟨*model*⟩`]{`⟨*specs*⟩`}{`⟨*text*⟩`}`

Defined in: color and xcolor packages.

Sets ⟨*text*⟩ with the foreground colour according to the given ⟨*specs*⟩. [§2.3]

`\textdollar`

Defined in: textcomp package.

Displays the dollar symbol $. [§4.2]

`\begin{textenum}`

Defined in: probsoln package.

An inline numbered list environment. Each item may be started with the standard `\item` command, but may also be started with `\correctitem` or `\incorrectitem`. [§9.3]

`\texteuro`

Defined in: textcomp package.

Displays the Euro symbol €. [§4.2]

`\TextField[`⟨*options*⟩`]{`⟨*label*⟩`}`

Defined in: hyperref package.

A text field (for use within the Form environment.) [§11.2]

`\textheight`

Defined in: LaTeX Kernel.

A length containing the height of the typeblock. Note that the actual contents of the page may fall short of the text height (underfull vbox) or extend beyond it (overfull vbox). This measurement does not include the header and footer areas. [§10.5]

`\textifsymbol[`⟨*font-family*⟩`]{`⟨*number*⟩`}`

Defined in: ifsym package.

Selects the symbol identified by ⟨*number*⟩ from the given font family. [§11.1]

`\textsubscript{`⟨*text*⟩`}`

Defined in: fixltx2e package (now in LaTeX Kernel).

Makes ⟨*text*⟩ a subscript. Unlike \sb, this command is for use in text mode. [§13.1]

^{⟨*text*⟩}

Defined in: LaTeX Kernel.

Makes ⟨*text*⟩ a superscript. Unlike \sp, this command is for use in text mode. [§13.1]

\texttt{⟨*text*⟩}

Defined in: LaTeX Kernel.

Renders ⟨*text*⟩ in the predefined monospaced font. See Volume 1 [92, §4.5.1]. [§2.1]

\textwidth

Defined in: LaTeX Kernel.

A length containing the width of the typeblock. Note that the actual contents of the line may fall short of the line width (underfull hbox) or extend beyond it (overfull hbox). This width does not include the area for marginal notes. [§2.1]

\the⟨*register*⟩

Defined in: TeX primitive.

Expands ⟨*register*⟩ to the current value of the register. [§2.1]

\begin{thebibliography}{⟨*widest entry label*⟩}

Defined in: Most classes that define sectioning commands.

Bibliographic list. (See also \bibitem and \cite). [§3]

\theenumi

Defined in: LaTeX Kernel.

Displays the current value of the enumi (first level enumerate) counter. [§6.5]

`\theenumii`

Defined in: LaTeX Kernel.

Displays the current value of the enumii (second level enumerate) counter. [§6.5]

`\theenumiii`

Defined in: LaTeX Kernel.

Displays the current value of the enumiii (third level enumerate) counter. [§6.5]

`\theenumiv`

Defined in: LaTeX Kernel.

Displays the current value of the enumiv (fourth level enumerate) counter. [§6.5]

`\begin{theorem}[`⟨*title*⟩`]`

Defined in: beamer class.

An environment for typesetting theorems. [§8.0]

`\thequestion`

Defined in: exam class.

Displays the current question number. [§9.1]

`\thesection`

Defined in: LaTeX Kernel.

Displays the current value of the section counter [§6.5]

`\thicklines`

Defined in: LaTeX Kernel.

This declaration switches to a thick line width for lines drawn within the picture environment. [§10.1]

`\thispagestyle{`⟨*style*⟩`}`

Defined in: LaTeX Kernel.

Like \pagestyle but only affects the current page. [§3.1]

\thisproblem

Defined in: probsoln package.

May be used within the ⟨*body*⟩ argument of \foreachproblem to display the current problem. [§9.3]

\thisproblemlabel

Defined in: probsoln package.

May be used within the ⟨*body*⟩ argument of \foreachproblem to access the current problem label. [§9.3]

\tick

Defined in: pgfplots package.

For use within one of the tick label options, this expands to the current tick element. [§12.5]

\ticket{⟨*content*⟩}

Defined in: ticket package.

Specifies the variable ticket content. [§10.2]

\ticketdefault

Defined in: ticket package.

The default ticket content. [§10.2]

\ticketDistance{⟨*x-dist*⟩}{⟨*y-dist*⟩}

Defined in: ticket package.

The horizontal and vertical distances between tickets in terms of \unitlength. [§10.2]

\ticketNumbers{⟨*num-cols*⟩}{⟨*num-rows*⟩}

Defined in: ticket package.

Specifies the number of tickets per sheet and their arrangement. [§10.2]

`\ticketSize{`⟨*width*⟩`}{`⟨*height*⟩`}`

Defined in: ticket package.

The width and height of the tickets in terms of `\unitlength`. [§10.2]

`\ticknum`

Defined in: pgfplots package.

For use within one of the tick label options, this expands to the current tick number (starting from 0). [§12.5]

`\begin{tikzpicture}[`⟨*options*⟩`]`

Defined in: tikz package.

Environment for drawing vector graphics. [§7.5]

`\time`

Defined in: TEX primitive.

The current time expressed as the number of minutes since midnight. [§7.4]

`\title{`⟨*text*⟩`}`

Defined in: Most classes that have the concept of a title page.

Specifies the document title. This command doesn't display any text so may be used in the preamble, but if it's not in the preamble it must be placed before `\maketitle`. Some classes, such as beamer, provide an optional argument for this command. [§5.2]

`\titledquestion{`⟨*title*⟩`}`
`[`⟨*points*⟩`]`

Defined in: exam class.

Like `\question` but assigns a title to the question. [§9.1]

`\titlegraphic{`⟨*graphic*⟩`}`

Defined in: beamer class.

Code to produce a title graphic used by `\maketitle`. [§8.0]

`\ToAddressWidth`

Defined in: envlab package.

A length register used to store the width of the recipient's address. [§3.6]

`\today`

Defined in: Most of the commonly-used classes.

Inserts into the output file the date when the LaTeX application created it from the source code. [§3.3]

`toFile(`⟨*filename*⟩`)`

Defined in: `arara` directive.

Create a file reference. [§1.2]

`\topic[`⟨*toc title*⟩`]{`⟨*title*⟩`}`

Defined in: minutes package.

Starts a new topic. [§6.3]

`\toprule[`⟨*wd*⟩`]`

Defined in: booktabs package.

Horizontal rule for the top of a tabular environment. [§2.6]

`\totalpoints`

Defined in: exsheets package.

Displays the total number of points, including bonus points. (Requires two LaTeX runs to ensure it's up to date.) The starred version omits the unit. [§9.2]

`\two@digits{`⟨*number*⟩`}`

Defined in: LaTeX Kernel.

Displays ⟨*number*⟩, ensuring that it has at least two digits. (If ⟨*number*⟩ is less than 10 a leading 0 is inserted.) [§7.4]

`\twocolumn`

Defined in: LaTeX Kernel.

Issues a page break and switches to two column mode. [§10.5]

U

unchanged(⟨*ref*⟩)

Defined in: arara directive.

Evaluates to true if the given file hasn't changed. The argument ⟨*ref*⟩ may either be a string "⟨*extension*⟩" which indicates the file extension or a file reference `toFile`("⟨*filename*⟩"). [§1.2]

`\unframedsolutions`

Defined in: exam class.

Switches off the default frame around the solutions. [§9.1]

`\unitlength`

Defined in: LaTeX Kernel.

A length used as the unit of measurement in the picture environment. [§10.1]

`unless` ⟨*condition*⟩

Defined in: arara directive.

Only run the application if ⟨*condition*⟩ is false. [§1.2]

`until` ⟨*condition*⟩

Defined in: arara directive.

Repeatedly run the application until ⟨*condition*⟩ is true. [§1.2]

`\url{`⟨*address*⟩`}`

Defined in: url package.

Typesets an URL in a typewriter font and allows you to use characters such as ~. [§6.2]

`\usecolortheme[`⟨*options*⟩`]{`⟨*name*⟩`}`

Defined in: beamer class.

Loads a beamer color theme. [§8.2]

`\usefonttheme[`⟨*options*⟩`]{`⟨*name*⟩`}`

Defined in: beamer class.

Loads a beamer font theme. [§8.2]

`\useinnertheme[`⟨*options*⟩`]{`⟨*name*⟩`}`

Defined in: beamer class.

Loads a beamer inner theme. [§8.2]

`\useoutertheme[`⟨*options*⟩`]{`⟨*name*⟩`}`

Defined in: beamer class.

Loads a beamer outer theme. [§8.2]

`\usepackage[`⟨*option-list*⟩`]{`⟨*package-list*⟩`}`

Defined in: LaTeX Kernel.

Loads the listed package(s). (See Volume 1 [92, §4.2].) [§1.1]

`\useproblem[`⟨*dataset*⟩`]{`⟨*label*⟩`}{`⟨arg$_1$⟩`}...{`⟨arg$_N$⟩`}`

Defined in: probsoln package.

Uses a previously defined problem. [§9.3]

`\usetheme[`⟨*options*⟩`]{`⟨*name*⟩`}`

Defined in: beamer class.

Loads a beamer presentation theme. [§8.2]

`\usetikzlibrary{`⟨*name*⟩`}`

Defined in: tikz package.

Load the tikz library called ⟨*name*⟩. (Preamble only.) [§7.5]

V

`\value{`⟨*counter*⟩`}`

Defined in: LaTeX Kernel.

References the value of the given counter where a number rather than a counter name is required. [§2.1]

`\vector(`⟨*h*⟩`,`⟨*v*⟩`){`⟨*length*⟩`}`

Defined in: LaTeX Kernel.

For use within the argument of `\put`, this draws a straight line with an arrowhead of the given length whose horizontal and vertical extent (gradient vector) is given by (⟨*h*⟩,⟨*v*⟩). [§10.1]

`\verb`⟨*char*⟩⟨*text*⟩⟨*char*⟩

Defined in: LaTeX Kernel.

Typesets ⟨*text*⟩ verbatim. The starred version replaces spaces with the visible space symbol. [§8.0]

`\begin{verbatim}`

Defined in: LaTeX Kernel.

Typesets the contents of the environment as is. (Can't be used in the argument of a command.) [§2.3]

`\verbatiminput{`⟨*filename*⟩`}`

Defined in: verbatim package.

Inputs the given file verbatim. [§9.4]

`\vfill`

Defined in: LaTeX Kernel.

Inserts a vertical space that will expand to fit the available height. [§3.6]

`\begin{Vote}`

Defined in: minutes package.

If multiple votes were made, the `\vote` commands are listed in this environment. [§6.3]

`\vote{`⟨*description*⟩`}{`⟨*yes*⟩`}{`⟨*no*⟩`}{`⟨*abstain*⟩`}[`⟨*decision*⟩`]`

Defined in: minutes package.

Indicates a vote took place. [§6.3]

`\vspace{`⟨*length*⟩`}`

Defined in: LaTeX Kernel.

Inserts a vertical gap of the given height. The unstarred version doesn't create a space if it occurs at the beginning or end of a page. The starred version always creates a space. [§9.1]

W

`while` ⟨*condition*⟩

Defined in: `arara` directive.

Repeatedly run the application while ⟨*condition*⟩ is true. [§1.2]

X

`\xappto`⟨*cs*⟩`{`⟨*code*⟩`}`

Defined in: etoolbox package.

Global version of `\eappto`. [§2.1]

`\xblackout{`⟨*text*⟩`}`

Defined in: censor package.

Like `\blackout`, but also blacks out the interword spaces. [§6.4]

`\XBox`

Defined in: wasysym package.

Produces a square with a cross in it ⊠. [§11.1]

\xdef⟨*cs*⟩⟨*arg-syntax*⟩ {⟨*definition*⟩}

Defined in: TEX primitive.

As \edef but the definition is global. [§2.1]

\xDTLassignfirstmatch{⟨*db-name*⟩}{⟨*col-label*⟩}{⟨*value*⟩} {⟨*assign-list*⟩}

Defined in: datatool package.

Finds the first row in the database ⟨*db-name*⟩ where entry in the column identified by the label ⟨*col-label*⟩ matches a *one level expansion* of ⟨*value*⟩ and applies the assignment list, which has the same format as for \DTLforeach and \DTLforeach*. See also \DTLassignfirstmatch. [§2.8]

\xifinlist{⟨*item*⟩}{⟨*list-cs*⟩} {⟨*true*⟩}{⟨*false*⟩}

Defined in: etoolbox package.

As \ifinlist but the test is performed on the expansion of ⟨*item*⟩. [§2.7]

\xifinlistcs{⟨*item*⟩}{⟨*list-csname*⟩}{⟨*true*⟩}{⟨*false*⟩}

Defined in: etoolbox package.

As \xifinlist but the control sequence name is supplied (without the backslash). [§2.7]

\xpreto⟨*cs*⟩{⟨*code*⟩}

Defined in: etoolbox package.

Global version of \epreto. [§2.1]

Y

\year

Defined in: TEX primitive.

The current year. [§7.2]

Index

Page numbers in **bold** indicate the entry definition in the summary.

A

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

B

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A **N**
B **O**
C **P**
D **Q**
E **R**
F **S**
G **T**
H **U**
I **V**
J **W**
K **X**
L **Y**
M **Z**

D

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols

Symbols

E

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A **N**
B **O**
C **P**
D **Q**
E **R**
F **S**
G **T**
H **U**
I **V**
J **W**
K **X**
L **Y**
M **Z**

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

F

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A **N**
B **O**
C **P**
D **Q**
E **R**
F **S**
G **T**
H **U**
I **V**
J **W**
K **X**
L **Y**
M **Z**

G

H

Symbols
A **N**
B **O**
C **P**
D **Q**
E **R**
F **S**
G **T**
H **U**
I **V**
J **W**
K **X**
L **Y**
M **Z**

I

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

N

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A **N**
B **O**
C **P**
D **Q**
E **R**
F **S**
G **T**
H **U**
I **V**
J **W**
K **X**
L **Y**
M **Z**

P

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A **N**
B **O**
C **P**
D **Q**
E **R**
F **S**
G **T**
H **U**
I **V**
J **W**
K **X**
L **Y**
M **Z**

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Q

R

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

S

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

T

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

Symbols
A **N**
B **O**
C **P**
D **Q**
E **R**
F **S**
G **T**
H **U**
I **V**
J **W**
K **X**
L **Y**
M **Z**

V

W

X

Symbols

Y

Z

Symbols
A N
B O
C P
D Q
E R
F S
G T
H U
I V
J W
K X
L Y
M Z

GNU Free Documentation License

Version 1.2, November 2002

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

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.

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

10. FUTURE REVISIONS OF THIS LICENSE

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

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

ADDENDUM: How to use this License for your documents

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

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

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

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

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

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

History

6th September 2021 (Version 1.3)

- Removed links to http://mirror.ctan.org/help/Catalogue (which no longer exists).

18th January 2016 (Version 1.2)

- Corrected wrong braces in argument to `\usepackage{babel}`.
- Fixed `.shtml` file extensions in electronic versions.

30th September 2015 (Version 1.1)

Corrected missing braces in `\ifdate` in calendar examples.

30th September 2015 (Version 1.0)

Initial version.

Back Cover Text

(See http://www.gnu.org/licenses/fdl-howto-opt.html#SEC2.)

If you choose to buy a copy of this book, Dickimaw Books asks for your support through buying the Dickimaw Books edition to help cover costs.