The LaTeXinfo Documentation Format

Version 1.7

Richard M. Stallman and Robert J. Chassell
The Free Software Foundation,
675 Massachusetts Ave., Cambridge MA,

Michael Clarkson
Centre for Earth and Space Science,
York University,
North York, Ontario, M3J 1P3

February 26, 1992

This is version 1.7 of the LaTeXinfo documentation, and is for Version 18 of GNU Emacs. This is the second edition of the LaTeXinfo documentation, and is also consistent with version 2 of Texinfo documentation '`texinfo.tex`'.

Contents

List of Tables

LaTeXinfo Copying Conditions

The programs currently being distributed that relate to LaTeXinfo include portions of GNU Emacs, plus other separate programs (including `latexinfo.sty`, `latexindex`, and `info`). These programs are *free*; this means that everyone is free to use them and free to redistribute them on a free basis. The Latexinfo-related programs are not in the public domain; they are copyrighted and there are restrictions on their distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of these programs that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of the programs that relate to LaTeXinfo, that you receive source code or else can get it if you want it, that you can change these programs or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of the LaTeXinfo related programs, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for the programs that relate to LaTeXinfo. If these programs are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the licenses for the programs currently being distributed that relate to LaTeXinfo are found in the General Public Licenses that accompany them.

Chapter 1

Overview of LaTeXinfo

LaTeXinfo[1] is a documentation system that uses a single source file to produce both on-line help (and other information) and a printed manual. This means that instead of writing two different documents, one providing on-line information and the other for a printed manual, you need write only one document. When the system is revised, you need revise only one document. You can print the manual with most laser printers, and you can read the on-line help, known as an *Info file*, with the Info documentation-reading programs. These documentation-reading programs are available for use under GNU Emacs, under X-windows, or for `termcap` based ordinary terminals.

Using LaTeXinfo, you can create a printed document with the normal features of a book, including chapters, sections, cross references, and indices. From the same LaTeXinfo source file, you can create a menu-driven, on-line Info file with nodes, menus, cross references, and indices. You can, if you wish, make the chapters and sections of the printed document correspond to the nodes of the on-line information, and use the same cross references and indices for both the Info file and the printed document.

To make a printed manual, process a LaTeXinfo source file with the LaTeX typesetting program. This creates a DVI file that you can typeset and print as a book. To create an Info file, you process a LaTeXinfo source file with Emacs's `latexinfo-format-buffer` command; this creates an Info file that you can install on-line.

Info works with almost every type of computer terminal; similarly, LaTeX works with many types of printer. This power makes LaTeXinfo a general purpose system, but brings with it a constraint, which is that a LaTeXinfo file may contain only the customary "typewriter" characters (letters, numbers, spaces, and punctuation marks) but no special graphics.

A LaTeXinfo file is a plain ASCII file containing text and *\-commands* (words preceded by an '\') that tell the typesetting and formatting programs what to do. You may edit a LaTeXinfo file with any text editor; but it is especially convenient to use GNU Emacs since that editor has a special mode, called LaTeXinfo mode, that provides various LaTeXinfo-related features.

[1] Note that the first syllable of "texinfo" is pronounced like "speck", not "hex". This odd pronunciation is derived from LaTeX, in which the '`X`' is actually the Greek letter "chi" rather than the English letter "ex" (the '`T`' and '`E`' are Greek letters also, but they happen to be pronounced the same way in Greek as in English).

(See section 15 [LaTeXinfo Mode], page 127.)

Before writing a LaTeXinfo source file, you should become familiar with the Info documentation reading program and learn about nodes, menus, cross references, and the rest. On Unix systems, these programs are called `info` for terminals, and `xinfo` for systems with X-Windows. (See Info file '`info`', node 'Top', for more information.)

LaTeXinfo creates both on-line help and a printed manual; moreover, it is freely redistributable.

1.1 Advantages of LaTeXinfo over TeXinfo

Documentation for GNU utilities and libraries is usually written in a format called *TeXinfo*. This document describes an enhancement of this format which can be used with LaTeX instead of TeX.

LaTeXinfo offers a number of advantages over TeXinfo:

1. The point size or layout style of a document can be changed easily, using the `documentstyle` (`article, report, book, twoside, ...`).

2. LaTeX has better error checking than TeX files, especially in begin/end environments. In addition, the LaTeX error messages are more informative. This makes it considerably easier to make extensions and enhancements (read *hacks*).

3. LaTeX delimits its arguments with braces, so it's easier to tell where a LaTeXinfo command starts, and where it ends. TeXinfo has to stand on its head to avoid using TeX's braces.

4. Any LaTeX commands not understood by the on-line manual generator ('`latexinfo.el`') are simply ignored. This means that you are free to add a considerable number of LaTeX commands to make you manual look pretty, as long as you don't care that there will be no action taken by the Info formatting program.

5. It is easy to add your own extensions to the on-line manual generator by making GNU Emacs handlers for your LaTeX extensions. This is the Emacs counterpart to the `documentstyle` options. LaTeXinfo looks in a specified directory for GNU Elisp code that corresponds to each style file. This makes it easy to modularize your style files.

6. LaTeX has many advantages over TeX, such as being able to easily incorporate the BibTeX bibliography formatting program.

1.2 Info files

A LaTeXinfo file can be transformed into a printed manual and an on-line Info file.

An on-line Info file is a file formatted so that the Info documentation reading program can operate on it. Info files are divided into pieces called *nodes*, each of which contains the discussion of one topic. Each node has a name, and contains both text for the user to read and pointers to other nodes, which are identified by their names. The Info program displays one node at a time, and provides commands with which the user can move to the other related nodes. See Info file '`info`', node 'Top', for more information about using Info.

Each node of an Info file may have any number of child nodes that describe subtopics of the node's topic. The names of these child nodes, if any, are listed in a *menu* within the parent node; this allows you to use certain Info commands to move to one of the child nodes. Generally, a LaTeXinfo file is organized like a book. If a node is at the logical level of a chapter, its child nodes are at the level of sections; likewise, the child nodes of sections are at the level of subsections.

All the children of any one parent are linked together in a bidirectional chain of 'Next' and 'Previous' pointers. This means that all the nodes that are at the level of sections within a chapter are linked together. Normally the order in this chain is the same as the order of the children in the parent's menu. Each child node records the parent node name, as its 'Up' pointer. The last child has no 'Next' pointer, and the first child has the parent both as its 'Previous' and as its 'Up' pointer.[2]

The book-like structuring of an Info file into nodes that correspond to chapters, sections, and the like is a matter of convention, not a requirement. The 'Up', 'Previous', and 'Next' pointers of a node can point to any other nodes, and a menu can contain any other nodes. Thus, the node structure can be any directed graph. But it is usually more comprehensible to follow a structure that corresponds to the structure of chapters and sections in a printed manual.

In addition to 'Next', 'Previous', and 'Up' pointers and menus, Info provides cross–references, that can be sprinkled throughout the text. This is usually the best way to represent links that do not fit a hierarchical structure. Usually, you will design a document so that its nodes match the structure of chapters and sections in the printed manual. But there are times when this is not right for the material being discussed. Therefore, LaTeXinfo uses separate commands to specify the node structure of the Info file and the section structure of the printed manual.

Generally, you enter an Info file through a node that by convention is called '`Top`'. This node normally contains just a brief summary of the file's purpose, and a large menu through which the rest of the file is reached. From this node, you can either traverse the file systematically by going from node to node, or you can go to a specific node listed in the main menu, or you can search the index menus and then go directly to the node that has the information you want.

[2] In some documents, the first child has no 'Previous' pointer. Occasionally, the last child has the node name of the next following higher level node as its 'Next' pointer.

1.3 Printed Manuals

A LaTeXinfo file can be formatted and typeset as a printed manual. To do this, you need to use LaTeX, a powerful, sophisticated typesetting program written by Leslie Lamport, based on the TeX typesetting system written by by Donald Knuth. A LaTeXinfo-based printed manual will be similar to any other book; it will have a title page, copyright page, table of contents, and preface, as well as chapters, numbered or unnumbered sections and subsections, page headers, cross references, footnotes, and indices.

You can use LaTeXinfo to write a book without ever having the intention of converting it into on-line information. You can use LaTeXinfo for writing a printed novel, and even to write a printed memo.

LaTeX is a general purpose typesetting program. LaTeXinfo provides a file called '`latexinfo.sty`' that contains information (definitions or *macros*) that LaTeX uses when it typesets a LaTeXinfo file. (The macros tell LaTeX how to convert the LaTeXinfo \-commands to LaTeX commands, which LaTeX can then process to create the typeset document.) LaTeX allows you to customize the design of your document by selecting different document styles and options. You can readily change the style in which the printed document is formatted; for example, you can change the sizes and fonts used, the amount of indentation for each paragraph, the degree to which words are hyphenated, and the like. By changing the specifications, you can make a book look dignified, old and serious, or light-hearted, young and cheery. See the LaTeX Manual for more details [Lam86].

LaTeX is freely distributable. It is written in a dialect of Pascal called WEB and can be compiled either in Pascal or (by using a conversion program that comes with the LaTeX distribution) in C. (See Info file '`emacs`', node 'TeX Mode', for information about LaTeX.)

LaTeX is very powerful and has a great many features. Because a LaTeXinfo file must be able to present information both on a character-only terminal in Info form and in a typeset book, the formatting commands that LaTeXinfo supports are necessarily limited. However, you are free to use any LaTeX extensions as long as you don't mind them being ignored by the Info formatting program. Or you can write your own extensions to the Info formatting program. See section 18 [Extending LaTeXinfo], page 153.

1.4 \-commands

In a LaTeXinfo file, the commands that tell LaTeX how to typeset the printed manual and tell `latexinfo-format-buffer` how to create an Info file are preceded by '`\`'; they are called *\-commands*. For example, `\node` is the command to indicate a node and `\chapter` is the command to indicate the start of a chapter.

Remark: Most of the \-commands, with a few exceptions such as `\LaTeX{}`, must be written entirely in lower case.

The LaTeXinfo \-commands are a limited subset of LaTeX commands. The limits make it possible for LaTeXinfo files to be understood both by LaTeX and by the code that converts them

into Info files. This is because you have to be able to display Info files on any terminal that displays alphabetic and numeric characters.

Because LaTeXinfo is an extension of LaTeX, it is assumed in this manual that you are familiar with LaTeX. There is a good reference manual available by the author [Lam86], and there are several beginner's introduction manuals alos available. You should read these first before trying to use LaTeXinfo.

Unlike LaTeX, all ASCII printing characters except '`\`', '`{`' and '`}`' can appear in body text in a LaTeXinfo file and stand for themselves. This means that the characters `# $ % ^ & _ |` all print as normal characters. This is for several reasons. Firstly, LaTeXinfo is designed for documenting computer programs, where these characters are used quite often. Secondly, the special uses in LaTeX of some of these characters, such as math mode, are not used in LaTeXinfo, so there is little point in making then special. And finally, because there is only one character in LaTeXinfo that starts a command (`\`), it is easier to implement the Info formating program, without making a complete implementation of LaTeX.

See section 5.8.1 [Using Ordinary LaTeX Commands], page 58, for how to make LaTeXinfo treat these characters as LaTeX does.

1.5 A Short Sample LaTeXinfo File

A LaTeXinfo file looks like the following, which is a complete but very short LaTeXinfo file. The `\comment` or `\c` command introduces comments that will not appear in either the Info file or the printed manual; they are for the person who reads the LaTeXinfo file.

The first part of the file, from '`\documentstyle`' through to '`\setfilename`', looks more intimidating than it is. Most of the material is standard boilerplate; when you write a manual, you just put in the name of your own manual in this section.

All the commands that tell LaTeX how to typeset the printed manual and tell `latexinfo-format-buffer` how to create an Info file are preceded by '`\`'; thus, `\node` indicates a node and `\chapter` indicates the start of a chapter.

```
\documentstyle[11pt,latexinfo]{book}

\begin{document}

\c Declare which indices you want to make use of.
\newindex{cp}
\c Declare the bibliography style you want for BibTeX.
\bibliographystyle{alpha}
\c No ugly overfull black boxes.
\finalout
\c \refill automatically.
\alwaysrefill

\c Anything before the \setfilename will not appear in the Info file.
\setfilename{plisp.info}

\c Start the stuff for the titlepage.

\title{The PLisp Manual}

\author{Fred Foobar,\\
Clarke Institute,\\
999 Queen Street,\\
Toronto, Ontario}

\date{\today}
\maketitle

\c The following commands start the copyright page for the printed manual.
\clearpage
\vspace*{0pt plus 1filll}
```

```
Copyright \copyright{} year copyright-owner

Permission is granted to copy and distribute modified versions of this
document under the conditions for verbatim copying, provided that the entire
resulting derived work is distributed under the terms of a permission
notice identical to this one.

\c End the Copyleft page and don't use headings on this page.
\pagestyle{empty}
\clearpage
\pagestyle{headings}

\c Use roman numerals for the page numbers and Insert the Table of Contents.
\pagenumbering{roman}
\tableofcontents

\c End the Table of Contents
\clearpage

\c Make a list of tables if you have any
\listoftables
\clearpage

\c The Top node contains the master menu for the Info file.
\c This appears only in the Info file, not the printed manual.
\node    Top,       First Chapter, (dir),    (dir)

\c A preface or overview to give the structure of the document.
\chapter*{Preface}

\clearpage
\c Start numbering from 1 with Arabic numbers
\pagenumbering{arabic}

\begin{menu}
* First Chapter::    The first chapter is the
                     only chapter in this sample.
\end{menu}

\node First Chapter, Concept Index, Top, Top
\chapter{First Chapter}
\cindex{Reference to First Chapter}
```

```
This is the contents of the first chapter.
Here is a numbered list.

\begin{enumerate}
\item
This is the first item.

\item
This is the second item.
\end{enumerate}

The \kbd{M-x latexinfo-format-buffer} command transforms a LaTeXinfo file
like this into an Info file; and \LaTeX\ typesets it for a printed manual.

\bibliography{plisp.bib}

\twocolumn
\node Concept Index, Top, First Chapter, Top
\unnumbered{Concept Index}

\printindex{cp}

\end{document}
```

Here is what the contents of the first chapter of the sample look like:

> This is the contents of the first chapter.
> Here is a numbered list.
>
> 1. This is the first item.
> 2. This is the second item.

The `M-x latexinfo-format-buffer` command in Emacs transforms a LaTeXinfo file like this into an Info file; and LaTeX typesets it for a printed manual.

1.6 The Structure of this Manual

This manual is structured in four parts:

LaTeX This introduces the LaTeX commands that are supported by LaTeXinfo. This includes topic such as chapter structuring, marking words and phrases, displayed material, making lists tables and descriptions, formatting paragraphs, citations and footnotes.

Info This introduces the concept of the `\node`, and the specific requirements of the Info formatting program. This includes nodes and menus, making cross references, creating indices, and creating and installing an info file.

Emacs This part show how to run LaTeX and Info to generate the printed and on–line versions of the manual. It also describes how Emacs can make your life easier when writing LaTeXinfo programs.

Appendices The appendices describe how to install LaTeXinfo, how to convert files from other formats to LaTeXinfo, and gives a summary of all of the commands.

Part I

LaTeX

Chapter 2

Beginning a LaTeXinfo File

2.1 General Syntactic Conventions

All ASCII printing characters except '`\`', '`{`' and '`}`' can appear in a LaTeXinfo file and stand for themselves. '`\`' is the escape character which introduces commands. '`{`' and '`}`' should be used only to surround arguments to certain commands. To put one of these special characters into the document, put an '`\`' character in front of it, like this: '`\back`', '`\{`', and '`\}`'.

It is customary in LaTeX to use doubled single-quote characters to begin and end quotations: ``` `` ``` and `''`. This convention should be followed in LaTeXinfo files. LaTeX converts doubled single-quote characters to left- and right-hand doubled quotation marks, "like this," and Info converts doubled single-quote characters to ASCII double-quotes: ``` `` ``` and `''` to '`"`'. See section 4.3.6 [Inserting Characters Verbatim], page 46 for how to protect sections of documentation from these global substitutions.

Use three hyphens in a row, '`---`', for a dash—like this. In LaTeX, a single or even a double hyphen produces a printed dash that is shorter than you want. Info reduces three hyphens to two for display on the screen.

LaTeX ignores the line-breaks in the input text, except for blank lines, which separate paragraphs. Info generally preserves the line breaks that are present in the input file. Therefore, break the lines in the LaTeXinfo file the way you want them to appear in the output Info file, and let LaTeX take care of itself. Since Info does not normally refill paragraphs when it processes them, a line with commands in it will sometimes look bad after Info has run on it. To cause Info to refill the paragraph after finishing with the other processing, you need to put the command `\refill` at the end of the paragraph. (See section 7.4 [Refilling Paragraphs], page 68.)

If you mark off a region of the LaTeXinfo file with the `\begin{iftex}` and `\end{iftex}` commands, that region will appear only in the printed copy; in that region, you can use commands borrowed from LaTeX that you cannot use in Info. Likewise, if you mark off a region with the `\begin{ifinfo}` and `\end{ifinfo}` commands, that region will appear only in the Info file; in that region, you can use Info commands that you cannot use in LaTeX. (See section 5.8 [Conditionals], page 57.)

Caution: Do not use tabs in examples in a LaTeXinfo file! LaTeX treats them like

single spaces.[1]

2.1.1 Comments

You can write comments in a LaTeXinfo file that will not appear in either the Info file or the printed manual by using the `\comment` command (which may be abbreviated to `\c`). Such comments are for the person who reads the LaTeXinfo file. All the text on a line that follows either `\comment` or `\c` is a comment; the rest of the line does not appear in either the Info file or the printed manual. (The `\comment` or `\c` does not have to be at the beginning of the line; only the text on the line that follows after the `\comment` or `\c` command does not appear.)

You can write long stretches of text that will not appear in either the Info file or the printed manual by using the `\begin{ignore}` and `\end{ignore}` commands. Write each of these commands on a line of its own, starting each command at the beginning of the line. Text between these two commands does not appear in the processed output. You can use `\begin{ignore}` and `\end{ignore}` for writing comments or for holding text you may wish to use in another version of your document. Often, `\begin{ignore}` and `\end{ignore}` is used to enclose a part of the copying permissions that applies to the LaTeXinfo source file of a document, but not to the Info or printed version of the document.

2.2 What a LaTeXinfo File Must Have

In order to be made into a printed manual, a LaTeXinfo file **must** begin with lines that looks like

```
\documentstyle[12pt,latexinfo]{book}
\pagestyle{headings}

\begin{document}

\setfilename{latexinfo.info}
```

The '`\documentstyle[12pt,latexinfo]{book}`' line tells LaTeX to use the '`latexinfo.sty`' style and '`book.sty`' documentstyle files. The `\pagestyle{headings}` command is the LaTeX command to put the chapter and section headings and page numbers at the top of each page. The `\begin{document}` command starts the document, and makes the characters `# $ % ^ & _ |` all begin to print as normal characters. (These characters retain their normal LaTeX meanings in the preamble between the `\documentstyle` and `\begin{document}` commands.) This line must be followed (sooner or later) by the '`\setfilename{`*info-file-name*`}`'. It is needed to provide a name for the Info file to output to. The `\setfilename` command *must* occur at the beginning of a line. The

[1] To avoid putting tabs into your file, you can set the `indent-tabs-mode` variable in Emacs to `nil` so that Emacs inserts multiple spaces when you press the `TAB` key. Also, you can run `untabify` to convert tabs in a region to multiple spaces.

```
\end{document}
```

line at the end of the file on a line of its own tells LaTeX that the file is ended and to stop typesetting.

Usually, you won't use quite such a spare format, but will include mode setting and index declarations at the beginning of a LaTeXinfo file, like this:

```
\documentstyle[12pt,latexinfo]{book}
\pagestyle{headings}

\begin{document}

\newindex{cp}
\bibliographystyle{alpha}

\finalout
\alwaysrefill
\setfilename{latexinfo.info}
```

Furthermore, you will usually provide a LaTeXinfo file with a title page, master menu, and the like. But the minimum, which can be useful for short documents, is just the three lines at the beginning and the one line at the end.

2.3 Six Parts of a LaTeXinfo File

Various pieces of information have to be provided to LaTeXinfo at the beginning of a LaTeXinfo file, such as the name of the file, the title of the document and the like. If you want to get elaborate, the beginning of a LaTeXinfo file has six parts:

1. The preamble, which includes the command to tell LaTeX what style files to use when processing the file. This starts with the `\documentstyle`, and is terminated by the `\begin{document}` command.

2. The header, which is terminated by the `\setfilename` command that contains the LaTeXinfo options needed to tailor your output to your needs.

3. The title page and the copyright page, which usually are set without any page numbers (`\pagestyle{empty}`). This is terminated by the `\maketitle` command.

4. Then the table of contents, list of figures and tables, and possibly a preface, which are usually set with roman page numbers (`\pagestyle{headings}` and `\pagenumbering{roman}`).

5. The '`Top`' node that contains an extensive menu for the whole Info file. This is written with the `\node` command, with a nodename of `Top`. The contents of this node should only appear in the Info file.

6. The beginning of the text, which is set with `\pagenumbering{arabic}`), and a chapter or section command.

For a a short sample latexinfo file, see the file '`lnfo-sample.tex`' which is supplied with the LaTeXinfo distribution.

2.4 The LaTeXinfo File Header

LaTeXinfo files start with at least three lines that provide Info and LaTeX with necessary information.

```
\documentstyle[12pt,latexinfo]{book}
\begin{document}
\setfilename{foo.info}
```

2.4.1 The Documentstyle

Every LaTeXinfo file that is to be the top-level input to LaTeX must begin with a line that looks like this:

```
\documentstyle[12pt,latexinfo]{book}
```

When the file is processed by LaTeX, it loads the macros listed as options to the `documentstyle` command. The option `latexinfo` is needed for processing a LaTeXinfo file, and LaTeX will then input the file '`latexinfo.sty`'; see section 16.5 [Preparing for LaTeX], page 143.

Unlike TeXinfo, you can also include other options that may also include style files. These LaTeXinfo style files may have an Emacs counterpart, so that you can extend LaTeXinfo by writing your own styles. See section 18 [Extending LaTeXinfo], page 153, for more information on writing your own styles. Also look in the inputs directory of your TeX distribution for other LaTeX styles that are provided with TeX.

Remark: The region of the file between the `\documentstyle` and the `\begin{document}` commands is know in LaTeX as the *preamble*. Only certain LaTeX commands are allowed there, and you'll need to consult the LaTeX manual for the list of allowed commands. It is best to put your commands that modify your LaTeX commands in the region between the `\begin{document}` and the `\setfilename`; they will be ignored by Info, and LaTeX will not object.

2.4.2 `\setfilename`

It is important to note that the `\setfilename` command is required for Info. In order to be made into an Info file, a LaTeXinfo file must contain a line that looks like this:

`\setfilename{`*info-file-name*`}`

Write the `\setfilename` command at the beginning of a line followed by the Info file name.

The `\setfilename` line specifies the name of the Info file to be generated. Specify the name with an '`.info`' extension, to produce an Info file name such as '`latexinfo.info`'.

Any text that appears before the `\setfilename` command is not included in the Info file. So if you want to include the `title` and `author` material, place the `\setfilename` command before them; if not, after them.

This region, between the `\begin{document}` command and the `\setfilename` command is known as the header, and should contain any of the commands that alter the overall style of your document.

2.4.3 New Indexes

In order to generate any of the indices, you must declare them with the `\newindex` command, before it is first used by one of the index commands. This is usually done after the `\begin{document}` but before the `\setfilename`.

`newindex` takes one argument, which is the two letter index type. For example, to declare a concept and function index, you would use

```
\documentstyle[12pt,latexinfo]{book}

\begin{document}
\newindex{cp}
\newindex{fn}
\setfilename{plisp.info}
```

See section 13.2.1 [Declaring indices], page 115, for the declaring indices and the definitions of the index types.

2.4.4 Customizing Your Layout

You may, if you wish, create your own, customized headings and footings. The `\markboth` and `\markright` commands are both supported in LaTeXinfo. These should occur on a line by themselves. See [Lam86, § 5.1], for a detailed discussion of this process.

The `\oddfoot` and `\evenfoot` commands specifiy the odd and even page footings respectively. These should occur on a line by themselves.

At the beginning of a manual or book, pages are not numbered—for example, the title and copyright pages of a book are not numbered. To accomplish this, use the command

```
\pagestyle{empty}
```

as shown in the sample file, '`lnfo-sample.tex`'.

By convention, table of contents pages are numbered with roman numerals and not in sequence with the rest of the document. To accomplish this, use the commands

```
\pagestyle{headings}
\pagenumbering{roman}
```

as shown in the sample file, '`lnfo-sample.tex`'.

Since an Info file does not have pages, the `\markboth`, `\markright`, `pagestyle` and `pagenumbering` commands have no effect on it. The lines containing these commands will be deleted from the Info file.

The `\footnotestyle` command to specify an Info file's footnote style. See section 8.1 [Footnotes], page 71 for how to use this command.

2.4.4.1 Paragraph Indenting

The Info formatting commands may insert spaces at the beginning of the first line of each paragraph, thereby indenting that paragraph. The `\paragraphindent` command specifies the indentation. Write `\paragraphindent` at the beginning of a line followed by either '`asis`' or a number in braces. The template is:

`\paragraphindent{`*indent*`}`

The Info formatting commands indent according to the value of *indent*:

- If the value of *indent* is '`asis`', the Info formatting commands do not change the existing indentation.
- If the value of *indent* is 0, the Info formatting commands delete existing indentation.
- If the value of *indent* is greater than 0, the Info formatting commands indent the paragraph by that number of spaces.

The default value of *indent* is '`asis`'.

Write the `\paragraphindent` command before the `setfilename` command at the beginning of a LaTeXinfo file.

The `latexinfo-format-buffer` and `latexinfo-format-region` commands do *not* automatically indent paragraphs. These commands only indent paragraphs that are ended by an `\refill` command. (See section 7.5 [Always Refilling Paragraphs], page 69, for how to avoid this.) See section 7.4 [Refilling Paragraphs], page 68, for more information about `\refill`.

2.5 The Title and Copyright Pages

2.5.1 Titlepage

The first printed material after the `\begin{document}` will make up the titlepage. The LaTeX commands `\title`, `\author` and `\date` are used the same way as in any LaTeX report or book. The title page is terminated by `\maketitle`. Following the material for the title page should be the copyright page.

```
\title{The PLisp Manual}

\author{Fred Foobar,\\
Clarke Institute}

\date{\today}
\maketitle
```

The `\title` command produces a line in which the title is set centered on the page in a larger than normal font. You can have many lines in the title by using `\\` to force a newline.

The `\author` command sets the names of the author or authors in a middle-sized font, centered on the page.

The `\date` command sets the date in a middle-sized font, centered on the page. You can put the date in yourself, or use the `\today` command, which will put in the date that the document is processed on.

The `\maketitle` command sets the author, title and date, and in the `book` documentstyle, emitts a new page. The should be no other printing text between the `documentstyle` command and the `maketitle` command.

In a `book` style, text is printed on both sides of the paper, chapters start on right-hand pages, and right-hand pages have odd numbers. But in a `report` style, text is printed only on one side of the paper unless the `twoside` LaTeX option is provided to the `documentstyle` command.

2.5.2 The Copyright Page and Printed Permissions

This part of the beginning of a LaTeXinfo file contains the text of the copying permissions that will appear in the manual. This is usually followed by the `\tableofcontents` command. If you put title and copyright pages before the `\setfilename` command, then this material will only appear only in the printed manual, not in the Info file.

By international treaty, the copyright notice for a book should be either on the title page or on the back of the title page. The copyright notice should include the year followed by the name of the organization or person who owns the copyright.

The following commands start the copyright page for the printed manual.

```
\maketitle
\clearpage
\vspace*{0pt plus 1filll}
Copyright \copyright{} year copyright-owner

Permission is granted to copy and distribute modified versions of this
document under the conditions for verbatim copying, provided that the entire
resulting derived work is distributed under the terms of a permission
notice identical to this one.
```

```
\pagestyle{empty}
\clearpage
```

When the copyright notice is on the back of the title page, the page is not numbered. Therefore, this is usually done while a `\pagestyle{empty}` is in effect. See the LaTeX Manual for more details on the `pagestyle` command [Lam86].

To cause a page break, the `\clearpage` command is used. In the sample, the `\clearpage` that ends the titlepage is followed by the somewhat mysterious line that reads: '`\vspace*{0pt plus 1filll}`'. This is a line that uses LaTeX commands to push the copyright notice and the other text on the copyright page towards the bottom of the page. The `\vspace*` command means to put in white space. The '`0pt plus 1filll`' means to put in zero points of mandatory white space, and as much optional white space as needed. Note the use of three '`l`'s in the word '`filll`'; this is the correct use in LaTeX.

The `\copyright` command generates a '`c`' inside a circle. The copyright notice itself has the following legally defined sequence:

`Copyright` © *year copyright-owner*

It is customary to put information on how to get a manual after the copyright notice (the address of the Free Software Foundation, for example) and the permissions.

When you write a manual about a computer program, you should write the version of the program to which the manual applies on the title page. If the manual changes more frequently than the program or is independent of it, you should also include an edition number[2] for the manual. This helps readers keep track of which manual is for which version of the program.

See section 2.8.1 [Sample Permissions], page 26, for recommended permission text.

2.6 Generating a Table of Contents

The commands `\chapter`, `\section`, etc., supply the information to make up a table of contents, but they do not cause an actual table to be generated. To do this, you must use the `\tableofcontents` command.

The table of contents command outputs (into a printed manual) a complete table of contents, based on the `\chapter`, `\section` and other sectioning commands. This command should be used on a line by itself. This command automatically generates a Table of Contents heading at the top of the page. Tables of contents should be generated at the beginning of the manual, usually just after the `\maketitle` command or copyright pages.

You can also use the `\listoftables` command to make a listing of all of the tables in the document. See section 6.5 [Figures and Tables], page 63, for how to define tables.

[2] We have found that it is helpful to refer to versions of manuals as 'editions' and versions of programs as 'versions'; otherwise, we find we are liable to confuse each other in conversation by referring to both the documentation and the software with the same words.

```
\pagestyle{empty}
\clearpage
\pagestyle{headings}
\pagenumbering{roman}
\tableofcontents
\clearpage

\c Make a list of tables if you have any
\listoftables
\clearpage
```

Since an Info file uses menus instead of tables of contents, the Info formatting commands ignore the `\tableofcontents` and `\listoftables` commands.

2.7 The Top Node and Master Menu

The 'Top' node is the node at which you enter the file when browing the Info file with one of the Info browsing programs.

A 'Top' node should contain a brief description of the file and an extensive, master menu for the whole Info file. The contents of anything other than the master menu should appear only in the Info file; none of it should appear in printed output, so enclose it between `\begin{ifinfo}` and `\end{ifinfo}` commands. LaTeX does not print either an `\node` line or a menu; they appear only in Info, so you do not have to enclose these parts between `\begin{ifinfo}` and `\end{ifinfo}`. See section Conditionals in Conditionally Visible Text.)

For example, the beginning of the Top node of a manual might like this:

```
...
\tableofcontents
\clearpage

\begin{ifinfo}
\node Top, Copying, (dir), (dir)

LaTeX info is a documentation system...

This is edition...
\end{ifinfo}

\begin{menu}
* Copying::                  LaTeX info is freely redistributable.
* Overview::                 What is LaTeXinfo?...
\end{menu}
```

In a 'Top' node, the 'Previous', and 'Up' nodes usually refer to the top level directory of the whole Info system, which is called '`(dir)`'. See section 14.2 [Installing an Info File], page 121, for more information about the `dir` Info file in the '`info`' directory.

2.7.1 Parts of a Master Menu

A *master menu* is a detailed main menu listing all the nodes in a file. A master menu is enclosed in `\begin{menu}` and `\end{menu}` commands and does not appear in the printed document. Generally, a master menu is divided into parts.

- The first part contains the major nodes in the LaTeXinfo file: the nodes for the chapters, chapter-like sections, and the appendices.

- The second part contains nodes for the indices.

- The third and subsequent parts contain a listing of the other, lower level nodes, often ordered by chapter. This way, rather than go through an intermediary menu, an inquirer can go directly to a particular node when searching for specific information. These menu items are not required; add them if you think they are a convenience.

Each section in the menu can be introduced by a descriptive line. So long as the line does not begin with an asterisk, it will not be treated as a menu item. (See section 11.3 [Menu Environment], page 96, for more information.)

For example, the master menu for a manual might look like the following:

```
\begin{menu}
* Copying::             LaTeX info is freely
                          redistributable.
* Overview::            What is LaTeXinfo?
* LaTeXinfo Mode::        Special features in GNU Emacs.
...
...
* Command and Variable Index::
                        An item for each  \-command.
* Concept Index::       An item for each concept.

 --- The Detailed Node Listing ---

Overview of LaTeXinfo

* Info Files::          What is an Info file?
* Printed Manuals::     Characteristics of
                          a printed manual.
...
...

Using LaTeX info Mode

* Info on a Region::    Formatting part of a file
                          for Info.
...
...
\end{menu}
```

2.8 Software Copying Conditions

If the LaTeXinfo file has a section containing the distribution information and a warranty disclaimer for the software that is being documented, this section usually follows the 'Top' node. The General Public License is very important to Project GNU software. It ensures that you and others will continue to have a right to use and share the software.

The copying and distribution information and the disclaimer are usually followed by a preface, or else by the first chapter of the manual.

Although a preface is not a required part of a LaTeXinfo file, it is very helpful. Ideally, it should state clearly and concisely what the file is about and who would be interested in reading it. In general, the preface would follow the licensing and distribution information, although sometimes people put it earlier in the document. Usually, a preface is put in an `\chapter*`

type of section. (See section 3.3 [Chapter], page 33.)

2.8.1 Sample Permissions

LaTeXinfo files should contain sections that tell the readers that they have the right to copy and distribute the Info file, the printed manual, and any accompanying software. Here are samples containing the standard text of the Free Software Foundation copying permission notice for an Info file and printed manual.

See Info file 'emacs', node 'Distrib', for an example of the text that could be used in the software Distribution, General Public License, and NO WARRANTY sections of a document.

2.8.2 Titlepage Copying Permissions

In the copyright section of the LaTeXinfo file, the standard Free Software Foundation copying permission notice follows the copyright notice and publishing information. The standard phrasing is:

```
Permission is granted to make and distribute verbatim
copies of this manual provided the copyright notice and
this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified
versions of this manual under the conditions for
verbatim copying, provided also that the sections
entitled ``Distribution'' and ``General Public License''
are included exactly as in the original, and provided
that the entire resulting derived work is distributed
under the terms of a permission notice identical to this
one.

Permission is granted to copy and distribute
translations of this manual into another language, under
the above conditions for modified versions, except that
the sections entitled ``Distribution'' and ``General
Public License'' may be included in a translation
approved by the author instead of in the original
English.
```

2.9 Ending a LaTeXinfo File

The end of a LaTeXinfo file should include the commands that create the bibliography, and the indices. It must include the `\end{document}` command that marks the last line that LaTeX processes. For example, a LaTeXinfo file might be ended as follows:

For example,

```
\bibliography{latexinfo}

\twocolumn
\node Concept Index,     , Previous Node, Top
\unnumbered{Concept Index}
\cindex{Concept Index}

\printindex{cp}

\end{document}
```

The `\end{document}` command should be on a line by itself and every LaTeXinfo file must end with such a line. This command terminates LaTeX processing and forces out unfinished pages.

2.9.1 Making a Bibliography

You may also choose to include a bibliography of citations in the document, using the `\cite` command . Citations are prepared using the program BibTeX, which formats the citations for use with LaTeX. See the LaTeX Manual for more details in BibTeX [Lam86, Appendix B].

Before you use the `\cite` command, you must declare the bibliography style that you are going to use. This is usually done at the beginning of the document, for example

```
\begin{document}
\bibliographystyle{alpha}
```

At the end of the document comes the bibliography itself. The `\bibliography` takes as an argument a comma separated list of filenames that contain the bibliography entries.

```
\bibliography{latexinfo}
```

With these two sections in your document, you can use the `\cite` command to refer to the bibliography. For example

```
a citation of the GNU Emacs Manual \cite{GNUEmacsManual}\dots
```

would produce

```
a citation of the GNU Emacs Manual [Sta86]...
```

and would cause an entry to be put in the Bibliography section something like

Sta86 Richard Stallman. *The GNU Emacs Manual*, The Free Software Foundation, 675 Massachusetts Ave., Cambridge MA, 02139, 1986.

See section 8.2 [Citations], page 72, for how to use citations in the document.

2.9.2 Index Menus and Printing an Index

Using LaTeXinfo, you can generate printed indices and Info file menus without having to sort and collate entries manually. LaTeXinfo will do this for you automatically. Each index covers a certain kind of entry (functions, or variables, or concepts, etc.) and lists all of those entries in alphabetical order, together with information on how to find the discussion of each entry. In a printed manual, this information consists of page numbers. In an Info file, it consists of a menu item leading to the first node where the entry is defined.

To print an index means to include it as part of a manual or Info file. This does not happen automatically just because you use `\cindex` or other index-entry generating commands in the LaTeXinfo file; those just cause the raw data for the index to be accumulated. To generate an index, you must include the `\printindex` command at the place in the document where you want the index to appear, and declare the index at the beginning of the document with the `\newindex` command. Also, as part of the process of creating a printed manual, you must run a program called `latexindex` (see section 16 [Printing Hardcopy], page 139) to sort the raw data to produce a sorted index file. The sorted index file is what will actually be used to print the index.

Like typesetting, the construction of an index is a highly skilled, professional art, the subtleties of which are not appreciated until you have to do it yourself. LaTeXinfo offers six different types of predefined index: the concept index, the function index, etc. (See section 13 [Creating Indices], page 113.) Each index type has a two-letter name. You may merge indices, or put them into separate sections (See section 13.3 [Combining Indices], page 116.).

The `\printindex` command does not generate a chapter heading for the index. Consequently, you should precede the `\printindex` command with a suitable section or chapter command (usually `\unnumbered`) to supply the chapter heading and put the index into the table of contents. Precede the `\unnumbered` command with an `\node` line. Also, if you want the index to be set in two-column mode, then you should precede the index with the LaTeX `\twocolumn` command. You can switch back to one-column mode with the LaTeX `\onecolumn` command. For example,

```
\twocolumn
\node Command Index, Concept Index, The Last Section, Top
\unnumbederd{Command Index}
\cindex{Command Index}

\printindex{fn}

\onecolumn
\node Concept Index,Top,   Command Index, Top
\unnumbered{Concept Index}
\cindex{Concept Index}

\printindex{cp}
```

```
\end{document}
```

(Readers often prefer that the concept index come last in a book, since that makes it easiest to find.)

2.9.3 `\end{document}` File Ending

An `\end{document}` command terminates LaTeX or Info formatting. None of the formatting commands see any of the file following `\end{document}`. The `\end{document}` command should be on a line by itself.

Optionally, you may follow an `\end{document}` line with a local variables list. See section Compile-Command in Using Local Variables and the Compile Command, for more information.

Chapter 3

Chapter Structuring

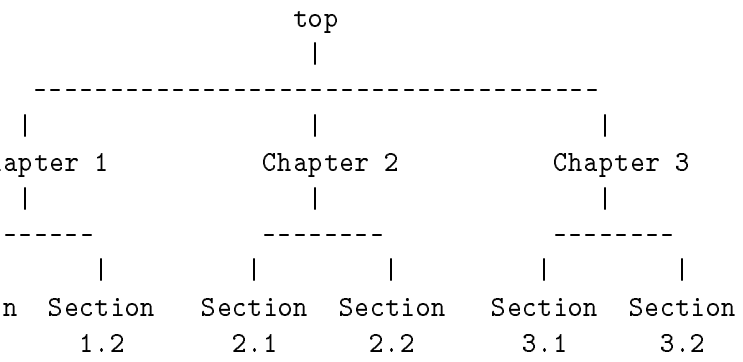
The chapter structuring commands divide a document into a hierarchy of chapters, sections, subsections, and subsubsections. These commands generate large headings; they also provide information for the table of contents of a printed manual.

The chapter structuring commands do not create an Info node structure, so normally you should put an `\node` command immediately before each chapter structuring command (see section Nodes and Menus in Nodes and Menus). The only time you are likely to use the chapter structuring commands without using the node structuring commands is if you are writing a document that contains no cross references and will never be transformed into Info format.

3.1 Tree Structure of Sections

A LaTeXinfo file is usually structured like a book with chapters, sections, subsections, and the like. This structure can be visualized as a tree (or rather as an upside-down tree) with the root at the top and the levels corresponding to chapters, sections, subsection, and subsubsections. In Info format, 'Next' and 'Previous' pointers of a node usually lead to other nodes at the same level; an 'Up' pointer usually leads to a node at the level above; and a 'Menu' leads to nodes at a level below. Cross references can point to nodes at any level. See section 12 [Cross References], page 101.

Here is a diagram that shows a LaTeXinfo file with three chapters, each of which has two sections.



In a LaTeXinfo file that has this structure, the beginning of Chapter 2 might look like this:

```
\node    Chapter 2,  Chapter 3, Chapter 1, top
\chapter{Chapter 2}
```

To get to Sections 2.1 and 2.2, you need a menu inside of Chapter 2 that says:

```
\begin{menu}
* Sect. 2.1::    Description of this section.
* Sect. 2.2::
\end{menu}
```

This menu is located inside Chapter 2, after the beginning of the chapter, just before Section 2.1.

Usually, a `\node` command and a chapter structuring command are used in sequence, along with indexing commands. For example, the node for the chapter on Ending a LaTeXinfo File looks like this:

```
\node Ending a LaTeXinfo File, Structuring, Beginning a LaTeXinfo File, Top
\chapter{Ending a LaTeXinfo File}
\cindex{Ending a LaTeXinfo File}
\cindex{LaTeXinfo file ending}
\cindex{File ending}
```

The `\node` command is the only one in LaTeXinfo where the arguments are not delineated by braces. The arguments are separated by commas, and are terminated at the end of the line. This is because the Info format itself requires the `node` arguments to be like this. Note that it also means that you cannot use a comma within any of the four arguments to the node command.

The chapter structuring commands are described in the sections that follow; the `\node` and `\begin{menu}` commands are described in a following chapter (see section 11 [Nodes and Menus], page 93).

3.2 Types of Structuring Command

There are four chapter-structuring commands for chapters, sections, subsections and subsubsections. The optional heading argument of LaTeX to these commands is not supported. You should avoid the use of any LaTeX commands in the headings: any such commands should be preceded by a `\protect`. See the LaTeX Manual for more details [Lam86].

3.3 Chapter

`\chapter` identifies a chapter in the document. It is followed by a single argument:

```
\chapter{Node and Chapter Structuring}
```

In LaTeX, it creates a chapter in the document, specifying the chapter title. The chapter will be numbered automatically in the printed manual. In the Info file, `\chapter` causes its argument to appear on a line by itself, with a line of asterisks inserted underneath. For example:

```
This is a Chapter
*****************
```

To start a chapter without it being numbered, use the `\unnumbered` command. To start a chapter without it being numbered or appearing in the table of contents, use the `\chapter*` command.

In the printed manual, the chapters will begin on a new page. If you want the chapters to appear on the odd–sided pages, use the `book` documentstyle.

3.4 Appendix

`\appendix` is the same as the LaTeX command of the same name. In a printed manual, all chapters that follow this command are numbered differently in the table of contents: they are given a letter instead of a number, and the letters restart from `A`.

3.5 Section

`\section` is like `\chapter` except that in LaTeX it makes a section rather than a chapter. (See section 3.3 [Chapter], page 33.) Sections go within chapters. In the Info file, `\chapter` and `\section` differ only in that `\section` underlines with '`=`'. For example,

```
This is a section
=================
```

To start a section without it being numbered, use the `\unnumberedsec` command. To start a section without it being numbered or appearing in the table of contents, use the `\section*` command.

3.6 Subsection

Subsections are to sections as sections are to chapters. (See section 3.5 [Section], page 33.) They are underlined with '-'. For example,

```
This is a subsection
--------------------
```

To start a subsection without it being numbered, use the `\unnumberedsubsec` command. To start a subsection without it being numbered or appearing in the table of contents, use the `\subsection*` command.

3.7 Subsubsection

Subsubsections are to subsections as subsections are to sections. (See section 3.6 [Subsection], page 34.) They are underlined with periods. For example,

```
This is a subsubsection
.......................
```

To start a subsubsection without it being numbered, use the `\unnumberedsubsubsec` command. To start a subsubsection without it being numbered or appearing in the table of contents, use the `\subsubsection*` command.

Chapter 4

Marking Words and Phrases

In LaTeXinfo, you can mark words and phrases in a variety of ways. These ways specify, for example, whether a word or phrase is a defining occurrence, a metasyntactic variable, or a symbol used in a program. Also, you can use fonts to emphasize text.

In addition, there are special commands for inserting single characters that have special meaning in LaTeXinfo, such as braces, and for inserting symbols with special handling, such as dots and bullets. Finally, there are ways to emphasize words.

4.1 Indicating Definitions, Commands, etc.

LaTeXinfo has commands for indicating just what kind of object a piece of text refers to. Metasyntactic variables, for example, are marked by `\var` and code by `\code`. LaTeXinfo uses this information to determine how to highlight the text. Since the pieces of text are labelled by commands that tell what kind of object they are, it is easy to change the way LaTeXinfo formats such text.

For example, code is usually illustrated in a typewriter font, but it would be easy to change the way LaTeXinfo highlights code to use another font. This change would not effect how keystroke examples are highlighted. If straight typesetting commands were used in the body of the file and you wanted to make a change, you would have to check every single occurrence to make sure that you were changing code and not something else that should not be changed.

The highlighting commands can be used to generate useful information from the file, such as lists of functions or file names. It is possible, for example, to write a LaTeXinfo style to insert an index entry after every paragraph that contains words or phrases marked by a specified command. You could do this to construct an index of functions automatically; see section 18.1.1 [The fvpindex Style], page 153 for an example.

The font changing commands commands serve a variety of purposes:

\code{*sample-code*} Indicate text that is a literal example of a piece of a program.

\kbd{*keyboard-characters*} Indicate keyboard input.

\key{*key-name*} Use for the conventional name for a key on a keyboard.

\samp{*text*} Indicate text that is a literal example of a sequence of characters.

\var{*metasyntactic-variable*} Indicate a metasyntactic variable.

\file{*file-name*} Indicate the name of a file.

\dfn{*term*} Use for the introductory or defining use of a term.

\ctrl{*ctrl-char*} Use for an ASCII control character.

4.1.1 `\code{`*sample-code*`}`

Use the `\code` command to indicate text that is a piece of a program and which consists of entire syntactic tokens. Enclose the text in braces.

Thus, you should use `\code` for an expression in a program, for the name of a variable or function used in a program, or for a keyword. Also, you should use `\code` for the name of a program, such as `diff`, that is a name used in the machine. (You should write the name of a program in the ordinary text font if you regard it as a new English word, such as 'Emacs' or 'Bison'.)

Use `\code` for the `TEXINPUTS` environment variable and other such variables.

Do not use the `\code` command for a string of characters shorter than a syntactic token. In particular, you should not use the `\code` command when writing about the characters used in a token; do not, for example, use `\code` when you are explaining what letters or printable symbols can be used in the names of functions. (Use `\samp`.) Also, you should not use `\code` to mark text that is considered input to programs unless the input is written in a language that is like a programming language. For example, you should not use `\code` for the single character commands of GNU Emacs (use `\kbd` instead) although you may use `\code` for the names of the Emacs Lisp functions that the keyboard commands invoke.

In the printed manual, `\code` causes LaTeX to typeset the argument in a typewriter face. In the Info file, it causes the Info formatting commands to use '...' quotation. For example:

```
Use  \code{diff} to compare two files.
```

produces this in the printed manual:

> Use `diff` to compare two files.

and this in Info file:

```
Use `diff' to compare two files.
```

4.1.2 \kbd{*keyboard-characters*}

Use the `\kbd` command for characters of input to be typed by users. For example, to refer to the characters `M-a`, write

```
\kbd{M-a}
```

and to refer to the characters `M-x shell`, write

```
\kbd{M-x shell}
```

The `\kbd` command has the same effect as `\code` in Info, but may produce a different font in a printed manual.

You can embed another `\`-command inside the braces of an `\kbd` command. Here, for example, is the way to describe a command that would be described more verbosely as "press an '`r`' and then press the `RET` key":

```
\kbd{r  \key{RET}}
```

This produces: `r RET`

You also use the `\kbd` command if you are spelling out the letters you type; for example:

```
To give the  \code{logout} command,
type the characters  \kbd{l o g o u t  \key{RET}}.
```

This produces

> To give the `logout` command, type the characters `l o g o u t RET`.

(Also, this example shows that you can add spaces for clarity. If you really want to mention a space character as one of the characters of input, write `\key{SPC}` for it.)

4.1.3 \key{*key-name*}

Use the `\key` command for the conventional name for a key on a keyboard, as in

```
\key{RET}
```

You can use the `\key` command within the argument of an `\kbd` command when the sequence of characters to be typed includes one or more keys that are described by name.

For example, to produce `C-x ESC` you would type:

```
\kbd{C-x  \key{ESC}}
```

The recommended names to use for keys are in upper case and are

SPC Space

RET Return

LFD Linefeed

TAB Tab

BS Backspace

ESC Escape

DEL Delete

SFT Shift

CTL Control

META Meta

There are subtleties to handling words like 'meta' or 'ctl' that are names of shift keys. When mentioning a character in which the shift key is used, such as `Meta-a`, use the `\kbd` command alone without the `\key` command, but when you are referring to the shift key in isolation, use the `\key` command. For example, write '`\kbd{Meta-a}`' to produce `Meta-a` and '`\key{META}`' to produce `META`. This is because `Meta-a` refers to keys that you press on a keyboard, but `META` refers to a key without implying that you press it.

4.1.4 Ctrl

`\ctrl` is used to describe an ASCII control character. The pattern of usage is `\ctrl{`*ch*`}`, where *ch* is an ASCII character whose control-equivalent is wanted. Thus you put in an '`f`' when you want to indicate a '`control-f`'. For example, to specify '`control-f`', you would enter

```
\ctrl{f}
```

which produces

↑f

In the Info file, this generates the specified control character, output literally into the file. This is done so a user can copy the specified control character (along with whatever else he or she wants) into another Emacs buffer and use it. Since the 'control-h','control-i', and 'control-j' characters are formatting characters, they should not be indicated this way.

In a printed manual, this generates text to describe or identify that control character: an uparrow followed by the character *ch*.

4.1.5 \samp{*text*}

Use the `\samp` command to indicate text that is a literal example of a sequence of characters in a file, string, pattern, etc. Enclose the text in braces. The argument appears within '...' quotation in both the Info file and the printed manual; in addition, it is printed in a fixed-width font.

```
To match  \samp{foo} at the end of the line,
use the regexp  \samp{foo$}.
```

produces

> To match '`foo`' at the end of the line, use the regexp '`foo$`'.

`\samp` is used for entire statements in C, for entire shell commands, and for names of command-line options. Use it for buffer names in Emacs and for node names in Info or LaTeX-info. Often `\samp` is a catchall for whatever is not covered by `\code`, `\kbd`, or `\key`.

Only include punctuation marks within braces if they are part of the string you are specifying. Write punctuation marks outside the braces if those punctuation marks are part of the English text that surrounds the string. In the following sentence, for example, the commas and period are outside of the braces:

```
In English, the vowels are  \samp{a},  \samp{e},
\samp{i},  \samp{o},  \samp{u}, and sometimes
\samp{y}.
```

This produces:

> In English, the vowels are '`a`', '`e`', '`i`', '`o`', '`u`', and sometimes '`y`'.

4.1.6 \var{*metasyntactic-variable*}

Use the `\var` command to indicate metasyntactic variables. A metasyntactic variable is something that stands for another piece of text. For example, you should use a metasyntactic variable in the documentation of a function to describe the arguments that are passed to that function.

Do not use `\var` for the names of particular variables in programming languages. These are specific names from a program, so `\code` is correct for them. For example, the Lisp variable `latexinfo-latex-command` is not a metasyntactic variable; it is properly formatted using `\code`.

The effect of `\var` in the Info file is to upcase the argument; in the printed manual, to italicize it. For example:

```
To delete file  \var{filename},
type  \code{rm  \var{filename}}.
```

produces

To delete file *filename*, type `rm` *filename*.

(Note that `\var` may appear inside of `\code`, `\samp`, `\file`, etc.)

Write a metasyntactic variable all in lower case without spaces, and use hyphens to make it more readable. In some documentation styles, metasyntactic variables are shown with angle brackets, for example:

```
..., type rm <filename>
```

Although this is not the style we use in LaTeXinfo, you can, of course, write your own LaTeXinfo formatting commands to output the `<...>` format if you wish. See section 18 [Extending LaTeXinfo], page 153.

4.1.7 `\file{`*file-name*`}`

Use the `\file` command to indicate text that is the name of a file, buffer, or directory, or is the name of a node in Info. You can also use the command for filename suffixes. Don't use `\file` for symbols in a programming language; thus, a node name is a name in an Info file but not an identifier in a programming language.

Currently, `\file` is equivalent to `\samp` in its effects on the output. For example,

```
The  \file{.el} files are in
the  \file{/usr/local/emacs/lisp} directory.
```

produces

The '`.el`' files are in the '`/usr/local/emacs/lisp`' directory.

4.1.8 `\dfn{`*term*`}`

Use the `\dfn` command to identify the introductory or defining use of a technical term. Use the command only in passages whose purpose is to introduce a term which will be used again or which the reader ought to know. Mere passing mention of a term for the first time doesn't deserve `\dfn`. The command generates italics in the printed manual, and double quotation marks in the Info file. For example:

```
Getting rid of a file is called  \dfn{deleting} it.
```

produces

Getting rid of a file is called *deleting* it.

As a general rule, a sentence containing the defining occurrence of a term should be a definition of the term. The sentence does not have to say explicitly that it is a definition, but it should contain the information of a definition—it should make the meaning clear.

4.2 Emphasizing Text

Usually, LaTeXinfo changes the font to mark words in the text according to what category the words belong to. The `\code` command, for example, does this. Most often, this is the best way to mark words. However, sometimes you will want to emphasize text without indicating a category. LaTeXinfo has two ways to do this: commands that tell LaTeXinfo to emphasize the text but leave the method to the program, and commands that specify the method to use. The first way is generally the best because it makes it possible to change the style of a document without needing to re-edit it line by line.

4.2.1 `\emph{`*text*`}` and `\strong{`*text*`}`

The `\emph` and `\strong` commands are for emphasis; `\strong` is stronger. In printed output, `\emph` produces *italics* and `\strong` produces **bold**.

For example,

```
\begin{quote}
\strong{Caution:}  \code{rm * .[^.]*} removes  \emph{all}
files in the directory.
\end{quote}
```

produces the following in printed output:

> **Caution**: `rm * .[^.]*` removes *all* files in the directory.

and the following in Info:

```
     *Caution*: `rm * .[^.]*' removes *all*
     files in the directory.
```

The `\strong` command is seldom used except to mark what is, in effect, a typographical element, such as the word 'Caution' in the preceding example.[1]

In the Info file, both `\emph` and `\strong` put asterisks around the text.

4.2.2 The Small Caps Font

Use the '`\scap`' command to set text in the printed output in A SMALL CAPS FONT and set text in the Info file in upper case letters.

Write the text between braces in lower case, like this:

```
The  \sc{acm} and  \sc{ieee} are technical societies.
```

This produces:

[1] Don't try to use `\emph` or `\strong` with the word '`Note`'; Info will mistake the combination for a cross reference. Use a phrase such as *Please note* or *Caution* instead.

```
The ACM and IEEE are technical societies.
```

LaTeX typesets the small caps font in a manner that prevents the letters from 'jumping out at you on the page'. This makes small caps text easier to read than text in all upper case. LaTeX typesets any upper case letters in the small caps fonts in FULL-SIZE CAPITALS. Use them sparingly. The Info formatting commands set all small caps text in upper case.

You may also use the small caps font for acronyms such as ATO (a NASA word meaning 'abort to orbit').

There are subtleties to using the small caps font with a jargon word such as CDR, a word used in Lisp programming. In this case, you should use the small caps font when the word refers to the second and subsequent elements of a list (the CDR of the list), but you should use '`\code`' when the word refers to the Lisp function of the same spelling.

4.2.3 Fonts for Printing, Not Info

LaTeXinfo provides four font commands that specify font changes in the printed manual but have no effect in the Info file. `\i` requests *italic* font (in some versions of LaTeX, a slanted font is used), `\b` requests **bold** face, `\t` requests the `fixed-width` font used by `\code`, and `\r` requests a roman font, which is the usual font in which text is printed. In addition `\n` requests the fontsize be set in the normal size of the typeface, All the commands apply to an argument that follows, surrounded by braces. All four commands apply to an argument that follows, surrounded by braces.

Only the `\r` command has much use: in example programs, you can use the `\r` command to convert code comments from the fixed-width font to a roman font. This looks better in printed output.

For example,

```
\begin{lisp}
(+ 2 2)    ;  \r{Add two plus two.}
\end{lisp}
```

produces

```
(+ 2 2)     ; Add two plus two.
```

If possible, you should avoid using the other three font commands. If you need to use one, it probably indicates a gap in the LaTeXinfo language.

4.3 Special Insertions

LaTeXinfo provides several commands for formatting dimensions, for inserting single characters that have special meaning in LaTeXinfo, such as braces, and for inserting special graphic symbols that do not correspond to characters, such as dots and bullets.

These are:

- Braces, '\' and periods.
- Format a dimension, such as '12pt'.
- Dots and bullets.
- The LaTeX logo and the copyright symbol.
- A minus sign.
- The \verb command, for inserting characters verbatim.

4.3.1 Inserting \, Braces, and Periods

'\' and curly braces are special characters in LaTeXinfo. Periods are also special. Depending on whether the period is inside of or at the end of a sentence, less or more space is inserted after a period in a typeset manual. Since it is not always possible for LaTeXinfo to determine when a period ends a sentence and when it is used in an abbreviation, special commands are needed in some circumstances. (Usually, LaTeXinfo can guess how to handle periods, so you don't have to use the special commands; you just enter a period as you would if you were using a typewriter, which means you put two spaces after the period, question mark, or exclamation mark that ends a sentence.) Do not put braces after any of these commands; they are not necessary.

4.3.1.1 Inserting \—\back

\back stands for a single '\' in either printed or Info output.

Do not put braces after an \back command.

4.3.1.2 Inserting '{' and '}'—\{ and \}

\{ stands for a single '{' in either printed or Info output.

\} stands for a single '}' in either printed or Info output.

Do not put braces after either an \{ or an \} command.

4.3.1.3 Spacing After Colons and Periods

Use the \: command after a period, question mark, exclamation mark, or colon that should not be followed by extra space. For example, use \: after periods that end abbreviations which are not at the ends of sentences. \: has no effect on the Info file output.

For example:

```
The U.S.A. \: is a continental nation.
```

produces

The U.S.A. is a continental nation.

Use `\.` instead of a period at the end of a sentence that ends with a single capital letter. Otherwise, LaTeX will think the letter is an abbreviation and will not insert the correct end-of-sentence spacing. Here is an example:

```
Give it to X. and to Y \.  Give it to Z \.
Give it to X. and to Y.  Give it to Z.
```

If you look carefully at this printed output, you will see a little more whitespace after the Y in the first line than the Y in the second line.

Give it to X. and to Y. Give it to Z.
Give it to X. and to Y. Give it to Z.

In the Info file output, `\.` is equivalent to a simple '.'. Do not put braces after either an `\:` or an `\.` command.

4.3.2 `\dmn{`*dimension*`}`: Format a Dimension

At times, you may want to write '`12pt`' or '`8.5in`' with little or no space between the number and the abbreviation for the dimension. You can use the `\dmn` command to do this. On seeing the command, LaTeX inserts just enough space for proper typesetting; the Info formatting commands insert no space at all, since the Info file does not require it. To use the `\dmn` command, write the number and then follow it immediately, with no intervening space, by `\dmn`, and then by the dimension within braces.

For example,

```
A4 paper is 8.27 \dmn{in} wide.
```

produces

```
A4 paper is 8.27in wide.
```

Not everyone uses this style. Instead of '`8.27in`', you may write '`8.27 in.`' or '`8.27 inches`'.

4.3.3 Inserting Ellipsis, Dots, and Bullets

An *ellipsis* (a line of dots) is typeset unlike a string of periods, so a special command is used for ellipsis in LaTeXinfo. The `\bullet` command is special, too. Each of these commands is followed by a pair of braces, '`{}`', without any whitespace between the name of the command and the braces.

`\dots{}` Use the `\dots{}` command to generate an ellipsis, which is three dots in a row, appropriately spaced, like this: '...'. Do not simply write three periods in the input file; that would work for the Info file output, but would produce the wrong amount of space between the periods in the printed manual.

Here is an ellipsis: ...

Here are three periods in a row: ...

In printed output, the three periods in a row are closer together than the dots in the ellipsis.

`\bullet{}` Use the `\bullet{}` command to generate a large round dot, or the closest possible thing to one. In Info, an asterisk is used.

Here is a bullet: •

4.3.4 Inserting LaTeX and the Copyright Symbol

The logo LaTeX is typeset in a special fashion and it needs an `\`-command, as does the command for inserting the copyright symbol. Each of these commands is followed by a pair of braces, '`{}`', without any whitespace between the name of the command and the braces.

`\LaTeX{}` Use the `\LaTeX{}` command to generate 'LaTeX'. In a printed manual, this is a special logo that is different from three ordinary letters. In Info, it just looks like '`LaTeX`'. The `\LaTeX{}` command is amongst the few LaTeXinfo commands in that the `L`, `T` and the `X` are in upper case.

`\copyright{}` Use the `\copyright{}` command to generate '© '. In a printed manual, this is a '`c`' inside a circle, and in Info, this is '`(C)`'.

4.3.5 Inserting a Minus Sign

Use the `\minus{}` command to generate a minus sign. In a fixed-width font, this is a single hyphen, but in a proportional font, the symbol is the customary length for a minus sign—a little longer than a hyphen.

You can compare the two forms:

```
'−' is a minus sign generated with ' \minus{}',

'-' is a hyphen generated with the character '-'.
```

In the fixed-width font used by Info, `\minus{}` is the same as a hyphen.

You should not use `\minus{}` inside of `\code` or `\begin{example}` because the width distinction is not made in the fixed-width font they use.

4.3.6 Inserting Characters Verbatim

You can use the LaTeX `\verb` command to inserting characters verbatim. The next character after the command must be a non-alphabetic or numeric character, such as '`+`'. Any characters between this marker character, and the next occurence of this marker character, will be protected from any operations in LaTeX or Info. The contents will be displayed in a fixed-width font. Unlike LaTeX, LaTeXinfo has a restriction on the use of the `\verb` command: it must occur at the beginning of a line (or preceded only by whitespace). Hence you will usually use it something like:

```
characters to \scap{ascii} double-quotes:
\verb+``+ and
\verb+''+
to \samp{"}.\refill
```

Chapter 5

Displaying Material

Displayed Material are blocks of text consisting of one or more whole paragraphs that are set off from the bulk of the text and treated differently. They are usually indented.

In LaTeXinfo, you always begin a quotation or example by writing an `\begin`-command at the beginning of a line by itself, and end it by writing an `\end` command that is also at the beginning of a line by itself. For instance, you begin an example by writing `\begin{example}` by itself at the beginning of a line and end the example by writing `\end{example}` on a line by at itself, at the beginning of that line.

Since the lines containing `\begin{example}` and `\end{example}` will be turned into blank lines, you won't need to put a blank line before the `\begin{example}`, and another blank line after the `\end{example}`. (Remember that blank lines between the beginning `\begin{example}` and the ending `\end{example}` will also appear in the Info output.)

There are a variety of commands for Displaying Material:

\begin{quote} Used to indicate text that is quoted. The text is filled, and printed in a roman font by default.

\begin{quotation} Used to indicate text that is quoted. The text is filled, indented, and printed in a roman font by default.

\begin{display} Used for illustrative text. The text is indented but not filled, and no font is specified (so, by default, the font is roman).

\begin{format} Used for illustrative text. The text is not indented and not filled and no font is specified (so, by default, the font is roman).

\begin{center} Used to center a body of text.

\begin{flushleft} Used to line up the left margin of unfilled text.

`\begin{flushright}` Used to line up the right margin of unfilled text.

\begin{lisp} Used to illustrate Lisp code. The text is printed in a fixed-width font without filling.

\begin{smalllisp} Used to illustrate Lisp code. The text is printed in a smaller fixed-width font.

\begin{example} Used to illustrate code, and commands. The text is printed in a fixed-width font without filling.

\begin{smallexample} Used to illustrate code, commands, and the like, in a smaller font.

\begin{verbatim} Used to illustrate code and commands, but the text is protected from processing by LaTeX and Info, and is printed in a fixed-width font without filling.

\begin{smallverbatim} Used to illustrate code, commands, and the like. The content is protected from processing by LaTeX and Info, and is set in a smaller font.

The `\exdent` command is used within the above constructs (except of course for the `verbatim` ones) to undo the indentation of a line.

The `\noindent` command may be used after one of the above constructs to prevent the following text from being indented as a new paragraph.

5.1 Quotations

5.1.1 Quotations

The text of a quote is processed normally except that

- The margins are closer to the center of the page, so the whole of the quotation is offset.
- The first lines of paragraphs are indented no more than the other lines.
- In the printed output, interline spacing and interparagraph spacing are reduced.

  This is an example of text written between an `\begin{quote}` command and an `\end{quote}` command. A `\begin{quote}` command is most often used to indicate text that is excerpted from another (real or hypothetical) printed work.

Write an `\begin{quote}` command as text of a line by itself. This line will disappear from the output. Mark the end of the quotation with a line beginning with and containing only `\end{quote}`. The `\end{quote}` line will likewise disappear from the output. Thus, the input

```
\begin{quote}
This is
a foo.
\end{quote}
```

produces

> This is a foo.

The text of a `\begin{quotation}` environment is processed the same way, except that the first line of the text is indented.

5.2 Justifying Text

5.2.1 Left Justification and Right Justification

The `\begin{flushleft}` and `\begin{flushright}` commands line up the left or right ends of lines on the left and right margins of a page, but do not fill the text. The commands are written on lines of their own, without braces. The `\begin{flushleft}` and `\begin{flushright}` commands are ended by `\end{flushleft}` and `\end flushright` commands on lines of their own.

For example,

```
\begin{flushleft}
This text is
written flushleft.
\end{flushleft}
```

produces

> This text is written flushleft. The \code{\begin{flushleft}}
> command left justifies every line but leaves the
> right end ragged.

Flushright produces the type of indentation often used in the return address of letters.

```
\begin{flushright}
Here is an example of text written
flushright.  The  \code{ \begin{flushright}} command
right justifies every line but leaves the
left end ragged.
\end{flushright}
```

produces

> Here is an example of text written flushright. The `\begin{flushright}` command right justifies every line but leaves the left end ragged.

5.2.1.1 Center Environment

Text enclosed in a `center` environment produces lines of output containing text centered between the margins. This is the same as the `center` environment of LaTeX, and different from the TeXinfo command of the same name.

5.3 Display Environments

5.3.1 \begin{display}

The \begin{display} command begins a kind of example. It is like the \begin{example} command except that, in a printed manual, \begin{display} does not select the fixed-width font. In fact, it does not specify the font at all, so that the text appears in the same font it would have appeared in without the \begin{display} command.

```
This is an example of text written between an
\begin{display} command and an \end{display}
command. The \begin{display} command indents the text,
but does not fill it.
```

5.3.2 \begin{format}

The \begin{format} command is similar to \begin{example} except that, in the printed manual, \begin{format} does not select the fixed-width font and does not narrow the margins.

```
This is an example of text written between an \
begin{format} command and an \end{format} command. The
\begin{format} command does not fill the text.
```

5.4 Examples and Verbatim

5.4.1 \begin{example}

The \begin{example} command is used to indicate an example that is not part of the running text, such as computer input or output.

```
This is an example of text written between an \begin{example}
command and an \end{example} command. The text is
indented but not filled.

In the printed manual, the text is typeset in a fixed-width font, and
extra spaces and blank lines are significant. In the Info file, an
analogous result is obtained by indenting each line with five extra
spaces.
```

Write an \begin{example} command at the beginning of a line (or possibly preceded by whitespace) as the only text on a line by itself. This line will turn into a blank line in the Info output. Mark the end of the example with a line beginning containing only \end{example} (or possibly preceded by whitespace). The \end{example} will likewise turn into a blank line in the Info output. For example:

```
\begin{example}
mv foo bar
\end{example}
```

produces

```
mv foo bar
```

> **Caution:** Do not use tabs in lines of an example (or anywhere else in LaTeXinfo, for that matter)! LaTeX treats tabs like single spaces, and that is not what they look like. This is a problem with LaTeX. (If necessary, in Emacs, you can use `M-x untabify` to convert tabs in a region to multiple spaces.)

Examples are often, logically speaking, "in the middle" of a paragraph, and the text continues after an example should not be indented. The `\noindent` command prevents a piece of text from being indented as if it were a new paragraph.

(The `\code` command is used for examples of code that is embedded within sentences, not set off from preceding and following text. See section code in `\code`.)

5.4.2 \noindent

If you have text following an `\begin{example}` or other similar inclusion that reads as a continuation of the text before the `\begin{example}`, it is good to prevent this text from being indented as a new paragraph. To accomplish this, write `\noindent` at the beginning of a line by itself preceding the continuation text. For example,

```
\begin{example}
This is an example
\end{example}

\noindent
This line will not be indented.  As you can see, the
beginning of the line is fully flush left with the line
that follows after it.  (This whole example is between
\begin{display} and  \end{display}.)
```

produces

```
This is an example
```

```
This line will not be indented.  As you can see, the beginning of the
line is fully flush left with the line that follows after it.  (This
whole example is between  \begin{display} and
\end{display}.)
```

To adjust the number of blank lines properly in the Info file output, remember that the line containing `\noindent` does not generate a blank line, and neither does the `\end{example}` line.

In the LaTeXinfo source file for this documentation, each of the lines that says 'produces' is preceded by a line containing `\noindent`. Do not put braces after an `\noindent` command.

The `smallexample` environment sets its contents in a smaller font.

5.4.3 `\begin{lisp}`

The `\begin{lisp}` command is used for Lisp code. It is synonymous with the `\begin{example}` command.

```
This is an example of text written between an
\begin{lisp} command and an  \end{lisp} command.
```

Use `\begin{lisp}` instead of `\begin{example}` so as to preserve information regarding the nature of the example. This is useful, for example, if you write a function that evaluates only and all the Lisp code in a LaTeXinfo file. Then you can use the LaTeXinfo file as a Lisp library.[1]

Mark the end of `\begin{lisp}` with `\end{lisp}` on a line by itself.

The `smalllisp` environment sets its contents in a smaller font.

5.4.4 Verbatim Environment

The `verbatim` environment is very similar to the `example` environment except that no parsing of the contents is carried out, and the text is not indented. In the Info file things will appear exactly as they have been typed in. In the printed manual, this is the same as the LaTeX command of the same name. Verbatim environments cannot be nested, nor can they appear inside another environment such as `example`. The `\begin{verbatim}` and `\end{verbatim}` *must* occur at the beginning of a line.

The `smallverbatim` environment sets its contents in a smaller font.

The `verbatimfile` command includes the contents of a file with a `verbatim` environment. The command is followed by an `\end{verbatim}` command, such as

```
\verbatimfile{foo.bar}
\end{verbatim}
```

The `smallverbatimfile` command sets its argument in a smaller font, and is terminated by an `\end{smallverbatim}` command.

[1] It would be straightforward to extend LaTeXinfo to work in a similar fashion for C, Fortran, or other languages.

5.5 Controlling Indentation

5.5.1 exdent: Undoing a Line's Indentation

The `\exdent` command removes any indentation a line might have. The command is written at the beginning of a line and applies only to the text that follows the command that is on the same line. Don't use braces around the text. In the printed manual, the text on the `\exdent` line is printed in the roman font.

`\exdent` is usually used within examples. Thus,

```
\begin{example}
This line follows an  \begin{example} command.
\exdent{This line is exdented.}
This line follows the exdented line.
The  \end{example} comes on the next line.
\end{example}
```

produces

```
   This line  follows an  \begin{example} command.
This line is exdented.
   This line follows the exdented line.
   The  \end{example} comes on the next line.
```

In practice, the `\exdent` command is rarely used. Usually, you un-indent text by ending the example and returning the page to its normal width.

5.6 Drawing Cartouches Around Examples

In a printed manual, the `cartouche` environment draws a box with rounded corners around its contents. Pair with `\end{cartouche}`. You can use this command to further highlight an example or quotation. For instance, you could write a manual in which one type of example is surrounded in a cartouche to emphasize them.

The `\cartouche` command affects only the printed manual; it has no effect in the Info file. For example,

```
\begin{example}
\cartouche
% pwd
/usr/local/lib/emacs/info
\end{cartouche}
\end{example}
```

produces

```
% pwd
/usr/local/lib/emacs/info
```

surrounds the two-line example with a box with rounded corners, in the printed manual.

5.7 Special Glyphs for Examples

In LaTeXinfo, code is often illustrated in examples that are delimited by `\begin{example}` and `\end{example}`, or by `\begin{lisp}` and `\end{lisp}`. In such examples, you can indicate the results of evaluation or an expansion using '⇒' or '↦'. Likewise, there are special symbols to indicate printed output, an error message, equivalence of expressions, and the location of point. The special glyph commands do not have to be used within an example. Every special glyph command is followed by a pair of left- and right-hand braces.

Here are the different special glyph commands:

⇒ `\result{}` points to the result of an expression.

↦ `\expansion{}` shows the results of a macro expansion.

⊣ `\print{}` indicates printed output.

error `\error{}` indicates that the following text is an error message.

≡ `\equiv{}` indicates the exact equivalence of two forms.

⋆ `\point{}` shows the location of point.

5.7.1 ⇒: Indicating Evaluation

Use the `\result{}` command to indicate the result of evaluating an expression. The `\result{}` command is displayed as '`=>`' in Info and as '⇒' in the printed output. Thus, the following,

```
(cdr '(1 2 3))
     ⇒ (2 3)
```

may be read as "`(cdr '(1 2 3))` evaluates to (2 3)".

5.7.2 ↦: Indicating an Expansion

When an expression is a macro call, it expands into a new expression. You can indicate the result of the expansion with the `\expansion{}` command. The `\expansion{}` command is displayed as '`==>`' in Info and as '↦' in the printed output. For example, the following

```
\begin{lisp}
(third '(a b c))
      \expansion{} (car (cdr (cdr '(a b c))))
      \result{} c
\end{lisp}
```

produces

```
(third '(a b c))
     ↦ (car (cdr (cdr '(a b c))))
     ⇒ c
```

which may be read as:

> `(third '(a b c))` expands to `(car (cdr (cdr '(a b c))))`; the result of evaluating the expression is `c`.

(Often, as in this case, an example looks better if the `\expansion{}` and `\result{}` commands are indented five spaces.)

5.7.3 ⊣: Indicating Printed Output

Sometimes an expression will print output during its execution. You can indicate the printed output with the `\print{}` command. The `\print{}` command is displayed as '`-|`' in Info and as '⊣' in the printed output.

In the following example, the printed text is indicated with '⊣', and the value of the expression follows on the last line.

```
(progn (print 'foo) (print 'bar))
     ⊣ foo
     ⊣ bar
     ⇒ bar
```

In a LaTeXinfo source file, this example is written as follows:

```
\begin{lisp}
(progn (print 'foo) (print 'bar))
      \print{} foo
      \print{} bar
      \result{} bar
\end{lisp}
```

5.7.4 error : Indicating an Error Message

A section of code may cause an error when you evaluate it. You can designate the error message with the `\error{}` command. The `\error{}` command is displayed as '`error-->`' in Info and as 'error ' in the printed output. Thus,

```
\begin{lisp}
(+ 23 'x)
\error{} Wrong type argument: integer-or-marker-p, x
\end{lisp}
```

produces

```
(+ 23 'x)
error  Wrong type argument: integer-or-marker-p, x
```

This indicates that the following error message is printed when you evaluate the expression:

```
Wrong type argument: integer-or-marker-p, x
```

Note that 'error ' itself is not part of the error message.

5.7.5 ≡: Indicating Equivalence

Sometimes two expressions produce identical results. You can indicate the exact equivalence of two forms with the `\equiv{}` command. The `\equiv{}` command is displayed as '`==`' in Info and as '≡' in the printed output.

Thus,

```
\begin{lisp}
(make-sparse-keymap)  \equiv{} (list 'keymap)
\end{lisp}
```

produces

```
(make-sparse-keymap) ≡ (list 'keymap)
```

This indicates that evaluating `(make-sparse-keymap)` produces identical results to evaluating `(list 'keymap)`.

5.7.6 Indicating Point in a Buffer

Sometimes you need to show an example of text in an Emacs buffer. In such examples, the convention is to include the entire contents of the buffer in question between two lines of dashes containing the buffer name.

You can use the '`\point{}`' command to show the location of point in the text in the buffer. (The symbol for point, of course, is not part of the text in the buffer; it indicates the place *between* two characters where point is located.) The `\point{}` command is displayed as '`-!-`' in Info and as '$\star$' in the printed output.

The following example shows the contents of buffer '`foo`' before and after evaluating a Lisp command to insert the word `changed`.

```
---------- Buffer: foo ----------
This is the ⋆contents of foo.
---------- Buffer: foo ----------


(insert "changed ")
     ⇒ nil
---------- Buffer: foo ----------
This is the changed ⋆contents of foo.
---------- Buffer: foo ----------
```

In a LaTeXinfo source file, the example is written like this:

```
\begin{example}
---------- Buffer: foo ----------
This is the  \point{}contents of foo.
---------- Buffer: foo ----------

(insert "changed ")
      \result{} nil
---------- Buffer: foo ----------
This is the changed  \point{}contents of foo.
---------- Buffer: foo ----------
\end{example}
```

5.8 Conditionally Visible Text

Sometimes it is good to use different text for a printed manual and its corresponding Info file. In this case, you can use the conditional commands to specify which text is for the printed manual and which is for the Info file.

`\begin{ifinfo}` begins text that should be ignored by LaTeX when it typesets the printed manual. The text appears only in the Info file. The `\begin{ifinfo}` command should appear on a line by itself. End the Info-only text with a line containing `\end{ifinfo}` by itself. The `\begin{iftex}` and `\end{iftex}` commands are used similarly but to delimit text that will appear in the printed manual but not in the Info file.

For example,

```
\begin{iftex}
This text will appear only in the printed manual.
\end{iftex}

\begin{ifinfo}
However, this text will appear only in Info.
\end{ifinfo}
```

The preceding example produces the following. Note how you only see one of the two lines, depending on whether you are reading the Info version or the printed version of this manual.

This text will appear only in the printed manual.

5.8.1 Using Ordinary LaTeX Commands

Inside a region delineated by `\begin{iftex}` and `\end{iftex}`, you can embed some LaTeX commands. Info will ignore these commands since they are only in that part of the file that is seen by LaTeX.

You can enter LaTeX completely by delineating a region with the `\begin{tex}` and `\end{tex}` commands. The characters `# $ % ^ & _ |` all revert to their normal LaTeX meanings. The `\begin{tex}` command also causes Info to ignore the region, like the `\begin{iftex}` command.

For example, here is some mathematics:

```
\begin{tex}
$ \bigl(x \in A(n) \bigm| x \in B(n) \bigr)$
\end{tex}
```

The output of this example will appear only in the printed manual. If you are reading this in Info, you will not see anything after this paragraph. In the printed manual, the above mathematics looks like this:

$\bigl(x \in A(n) \bigm| x \in B(n) \bigr)$

Chapter 6

Making Lists Tables and Descriptions

LaTeXinfo has several ways of making lists and tables. Lists can be bulleted or numbered, while descriptions can highlight the items in the first column.

LaTeXinfo automatically indents the text in lists or descriptions, and numbers an enumerated list. This last feature is useful if you modify the list, since you do not have to renumber it yourself.

Numbered lists and tables begin with the appropriate `\begin` command at the beginning of a line, and end with the corresponding `\end` command on a line by itself. Begin an enumerated list, for example, with an `\begin{enumerate}` command and end the list with an `\end{enumerate}` command. Begin an itemized list with an `\begin{itemize}` command, and end the list with an `\end{itemize}` command. You precede each element of a list with an `\item` command.

Here is an itemized list of the different kinds of table and lists:

- Itemized lists with and without bullets.
- Numbered lists.
- Descriptions with highlighting.

Here is an enumerated list with the same items:

1. Itemized lists with and without bullets.
2. Numbered lists.
3. Descriptions with highlighting.

And here is a description with the same items and their \-commands:

\begin{itemize} Itemized lists with and without bullets.

\begin{enumerate} Numbered lists.

\begin{description} two-column descriptions with highlighting.

\begin{tabular} Multio–column tables.

6.1 Itemize Environment

The `\begin{itemize}` is used to produce sequences of indented paragraphs, with a mark inside the left margin at the beginning of each paragraph. The text of the indented paragraphs themselves come after the `\begin{itemize}`, up to another line that says `\end{itemize}`. Before each paragraph for which a mark in the margin is desired, place a line that says just `\item`. It's best not to put any other text on this line.

Before each paragraph for which a mark in the margin is desired, place a line that says just `\item`. Don't put any other text on this line.

Usually, you should put a blank line before an `\item`. This puts a blank line in the Info file. (LaTeX inserts the proper interline whitespace in either case.) Except when the entries are very brief, these blank lines make the list look better.

Here is an example of the use of `\begin{itemize}`, followed by the output it produces.

```
\begin{itemize}
\item
Some text for foo.

\item
Some text
for bar.
\end{itemize}
```

produces

- Some text for foo.
- Some text for bar.

Itemized lists may be embedded within other itemized lists.

6.2 Enumerate Environment

`\begin{enumerate}` is like `\begin{itemize}` except that the marks in the left margin contain successive integers starting with 1. (See the preceeding section.) Do not put any argument on the same line as `\begin{enumerate}`.

Normally, you should put a blank line between the entries in the list. This generally makes it easier to read the Info file.

```
\begin{enumerate}
\item
Some text for foo.

\item
Some text for bar.
\end{enumerate}
```

produces

1. Some text for foo.
2. Some text for bar.

6.3 Description Environment

The `description` environment is similar to `\begin{itemize}`, but allows you to specify a name or heading line for each item. (See section 6.1 [Itemize Environment], page 60.) The command is used to produce two-column descriptions, and is especially useful for glossaries and explanatory exhibits. You must follow each use of `\item` inside of the description environment with text to serve as the heading line for that item. This text is put inside square brackets on the same line as the `\item` command. Each heading line is put into the first column of the table and the supporting text, which you put on the line following the line beginning with `\item`, goes into the second column.

Usually, you should put a blank line before a `\item`. This puts a blank like in the Info file. Except when the entries are very brief, a blank line looks better. The following description highlights the text in the first column:

```
\begin{description}
\item[foo]
This is the text for \samp{foo}.
\item[bar]
This is the text for \samp{bar}.
\end{description}
```

produces

foo This is the text for ‘`foo`’.

bar This is the text for ‘`bar`’.

Info indents the lines of text in the second column, but does not automatically fill them. As a result, the lines in the Info file may be too wide. To prevent this, cause Info to refill the paragraphs after processing by adding the command `\refill` to the end of the paragraph. (See section 7.4 [Refilling Paragraphs], page 68, for more information about the use of the `\refill` command.)

6.4 Tabular Environment

The LaTeX `tabular` environment is weakly supported by LaTeXinfo. This environment makes it easy to set small multi–column tables. The ampersand character has its special LaTeX meaning of a separator in tables. To insert a `&`, type `\&`.

In the `tabular` environment, you must line the columns up the way you want them to appear in the Info file, and you must use `&` as a separator. In the Info file, the separator will become a `SPC` character thus preserving the alignment. The trailing `\\` will be stripped; these *must* occur at the end of the line.

The `hline` command is supported by the Info program. It will insert a line of hyphens all the way to the current `fill-column`. Neither the `cline` or `multicolumn` commands are supported. For example:

```
\begin{table}[hbtp]
\caption{The First Table's Caption}
\begin{tabular}{||l|l|l|l||}
\hline
Column A        & Column B      & Column C      & Column D      \\
\hline
A 1             & B 1           & C 1           & D 1           \\
A 2             & B 2           & C 2           & D 2           \\
A 3             & B 3           & C 3           & D 3           \\
\hline
\end{tabular}
\end{table}
```

produces in the Info file

```
                  Table 1 : The First Table's Caption
------------------------------------------------------------------------
Column A          Column B        Column C        Column D
------------------------------------------------------------------------
A 1               B 1             C 1             D 1
A 2               B 2             C 2             D 2
A 3               B 3             C 3             D 3
------------------------------------------------------------------------
```

and in the LaTeX file this produces:

Table 6.1: The First Table's Caption

Column A	Column B	Column C	Column D
A 1	B 1	C 1	D 1
A 2	B 2	C 2	D 2
A 3	B 3	C 3	D 3

The LaTeX math environments `displaymath`, `equation`, `eqnarray` and `array` are completely ignored in the Info file, but will have their LaTeX definitions in the printed manual.

6.5 Figures and Tables

Tables and Figures are only weakly supported by LaTeXinfo. Anything within a `figure` environment is completely ignored in the Info file: `\begin{figure}` is equivalent to `\begin{tex}`

`\begin{table}` is supported by LaTeXinfo, as is the `caption` command. The lines containing the `\begin{table}` and `\end{table}` are deleted from the Info file. The `caption` command causes its argument to be centered on a line, preceded by the word Table and the table number. `caption`s are assumed to be within tables because figures are not supported.

Chapter 7

Formatting Paragraphs

7.1 Making and Preventing Breaks

Usually, a LaTeXinfo file is processed both by LaTeX and by one of the Info formatting commands. Sometimes line, paragraph, or page breaks occur in the 'wrong' place in one or other form of output. You must ensure that text looks right both in the printed manual and in the Info file.

For example, in a printed manual, page breaks may occur awkwardly in the middle of an example; to prevent this, you can hold text together using a grouping command that keeps the text from being split across two pages. Conversely, you may want to force a page break where none would occur normally. Fortunately, problems like these do not often arise. When they do, use the following commands.

7.2 The Line Breaking Commands

The line break commands create line breaks:

***** Force a line break in the printed manual and in the Info file.

**** Force a line break in the Info file.

\sp{*n***}** Skip *n* blank lines.

The line-break-prevention command holds text together all on one line.

\w{*text***}** Prevent *text* from being split across two lines.

7.2.1 *: Generate Line Breaks

The ***** command forces a line break in both the printed manual and in Info. The **** command forces a line break in the printed manual. The optional argument to the LaTeX **** command is not supported in LaTeXinfo.

For example,

```
This line  \* is broken  \*in two places.
```

produces

```
This line
is broken
in two places.
```

(Note that the space after the first `\*` command is faithfully carried down to the next line.)

```
This is version 2.0 of the LaTeX info documentation, \*
and is for ...
```

In this case, the `\*` command keeps LaTeX from stretching the line across the whole page in an ugly manner.

Do not write braces after an `\*` command; they are not needed. Do not write an `\refill` command at the end of a paragraph containing an `\*` command; it will cause the paragraph to be refilled after the line break occurs, negating the effect of the line break.

7.2.2 Preventing Line Breaks

`\w{`*text*`}` outputs *text* and prohibits line breaks within *text*.

You can use the `\w` command to prevent LaTeX from automatically hyphenating a long name or phrase that accidentally falls near the end of a line.

```
You can copy GNU software from  \w{ \file{prep.ai.mit.edu}}.
```

produces

> You can copy GNU software from '`prep.ai.mit.edu`'.

In the LaTeXinfo file, you must write the `\w` command and its argument (all the affected text) all on one line.

Do not write an `\refill` command at the end of a paragraph containing an `\w` command; it will cause the paragraph to be refilled and may thereby negate the effect of the `\w` command.

7.2.3 Inserting Blank Lines

A line beginning with and containing only `\sp` *n* generates *n* blank lines of space in both the printed manual and the Info file. `\sp` also forces a paragraph break. For example,

```
\sp{2}
```

generates two blank lines.

7.3 The Page Breaking Commands

The pagination commands apply only to printed output, since Info files do not have pages.

\clearpage Start a new page in the printed manual.

\begin{same} Hold text together that must appear on one printed page.End the text to be held together with `\begin{same}`

\need{*mils*} Start a new printed page if not enough space on this one.

7.3.1 Start a New Page

A line containing only `\clearpage` starts a new page in a printed manual. The command has no effect on Info files since they are not paginated. An `\clearpage` command is often used in the title section of a LaTeXinfo file to start the copyright page.

7.3.2 Putting things on the Same Page

The `\begin{same}` command (on a line by itself) is used inside of an `\begin{example}` or similar construct to begin an unsplittable vertical group, which will appear entirely on one page in the printed output. The group is terminated by a line containing only `\end{same}`. These two lines produce no output of their own, and in the Info file output they have no effect at all.

Although `\begin{same}` would make sense conceptually in a wide variety of contexts, its current implementation works reliably only within `\begin{example}` and variants, and within `\begin{quote}`, `\begin{display}`, `\begin{format}`, `\begin{flushleft}` and `\begin{flushright}`. (What all these commands have in common is that they turn off vertical spacing between "paragraphs".) In other contexts, `\begin{same}` can cause anomalous vertical spacing. See section 5 [Displaying Material], page 47.
with the `\begin{same}` and `\end{same}` command insides of the `\begin{example}` and `\end{example}` commands.

The `\begin{same}` command is most often used to hold an example together on one page. In this LaTeXinfo manual, about 100 examples contain text that is enclosed between `\begin{same}` and `\end group`.

7.3.3 Prevent Page Breaks

A line containing only `\need` *n* starts a new page in a printed manual if fewer than *n* mils (thousandths of an inch) remain on the current page. The `\need` command has no effect on Info files since they are not paginated.

This paragraph is preceded by an `\need` command that tells LaTeX to start a new page if fewer than 300 mils (nearly one-third inch) remain on the page. It looks like this:

```
\need{300}
This paragraph is preceded by ...
```

The `\need` command is useful for preventing orphans (single lines at the bottoms of printed pages).

7.4 Refilling Paragraphs

The `\refill` command refills and, optionally, indents the first line of a paragraph.[1]

If a paragraph contains long `\`-constructs, the paragraph may look badly filled after being formatted by `latexinfo-format-region` or `latexinfo-format-buffer`. This is because both of thes commands remove `\`-commands from formatted text but do not refill paragraphs automatically although LaTeX does. Consequently, some lines become shorter than they were. To cause these commands to refill a paragraph, write `\refill` at the end of the paragraph. This command refills a paragraph in the Info file after all the other processing has been done. `\refill` has no effect on LaTeX, which always fills every paragraph that ought to be filled.

For example, without any indenting, the following

```
To use  \code{foo}, pass  \samp{xx%$} and
\var{flag} and type  \kbd{x} after running
\code{make-foo}. \refill
```

produces (in the Info file)

```
To use ‘foo’, pass ‘xx%$’ and FLAG and type ‘x’ after
running ‘make-foo’.
```

whereas without the `\refill` it would produce

```
To use ‘foo’, pass ‘xx%$’ and
FLAG and type ‘x’ after running
‘make-foo’.
```

with the line broken at the same place as in the LaTeXinfo input file.

Write the `\refill` command at the end of the paragraph. Do not put a space before `\refill`; otherwise the command might be put at the beginning of the line when you refill the paragraph in the LaTeXinfo file with Emacs command `M-q` (`fill-paragraph`). If this were to happen, the `\refill` command might fail to work. Do not put braces after `\refill`. Because an `\refill` command is placed at the end of a paragraph and never at the beginning of a line, the braces are not necessary.

[1] Perhaps the command should have been called the `\refillandindent` command, but `\refill` is shorter and the name was chosen before indenting was available.

You can write an `\refill` command at the end of a footnote before the footnote's closing brace, even if the footnote text is embedded in a the middle of a paragraph in the LaTeXinfo file. This is because the footnote text is extracted from the surrounding text and formatted on its own.

Also, do not end a paragraph that uses either `\*` or `\w` with an `\refill` command; otherwise, `latexinfo-format-buffer` or `latexinfo-format-buffer` will refill the paragraph in spite of those commands.

In addition to refilling, the `\refill` command may insert spaces at the beginning of the first line of the paragraph, thereby indenting that line. The argument to the `\paragraphindent` command specifies the amount of indentation: if the value of the argument is 0, an `\refill` command deletes existing indentation. If the value of the argument is greater than 0, an `\refill` command indents the paragraph by that number of spaces. And if the value of the argument is '`asis`', an `\refill` command does not change existing indentation. For more information about the `\paragraphindent` command, section paragraphindent in Paragraph Indenting.

The `\refill` command does not indent entries in a list, table, or definition, nor does `\refill` indent paragraphs preceded by `\noindent`.

7.5 Always Refilling Paragraphs

In practice, one finds that many paragraphs in a LaTeXinfo document need refilling, and one's document is littered with `\refill` commands. One solution is to write a 6000 line '`C`' program to do the refilling automatically. This would have the advantage of great speed, but would mean maintaining a two versions of the Info formating program, one in '`C`' and one in Emacs lisp.

Another solution is to implement a heuristic[2] that searches for likely candidates for refilling, and inserts a `\refill` command there. At the moment, the replacement takes place at any period followed by two newlines, or a period followed by a newline, followed by `\end{`. Of course, no replacements are made within `verbatim` or `smallverbatim` environments.

Implementation note: This is implemented as a search and replace of all occurences matching the string `".\n\n"` or `".\n\end{"`. This feature is likely to slow things down on a large document. This matching string should probably be changed to the regular expression `\\s.\n\n` or `\\s.\n\end{`

[2] The '`H`' in Heuristic is pronounced, as in Hack.

Chapter 8

Citations and Footnotes

8.1 Footnotes

A *footnote* is for a reference that documents or elucidates the primary text.[1] In LaTeXinfo, footnotes are created with the `\footnote` command. This command is followed immediately by a left brace, then by the text of the footnote, and then by a terminating right brace. The template is: `\footnote{`*text*`}`.

For example, this clause is followed by a sample footnote;[2] in the LaTeXinfo source, it looks like this:

```
...a sample footnote  \footnote;{Here is the sample
footnote.} in the LaTeX info source...
```

In a printed manual or book, the reference mark for a footnote is a small, superscripted number; the text of the footnote is written at the bottom of the page, below a horizontal line.

In Info, the reference mark for a footnote is a pair of parentheses with the footnote number between them, like this: '`(1)`'. Info has two footnote styles, which determine where the text of the footnote is located:

- In the *end* of node style, all the footnotes for a single node are placed at the end of that node. The footnotes are separated from the rest of the node by a line of dashes with the word '`Footnotes`' within it. Each footnote begins with an '`(`*n*`)`' reference mark.

  Here is an example of a single footnote in the end of node style:

```
--------- Footnotes ---------

(1)  Here is a sample footnote.
```

[1] A footnote should complement or expand upon the primary text, but a reader should not need to read a footnote to understand the primary text. For a thorough discussion of footnotes, see *The Chicago Manual of Style*, which is published by the University of Chicago Press.

[2] Here is the sample footnote.

- In the *separate* style, all the footnotes for a single node are placed in an automatically constructed node of their own. In this style, a "footnote reference" follows each '(*n*)' reference mark in the body of the node. The footnote reference is actually a cross reference and you use it to reach the footnote node.

  The name of the footnotes' node is constructed by appending '`-Footnotes`' to the name of the node that contains the footnotes. (Consequently, the footnotes' node for the '`Footnotes`' node is '`Footnotes-Footnotes`'!) The footnotes' node has an 'Up' node pointer that leads back to its parent node.

Here is how the first footnote in this manual looks after being formatted for Info in the separate node style:

```
File: latexinfo.info  Node: Overview-Footnotes, Up: Overview

(1) Note that the first syllable of "texinfo" is pronounced like
"speck", not "hex". ...
```

A LaTeXinfo file may be formatted into an Info file with either footnote style.

Use the `\footnotestyle` command to specify an Info file's footnote style. Write this command at the beginning of a line followed by an argument, either '`end`' for the end node style or '`separate`' for the separate node style. For example:

```
\footnotestyle{end}
```

or

```
\footnotestyle{separate}
```

The `\footnotestyle` command should be written in the header, before the `\setfilename` and shortly after the `\begin{document}` at the beginning of a LaTeXinfo file. See section 2.4.4 [Custom Headings], page 19. (If you include the `\footnotestyle` command between the start of header and end of header lines, the region formatting commands will format footnotes as specified.) If you do not specify a footnote style, the formatting commands will chose a default style.

8.2 Citations

`\cite` is the LaTeX command for a bibliographic citations. Citations are usually prepared using the program BibTeX, which formats the citations for use with LaTeX. The argument to the `\cite` command is the citation key, which appears in the printed manual as the citation key surrounded by square brackets. How it appears in the printed manual is dependent on the bibliographic style chosen. See the LaTeX Manual for more details [Lam86].

Before you use the `\cite` command, you must declare the bibliography style that you are going to use. See section 2.9.1 [Making a Bibliography], page 27.

Chapter 9

Input and Include Files

LaTeX has two ways of including files: with the `\input` command, and with the `\include` command. LaTeX makes some important distinctions between the two. See [Lam86, § 4.4] for the exact nature of the differences. In LaTeX. Input files are simply inserted at the place where the `input` command occurs, both in the Info file and the LaTeX file. `include` files have seperate auxilliarly files ('`.aux`'), and you can control which files are processed with the `includeonly` command.

In LaTeXinfo, the Info program ignores the `includeonly` command. Both `include` and `input` files are always processed. `input` files are always ignored by the `latexinfo-multiple-files-update` command, which creates or updates or updates the `\node` entries in a file, whereas, under certain conditions, this command will recognize the structure of `include` files. See section 15.3.3 [latexinfo-multiple-files-update], page 133 for details

9.1 Input Files

A line of the form `\input{`*filename*`}` will include the contents of the file *filename* at that point. A standard technique is to have a top–level file, used only for making a comprehensive manual, containing nothing but the beginning, the end, and a series of `\input` commands. The `\input` *must* occur at the beginning of a line.

A file that is intended to be processed with `\input` should not end with `\end{document}`, since that would terminate LaTeX processing immediately.

9.2 Include Files

When LaTeX or an Info formatting command sees an `\include` command in a LaTeXinfo file, it processes the contents of the file named by the command and incorporates them into the DVI or Info file being created. Index entries from the included file are incorporated into the indices of the output file.

An included file should simply be a segment of text that you expect to be included as-is into the overall or *outer* LaTeXinfo file; it should not contain the standard beginning and end parts

of a LaTeXinfo file. In particular, you should not start an included file with a '`\documentstyle`' command. Likewise, you should not end an included file with an `\end{document}` command; that command will stop LaTeX processing immediately.

9.2.1 How to Use Include Files

To include another file within a LaTeXinfo file, write the `\include` command at the beginning of a line and follow it on the same line by the name of a file to be included. For example:

```
\include{chap47.tex}
```

Conventionally, an included file begins with an `\node` line that is followed by an `\chapter` line. Each included file is one chapter. This makes it easy to use the regular node and menu creating and updating commands to create the node pointers and menus within the included file. However, the simple Emacs node and menu creating and updating commands do not work with multiple LaTeXinfo files. Thus you cannot use these commands to fill in the 'Next', 'Previous', and 'Up' pointers of the `\node` line that begins the included file. Also, you cannot use the regular commands to create a master menu for the whole file. Either you must insert the menus and the first and last 'Next', 'Previous', and 'Up' pointers by hand, or you must use the `latexinfo-multiple-files-update` command that is designed for `\include` files. See section 15.3.3 [latexinfo-multiple-files-update], page 133

9.2.2 Sample File with `\include`

If you plan to use the `latexinfo-multiple-files-update` command, the outer LaTeXinfo file that lists included files within it should contain nothing but the beginning and end parts of a LaTeXinfo file, and a number of `\include` commands listing the included files. It should not even include indices, which should be listed in an included file of their own.

Moreover, each of the included files must contain exactly one highest level node (conventionally, an `\chapter` node or equivalent), and this node must be the first node in the included file. Furthermore, each of these highest level nodes in each included file must be at the same hierarchical level in the file structure. Usually, each is an `\chapter`, an `\chapter`, or an `\unnumbered` node. Thus, normally, each included file contains one, and only one, chapter or equivalent-level node.

The outer file should *not* contain any nodes besides the single 'Top' node. The `latexinfo-multiple-files-update` command will not process them.

Here is an example of an outer LaTeXinfo file with `\include` files within it before running `latexinfo-multiple-files-update`, which would insert a main or master menu:

```
 \documentstyle[12pt,latexinfo]{book}
 \pagestyle{headings}
 \begin{document}
 \bibliographystyle{alpha}

 \newindex{fn}

 \title{The Manual}

 \author{Fred Foobar,\\
Clarke Institute,\\
999 Queen Street,\\
Toronto, Ontario}

 \date{\today}

 \maketitle
 \tableofcontents
 \clearpage

 \setfilename{themanual.info}

 \include{foo.tex}
 \include{bar.tex}

 \bibliography{references}

 \twocolumn
 \unnumbered{Function Index}
 \printindex{fn}

\end{document}
```

An included file, such as '`foo.tex`', might look like this:

```
\node First, Second, , Top
\chapter{First Chapter}

Contents of first chapter ...
```

The full contents of '`index.tex`' might be as simple as this:

```
\unnumbered{Concept Index, , Second, Top}
\printindex{cp}
```

Chapter 10

Definition Commands

The `\deffn` command and the other *definition commands* enable you to describe functions, variables, macros, commands, user options, special forms and other such constructs in a uniform format.

These constructs are most often used for documenting Lisp and C programs, and the table below summarizes the different constucts, their language on usual usage, and their class. We will order these functions by their usage: untyped languages such as Lisp, typed languages such as C, or object–oriented langauges such as CLOS.

Command Name	Language	Class
deffn	Lisp	general functions
deffun	Lisp	functions
defspec	Lisp	special forms
defmac	Lisp	macros
defvr	Lisp	general variables
defvar	Lisp	variables
deftypefn	C	general typed functions
deftypefun	C	typed functions
deftypevr	C	general typed variables
deftypevar	C	typed variables
defcv	CLOS	general classes
defvar	CLOS	classes
defivar	CLOS	instances
defop	CLOS	generic functions
defmethod	CLOS	methods
deftp	All	data types
defopt	All	User Options

Table 10.1: The Definition Commands

In the Info file, a definition causes the category entity—'Function', 'Variable', or whatever—

to appear at the beginning of the first line of the definition, followed by the entity's name and arguments. In the printed manual, the command causes LaTeX to print the entity's name and its arguments on the left margin and print the category next to the right margin. In both output formats, the body of the definition is indented.

The name of the entity is entered into the appropriate index: `\deffn` enters the name into the index of functions, `\defvr` enters it into the index of variables, and so on.

As these functions are not always wanted, their definitions are contained in the LaTeXinfo style `elisp`. To make these commands available to LaTeXinfo, include the `elisp` option in the list of `documentstyle` options, such as

```
\documentstyle[latexinfo,elisp]{book}
```

Note: The Lisp documentation functions in the `elisp` style are compatible with the Emacs TeXinfo fuunctions, and are intended to document the GNU Emacs `elisp`. As such, they are oriented to the older `Maclisp` style of programming. See section 18.1.3 [Clisp Style], page 154, for a more modern approach to a Lisp documentation style, as would be used for Common Lisp.

10.1 Untyped Languages Definition Commands

10.1.1 The Template for a Definition

The `\deffn` command is used for definitions of entities that resemble functions. To write a definition using the `\deffn` command, write the `\deffn` command at the beginning of a line and follow it by the category of the entity, the name of the entity itself, and its arguments in braces. Then write the body of the definition on succeeding lines. (You may embed examples in the body.) Finally, end the definition with an `\enddeffn` command written on a line of its own. The other definition commands follow the same format. The template for a definition looks like this:

```
\deffn{category}{name}{arguments...}
body-of-definition
\enddeffn
```

For example,

```
\deffn{Command}{forward-word}{count}
This command moves point forward  \var{count} words
(or backward if  \var{count} is negative). ...
\enddeffn
```

produces

`forward-word` *count* Command

This function moves point forward *count* words (or backward if *count* is negative). ...

Some of the definition commands are more general than others. The `\deffn` command, for example, is the general definition command for functions and the like—for entities that may take arguments. When you use this command, you specify the category to which the entity belongs. The `\deffn` command possesses three predefined, specialized variations, `\defun`, `\defmac`, and `\defspec`, that specify the category for you: "Function", "Macro", and "Special Form" respectively. The `\defvr` command also is accompanied by several predefined, specialized variations for describing particular kinds of variables.

The template for a specialized definition, such as `\defun`, is similar to the template for a generalized definition, except that you don't have to specify the category:

```
\defun{name}{arguments...}
body-of-definition
\enddefun
```

Thus,

```
\defun{buffer-end}{flag}
This function returns  \code{(point-min)} if  \var{flag}
is less than 1,  \code{(point-max)} otherwise.
...
\enddefun
```

produces

`buffer-end` *flag* Function

This function returns `(point-min)` if *flag* is less than 1, `(point-max)` otherwise.
...

See section Sample Function Definition in A Sample Function Definition, for a more detailed example of a function definition, including the use of `\begin{example}` inside of the definition.

The other specialized commands work like `\defun`.

10.1.2 Optional and Repeated Parameters

Some entities take optional or repeated parameters, which may be specified by a distinctive special glyph that uses square brackets and ellipses. For example, a special form often breaks its argument list into separate arguments in more complicated ways than a straightforward function.

An argument enclosed within square brackets is optional. Thus, the phrase '[*optional-arg*]' means that *optional-arg* is optional. An argument followed by an ellipsis is optional and may be repeated more than once. Thus, '*repeated-args*...' stands for zero or more arguments. Parentheses are used when several arguments are grouped into additional levels of list structure in Lisp. Here is the `\defspec` line of an example of an imaginary special form:

`foobar` *var [from to [inc]]* Special Form
body...

In this example, the arguments *from* and *to* are optional, but must both be present or both absent. If they are present, *inc* may optionally be specified as well. In a LaTeXinfo source file, this `\defspec` line is written like this:

```
\defspec{foobar}{ \var{var} [ \var{from}  \var{to}
     [ \var{inc}]]}
\var{body} \dots{}
\enddefspec
```

The function is listed in the Command and Variable Index under '`foobar`'.

10.1.3 The Definition Commands

The definition commands automatically enter the name of the entity in the appropriate index: for example, `\deffn`, `\defun`, and `\defmac` enter function names in the index of functions; `\defvr` and `\defvar` enter variable names in the index of variables. Remember to declare the necessary indices with the `\newindex` commands (see section 2.4.3 [New Indexes], page 19).

Although the examples that follow mostly illustrate Lisp, the commands can be used for other programming languages.

10.1.4 Functions

This section describes the commands for describing functions and similar entities.

\deffn{*category*}{*name*}{*arguments...*} The `\deffn` command is the general definition command for functions, interactive commands, that may take arguments. You must choose a term to describe the category of entity being defined; for example, "Function" could be used if the entity is a function. The `\deffn` command is written at the beginning of a line and is followed by the category of entity being described, the name of this particular entity, and its arguments, if any. Terminate the definition with `\enddeffn` on a line of its own.

For example,

```
\deffn{Command}{forward-char}{nchars}
Move point forward  \var{nchars} characters.
\enddeffn
```

shows a rather terse definition for a "command" named `forward-char` with one argument, *nchars*.

`\deffn` prints argument names such as *nchars* in italics or upper case, as if `\var` had been used, because we think of these names as metasyntactic variables—they stand for the actual argument values. Within the text of the description, write an argument name explicitly with `\var` to refer to the value of the argument. In the example above, we used '`\var{nchars}`' in this way. The template for `\deffn` is:

```
\deffn{category}{name}{arguments...}
body-of-definition
\enddeffn
```

\defun{*name*}{*arguments...*} The `\defun` command is the definition command for functions. `\defun` is equivalent to '`\deffn{Function} ...`'.

For example,

```
\defun{set}{symbol new-value}
Change the value of the symbol symbol to new-value.
\enddefun
```

shows a rather terse definition for a function `set` whose arguments are *symbol* and *new-value*. The argument names on the `\defun` line automatically appear in italics or upper case as if they were enclosed in `\var`. Terminate the definition with `\enddefun` on a line of its own. The template is:

```
\defun{function-name}{arguments...}
body-of-definition
\enddefun
```

`\defun` creates an entry in the index of functions.

\defmac{*name*}{*arguments...*} The `\defmac` command is the definition command for macros. `\defmac` is equivalent to '`\deffn{Macro}...`' and works like `\defun`.

\defspec{*name*}{*arguments...*} The `\defspec` command is the definition command for special forms. `\defspec` is equivalent to '`\deffn{Special Form} ...`' and works like `\defun`.

10.1.5 Variables

Here are the commands for defining variables and similar entities:

\defvr{*category*}{*name*} The `\defvr` command is a general definition command for something like a variable—an entity that records a value. You must choose a term to describe the category of entity being defined; for example, "Variable" could be used if the entity is a variable. Write the `\defvr` command at the beginning of a line and follow it by the category of the entity and the name of the entity. Terminate the definition with `\enddefvr` on a line of its own. For example:

```
\defvr{User Option}{fill-column}
This buffer-local variable specifies
the maximum width of filled lines.
...
\enddefvr
```

The template is:

```
\defvr{category}{name}
body-of-definition
\enddefvr
```

`\defvr` creates an entry in the index of variables for *name*.

\defvar{*name***}** The `\defvar` command is the definition command for variables. `\defvar` is equivalent to '`\defvr{Variable}...`'. For example,

```
\defvar{kill-ring}
...
\enddefvar
```

The template is:

```
\defvar{name}
body-of-definition
\enddefvar
```

`\defvar` creates an entry in the index of variables for *name*.

10.2 C Functions

10.2.1 Functions in Typed Languages

The `\deftypefn` command and its variations are for describing functions in C or any other language in which you must declare types of variables and functions.

\deftypefn{*category***}{***data-type***}{***name***}{***arguments...***}** The `\deftypefn` command is the general definition command for functions that may take arguments and that are typed. The `\deftypefn` command is written at the beginning of a line and is followed the category of entity being described, the type of the returned value, the name of this particular entity, and its arguments, if any.

For example,

```
\deftypefn{Library Function}{int}{foobar}{(int  \var{foo}, float  \var{bar})}
...
\enddeftypefn
```

produces the following in Info:

```
-- Library Function: int foobar (int FOO, float BAR)
...
```

In a printed manual, it produces:

`int foobar (int` *foo*`, float` *bar*`)` Library Function
a "library function" that returns an `int`

This means that `foobar` is a "library function" that returns an `int`, and its arguments are *foo* (an `int`) and *bar* (a `float`).

The argument names that you write in `\deftypefn` are not subject to an implicit `\var`—since the actual names of the arguments in `\deftypefn` are typically scattered among data type names and keywords, LaTeXinfo can't find them without help. Instead, you must write `\var` explicitly around the argument names. In the example above, the argument names are '`foo`' and '`bar`'.

The template for `\deftypefn` is:

```
\deftypefn{category}{data-type}{name}{arguments} ...
body-of-description
\enddeftypefn
```

Note that if the *category* or *data type* is more than one word then it must be enclosed in braces to make it a single argument.

If you are describing a procedure in a language that has packages, such as Ada, you might consider using `\deftypefn` in a manner somewhat contrary to the convention described in the preceding paragraphs. For example:

```
\deftypefn{stacks}{private}{push}
{( \var{s}:in out stack;  \var{n}:in integer)}
...
\enddeftypefn
```

In this instance, the procedure is classified as belonging to the package `stacks` rather than classified as a 'procedure' and its data type is described as `private`. (The name of the procedure is `push`, and its arguments are *s* and *n*.)

`\deftypefn` creates an entry in the index of functions for *name*.

`\deftypefun``{`*data-type*`}{`*name*`}{`*arguments*`...}` The `\deftypefun` command is the specialized definition command for functions in typed languages. The command is equivalent to '`\deftypefn{Function}...`'.

```
\deftypefun{int}{foobar}
{(int  \var{foo}, float  \var{bar})}
...
\enddeftypefun
```

produces the following in Info:

```
-- Function: int foobar (int FOO, float BAR)
...
```

and the following in a printed manual:

`int foobar (int` *foo*`, float` *bar*`)` Function

...

The template is:

`\deftypefun{`*type*`}{`*name*`}{`*arguments...*`}`
body-of-description
`\enddeftypefun`

`\deftypefun` creates an entry in the index of functions for *name*.

10.2.2 Variables in Typed Languages

Variables in typed languages are handled in a manner similar to functions in typed languages. (See section 10.2.1 [Typed Functions], page 82.) The general definition command `\deftypevr` corresponds to `\deftypefn` and the specialized definition command `\deftypevar` corresponds to `\deftypefun`.

`\deftypevr``{`*category*`}{`*data-type*`}{`*name*`}` The `\deftypevr` command is the general definition command for something like a variable in a typed language—an entity that records a value. You must choose a term to describe the category of the entity being defined; for example, "Variable" could be used if the entity is a variable.

The `\deftypevr` command is written at the beginning of a line and is followed by the category of the entity being described, the data type, and the name of this particular entity. For example:

```
\deftypevr{Global Flag}{int}{enable}
...
\enddeftypevr
```

produces the following in Info:

```
-- Global Flag: int enable
...
```

and the following in a printed manual:

`int enable` Global Flag

...

The template is:

```
\deftypevr{category}{data-type}{name}
body-of-description
\enddeftypevr
```

`\deftypevr` creates an entry in the index of variables for *name*.

\deftypevar{*data-type*}{*name*} The `\deftypevar` command is the specialized definition command for variables in typed languages. `\deftypevar` is equivalent to '`\deftypevr{Variable}...`'.

For example,

```
\deftypevar{int}{foobar}
...
\enddeftypevar
```

produces the following in Info:

```
-- Variable: int foobar
...
```

and the following in a printed manual:

`int foobar` Variable

...

The template is:

```
\deftypevar{data-type}{name}
body-of-description
\enddeftypevar
```

`\deftypevar` creates an entry in the index of variables for *name*.

10.3 Object-Oriented Programming

LaTeXinfo has commands for formatting descriptions about abstract objects, such as are used in object-oriented programming. A class is a defined type of abstact object. An instance of a class is a particular object that has the type of the class. An instance variable is a variable that belongs to the class but for which each instance has its own value.

In a definition, if the name of a class is truly a name defined in the programming system for a class, then you should write an `\code` around it. Otherwise, it is printed in the usual text font.

\defcv{*category*}{*class*}{*name*} The `\defcv` command is the general definition command for variables associated with classes in object-oriented programming. The `\defcv` command is followed by three arguments: the category of thing being defined, the class to which it belongs, and its name. Thus,

```
\defcv{Class Option}{Window}{border-pattern}
...
\enddefcv
```

illustrates how you would write the first line of a definition of the `border-pattern` class option of the class `Window`. The template is:

```
\defcv{category}{class}{name}
...
\enddefcv
```

`\defcv` creates an entry in the index of variables.

\defivar{*class*}{*name*} The `\defivar` command is the definition command for instance variables in object-oriented programming. `\defivar` is equivalent to '`\defcv{Instance Variable}...`'

The template is:

```
\defivar{class}{instance-variable-name}
body-of-definition
\enddefivar
```

`\defivar` creates an entry in the index of variables.

\defop{*category*}{*class*}{*name*}{*arguments...*} The `\defop` command is the general definition command for entities that may resemble methods in object-oriented programming. These entities take arguments, as functions do, but are associated with particular classes of objects. For example, some systems have constructs called *wrappers* that are associated with classes as methods are, but that act more like macros than like functions. You could use `\defop{Wrapper}` to describe one of these.

Sometimes it is useful to distinguish methods and *operations*. You can think of an operation as the specification for a method. Thus, a window system might specify that all window classes have a method named `expose`; we would say that this window system defines an `expose` operation on windows in general. Typically, the operation has a name and also specifies the pattern of arguments; all methods that implement the operation must accept the same arguments, since applications that use the operation do so without knowing which method will implement it.

Often it makes more sense to document operations than methods. For example, window application developers need to know about the `expose` operation, but need not be concerned with whether a given class of windows has its own method to implement this operation. To describe this operation, you would write:

```
\defop{Operation}{windows}{expose}{}
```

The `\defop` command is written at the beginning of a line and is followed by the overall name of the category of operation, the name of the class of the operation, the name of the operation, and its arguments.

The template is:

```
\defop{category}{class}{name}{arguments...}
body-of-definition
\enddefop
```

`\defop` creates an entry, such as '`expose` on `windows`', in the index of functions.

\defmethod{*class*}{*name*}{*arguments...*} The `\defmethod` command is the definition command for methods in object-oriented programming. A method is a kind of function that implements an operation for a particular class of objects and its subclasses.

`\defmethod` is equivalent to '`\defop{Method ...}`'. The command is written at the beginning of a line and is followed by the name of the class of the method, the name of the method, and its arguments, if any. For example,

```
\defmethod{bar-class}{bar-method}{argument}
...
\enddefmethod
```

illustrates the definition for a method called `bar-method` of the class `bar-class`. The method takes an argument.

The template is:

```
\defmethod{class}{method-name}{arguments...}
body-of-definition
\enddefmethod
```

`\defmethod` creates an entry, such as '`bar-method` on `bar-class`', in the index of functions.

10.3.1 Data Types

Here is the command for data types:

\deftp{*category*}{*name*}{*attributes...*} The `\deftp` command is the generic definition command for data types. The command is written at the beginning of a line and is followed by the category, by the name of the type (which is a word like `int` or `float`, and then by names of attributes of objects of that type. Thus, you could use this command for describing `int` or `float`, in which case you could use `data type` as the category. (A data type is a category of certain objects for purposes of deciding which operations can be performed on them.)

In Lisp, for example, *pair* names a particular data type, and an object of that type has two slots called the CAR and the CDR. Here is how you would write the first line of a definition of `pair`.

```
\deftp{Data type}{pair}{car cdr}
...
\enddeftp
```

The template is:

```
\deftp{category}{name-of-type}{attributes...}
body-of-definition
\enddeftp
```

`\deftp` creates an entry in the index of data types.

\defopt{*name***}** The `\defopt` command is the definition command for user options. `\defopt` is equivalent to '`\defvr {User Option} ...`' and works like `\defvar`.

10.4 A Sample Function Definition

A function definition uses the `\defun` and `\enddefun` commands. The name of the function follows immediately after the `\defun` command and it is followed by the parameter list.

`apply` *function &rest arguments* Function

`apply` calls *function* with *arguments*, just like `funcall` but with one difference: the last of *arguments* is a list of arguments to give to *function*, rather than a single argument. We also say that this list is *appended* to the other arguments.

`apply` returns the result of calling *function*. As with `funcall`, *function* must either be a Lisp function or a primitive function; special forms and macros do not make sense in `apply`.

```
(setq f 'list)
     ⇒ list
(apply f 'x 'y 'z)
error  Wrong type argument: listp, z
(apply '+ 1 2 '(3 4))
     ⇒ 10
(apply '+ '(1 2 3 4))
     ⇒ 10

(apply 'append '((a b c) nil (x y z) nil))
     ⇒ (a b c x y z)
```

An interesting example of using `apply` is found in the description of `mapcar`.

In the LaTeXinfo source file, this example looks like this:

```
\defun{apply}{function &rest arguments}
\code{apply} calls \var{function} with \var{arguments}, just like
\code{funcall} but with one difference: the last of \var{arguments} is a
list of arguments to give to \var{function}, rather than a single
argument.  We also say that this list is \dfn{appended} to the other
arguments.

\code{apply} returns the result of calling \var{function}.  As with
\code{funcall}, \var{function} must either be a Lisp function or a
primitive function; special forms and macros do not make sense in
\code{apply}.
\begin{example}
(setq f 'list)
     \result{} list
(apply f 'x 'y 'z)
\error{} Wrong type argument: listp, z
(apply '+ 1 2 '(3 4))
     \result{} 10
(apply '+ '(1 2 3 4))
     \result{} 10

(apply 'append '((a b c) nil (x y z) nil))
     \result{} (a b c x y z)
```

In this manual, this function is listed in the Command and Variable Index under `apply`.

Ordinary variables and user options are described using a format like that for functions except that variables do not take arguments.

Part II

Info

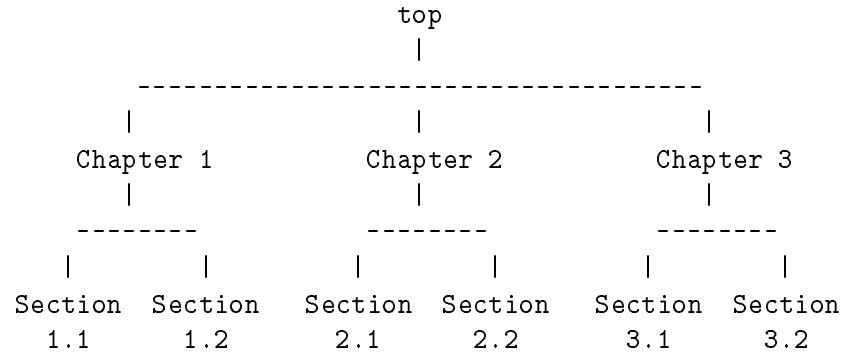
Chapter 11

Nodes and Menus

Most LaTeXinfo files are organized hierarchically like books, with chapters, sections, subsections, and subsubsections. Such a hierarchy is tree-like; the chapters are the major limbs from which the sections branch out. In a conventional diagram, however, such a hierarchy is drawn with the "root" at the top and the "leaves" at the bottom—as an upside-down tree. The root node is called the 'Top' node, and 'Up' pointers carry you closer to the root.

11.1 Node and Menu Illustration

Here is a copy of the diagram shown earlier that illustrates a LaTeXinfo file with three chapters, each of which contains two sections.



In a LaTeXinfo file that has this organization, you would write the beginning of the node for Chapter 2 like this:

```
\node     Chapter 2,  Chapter 3, Chapter 1, top
\comment  node-name,  next,      previous,  up
```

To go to Sections 2.1 and 2.2 using Info, you need a menu inside of Chapter 2 that says:

```
\begin{menu}
* Sect. 2.1::    Description of this section.
* Sect. 2.2::
\end{menu}
```

You would locate this menu inside Chapter 2, after the beginning of the chapter and before Section 2.1.

The node for Sect. 2.1 will look like this:

```
\node     Sect. 2.1, Sect. 2.2, Chapter 2, Chapter 2
\comment  node-name, next,      previous,  up
```

Usually, an `\node` command and a chapter structuring command are used in sequence, along with indexing commands. (The updating commands require this sequence. See section 15.3.1 [Updating Requirements], page 132.) Also, you may want to follow the `\node` line with a comment line that reminds you which pointer is which. For example, the beginning of the node for the chapter on ending a file looks like this:

```
\node    Ending a File, Structuring, Beginning a File, Top
\comment node-name,     next,        previous,         up
\chapter{Ending a LaTeXinfo File}
\cindex{Ending a LaTeXinfo file}
\cindex{LaTeXinfo file ending}
\cindex{File ending}
```

The following two sections describe the `\node` and `\begin{menu}` commands in detail.

11.2 \node

`\node` defines the beginning of a new node in the Info output file. (See Info file '`info`', node 'Top'.) Write the command at the beginning of a line, followed by four arguments, separated by commas, that make up the rest of the line. These arguments are the name of the node, and the names of the 'Next', 'Previous', and 'Up' pointers, in that order. You may insert spaces before each pointer if you wish. The spaces are ignored.

In LaTeX, `\node` is nearly ignored. It generates nothing visible. Its only function is to identify the name to use for cross references to the chapter or section which follows the `\node` command and which makes up the body of the node. (Cross references, such as the one following this sentence, are made with `\xref` and its related commands. See section 12 [Cross References], page 101.)

In general, an `\node` line is followed immediately by a chapter-structuring command such as `\chapter`, `\section`, `\subsection`, or `\subsubsection`. (See section Structuring Command Types in Types of Structuring Command.)

The name of the node identifies the node. The pointers, which enable you to reach other nodes, consist of the names of those nodes.

All the node names for a single Info file must be unique. Duplications confuse the Info movement commands. This means, for example, that if you end each chapter with a summary, you must name every summary node differently. You may, however, duplicate section titles (although this practice may confuse a reader).

Try to pick node names that are informative but short. In the Info file, the file name, node name, and pointer names are all inserted on one line, which may run into the right edge of the window. (This does not cause a problem with Info, but is ugly.)

By convention, node names are capitalized just as they would be for section or chapter titles.

> **Caution:** Do not use any of the LaTeXinfo \-commands in a node name; these commands confuse Info.
>
> Do not use commas within a node name; a comma terminates the node name.

Pointer names must be the names of nodes defined elsewhere. It does not matter whether pointers are before or after the node that refers to them.

Normally, a node's 'Up' pointer should contain the name of the node whose menu mentions that node. The node's 'Next' pointer should contain the name of the node that follows that node and its 'Previous' pointer should contain the name of the node that precedes it in that menu. When a node's 'Up' node is the same as its 'Previous' node, both node pointers should name the same node.

11.2.1 Writing a Node Line

The easiest way to write a node line is to write `\node` at the beginning of a line and then the name of the node. You can use update node commands provided by LaTeXinfo mode to insert the names of the pointers; see section 15 [LaTeXinfo Mode], page 127.

Alternatively, you may insert the 'Next', 'Previous', and 'Up' pointers yourself. If you do this, you may find it helpful to use the LaTeXinfo mode keyboard command `C-c C-c n`. This command inserts '`\node`' and a comment line listing the names of the pointers in their proper order. The comment line helps you keep track of which arguments are for which pointers. This template is especially useful if you are not familiar with LaTeXinfo.

If you wish, you can ignore node lines altogether in your first draft and then use the `latexinfo-insert-node-lines` command to create node lines for you. However, this practice is not recommended. It is better to name the node itself at the same time you write a section so you can easily make cross references. A large number of cross references are an especially important feature of a good Info file.

After you have inserted a node line, you should immediately write an `\`-command for the chapter or section and insert its name. Next (and this is important!), put in several index entries. Usually, you will find at least two and often as many as four or five ways of referring to the node in the index. Use them all. This will make it much easier for people to find the node.

The top node of the file (which must be named '`top`' or '`Top`') should have as its 'Up' and 'Previous' nodes the name of a node in another file, where there is a menu that leads to this file. Specify the file name in parentheses. If the file is to be installed directly in the Info directory file, use '`(dir)`' as the parent of the 'Top' node; this is short for '`(dir)top`', and specifies the 'Top' node in the '`dir`' file, which contains the main menu for Info. For example, the 'Top' node line of this manual looks like this:

```
\node Top, Overview, (dir), (dir)
```

(You may use the LaTeXinfo updating commands to insert these 'Next' and '`(dir)`' pointers automatically.)

See section 14.2 [Installing an Info File], page 121, for more information about installing an Info file in the '`info`' directory.

11.3 Menu Environment

The `\begin{menu}` command is used to create *menus*, which contain pointers to subordinate nodes. In Info, you use menus to go to such nodes. Menus have no effect in printed manuals and do not appear in them.

By convention, a menu is put at the end of a node. This way, it is easy for someone using Info to find the menu, using the `M->` (`end-of-buffer`) command.

A node that has a menu should not contain much text. If you have a lot of text and a menu, move most of the text into a new subnode—all but a few lines. Otherwise, a reader with a terminal that displays only a few lines may miss the menu and its associated text. As a practical matter, you should locate a menu within 20 lines of the beginning of the node.

The short text before a menu may look awkward in a printed manual. To avoid this, you can write a menu near the beginning of its node and follow the menu by an `\node` line and an `\section*` line within `\begin{ifinfo}` and `\end{ifinfo}`. This way, the menu, node line, and title appear only in the Info file, not the printed document.

The preceding two paragraphs follow an Info-only menu, node line, and heading, and look like this:

```
\begin{menu}
* Menu Location::               Put a menu in a short node.
* Menu Item::                   How to write a menu item.
* Menu Example::                A menu example.
\end{menu}

\node Menu Location
\begin{ifinfo}
\subsection*{Menus Need Short Nodes}
\end{ifinfo}
```

See the beginning of the "Cross References" chapter in the LaTeXinfo source for this document for another example this procedure.

11.3.1 Writing a Menu Item

In a menu, every line that begins with a '* ' is a menu item. (Note the space after the asterisk.) A line that does not start with a '* ' can appear in the menu but is not a menu item, just a comment.

A menu item has three parts, only the second of which is required:

1. The menu item name.
2. The name of the node.
3. A description of the item.

A menu item looks like this:

```
* Item name: Node name.      Description.
```

Follow the menu item name with a single colon and follow the node name with tab, comma, period, or newline.

In Info, a user can select a node with the `m` (`Info-menu`) command. The menu item name is what the user types after the `m` command.

If the menu item name and the node name are the same, you can write the name immediately after the asterisk and space at the beginning of the line and follow the name with two colons.

For example, write

```
* Name::
```

instead of

```
* Name: Name.
```

You should use the node name for the menu item name whenever possible, since it reduces visual clutter in the menu.

The third part of a menu entry is a short descriptive phrase or sentence. Menu item names and node names are often short; the description explains to the reader what the node is about. The description, which is optional, can spread over two or more lines. A useful description complements the node name rather than repeating it.

11.3.2 A Menu Example

A menu looks like this in LaTeXinfo:

```
\begin{menu}
* Menu item name: Node name.    A short description.
* Node name::                   This form is preferred.
\end{menu}
```

This produces:

```
* menu:

* Menu item name: Node name.    A short description.
* Node name::                   This form is preferred.
```

Here is an example as you might see it in a LaTeXinfo file:

```
\begin{menu}
Larger Units of Text

* Files::                 All about handling files.
* Multiples: Buffers.     Multiple buffers; editing
                            several files at once.
\end{menu}
```

This produces:

```
* menu:
Larger Units of Text

* Files::                 All about handling files.
* Multiples: Buffers.     Multiple buffers; editing
                            several files at once.
```

In this example, the menu has two entries. '`Files`' is both a menu item name and the name of the node referred to by that item. In the other entry, '`Multiples`' is the item name, and it refers to the node named '`Buffers`'.

Since no file name is specified with either '`Files`' or '`Buffers`', they must be the names of nodes in the same Info file. (See section Other Info Files in Referring to Other Info Files.)

The line '`Larger Units of Text`' is a comment.

11.4 Referring to Other Info Files

You can refer to nodes in other Info files by writing the file name in parentheses just before the node name. In this case, you should use the three-part menu item format, which saves the reader from having to type the file name.

If you do not list the node name, but only name the file, then Info presumes that you are referring to the '`Top`' node.

The format looks like this:

```
\begin{menu}
* first-item: (filename) nodename.          description
* second-item: (filename) second-node.      description
\end{menu}
```

The '`dir`' top level directory for the Info system has menu entries that take you directly to the '`Top`' nodes of each Info document. (See section 14 [Creating and Installing an Info File], page 119.)

For example,

```
...
* Info: (info).          Documentation browsing system.
* Emacs: (emacs).        The extensible, self-documenting
                         text editor.
...
```

To refer directly to the '`Outlining`' and '`Rebinding`' nodes in the *Emacs Manual*, you would write a menu similar to the following:

```
\begin{menu}
* Outlining: (emacs)Outline Mode. The major mode for
                                  editing outlines.
* Rebinding: (emacs)Rebinding.    How to redefine the
                                  meaning of a key.
\end{menu}
```

Chapter 12

Making Cross References

Cross references are used to refer the reader to other parts of the same or different LaTeXinfo files. In LaTeXinfo, nodes are the points to which cross references can refer.

Often, but not always, a printed document should be designed so that it can be read sequentially. People tire of flipping back and forth to find information that should be presented to them as they need it.

However, in any document, some information will be too detailed for the current context, or incidental to it; use cross references to provide access to such information. Also, an on-line help system or a reference manual is not like a novel; few read such documents in sequence from beginning to end. Instead, people look up what they need. For this reason, such creations should contain many cross references to help readers find other information that they may not have read.

In a printed manual, a cross reference creates a page reference, unless it is to another manual altogether, in which case it names that manual.

In Info, a cross reference creates an entry that you can follow using the Info '`f`' command. (See Info file '`info`', node 'Help-Adv'.)

The various cross reference commands use nodes to define cross reference locations. This is evident in Info, in which a cross reference takes you to the specified node. LaTeX also uses nodes to define cross reference locations, but the action is less obvious. When LaTeX generates a DVI file, it records nodes' page numbers and uses the page numbers in making references. Thus, if you are writing a manual that will only be printed, and will not be used on-line, you must nonetheless write `\node` lines to name the places to which you make cross references.

12.1 Different Cross Reference Commands

There are several different cross reference commands:

\xref Used to start a sentence in the printed manual saying 'See ...' or an entry in the Info file saying '`*Note` ...'.

\nxref Used within or, more often, at the end of a sentence; produces just the reference in the printed manual without a preceding 'See'. ('`n`' is for 'node'.)

\pxref Used within parentheses to make a reference that starts with a lower case 'see' within the printed manual. ('`p`' is for 'parenthesis'.)

\inforef Used to make a reference to an Info file. manual.

12.2 Parts of a Cross Reference

A cross reference command requires only one argument, which is the name of the node to which it refers. But a cross reference command may contain up to four additional arguments. By using these arguments, you can provide a menu item name for Info, a descriptive phrase for the printed output, the name of a different Info file, and the name of a different printed manual.

Here is a simple cross reference example:

```
\xref{Node name}.
```

which produces

```
*Note Node name::.
```

and in LaTeX, it turns into a sentence of the form

> See section *nnn* [Node name], page *ppp*.

Here, however, is an example of a full five-part cross reference:

```
\xref{Node name, Item name, Topic, info-file-name,
A Printed Manual}, for details.
```

which produces

```
*Note Item name: (info-file-name)Node name, for details.
```

and

> See section Topic of *A Printed Manual*, for details.

The five arguments for a cross reference are:

1. The node name (required). This is the node to which the cross reference takes you. In a printed document, the location of the node provides the page reference (for references within the same document).

2. The item name for the Info reference, if it is to be different from the node name. It is usually omitted.

3. A topic description or section name. Often, this is the title of the section. This is used as the name of the reference in the printed manual. If omitted, the node name is used.

4. The name of the Info file in which the reference is located, if it is different from the current file.

5. The name of another printed manual.

Cross references with one, two, three, four, and five arguments are described separately following the description of `\xref`.

You can write cross reference commands within a paragraph, but note how Info and LaTeX format the output of each of the various commands: write `\xref` at the beginning of a sentence; write `\pxref` only within parentheses, and so on.

12.3 \xref

The `\xref` command generates a cross reference for the beginning of a sentence. The Info formatting commands convert it into an Info cross reference, which the Info '`f`' command can use to bring you directly to another node. The LaTeX typesetting commands convert it into a page reference, or a reference to another book or manual.

Most often, an Info cross reference looks like this:

`*Note` *node-name*`::.`

or like this

`*Note` *item-name*`:` *node-name*`.`

In LaTeX, a cross reference looks like this:

`See section` *section* `[`*node-name*`], page` *page*

or like this

`See section` *section* `[`*topic*`], page` *page*

The `\xref` command does not generate a period or comma to end the cross reference in either the Info file or the printed output. You must write that period or comma yourself; otherwise, Info will not recognize the end of the reference. (The `\pxref` command works differently. See section pxref in `\pxref`.)

> *Please note:* A period or comma **must** follow the closing brace of an `\xref`. It is required to terminate the cross reference. This period or comma will appear in the output, both in the Info file and in the printed manual.

`\xref` must refer to an Info node by name. Use `\node` to define the node (see section 11.2.1 [Writing a Node], page 95).

`\xref` is followed by several arguments inside braces, separated by commas. Whitespace before and after these commas is ignored.

A cross reference requires only the name of a node; but it may contain up to four additional arguments. Each of these variations produces a cross reference that looks somewhat different.

12.3.1 `\xref` with One Argument

The simplest form of `\xref` takes one argument, the name of another node in the same Info file.
For example,

```
\xref{Tropical Storms}.
```

produces

```
*Note Tropical Storms::.
```

and

> See section *nnn* [Tropical Storms], page *ppp*.

(Note that in the preceding example the closing brace is followed by a period.)

You can write a clause after the cross reference, like this:

```
\xref{Tropical Storms}, for more info.
```

which produces:

```
*Note Tropical Storms::, for more info.
```

> See section *nnn* [Tropical Storms], page *ppp*, for more info.

(Note that in the preceding example the closing brace is followed by a comma, and then by the clause.)

12.3.2 `\xref` with Two Arguments

With two arguments, the second one is used as the name of the Info cross reference, while the first argument is still the node that the cross reference points to:
The template is like this:

`\xref` *node-name*`,` *item-name*.

For example:

```
\xref{Electrical Effects, Lightning}.
```

which produces:

```
*Note Lightning: Electrical Effects.
```

and

> See section *nnn* [Electrical Effects], page *ppp*.

(Note that in the preceding example the closing brace is followed by a period; and that the node name is printed, not the item name.)

You can write a clause after the cross reference, like this:

```
\xref{Electrical Effects, Lightning}, for more info.
```

produces

```
*Note Lightning: Electrical Effects, for more info.
```

and

> See section *nnn* [Electrical Effects], page *ppp*, for more info.

(Note that in the preceding example the closing brace is followed by a comma, and then by the clause.)

12.3.3 `\xref` with Three Arguments

A third argument replaces the node name in the LaTeX output. The third argument should state the topic discussed by the section being referenced, or be the name of the section. Often, you will want to use initial upper case letters so it will be easier to read when the reference is printed. Use a third argument when the node name is unsuitable because of syntax or meaning.

Remember that a comma or period must follow the closing brace of an `\xref` command to terminate the cross reference. In the following examples, a clause follows a terminating comma.
The template is like this:

`\xref` *node-name*, *item-name*, *topic*.

For example,

```
\xref{Electrical Effects, Lightning, Thunder and Lightning}, for details.
```

which produces

```
*Note Lightning: Electrical Effects, for details.
```

and

> See section *nnn* [Thunder and Lightning], page *ppp*, for details.

If a third argument is given and the second one is empty, then the third argument serves both. (Note how two commas, side by side, mark the empty second argument.)

```
\xref{Electrical Effects, , Thunder and Lightning}, for details.
```

produces

```
*Note Thunder and Lightning: Electrical Effects, for details.
```

and

> See section *nnn* [Thunder and Lightning], page *ppp*, for details.

12.3.4 `\xref` with Four and Five Arguments

In a cross reference, a fourth argument specifies the name of another Info file, different from the file in which the reference appears, and a fifth argument specifies its title as a printed manual.

Remember that a comma or period must follow the closing brace of an `\xref` command to terminate the cross reference. In the following examples, a clause follows a terminating comma. The template is:

`\xref{`*node-name*, *item-name*, *topic*, *info-file-name*, *printed-title*`}`.

For example,

```
\xref{Electrical Effects, Lightning, Thunder and Lightning,
weather, An Introduction to Meteorology}, for details.
```

which produces

```
*Note Lightning: (weather)Electrical Effects, for details.
```

The name of the Info file is enclosed in parentheses and precedes the name of the node. In a printed manual, the reference looks like this:

> See section Thunder and Lightning of *An Introduction to Meteorology*, for details.

The name of the printed manual is typeset in italics; and the reference lacks a page number since LaTeX cannot know to which page a refer refers when the reference is to another manual.

Often, you will leave out the second argument when you use the long version of `\xref`. In this case, the third argument, the topic description, will be used as the item name in Info. The template looks like this:

`\xref{`*node-name*`, , `*topic*`, `*info-file-name*`, `*printed-title*`}, for details.`

which produces

`*Note `*topic*`: (`*info-file-name*`)`*node-name*`, for details.`

and

> See section *topic* of *printed-manual-title*, for details.

For example:

```
\xref{Electrical Effects, , Thunder and Lightning,
weather, An Introduction to Meteorology}, for details.
```

which produces

```
*Note Thunder and Lightning: (weather)Electrical Effects, for details.
```

and

> See section Thunder and Lightning of *An Introduction to Meteorology*, for details.

On rare occasions, you may want to refer to another Info file that is is within a single printed manual—when multiple LaTeXinfo files are incorporated into the same LaTeX run but make separate Info files. In this case, you need to specify only the fourth argument, and not the fifth.

12.4 Naming a 'Top' Node

In a cross reference, you must always name a node. This means that in order to refer to a whole manual, you must identify the 'Top' node by writing it as the first argument to the `\xref` command. (This is different from the way you write a menu entry. See section Other Info Files in Referring to Other Info Files.) At the same time, to provide a meaningful section topic or title in the printed cross reference (instead of the word 'Top'), you must write an appropriate entry for the third argument to the `\xref` command.

Thus, to make a cross reference to *The GNU Make Manual*, write:

```
\xref{Top, , Overview, make, The GNU Make Manual}.
```

which produces

```
*Note Overview: (make)Top.
```

and

> See section Overview of *The GNU Make Manual*.

In this example, '`Top`' is the name of the node, and '`Overview`' is the name of the first section of the manual.

12.5 `\nxref`

`\nxref` is nearly the same as `\xref` except that it does not generate a 'See' in the printed output, just the reference itself. This makes it useful as the last part of a sentence.
For example:

```
For more information, see  \nxref{Orogenesis, ,
Mountaing Building}.
```

produces

```
For more information, see *Note Mountain
Building: Orogenesis.
```

and

> For more information, see section *nnn* [Mountain Building]. page *ppp*.

The `\nxref` command sometimes leads writers to express themselves in a manner that is suitable for a printed manual but looks awkward in the Info format. Bear in mind that your audience will be using both the printed and the Info format.
For example,

```
Sea surges are described in  \nxref{Hurricanes}.
```

produces

> Sea surges are described in section *nnn* [Hurricanes].

in a printed document, but

```
Sea surges are described in *Note Hurricanes::.
```

in Info.

> **Caution:** You *must* write a period or comma immediately after an `\nxref` command with two or more arguments. Otherwise, Info will not find the end of the cross reference entry and attempts to follow the cross reference will fail. As a general rule, you should write a period or comma after every `\nxref` command. This looks best in both the printed and the Info output.

12.6 `\pxref`

The parenthetical reference command, `\pxref`, is nearly the same as `\xref`, but you use it *only* inside parentheses and you do *not* type a comma or period after the command's closing brace. The command differs from `\xref` in two ways:

1. LaTeX typesets the reference for the printed manual with a lower case 'see' rather than an upper case 'See'.
2. The Info formatting commands automatically end the reference with a closing colon or period.

Because one type of formatting automatically inserts closing punctuation and the other does not, you should use `\pxref` *only* inside parentheses as part of another sentence. Also, you yourself should not insert punctuation after the reference, as you do with `\xref`.

`\pxref` is designed so that the output looks right and works right between parentheses both in printed output and in an Info file. In a printed manual, a closing comma or period should not follow a cross reference within parentheses; such punctuation is wrong. But in an Info file, suitable closing punctuation must follow the cross reference so Info can recognize its end. `\pxref` spares you the need to use complicated methods to put a terminator into one form of the output and not the other.

Don't try to use `\pxref` as a clause in a sentence. It will look bad in either the Info file, the printed output, or both. Use it only as a parenthetical reference.

With one argument, a parenthetical cross reference looks like this:

```
... large storms ( \pxref{Hurricanes}) may cause flooding
...
```

which produces

```
... large storms (*Note Hurricanes::) may cause flooding ...
```

and

... large storms (see section *nnn* [Hurricanes], page *ppp*) may cause flooding ...

With two arguments, a parenthetical cross reference has this template:

... (\pxref{*node-name*, *item-name*}) ...

which produces

... (*Note *item-name*: *node-name*.) ...

and

... (see section *nnn* [*node-name*], page *ppp*) ...

`\pxref` can be used with up to five arguments just like `\xref` (see section xref in `\xref`).

12.7 \inforef

`\inforef` is used for cross references to Info files for which there are no printed manuals. Even in a printed manual, `\inforef` generates a reference directing the user to look in an Info file.

The command takes either two or three arguments, in the following order:

1. The node name.
2. The item name (optional).
3. The Info file name.

Separate the arguments with commas, as with `\xref`. Also, you must terminate the reference with a comma or period after the '`}`', as you do with `\xref`.
The template is:

\inforef{*node-name*, *item-name*, *info-file-name*},

Thus,

```
\inforef{Expert, Advanced Info commands, info},
for more information.
```

produces

```
*Note Advanced Info commands: (info)Expert,
for more information.
```

and

> See Info file '`info`', node '`Expert`', for more information.

Similarly,

```
\inforef{Expert, , info}, for more information.
```

produces

```
*Note (info)Expert::, for more information.
```

and

> See Info file '`info`', node '`Expert`', for more information.

The converse of `\inforef` is `\cite`, which is used to refer to printed works for which no Info form exists. See section 8.2 [Citations], page 72.

Chapter 13

Creating Indices

Using LaTeXinfo, you can generate indices without having to sort and collate entries manually. In an index, the entries are listed in alphabetical order, together with information on how to find the discussion of each entry. In a printed manual, this information consists of page numbers. In an Info file, this information is a menu item leading to the first node referenced.

LaTeXinfo provides several predefined kinds of index: an index for functions, an index for variables, an index for concepts, and so on. You can combine indices or use them for other than their canonical purpose. If you wish, you can define your own indices.

13.1 Making Index Entries

When you are making index entries, it is good practice to think of the different ways people may look for something. Different people *do not* think of the same words when they look something up. A helpful index will have items indexed under all the different words that people may use. For example, someone might think it obvious that the two-letter names for indices should be listed under "Indices, two-letter names", since the word "Index" is the general concept. But another reader may remember the specific concept of two-letter names and search for the entry listed as "Two letter names for indices". A good index will have both entries and will help both kinds of user.

Like typesetting, the construction of an index is a highly skilled, professional art, the subtleties of which are not appreciated until you have to do it yourself.

See section 2.9.2 [Printing an Index and Generating Menus], page 28, for information about the commands to put at the beginning and end of the file, for printing an index, or creating an index menu in an Info file.

LaTeXinfo provides six predefined indices:

- A *concept index* listing concepts that are discussed.
- A *function index* listing functions (such as, entry points of libraries).
- A *variables index* listing variables (such as, global variables of libraries).
- A *keystroke index* listing keyboard commands.
- A *program index* listing names of programs.
- A *data type index* listing data types (such as, structures defined in header files).

Not every manual needs all of these. This manual has two indices: a concept index and an \-command index (that is actually the function index but is called a command index in the chapter heading). Two or more indices can be combined into one using the `\synindex` or `\syncodeindex` commands. See section 13.3 [Combining Indices], page 116.

13.2 Defining the Entries of an Index

The data to make an index comes from many individual indexing commands scattered throughout the LaTeXinfo source file. Each command says to add one entry to a particular index; after processing, it will give the current page number or node name as the reference.An index entry consists of an indexing command at the beginning of a line followed by the entry in braces. For example, this section begins with the following five entries for the concept index:

```
\cindex{Defining indexing entries}
\cindex{Index entries}
\cindex{Entries for an index}
\cindex{Specifying index entries}
\cindex{Creating index entries}
```

Each declared index has its own indexing command—`\cindex` for the concept index, `\findex` for the function index, and so on. An index must be declared at the beginning of the document with the `\newindex` command, before the first use of the corresponding index command. See section 13.2.1 [Declaring indices], page 115 for how to use this command.

The usual convention is to capitalize the first word of each index entry, unless that word is the name of a function, variable, or other such entitity that should not be capitalized. Thus, if you are documenting Emacs Lisp, your concept index entries are usually capitalized, but not your function index entries. However, if your concept index entries are consistently short (one or two words each) it may look better for each regular entry to start with a lower case letter. Which ever convention you adapt, please be consistent!

By default, entries for a concept index are printed in a small roman font and entries for the other indices are printed in a small `\code` font. You may change the way part of an entry

is printed with the usual LaTeXinfo commands, such as `\file` for file names and `\emph` for emphasis (see section 4 [Marking Text], page 35).

The six indexing commands for predefined indices are:

\cindex{*concept* Make an entry in the concept index for *concept*.

\findex{*function* Make an entry in the function index for *function*.

\vindex{*variable* Make an entry in the variable index for *variable*.

\kindex{*key* Make an entry in the key index for *key*.

\pindex{*program* Make an entry in the program index for *program*.

\tindex{*data***}** *type* Make an entry in the data type index for *data type*.

> **Caution:** Do not use a colon in an index entry. In Info, a colon separates the menu item name from the node name. An extra colon confuses Info. See section Menu Item in Writing a Menu Item, for more information about the structure of a menu entry.

If the same name is indexed on several pages, all the pages are listed in the printed manual's index. However, **only** the **first** node referenced will appear in the index of an Info file. This means that it is best to write indices in which each entry will refer to only one place in the LaTeXinfo file.

13.2.1 Declaring Indices

The `\newindex` command takes a two-letter index name, and makes the index commands for that index available for use. The `\printindex` command takes a two-letter index name, reads the corresponding sorted index file and formats it appropriately into an index. Normally, six indices are provided for, and are referred to by their two-letter abbreviations:

cp A *concept index* listing concepts that are discussed.

pg A *program index* listing names of programs and leading to the places where those programs are documented.

fn A *function index* listing functions (such as, entry points of libraries).

vr A *variables index* listing variables (such as, external variables of libraries).

tp A *data type index* listing data types (such as, structures defined in header files).

ky A *keystroke index* listing keyboard commands.

Not every manual needs all of these. This manual has two indices: a concept index and a command index (that uses the function index but is called a command index in the chapter heading). Two or more indices can be combined into one using the `\synindex` command. See section 13.3 [Combining Indices], page 116.

You are not actually required to use the predefined indices for their canonical purposes. For example, suppose you wish to index some C preprocessor macros. You could put them in the function index along with actual functions, just by writing `\findex` commands for them; then, when you print the "function index" as an unnumbered chapter, you could give it the title 'Function and Macro Index' and all will be consistent for the reader. Or you could put the macros in with the data types by writing `\tindex` commands for them, and give that index a suitable title so the reader will understand. (See section 2.9.2 [Printing an Index and Generating Menus], page 28.)

13.2.2 Special Index Entries

The concept index has two special index entries to help you make more elaborate concept indices.

`\cpsubindex{topic}{subtopic}` defines an entry in the concept index, which has a subtopic. In the Info manual, this line and anything on it is deleted.

`\cpindexbold{topic}` defines an entry in the concept index, which is set in bold type. In the Info manual, this line and anything on it is deleted.

All other indices have just one special index, the `\??indexbold` command, which sets its entry in bold type.

13.3 Combining Indices

Sometimes you will want to combine two disparate indices such as functions and variables, perhaps because you have few enough of one of them that a separate index for them would look silly.

You could put functions into the concept index by writing `\cindex` commands for them instead of `\findex` commands, and produce a consistent manual by printing the concept index with the title 'Function and Concept Index' and not printing the 'Function Index' at all; but this is not a robust procedure. It works only if your document is never included in part of or together with another document that is designed to have a separate function index; if your document were to be included with such a document, the functions from your document and those from the other would not end up together. Also, to make your function names appear in the right font in the concept index, you would have to enclose every one of them between `\code` and `\end{code}`.

What you should do instead when you want functions and concepts in one index is to index the functions with `\findex` as they should be, but use the `\syncodeindex` command to redirect these `\findex` commands to the concept index.

The `\syncodeindex` command takes two arguments: the name of an index to redirect, and the name of an index to redirect it to:

`\syncodeindex{`*from*`}{`*to*`}`

For this purpose, the indices are given two-letter names:

cp the concept index

vr the variable index

fn the function index

ky the key index

pg the program index

tp the data type index

Write an `\syncodeindex` command before or shortly after the end of header line at the beginning of a LaTeXinfo file. For example, to merge a function index with a concept index, write the following:

```
\syncodeindex{fn}{cp}
```

This will cause all entries designated for the function index to go to the concept index instead.

The `\syncodeindex` command puts all the entries from the redirected index into the `\code` font, overriding whatever default font is used by the index to which the entries are redirected. This way, if you redirect function names from a function index into a concept index, all the function names are printed the `\code` font as you would expect.

The `\synindex` command is nearly the same as the `\syncodeindex` command, except that it does not put the redirected index into the `\code` font, but puts it in the roman font.

See section 2.9.2 [Printing an Index and Generating Menus], page 28, for information about printing an index at the end of a book or creating an index menu in an Info file.

Chapter 14

Creating and Installing an Info File

14.1 Creating an Info file

In GNU Emacs, the way to create an Info file is to visit the file and invoke

```
M-x latexinfo-format-buffer
```

A new buffer is created and the Info file text is generated there. ↑`x` ↑`s` (`save-buffer`) will save it under the name specified in the `\setfilename` command. `latexinfo-format-region` and `latexinfo-format-buffer` are the two Emacs commands that you can also use for formatting. A LaTeXinfo file must possess an `\setfilename` line near its beginning, otherwise the formatting commands will fail.

For information on installing the Info file in the Info system, see section 14.2 [Installing an Info File], page 121.

14.1.1 The latexinfo-format Commands

In GNU Emacs in LaTeXinfo mode, you can format part or all of a LaTeXinfo file with the `latexinfo-format-region` command. This formats the current region and displays the formatted text in a temporary buffer called '`*Info Region*`'.

Similarly, you can format the whole file with the `latexinfo-format-buffer` command. This command creates a new buffer and generates the Info file in it. Typing `C-x C-s` will save the Info file under the name specified by the `\setfilename` line which must be near the beginning of the LaTeXinfo file. See section 15.4 [Info Formatting], page 134, for how to use the commands:

C-c C-e C-r (`latexinfo-format-region`) Format the current region for Info.

C-c C-e C-b (`latexinfo-format-buffer`) Format the current buffer for Info.

The `latexinfo-format-region` and `latexinfo-format-buffer` commands provide you with some error checking; and other functions provide you with further help in finding formatting errors. These procedures are described elsewhere, see section 17 [Catching Formatting Mistakes], page 145.

14.1.2 Tag Files and Split Files

If a LaTeXinfo file has more than 30,000 bytes, `latexinfo-format-buffer` automatically creates a *tag table* for its Info file. With a tag table, Info can jump to new nodes more quickly than it can otherwise.

In addition, if the LaTeXinfo file contains more than about 70,000 bytes, `latexinfo-format-buffer` splits the large Info file into shorter *indirect* subfiles of about 50,000 bytes each. Big files are split into smaller files so that Emacs does not have to make a large buffer to hold the whole of a large Info file; instead, Emacs allocates just enough memory for the small, split off file that is needed at the time. This way, Emacs avoids wasting memory when you run Info. (Before splitting was implemented, Info files were always kept short and *include* files were designed as a way to create a single, large printed manual out of the smaller Info files. See section 9.2 [Include Files], page 73, for more information. Include files are still used for very large documents, such as *The Emacs Lisp Reference Manual*, in which each chapter is a separate file.)

When a file is split, Info itself makes use of a shortened version of the original file that contains just the tag table and references to the files that were split off. The split off files are called *indirect* files.

The split off files have names that are created by appending '`-1`', '`-2`', '`-3`' and so on to the file names specified by the `\setfilename` command. The shortened version of the original file continues to have the name specified by `\setfilename`.

At one stage in writing a document, for example, the Info file called '`test-latexinfo`' might have looked like this:

```
Info file: test-latexinfo,    -*-Text-*-
produced by latexinfo-format-buffer
from file: new-manual.tex

^_
Indirect:
test-latexinfo-1: 102
test-latexinfo-2: 50422
test-latexinfo-3: 101300
^_^L
Tag table:
(Indirect)
Node: overview^?104
Node: info file^?1271
Node: printed manual^?4853
Node: conventions^?6855
...
```

Each of the split off, indirect files, '`test-latexinfo-1`', '`test-latexinfo-2`', and '`test-`

`latexinfo-3`', is listed in this file after the line that says '`Indirect:`'. The tag table is listed after the line that says '`Tag table:`'.

If you are using `latexinfo-format-buffer` to create Info files, you may want to run the `Info-validate` command. However, you cannot run the `M-x Info-validate` node-checking command on indirect files. For information on how to prevent files from being split and how to validate the structure of the nodes, see section 17.5.1 [Using Info-validate], page 149.

14.2 Installing an Info File

Info files are usually kept in the '`.../emacs/info`' directory. This directory is the values of the Emacs variable `Info-directory`.

14.2.1 The '`dir`' File

For Info to work, the '`info`' directory must contain a file that serves as a top level directory for the Info system. By convention, this file is called '`dir`'. The '`dir`' file is itself an Info file. It contains the top level menu for all the Info files in the system. The menu looks like this:

```
* Menu:

* Info:    (info).     Documentation browsing system.
* Emacs:   (emacs).    The extensible, self-documenting
                       text editor.
* LaTeXinfo: (latexinfo).  With one source file, make
                       either a printed manual using
                       LaTeX or an Info file.
...
```

Each of these menu entries points to the 'Top' node of the Info file that is named in parentheses. [1] Thus, the '`Info`' entry points to the 'Top' node of the '`info`' file and the '`Emacs`' entry points to the 'Top' node of the '`emacs`' file.

In each of the Info files, the 'Up' pointer of the 'Top' node refers back to the `dir` file. For example, the node line for the 'Top' node of the Emacs manual looks like this:

```
File: emacs  Node: Top, Up: (DIR), Next: Distrib
```

(Note that in this case, the file name is written in upper case letters—it can be written in either upper or lower case. Info has a feature that it will change the case of the file name to lower case if it cannot find the name as written.)

[1] The menu entry does not have to specify the 'Top' node, since Info goes to the 'Top' node if no node name is mentioned. See section Other Info Files in Nodes in Other Info Files.

14.2.2 Listing a New Info File

To add a new Info file to your system, add the name to the menu in the '`dir`' file by editing the '`dir`' file by hand. Also, put the new Info file in the '`.../emacs/info`' directory. For example, if you were adding documentation for GDB, you would make the following new entry:

```
* GDB: (gdb).           The source-level C debugger.
```

The first item is the menu item name; it is followed by a colon. The second item is the name of the Info file, in parentheses; it is followed by a period. The third part of the entry is the description of the item.

Conventionally, the name of an Info file has a '`.info`' extension. Thus, you might list the name of the file like this:

```
* GDB: (gdb.info).           The source-level C debugger.
```

However, Info will look for a file with a '`.info`' extension if it does not find the file under the name given in the menu. This means that you can write to '`gdb.info`' in a menu as '`gdb`', as shown in the first example. This looks better.

14.2.3 Info Files in Other Directories

If an Info file is not in the '`info`' directory, there are two ways to specify its location:

- Write the menu's second part as a pathname, or;
- Specify an environment variable in your '`.profile`' or '`.login`' initialization file.

For example, to reach a test file in the '`~bob/manuals`' directory, you could add an entry like this to the menu in the '`dir`' file:

```
* Test: (~bob/manuals/info-test).  Bob's own test file.
```

In this case, the absolute file name of the '`info-test`' file is written as the second item of the menu entry.

Alternatively, you may set the `INFOPATH` environment variable in your '`.login`' or '`.profile`' file. The `INFOPATH` environment variable will tell Info where to look.

If you use `sh` or `bash` for your shell command interpreter, you must set the `INFOPATH` environment variable in the '`.profile`' initialization file; but if you use `csh` or `tcsh`, you must set the variable in the '`.login`' initialization file. The two files require slightly different command formats.

- In a '`.login`' file, you could set the `INFOPATH` variable as follows:

```
setenv INFOPATH .:~bob/manuals:/usr/local/emacs/info
```

- In a '`.profile`' file, you would achieve the same effect by writing:

```
INFOPATH=.:~bob/manuals:/usr/local/emacs/info
export INFOPATH
```

Either form would cause Info to look first in the current directory, indicated by the '`.`', then in the '`~bob/manuals`' directory, and finally in the '`/usr/local/emacs/info`' directory (which is the usual location for the standard Info directory).

Part III

Emacs

Chapter 15

Using LaTeXinfo Mode

In GNU Emacs, LaTeXinfo mode provides commands and features especially designed for working with LaTeXinfo files. The special LaTeXinfo commands are in addition to the usual editing commands, which are generally the same as the commands of Text mode. There are special commands to:

- Insert commonly used strings of text.
- Automatically create node lines.
- Show the structure of a LaTeXinfo source file.
- Automatically create or update the 'Next', 'Previous', and 'Up' pointers of a node.
- Automatically create or update menus.
- Automatically create a master menu.
- Format a part or all of a file for Info.
- Typeset and print part or all of a file.

Implementation note: In LaTeXinfo mode the paragraph separation variable and syntax table are redefined so that LaTeXinfo commands that should be on lines of their own are not inadvertently included in paragraphs. Thus, the `M-q` (`fill-paragraph`) command will refill a paragraph but not mix an indexing command on a line adjacent to it into the paragraph.

In addition, LaTeXinfo mode sets the `page-delimiter` variable to the value of `latexinfo-chapter-level-regexp`; by default, this is a regular expression matching the commands for chapters and sections. With this value for the page delimiter, you can jump from chapter title to chapter title with the `C-x ]` (`forward-page`) and `C-x [` (`backward-page`) commands and narrow to a chapter with the `C-x p` (`narrow-to-page`) command. (See Info file '`emacs`', node 'Pages', for details about the page commands.)

You may name a LaTeXinfo file however you wish, but the convention is to end a LaTeXinfo file name with '`.tex`'. Emacs switches to LaTeXinfo mode for a file that has '`-*-latexinfo-*-`'

in its first line. If ever you are in another mode and wish to switch to LaTeXinfo mode, type `M-x latexinfo-mode`.

Like all other Emacs features, you can customize or enhance LaTeXinfo mode as you wish. In particular, the keybindings are very easy to change. The keybindings described here are the default or standard ones.

15.1 Inserting Frequently Used Commands

LaTeXinfo mode provides commands to insert various frequently used `\`-commands into the buffer. You can use these commands to save keystrokes.

The insert commands are invoked by typing `C-c` twice and then the first letter of the `\`-command. In the following description, we will list the key sequence, and then the name of the LaTeXinfo function that is invoked.

C-c C-c c (latexinfo-insert-code) Insert `\code{}` and put the cursor between the braces.

C-c C-c d (latexinfo-insert-dfn) Insert `\dfn{}` and put the cursor between the braces.

C-c C-c e (latexinfo-insert-end) Insert `\end`.

C-c C-c i (latexinfo-insert-item) Insert `\item` and put the cursor at the beginning of the next line.

C-c C-c k (latexinfo-insert-kbd) Insert `\kbd{}` and put the cursor between the braces.

C-c C-c n (latexinfo-insert-node) Insert `\node` and a comment line listing the sequence for the 'Next', 'Previous', and 'Up' nodes. Leave cursor after the `\node`.

C-c C-c o (latexinfo-insert-noindent) Insert `\noindent` and put the cursor in between.

C-c C-c s (latexinfo-insert-samp) Insert `\samp{}` and put the cursor between the braces.

C-c C-c v (latexinfo-insert-var) Insert `\var{}` and put the cursor between the braces.

C-c C-c x (latexinfo-insert-example) Insert `\begin{example} \end{example}` and put the cursor at the beginning of the next line.

C-c C-c { (latexinfo-insert-braces) Insert `{}` and put the cursor between the braces.

C-c C-c } (up-list) Move from between a set of braces forward past the closing brace.

Remark: This set of insert commands was created after analyzing the frequency with which different `\`-commands are used in the *GNU Emacs Manual* and the *GDB Manual*. If you wish to add your own insert commands, you can bind a keyboard macro to a key, use abbreviations, or extend the code in '`latexinfo-mde.el`'.

15.2 Showing the Section Structure of a File

You can show the section structure of a LaTeXinfo file by using the `C-c C-s` command (`latexinfo-show-structure`). This command shows the section structure of a LaTeXinfo file by listing the lines that begin with the `\`-commands for `\chapter`, `\section`, and the like. The command constructs what amounts to a table of contents. These lines are displayed in another buffer called the '`*Occur*`' buffer. In that buffer, you can position the cursor over one of the lines and use the `C-c C-c` command (`occur-mode-goto-occurrence`), to jump to the corresponding spot in the LaTeXinfo file.

C-c C-s (latexinfo-show-structure) Show the `\chapter`, `\section`, and such lines of a LaTeXinfo file.

C-c C-c (occur-mode-goto-occurrence) Go to the line in the LaTeXinfo file corresponding to the line under the cursor in the '`*Occur*`' buffer.

If you call `latexinfo-show-structure` with a prefix argument by typing `C-u C-c C-s`, it will list not only those lines with the `\`-commands for `\chapter`, `\section`, and the like, but also the `\node` lines. You can use `latexinfo-show-structure` with a prefix argument to inspect whether the 'Next', 'Previous', and 'Up' pointers of a node line are correct.

Often, when you are working on a manual, you will be interested only in the structure of the current chapter. In this case, you can mark off the region of the buffer that you are interested in with the `C-x n` (`narrow-to-region`) command and `latexinfo-show-structure` will work on only that region. To see the whole buffer again, use `C-x w` (`widen`). (See Info file '`emacs`', node 'Narrowing', for more information about the narrowing commands.)

In addition to providing the `latexinfo-show-structure` command, LaTeXinfo mode sets the value of the page delimiter variable to match the chapter-level `\`-commands. This enables you to use the `C-x ]` (`forward-page`) and `C-x [` (`backward-page`) commands to move forward and backward by chapter, and to use the `C-x p` (`narrow-to-page`) command to narrow to a chapter. See Info file '`emacs`', node 'Pages', for more information about the page commands.

See section 17.3 [Using latexinfo-show-structure], page 147, for how to detect formatting errors using this command.

15.3 Updating Nodes and Menus

LaTeXinfo mode provides commands for automatically creating or updating menus and node pointers. The commands are called "update" commands because their most frequent use is for updating a LaTeXinfo file after you have worked on it.

You can use the updating commands

- to insert or update the 'Next', 'Previous', and 'Up' pointers of a node,
- to insert or update the menu for a section, and

- to create a master menu for a LaTeXinfo source file.

You can also use the commands to update all the nodes and menus in a region or in a whole LaTeXinfo file.

LaTeXinfo mode has five updating commands that are used most often: two are for updating the node pointers or menu of a single node (or a region), two are for updating every node pointer and menu in a file, and one, the `latexinfo-master-menu` command, is for creating a master menu for a complete file, and optionally, for updating every node and menu in the whole LaTeXinfo file.

The `latexinfo-master-menu` command is the primary command:

C-c C-u m (latexinfo-master-menu) Create or update a master menu that includes all the other menus (incorporating the descriptions from pre-existing menus, if any).

With an argument (prefix argument, if interactive), first create or update all the nodes and all the regular menus in the buffer before constructing the master menu. (See section The Top Node in The Top Node and Master Menu, for more about a master menu.) For `latexinfo-master-menu` to work, the LaTeXinfo file must have a node called 'Top'.

After extensively editing a LaTeXinfo file, it is common to type `C-u C-c C-u m` or `C-u M-x latexinfo-master-menu` to update all the nodes and menus completely and all at once.

The other major updating commands do smaller jobs and are designed for the person who updates nodes and menus as he or she writes a LaTeXinfo file. These commands are:

C-c C-u C-n (latexinfo-update-node) Insert the 'Next', 'Previous', and 'Up' pointers for the node point is within (i.e., for the `\node` line preceding point). If the `\node` line has pre-existing 'Next', 'Previous', or 'Up' pointers in it, the old pointers are removed and new ones inserted. With an argument (prefix argument, if interactive), this command updates all `\node` lines in the region (which is the text between point and mark).

C-c C-u C-m (latexinfo-make-menu) Create or update the menu in the node that point is within. With an argument (prefix argument, if interactive), the command makes or updates menus for the nodes within or part of the region.

Whenever `latexinfo-make-menu` updates an existing menu, the descriptions from that menu are incorporated into the new menu. This is done by copying descriptions from the existing menu to the entries in the new menu that have the same node names. If the node names are different, the descriptions are not copied to the new menu.

Menu entries that refer to other Info files are removed since they do not refer to nodes within the current buffer. This is a deficiency.

C-c C-u C-e (latexinfo-every-node-update) Insert or update the 'Next', 'Previous', and 'Up' pointers for every node in the buffer .

C-c C-u C-a (latexinfo-all-menus-update) Create or update all the menus in the buffer. With an argument (prefix argument, if interactive), first insert or update all the node pointers before working on the menus.

If a master menu exists, the `latexinfo-all-menus-update` command updates it; but the command does not create a new master menu if none already exists. (Use the `latexinfo-master-menu` command for that.)

Implementation note: The `latexinfo-column-for-description` variable specifies the column to which menu descriptions are indented. By default, the value is 32 although it is often useful to reduce it to as low as 24. You can set the variable with the `M-x edit-options` command (See Info file '`emacs`', node 'Edit Options'), or with the `M-x set-variable` command (See Info file '`emacs`', node 'Examining').

Also, the `latexinfo-indent-menu-description` command may be used to indent existing menus to a specified column.Finally, if you wish, you can use the `latexinfo-insert-node-lines` command to insert missing `\node` lines into a file. (See section 15.3.2 [Other Updating Commands], page 132, for more information.)

15.3.1 Updating Requirements

To use the updating commands, you must organize the LaTeXinfo file hierarchically with chapters, sections, subsections, and the like. Each `\node` line, with the exception of the line for the 'Top' node, must be followed by a line with a structuring command such as `\chapter`, `\section`, or `\unnumberedsubsec`. Each `\node` line/structuring-command line combination must look either like this:

```
\node     Comments,  Minimum, Conventions, Overview
\comment  node-name, next,    previous,    up
\section{Comments}
```

or like this (without the `\comment` line):

```
\node Comments, Minimum, Conventions, Overview
\section{Comments}
```

(In this example, 'Comments' is the name of both the node and the section. The next node is called 'Minimum' and the previous node is called 'Conventions'. The 'Comments' section is within the 'Overview' node, which is specified by the 'Up' pointer.)

If a file has a 'Top' node, it must be called '`top`' or '`Top`' and be the first node in the file.

15.3.2 Other Updating Commands

In addition to the five major updating commands, LaTeXinfo mode possesses several less frequently used updating commands.

C-c C-u C-i (latexinfo-insert-node-lines) Insert `\node` before the `\chapter`, `\section`, and other sectioning commands wherever it is missing throughout a region in a LaTeXinfo file. With an argument (prefix argument, if interactive), the `latexinfo-insert-node-lines` command not only inserts `\node` lines but also inserts the chapter or section titles as the names of the corresponding nodes; and it inserts their titles for node names in pre-existing `\node` lines that lack names. Since node names should be more concise than section or chapter titles, node names so inserted should be edited manually. Also, section titles cannot contain commas if this command is used, or else only the title yp to the first comma will be used.

C-c C-u C-f (latexinfo-multiple-files-update) Update nodes and menus in a document built from several separate files. With a prefix argument if called interactively (a non-`nil` 'make-master-menu' argument, if called non-interactively), create and insert a master menu in the outer file. With a numeric prefix argument if called interactively (a non-`nil` 'update-everything' argument if called non-interactively), first update all the menus

and all the 'Next', 'Previous', and 'Up' pointers of all the included files before creating and inserting a master menu in the outer file. The `latexinfo-multiple-files-update` command is described in the section on `\include` files. See section 9.2 [Include Files], page 73.

C-c C-u C-d (latexinfo-indent-menu-description) Indent every description in the menu following point to the specified column. You can use this command to give yourself more space for descriptions. With an argument (prefix argument, if interactive), the `latexinfo-indent-menu-description` command indents every description in every menu in the region. However, this command does not indent the second and subsequent lines of a multi-line description.

C-c C-u C-s (latexinfo-sequential-node-update) Insert the names of the nodes immediately following and preceding the current node as the 'Next' or 'Previous' pointers regardless of those nodes' hierarchical level. This means that the 'Next' node of a subsection may well be the next chapter. Sequentially ordered nodes are useful for documents that you read through sequentially. (However, in Info, the `g* RET` command lets you look through the file sequentially, so sequentially ordered nodes are not strictly necessary.) With an argument (prefix argument, if interactive), the `latexinfo-sequential-node-update` command sequentially updates all the nodes in the region.

15.3.3 `latexinfo-multiple-files-update`

The `latexinfo-multiple-files-update` command creates or updates 'Next', 'Previous', and 'Up' pointers of included files as well as those in the outer or over all LaTeXinfo file, and it creates or updates a main menu in the outer file. See section 9.2 [Include Files], page 73. Depending whether you call it with optional arguments, it updates only the pointers in the first `\node` line of the included files or all of them.

C-u C-c C-u C-f (latexinfo-multiple-files-update) Called without any arguments, will:

- Create or update the 'Next', 'Previous', and 'Up' pointers of the first `\node` line in each file included in an outer or overall LaTeXinfo file.
- Create or update the 'Top' level node pointers of the outer or overall file.
- Create or update a main menu in the outer file.

C-u C-c C-u C-f (latexinfo-multiple-files-update) Called with a prefix argument (a non-`nil` *make-master-menu* argument, if called from a program), create and insert a master menu in the outer file in addition to creating or updating pointers in the first `\node` line in each included file and creating or updating the 'Top' level node pointers of the outer file. The master menu is made from all the menus in all the included files.

C-u 8 C-c C-u C-f (latexinfo-multiple-files-update) Called with a numeric prefix argument (a non-`nil` *update-everything* argument, if called from a program):

- Create or update the 'Top' level node pointers of the outer or overall file.
- Create or update *all* the 'Next', 'Previous', and 'Up' pointers of all the included files.
- Create or update *all* the menus of all the included files.
- And then create a master menu in the outer file. This is similar to invoking `latexinfo-master-menu` with an argument when you are working with just one file.

Note the use of the prefix argument in interactive use: with a regular prefix argument, just `C-u`, the `latexinfo-multiple-files-update` command inserts a master menu; with a numeric prefix argument, such as `C-u 8`, the command updates every pointer and menu in all the files and then inserts a master menu.

15.4 Formatting for Info

LaTeXinfo mode provides several commands for formatting part or all of a LaTeXinfo file for Info. Often, when you are writing a document, you want to format only part of a file—that is, a region. You can use the `latexinfo-format-region` command to format a region.

C-c C-e C-r (latexinfo-format-region) Format the current region for Info.

You can use the `latexinfo-format-buffer` command to format a whole buffer:

C-c C-e C-b (latexinfo-format-buffer) Format the current buffer for Info.

After writing a LaTeXinfo file, you can type `C-u C-c C-u m` or `C-u M-x latexinfo-master-menu` to update all the nodes and menus and then type `C-c C-u b` or `M-x latexinfo-format-buffer` to create an Info file.

For the Info formatting commands to work, the file *must* include a line that has `\setfilename` in its header. See section 14 [Creating and Installing an Info File], page 119, for details about Info formatting.

15.5 Formatting and Printing

Typesetting and printing a LaTeXinfo file is a multi-step process in which you first create a file for printing (called a DVI file), and then you print the file. Optionally, also, you may create indices.

Often, when you are writing a document, you want to typeset and print only part of a file, to see what it will look like. You can use the `latexinfo-latex-region` and related commands for this purpose. Use the `latexinfo-latex-buffer` command to format all of a buffer.

C-c C-t C-r (latexinfo-latex-region) Run LaTeX on the region.

C-c C-t C-b (latexinfo-latex-buffer) Run LaTeX on the buffer.

C-c C-t C-i (latexinfo-latexindex) Sort the indices of a LaTeXinfo file formatted with `latexinfo-latex-region` or `latexinfo-latex-buffer`. You must run the `latex` command a second time after sorting the raw index files.

C-c C-t C-p (latexinfo-latex-print) Print the file (or the part of the file) previously formatted with `latexinfo-latex-buffer` or `latexinfo-latex-region`.

For `latexinfo-latex-region` or `latexinfo-latex-buffer` to work, the file *must* start with a '`\documentstyle`' line and must include an `\setfilename` command as an end of header line. The file must end with `\end{document}` on a line by itself.

See section 16 [Printing Hardcopy], page 139, for a description of the other LaTeX related commands, such as `latexinfo-latexindex` and `latex-show-print-queue`.

15.6 LaTeXinfo Mode Summary

In LaTeXinfo mode, each set of commands has default keybindings that begin with the same keys. All the commands that are custom-created for LaTeXinfo mode begin with `C-c`. The keys that follow are arranged mnemonically.

Insert Commands The insert commands begin with `C-c` twice and then the first letter of the `\`-command to be inserted.

`C-c C-c c`	Insert ‘ `\code`’.
`C-c C-c d`	Insert ‘ `\dfn`’.
`C-c C-c e`	Insert ‘ `\end{}`’.
`C-c C-c i`	Insert ‘ `\item`’.
`C-c C-c n`	Insert ‘ `\node`’.
`C-c C-c s`	Insert ‘ `\samp`’.
`C-c C-c v`	Insert ‘ `\var`’.
`C-c C-c {`	Insert braces.
`C-c C-c }`	Move out of enclosing braces.

Show Structure `latexinfo-show-structure` is often used within a narrowed region.

`C-c C-s`	List all the headings.

The Master Update Command The `latexinfo-master-menu` command creates a master menu; and can be used to update every node and menu in a file as well.

`C-c C-u m`	Create or update a master menu.
`C-u C-c C-u m`	First create or update all nodes and regular menus.

Update Pointers The update pointer commands begin with `C-c C-u`:

`C-c C-u C-n`	Update a node.
`C-c C-u C-e`	Update every node in the buffer.

Update Menus The update menu commands begin with `C-c C-u`. You may precede a `C-c C-u C-a` so as to update both nodes and menus.

`C-c C-u C-m`	Make or update a menu.
`C-c C-u C-a`	Make or update all the menus in a buffer;
`C-u C-c C-u C-a`	first update all the nodes.

Format for Info The Info formatting commands begin with `C-c C-e`:

`C-c C-e C-r`	Format the region.
`C-c C-e C-b`	Format the buffer.

Typeset and Print The typesetting and printing commands begin with `C-c C-t`:

`C-c C-t C-r`	Run LaTeX on the region.
`C-c C-t C-b`	Run LaTeX on the buffer.
`C-c C-t C-i`	Run `latexindex`.
`C-c C-t C-p`	Print the DVI file.
`C-c C-t C-q`	Show the print queue.
`C-c C-t C-d`	Delete a job from the print queue.
`C-c C-t C-k`	Kill the current LaTeX formatting job.
`C-c C-t C-x`	Quit a currently stopped LaTeX formatting job.
`C-c C-t C-l`	Recenter the output buffer.

Other Updating Commands The 'other updating commands' begin with `C-c C-u`

`C-c C-u C-i`	Insert missing node lines using section titles as node names.
`C-c C-u C-f`	Update a multi-file document.
`C-c C-u C-d`	Indent descriptions.
`C-c C-u C-s`	Insert node pointers in strict sequence.

Chapter 16

Printing Hardcopy

The typesetting program LaTeX is used for formatting a LaTeXinfo file. LaTeX is a very powerful typesetting program and, if used correctly, does an exceptionally good job.

There are three major stages for printing hardcopy of a LaTeXinfo file. One is for formatting the file, the second is for sorting the index, and the third is for printing the formatted document. When you use the shell commands, you can either work directly in the operating system shell or work within a shell inside of GNU Emacs.

Instead of shell commands, you can use commands provided by LaTeXinfo mode. In addition to three commands to format a file, sort the indices, and print the result, LaTeXinfo mode offers key bindings for commands to recenter the output buffer, show the print queue, and delete a job from the print queue.

16.1 How to Print Using Shell Commands

Format the LaTeXinfo file with the shell command `latex` followed by the name of the LaTeXinfo file. This produces a formatted DVI file as well as several auxiliary files containing indices, cross references, etc. The DVI file (for *DeVice Independent* file) can be printed on a wide variety of printers.

The `latex` formatting command itself does not sort the indices; it writes an output file of unsorted index data. Hence, to generate a printed index, you first need a sorted index to work from. The `latexindex` command sorts indices.[1]

The `latex` formatting command outputs unsorted index files under names that obey a standard convention. These names are the name of your main input file to the `latex` formatting command, with everything after the first period thrown away, and the two letter names of indices added at the end. For example, the raw index output files for the input file '`foo.tex`' would be '`foo.cp`', '`foo.vr`', '`foo.fn`', '`foo.tp`', '`foo.pg`' and '`foo.ky`'. Those are exactly the arguments to give to `latexindex`. Or else, under Unix you can use '`??`' as "wild-cards" and give the command in this form:

[1] The source file '`latexindex.c`' comes as part of the standard LaTeXinfo distribution and is usually installed when LaTeXinfo is installed.

```
latexindex foo.??
```

This command will run `latexindex` on all the unsorted index files. (You may execute '`latexindex foo.??`' even if there are similarly named files with two letter extensions that are not index files, such as '`foo.el`'. The `latexindex` command reports but otherwise ignores such files.) For each file specified, `latexindex` generates a sorted index file whose name is made by appending '`s`' to the input file name. The `\printindex` command knows to look for a file of that name. `latexindex` does not alter the raw index output file.

If you have a bibliography, you must also run the BibTeX program on the '`aux`' file generated by LaTeX. For example,

```
bibtex foo.aux
```

After you have sorted the indices or formatted the bibliography, you need to rerun the `latex` formatting command on the LaTeXinfo file. This regenerates a formatted DVI file with up-to-date index entries.[2]

To summarize, this is a three step process:

1. Run the `latex` formatting command on the LaTeXinfo file. This generates the formatted DVI file as well as the raw index files with two letter extensions.

2. Run the shell command `latexindex` on the raw index files to sort them. This creates the corresponding sorted index files.

3. Run the shell command `bibtex` on the raw index files to format the bibliography. This creates the corresponding '`.bbl`' file.

4. Rerun the `latex` formatting command on the LaTeXinfo file. This regenerates a formatted DVI file with the index entries in the correct order. This second run also makes all the cross references correct as well.

You need not run `latexindex` each time after you run the `latex` formatting. If you don't, on the next run, the `latex` formatting command will use whatever sorted index files happen to exist from the previous use of `latexindex`. This is usually OK while you are in the early stages of writing a document.

Rather than type the `latex`, `bibtex` and `latexindex` commands yourself, you can use the shell script `latex2dvi`. This shell script is designed to simplify the `latex`, `latexindex`, `bibtex`, `latex` sequence by figuring out whether index files and DVI files are up-to-date. It runs `latexindex` and `latex` only when necessary. The syntax for `latex2dvi` is like this (where '`%`' is the shell prompt):

[2] If you use more than one index and have cross references to an index other than the first, you must run `latex` *three times* to get correct output: once to generate raw index data; again (after `latexindex`) to output the text of the indices and determine their true page numbers; and a third time to output correct page numbers in cross references to them. This may also be necessary to update the table of contents if the number of pages used by the indices or bibliography changed.

```
% latex2dvi filenames...
```

Finally, you can print a DVI file with the DVI print command. The precise command to use depends on the system; '`lpr -d`' is common. The DVI print command may require a file name without any extension or with a '`.dvi`' extension.

The following commands, for example, sort the indices, format, and print the *Foo Lisp Manual* (where '`%`' is the shell prompt):

```
% latex foo.tex
% latexindex foo.??
% bibtex foo.aux
% latex foo.tex
% lpr -d foo.dvi
```

(Remember that the shell commands may be different at your site; but these are commonly used versions.)

16.2 Printing from an Emacs Shell

You can give formatting and printing commands from a shell within GNU Emacs. To create a shell within Emacs, type `M-x shell`. In this shell, you can format and print the document. See section How to Print in How to Print Using Shell Commands, for details.

You can switch to and from the shell buffer while `latex` is running and do other editing. If you are formatting a long document on a slow machine, this can be very convenient. You can also use `latex2dvi` from an Emacs shell. (See section How to Print in How to Print Using Shell Commands.)

16.3 Formatting and Printing in LaTeXinfo Mode

LaTeXinfo mode provides several predefined key commands for LaTeX formatting and printing. These include commands for sorting indices, looking at the printer queue, killing the formatting job, and recentering display of the buffer in which the operations occur.

C-c C-t C-r (latexinfo-latex-region) Run LaTeX on the current region.

C-c C-t C-b (latexinfo-latex-buffer) Run LaTeX on the current buffer.

C-c C-t C-i (latexinfo-latexindex) Sort the indices of a LaTeXinfo file formatted with `latexinfo-latex-region` or `latexinfo-latex-buffer`.

C-c C-t C-p (latexinfo-latex-print) Print a DVI file that was made with `latexinfo-latex-region` or `latexinfo-latex-buffer`.

C-c C-t C-q (latexinfo-show-latex-print-queue) Show the print queue.

`C-c C-t C-b`	Run LaTeX on the buffer.
`C-c C-t C-i`	Sort the indices.
`C-c C-t C-b`	Rerun LaTeX to regenerate indices.
`C-c C-t C-p`	Print the DVI file.
`C-c C-t C-q`	Display the printer queue.

Table 16.1: Formatting a Buffer Commands

C-c C-t C-d (latexinfo-delete-from-latex-print-queue) Delete a job from the print queue; you will be prompted for the job number shown by a preceding `C-c C-t C-q` command (`latexinfo-show-latex-print-queue`).

C-c C-t C-k (latexinfo-kill-latex-job) Kill the currently running LaTeX job started by `latexinfo-latex-region` or `latexinfo-latex-buffer`, or any other process running in the LaTeXinfo shell buffer.

C-c C-t C-x (latexinfo-quit-latex-job) Quit a LaTeX formatting job that has stopped because of an error by sending an `X` to it. When you do this, LaTeX preserves a record of what it did in a '`.log`' file.

C-c C-t C-l (latexinfo-recenter-latex-output-buffer) Redisplay the shell buffer in which the LaTeX printing and formatting commands are run to show its most recent output.

Thus, the usual sequence of commands for formatting a buffer is as follows (with comments to the right):

The LaTeXinfo mode LaTeX formatting commands start a subshell in Emacs called the '`*latexinfo-latex-shell*`'. The `latexinfo-latex-command`, `latexinfo-latexindex-command`, and `latex-dvi-print-command` commands are all run in this shell. You can watch the commands operate in the '`*latexinfo-latex-shell*`' buffer, and you can switch to and from and use the '`*latexinfo-latex-shell*`' buffer as you would any other shell buffer.

The formatting and print commands depend on the values of several variables. The default values are:

The default values of `latexinfo-latex-command` and `latexinfo-latexindex-command` are set in the '`latexnfo-tex.el`' file.

You can change the values of these variables with the `M-x edit-options` command (See Info file '`emacs`', node 'Edit Options'), with the `M-x set-variable` command (See Info file '`emacs`', node 'Examining'), or with your '`.emacs`' initialization file (See Info file '`emacs`', node 'Init File').

16.4 Using the Local Variables List

Yet another way to apply the LaTeX formatting command to a LaTeXinfo file is to put that command in a *local variables list* at the end of the LaTeXinfo file. You can then specify the LaTeX

Variable	Default value
latexinfo-latex-command	"latex"
latexinfo-latexindex-command	"latexindex"
latexinfo-latex-shell-cd-command	"cd"
latexinfo-latex-dvi-print-command	"lpr -d"
latexinfo-show-latex-queue-command	"lpq"
latexinfo-delete-from-print-queue-command	"lprm"
latexinfo-start-of-header	"\begin{document}"
latexinfo-end-of-header	"\setfilename"
latexinfo-latex-trailer	"\end{document}"

Table 16.2: Formatting a Document Commands

formatting command as a `compile-command` and have Emacs run the LaTeX formatting command by typing `M-x compile`. This creates a special shell called the '`*compilation buffer*`' in which Emacs runs the compile command. For example, at the end of the '`gdb.texinfo`' file, after the `\end{document}`, you would put the following:

```
\c Local Variables:
\c compile-command: "latex2dvi foo.tex"
\c End:
```

This technique is most often used by programmers who also compile '`C`' programs this way. (See Info file '`emacs`', node 'Compilation'.)

Usually, the file's first line contains an '`\c -*-latexinfo-*-`' comment that causes Emacs to switch to LaTeXinfo mode when you edit the file. In addition, the beginning must include a `\begin{document}`. After this follows the title page, a copyright page, and permissions, and a table of contents. Besides an `\end{document}`, the end of a file usually includes indices and the bibliography.

16.5 Preparing for Use of LaTeX

You must put `latexinfo` as an option to the `documentstyle` of every LaTeXinfo file to tell LaTeX to use the '`latexinfo.sty`' file when it is processing the LaTeXinfo source file. Otherwise LaTeX will not know what to do with the commands. See section 2.4.1 [The Documentstyle], page 18. If LaTeXinfo has been installed properly, LaTeX should find the file automatically. See section A [Installing LaTeXinfo], page 163, if you have troubles.

16.6 Overfull "Hboxes"

LaTeX is sometimes unable to typeset a line without extending it into the right margin. This can occur when LaTeX comes upon what it interprets as a long word that it cannot hyphenate,

such as an electronic mail network address or a very long title. When this happens, LaTeX prints an error message like this:

```
Overfull  \hbox (20.76302pt too wide)
```

(In LaTeX, lines are in "horizontal boxes", hence the term, "hbox".)

LaTeX also provides the line number in the LaTeXinfo source file and the text of the offending line, which is marked at all the places that LaTeX knows how to hyphenate words.

If the LaTeXinfo file has an overfull hbox, you can rewrite the sentence so the overfull hbox does not occur, or you can decide to leave it. A small excursion into the right margin often does not matter and may not even be noticeable.

However, if you do leave an overfull hbox, unless told otherwise, LaTeX will print a large, ugly, black rectangle beside the line. This is so you will notice the location of the problem if you are correcting a draft. To prevent such a mark from marring your final printout, put the following in the beginning of the LaTeXinfo file on a line of its own, before the `\maketitle` command:

```
\finalout
```

Chapter 17

Catching Formatting Mistakes

Besides mistakes with the content of what ever you are describing, there are two kinds of mistake you can make with LaTeXinfo: you can make mistakes with commands, and you can make mistakes with the structure of the nodes and chapters. There are two tools for catching the first kind of mistake and two for catching the second.

For finding problems with commands, your best action is to run `M-x latexinfo-format-region` on regions of your file as you write it. In LaTeXinfo mode, the `latexinfo-format-region` command is bound to `↑c ↑f`. In addition, you can run LaTeX on the whole file.

For finding problems with the structure of nodes and chapters, you can use `↑c ↑s` (`latexinfo-show-structure`) (see section 17.3 [Using latexinfo-show-structure], page 147,) the related `occur` command (pxrefUsing occur,) and you can use the `M-x Info-validate` command (see section 17.5 [Running Info-Validate], page 149.)

17.1 Catching Errors with Info Formatting

After you have written part of a LaTeXinfo file, you can use the `M-x latexinfo-format-region` command to see whether the region formats properly. In LaTeXinfo Mode, this command is bound to the keyboard command `↑c ↑f`. If you have made a mistake with a command, `M-x latexinfo-format-region` will stop processing at or after the error and give an error message. To see where in the file the error occurred, switch to the '`*Info Region*`' buffer; the cursor will be in a position that is after the location of the error. Also, the text will not be formatted after the place the error occurred (or more precisely, where it was detected).

The `latexinfo-format-region` command sometimes provides slightly odd error messages. For example, if you forget a closing brace,

```
( \xref{Catching Formatting Mistakes, for more info.)
```

In this case, `latexinfo-format-region` detects the missing closing brace but displays a message that says '`Unbalanced parentheses`' rather than '`Unbalanced braces`'. This is because

the formatting command looks for mismatches between braces as if they were parentheses.

Sometimes `latexinfo-format-region` fails to detect mistakes. For example, in the following, the closing brace is swapped with the closing parenthesis:

```
( \xref{Catching Formatting Mistakes), for more info.}
```

Formatting produces:

```
(*Note for more info.: Catching Formatting Mistakes)
```

The only way for you to detect this error is to realize that the reference should have looked like this:

```
(*Note Catching Formatting Mistakes::, for more info.)
```

17.2 Catching Errors with LaTeX Formatting

You can also catch mistakes when you format a file with LaTeX. Usually, you will want to do this after you have run `latexinfo-format-buffer` on the same file, because `latexinfo-format-buffer` sometimes displays error messages that make more sense than LaTeX. (See section 17.1 [Debugging with Info], page 145, for more information.)

For example, LaTeX was run on a LaTeXinfo file, part of which is shown here:

```
---------- Buffer: latexinfo.tex ----------
name of the latexinfo file as an extension.  The
\samp{??} are `wildcards' that cause the shell to
substitute all the raw index files.  ( \xref{sorting
indices, for more information about sorting
indices.) \refill
---------- Buffer: latexinfo.tex ----------
```

(The cross reference lacks a closing brace.) LaTeX produced the following output, after which it stopped:

```
---------- Buffer: *latexinfo-latex-shell* ----------
Runaway argument?
{sorting indices, for more information about sorting
indices.)  \refill  \ETC.
! Paragraph ended before  \xref was complete.
<to be read again>
                    \par
l.27

?
---------- Buffer: *latexinfo-latex-shell* ----------
```

In this case, LaTeX produced an accurate and understandable error message:

```
Paragraph ended before  \xref was complete.
```

('`\par`' is an internal LaTeX command, which is how it represents a new paragraph marker.) Because the `}` was forgotten from the `\xref` command, LaTeXnoticed that the paragraph ended before the command was complete.

Unfortunately, LaTeX is not always so helpful, and sometimes you have to be truly a Sherlock Holmes to discover what went wrong. In any case, if you run into a problem like this, you can do one of two things.

1. You can tell LaTeX to continue running and to ignore errors as best it can by typing `r RET` at the '`?`' prompt.

   This is often the best thing to do. However, beware: the one error may produce a cascade of additional error messages as its consequences are felt through the rest of the file. (To stop LaTeX when it is producing such an avalanche of error messages, type `C-c` (or `C-c C-c`, if you running a shell inside of Emacs.))

2. You can tell LaTeX to stop this run by typing `x RET` at the '`?`' prompt.

Sometimes LaTeX will format a file without producing error messages even though there is a problem. This usually occurs if a command is not ended but LaTeX is able to continue processing anyhow. For example, if you fail to end an itemized list with the `\end{itemize}` command, LaTeX will write a DVI file that you can print out. The only error message that LaTeX will give you is the somewhat mysterious comment that

```
( \end occurred inside a group at level 1)
```

However, if you print the DVI file, you will find that the text of the file that follows the itemized list is entirely indented as if it were part of the last item in the itemized list. The error message is the way LaTeX says that it expected to find an `\end` command somewhere in the file; but that it could not determine where it was needed.

17.3 Using `latexinfo-show-structure`

It is not always easy to keep track of the nodes, chapters, sections, and subsections of a LaTeXinfo file. This is especially true if you are revising or adding to a LaTeXinfo file that someone else has written.

In GNU Emacs, in LaTeXinfo mode, the `latexinfo-show-structure` command lists all the lines that begin with the `\`-commands that specify the structure: `\chapter`, `\section`, `\chapter`, and so on. With an argument (prefix, if interactive), the command also shows the `\node` lines. The `latexinfo-show-structure` command is bound to `C-c C-s` in LaTeXinfo mode, by default.

The lines are displayed in a buffer called the '`*Occur*`' buffer. For example, when `latexinfo-show-structure` was run on an earlier version of this appendix, it produced the following:

```
Lines matching "^ \ \ \(chapter \ \|sect \ \|sub \ \|unnum \
\)" in buffer latexinfo.tex.
  4: \chapter{Catching Formatting Mistakes}
 52: \section{Catching Errors with Info Formatting}
222: \section{Catching Errors with  \LaTeX{} Formatting}
338: \section{Using  \code{latexinfo-show-structure}}
407: \subsection{Using  \code{occur}}
444: \section{Finding Badly Referenced Nodes}
513: \subsection{Running  \code{Info-validate}}
573: \subsection{Splitting a File Manually}
```

This says that lines 4, 52, and 222 of '`latexinfo.tex`' begin with the `\chapter`, `\section`, and `\section` commands respectively. If you move your cursor into the '`*Occur*`' window, you can position the cursor over one of the lines and use the `C-c C-c` command (`occur-mode-goto-occurrence`), to jump to the corresponding spot in the LaTeXinfo file. See Info file '`emacs`', node 'Other Repeating Search', for more information about `occur-mode-goto-occurrence`.

Remark: The first line in the '`*Occur*`' window describes the *regular expression* specified by *latexinfo-heading-pattern*. This regular expression is the pattern that `latexinfo-show-structure` looks for. See Info file '`emacs`', node 'Regexps', for more information.

When you invoke the `latexinfo-show-structure` command, Emacs will display the structure of the whole buffer. If you want to see the structure of just a part of the buffer, of one chapter, for example, use the `C-x n` (`narrow-to-region`) command to mark the region. (See Info file '`emacs`', node 'Narrowing'.) This is how the example used above was generated. To see the whole buffer again, use the command `C-x w` (`widen`).

If you call `latexinfo-show-structure` with a prefix argument by typing `C-u C-c C-s`, it will list lines beginning with `\node` as well as the lines beginning with the `\`-commands for `\chapter`, `\section`, and the like.

You can remind yourself of the structure of a LaTeXinfo file by looking at the list in the '`*Occur*`' window; and if you have mis-named a node or left out a section, you can correct the mistake.

17.4 Using `occur`

Sometimes the `latexinfo-show-structure` command produces too much information. Perhaps you want to remind yourself of the overall structure of a LaTeXinfo file, and are overwhelmed by the detailed list produced by `latexinfo-show-structure`. In this case, you can use the `occur` command directly. To do this, type

```
M-x occur
```

and then, when prompted, type a *regexp*, a regular expression for the pattern you want to match. (See Info file '`emacs`', node 'Regexps'.) The `occur` command works from the current

location of the cursor in the buffer to the end of the buffer. If you want to run `occur` on the whole buffer, place the cursor at the beginning of the buffer.

For example, to see all the lines that contain the word '`\chapter`' in them, just type '`\\chapter`'. This will produce a list of the chapters. It will also list all the sentences with '`\chapter`' in the middle of the line. If you want to see only those lines that start with the word '`\chapter`', type '`^\\chapter`' when prompted by `occur`. If you want to see all the lines that end with a word or phrase, end the last word with a '`$`'; for example, '`Catching Formatting Mistakes$`'. This can be helpful when you want to see all the nodes that are part of the same chapter or section and therefore have the same 'Up' pointer.See Info file '`emacs`', node 'Other Repeating Search', for more information.

17.5 Finding Badly Referenced Nodes

You can use the `Info-validate` command to check whether any of the 'Next', 'Previous', 'Up' or other node pointers fail to point to a node. This command checks that every node pointer points to an existing node. The `Info-validate` command works only on Info files, not on LaTeXinfo files.

17.5.1 Running `Info-validate`

To use `Info-validate`, visit the Info file you wish to check and type:

```
M-x Info-validate
```

(Note that the `Info-validate` command requires an upper case 'I'. You may also need to create a tag table before running `Info-validate`. See section 17.5.3 [Tagifying], page 150.)

If your file is valid, you will receive a message that says "File appears valid". However, if you have a pointer that does not point to a node, error messages will be displayed in a buffer called '`*problems in info file*`'.

For example, `Info-validate` was run on a test file that contained only the first node of this manual. One of the messages said:

```
In node "Overview", invalid Next: LaTeXinfo Mode
```

This meant that the node called '`Overview`' had a 'Next' pointer that did not point to anything (which was true in this case, since the test file had only one node in it).

Now suppose we add a node named '`LaTeXinfo Mode`' to our test case but we don't specify a 'Previous' for this node. Then we will get the following error message:

```
In node "LaTexinfo Mode", should have Previous: Overview
```

This is because every 'Next' pointer should be matched by a 'Previous' (in the node where the 'Next' points) which points back. `Info-validate` also checks that all menu items and cross references point to actual nodes.

Note that `Info-validate` requires a tag table and does not work with files that have been split. (The `latexinfo-format-buffer` command automatically splits files larger than 100,000 bytes.) In order to use `Info-validate` on a large file, you must run `latexinfo-format-buffer` with an argument so that it does not split the Info file; and you must create a tag table for the unsplit file.

17.5.2 Creating an Unsplit File

You can run `Info-validate` only on a single Info file that has a tag table. The command will not work on the indirect subfiles that are generated when a master file is split. If you have a large file (longer than 70,000 bytes or so), you need to run the `latexinfo-format-buffer` command in such a way that it does not create indirect subfiles. You will also need to create a tag table for the Info file. After you have done this, you can run `Info-validate` and look for badly referenced nodes.

The first step is to create an unsplit Info file. To prevent `latexinfo-format-buffer` from splitting a LaTeXinfo file into smaller Info files, give a prefix to the `M-x latexinfo-format-buffer` command:

```
C-u  M-x latexinfo-format-buffer
```

When you do this, LaTeXinfo will not split the file and will not create a tag table for it.

17.5.3 Tagifying a File

After creating an unsplit Info file, you must create a tag table for it. Visit the Info file you wish to tagify and type:

```
M-x Info-tagify
```

(Note the upper case `I` in `Info-tagify`.) This creates an Info file with a tag table that you can validate.

The third step is to validate the Info file:

```
M-x Info-validate
```

(Note the upper case `I` in `Info-validate`.) In brief, the steps are:

```
C-u M-x latexinfo-format-buffer
M-x Info-tagify
M-x Info-validate
```

After you have validated the node structure, you can rerun `latexinfo-format-buffer` in the normal way so it will construct a tag table and split the file automatically, or you can make the tag table and split the file manually.

17.5.4 Splitting a File Manually

You should split a large file or else let the `latexinfo-format-buffer` command do it for you automatically. (Generally you will let one of the formatting commands do this job for you. See section 14 [Creating and Installing an Info File], page 119.)

The split off files are called the indirect subfiles. Info files are split to save memory. With smaller files, Emacs does not have make such a large buffer to hold the information. If an Info file has more than 30 nodes, you should also make a tag table for it. See section 17.5.1 [Using Info-validate], page 149, for information about creating a tag table. (Again, tag tables are usually created automatically by the formatting command; you only need to create a tag table yourself if you are doing the job manually. Most likely, you will do this for a large, unsplit file on which you have run `Info-validate`.)

Visit the file you wish to tagify and split and type the two commands:

```
M-x Info-tagify
M-x Info-split
```

(Note that the '`I`' in '`Info`' is upper case.)

When you use the `Info-split` command, the buffer is modified into a (small) Info file which lists the indirect subfiles. This file should be saved in place of the original visited file. The indirect subfiles are written in the same directory the original file is in, with names generated by appending '`-`' and a number to the original file name.

The primary file still functions as an Info file, but it contains just the tag table and a directory of subfiles.

Chapter 18

Extending LaTeXinfo

One of the advantages of LaTeXinfo is that it is easy to add your own extensions. Adding new styles in a standard feature of LaTeX, and this makes it easy to modularize your additions by plcing them in style files. There are a large number of publically available style files that can be found on the `Internet` by `anonymous ftp`, for example on `soe.clarkson.edu`.

In LaTeXinfo, you can similarly make additions to the on-line manual generator by making GNU Emacs handlers for your LaTeX extensions. This is the Emacs counterpart to the `documentstyle` options. LaTeXinfo looks in a specified directory for GNU Elisp code that corresponds to each style file, by looking for the file named *style*`-fmt.el`. If this file is found, then it is loaded into the Emacs session when `latexinfo-format-buffer` is called (see section 14.1 [Creating an Info file], page 119). Look in the `styles` and `elisp` directories of the LaTeXinfo distribution for examples of this, and in the next section we will show a simple example of how this works.

18.1 Optional Style Files

LaTeX provides a number of optional style files by default. These include `latexinfo`, `11pt`, `12pt`, `twoside` and `titlepage`. If any of the optional styles is a member of the Emacs variable `latexinfo-known-document-styles`, then LaTeXinfo does not bother to look for the associated `-fmt` file. By default this list is:

```
'(latexinfo 11pt 12pt twoside titlepage A4 a4 dina4 psfonts format))
```

18.1.1 The fvpindex Style

18.1.2 fvpindex Style

Let's say that you wanted to develop a special style for a program, which defined the command `\f` to be used for specifying functions. This command would put its argument in the function index, and set the function in the printed manual in a special font. The LaTeX commands to do this are quite simple. Firstly, define the `\f` command, to put its argument in the `fn` index, and set its argument in `sf` font.

```
\def\f#1{\findex{#1}{\sf #1}}
```

But what about the Info file? As it stands, the command `\f` is not defined in LaTeXinfo, so when you formatted the buffer it would ignore all the `\f` commands, and their arguments. You need to introduce the appropriate Emacs lisp code to provide the definition of the command that you have added. For each option in the `documentstyle` command, LaTeXinfo looks to see if the file name *option*`-fmt.el` exists in the directory defined by the Emacs variable `latexinfo-formats-directory`. (This variable defaults to the value of the environment variable `LATEXINFO`, or if that has not been defined, then the current directory). If it does exist, then it loads this file. So continuing with our example, if the file '`fvpindex-fmt.el`' contained the code

```
(put 'f 'latexinfo-format 'latexinfo-format-code)
```

then it would define the `\f` command to treat its argument the same way that the `\code` command does.

After the *option*`-fmt.el` has been loaded, LaTeXinfo checks to see if a function (of no arguments) called *option*`-fmt-hook` has been defined. If so, this function is called. This allows you to define functions in the *option*`-fmt.el` file that operate on the whole LaTeXinfo file.

You can use the `\documentstyle` optional called `fvpindex` that loaded the style '`fvpindex.sty`', which contains these definitions, and similar definitions for `\v` and `\p`. Include `fvpindex` in the list of options to the `documentstyle` command, *after* the `latexinfo` option. Your LaTeXinfo file would begin with something like:

```
\documentstyle[12pt,latexinfo,fvpindex]{book}
```

This provides a convenient way of documenting all functions, variables ans packages of a program, and having their names automatically entered in the appropriate index, and set in the font of your choice. Additionally, if you are using `fvpindex` in conjunction with the `elisp` or `clisp` styles, you will find that the `\defun` commands put their index entries in in index in bold type, whereas the definitions for `\f`, `\v` and `\p` set their entries in normal type. This allows you to distinguish where the function was defined, and where it was simply referenced.

18.1.3 Clisp Style

A more modern approach to the Lisp `back defun` commands can be found in the style `clisp`. The format of the commands is similar to that found in the earlier chapter on Definition Commands (see section 10 [Definition Commands], page 77). This style is still evolving, and may have new features of changes in the next release of LaTeXinfo. The commands of this style are summarized below.

The principal differences between this style and the `elisp` style are the following:

- An optional parameter can be defined after the name of the command, that is used to indicate the package to which the entity belongs. Insert this optional argument in the traditional LaTeX style of using square brackets.

Command Name	Language	Class
deffn	Lisp	general functions
deffun	Lisp	functions
defspec	Lisp	special forms
defmac	Lisp	macros
defvr	Lisp	general variables
defvar	Lisp	variables
defconst	Lisp	constants

Table 18.1: The Clisp Definition Commands

- The function arguments can contain the keywords `\&optional`, `\&rest`, and `\&key`.
- The function arguments can contain the functions `\keys{...}` and `\morekeys{...}` to properly align the keyword arguments of a function.
- The variable and function index entries are coerced to lower case.
- The commands `\true`, `\false`, `\empty` and `\nil` are defined to print as `t`, `nil`, `()`and `nil` respectively.

`\defun``{`*name*`}[`*package*`]{`*arguments*`...}` The `\defun` command is the definition command for functions. `\defun` is equivalent to '`\deffn{Function} ...`'. The package argument is optional, and the square brackets are omitted if no package is provided. Within the argument list, the following keywords are recognized: `\&optional`, `\&rest`, and `\&key`. They print as themselves in the `\code` font.

The argument names on the `\defun` line do not automatically appear in italics in the printed manual; they should be enclosed in `\var`. Terminate the definition with `\enddefun` on a line of its own.

Within the argument list, the following commands are recognized:

`\args{}` which does nothing.

`\keys{...}` prints the word `& key`, and sets the tab stop to be align subsequent keys.

`\morekeys{...}` starts a new line and moves to the tab stop set by `\keys`.

`\yetmorekeys{...}` the same as `\morekeys`.

Implementation note: The arguments to these functions are set with a LaTeX minpage environment. This means that new lines within the argument list will start new lines in the region between the function name and the function type. Furthermore, the arguments are contained within a `tabbing` environment, that allows the use of the `\=` and `\>` tab–set and tab commands. This allows one to line up parts of the argument list, such as keys, and the `\*keys` commands are implemented in terms of these.

\defmac{*name*}[*package*]{*arguments...*} The `\defmac` command is the definition command for macros. `\defmac` is equivalent to '`\deffn{Macro} ...`'. The package argument is optional, and the square brackets are omitted if no package is provided. Within the argument list, the following commands are recognized: `\&optional`, `\&rest`, and `\&key`. They print as themselves in the `\code` font. `\defspec` is similarly defined for special forms.

Within the `\defun` and `\defmac` argument lists, the following special functions are recognized:

mopt To indicate [*optional forms*] .

mchoice To indicate ⟦ *a choice of forms* ⟧ .

mstar To indicate 0 or more {*optional forms*}* .

mplus To indicate 1 or more {*forms*}+ .

mgroup To indicate a group of {*forms*}+ .

mor To indicate an or between | forms.

mind To indicate an ↓*form*.

For more information on this syntax, see [Ste90].

\defvar{*name*}[*package*] The `\defvar` command is the definition command for variables. `\defvar` is equivalent to '`\deffn{Variable}`'. `\defconst` is similarly defined for constants.

In addition to these commands, there are the corresponding "head-less" commands: `\deffnx`, `\deffunx`, `\defspecx`, `\defmacx`, `\defvrx`, `\defvarx`, `\defconstx`, which are defined identically to the corresponding commands expect that no extra space is put before the command heading. You can use these on the second or more of a section that describes a number of definitions.

18.2 LaTeXinfo support for European languages

LaTeXinfo tries to support European languages, but it is an area that is in great flux right now. '`german.sty`' is supported as an optional file, and this will also provide some support for French.

The following diacrtical marks are supported by default in LaTeXinfo, either in the form `\`*letter* or `\{`*letter*`}`

`\^` Circumflex accent: ĉ.

`\‘` Accute accent: è.

`\’` Grave accent: é.

`\"` Trémat: ö.

In the Info file, these marks are removed.

But note that by default, the commands `\c`, `\b` `\i` are used for other purposes than their LaTeX usage as diacritical marks.

The hyphenation character `\-` is also supported.

To support Multi-lingual TeX, `latexinfo.sty` looks for the presence of the LaTeX number `\language`, which are assumed to be defined as follows:

```
\newcount\USenglish  \global\USenglish=0
\newcount\german     \global\german=1
\newcount\austrian   \global\austrian=2
\newcount\french     \global\french=3
\newcount\english    \global\english=4
```

The presence of `\language` set to any of `\english` `\english`, `\french` or `\german` changes the way the cross-references are printed in LaTeX. The default is `\english`.

18.2.1 german.sty

LaTeXinfo has support for the file '`german.sty`', as of Vers. 2.3, 7 Aug 1990, collected by H. Partl (TU Wien), using ideas

by W. Appelt, F. Hommes et al. (GMD St.Augustin), T. Hofmann (CIBA-GEIGY Basel), N. Schwarz (Uni Bochum), J. Schrod (TH Darmstadt), D. Armbruster (Uni Stuttgart), R. Schoepf (Uni Mainz), and others. It is a document style option for writing german texts with TeX or LaTeX. It can be called via adding the `german` option to the `\documentstyle` command. **Note:** User's should resort to their already-installed version of '`german.sty`' (if any) before using the one from LaTeXinfo, so the existing LaTeX site documentation won't break. Various copies of this file exist from different eras; you may wish to inquire if one is already installed at your site, and look to see if it is more or less recent than the one distributed with LaTeXinfo.

To support Multi-lingual TeX, `latexinfo.sty` looks for the presence of the LaTeX number `\language`, and if it is set to `\german`, it sets the cross-references in German, and looks to see

if `\mdqon` is defined. If so, it lets double quotes have their special meaning, and otherwise sets them as double quotes in typewriter font.
Implementation note: This file conforms to the standard for *Einheitliche deutsche TeX-Befehle* as proposed at the 6th Meeting of German TeX Users in Muenster, October 1987.

18.2.1.1 Commands to be used by the end users

"a for Umlaut-a (like ä), also for all other vowels.

"s for sharp s (like `\ss`).

"ck for ck to be hyphenated as k-k.

"ff for ff to be hyphenated as ff-f, also for certain other consonants.

"| to separate ligatures.

"- like , but allowing hyphenation in the rest of the word.

"" like "-, but producing no hyphen sign.

"‘ or `\glqq` for german left double quotes (similar to ,,)

"’ or `\grqq` for german right double quotes (similar to “)

`\glq` for german left single quotes (similar to ,)

`\grq` for german right single quotes (similar to ‘)

"< or `\flqq` for french left double quotes (similar to <<)

"> or `\frqq` for french right double quotes (similar to >>)

`\flq` for french left single quotes (similar to <)

`\frq` for french right single quotes (similar to >)

`\dq` for the original quotes character (")

`\setlanguage`*n* to switch to the language specified by n, which should be one of the following command names:

`\austrian` `\french` `\english` `\german` `\USenglish` this changes the date format, captions and (if “multilingual TeX”is installed) hyphenation.

`\originalTeX` to restore everything to the original settings of TEX and LATEX (well, almost everything).

`\germanTeX` to re-activate the german settings.

18.2.1.2 Obsolete Commands

Obsolete commands, provided for compatibility with existing applications:

\3 for sharp s (like "s).

\ck for ck to be hyphenated as k-k (like "ck).

18.2.1.3 Lower Level Commands and Features

\umlautlow redefines the Umlaut accent such that the dots come nearer to the letter and that hyphenation is enabled in the rest of the word.

\umlauthigh restores ¨ to its original meaning.

\ss is **\lccode**'d to enable hyphenation.

\mdqon makes " an active (meta-) character that does the pretty things described above.

\mdqoff restores " to its original meaning.

\dospecials, sanitize are redefined to include ".

\dateaustrian, \dategerman, \dateenglish, \dateUSenglish, \datefrench redefine **\today** to use the respective date format.

\captionsgerman, \captionsenglish, \captionsfrench switch to german, english or french chapter captions and the like, resp. This will have an effect only if the document style files use the symbolic names **\chaptername** etc. instead of the original english words.

\language a count that is set by **\setlanguage** and can be used by document style declarations like

```
\ifnum\language=\english .textengl.\else
\ifnum\language=\german  .textgerm.\fi\fi
```

and/or by M.Ferguson's "Multilingual TêX".

Finally, **\germanTeX** is switched on.

This file can be used both with Plain TEX and with LATEX and other macro packages, and with the original TEX and LATEX fonts. Usage of german hyphenation patterns is recommended to accompany this style file when writing german texts.

The file should be read in vertical mode only (usually at the beginning of the document) to avoid spurious spaces. **\undefined** must be an undefined control sequence.

Multiple calls of this file (e.g. at the beginning of each subfile) will do no harm. Only the first call (i.e., if **\mdqon** is undefined) performs all the definitions and settings.

The catcode of `@` remains unchanged after processing of this file. All definitions are global, the switching on of the german options is local.

The commands `\mdqon`, `\mdqoff`, `\originalTeX`, `\germanTeX`, and `\setlanguage` are "fragile" with LaTeX and should not be used within arguments of macro calls.

In Plain TeX, '`\protect`' should be `\let` to '`\relax`' normally and to something like '`\string`' inside the arguments of '`\write`' or '`\message`' (see LaTeX.TEX for all the details).

The command `\umlautlow` may need adaption to font parameters (see comments there for details).

The commands `\flqq`, `\frqq`, `\flq`, `\frq`, and `\datefrench` in their present forms do not work properly with all font sizes and styles, they still require a better solution.

18.3 Writing Your Own Style Files

<to be written>

Part IV

Appendices

Appendix A

Installing LaTeXinfo

A.1 Compiling LaTeXinfo

To compile LaTeXinfo:

1. Run the shell script '`configure`'. You will be asked to provide the following:

   BINDIR Where to install the executables.
   INFODIR Where to install the info files.
   EMACS the name of your GNU Emacs.

   These must exists, and you must be able to write to these directories. For example,

```
Where would you like to install the binaries?

  Please type the full path to your binaries directory:
  >/usr5/gnu/bin-sparc

  The binaries path was verfied to be [/usr5/gnu/bin-sparc]

Where are the Gnu Info files located?

  Please type the full path to your info directory:
  >/usr5/gnu/info

  The Info directory was verfied to be [/usr5/gnu/info]

Where is your GNU Emacs command:

  Please type the name of your GNU Emacs command:
  >xemacs
```

2. Then you will be asked:

```
Would you like to install the elisp and LaTeX files elsewhere,
or leave them here, and set an environment variable to point to here?

Set an environment variable to point to here [y/n]?
```

   If you choose

   y '`configure`' will set the environment variable `LATEXINFO` in the '`.login`' to point to this directory, and you won't need to `make install`.

   n You will be asked about:

   ELISPDIR Where to install the compiled Elisp code.

   TEXDIR Where to install the style files.

   For example,

```
Set an environment variable to point to here [y/n]? n

Where would you like to install GNU Emacs code (elisp)?

  Please type the full path to your elisp directory:
  >/usr5/gnu/lib/emacs/latexinfo

  The elisp path was verfied to be [/usr5/gnu/lib/emacs/latexinfo]

Where would you like to install the LaTeX style files?

  Please type the full path to your LaTeX style directory:
  >/usr5/gnu/lib/tex

  The  LaTeX style path was verfied to be [/usr5/gnu/lib/tex]
```

3. Type `make`. This will make the executables `latexindex`, `info`, make the manual, and compile the '`.el`' files. It will also make the files `.emacs` and `.login`.

You may also have to change the definitions of your LaTeX commands in the shell script '`manual/latex2dvi`' is you are unusual.

A.2 Installing the LaTeXinfo Distribution

1. If you chose to install the elisp and LaTeX files elsewhere, type `make install` to make the executables, the manual and compile the '`.el`' files. This will

   `make install.C` which will move the executables to `BINDIR`.

   `make install.manual` which will move a copy of the files '`manual/latexinfo.info*`' to the info directory of the GNU Emacs distribution specified by `INFODIR`, and a copy of the sample file to the LaTeX styles directory specified by `TEXDIR`.

   `make install.elisp` which will move a copy of the files '`styles/*.elc`' to the the GNU Emacs lisp directory specified by `ELISPDIR`.

   `make install.styles` which will move a copy of the files '`styles/*.sty`' to the the LaTeX styles directory specified by `TEXDIR`.

2. Edit the '`dir`' file in `INFO` directory to include lines like

   ```
   * LaTeXinfo: (latexinfo2.info).   With one source file, make either a
                            manual using LaTeX or an Info file.
   ```

3. Include a copy of the '`.emacs`' file in your `~/.emacs`.

4. Include a copy of the '`.login`' file in your `~/.login`.

5. Print a copy of the '`manual/latexinfo.dvi`' file and enjoy.

See section 14.2 [Installing an Info File], page 121, for more information on installing an info file. See section A.2.1 [Installing the Style Files], page 165, for more information on installing style files.

A.2.1 Installing the Style Files

Usually, the '`latexinfo.sty`' file is put in the default directory that contains LaTeX macros, something like the directory '`/usr/local/lib/tex/inputs`', which is created when LaTeX is installed. In this case, LaTeX will find the file and you don't have to do anything special. Alternatively, you can put '`latexinfo.sty`' in the directory in which each LaTeXinfo source file is located, and LaTeX will find it there.

However, you may want to specify the location of the `\input` file yourself. One way is to set the `TEXINPUTS` environment variable in your '`.login`' or '`.profile`' file. The `TEXINPUTS` environment variable will tell LaTeX where to find the '`latexinfo.sty`' file and any other file that you might want LaTeX to use. This is done by the `.login` file supplied with the LaTeXinfo distribution.

Whether you use a '`.login`' or '`.profile`' file depends on whether you use `csh`, `sh`, or `bash` for your shell command interpreter. When you use `csh`, it looks to the '`.login`' file for initialization information, and when you use `sh` or `bash`, it looks to the '`.profile`' file.

In a '`.login`' file, you could use the following `csh` command sequence:

```
setenv LATEXINFO /usr/me/mylib
# Add the format files to the list of directories that LaTeX searches.
if ( $?TEXINPUTS ) then
setenv TEXINPUTS "$TEXINPUTS"':'"$LATEXINFO"
  else
setenv TEXINPUTS "$LATEXINFO"
endif
```

In a '`.profile`' file, you could use the following `sh` command sequence:

```
TEXINPUTS=.:/usr/me/mylib:/usr/lib/tex/macros
export TEXINPUTS
```

This would cause LaTeX to look for the style file first in the current directory, indicated by the '`.`', then in a hypothetical user's '`me/mylib`' directory, and finally in the system library.

Appendix B

Converting Files to LaTeXinfo

B.1 Converting LaTeX Files to LaTeXinfo

LaTeXinfo files are essentially a special style of standard LaTeX files. To make a standard LaTeX file into a LaTeXinfo file, you must begin it with the lines

```
\documentstyle[12pt,latexinfo]{book}
\pagestyle{headings}

\begin{document}

\setfilename{latexinfo.info}
```

See section 1.5 [A Short Sample LaTeXinfo File], page 8, for details of how a LaTeXinfo file begins. Once you have added these lines, you will have a document that will pass both LaTeX, and Info formating program, but it will be a document with any node structure, so it will be in essence one large node. (See section 11 [Nodes and Menus], page 93 for more information on nodes.)

This is not very useful for the people who read the document under the info program. To add nodes and menus to the document, you can do it by hand, or you can use the function `latexinfo-insert-node-lines` (see section 15.3.2 [Other Updating Commands], page 132.) Alternatively, use the `l2latexinfo.el` file provided with LaTeXinfo, which does this, and makes a number of other conversions as well. See section B.1.1 [l2latexinfo.el], page 168.

If you want to use LaTeX commands for which there is no LaTeXinfo support of any kind, you can always wrap them in a `tex` environment:

```
\begin{tex}
...
\end{tex}
```

This ensures that this part will be ignored by the Info processor, and that all special characters will be processed according to the normal LaTeX definitions.

The following LaTeX commands are also supported by the Info formatter, although they might not do everything in Info that they do in LaTeX.

```
\LaTeX
\S
\arrow
\geq
\hfill
\label
\leq
\newblock
\newpage
\onecolumn
\pi
\pm
\protect
\qquad
\quad
\ss
\thebibliography
\thispagestyle
\tie
\twocolumn
\vspace
\vspace*
```

B.1.1 l2latexinfo.el

With the LaTeXinfo distribution is a file called '`l2latexinfo.el`', which helps convert a LaTeX file to a LaTeXinfo file. Although it is not a perfectly automatic conversion, it will convert most of a file to LaTeXinfo. To convert a LaTeX File into an LaTeXInfo file, just visit a LaTeXfile in GNU Emacs and invoke

```
Meta-x latex-to-latexinfo
```

to convert it to a LaTeXInfo file. Then search through the buffer to see if there are any command that were not converted.

When you run `latex-to-latexinfo`, you will be asked

```
Would you like to do the  \input files now, to do it all at once?
```

If you say `yes`, all the `\input` files will be included, so you can do all of the subfiles at the same time.

Remember that the characters `& ^ % $ #` are not special in LaTeXinfo. There is no support for any of the mathematics commands. Braces that are not required for LaTeXinfo commands will appear in the Info file.

B.2 Converting TeXinfo Files into LaTeXinfo Files

Documentation for GNU utilities and libraries is usually written in a format called *TEXinfo*. Perhaps the most significant difference of LaTeXinfo from TeXinfo is that if a LaTeX command is found that the Info formatter does not know about, an error is not signalled, and processing simply continues. This means that as long as you don't mind having the commands ignored in the Info file, you can use any LaTeX command.

B.2.1 Differences from TeXinfo

The following TeXinfo commands have been deleted:

```
@asis                   Not needed.
@defindex               Not needed (how many more indexes do you want??)
@dmn                    Not needed.
@ftable                 Not needed.
@itemx                  Not needed.
@setchapternewpage      Use documentstyle type and options instead.
@subtitle               You are free to use fonts in \title command.
@summarycontents        Controlled by LaTeX parameter \setcounter{tocdepth}
@titlefont              Not needed.
```

The following commands have been replaced by their LaTeX equivalents:

```
\appendixsec            replaced by \section
\appendixsubsec         replaced by \subsection
\appendixsubsubsec      replaced by \subsubsection
\bye                    replaced by \end{document}
\center                 replaced by \begin{center} .. \end{center}
\chapheading            replaced by \chapter*
\contents               replaced by \tableofcontents
\group                  replaced by \begin{same}
\heading                replaced by \section*
\headings               replaced by \pagestyle
\majorheading           replaced by \chapter*
\page                   replaced by \clearpage
\sc                     replaced by \scap
\settitle               replaced by \title
\heading                replaced by \section*
\subheading             replaced by \subsection*
\subsubheading          replaced by \subsubsection*
\table                  replaced by \begin{description}
\titlepage              replaced by \maketitle
\vskip                  replaced by \vspace
```

The following commands have been changed to their LaTeX definitions:

```
\appendix
\author
\center
\chapter
\date
\section
\subsection
\subsubsection
\begin{enumerate}
\begin{flushleft}
\begin{flushright}
\title
\today
```

The TeXinfo custom headings are supplanted by the LaTeX commands.

B.2.2 t2latexinfo.el

With the LaTeXinfo distribution is a file called '`t2latexinfo.el`', which helps convert a TeXinfo file to a LaTeXinfo file. Although it is not a perfectly automatic conversion, it will convert most of a file to LaTeXinfo. To convert a TeXinfo File into an LaTeXInfo file, just visit a TeXinfo file in GNU Emacs and invoke

```
Meta-x tex-to-latexinfo
```

to convert it to a LaTeXInfo file. Then search through the buffer to see if there are any command that were not converted. These start with the symbol '`@`'. You may have to fix up the titlepage to use `\author` and `\title` etc, and may choose to move the `setfilename` command down to somewhere after the title and copyright pages. You will also have to fix up any places where you have embedded TeX code such as

```
@tex
\overfullrule=0pt
@end tex
```

which will be converted into

```
\begin{tex}
\back overfullrule=0pt
\end{tex}
```

When you run `tex-to-latexinfo`, you will be asked

```
Would you like to do the @input files now, to do it all at once?
```

If you say `yes`, all the `@input` files will be included, so you can do all of the subfiles at the same time. You will also be asked:

```
Would you like all occurences of '@@' replaced by '@'?
```

This is normally the case, but if you say no, you will be asked

```
Would you like all occurences of '@@' replaced by ' \ \'?
```

You must choose one of these two options. The first option is normal for most TEXinfo files; the second option is only normally used for converting the TEXinfo manual itself.

B.3 Converting Scribe Files to LaTeXinfo

With the LATEXinfo distribution is a file called '`s2latexinfo.el`', which helps convert a Scribe file to a LATEXinfo file. Although it is not a perfectly automatic conversion, it will convert most of a file to LATEXinfo. To convert a Scribe file into a LATEXInfo file, just visit the Scribe file in GNU Emacs and invoke

```
Meta-x scribe-to-latexinfo
```

to convert it to a LATEXInfo file. Then search through the buffer to see if there are any commands that were not converted. These start with the symbol '`@`'. When you run `scribe-to-latexinfo`, you will be asked:

```
Would you like to do the @include files now, to do it all at once?
```

If you say `yes`, all the `@include` files will be included, so you can do all of the subfiles at the same time. You will also be asked:

Implementation note: This program was written to convert the CMU Lisp manuals. It is very heavily tailored to CMU and Common Lisp. Expect to have to alter this file to tailor it to your needs.

Appendix C

Obtaining LaTeX

TeX is freely redistributable. You can obtain TeX for Unix systems from the University of Washington for a distribution fee. LaTeX is included with TeX

To order a full distribution, send $140.00 for a 1/2-inch 9-track 1600 bpi (tar or cpio) tape reel, or $165.00 for a 1/4-inch 4-track QIC-24 (tar or cpio) cartridge, to:

```
Northwest Computing Support Center
DR-10, Thomson Hall 35
University of Washington
Seattle, Washington 98195
```

Please make checks payable to the University of Washington.

Prepaid orders are preferred but purchase orders are acceptable; however, purchase orders carry an extra charge of $10.00, to pay for processing.

Overseas sites: please add to the base cost $20.00 for shipment via air parcel post, or $30.00 for shipment via courier.

Please check with the Northwest Computing Support Center at the University of Washington for current prices and formats:

```
telephone:  (206) 543-6259
email:      elisabet@max.u.washington.edu
```

Appendix D

Command List

Here is an alphabetical list of the \-commands in LaTeXinfo. The alphabetical order ignores the `\begin{}` and `\end{}` of environment commands.

***** Force a line break. Do not end a paragraph that uses `\*` with an `\refill` command. See section 7.2.1 [Line Breaks], page 65.

**** Force a line break in the LaTeX file. See section 7.2.1 [Line Breaks], page 65.

\. Stands for a period that really does end a sentence. See section 4.3.1.3 [Controlling Spacing], page 43.

\: Indicate to LaTeX that an immediately preceding period, question mark, exclamation mark, or colon does not end a sentence. Prevent LaTeX from inserting extra whitespace as it does at the end of a sentence. The command has no effect on the Info file output. See section 4.3.1.3 [Controlling Spacing], page 43.

\{ Stands for a left-hand brace, '`{`'. See section Braces Atsigns Periods in Inserting \braces and periods.

\} Stands for a right-hand brace, '`}`'. See section Braces Atsigns Periods in Inserting \braces and periods.

\appendix Begin the appendices. All chapters and sections after this command will be treated as appendices, and marked with alphabetical chapter numbers.

\author{*author***}** Typeset *author* according to the current `documentstyles`. See section 2.5.1 [Titlepage], page 20.

\b{*text***}** Print *text* in **bold** font. No effect in Info. See section 4.2.3 [Fonts], page 42.

\back Stands for '\'. See section Braces Atsigns Periods in Inserting '\'.

\BibTeX{} Insert the logo BibTeX.

\bullet{} Generate a large round dot, or the closest possible thing to one. See section 4.3.3 [Dots Bullets], page 44.

\c *comment* Begin a comment in Texinfo. The rest of the line does not appear in either the Info file or the printed manual. A synonym for `\comment`. See section Conventions in General Syntactic Conventions.

\cartouche Highlight an example or quotation by drawing a box with rounded corners around it. Pair with `\end{cartouche}`. No effect in Info. See section cartouche in Drawing Cartouches Around Examples.)

\begin{center} Center the text following. See section 5.2.1.1 [Center Environment], page 49.

\chapter{*title***}** Begin a chapter. The chapter title appears in the table of contents of a printed manual. In Info, the title is underlined with asterisks. See section 3.3 [Chapter], page 33.

\cindex{*entry***}** Add *entry* to the index of concepts. See section Index Entries in Defining the Entries of an Index.

\cite{*reference***}** Refer to a BibTeX bibliography item. See section 8.2 [Citations], page 72.

\clearpage Start a new page in a printed manual. No effect in Info. See section page in Start a New Page.

\code{*sample-code***}** Highlight text that is an expression, a syntactically complete token of a program, or a program name. See section code in `\code`.

\comment *comment* Begin a comment in LaTeXinfo. The rest of the line does not appear in either the Info file or the printed manual, nor does following whitespace. See section Conventions in General Syntactic Conventions.

\copyright{} Generate a copyright symbol. See section 4.3.4 [LaTeX and copyright], page 45.

\defcv{*category***}{***class***}{***name***}** Format a description for a variable associated with a class in object-oriented programming. Takes three arguments: the category of thing being defined, the class to which it belongs, and its name. See section 10 [Definition Commands], page 77.

\deffn{*category***}{***name***}{***arguments...***}** Format a description for a function, interactive command, or similar entity that may take arguments. `\deffn` takes as arguments the category of entity being described, the name of this particular entity, and its arguments, if any. See section 10 [Definition Commands], page 77.

\defivar{*class***}{***instance-variable-name***}** Format a description for an instance variable in object-oriented programming. The command is equivalent to '`\defcv{Instance Variable} ...`'. See section 10 [Definition Commands], page 77.

\defmac{*macro-name*}{*arguments*...} Format a description for a macro. The command is equivalent to '`\deffn{Macro}`...'. See section 10 [Definition Commands], page 77.

\defmethod{*class*}{*method-name*}{*arguments*...} Format a description for a method in object-oriented programming. The command is equivalent to '`\defop{Method}` ...'. Takes as arguments the name of the class of the method, the name of the method, and its arguments, if any. See section 10 [Definition Commands], page 77.

\defop{*category*}{*class*}{*name*}{*arguments*...} Format a description for an operation in object-oriented programming. `\defop` takes as arguments the overall name of the category of operation, the name of the class of the operation, the name of the operation, and its arguments, if any. See section 10 [Definition Commands], page 77.

\defopt{*option-name*} Format a description for a user option. The command is equivalent to '`\defvr{User Option}` ...'. See section 10 [Definition Commands], page 77.

\defspec{*special-form-name*}{*arguments*}... Format a description for a special form. The command is equivalent to '`\deffn{Special Form}`...'. See section 10 [Definition Commands], page 77.

\deftp{*category*}{*name-of-type*}{*attributes*...} Format a description for a data type. `\deftp` takes as arguments the category, the name of the type (which is a word like 'int' or 'float'), and then the names of attributes of objects of that type. See section 10 [Definition Commands], page 77.

\deftypefn{*classification*}{*data-type*}{*name*}{*arguments*...} Format a description for a function or similar entity that may take arguments and that is typed. `\deftypefn` takes as arguments the classification of entity being described, the type, the name of the entity, and its arguments See section 10 [Definition Commands], page 77.

\deftypefun{*data-type*}{*function-name*}{*arguments*...} Format a description for a function in a typed language. The command is equivalent to '`\deftypefn{Function}`...'. See section 10 [Definition Commands], page 77.

\deftypevr{*classification*}{*data-type*}{*name*} Format a description for something like a variable in a typed language—an entity that records a value. Takes as arguments the classification of entity being described, the type, and the name of the entity. See section 10 [Definition Commands], page 77.

\deftypevar{*data-type*}{*variable-name*} Format a description for a variable in a typed language. The command is equivalent to '`\deftypevr{Variable}`...'. See section 10 [Definition Commands], page 77.

\defun{*function-name*}{*arguments*...} Format a description for functions. The command is equivalent to '`\deffn{Function}`...'. See section 10 [Definition Commands], page 77.

\defvar{*variable-name*} Format a description for variables. The command is equivalent to '`\defvr{Variable}`...'. See section 10 [Definition Commands], page 77.

\defvr{*category***}{***name***}** Format a description for any kind of variable. `\defvr` takes as arguments the category of the entity and the name of the entity. See section 10 [Definition Commands], page 77.

\begin{description} Begin a description, using `\item` for each entry. Write each first column entry as `\item[`*entry*`]`. See section 6.3 [Description Environment], page 61.

\dfn{*term***}** Highlight the introductory or defining use of a term. See section dfn in `\dfn`.

\begin{display} Begin a kind of example. Indent text, do not fill, do not select a new font. Pair with `\end{display}`. See section display in `\begin{display}`.

\dmn{*dimension***}** Format a dimension. Causes LaTeX to insert a narrow space before *dimension*. Has no effect in Info. Used for writing a number followed by an abbreviation of a dimension name, such as '`12pt`', written as '`12\dmn{pt}`', with no space between the number and the `\dmn` command. See section dmn in `\dmn`.

\end{document} Terminate LaTeX processing on the file. LaTeX does not see any of the contents of the file following the `\end{document}` command. See section 2.9 [Ending a File], page 26.

\dots{} Insert an ellipsis: '...'. See section 4.3.3 [Dots Bullets], page 44.

\emph{*text***}** Highlight *text*. See section Emphasis in Emphasizing Text.

\begin{enumerate} Begin a numbered list, using `\item` for each entry. Pair with `\end{enumerate}`. See section enumerate in `\begin{enumerate}`.

\equiv{} Indicate the exact equivalence of two forms to the reader with a special glyph: '≡'. See section 5.7.5 [Equivalence], page 56.

\error{} Indicate to the reader with a special glyph that the following text is an error message: 'error '. See section 5.7.4 [Error Special Glyph], page 56.

\begin{example} Begin an example. Indent text, do not fill, select fixed-width font. Pair with `\end{example}`. See section example in `\begin{example}`.

\exdent *line-of-text* Remove any indentation a line might have. See section exdent in Undoing the Indentation of a Line.

\expansion{} Indicate the result of a macro expansion to the reader with a special glyph: '↦'. See section 5.7.2 [expansion], page 55.

\file{*filename***}** Highlight the name of a file or directory. See section file in `\file`.

\finalout Prevent LaTeX from printing large black warning rectangles beside over-wide lines. See section 16.6 [Overfull Hboxes], page 143.

\findex{*entry***}** Add *entry* to the index of functions. See section Index Entries in Defining the Entries of an Index.

\begin{flushleft} Left justify every line but leave the right end ragged. Leave font as is. Pair with `\end{flushleft}`. See section flushleft & flushright in `\begin{flushleft}` and `\begin{flushright}`.

\begin{flushright} Right justify every line but leave the left end ragged. Leave font as is. Pair with `\end{flushright}`. See section flushleft & flushright in `\begin{flushleft}` and `\begin{flushright}`.

\footnote{*text-of-footnote***}** Enter a footnote. Footnote text is printed at the bottom of the page by LaTeX; Info may format in either 'End Node' or 'Make Node' style. See section 8.1 [Footnotes], page 71.

\footnotestyle{*style***}** Specify an Info file's footnote style, either '`end`' for the end node style or '`separate`' for the separate node style. See section 8.1 [Footnotes], page 71.

\begin{format} Begin a kind of example. Like `\begin{example}` or `\begin{display}`, but do not narrow the margins and do not select the fixed-width font. Pair with `\end{format}`. See section example in `\begin{example}`.

\i{*text***}** Print *text* in *italic* font. No effect in Info. See section 4.2.3 [Fonts], page 42.

\begin{ifinfo} Begin a stretch of text that will be ignored by LaTeX when it typesets the printed manual. The text appears only in the Info file. Pair with `\end{ifinfo}`. See section Conditionals in Conditionally Visible Text.

\begin{iftex} Begin a stretch of text that will not appear in the Info file, but will be processed only by LaTeX. Pair with `\end{iftex}`. See section Conditionals in Conditionally Visible Text.

\begin{ignore} Begin a stretch of text that will not appear in either the Info file or the printed output. Pair with `\end{ignore}`. See section Comments in Comments and Ignored Text.

\include{*filename***}** Incorporate the contents of the file *filename* into the Info file or printed document. See section 9.2 [Include Files], page 73.

\inforef{*node-name***,** [*entry-name* , *info-file-name*}] Make a cross reference to an Info file for which there is no printed manual. See section inforef in Cross references using `\inforef`.

\input{*filename***}** Input the contents of the file *filename* into the Info file or printed document. See section 9.1 [Input Files], page 73.

\item Indicate the beginning of a marked paragraph for `\begin{itemize}` and `\begin{enumerate}` and `\begin{description}` environments.

\begin{itemize} Produce a sequence of indented paragraphs, with a mark inside the left margin at the beginning of each paragraph. Pair with `\end{itemize}`. See section 6.1 [Itemize Environment], page 60.

\kbd{*keyboard-characters*} Indicate text that consists of characters of input to be typed by users. See section kbd in `\kbd`.

\key{*key-name*} Highlight *key-name*, a conventional name for a key on a keyboard. See section key in `\key`.

\kindex{*entry*} Add *entry* to the index of keys. See section Index Entries in Defining the Entries of an Index.

\LaTeX{} Insert the logo LaTeX.

\begin{lisp} Begin an example of Lisp code. Indent text, do not fill, select fixed-width font. Pair with `\end{lisp}`. See section Lisp Example in `\begin{lisp}`.

\begin{menu} Mark the beginning of a menu of nodes in Info. No effect in a printed manual. Pair with `\end{menu}`. See section 11.3 [Menu Environment], page 96.

\minus{} Generate a minus sign. See section minus in `\minus`.

\need{*n*} Start a new page in a printed manual if fewer than *n* mils (thousandths of an inch) remain on the current page. See section need in `\need`.

\node *name, next, previous, up* Define the beginning of a new node in Info, and serve as a locator for references for LaTeX. See section node in `\node`.

\noindent Prevent text from being indented as if it were a new paragraph. See section noindent in `\noindent`.

\nxref{*node-name*, [*entry*], [*topic*], [*info-file*], [*manual*]} Make a reference. In a printed manual, the reference does not start with a 'See'. Follow command with a punctuation mark. Only the first argument is mandatory. See section ref in `\nxref`.

\paragraphindent{*indent*} Indent paragraphs by *indent* number of spaces; delete indentation if the value of *indent* is 0; and do not change indentation if *indent* is `asis`. See section paragraphindent in Paragraph Indenting.

\pindex{*entry*} Add *entry* to the index of programs. See section Index Entries in Defining the Entries of an Index.

\point{} Indicate the position of point in a buffer to the reader with a special glyph: '$\star$'. See section Point Special Glyph in Indicating Point in a Buffer.

\print{} Indicate printed output to the reader with a special glyph: '$\dashv$'. See section 5.7.3 [Print Special Glyph], page 55.

\printindex{*index-name*} Print an alphabetized two-column index in a printed manual or generate an alphabetized menu of index entries for Info. See section 2.9.2 [Printing an Index and Generating Menus], page 28.

\pxref{*node-name*, [*entry*], [*topic*], [*info-file*], [*manual*]} Make a reference that starts with a lower case 'see' in a printed manual. Use within parentheses only. Do not follow command with a punctuation mark. The Info formatting commands automatically insert terminating punctuation as needed, which is why you do not need to insert punctuation. Only the first argument is mandatory. See section pxref in `\pxref`.

\begin{quotation} Narrow the margins to indicate text that is quoted from another real or imaginary work, and indent following text. Write command on a line of its own. Pair with `\end{quotation}`. See section 5.1 [Quotations], page 48.

\begin{quote} Narrow the margins to indicate text that is quoted from another real or imaginary work. Write command on a line of its own. Pair with `\end{quote}`. See section 5.1 [Quotations], page 48.

\r{*text*} Print *text* in roman font. No effect in Info. See section 4.2.3 [Fonts], page 42.

\refill In Info, refill and indent the paragraph after all the other processing has been done. No effect on LaTeX, which always refills. See section 7.4 [Refilling Paragraphs], page 68.

\result{} Indicate the result of an expression to the reader with a special glyph: '⇒'. See section result in `\result`.

\samp{*text*} Highlight *text* that is a literal example of a sequence of characters. Used for single characters, for statements and often for entire shell commands. See section samp in `\code`.

\scap{*text*} Set *text* in a printed output in THE SMALL CAPS FONT and set text in the Info file in uppercase letters. See section 4.2.2 [Smallcaps], page 41.

\setfilename{*info-file-name*} Provide a name for the Info file. See section Conventions in General Syntactic Conventions.

\begin{smalllisp} Begin an example of Lisp code. Indent text, do not fill, select a small fixed-width font. Pair with `\end{lisp}`. See section Lisp Example in `\begin{lisp}`.

\title{*title*} Provide a title for page headers in a printed manual, and for the titlepage. See section Conventions in General Syntactic Conventions.

\begin{same} Hold text together that must appear on one printed page. Pair with `\end{same}`. Not relevant to Info. See section group in `\begin{same}`.

\begin{smallexample} Indent text to indicate an example. Do not fill, select fixed-width font. In `\smallbook` format, print text in a smaller font than with `\begin{example}`. Pair with `\end{smallexample}`. See section 5.4 [Examples and Verbatim], page 50.

\smalllisp Begin an example of Lisp code. Indent text, do not fill, select fixed-width font. In `\smallbook` format, print text in a smaller font. Pair with `\end{smalllisp}`. See section 5.4 [Examples and Verbatim], page 50.

\sp{*n*} Skip *n* blank lines. See section sp in `\sp`.

\strong{*text*} Emphasize *text* by typesetting it in a **bold** font for the printed manual and by surrounding it with asterisks for Info. See section emph & strong in Emphasizing Text.

\subsection{*title*} Begin a subsection within a section. In a printed manual, the subsection title is numbered and appears in the table of contents. In Info, the title is underlined with hyphens. See section 3.6 [Subsection], page 34.

\subsubsection{*title*} Begin a subsubsection within a subsection. In a printed manual, the subsubsection title is numbered and appears in the table of contents. In Info, the title is underlined with periods. See section 3.7 [Subsubsection], page 34.

\syncodeindex{*from-index*}{*into-index*} Merge the index named in the first argument into the index named in the second argument, printing the entries from the first index in `\code` font. See section 13.3 [Combining Indices], page 116.

\synindex{*from-index*}{*into-index*} Merge the index named in the first argument into the index named in the second argument. Do not change the font of *from-index* entries. See section 13.3 [Combining Indices], page 116.

\t{*text*} Print *text* in a `fixed-width` font. No effect in Info. See section 4.2.3 [Fonts], page 42.

\tableofcontents Print a complete table of contents. Has no effect in Info, which uses menus instead. See section 2.6 [Generating a Table of Contents], page 22.

\TeX{} Insert the logo TEX.

\begin{tex} Enter LATEX completely. Pair with `\end{tex}`. See section 5.8.1 [Using Ordinary LaTeX Commands], page 58.

\tindex{*entry*} Add *entry* to the index of data types. See section Index Entries in Defining the Entries of an Index.

\title{*title*} In a printed manual, set a title in a larger than normal font and underline it with a black rule. See section 2.5.1 [Titlepage], page 20.

\today{} Insert the current date. See section 2.4.4 [Custom Headings], page 19.

\unnumbered{*title*} In a printed manual, begin a chapter that appears without chapter numbers of any kind. The title appears in the table of contents of a printed manual. In Info, the title is underlined with asterisks. See section 3.3 [Chapter], page 33.

\unnumberedsec{*title*} In a printed manual, begin a section that appears without section numbers of any kind. The title appears in the table of contents of a printed manual. In Info, the title is underlined with equal signs. See section 3.5 [Section], page 33.

\unnumberedsubsec{*title*} In a printed manual, begin an unnumbered subsection within a chapter. The title appears in the table of contents of a printed manual. In Info, the title is underlined with hyphens. See section 3.6 [Subsection], page 34.

\unnumberedsubsubsec{*title*} In a printed manual, begin an unnumbered subsubsection within a chapter. The title appears in the table of contents of a printed manual. In Info, the title is underlined with periods. See section 3.7 [Subsubsection], page 34.

\var{*metasyntactic-variable*} Highlight a metasyntactic variable, which is something that stands for another piece of text. Thus, in this entry, the word *metasyntactic-variable* is highlighted with `\var`. See section var in Indicating Metasyntactic Variables.

\vindex{*entry*} Add *entry* to the index of variables. See section Index Entries in Defining the Entries of an Index.

\vspace{*amount*} In a printed manual, insert whitespace so as to push text on the remainder of the page towards the bottom of the page. Used in formatting the copyright page with the argument '`0pt plus 1filll`'. (Note spelling of '`filll`'.) `\vspace` is ignored for Info. See section 2.5.2 [The Copyright Page and Printed Permissions], page 21.

\w{*text*} Prevent *text* from being split across two lines. Do not end a paragraph that uses `\w` with an `\refill` command. In the LaTeXinfo file, keep *text* on one line. See section w in `\w`.

\xref{*node-name*, [*entry*], [*topic*], [*info-file*], [*manual*]} Make a reference that starts with 'See' in a printed manual. Follow command with a punctuation mark. Only the first argument is mandatory. See section xref in `\xref`.

Bibliography

[Lam86] Leslie Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, Reading, MA, 1986.

[Sta86] Richard Stallman. *The GNU Emacs Manual.* The Free Software Foundation, 675 Massachusetts Ave., Cambridge MA, 02139, 1986.

[Ste90] Guy˜L. Steele. *Common Lisp - the Language II.* Addison–Wesley, Reading, MA, 1990.

Command Index

This is an alphabetical list of all the \-commands and several variables. To make the list easier to use, the commands are listed without their preceding '\'.

Concept Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W