This manual is for GNU Texinfo (version 7.0.3, 26 March 2023), a documentation system that can produce both online information and a printed manual from a single source using semantic markup.


Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Published by the Free Software Foundation
51 Franklin St, Fifth Floor
Boston, MA 02110-1301
USA
ISBN 1-882114-67-1
## Short Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texinfo Copying Conditions</td>
<td>1</td>
</tr>
<tr>
<td>Overview of Texinfo</td>
<td>2</td>
</tr>
<tr>
<td>Writing a Texinfo File</td>
<td>9</td>
</tr>
<tr>
<td>Nodes</td>
<td>27</td>
</tr>
<tr>
<td>Chapter Structuring</td>
<td>37</td>
</tr>
<tr>
<td>Cross-references</td>
<td>43</td>
</tr>
<tr>
<td>Marking Text, Words and Phrases</td>
<td>54</td>
</tr>
<tr>
<td>Quotations and Examples</td>
<td>65</td>
</tr>
<tr>
<td>Lists and Tables</td>
<td>75</td>
</tr>
<tr>
<td>Special Displays</td>
<td>83</td>
</tr>
<tr>
<td>Indices</td>
<td>90</td>
</tr>
<tr>
<td>Special Insertions</td>
<td>97</td>
</tr>
<tr>
<td>Forcing and Preventing Breaks</td>
<td>113</td>
</tr>
<tr>
<td>Definition Commands</td>
<td>117</td>
</tr>
<tr>
<td>Internationalization</td>
<td>130</td>
</tr>
<tr>
<td>Conditionally Visible Text</td>
<td>133</td>
</tr>
<tr>
<td>Defining New Texinfo Commands</td>
<td>143</td>
</tr>
<tr>
<td>Include Files</td>
<td>152</td>
</tr>
<tr>
<td>Formatting and Printing with TeX</td>
<td>154</td>
</tr>
<tr>
<td><strong>texi2any: The Translator for Texinfo</strong></td>
<td>164</td>
</tr>
<tr>
<td>Creating and Installing Info Files</td>
<td>194</td>
</tr>
<tr>
<td>Generating HTML</td>
<td>203</td>
</tr>
<tr>
<td>@-Command Details</td>
<td>214</td>
</tr>
<tr>
<td>Tips and Hints</td>
<td>238</td>
</tr>
<tr>
<td>Sample Texinfo Files</td>
<td>243</td>
</tr>
<tr>
<td>Using Texinfo Mode</td>
<td>247</td>
</tr>
<tr>
<td>Global Document Commands</td>
<td>269</td>
</tr>
<tr>
<td>Info Format Specification</td>
<td>278</td>
</tr>
<tr>
<td>GNU Free Documentation License</td>
<td>284</td>
</tr>
<tr>
<td>Command and Variable Index</td>
<td>292</td>
</tr>
<tr>
<td>General Index</td>
<td>298</td>
</tr>
</tbody>
</table>
Table of Contents

Texinfo Copying Conditions ...................................... 1

1 Overview of Texinfo ............................................. 2
  1.1 Reporting Bugs ..................................................... 2
  1.2 Output Formats ..................................................... 3
  1.3 Info Files .......................................................... 4
  1.4 Printed Books ....................................................... 5
  1.5 Adding Output Formats ............................................ 5
  1.6 History .............................................................. 6

2 Writing a Texinfo File ........................................... 9
  2.1 General Syntactic Conventions ................................. 9
  2.2 Comments ........................................................... 10
  2.3 What a Texinfo File Must Have ................................ 10
  2.4 Short Sample ....................................................... 11
  2.5 Texinfo File Header ............................................... 12
    2.5.1 The First Line of a Texinfo File ....................... 13
    2.5.2 @setfilename: Set the Output File Name .............. 13
    2.5.3 @settitle: Set the Document Title .................... 14
    2.5.4 Preamble ...................................................... 14
    2.5.5 Start and End of Header for Emacs .................... 15
  2.6 Directory Category .............................................. 16
  2.7 Document Permissions .......................................... 16
    2.7.1 @copying: Declare Copying Permissions ............... 17
    2.7.2 @insertcopying: Include Permissions Text ............ 18
  2.8 Title and Copyright Pages .................................... 18
    2.8.1 @titlepage .................................................. 19
    2.8.2 @title, @subtitle, and @author ......................... 19
    2.8.3 @titlefont, @center, and @sp .......................... 20
    2.8.4 Copyright Page ............................................. 21
    2.8.5 Heading Generation ........................................ 22
  2.9 Generating a Table of Contents ............................. 22
  2.10 The ‘Top’ Node and Master Menu ............................ 23
    2.10.1 Parts of a Master Menu .................................. 24
  2.11 The Body of the Document ................................... 24
  2.12 Ending a Texinfo File ......................................... 26

3 Nodes .................................................................. 27
  3.1 Writing a @node Line ............................................. 27
  3.2 Choosing Node Names ............................................. 28
  3.3 @node Line Requirements ....................................... 29
  3.4 The First Node .................................................... 30
3.5 The **top** Sectioning Command .................................. 31
3.6 Texinfo Document Structure ........................................ 31
3.7 Node and Menu Illustration .......................................... 32
3.8 Menus ......................................................................... 33
  3.8.1 Writing a Menu .................................................... 33
  3.8.2 A Menu Example .................................................... 34
  3.8.3 Menu Location ....................................................... 35
  3.8.4 The Parts of a Menu ............................................... 35
  3.8.5 Less Cluttered Menu Entry ...................................... 36
  3.8.6 Referring to Other Info Files ................................. 36

4 Chapter Structuring ...................................................... 37
4.1 Tree Structure of Sections ............................................ 37
4.2 Structuring Command Types .......................................... 37
4.3 **chapter**: Chapter Structuring ................................... 38
4.4 **unnumbered, appendix**: Chapters with Other Labeling .... 39
4.5 **majorheading, chapheading**: Chapter-level Headings ....... 39
4.6 **section**: Sections Below Chapters ............................... 39
4.7 **unnumberedsec, appendixsec, heading** ....................... 40
4.8 **subsection**: Subsections Below Sections ....................... 40
4.9 The **subsection**-like Commands .................................... 40
4.10 **subsubsection** and Other Subsub Commands .................. 41
4.11 **part**: Groups of Chapters ....................................... 41
4.12 Raise/lower Sections: **raisesections** and **lowersections** .. 42

5 Cross-references ......................................................... 43
5.1 Different Cross-reference Commands ............................... 43
5.2 Parts of a Cross-reference ........................................... 43
5.3 **xref** with One Argument ......................................... 44
5.4 **xref** with Two Arguments ....................................... 45
5.5 **xref** with Three Arguments ..................................... 45
5.6 **xref** with Four and Five Arguments ............................ 46
5.7 Referring to a Manual as a Whole ................................. 47
5.8 **xref** ................................................................. 48
5.9 **ref** ....................................................................... 48
5.10 **pxref** ................................................................. 49
5.11 **anchor**: Defining Arbitrary Cross-reference Targets ....... 49
5.12 **inforef**: Cross-references to Info-only Material .......... 50
5.13 **url**{url[, text][, replacement]} ................................ 50
  5.13.1 **url** Examples .................................................. 50
  5.13.2 URL Line Breaking .............................................. 51
  5.13.3 **url** PDF Output Format ..................................... 52
  5.13.4 PDF Colors ....................................................... 52
5.14 **cite**{reference} ..................................................... 52
6  Marking Text, Words and Phrases ............ 54
6.1  Indicating Definitions, Commands, etc. ............. 54
  6.1.1  Highlighting Commands are Useful ............. 54
  6.1.2  @code{sample-code} ....................... 55
  6.1.3  @kbd{keyboard-characters} ................. 56
  6.1.4  @key{key-name} ....................... 57
  6.1.5  @samp{text} ....................... 57
  6.1.6  @verb{chartextchar} ....................... 58
  6.1.7  @var{metasyntactic-variable} ............... 58
  6.1.8  @env{environment-variable} ............... 59
  6.1.9  @file{file-name} ....................... 59
  6.1.10 @command{command-name} ............... 60
  6.1.11 @option{option-name} ............... 60
  6.1.12 @dfn{term} .................................. 60
  6.1.13 @abbr{abbreviation, meaning} .......... 60
  6.1.14 @acronym{acronym, meaning} .......... 61
  6.1.15 @indicateurl{uniform-resource-locator} .......... 62
  6.1.16 @email{email-address, displayed-text} .......... 62
6.2  Emphasizing Text .................................. 62
  6.2.1  @emph{text} and @strong{text} .......... 62
  6.2.2  @sc{text}: The Small Caps Font .......... 63
  6.2.3  Fonts for Printing .......................... 63
7  Quotations and Examples ............................ 65
  7.1  Block Enclosing Commands ....................... 65
  7.2  @quotation: Block Quotations ..................... 66
  7.3  @indentedblock: Indented text blocks ............ 67
  7.4  @example: Example Text ..................... 67
  7.5  @verbatim: Literal Text ..................... 68
  7.6  @lisp: Marking a Lisp Example ............. 69
  7.7  @display: Examples Using the Text Font .......... 69
  7.8  @format: Examples Using the Full Line Width .......... 69
  7.9  @exdent: Undoing a Line’s Indentation ............ 70
  7.10 @flushleft and @flushright .................... 70
  7.11 @raggedright: Ragged Right Text ............. 71
  7.12 @noindent: Omitting Indentation ............. 72
  7.13 @indent: Forcing Indentation ............. 72
  7.14 @cartouche: Rounded Rectangles ............. 73
  7.15 @small... Block Commands .................... 73
11.3.1 Multiple Spaces ........................................ 100
11.3.2 Not Ending a Sentence .................................. 100
11.3.3 Ending a Sentence ...................................... 101
11.3.4 \frenchspacing \textit{val}: Control Sentence Spacing .... 102
11.3.5 \dms{dimension}: Format a Dimension ................ 102
11.4 Inserting Accents ........................................... 103
11.5 Inserting Quotation Marks .................................. 104
11.6 \texttt{\textbackslash sub} and \texttt{\textbackslash sup}: Inserting Subscripts and Superscripts .... 105
11.7 \texttt{\textbackslash math} and \texttt{\textbackslash displaymath}: Formatting Mathematics .... 105
11.8 Glyphs for Text ............................................. 106
11.8.1 \TeX{\{}} (\TeX) and \LaTeX{\{}} (\LaTeX) ............. 106
11.8.2 \copyright{\{}} (©) .................................... 107
11.8.3 \registered{\{}} (®) .................................. 107
11.8.4 \textit{...} and \textit{...}: Inserting Relations .......... 107
11.8.5 \textbullet{} .......................................... 107
11.8.6 \euro{} (€): Euro Currency Symbol .................... 107
11.8.7 \pounds{} (£): Pounds Sterling ......................... 107
11.8.8 \textdegree{} (°): Degrees Symbol .................... 107
11.8.9 \textbf{\textit{...}}: Inserting a Minus Sign .......... 107
11.8.10 \textbf{\textit{...}} and \textbf{\textit{...}}: Inserting Relations ....... 107
11.9 Glyphs for Programming .................................... 108
11.9.1 Glyphs Summary ....................................... 108
11.9.2 \result{} (⇒): Result of an Expression .............. 108
11.9.3 \expansion{} (↦→): Indicating an Expansion ........ 109
11.9.4 \print{} (-): Indicating Generated Output .......... 109
11.9.5 \error{} ([error]): Indicating an Error Message ... 110
11.9.6 \equiv{} (=): Indicating Equivalence ................ 110
11.9.7 \point{} (⋆): Indicating Point in a Buffer .......... 110
11.9.8 Click Sequences ........................................ 111
11.10 Inserting Unicode: \U .................................... 112
12 Forcing and Preventing Breaks .................. 113
12.1 Break Commands ............................................. 113
12.2 \textbf{\textit{...}} and \textbf{\textit{...}}: Generate and Allow Line Breaks ......... 113
12.3 \textbf{\textit{...}} and \textbf{\textit{...}}: Hyphenation in Printed Output .... 114
12.4 \allowcodebreaks: Control Line Breaks in \texttt{\textbackslash code} .... 114
12.5 \texttt{\textbackslash w{\{text}}}: Prevent Line Breaks .......... 114
12.6 \texttt{\textbackslash tie{}}: Inserting an Unbreakable Space .... 115
12.7 \texttt{\textbackslash sp \textit{n}}: Insert Blank Lines .......... 115
12.8 \texttt{\textbackslash page}: Start a New Page ................. 115
12.9 \texttt{\textbackslash group}: Prevent Page Breaks .......... 115
12.10 \texttt{\textbackslash need \textit{mils}}: Prevent Page Breaks .......... 116
13 Definition Commands .................. 117
  13.1 The Template for a Definition ....................... 117
  13.2 Definition Command Continuation Lines ................. 118
  13.3 Optional and Repeated Arguments .................. 118
  13.4 Omitting the Space After a Definition Name ........... 119
  13.5 @deffnx, et al.: Two or More ‘First’ Lines .......... 119
  13.6 The Definition Commands ......................... 119
      13.6.1 Functions and Similar Entities ................. 119
      13.6.2 Functions in Typed Languages .................. 120
      13.6.3 Variables and Similar Entities ................. 122
      13.6.4 Variables in Typed Languages .................. 123
      13.6.5 Data Types ...................................... 124
      13.6.6 Object-Oriented Programming .................... 124
          13.6.6.1 Object-Oriented Variables ................. 124
          13.6.6.2 Object-Oriented Methods ................. 126
  13.7 Conventions for Writing Definitions ................ 127
  13.8 A Sample Function Definition .................... 128

14 Internationalization ....................... 130
  14.1 @documentlanguage ll[_cc]: Set the Document Language ... 130
  14.2 @documentencoding enc: Set Input Encoding .......... 131

15 Conditionally Visible Text .................. 133
  15.1 Conditional Commands .................................. 133
  15.2 Conditional Not Commands .......................... 134
  15.3 Raw Formatter Commands ........................... 135
  15.4 Inline Conditionals: @inline, @inlineifelse, @inlineraw .. 136
  15.5 Flags: @set, @clear, conditionals, and @value .......... 137
      15.5.1 @set and @value .................................. 137
      15.5.2 @ifset and @ifclear ............................ 138
      15.5.3 @inlineifset and @inlineifclear ............. 139
      15.5.4 @value Example ................................... 139
  15.6 Testing for Texinfo Commands: 
      @ifcommanddefined, @ifcommandnotdefined ............. 140
  15.7 Conditional Nesting ................................ 141

16 Defining New Texinfo Commands ............ 143
  16.1 Defining Macros ...................................... 143
  16.2 Invoking Macros .................................... 144
  16.3 Macro Details and Caveats .......................... 146
  16.4 ‘@alias new=existing’ ............................... 146
  16.5 @definfoenclose: Customized Highlighting ............. 149
  16.6 External Macro Processors: Line Directives .......... 150
      16.6.1 ‘#line’ Directive .............................. 150
      16.6.2 ‘#line’ and \TeX ......................... 151
16.6.3 ‘#line’ Syntax Details........................................ 151

17 Include Files .................................................. 152
  17.1 How to Use Include Files.................................... 152
  17.2 Sample File with @include................................. 152
  17.3 @verbatiminclude file: Include a File Verbatim........ 153

18 Formatting and Printing with T\textsc{e}X .................... 154
  18.1 Use T\textsc{e}X.................................................. 154
  18.2 Format with texi2dvi or texi2pdf.......................... 154
  18.3 Format with tex/texindex.................................... 156
    18.3.1 Formatting Partial Documents.......................... 157
    18.3.2 Details of texindex.................................... 158
  18.4 Print with lpr from Shell................................. 158
  18.5 Formatting and Printing with Emacs....................... 159
    18.5.1 Printing From an Emacs Shell......................... 159
    18.5.2 Formatting and Printing in Texinfo Mode.............. 159
    18.5.3 Using the Local Variables List....................... 161
  18.6 \textsc{t}ex Formatting Requirements Summary.............. 161
  18.7 Preparing for \textsc{t}ex.................................... 161
    18.8 Overfull “hboxes”....................................... 162

19 texi2any: The Translator for Texinfo ............. 164
  19.1 Invoking texi2any from a Shell........................... 164
  19.2 Environment Variables Recognized by texi2any.......... 170
  19.3 texi2any Printed Output.................................. 171
  19.4 Customization Variables.................................. 172
    19.4.1 Customization Variables for @-Commands.............. 172
    19.4.2 Customization Variables and Options............... 173
    19.4.3 HTML Customization Variables......................... 174
    19.4.4 MathJax Customization Variables..................... 180
    19.4.5 latex2html Customization Variables............... 181
    19.4.6 tex4ht Customization Variables..................... 181
    19.4.7 Other Customization Variables....................... 182
  19.5 Internationalization of Document Strings.............. 188
  19.6 Invoking pod2texi: Convert Pod to Texinfo.............. 189
    19.6.1 pod2texi............................................. 190
  19.7 texi2html: Ancestor of texi2any......................... 192
Appendix D  Using Texinfo Mode ............... 247
  D.1  Texinfo Mode Overview .......................... 247
  D.2  The Usual GNU Emacs Editing Commands .......... 247
  D.3  Inserting Frequently Used Commands ............. 248
  D.4  Showing the Sectioning Structure of a File ....... 250
    D.4.1  Using texinfo-show-structure ................. 250
    D.4.2  Using occur .................................... 251
  D.5  Updating Nodes and Menus ......................... 252
    D.5.1  The Updating Commands ........................ 252
    D.5.2  Updating Requirements .......................... 254
    D.5.3  Update Outer File and Include Files .......... 255
    D.5.4  Include Files Requirements .................... 256
    D.5.5  Other Updating Commands ....................... 256
  D.6  Formatting for Info ................................ 257
    D.6.1  Running texi2any/makeinfo Within Emacs ...... 257
    D.6.2  The texinfo-format... Commands ................ 258
  D.7  Printing ........................................... 259
  D.8  Texinfo Mode Summary ................................ 260
  D.9  Direct Formatting of Info files .................... 262
    D.9.1  Tagifying a File ................................ 262
    D.9.2  Splitting a File Manually ...................... 263
  D.10 Catching Mistakes .................................. 263
    D.10.1  texi2any Preferred ............................ 263
    D.10.2  Catching Errors with Info Formatting ........ 263
    D.10.3  Debugging with TeX ............................. 265
    D.10.4  Finding Badly Referenced Nodes ............... 266
      D.10.4.1  Using info-validate ......................... 267
      D.10.4.2  Creating an Unsplit File and Adding a Tag Table.. 267
  D.11 Batch Formatting .................................... 268
Appendix E  Global Document Commands ................ 269
  E.1  @setchapternewpage: Blank Pages Before Chapters .... 269
  E.2  Page Headings ...................................... 270
    E.2.1  The @headings Command .......................... 270
    E.2.2  Standard Heading Formats ....................... 271
    E.2.3  How to Make Your Own Headings .................. 271
  E.3  @paragraphindent: Controlling Paragraph Indentation .... 274
  E.4  @firstparagraphindent: Indenting After Headings .... 274
  E.5  @exampleindent: Environment Indenting .............. 275
  E.6  @smallbook: Printing “Small” Books ................. 275
  E.7  Printing on A4 Paper ................................ 275
  E.8  @pagesizes [width][, height]: Custom Page Sizes .... 276
  E.9  Microtypography .................................... 276
  E.10 Magnification ...................................... 276
Appendix F  Info Format Specification ........ 278
  F.1 Info Format: A Whole Manual ....................... 278
  F.2 Info Format: Preamble ................................ 278
  F.3 Info Format: Indirect Table .......................... 279
  F.4 Info Format: Tag Table ................................. 280
  F.5 Info Format: Local Variables .......................... 280
  F.6 Info Format: Regular Nodes ............................ 280
  F.7 Info Format: Menu ...................................... 281
  F.8 Info Format: Image ...................................... 281
  F.9 Info Format: Printindex ................................. 282
  F.10 Info Format: Cross-reference .......................... 283

Appendix G  GNU Free Documentation License .... 284

Command and Variable Index ...................... 292

General Index ........................................... 298
Texinfo Copying Conditions

GNU Texinfo is free software; this means that everyone is free to use it and free to redistribute it on certain conditions. Texinfo is not in the public domain; it is copyrighted and there are restrictions on its 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 Texinfo 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 Texinfo, 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 Texinfo 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 Texinfo. 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 Texinfo are found in the General Public Licenses that accompany them. This manual is covered by the GNU Free Documentation License (see Appendix G [GNU Free Documentation License], page 284).
1 Overview of Texinfo

Texinfo is a documentation system that uses a single source file to produce both online information and printed output. This means that instead of writing several different documents, one for each output format, you need only write one document.

Using Texinfo, you can create a printed document (via the TeX typesetting system) in PDF format, including chapters, sections, cross-references, and indices. From the same Texinfo source file, you can create HTML output suitable for use with a web browser, you can create an Info file for use in GNU Emacs or other Info-reading programs, and also create DocBook, EPUB 3 or HTML files.

A Texinfo source file is a plain text file containing text interspersed with @-commands (words preceded by an ‘@’) that tell the Texinfo processors what to do. Texinfo’s markup commands are almost entirely semantic; that is, they specify the intended meaning of text in the document, rather than physical formatting instructions.

GNU Emacs has a special mode, called Texinfo mode, that provides various Texinfo-related features. (See Appendix D [Texinfo Mode], page 247.)

Texinfo was devised specifically for the purpose of writing software documentation and manuals. If you want to write a good manual for your program, Texinfo has many features which we hope will make your job easier. However, Texinfo is not intended to be a general-purpose formatting program. It provides almost no commands for controlling the final formatting, so may be inappropriate for your needs if you if you want to lay out a newspaper, devise a glossy magazine ad, or follow the exact formatting requirements of a publishing house.

Spell “Texinfo” with a capital “T” and the other letters in lowercase. The first syllable of “Texinfo” is pronounced like “speck”, not “hex”. This odd pronunciation is derived from the pronunciation of TeX. Pronounce TeX as if the ‘X’ were the last sound in the name ‘Bach’. In the word TeX, the ‘X’ is, rather than the English letter “ex”, actually the Greek letter “chi”.

Texinfo is the official documentation format of the GNU project. More information, including manuals for GNU packages, is available at the GNU documentation web page (http://www.gnu.org/doc/).

1.1 Reporting Bugs

We welcome bug reports and suggestions for any aspect of the Texinfo system: programs, documentation, installation, etc. Please email them to bug-texinfo@gnu.org. You can get the latest version of Texinfo via its home page, http://www.gnu.org/software/texinfo.

For bug reports, please include enough information for the maintainers to reproduce the problem. Generally speaking, that means:

- The version number of Texinfo and the program(s) or manual(s) involved.
- The contents of any input files necessary to reproduce the bug.
- Precisely how you ran any program(s) involved.
- A description of the problem and samples of any erroneous output.
- Hardware and operating system names and versions.
- Anything else that you think would be helpful.

When in doubt whether something is needed or not, include it. It’s better to include too much than to leave out something important.

It is critical to send an actual input file that reproduces the problem.

Any problems with the Info reader in Emacs should be reported to the Emacs developers: see Section “Bugs” in The GNU Emacs Manual.

Patches are welcome; if possible, please make them with ‘diff -c’, ‘diff -u’ (see Comparing and Merging Files) or ‘git diff’ and include ChangeLog entries (see Section “Change Log” in GNU Coding Standards), and follow the existing coding style.

### 1.2 Output Formats

Here is an overview of the output formats currently supported by Texinfo.

**Info**  
(Generated via texi2any.) Info format is mostly a plain text transliteration of the Texinfo source. It adds a few control characters to provide navigational information for cross-references, indices, and so on. The Emacs Info subsystem (see Info), and the standalone info program (see GNU Info), among others, can read these files. See Section 1.3 [Info Files], page 4, and Chapter 20 [Creating and Installing Info Files], page 194.

**Plain text**  
(Generated via texi2any --plaintext.) This is almost the same as Info output with the navigational control characters are omitted.

**HTML**  
(Generated via texi2any --html.) HTML, standing for Hyper Text Markup Language, is the language for writing documents on the World Wide Web. Web browsers can render this language online. There are many versions of HTML, both different standards and browser-specific variations. texi2any uses a subset of the language that can be interpreted by any common browser, intentionally not using many newer or less widely-supported tags. Although the native output is thus rather plain, it can be customized at various levels, if desired. See Chapter 21 [Generating HTML], page 203.

**EPUB 3**  
(Generated via texi2any --epub3.) EPUB is a format designed for reading electronic books on portable devices. It is a derivative of HTML. The format was developed by the International Digital Publishing Forum (IDPF), which is now part of the World Wide Web Consortium (W3C). The latest major revision, EPUB 3, dates from 2011.

**DVI**  
(Generated via texi2dvi.) The DeVIce Independent binary format is output by the TeX typesetting program (http://tug.org). This is then read by a DVI ‘driver’, which knows the actual device-specific commands that can be viewed or printed, notably Dvips for translation to PostScript (see Dvips) and Xdvi for viewing on an X display (http://sourceforge.net/projects/xdvi/). See Chapter 18 [Hardcopy with TeX], page 154. (Be aware that the Texinfo language is very different from TeX’s usual languages: plain TeX, LATEX, ConTEXt, etc.)

**PostScript**  
(Generated via texi2dvi --ps.) PostScript is a page description language that became widely used around 1985 and is still used today. https://en.
wikipedia.org/wiki/PostScript gives a basic description and more preferences. By default, Texinfo uses the dvips program to convert TeX’s DVI output to PostScript. See Dvips.

PDF (Generated via texi2dvi --pdf or texi2pdf.) This format was developed by Adobe Systems for portable document interchange, based on their previous PostScript language. It can represent the exact appearance of a document, including fonts and graphics, and supporting arbitrary scaling. It is intended to be platform-independent and easily viewable, among other design goals; https://en.wikipedia.org/wiki/Portable_Document_Format and http://tug.org/TUGboat/tb22-3/tb72beebe-pdf.pdf have some background. By default, Texinfo uses the pdftex program, an extension of TeX, to output PDF; see http://tug.org/applications/pdftex. See [PDF Output], page 154.

\LaTeX{} (Generated via texi2any --latex.) This is a typesetting system built on top of \TeX{}. It was originally released by Leslie Lamport in 1984. \LaTeX{} adds more definitions to those of \TeX{} and has a wide range of packages built on it. \LaTeX{} is ubiquitous in academic literature. The current version of \LaTeX{} is under active development; more information is available online at https://www.latex-project.org/.

The \LaTeX{} output can be further processed into DVI, PostScript or PDF. In theory, the \LaTeX{} output should allow for much more customizability of the output than would be possible with the plain \TeX{} implementation of Texinfo.

DocBook (Generated via texi2any --docbook.) This is an XML-based format, primarily for technical documentation. It therefore bears some resemblance, in broad outline, to Texinfo. See http://www.docbook.org. Various converters from DocBook to Texinfo have also been developed; see the Texinfo web pages.

XML (Generated via texi2any --xml.) The texi2any XML output, unlike all the other output formats, is a transliteration of the Texinfo source, rather than finished output. Texinfo XML files cannot be viewed in web browsers or other programs.

XML is a generic syntax specification usable for any sort of content. (A reference is at http://www.w3.org/XML.) The purpose of the Texinfo XML output is to allow further processing by XML tools. The output syntax is defined in an XML DTD, which is contained in a file texinfo.dtd included in the Texinfo source distribution.

The Texinfo source distribution includes a utility script txixml2texi to do a backward transformation to recreate the original Texinfo content (except for Texinfo macros and conditionals).

1.3 Info Files

As mentioned above, Info format is mostly a plain text transliteration of the Texinfo source, with the addition of a few control characters to separate nodes and provide navigational information, so that Info-reading programs can operate on it.
Info files are nearly always created by processing a Texinfo source document. \texttt{texi2any}, also known as \texttt{makeinfo}, is the principal command that converts a Texinfo file into an Info file; see Chapter 19 [Generic Translator \texttt{texi2any}], page 164.

Generally, you enter an Info file through a node that by convention is named ‘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. Alternatively, with the standalone Info program, you can specify specific menu items on the command line (see \textit{Info}).

If you want to read through an Info file in sequence, as if it were a printed manual, you can hit \texttt{SPC} repeatedly, or you get the whole file with the advanced Info command \texttt{g *}. (See Section “Advanced Info commands” in \textit{Info}.)

The \texttt{dir} file in the \texttt{info} directory serves as the departure point for the whole Info system. From it, you can reach the ‘Top’ nodes of each of the documents in a complete Info system.

If you wish to refer to an Info file via a URI, you can use the (unofficial) syntax exemplified by the following. This works with Emacs/W3, for example:

\begin{verbatim}
info:emacs#Dissociated%20Press
info:///usr/info/emacs#Dissociated%20Press
info://localhost/usr/info/emacs#Dissociated%20Press
\end{verbatim}

The \texttt{info} program itself does not follow URIs of any kind.

\section*{1.4 Printed Books}

A Texinfo file can be formatted and typeset as a printed book or manual. To do this, you need \TeX{}, a sophisticated typesetting program written by Donald Knuth of Stanford University. It is not part of the Texinfo distribution.

Texinfo provides a file \texttt{texinfo.tex} that contains the definitions that \TeX{} uses when it typesets a Texinfo file. You can get the latest version of \texttt{texinfo.tex} from the Texinfo home page, \url{http://www.gnu.org/software/texinfo/}.

\TeX{} is very powerful and has a great many features. However, because a Texinfo 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 Texinfo supports are necessarily limited.

See Chapter 18 [Hardcopy with \TeX{}], page 154, for more information on processing a manual with \TeX{}.

\section*{1.5 Adding Output Formats}

The output formats in the previous sections handle a wide variety of usage, but of course there is always room for more.

If you are a programmer and would like to contribute to the GNU project by implementing additional output formats for Texinfo, that would be excellent. The way to do this that
would be most useful is to write a new back-end for \texttt{texi2any}, our reference implementation of a Texinfo parser; it creates a tree representation of the Texinfo input that you can use for the conversion. The documentation in the source file \texttt{tp/Texinfo/Convert/Converter.pm} is a good place to start (see Section “Texinfo::Convert::Converter” in \textit{Texinfo modules documentation}). See Chapter 19 [Generic Translator \texttt{texi2any}], page 164.

Another viable approach is use the Texinfo XML output from \texttt{texi2any} as your input. This XML is an essentially complete representation of the input, but without the Texinfo syntax and option peculiarities, as described above.

If you still cannot resist the temptation of writing a new program that reads Texinfo source directly, let us give some more caveats: please do not underestimate the amount of work required. Texinfo is by no means a simple language to parse correctly, and remains under development, so you would be committing to an ongoing task. You are advised to check that the tests of the language that come with \texttt{texi2any} give correct results with your new program.

From time to time, proposals are made to generate traditional Unix man pages from Texinfo source. However, because man pages have a strict conventional format, creating a good man page requires a completely different source from that needed for the typical Texinfo applications of writing a good user tutorial and/or a good reference manual. This makes generating man pages incompatible with the Texinfo design goal of not having to document the same information in different ways for different output formats. You might as well write the man page directly.

As an alternative way to support man pages, you may find the program \texttt{help2man} to be useful. It generates a traditional man page from the ‘\texttt{--help}’ output of a program. In fact, the man pages for the programs in the Texinfo distribution are generated with this. It is GNU software written by Brendan O’Dea, available from \url{http://www.gnu.org/software/help2man}.

1.6 History

Richard M. Stallman invented the Texinfo format, wrote the initial processors, and created Edition 1.0 of this manual. Robert J. Chassell greatly revised and extended the manual, starting with Edition 1.1. Brian Fox was responsible for the standalone Texinfo distribution until version 3.8. Karl Berry continued maintenance from Texinfo 3.8 (manual edition 2.22), and Gavin Smith has continued maintenance since Texinfo 6.0.

Beginnings

Richard Stallman included an on-line, hypertext help system called Info in the original implementation of Emacs (in 1975/6). Stallman had been inspired after seeing a presentation a couple of years earlier on the “NLS” hypertext system of Douglas Engelbart.

In another development, in the 1970’s at CMU, Brian Reid developed a program and format named Scribe to mark up documents for printing. It used the '@' character to introduce commands, as Texinfo does. Much more consequentially, it strove to describe document contents rather than formatting, an idea wholeheartedly adopted by Texinfo.

Meanwhile, people at MIT developed another format called Bolio. Richard Stallman (RMS) worked on converting Bolio to use \TeX as its typesetting language, resulting in Bo\TeX. The earliest Bo\TeX version seems to have been 0.02 on October 31, 1984.
BoT\TeX\ could only be used as a markup language for documents to be printed, not for online documents. RMS combined BoT\TeX\ and Info to create Texinfo, a mark-up language for text that is intended to be read both online and as printed hard copy.

The original translator to create Info was written (primarily by RMS and Bob Chassell) in Emacs Lisp, namely the texinfo-format-buffer and other functions. In the early 1990s, Brian Fox reimplemented the conversion program in C, now called makeinfo, as well as the standalone info program.

Reimplementing in Perl

In 2012, the C makeinfo was itself replaced by a Perl implementation generically called texi2any. This version supports the same level of output customization as texi2html, an independent program originally written by Lionel Cons, later with substantial work by many others. The many additional features needed to make texi2html a replacement for makeinfo were implemented by Patrice Dumas. The first, never-released version of texi2any was based on the texi2html code.

That implementation, however, was abandoned in favor of the current program (also written by Patrice Dumas), which parses the Texinfo input into a tree for processing. It inherited the design of customization and other features from texi2html (for more on texi2html compatibility, see Section 19.7 [texi2html], page 192). However, texi2any is a full reimplementation: it constructs a tree-based representation of the input document for all back-ends to work from.

The new Perl program is much slower than the old C program. The speed gap has partially closed since first release, but it may not ever be entirely comparable. So why did we switch? In short, we intend and hope that the present program will be much easier than the previous C implementation of makeinfo to extend to different output styles, back-end output formats, and all other customizations. In more detail:

- HTML customization. Many GNU and other free software packages had been happily using the HTML customization features in texi2html for years. Thus, in effect two independent implementations of the Texinfo language had developed, and keeping them in sync was not simple. Adding the HTML customization possible in texi2html to a C program would have been an enormous effort.

- Unicode, and multilingual support generally, especially of east Asian languages. At that time, doing it in C would have been tantamount to rewriting the entire program. Since then, the parser and bits of converter back-ends have been rewritten in C, but the converter back-ends are still mostly in Perl which has good multilingual support built in.

- Additional back-ends. The makeinfo code had become convoluted to the point where adding a new back-end was quite complex, requiring complex interactions with existing back-ends. In contrast, the new implementation provides a clean tree-based representation for all back-ends to work from. People have requested numerous different back-ends (\LaTeX, the latest (X)HTML, \ldots); this change made them much more feasible to implement. Which leads to the last item:

- Making contributions easier. In general, due to the cleaner structure, the separate parser/back-ends implementation should be considerably easier than the former C makeinfo implementation for anyone to read and contribute to, with the resulting
obvious benefits. After ten years, contributed back-ends were yet to happen, but it is still believed that this structure could in theory lend better to contributions.

text2any is intended to be a reference implementation that defines parts of the language not fully specified by the manual. Without such a reference, alternative implementations would be very likely to have subtle, or not-so-subtle, differences in behavior, and thus Texinfo documents would become dependent on the processor. It is also important to have consistent command-line options for all processors. Extensive tests of the language and processor were developed at the same time as text2any; we encourage anyone thinking of writing a program to parse Texinfo input to make use of these tests.

With the release of text2any as the reference implementation, development of both the C implementation of makeinfo and text2html has been halted. Going forward, we ask authors of Texinfo documents to use only text2any.
2 Writing a Texinfo File

This chapter describes Texinfo syntax and what is required in a Texinfo file, and gives a short sample file.

2.1 General Syntactic Conventions

This section describes the general conventions used in all Texinfo documents.

- All printable ASCII characters except ‘@’, ‘{’ and ‘}’ can appear in a Texinfo file and stand for themselves. ‘@’ is the escape character which introduces commands, while ‘{’ and ‘}’ are used 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: ‘@@’, ‘@{’, and ‘@}’.

- In a Texinfo file, the commands you write to describe the contents of the manual are preceded by an ‘@’ character; they are called @-commands. (The ‘@’ in Texinfo has the same meaning that ‘\’ has in plain TeX.) Depending on what they do or what arguments they take, you need to write @-commands on lines of their own, or as part of sentences. As a general rule, a command requires braces if it mingles among other text; but it does not need braces if it is on a line of its own. For more details of Texinfo command syntax, see Section A.1 [Command Syntax], page 214.

- Whitespace following an @-command name is optional and (usually) ignored if present. The exceptions are contexts when whitespace is significant, e.g., an @example environment.

- Texinfo supports the usual quotation marks used in English and in other languages; see Section 11.5 [Inserting Quotation Marks], page 104.

- Use three hyphens in a row, ‘---’, to produce a long dash—like this (called an em dash), used for punctuation in sentences. Use two hyphens, ‘--’, to produce a medium dash (called an en dash), used primarily for numeric ranges, as in “June 25–26”. Use a single hyphen, ‘-’, to produce a standard hyphen used in compound words. For display on the screen, Info reduces three hyphens to two and two hyphens to one (not transitively!). Of course, any number of hyphens in the source remain as they are in literal contexts, such as @code and @example.

- Whitespace. Texinfo files are plain text files composed of lines terminated by the usual newline character (line feed). Texinfo processors read input one line at a time. Paragraphs are terminated by an empty line or a line containing only spaces. A sequence of several spaces in text is usually treated the same as a single space (except in verbatim modes).

Form feed (CTRL-l) characters in normal text end any open paragraph. Other ASCII whitespace (tab, carriage return) may be treated the same as space characters, although the results may differ depending on output format. Hence, there is not much point in using these in documents. Non-ASCII spaces, such as Unicode “em space”, are not recognized as whitespace at all and will be treated as regular, non-whitespace characters. However, in verbatim modes, for example in code samples, tab characters may produce the correct formatting in the output.
2.2 Comments

You can write comments in a Texinfo file by using the @comment command, which may be abbreviated to @c. Such comments are for a person looking at the Texinfo source 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 the visible output. (To be precise, the character after the @c or @comment must be something other than a dash or alphanumeric, or it will be taken as part of the command.)

Often, you can write the @comment or @c in the middle of a line, and only the text that follows after the @comment or @c command does not appear; but some commands, such as @settitle, work on a whole line. You cannot use @comment or @c within a line beginning with such a command.

In cases of nested command invocations, complicated macro definitions, etc., @c and @comment may provoke an error when processing with \TeX. Therefore, you can also use the \texttt{DEL} character (ASCII 127 decimal, 0x7f hex, 0177 octal) as a true \TeX comment character (catcode 14, in \TeX internals). Everything on the line after the \texttt{DEL} will be ignored.

You can also have long stretches of text ignored by the Texinfo processors with the @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 @ignore and @end ignore for writing comments. (For some caveats regarding nesting of such commands, see Section 15.7 [Conditional Nesting], page 141.)

2.3 What a Texinfo File Must Have

By convention, the name of a Texinfo file ends with one of the extensions .texi, .texinfo, .txi, or .tex. Using .tex is discouraged as this extension is already used by \TeX and \LaTeX\ input files. The most common and recommended extension is .texi. The name of a Texinfo file should only contain ASCII characters.

The output name is based on the input file name, in the default case. First, any of the extensions .texi, .tex, .txi or .texinfo is removed from the input file name; then, the output format specific extension is added—.html when generating HTML, .info when generating Info, etc. The output name should only contain ASCII characters\footnote{\texttt{texi2any} can handle non ASCII characters in input file names, but non ASCII characters in output name will create problems for some output formats, especially for cross-references.}.

In order to be made into a printed manual, a Texinfo file must begin with a line like this:

\texttt{\input texinfo}

The contents of the file follow this beginning, and then you must end the Texinfo source with a line like this:

\texttt{@bye}

The @bye line at the end of the file on a line of its own tells \TeX that the file is ended and to stop formatting. If you leave this out, you’ll be dumped at \TeX’s prompt at the end of the run.

Furthermore, you will usually provide a Texinfo file with a title, a title page, indices, and the like, all of which are explained in this manual. But the minimum, which can be useful for short documents, is just the one line at the beginning and the one line at the end.
Without additional information, the input and output encodings are assumed to be UTF-8, an universal codeset compatible with 7-bit ASCII.

## 2.4 Short Sample

Here is a short sample Texinfo file.

```
\input texinfo
@settitle Sample Manual 1.0

@copying
This is a short example of a complete Texinfo file.

Copyright @copyright{} 2016 Free Software Foundation, Inc.
@end copying

@titlepage
@title Sample Title
@page
@vskip 0pt plus 1filll
@insertcopying
@end titlepage

@contents

@node Top
@top GNU Sample

This manual is for GNU Sample
{version @value{VERSION}, @value{UPDATED}).

@menu
* First Chapter:: The first chapter is the only chapter in this sample.
  
* Index:: Complete index.
@end menu

@node First Chapter
@chapter First Chapter

@cindex chapter, first
This is the first chapter.
@cindex index entry, another
Here is a numbered list.

@enumerate
@item
This is the first item.
@item
This is the second item.
@end enumerate

@node First Section
@section First Section

First section of first chapter.

@node Second Section
@section Second Section

Second section of first chapter.

@node Index
@unnumbered Index
@printindex cp

@bye

2.5 Texinfo File Header

Texinfo files start with the line:

\input texinfo

The ‘\input texinfo’ line tells \TeX{} to use the texinfo.tex file, which tells \TeX{} how to translate the Texinfo @-commands into \TeX{} typesetting commands. (Note the use of the backslash, ‘\’; this is correct for \TeX{}.)

It makes sense to include any command that affects document formatting as a whole in the header. The @settitle line is usually present at the beginning of the header:

@settitle Sample Manual 1.0

The @settitle line specifies a title for the page headers (or footers) of the printed manual, and the default title and document description for the ‘<head>’ in HTML. @synindex (see Section 10.6.2 [@synindex], page 95), for instance, is another command often included in the header.

The start of the Texinfo file up to the first content that is output as part of the main body of the document is the preamble. It includes the header, Section 2.7 [Document Permissions],
page 16, and Section 2.8 [Titlepage & Copyright Page], page 18, specification. It is important
for the \TeX output format as the end of preamble is where the \texttt{\begin{document}} line
is output. In other output formats it may be used to determine how some special output
is formatted, for example Section 2.7.1 [@copying], page 17, output as a comment at the
beginning of output files, or the language used in file headers.

\subsection*{2.5.1 The First Line of a Texinfo File}

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

\begin{verbatim}
\input texinfo
\end{verbatim}

When the file is processed by \TeX, the `\texttt{\input texinfo}` command tells \TeX to load the
macros needed for processing a Texinfo file. These are in a file called \texttt{texinfo.tex}, which
should have been installed on your system along with either the \TeX or Texinfo software.
\TeX uses the backslash, `\`, to mark the beginning of a command, exactly as Texinfo uses
`@`. The \texttt{texinfo.tex} file causes the switch from `\` to `@`; before the switch occurs, \TeX requires `\`, which is why it appears at the beginning of the file.

You may optionally follow this line with a comment to tell GNU Emacs to use Texinfo
mode when the file is edited:

\begin{verbatim}
\input texinfo @c -*-texinfo-*- 
\end{verbatim}

This may be useful when Emacs doesn't detect the file type from the file extension automat-
ically.

\subsection*{2.5.2 @setfilename: Set the Output File Name}

The @setfilename line specifies the name of the output file to be generated by texi2any.
This command is ignored for \TeX formatting. When present, it should be the first Texinfo
command (that is, after `\texttt{\input texinfo}`). Write the @setfilename command at the
beginning of a line and follow it on the same line by the Info file name.

\begin{verbatim}
@setfilename info-file-name
\end{verbatim}

The name must be different from the name of the Texinfo file. There are two conventions
for choosing the name: you can either remove the extension (such as `.texi`) entirely from
the input file name, or (recommended) replace it with the `.info` extension. It is not advised
to base the @setfilename name on a entirely different name than the input file name.

When a @setfilename line is present, the Texinfo processors ignore everything written
before the @setfilename line. This is why the very first line of the file (the \texttt{\input} line)
does not show up in the output.

If there is no @setfilename line, texi2any uses the input file name to determine the
output name (see Section 2.3 [Minimum], page 10). The \texttt{\input} line is still ignored in this
processing, as well as leading blank lines.

When producing another output format, texi2any will replace any final extension with
the output format-specific extension (`html` when generating HTML, for example), or add a
dot followed by the extension (`html` for HTML) if the given name has no extension.

@-commands are not allowed in @setfilename, except for @&, @,{, @} and associated
@-commands such as @atchar{.}.
@setfilename used to be required by the Texinfo processors and some other programs. This should not be the case any more; @setfilename can be omitted. If the Texinfo input is processed from standard input, without an input file name to deduce the base file name from, @setfilename could still be relevant. This is not the only way, however: --output option specifies the output file name on the texi2any command-line (see Section 19.1 [Invoking texi2any], page 164).

Although an explicit ‘.info’ extension is preferable, some operating systems cannot handle long file names. You can run into a problem even when the file name you specify is itself short enough. This occurs because the Info formatters split a long Info file into short indirect subfiles, and name them by appending ‘-1’, ‘-2’, ..., ‘-10’, ‘-11’, and so on, to the original file name. (See Section 20.2 [Tag and Split Files], page 199.) The subfile name texinfo.info-10, for example, is too long for old systems with a 14-character limit on filenames; so the Info file name for this document could be texinfo rather than texinfo.info on such a system. @setfilename is a way to specify an alternative name.

2.5.3 @settitle: Set the Document Title
A Texinfo file should contain a line that looks like this:

@settitle title

Write the @settitle command at the beginning of a line and follow it on the same line by the title. Do not write anything else on the line. The @settitle command should precede everything that generates actual output. The best place for it is right after the @setfilename command, if present (described in the previous section).

This command gives the title to use in a header or footer for double-sided printed output, in case such headings are output. For more on headings for printed output, see Section 2.8.5 [Heading Generation], page 22.

In HTML, title serves as the document ‘<title>’ and it becomes the default document description in the ‘<head>’ part.

When the title page is used in the output, as is generally the case for printed output, the title in the @settitle command does not affect the title as it appears on the title page. Thus, the two do not need not to match exactly. A practice we recommend is to include the version or edition number of the manual in the @settitle title; on the title page, the version number generally appears as a @subtitle so it would be omitted from the @title. See Section 2.8.1 [@titlepage], page 19.

2.5.4 Preamble
The preamble starts at the beginning of the Texinfo file and continues until the first directly output material. It typically includes the file header (see Section 2.5 [Texinfo File Header], page 12), the @copying block specifying the document permissions (see Section 2.7.1 [@copying], page 17) and the @titlepage specification (see Section 2.8 [Titlepage & Copyright Page], page 18).

The preamble may contain commands that affect document formatting as a whole but which do not produce output, or do not produce output straight away, such as @settitle (see Section 2.5.3 [@settitle], page 14), @documentlanguage, (see Section 14.1 [@documentlanguage], page 130), commands setting the headings, controlling indentation or hyphenation, or the table of contents (see Section 2.9 [Contents], page 22).
Any text that starts a paragraph, \@-commands that are formatted as quotations, tables, lists and so on, and \@node (see Chapter 3 [Nodes], page 27) and chapter structuring commands (see Chapter 4 [Chapter Structuring], page 37) end the preamble.

The concept of the preamble is significant for L\TeX output, as the \texttt{\begin{document}} line is output at the end of the preamble.

In plaintext, the preamble is simply output as usual at the beginning of the document; for example, a \@contents in the preamble is output as the table of contents (see Section 2.9 [Contents], page 22).

There is not much special treatment of the preamble for HTML and Info output either. However, some settings current at the very end of the preamble may be used for the document as a whole, regardless of what follows. This may apply to commands specifying the indentation, or the language (see Section 14.1 [\@documentlanguage], page 130).

For example, for the following document, the HTML and Info copying comments are formatted with \@documentlanguage set to ‘pt’, as it is the last \@documentlanguage before the end of the preamble.

\begin{verbatim}
\input texinfo
@documentlanguage fr
@copying
The copying information \@error{} some text
@end copying

@documentlanguage pt

Text ending the preamble

@documentlanguage de

@node Top
\end{verbatim}

2.5.5 Start and End of Header for Emacs

In Emacs, start- and end-of-header lines can be used to enclose commands that globally affect the document in the Texinfo preamble. This allows you to format only part of a Texinfo file for Info or printing. See Section D.6.2 [texinfo-format commands], page 258.

A start-of-header line is a Texinfo comment that looks like this:

\begin{verbatim}
@c %**start of header
Write the start-of-header line on the second line of a Texinfo file. Follow the start-
of-header commands that globally affect the document formatting, such as \@settitle, \@synindex or \@footnotestyle; and then by an end-of-header line.

A end-of-header line is a Texinfo comment that looks like this:

@c %**end of header
The odd string of characters, ‘%**’, is to ensure that no other comment is accidentally taken for a start-of-header line. You can change it if you wish by setting the tex-start-
of-header and/or tex-end-of-header Emacs variables. See Section 18.5.2 [Texinfo Mode Printing], page 159.

The start- and end-of-header lines are not part of the Texinfo format specification, which is why they are implemented with comments.

2.6 Directory Category

Use the @dircategory command to specify a category for the manual. When choosing the name of the category, we recommend consulting the Free Software Directory (http://www.gnu.org/directory). If your program is not listed there, or listed incorrectly or incompletely, please report the situation to the directory maintainers (http://directory.fsf.org) so that the category names can be kept in sync.

Here are a few examples of category names:

- Basics
- Text creation and manipulation
- Archiving
- Compression
- Database
- Editors
- Emacs
- Email
- Graphics
- Localization
- Network applications
- Printing
- Science
- Software development
- Software libraries
- Version control

@dircategory commands are usually followed by a @dirententry blocks, which are used by install-info. See Section 20.1.4 [Installing Dir Entries], page 196, for details.

The first @dircategory command in a manual is the category for the entire manual. Subsequent uses of @dircategory set the category for following @dirententry blocks only.

2.7 Document Permissions

This segment describes the document and contains the copyright notice and copying permissions. This is done with the @copying command. A real manual includes more text here, according to the license under which it is distributed.

@copying

This is a short example of a complete Texinfo file, version 1.0.

Copyright @copyright{} 2016 Free Software Foundation, Inc.
@end copying

The copyright notice and copying permissions for a document need to appear in several places in the various Texinfo output formats. Therefore, Texinfo provides a command
(\texttt{@copying}) to declare this text once, and another command (\texttt{@insertcopying}) to insert the text at appropriate points.

If the document is a software manual, the software is typically under a different license—
for GNU and many other free software packages, software is usually released under the GNU GPL, and manuals are released under the GNU FDL. It is helpful to state the license of the software of the manual, but giving the complete text of the software license is not necessarily required.

\subsection*{2.7.1 \texttt{@copying}: Declare Copying Permissions}

The \texttt{@copying} command should be given very early in the document; the recommended location is right after the header material (see Section 2.5 [Texinfo File Header], page 12). It conventionally consists of a sentence or two about what the program is, identification of the documentation itself, the legal copyright line, and the copying permissions. Here is a skeletal example:

\begin{verbatim}
\texttt{@copying}
This manual is for \texttt{program} (version \texttt{version}, updated \texttt{date}), which ...

Copyright \texttt{\copyright\{} \texttt{years copyright-owner}.

\texttt{@quotation}
Permission is granted to ...
\texttt{@end quotation}
\texttt{@end copying}
\end{verbatim}

The \texttt{@quotation} has no legal significance; it’s there to improve readability in some contexts.

The text of \texttt{@copying} appears as a comment at the beginning of Info and HTML output files. This information is also output at the beginning of the DocBook output files using appropriate markup. It is \emph{not} output implicitly in plain text or printed output; it’s up to you to use \texttt{@insertcopying} to emit the copying information. See the next section for details.

The \texttt{@copyright\{} command generates a ‘c’ inside a circle when the output format supports this glyph (print and HTML always do, for instance). When the glyph is not supported in the output, it generates the three-character sequence ‘(C)’.

The copyright notice itself has the following legally-prescribed form:

\begin{verbatim}
Copyright \texttt{\copyright\{} \texttt{years copyright-owner}.

The word ‘Copyright’ must always be written in English, even if the document is otherwise written in another language. This is due to international law.

The list of years should include all years in which a version was completed (even if it was released in a subsequent year). It is simplest for each year to be written out individually and in full, separated by commas.

The copyright owner (or owners) is whoever holds legal copyright on the work. In the case of works assigned to the FSF, the owner is ‘Free Software Foundation, Inc.’.

The copyright ‘line’ may actually be split across multiple lines, both in the source document and in the output. This often happens for documents with a long history, having
many different years of publication. If you do use several lines, do not indent any of them (or anything else in the @copying block) in the source file.

See Section “Copyright Notices” in GNU Maintainer Information, for additional information. See Section C.1 [GNU Sample Texts], page 243, for the full text to be used in GNU manuals. See Appendix G [GNU Free Documentation License], page 284, for the license itself under which GNU and other free manuals are distributed.

### 2.7.2 @insertcopying: Include Permissions Text

The @insertcopying command is simply written on a line by itself, like this:

```
@insertcopying
```

This inserts the text previously defined by @copying. To meet legal requirements, it must be used on the copyright page in the printed manual (see Section 2.8.4 [Copyright], page 21).

The @copying command itself causes the permissions text to appear in an Info file before the first node. The text is also copied into the beginning of each split Info output file, as is legally necessary. This location implies a human reading the manual using Info does not see this text (except when using the advanced Info command g *). This does not matter for legal purposes, because the text is present. But to get a visible text in the output, @insertcopying should be used.

Similarly, the @copying text is automatically included at the beginning of each HTML output file, as an HTML comment. Again, this text is not visible without @insertcopying (unless the reader views the HTML source).

The permissions text defined by @copying also appears automatically at the beginning of the DocBook output files using appropriate markup. @insertcopying can be used to output the permission text within normal text.

### 2.8 Title and Copyright Pages

In hard copy output, the manual’s name and author are usually printed on a title page. Copyright information is usually printed on the back (verso) of the title page. This segment must be enclosed between @titlepage and @end titlepage commands:

```
@titlepage
<title>Sample Title</title>
@c The following two commands start the copyright page.
@page
@vskip 0pt plus 1filll
@insertcopying
@end titlepage
```

We use the @insertcopying command to include the permission text from the previous section, instead of writing it out again.

The title and copyright pages appear in printed manuals, but not in most other output formats. In HTML, the best way to get a title page similar to printed manuals is to set the NO_TOP_NODE_OUTPUT customization variable (see [NO_TOP_NODE_OUTPUT], page 184).
2.8.1 \texttt{@titlepage}

Start the material for the title page and following copyright page with \texttt{@titlepage} on a line by itself and end it with \texttt{@end titlepage} on a line by itself.

The \texttt{@end titlepage} command starts a new page and turns on page numbering (see Section 2.8.5 [Heading Generation], page 22). All the material that you want to appear on unnumbered pages should be put between the \texttt{@titlepage} and \texttt{@end titlepage} commands.

By using the \texttt{@page} command, you can force a page break within the region delineated by the \texttt{@titlepage} and \texttt{@end titlepage} commands and thereby create more than one unnumbered page. This is how the copyright page is produced. (The \texttt{@titlepage} command might perhaps have been better named the \texttt{@titleandadditionalpages} command, but that would have been rather long!)

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\textsuperscript{2} for the manual. This helps readers keep track of which manual is for which version of the program. (The ‘Top’ node should also contain this information; see Section 2.10 [The Top Node], page 23.)

One method uses the \texttt{@title}, \texttt{@subtitle}, and \texttt{@author} commands to create a title page. With this method, you do not specify any of the actual formatting of the title page. You specify the text you want, and Texinfo does the formatting. The usual formatting consist of black rules under the title and author lines and the subtitle text set flush to the right-hand side of the page.

Texinfo also provides a second method for creating a title page. using typesetting commands that are not to be used in the main text. This method uses uses the \texttt{@titlefont}, \texttt{@sp}, and \texttt{@center} commands to generate a title page in which the words on the page are centered.

For sufficiently simple documents, and for the bastard title page in traditional book frontmatter, Texinfo also provides a command \texttt{@shorttitlepage} which takes the rest of the line as the title. The argument is typeset on a page by itself and followed by a blank page. In HTML, \texttt{@shorttitlepage} can play the same role as \texttt{@settitle}, if \texttt{@settitle} is not set. See Section 2.5.3 [\texttt{@settitle}], page 14.

2.8.2 \texttt{@title}, \texttt{@subtitle}, and \texttt{@author}

You can use the \texttt{@title}, \texttt{@subtitle}, and \texttt{@author} commands to create a title page in which the vertical and horizontal spacing is done for you automatically.

Write the \texttt{@title}, \texttt{@subtitle}, or \texttt{@author} commands at the beginning of a line followed by the title, subtitle, or author. The \texttt{@author} command may be used for a quotation in an \texttt{@quotation} block (see Section 7.2 [\texttt{@quotation}], page 66); except for that, it is an error to use any of these commands outside of \texttt{@titlepage}.

The \texttt{@title} command normally produces a line in which the title is set flush to the left-hand side of the page in a larger than normal font. The title is underlined with a black

\footnote{We have found that it is helpful to refer to versions of independent 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.}
rule. The title must be given on a single line in the source file; it will be broken into multiple
lines of output is needed.

For long titles, the @* command may be used to specify the line breaks in long titles
if the automatic breaks do not suit. Such explicit line breaks are generally reflected in all
output formats; if you only want to specify them for the printed output, use a conditional
(see Chapter 15 [Conditionals], page 133). For example:

@title This Long Title@inlinefmt{tex,@*} Is Broken in @TeX{}

The @subtitle command normally sets subtitles in a normal-sized font flush to the
right-hand side of the page.

The @author command normally sets the names of the author or authors in a middle-sized
font flush to the left-hand side of the page on a line near the bottom of the title page. The
names are followed by a black rule that is thinner than the rule that normally underlines
the title.

There are two ways to use the @author command: you can write the name or names on
the remaining part of the line that starts with an @author command:

@titlepage
@title name-of-manual-when-printed
@subtitle subtitle-if-any
@subtitle second-subtitle
@author author
@page
... 
@end titlepage

2.8.3 @titlefont, @center, and @sp

You can also use the @titlefont, @sp, and @center commands to create a title page for a
printed document.

Use the @titlefont command to select a large font suitable for the title itself. You can
use @titlefont more than once if you have an especially long title.

For HTML output, each @titlefont command produces an <h1> heading, but the HTML
document <title> is not affected. For that, you could put a @settitle command before
the @titlefont command (see Section 2.5.3 [@settitle], page 14).

For example:

@titlefont{Texinfo}

Use the @center command at the beginning of a line to center the remaining text on
that line. Thus,

@center @titlefont{Texinfo}

centers the title, which in this example is “Texinfo” printed in the title font.
Use the \texttt{\textbackslash sp} command to insert vertical space. For example:

\texttt{\textbackslash sp 2}

This inserts two blank lines on the printed page. (See Section 12.7 \texttt{\textbackslash sp}, page 115, for more information about the \texttt{\textbackslash sp} command.)

A template for this method looks like this:

\begin{verbatim}
@titlepage
@sp 10
@center @titlefont{\textit{name-of-manual-when-printed}}
@sp 2
@center subtitle-if-any
@sp 2
@center author
...
@end titlepage
\end{verbatim}

The spacing of the example fits an 8.5 by 11 inch manual.

### 2.8.4 Copyright Page

By international treaty, the copyright notice for a book must be either on the title page or on the back of the title page. When the copyright notice is on the back of the title page, that page is customarily not numbered. Therefore, in Texinfo, the information on the copyright page should be within \texttt{\textbackslash titlepage} and \texttt{\textbackslash end titlepage} commands.

Use the \texttt{\textbackslash page} command to cause a page break. To push the copyright notice and the other text on the copyright page towards the bottom of the page, use the following incantation after \texttt{\textbackslash page}:

\texttt{\textbackslash vskip 0pt plus 1filll}

The \texttt{\textbackslash vskip} command inserts whitespace in the \texttt{\TeX} output; it is ignored in all other output formats. The \texttt{\textbackslash vskip 0pt plus 1filll} means to put in zero points of mandatory whitespace, and as much optional whitespace as needed to push the following text to the bottom of the page. Note the use of three \texttt{\textbackslash l}'s in the word \texttt{\textbackslash filll}; this is correct.

To insert the copyright text itself, write \texttt{\textbackslash insertcopying} next (see Section 2.7 [Document Permissions], page 16):

\begin{verbatim}
\texttt{\textbackslash insertcopying}
\end{verbatim}

Follow the copying text by the publisher, ISBN numbers, cover art credits, and other such information.

Here is an example putting all this together:

\begin{verbatim}
@titlepage
...
@page
@vskip 0pt plus 1filll
@insertcopying
Published by ...

Cover art by ...
\end{verbatim}
We have one more special case to consider: for plain text output, you must insert the copyright information explicitly if you want it to appear. For instance, you could have the following after the copyright page:

@ifplaintext
@insertcopying
@end ifplaintext

You could include other title-like information for the plain text output in the same place.

### 2.8.5 Heading Generation

Like all @end commands (see Chapter 7 [Quotations and Examples], page 65), the @end titlepage command must be written at the beginning of a line by itself, with only one space between the @end and the titlepage. It not only marks the end of the title and copyright pages, but also causes T\TeX to start generating page headings and page numbers.

Texinfo has two standard page heading formats, one for documents printed on one side of each sheet of paper (single-sided printing), and the other for documents printed on both sides of each sheet (double-sided printing).

In full generality, you can control the headings in different ways:

- The conventional way is to write a @setchapternewpage command before the title page commands, if required, and then have the @end titlepage command start generating page headings in the manner desired.

Most documents are formatted with the standard single-sided or double-sided headings, (sometimes) using @setchapternewpage odd for double-sided printing and (almost always) no @setchapternewpage command for single-sided printing (see Section E.1 [@setchapternewpage], page 269).

- Alternatively, you can use the @headings command to prevent page headings from being generated or to start them for either single or double-sided printing. Write a @headings command immediately after the @end titlepage command. To turn off headings, write @headings off. See Section E.2.1 [@headings], page 270.

- Or, you may specify your own page heading and footing format. See Section E.2 [Headings], page 270.

### 2.9 Generating a Table of Contents

The @chapter, @section, and other structuring commands (see Chapter 4 [Chapter Structuring], page 37) supply the information to make up a table of contents, but they do not cause an actual table to appear in the manual. To do this, you must use the @contents and/or @summarycontents command(s).

@contents

Generates a table of contents in a printed manual, including all chapters, sections, subsections, etc., as well as appendices and unnumbered chapters. Headings generated by @majorheading, @chapheading, and the other @...heading commands do not appear in the table of contents (see Section 4.2 [Structuring Command Types], page 37).
@shortcontents
@summarycontents

(@summarycontents is a synonym for @shortcontents.)

Generates a short or summary table of contents that lists only the chapters, appendices, and unnumbered chapters. Sections, subsections and subsubsection are omitted. Only a long manual needs a short table of contents in addition to the full table of contents.

Both contents commands should be written on a line by themselves, and placed near the beginning of the file, after the @end titlepage (see Section 2.8.1 [titlepage], page 19), before any sectioning command. The contents commands automatically generate a chapter-like heading at the top of the first table of contents page, so don't include any sectioning command such as @unnumbered before them.

Since an Info file uses menus instead of tables of contents, the Info formatting commands ignore the contents commands. But the contents are included in plain text output and in other output formats, such as HTML.

In HTML output, the links in the short table of contents point to corresponding entries in the full table of contents rather than the text of the document. The links in the full table of contents point to the main text of the document.

@shortcontents is not implemented for \LaTeX output.

2.10 The ‘Top’ Node and Master Menu

The ‘Top’ node is the node in which a reader enters an Info manual. As such, it should contain a very brief description of the manual (including the version number). The contents of the ‘Top’ node do not appear in printed output nor in DocBook output.

It is conventional to write a @top sectioning command line containing the title of the document immediately after the @node Top line (see Section 3.5 [top Command], page 31).

We repeat the short description from the beginning of the ‘@copying’ text, but there's no need to repeat the copyright information, so we don't use ‘@insertcopying’ here.

The ‘Top’ node contains a top-level menu listing the chapters, and possibly a detailed menu listing all the nodes in the entire document.

@node Top
@top Short Sample

This is a short sample Texinfo file.

@menu
  * First Chapter:: The first chapter is the only chapter in this sample.
  * Index:: Complete index.
@end menu
2.10.1 Parts of a Master Menu

A master menu is the main menu. It is customary to include a detailed menu listing all the nodes in the document in this menu. Like any other menu, a master menu is enclosed in \texttt{@menu} and \texttt{@end menu} and does not appear in the printed output nor in DocBook output.

The master menu contains entries for the major nodes in the Texinfo file: the nodes for the chapters, chapter-like sections, and the appendices, followed by nodes for the indices.

You may choose to follow these entries with a detailed menu. This lists other, lower-level nodes, often ordered by chapter. These items may be a convenience for an inquirer who can go directly to a particular node when searching for specific information, rather than going through an intermediate menu. If you use a detailed menu in your master menu, mark it with the \texttt{@detailmenu} ... \texttt{@end detailmenu} environment.

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 entry. (See Section 3.8.1 [Writing a Menu], page 33, for more information.)

For example, the master menu for this manual looks like the following (but has many more entries):

\begin{verbatim}
@menu
  * Copying Conditions:: Your rights.
  * Overview:: Texinfo in brief.
  ...
  * Command and Variable Index::
  * General Index::
@end menu

--- The Detailed Node Listing ---

Overview of Texinfo

  * Reporting Bugs:: ...
  ...

Beginning a Texinfo File

  * Sample Beginning:: ...
  ...
@end detailmenu
@end menu
\end{verbatim}

2.11 The Body of the Document

The body segment contains all the text of the document. A manual is divided into one or more nodes (see Chapter 3 [Nodes], page 27). The example illustrates a chapter made of three nodes, one for introductory material in the chapter, and two sections. The introductory material contains an enumerated list.
Here is a numbered list.

1. This is the first item.
2. This is the second item.

Second section of first chapter.

In the Info output, the ‘First Chapter’ node will contain a menu listing the two sections in the chapter. Similarly, when this node is output in its own HTML file, it will contain a table of contents for the chapter.

Here is what the contents of this chapter will look like:

1. First Chapter

This is the first chapter.

Here is a numbered list.

1. This is the first item.
2. This is the second item.

1.1 First Section

First section of first chapter.
1.2 Second Section

Second section of first chapter.
(In the Info and HTML output, the chapter would also be split into nodes.)

2.12 Ending a Texinfo File

The end of a Texinfo file should include commands to create indices (see Section 10.5 [Printing Indices & Menus], page 93), and the `@bye` command to mark the last line to be processed. For example:

```texinfo
@node Index
@unnumbered Index
@printindex cp
@bye
```

A `@bye` command terminates Texinfo processing. It should be on a line by itself. Anything following `@bye` is completely ignored.
3 Nodes

A node is a region of text that begins at a `@node` command, and continues until the next `@node` command. To specify a node, write a `@node` command at the beginning of a line, and follow it with the name of the node. Info readers display one node at a time, and provide commands for the user to move to related nodes. The HTML output can be similarly navigated.

Nodes are used as the targets of cross-references. Cross-references, such as the one at the end of this sentence, are made with `@xref` and related commands; see Chapter 5 [Cross References], page 43. Cross-references can be sprinkled throughout the text. Other `@`-commands may also be the target of cross-references (see Section 5.11 [@anchor], page 49, see Section 9.1 [Floats], page 83).

Normally, you put a node command immediately before each chapter structuring command—for example, an `@section` or `@subsection` line. (See Chapter 4 [Chapter Structuring], page 37.) You should do this even if you do not intend to format the file for Info. This is because printed output uses both `@node` names and chapter-structuring names in the output for cross-references. The only time you are likely to use the chapter structuring commands without also using nodes is if you are writing a document that contains no cross references and will only be printed, not transformed into Info, HTML, or other formats.

3.1 Writing a `@node` Line

Write `@node` at the beginning of a line followed by the name of the node, like this:

```latex
@node node-name
```

After you have inserted a `@node` line, you should immediately write the `@`-command for the associated chapter or section (if any) and insert its name.

You may optionally follow the node name argument to `@node` with up to three optional arguments on the rest of the same line, separating the arguments with commas. These are the names of the ‘Next’, ‘Previous’, and ‘Up’ pointers, in that order. Hence, the template for a fully-written-out node line with ‘Next’, ‘Previous’, and ‘Up’ pointers looks like this:

```latex
@node node-name, next, previous, up
```

The `node-name` argument must be present, but the others are optional. If you wish to specify some but not others, just insert commas as needed, as in: ‘@node mynode,,uppernode’. Any spaces before or after each name on the `@node` line are ignored. However, if your Texinfo document is hierarchically organized, as virtually all are, we recommend leaving off all the pointers and letting `makeinfo` determine them.

The `makeinfo` program automatically determines node pointers for a hierarchically organized document. For it to do this, each `@node` command should be followed immediately by a sectioning command such as `@chapter` or `@section` (except that comment lines may intervene). Finally, you must follow the ‘Top’ `@node` line with a line beginning with `@top` to mark the top-level node in the file. See Section 3.5 [@top Command], page 31.

Even when you explicitly specify all pointers, you cannot write the nodes in the Texinfo source file in an arbitrary order. You must write the nodes in the order you wish them to appear in the output. For Info format one can imagine that the order may not matter, but it matters for the other formats.
In most cases, you will want to take advantage of the pointer creation feature, and not redundantly specify node pointers that the programs can determine. However, Texinfo documents are not required to be organized hierarchically or in fact to contain sectioning commands at all (for example, if you never intend the document to be printed), so node pointers may still be specified explicitly, in full generality.

If you are using GNU Emacs, and want explicit pointers, you can use the update node commands provided by Texinfo mode to insert the names of the pointers. (See Section D.5 [Updating Nodes and Menus], page 252.) Alternatively, you can insert the ‘Next’, ‘Previous’, and ‘Up’ pointers yourself. If you do this in Emacs, you may find it helpful to use the Texinfo mode keyboard command \texttt{C-c C-c n}. This command inserts \texttt{@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.

### 3.2 Choosing Node Names

The name of a node identifies the node. For all the details of node names, see Section 3.3 [Node Line Requirements], page 29.

Here are some suggestions for node names:

- 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.)

- Try to pick node names that differ from each other near the beginnings of their names. This way, it is easy to use automatic name completion in Info.

- Conventionally, node names are capitalized in the same way as section and chapter titles. In this manual, initial and significant words are capitalized; others are not. In other manuals, just initial words and proper nouns are capitalized. Either way is fine; we recommend just being consistent.

- In HTML output, any characters in the node name other than plain ASCII letters, numbers or spaces will be changed in the file name. (See Section 21.6.2 [HTML Xref Node Name Expansion], page 208.) This can make the URLs for the pages in your manual less user-friendly; for example, in this manual the ‘@dots’ node is output as \texttt{___0040dots.html}.

Because node names are used in cross-references, it is not desirable to casually change them once published. When you delete or rename a node, it is usually a good idea to define an \texttt{@anchor} with the old name. That way, references from other manuals, from mail archives, and so on are not invalidated. See Section 5.11 [\texttt{@anchor}], page 49.

The pointers from a given node enable you to reach other nodes and consist simply of the names of those nodes.

Normally, a node’s ‘Up’ pointer contains the name of the node whose menu mentions that node. The node’s ‘Next’ pointer contains the name of the node that follows the present node in that menu and its ‘Previous’ pointer contains the name of the node that precedes it in that menu. When a node’s ‘Previous’ node is the same as its ‘Up’ node, both pointers name the same node.
Usually, the first node of a Texinfo file is the ‘Top’ node, and its ‘Up’ pointer points to the dir file, which contains the main menu for all of Info.

### 3.3 @node Line Requirements

Names used with @node have several requirements:

- All the node names in a single Texinfo file must be unique.
  
  This means, for example, that if you end every chapter with a summary, you must name each summary node differently. You cannot just call them all “Summary”. You may, however, duplicate the titles of chapters, sections, and the like. Thus you can end each chapter with a section called “Summary”, so long as the node names for those sections are all different.

  Node names, anchor names (see Section 5.11 [@anchor], page 49), and float labels (see Section 9.1.1 [@float], page 83) must all be unique.

- Node names can contain @-commands\(^1\). For example, using @TeX{} in a node name results in the \TeX{} logo being output, as it would be in normal text. Cross-references should use @TeX{} just as the node name does.

  Some commands do not make sense in node names; for instance, environments (e.g., @quotation), commands that read a whole line as their argument (e.g., @sp), and plenty of others. For the complete list of commands that are allowed, and their expansion for HTML identifiers and file names, see Section 21.6.3 [HTML Xref Command Expansion], page 209.

- A node name may not start with a left parenthesis preceding a right parenthesis, as in (not)allowed, since this syntax is used to specify an external manual.

- Unfortunately, you cannot reliably use periods, commas, or colons within a node name; these can confuse some Info readers. texi2any quotes problematic node names and labels by default, but some Info readers do not recognize this syntax. Node name and label quoting causes DEL characters (‘CTRL-?’, character number 127, often rendered as ‘^?’) to appear around the name. To remove node names and labels quoting, you can set the customization variable INFO\_SPECIAL\_CHARS\_QUOTE to ‘0’ (see Section 19.4.7 [Other Customization Variables], page 182).

  texi2any warns about such problematic usage in node names, menu items, and cross-references. If you don’t want to see the warnings, you can set the customization variable INFO\_SPECIAL\_CHARS\_WARNING to ‘0’ (see Section 19.4.7 [Other Customization Variables], page 182).

  If you insist on using these characters in node names, in order not to confuse the Texinfo processors you must still escape those characters, by using either special insertions (see Section 11.1.3 [Inserting a Comma], page 98) or @asis (see [@asis], page 79). For example:

  @node foo@asis{::}bar@comma{} baz

  As an example of avoiding the special characters, the following is a section title in this manual:

  @section @code{@unnumbered}, @code{@appendix}: Chapters with...

\(^1\) Prior to the Texinfo 5 release in 2013, this feature was supported in an ad hoc way (the --commands-in-node-names option to makeinfo). Now it is part of the language.
But the corresponding node name lacks the commas and the subtitle:

\@node \@code{@@unnumbered @@appendix}

- Case is significant in node names.
- Spaces before and after names on the ‘@node’ line are ignored. Multiple whitespace characters “inside” a name are collapsed to a single space. For example:

  \@node foo bar
  \@node foo bar ,
  \@node foo bar ,
  \@node foo bar ,

all define the same node, namely ‘foo bar’. In menu entries, a single internal space should be used in node names or some versions of some Info readers will not find the node.

3.4 The First Node

The first node of a Texinfo file is the Top node, except in an included file (see Chapter 17 [Include Files], page 152). The Top node should contain a short summary and a master menu. See Section 2.10 [The Top Node], page 23, for more information on the Top node contents and examples. Straight text before the Top node outside of any node should be avoided. Such text, if present, is not output for DocBook.

Here is a description of the node pointers to be used in the Top node:

- The Top node (which must be named ‘top’ or ‘Top’) should have as its ‘Up’ node the name of a node in another file, where there is a menu that leads to this file. Specify the file name in parentheses. Usually, all Info files are available through a single virtual Info tree, constructed from multiple directories. In this case, use ‘(dir)’ as the parent of the Top node; this specifies the top-level node in the dir file, which contains the main menu for the Info system as a whole. (Each directory with Info files is intended to contain a file named dir.)

That’s fine for Info, but for HTML output, one might well want the Up link from the Top node to go to some specific place. For example, for GNU the natural place would be http://www.gnu.org/manual/ (a web page collecting links to most GNU manuals), better specified as just /manual/ if the manual will be installed on www.gnu.org. This can be specified with the TOP_NODE_UP_URL customization variable (see Section 19.4.3 [HTML Customization Variables], page 174), as in

  $ texi2any --html -c TOP_NODE_UP_URL=/manual/ ...

- The ‘Prev’ node of the Top node is usually omitted.
- The ‘Next’ node of the Top node should be the first chapter in your document.

See Section 20.1 [Installing an Info File], page 194, for more information about installing an Info file in the info directory.

It is usually best to leave the pointers off entirely and let the tools implicitly define them, with this simple result:

\@node Top
3.5 The @top Sectioning Command

The @top command is a special sectioning command that you should only use after a ‘@node Top’ line at the beginning of a Texinfo file.

It produces the same sort of output as @unnumbered (see Section 4.4 [@unnumbered @appendix], page 39). In @LaTeX \part is used.

@top is ignored when raising or lowering sections. That is, it is never lowered and nothing can be raised to it (see Section 4.12 [Raise/lower sections], page 42).

It used to be conventional to wrap the ‘Top’ node in an @ifnottex conditional so that it would not appear in printed output (see Chapter 15 [Conditionals], page 133). Thus, a Top node often looked like this:

@ifnottex
  @node Top
  @top your-manual-title
  very-high-level-summary
@end ifnottex

This is no longer necessary, as the ‘Top’ node is now never output for printed output. The ‘Top’ node is not output for DocBook either.

3.6 Texinfo Document Structure

Nodes can contain menus, which contain the names of child nodes within the parent node; for example, a node corresponding to a chapter would have a menu of the sections in that chapter. The menus allow the user to move to the child nodes in the Info output.

In addition, nodes contain node pointers that name other nodes. The ‘Next’ and ‘Previous’ pointers link nodes at the same sectioning level into a chain. As you might imagine, the ‘Next’ pointer links to the next node, and the ‘Previous’ pointer links to the previous node. In general, ‘Next’ and ‘Previous’ refer to nodes at the same hierarchical level in the manual, not necessarily to the next node within the Texinfo file. In the Texinfo file, the subsequent node may be at a lower level—a section-level node most often follows a chapter-level node, for example. Thus, for example, all the nodes that are at the level of sections within a chapter are linked together, and the order in this chain is the same as the order of the children in the menu of the parent chapter. Each child node records the parent node name as its ‘Up’ pointer.

Since the ‘Top’ node is the only node at that level, ‘Next’ refers to the first following node, which is almost always a chapter or chapter-level node. This is an exception to the rule of ‘Next’ being at the same hierarchical level.

The Info and HTML output for each node includes links to the ‘Next’, ‘Previous’, and ‘Up’ nodes. The HTML also uses the accesskey attribute with the values ‘n’, ‘p’, and ‘u’ respectively. This allows people using web browsers to follow the navigation using (typically) M-letter, e.g., M-n for the ‘Next’ node, from anywhere within the node. Node pointers and menus provide structure for Info files just as chapters, sections, subsections, and the like provide structure for printed books. The two structures are theoretically distinct; in practice, however, the tree structure of printed books is essentially always used for the node and menu structure also, as this leads to a document which is easy to follow.
Typically, the sectioning structure and the node structure are completely parallel, with one node for each chapter, section, etc., and with the nodes following the same hierarchical arrangement as the sectioning. Thus, if a node is at the logical level of a chapter, its child nodes are at the level of sections; similarly, the child nodes of sections are at the level of subsections.

It is technically possible to create Texinfo documents with only one structure or the other, or for the two structures not to be parallel, or for either the sectioning or node structure to be different from the conventional structure. To the best of our knowledge, however, all the Texinfo manuals currently in general use do follow the conventional parallel structure.

### 3.7 Node and Menu Illustration

Here is a diagram that illustrates a Texinfo file with three chapters, each of which contains two sections.

The “root” is at the top of the diagram and the “leaves” are at the bottom. This is how such a diagram is drawn conventionally; it illustrates an upside-down tree. For this reason, the root node is called the ‘Top’ node, and ‘Up’ node pointers carry you closer to the root.

```
Top
 |
 |--------------------------|
 |                           |
 | Chapter 1                Chapter 2          Chapter 3 |
 |                           |
 |--------------------------|--------------------------|
 |                           |
 | Section 1.1              Section 1.2         Section 2.1 |
 |                           |
 |----------------------------|----------------------------|
 |                           |
 |                              Section Section Section Section Section Section |
 |                              1.1  1.2  2.1  2.2  3.1  3.2 |
```

Using explicit pointers (not recommended, but shown for purposes of the example), the fully-written command to start Chapter 2 would be this:

```
@node Chapter 2, Chapter 3, Chapter 1, Top
@comment node-name, next, previous, up
```

This `@node` line says that the name of this node is “Chapter 2”, the name of the ‘Next’ node is “Chapter 3”, the name of the ‘Previous’ node is “Chapter 1”, and the name of the ‘Up’ node is “Top”. You can (and should) omit writing out these node names if your document is hierarchically organized, but the pointer relationships still obtain.

To go to Sections 2.1 and 2.2 using Info, you need a menu inside Chapter 2. (See Section 3.8 [Menus], page 33.) You would write the menu just before the beginning of Section 2.1, like this:

```
@menu
  * Sect. 2.1:: Description of this section.
  * Sect. 2.2:: Description.
@end menu
```

The automatic pointers for the node for Sect. 2.1 correspond to:

```
@node Sect. 2.1, Sect. 2.2, , Chapter 2
@comment node-name, next, previous, up
```
Chapter 3: Nodes

Note that no ‘Prev’ pointer is generated, since there is no other node at the same hierarchical level before Sect. 2.1.

Using explicit pointers, the node for Sect. 2.1 could be written like this:
\begin{verbatim}
@node Sect. 2.1, Sect. 2.2, Chapter 2, Chapter 2
@comment node-name, next, previous, up
\end{verbatim}

With automatic pointers, the ‘Next’ and ‘Previous’ pointers of a node lead to other nodes at the same level—from chapter to chapter or from section to section. As shown, when using explicit pointers, the pointers can also lead somewhere else, here, for example, the ‘Previous’ pointer points up. An ‘Up’ pointer usually leads to a node at the level above (closer to the ‘Top’ node; and a ‘Menu’ leads to nodes at a level below (closer to ‘leaves’). (A cross-reference can point to a node at any level; see Chapter 5 [Cross References], page 43.)

Technically, explicit node pointers can carry you to any node, regardless of the structure of the document; even to nodes in a different Info file. However, it would be very confusing for readers to have the ‘Next’, ‘Previous’ and ‘Up’ pointers lead to nodes that do not correspond, even loosely, to the next, previous and up node.

A @node command and a chapter structuring command are conventionally used together, in that order, often followed by indexing commands. (As shown in the example above, you may follow the @node line with a comment line, e.g., to show which pointer is which if explicit pointers are used.) The Texinfo processors use this construct to determine the relationships between nodes and sectioning commands.

Here is the beginning of the chapter in this manual called “Ending a Texinfo File”. This shows a @node line followed by a @chapter line, and then by indexing lines.
\begin{verbatim}
@node Ending a File
@chapter Ending a Texinfo File
@cindex Ending a Texinfo file
@cindex Texinfo file ending
@cindex File ending
\end{verbatim}

3.8 Menus

Menus contain pointers to subordinate nodes. In Info output, you use menus to go to such nodes. Menus can be used in HTML output but are not used in the default case. Menus have no effect in printed manuals and do not appear in them. Menus are ignored in DocBook output.

Menus are automatically generated by texi2any when outputting Info for nodes followed by a sectioning command, without menu, and with automatic pointers. In a hierarchically organized manual with sectioning commands associated with nodes and with node pointers left out, you should only write menus if you want to have descriptions especially formatted in Info.

3.8.1 Writing a Menu

A menu consists of a @menu command on a line by itself, followed by menu entry lines or menu comment lines, and then followed by an @end menu command on a line by itself.

A menu looks like this:
In a menu, every line that begins with an ‘*’ is a menu entry. (Note the space after the asterisk.)

A line that does not start with an ‘*’ may also appear in a menu. Such a line is not a menu entry but rather a menu comment line that appears in the Info file. In the example above, the line ‘Larger Units of Text’ is such a menu comment line; the two lines starting with ‘*’ are menu entries.

Technically, menus can carry you to any node, regardless of the structure of the document; even to nodes in a different Info file. However, the makeinfo implicit pointer creation feature (see Section 3.1 [Writing a Node], page 27) and GNU Emacs Texinfo mode updating commands work only to create menus of subordinate nodes in a hierarchically structured document. In a hierarchically structured document, it is much better to use cross-references to refer to arbitrary nodes.

In Info, a user selects a node with the m (Info-menu) command. The menu entry name is what the user types after the m command. In the HTML output, the accesskey attribute is used with the values ‘1’...‘9’ for the first nine entries. This allows people using web browsers to follow the first menu entries using (typically) M-digit, e.g., M-1 for the first entry.

### 3.8.2 A Menu Example

A menu looks like this in Texinfo:

```texinfo
@menu
Larger Units of Text

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

This produces:

```
* Menu:

* menu entry name: Node name. A short description.
* Node name:: This form is preferred.
```

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

```texinfo
@menu
Larger Units of Text

* Files:: All about handling files.
* Multiples: Buffers. Multiple buffers; editing several files at once.
@end menu
```
This produces:

```plaintext
* 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 entry name and the name of the node referred to by that name. ‘Multiples’ is the menu entry name; it refers to the node named ‘Buffers’. The line ‘Larger Units of Text’ is a comment; it appears in the menu, but is not an entry.

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 3.8.6 [Referring to Other Info Files], page 36).

### 3.8.3 Menu Location

There may be at most one menu in a node. A menu is conventionally located at the end of a node, without any regular text or additional commands between the `@end menu` and the beginning of the next node.

This convention is useful, since a reader who uses the menu could easily miss any such text. Also, any such post-menu text will be considered part of the menu in Info output (which has no marker for the end of a menu). Thus, a line beginning with ‘*’ will likely be incorrectly handled.

It’s usually best if a node with a menu does not contain much text. If you find yourself with a lot of text before a menu, we generally recommend moving all but a couple of paragraphs into a new subnode. Otherwise, it is easy for readers to miss the menu.

### 3.8.4 The Parts of a Menu

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

1. The menu entry name (optional).
2. The name of the node (required).
3. A description of the item (optional).

The template for a generic menu entry looks like this (but see the next section for one more possibility):

```plaintext
* menu-entry-name: node-name. description
```

Follow the menu entry name with a single colon, and follow the node name with tab, comma, newline, or the two characters period and space (‘.’).

The third part of a menu entry is a descriptive phrase or sentence. Menu entry names and node names are often short; the description explains to the reader what the node is about. A useful description complements the node name rather than repeats it. The description, which is optional, can spread over multiple lines; if it does, some authors prefer to indent the second line while others prefer to align it with the first (and all others). It’s up to you. An empty line, or the next menu entry, ends a description.

Space characters in a menu are preserved as-is in the Info output; this allows you to format the menu as you wish. Unfortunately you must type node names without any extra
spaces or some versions of some Info readers will not find the node (see Section 3.3 [Node Line Requirements], page 29).

`makeinfo` warns and protect names when the text of a menu item (and node names and cross-references) contains a problematic construct that could interfere with its parsing in Info. See [Info Node Names Constraints], page 29.

### 3.8.5 Less Cluttered Menu Entry

When the menu entry name and 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:: description
```

instead of

```
* Name: Name.
```

We recommend using the node name for the menu entry name whenever possible, since it reduces visual clutter in the menu.

### 3.8.6 Referring to Other Info Files

You can create a menu entry that enables a reader in Info to go to a node in another Info file by writing the file name in parentheses just before the node name. Some examples:

```latex
@menu
* first-entry-name:(filename)nodename. description
* (filename)second-node:: description
@end menu
```

For example, to refer directly to the ‘Outlining’ and ‘Rebinding’ nodes in the *Emacs Manual*, you could write a menu like this:

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

If you do not list the node name, but only name the file, then Info presumes that you are referring to the ‘Top’ node. Examples:

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

The GNU Emacs Texinfo mode menu updating commands only work with nodes within the current buffer, so you cannot use them to create menus that refer to other files. You must write such menus by hand. See Section D.5 [Updating Nodes and Menus], page 252.
4 Chapter Structuring

Texinfo’s *chapter structuring* commands divide a document into a hierarchy of chapters, sections, subsections, and subsubsections. These commands generate large headings in the text, like the one above. They also provide information for generating the table of contents (see Section 2.9 [Generating a Table of Contents], page 22).

Normally you put a `@node` command immediately before each chapter structuring command. See Chapter 3 [Nodes], page 27.

4.1 Tree Structure of Sections

A Texinfo 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.

Here is a diagram that shows a Texinfo file with three chapters, each with two sections.

```
<table>
<thead>
<tr>
<th>Top</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Chapter 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Section</td>
</tr>
<tr>
<td>1.1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

In a Texinfo file that has this structure, the beginning of Chapter 2 would be written like this:

```
@node Chapter 2
@chapter Chapter 2
```

For purposes of example, here is how it would be written with explicit node pointers:

```
@node Chapter 2, Chapter 3, Chapter 1, Top
@chapter Chapter 2
```

The chapter structuring commands are described in the sections that follow; the `@node` command is described in the previous chapter (see Chapter 3 [Nodes], page 27).

4.2 Structuring Command Types

The chapter structuring commands fall into four groups, each of which contains structuring commands at the levels of chapters, sections, subsections, and subsubsections:

- The `@chapter`-like commands and `@appendix`-like commands produce numbered or lettered entries both in the body of a document and in its table of contents.
- The `@unnumbered`-like commands produce unnumbered entries both in the body of a document and in its table of contents. The `@top` command, which has a special use, is
a member of this group (see Section 3.5 [top Command], page 31). An unnumbered section is a normal part of the document structure.

- The heading-like commands produce simple unnumbered headings that do not appear in a table of contents, are not associated with nodes, and cannot be cross-referenced. These heading commands never start a new page.

In printed output, the chapter structuring commands produce headings in the document. When a setchapternewpage command says to do so, the chapter, unnumbered, and appendix commands start new pages in the printed manual; the heading commands do not. See Section E.1 [setchapternewpage], page 269.

In Info and plain text output, the command causes the title to appear on a line by itself, with a line of an ASCII character (‘*’, ‘=’, . . .) inserted underneath. For example, the “Chapter Structuring” heading could be:

5 Chapter Structuring

The underlining character is the same for all the commands at the same level. For instance, it is the same for the chapter-level commands chapter, appendix, unnumbered and chapheading.

In HTML, the chapter-level commands produce an <h2>-level header by default (controlled by the CHAPTER_HEADER_LEVEL customization variable, see Section 19.4.7 [Other Customization Variables], page 182). The heading element level is adjusted for the other commands.

In the DocBook output, the appropriate level of element is used. The produced element includes all following sections up to the next command at the same or higher level. For example, a <chapter> element is produced for chapter, and contains any sections or subsections in the chapter.

Here is a summary:

<table>
<thead>
<tr>
<th>Numbered</th>
<th>Unnumbered</th>
<th>Lettered/numbered</th>
</tr>
</thead>
<tbody>
<tr>
<td>In contents</td>
<td>In contents</td>
<td>In contents</td>
</tr>
<tr>
<td>chapter</td>
<td>unnumbered</td>
<td>appendix</td>
</tr>
<tr>
<td>section</td>
<td>numberedsec</td>
<td>appendixsec</td>
</tr>
<tr>
<td>subsection</td>
<td>numberedsubsec</td>
<td>appendixsubsec</td>
</tr>
<tr>
<td>subsubsection</td>
<td>numberedsubsubsec</td>
<td>appendixsubsubsec</td>
</tr>
<tr>
<td>No new page</td>
<td>Unnumbered</td>
<td>Not in contents</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Majorheading</th>
<th>Chapheading</th>
<th>Heading</th>
<th>Subheading</th>
<th>Subsubheading</th>
</tr>
</thead>
</table>

4.3 chapter: Chapter Structuring

chapter identifies a chapter in the document—the highest level of the normal document structuring hierarchy. Write the command at the beginning of a line and follow it on the same line by the title of the chapter. The chapter is numbered automatically, starting from 1.

For example, the present chapter in this manual is entitled “Chapter Structuring”; the chapter line looks like this:

chapter Chapter Structuring
4.4 \texttt{@unnumbered}, \texttt{@appendix}: Chapters with Other Labeling

Use the \texttt{@unnumbered} command to start a chapter-level element that appears without chapter numbers of any kind. Use the \texttt{@appendix} command to start an appendix that is labeled by letter (‘A’, ‘B’, ...) instead of by number; appendices are also at the chapter level of structuring.

Write an \texttt{@appendix} or \texttt{@unnumbered} command at the beginning of a line and follow it on the same line by the title, just as with \texttt{@chapter}.

TeXinfo also provides a command \texttt{@centerchap}, which is analogous to \texttt{@unnumbered}, but centers its argument in the printed and HTML outputs. This kind of stylistic choice is not usually offered by Texinfo. You are recommended not to use this command, as it may be removed in future releases of Texinfo.

With \texttt{@unnumbered}, if the name of the associated node is one of these English words (case-insensitive):

\begin{itemize}
  \item Acknowledgements
  \item Colophon
  \item Dedication
  \item Preface
\end{itemize}
then the DocBook output uses corresponding special tags (\texttt{<preface>}, etc.) instead of the default \texttt{<chapter>}. The argument to \texttt{@unnumbered} itself can be anything, and is output as the element title as usual.

4.5 \texttt{@majorheading}, \texttt{@chapheading}: Chapter-level Headings

The \texttt{@majorheading} and \texttt{@chapheading} commands produce chapter-like headings in the body of a document.

However, neither command produces an entry in the table of contents, and neither command causes TEX to start a new page in a printed manual.

In TEX, a \texttt{@majorheading} command generates a larger vertical whitespace before the heading than a \texttt{@chapheading} command but is otherwise the same.

In other output formats, the \texttt{@majorheading} and \texttt{@chapheading} commands produce a similar output to \texttt{@chapter}. The difference is the lack of numbering and the lack of any association with nodes. See Section 4.3 [\texttt{@chapter}], page 38.

4.6 \texttt{@section}: Sections Below Chapters

An \texttt{@section} command identifies a section within a chapter unit, whether created with \texttt{@chapter}, \texttt{@unnumbered}, or \texttt{@appendix}, following the numbering scheme of the chapter-level command. Thus, within a \texttt{@chapter} chapter numbered ‘1’, the sections are numbered ‘1.1’, ‘1.2’, etc.; within an \texttt{@appendix} “chapter” labeled ‘A’, the sections are numbered ‘A.1’, ‘A.2’, etc.; within an \texttt{@unnumbered} chapter, the section gets no number.

To make a section, write the \texttt{@section} command at the beginning of a line and follow it on the same line by the section title. For example:

\begin{verbatim}
@section This is a section
\end{verbatim}

Section titles are listed in the table of contents.
4.7 @unnumberedsec, @appendixsec, @heading

The @unnumberedsec, @appendixsec, and @heading commands are, respectively, the unnumbered, appendix-like, and heading-like equivalents of the @section command (see the previous section).

@unnumberedsec and @appendixsec do not need to be used in ordinary circumstances, because @section may also be used within @unnumbered and @appendix chapters; again, see the previous section.

@unnumberedsec
The @unnumberedsec command may be used within an unnumbered chapter or within a regular chapter or appendix to produce an unnumbered section.

@appendixsec
@appendixsection
@appendixsection is a longer spelling of the @appendixsec command; the two are synonymous.

Conventionally, the @appendixsec or @appendixsection command is used only within appendices.

@heading
You may use the @heading command (almost) anywhere for a section-style heading that will not appear in the table of contents. The @heading-series commands can appear inside most environments, for example, though pathological and useless locations such as an argument to another command, etc., are not allowed.

4.8 @subsection: Subsections Below Sections

Subsections are to sections as sections are to chapters; see Section 4.6 [@section], page 39. For example:

@subsection This is a subsection

Subsection titles are listed in the table of contents.

4.9 The @subsection-like Commands

The @unnumberedsubsec, @appendixsubsec, and @subheading commands are, respectively, the unnumbered, appendix-like, and heading-like equivalents of the @subsection command. (See Section 4.8 [@subsection], page 40.)

@unnumberedsubsec and @appendixsubsec do not need to be used in ordinary circumstances, because @subsection may also be used within sections of @unnumbered and @appendix chapters (see Section 4.6 [@section], page 39).

An @subheading command produces a heading like that of a subsection except that it is not numbered and does not appear in the table of contents. Similarly, an @unnumberedsubsec command produces an unnumbered heading like that of a subsection and an @appendixsubsec command produces a subsection-like heading labeled with a letter and numbers; both of these commands produce headings that appear in the table of contents.
4.10 \texttt{@subsubsection} and Other Subsub Commands

The fourth and lowest level sectioning commands in Texinfo are the ‘subsub’ commands. They are:

\texttt{@subsubsection}

Subsubsections are to subsections as subsections are to sections. (See Section 4.8 [\texttt{@subsection}], page 40.) Subsection titles appear in the table of contents.

\texttt{@unnumberedsubsubsec}

Unnumbered subsubsection titles appear in the table of contents, but lack numbers. Otherwise, unnumbered subsubsections are the same as subsubsections.

\texttt{@appendixsubsubsec}

Conventionally, appendix commands are used only for appendices and are lettered and numbered appropriately. They also appear in the table of contents.

\texttt{@subsubheading}

The \texttt{@subsubheading} command may be used anywhere that you want a small heading that will not appear in the table of contents.

As with subsections, \texttt{@unnumberedsubsubsec} and \texttt{@appendixsubsubsec} do not need to be used in ordinary circumstances, because \texttt{@subsubsection} may also be used within subsections of \texttt{@unnumbered} and \texttt{@appendix} chapters (see Section 4.6 [\texttt{@section}], page 39).

4.11 \texttt{@part}: Groups of Chapters

The final sectioning command is \texttt{@part}, to mark a part of a manual, that is, a group of chapters or (rarely) appendices. This behaves quite differently from the other sectioning commands, to fit with the way such “parts” are conventionally used in books.

No \texttt{@node} command is associated with \texttt{@part}. Just write the command on a line by itself, including the part title, at the place in the document you want to mark off as starting that part. For example:

\texttt{@part Part I:}@* The beginning

As can be inferred from this example, no automatic numbering or labeling of the \texttt{@part} text is done. The text is taken as-is.

Because parts are not associated with nodes, no general text can follow the \texttt{@part} line. To produce the intended output, it must be followed by a chapter-level command (including its node). Thus, to continue the example:

\texttt{@part Part I:}@* The beginning

\texttt{@node Introduction}

\texttt{@chapter Introduction}

\texttt{...}

In the \LaTeX output, the \texttt{@part} text is included in both the normal and short tables of contents (see Section 2.9 [Contents], page 22), without a page number (since that is the normal convention). In addition, a “part page” is output in the body of the document, with just the \texttt{@part} text. In the example above, the @* causes a line break on the part page (but is replaced with a space in the tables of contents). This part page is always forced
to be on an odd (right-hand) page, regardless of the chapter pagination (see Section E.1 [\@setchapternewpage], page 269). In the LaTeX output, the \part is output as \part.

In the HTML output, the \part text is similarly included in the tables of contents, and a heading is included in the main document text, as part of the following chapter or appendix node.

In the DocBook output, the <part> element includes all the following chapters, up to the next <part>. A <part> containing chapters is also closed at an appendix node.

In the Info and plain text output, \part has no effect.

\part is ignored when raising or lowering sections (see next section). That is, it is never lowered and nothing can be raised to it.

4.12 Raise/lower Sections: \raisesections and \lowersections

The \raisesections and \lowersections commands implicitly raise and lower the hierarchical level of following chapters, sections and the other sectioning commands (excluding parts).

That is, the \raisesections command changes sections to chapters, subsections to sections, and so on. Conversely, the \lowersections command changes chapters to sections, sections to subsections, and so on. Thus, a \lowersections command cancels a \raisesections command, and vice versa.

As a practical matter, you generally only want to raise or lower large chunks, usually in external files. You can use \lowersections to include text written as an outer or standalone Texinfo file in another Texinfo file as an inner, included file (see Chapter 17 [Include Files], page 152). Typical usage looks like this:

\begin{verbatim}
\lowersections
@include somefile.texi
\raisesections
\end{verbatim}

(Without the \raisesections, all the subsequent sections in the main file would also be lowered.)

If the included file being lowered has a \top node, you’ll need to conditionalize its inclusion with a flag (see Section 15.5.1 [\set @value], page 137).

Any menus in the final result have to take the raising and lowering into account, so arbitrarily sprinkling \raisesections and \lowersections commands throughout the document will likely lead to errors (unless the menus in your document are all generated automatically).

Repeated use of the commands continues to raise or lower the hierarchical level a step at a time. An attempt to raise above ‘chapter’ reproduces chapter commands; an attempt to lower below ‘subsubsection’ reproduces subsubsection commands. Also, lowered subsubsections and raised chapters will not work with makeinfo’s feature of implicitly determining node pointers, since the menu structure cannot be represented correctly.

Write each \raisesections and \lowersections command on a line of its own.
5 Cross-references

Cross-references are used to refer the reader to other parts of the same or different Texinfo files.

Use cross-references to provide access to information that is too detailed for the current context, or incidental to it. An online 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 results in a page reference, unless it is to another manual altogether, in which case the cross-reference names that manual. In Info, a cross-reference results in an entry that you can follow using the Info ‘f’ command. (See Section “Following cross-references” in Info.) In HTML, a cross-reference results in an hyperlink. In DocBook, the <link> element is used for cross-references unless it is to another manual, in which case the cross-reference names that manual.

The various cross-reference commands use nodes, anchors (see Section 5.11 [@anchor], page 49) or float labels (see Section 9.1.1 [@float], page 83) to define cross-reference locations. When TeX generates a DVI file, it records each cross-reference location page number and uses the page numbers in making references. Thus, even if you are writing a manual that will only be printed, and not used online, you must nonetheless write @node lines (or @anchor anchors) in order to name the places to which you make cross-references.

5.1 Different Cross-reference Commands

There are three different cross-reference commands:

@xref Used to start a sentence with an Info cross-reference saying ‘*Note name: node.’ or with ‘See . . . ’ in other output formats.

@ref Used within or, more often, at the end of a sentence; produces an Info cross-reference saying ‘*note name: node.’, and just the reference in other output formats, without the preceding ‘See’.

@pxref Used within parentheses, at the end of a sentence, or otherwise before punctuation, to make a reference. Its output starts with a lowercase ‘*note’ in Info, and with a lowercase ‘see’ in the other output formats. (‘p’ is for ‘parenthesis’.)

Additionally, there are commands to produce references to documents outside the Texinfo system. The @cite command is used to make references to books and manuals. @url produces a URL, for example a reference to a page on the World Wide Web.

5.2 Parts of a Cross-reference

A cross-reference command requires only one argument, which is the name of the node to which it refers. A cross-reference command may contain up to four additional arguments. The template for a full five argument cross-reference looks like this:

@xref{node-name, online-label, printed-label, manual-name, printed-manual-title}
The five possible arguments for a cross-reference are:

1. The node or anchor name. This is the location to which the cross-reference takes you. In a printed document, the location of the node provides the page reference only for references within the same document. Use `@node` to define the node (see Section 3.1 [Writing a Node], page 27), `@anchor` (see Section 5.11 [@anchor], page 49), or `@float` (see Section 9.1.1 [@float], page 83) with a label. This argument is required (except for reference to whole manuals).

Write a node name in a cross-reference in exactly the same way as in the `@node` line, including the same capitalization; otherwise, the processors may not find the reference.

2. A label for online output. It is usually omitted; then the topic description (third argument) is used if it was specified; if that was omitted as well, the node name is used.

3. A label for printed output. Often, this is the title or topic 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 manual to which the reference refers, if it is outside the current manual, in a different Texinfo file.

5. The title of the printed manual to which the reference refers, from a different Texinfo file.

Whitespace before and after the commas separating these arguments is ignored. To include a comma in one of the arguments, use `@comma{}` (see Section 11.1.3 [Inserting a Comma], page 98).

Cross-references with one, two, three, four, and five arguments are described separately in following sections.

When processing with TeX, a comma is automatically inserted after the page number for cross-references to within the same manual, unless the closing brace of the argument is followed by non-whitespace (such as a comma or period). This gives you the choice of whether to have a comma there in other output formats. For example,

```
@xref{Another Section} for more info
```

with TeX produces ‘See Another Section, page ppp, for more info’, and in the Info output produces ‘*Note Another Section:: for more info’.

If an unwanted comma is added, follow the argument with a command such as ‘@:`. For example,

```
@xref{Hurricanes}@:` --- for the details
```

See Hurricanes, page ppp — for the details

instead of ‘See Hurricanes, page ppp, — for the details’.

`makeinfo` warns and protect names when the text of a cross-reference (and node names and menu items) contains a problematic construct that could interfere with its parsing in Info. See [Info Node Names Constraints], page 29.

### 5.3 @xref with One Argument

The simplest form of `@xref` takes one argument, the name of another node in the same Texinfo file.
For example,
\texttt{@xref{Tropical Storms}.}
produces

*Note Tropical Storms::.
in Info and

See Section 3.1 [Tropical Storms], page 24.
in a printed manual.

5.4 \texttt{@xref} with Two Arguments

With two arguments, the second is used as a label for the online output. The template is like this:

\texttt{@xref\{node-name, online-label\}.}

For example,
\texttt{@xref\{Electrical Effects, Lightning\}.}
produces:

*Note Lightning: Electrical Effects.
in Info and

See Section 5.2 [Electrical Effects], page 57.
in a printed manual, where the node name is printed.

The second argument to cross-references shares constraints with node names. The potentially problematic character in this context is the colon. See [Info Node Names Constraints], page 29.

5.5 \texttt{@xref} with Three Arguments

A third argument replaces the node name in the printed output. The third argument should be the name of the section in the printed output, or else state the topic discussed by that section.
The template is like this:

\texttt{@xref\{node-name, online-label, printed-label\}.}

For example,
\texttt{@xref\{Electrical Effects, Lightning, Thunder and Lightning\}, for details.}
produces

*Note Lightning: Electrical Effects, for details.
in Info and

See Section 5.2 [Thunder and Lightning], page 57, for details.
in a printed manual.

If a third argument is given and the second one is empty, then the third argument serves for both. (Note how two commas, side by side, mark the empty second argument.)
\texttt{@xref\{Electrical Effects, , Thunder and Lightning\}, for details.}
produces

*Note Thunder and Lightning: Electrical Effects, for details.

in Info and

See Section 5.2 [Thunder and Lightning], page 57, for details.

in a printed manual.

The third argument to cross-references shares constraints with node names. The potentially problematic character in this context is the colon. See [Info Node Names Constraints], page 29.

As a practical matter, it is often best to write cross-references with just the first argument if the node name and the section title are the same (or nearly so), and with the first and third arguments only if the node name and title are different.

Texinfo offers a setting to use the section title instead of node names by default in cross-references (an explicitly specified third argument still takes precedence):

@xrefautomaticsectiontitle on

Typically this line would be given near the beginning of the document and used for the whole manual. But you can turn it off if you want (@xrefautomaticsectiontitle off), for example, if you’re including some other sub-document that doesn’t have suitable section names. This setting also applies to node headers in HTML, if @xrefautomaticsectiontitle is on, the sections names are used in node headers instead of the node names when possible.

5.6 @xref with Four and Five Arguments

In a cross-reference, a fourth argument specifies the name of another manual, different from the file in which the reference appears, and a fifth argument specifies its title as a printed manual.

The full template is:

@xref{node-name, online-label, printed-label, 
manual-name, printed-manual-title}.

For example,

@xref{Electrical Effects, Lightning, Thunder and Lightning, 
weather, An Introduction to Meteorology}.

produces this output in Info:

*Note Lightning: (weather)Electrical Effects.

As you can see, the name of the manual is enclosed in parentheses and precedes the name of the node. In HTML, the manual name and the node name are used to construct the hyperlink URL (see Section 21.6 [HTML Xref], page 207), while the link text is based on the label.

In a printed manual, the reference looks like this:

See section “Thunder and Lightning” in An Introduction to Meteorology.

The title of the printed manual is typeset like @cite; and the reference lacks a page number since the page a reference refers when that reference is to another manual cannot be known.
Next case: often, you will leave out the second argument when you use the long version of \texttt{@xref}. In this case, the third argument, the topic description, will be used as the cross-reference name in online formats. For example,

\begin{verbatim}
@xref{Electrical Effects, , Thunder and Lightning, weather, An Introduction to Meteorology}.
\end{verbatim}

produces

\begin{verbatim}
*Note Thunder and Lightning: (weather)Electrical Effects.
\end{verbatim}

in Info and

\begin{verbatim}
See section “Thunder and Lightning” in An Introduction to Meteorology.
\end{verbatim}
in a printed manual.

Next case: If the node name and the section title are the same in the other manual, you may also leave out the section title. In this case, the node name is used in both instances. For example,

\begin{verbatim}
@xref{Electrical Effects,, weather, An Introduction to Meteorology}.
\end{verbatim}

produces

\begin{verbatim}
*Note (weather)Electrical Effects::.
\end{verbatim}

in Info and

\begin{verbatim}
See section “Electrical Effects” in An Introduction to Meteorology.
\end{verbatim}
in a printed manual.

In general, there is no reason to have a manual name argument without a printed manual argument, unless no printed manual is generated. You may also want to refer to another manual file that is within a single printed manual—when multiple Texinfo files are incorporated into the same printed manual but can create separate output files in other output formats. In this case, you need to specify only the fourth argument, and not the fifth. If the printed manual title argument is missing, the manual name will be used instead in printed output.

A printed manual title argument without an online manual argument is of little use unless only a printed manual is generated from the Texinfo source. The result in online formats depends on the format, and can be, for example, an empty manual name or a reference to the printed manual formatted in a similar way to the printed output.

Finally, it’s also allowed to leave out all the arguments \texttt{except} the fourth and fifth, to refer to another manual as a whole. See the next section.

\section*{5.7 Referring to a Manual as a Whole}

Ordinarily, you must always name a node in a cross-reference. However, it’s not unusual to want to refer to another manual as a whole, rather than a particular section within it. In this case, giving any section name is an unnecessary distraction.

So, with cross-references to other manuals (see Section 5.6 \textquote[46]{Four and Five Arguments}), if the first argument is either ‘Top’ (capitalized just that way) or omitted entirely, and the third argument is omitted, the printed output includes no node or section name. (The Info output includes ‘Top’ if it was given.) For example,

\begin{verbatim}
@xref{Top,, make, The GNU Make Manual}.
\end{verbatim}
produces

*Note (make)Top:.

and

See The GNU Make Manual.

Info readers will go to the Top node of the manual whether or not the ‘Top’ node is explicitly specified.

It’s also possible (and is historical practice) to refer to a whole manual by specifying the ‘Top’ node and an appropriate entry for the third argument to the @xref command. Using this idiom, to make a cross-reference to The GNU Make Manual, you would write:

@xref{Top,, Overview, make, The GNU Make Manual}.

which produces

*Note Overview: (make)Top.

in Info and


in a printed manual.

In this example, ‘Top’ is the name of the first node, and ‘Overview’ is the name of the first section of the manual. There is no widely-used convention for naming the first section in a printed manual, this is just what the Make manual happens to use. This arbitrariness of the first name is a principal reason why omitting the third argument in whole-manual cross-references is preferable.

5.8 @xref

The @xref command generates a cross-reference for the beginning of a sentence. Examples of using @xref are in previous sections.

5.9 @ref

@ref is nearly the same as @xref except that it does not generate a ‘See’ in the output, just the reference itself. This makes it useful as the last part of a sentence.

For example,

For more information, @pxref{This}, and @ref{That}.

produces in Info:

For more information, *note This:; and *note That:.

and in printed output:

For more information, see Section 1.1 [This], page 1, and Section 1.2 [That], page 2.

The @ref command can tempt 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 could be using both the printed and other output formats such as Info. For example:

Sea surges are described in @ref{Hurricanes}.

looks ok in the printed output:

Sea surges are described in Section 6.7 [Hurricanes], page 72.
but is awkward to read in Info, “note” being a verb:

Sea surges are described in *note Hurricanes::.

### 5.10 @pxref

The parenthetical reference command, @pxref, is nearly the same as @xref, but it is best used within parentheses. The command differs from @xref in that the reference is typeset with a lowercase ‘see’ rather than an uppercase ‘See’. In Info, ‘*note’ is output.

With one argument, a parenthetical cross-reference looks like this:

... storms cause flooding (@pxref{Hurricanes}) ...

which produces

... storms cause flooding (*note Hurricanes::) ...

in Info and

... storms cause flooding (see Section 6.7 [Hurricanes], page 72) ...

in a printed manual.

In past versions of Texinfo, it was not allowed to write punctuation after a @pxref, so it could be used only before a right parenthesis. This is no longer the case. The effect of ‘@pxref{node-name}’ is similar to that of ‘see @ref{node-name}’. However, in many circumstances the latter is preferable, as this makes it clear in the Info output that the word “see” should be present.

### 5.11 @anchor: Defining Arbitrary Cross-reference Targets

An anchor is a position in your document, labelled so that cross-references can refer to it, just as they can to nodes. You create an anchor with the @anchor command, and give the label as a normal brace-delimited argument. For example:

This marks the @anchor{x-spot}spot.

... @xref{x-spot,,the spot}.

produces:

This marks the spot.

... See [the spot], page 1.

As you can see, the @anchor command itself produces no output. This example defines an anchor ‘x-spot’ just before the word ‘spot’. You can refer to it later with an @xref or other cross reference command, as shown (see Chapter 5 [Cross References], page 43).

It is best to put @anchor commands just before the position you wish to refer to; that way, the reader’s eye is led on to the correct text when they jump to the anchor. You can put the @anchor command on a line by itself if that helps readability of the source. Whitespace (including newlines) is ignored after @anchor.

Anchor names, node names and float labels may not conflict. Anchors, nodes and float labels are given similar treatment in some ways; for example, the goto-node command takes either an anchor name or a node name as an argument. (See Section “Go to node” in Info.). Anchors names and float labels could also appear in menus (see Section 3.8 [Menus],
Anchor names share the same constraints as nodes on the characters that can be included (see [Info Node Names Constraints], page 29).

Because of this duality, when you delete or rename a node, it is usually a good idea to define an @anchor with the old name. That way, any links to the old node, whether from other Texinfo manuals or general web pages, keep working.

5.12 @inforef: Cross-references to Info-only Material

@inforef is used for making cross-references to Info documents—even from a printed manual. This was originally used for Info files that were not generated from any Texinfo source. The command is now obsolete and should not be used. In addition to having little use, similar output can be obtained with @xref, @ref or @pxref with the Info file name as the fourth argument and no fifth argument.

The command takes either two or three arguments, in the following order:
1. The node name.
2. The cross-reference name (optional).
3. The Info file name.

The template is:

@inforef{node-name, cross-reference-name, info-file-name}

5.13 @url, @uref{url[, text][, replacement]}

@url produces a reference to a uniform resource locator (URL). It takes one mandatory argument, the URL, and two optional arguments which control the text that is displayed. In HTML and PDF output, @url produces a link you can follow. (To merely indicate a URL without creating a link people can follow, use @indicateurl, see Section 6.1.15 [indicateurl], page 62.)

@uref is a synonym for @url. (Originally, @url had the meaning of @indicateurl and @uref was required to produce a working link, but in practice @url was almost always misused. So we’ve changed the meaning.)

The second argument, if specified, is the text to display (the default is the URL itself); in output formats other than HTML, the URL is output in addition to this text.

The third argument, if specified, is the text to display, but in this case the URL is not output in any format. This is useful when the text is already sufficiently referential, as in a man page. Also, if the third argument is given, the second argument is ignored.

5.13.1 @url Examples

First, here is an example of the simplest form of @url, with just one argument. The given URL is both the target and the visible text of the link:

The official GNU ftp site is @url{http://ftp.gnu.org/gnu}.

produces:

Two-argument form of @url

Here is an example of the two-argument form:

The official @url{http://ftp.gnu.org/gnu, GNU ftp site}
holds programs and texts.

which produces:

The official GNU ftp site (http://ftp.gnu.org/gnu)
holds programs and texts.

The HTML output is this:

The official <a href="http://ftp.gnu.org/gnu">GNU ftp site</a>
holds programs and texts.

In other formats, the output is like this:

The official GNU ftp site (http://ftp.gnu.org/gnu)
holds programs and texts.

Three-argument form of @url

Finally, an example of the three-argument form:

The @url{/man.cgi/1/ls,,ls} program ...

which, except for HTML, produces:

The ls program ...

but with HTML:

The <a href="/man.cgi/1/ls">ls</a> program ...

By the way, some people prefer to display URLs in the unambiguous format:

<URL:http://host/path>

You can use this form in the input file if you wish. We feel it’s not necessary to include the
‘<URL:’ and ‘>’ in the output, since to be useful any software that tries to detect URLs in
text already has to detect them without the ‘<URL:’.

5.13.2 URL Line Breaking

\TeX{} allows line breaking within URLs at only a few characters (which are special in URLs):
‘&’, ‘.’, ‘#’, ‘?’, and ‘/’ (but not between two ‘/’ characters). A tiny amount of stretchable
space is also inserted around these characters to help with line breaking.

For HTML output, modern browsers will also do line breaking within displayed URLs. If
you need to allow breaks at other characters you can insert @/ as needed (see Section 12.2
[Line Breaks], page 113).

By default, in \TeX{} any such breaks at special characters will occur after the character.
Some people prefer such breaks to happen before the special character. This can be controlled
with the @urefbreakstyle command (this command has effect only in \TeX{}):

@urefbreakstyle how

where the argument how is one of these words:

‘after’ (the default) Potentially break after the special characters.
‘before’ Potentially break before the special characters.
Chapter 5: Cross-references

5.13.3 \url PDF Output Format

If the ultimate purpose of a PDF is only to be viewed online, perhaps similar to HTML in some inchoate way, you may not want the URLs to be included in the visible text (just as URLs are not visible to readers of web pages). Texinfo provides a PDF-specific option for this, which must be used inside @tex:

```
\tex
  \global\urefurlonlylinktrue
@end tex
```

The result is that @url{http://www.gnu.org, GNU} has the visible output of just ‘GNU’, with a link target of http://www.gnu.org. Ordinarily, the visible output would include both the label and the url: ‘GNU (http://www.gnu.org)’.

This option only has effect when the PDF output is produced with the pdf\TeX\ program, not with other ways of getting from Texinfo to PDF (e.g., \TeX\ to DVI to PDF). Consequently, it is ok to specify this option unconditionally within @tex, as shown above. It is ignored when DVI is being produced.

5.13.4 PDF Colors

By default, URLs and cross-reference links are printed in black in PDF output. Very occasionally, however, you may want to highlight such “live” links with a different color, as is commonly done on web pages. Texinfo provides a PDF-specific option for specifying these colors, which must be used inside @tex:

```
\tex
  \global\def\linkcolor{1 0 0} % red
  \global\def\urlcolor{0 1 0} % green
@end tex
```

\urlcolor changes the color of @url output (both the actual URL and any textual label), while \linkcolor changes the color for cross-references to nodes, etc. They are independent.

The three given values must be numbers between 0 and 1, specifying the amount of red, green, and blue respectively.

These definitions only have an effect when the PDF output is produced with the pdf\TeX\ program, not with other ways of getting from Texinfo to PDF (e.g., \TeX\ to DVI to PDF). Consequently, it is ok to specify this option unconditionally within @tex, as shown above. It is ignored when DVI is being produced.

We do not recommend colorizing just for fun; unless you have a specific reason to use colors, best to skip it.

5.14 @cite{reference}

Use the @cite command for the name of a book that lacks a companion Info file. For example, we could refer to A Book. The command selects a slanted font in the printed manual, and generates quotation marks in the Info file.
If a book is written in Texinfo, it is better to use a cross-reference command since a reader can easily follow such a reference in Info. See Section 5.8 [@xref], page 48.
6 Marking Text, Words and Phrases

In Texinfo, you can mark words and phrases in a variety of ways. The Texinfo processors use this information to determine how to highlight the text. You can 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 emphasize text, in several different ways.

6.1 Indicating Definitions, Commands, etc.

Texinfo has commands for indicating just what kind of object a piece of text refers to. For example, email addresses are marked by @email; that way, the result can be a live link to send email when the output format supports it. If the email address was simply marked as “print in a typewriter font”, that would not be possible.

6.1.1 Highlighting Commands are Useful

The commands serve a variety of purposes:

@code{(sample-code)}
    Indicate text that is a literal example of a piece of a program. See Section 6.1.2 [@code], page 55.

@kbd{(keyboard-characters)}
    Indicate keyboard input. See Section 6.1.3 [@kbd], page 56.

@key{(key-name)}
    Indicate the conventional name for a key on a keyboard. See Section 6.1.4 [@key], page 57.

@samp{(text)}
    Indicate text that is a literal example of a sequence of characters. See Section 6.1.5 [@samp], page 57.

@verb{(text)}
    Write a verbatim sequence of characters. See Section 6.1.6 [@verb], page 58.

@var{(metasyntactic-variable)}
    Indicate a metasyntactic variable. See Section 6.1.7 [@var], page 58.

@env{(environment-variable)}
    Indicate an environment variable. See Section 6.1.8 [@env], page 59.

@file{(file-name)}
    Indicate the name of a file. See Section 6.1.9 [@file], page 59.

@command{(command-name)}
    Indicate the name of a command. See Section 6.1.10 [@command], page 60.

@option{(option)}
    Indicate a command-line option. See Section 6.1.11 [@option], page 60.

@dfn{(term)}
    Indicate the introductory or defining use of a term. See Section 6.1.12 [@dfn], page 60.
@cite{reference}
Indicate the name of a book. See Section 5.14 [@cite], page 52.

@abbr{abbreviation}
Indicate an abbreviation, such as ‘Comput.’.

@acronym{acronym}
Indicate an acronym. See Section 6.1.14 [@acronym], page 61.

@indicateurl{uniform-resource-locator}
Indicate an example (that is, nonfunctional) uniform resource locator. See Section 6.1.15 [@indicateurl], page 62. (Use @url (see Section 5.13 [@url], page 50) for live URLs.)

@email{email-address[, displayed-text]}
Indicate an electronic mail address. See Section 6.1.16 [@email], page 62.

6.1.2 @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 in a programming language.

Use @code for command names in languages that resemble programming languages, such as Texinfo. For example, @code and @samp are produced by writing ‘@code{@@code}’ and ‘@code{@@samp}’ in the Texinfo source, respectively.

It is incorrect to alter the case of a word inside a @code command when it appears at the beginning of a sentence. Most computer languages are case sensitive. In C, for example, Printf is different from the identifier printf, and most likely is a misspelling of it. Even in languages which are not case sensitive, it is confusing to a human reader to see identifiers spelled in different ways. Pick one spelling and always use that. If you do not want to start a sentence with a command name written all in lowercase, you should rearrange the sentence.

The @code argument is typeset in a typewriter (monospace) font. where the output format allows this. For example,

The function returns @code{nil}.

produces this:

The function returns nil.

Here are some cases for which it is preferable not to use @code:

• For shell command names, such as ls (use @command).
• For environment variables, such as TEXINPUTS (use @env).
• For shell options, such as ‘-c’, when such options stand alone (use @option).
• An entire shell command often looks better if written using @samp rather than @code. In this case, the rule is to choose the more pleasing format.
• For a string of characters shorter than a syntactic token. For example, if you are writing about ‘goto-ch’, which is just a part of the name for the goto-char Emacs Lisp function, you should use @samp.
• In general, when writing about the characters used in a token; for example, do not use \texttt{\@code} when you are explaining what letters or printable symbols can be used in the names of functions. (Use \texttt{\@samp}.) Also, you should not use \texttt{\@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 \texttt{\@code} for the keystroke commands of GNU Emacs (use \texttt{\@kbd} instead) although you may use \texttt{\@code} for the names of the Emacs Lisp functions that the keystroke commands invoke.

By default, \TeX{} will consider breaking lines at ‘-’ and ‘_’ characters within \texttt{\@code} and related commands. This can be controlled with \texttt{allowcodebreaks} (see Section 12.4 [allowcodebreaks], page 114). In the HTML output breaking lines is up to the browser’s behavior. For Info, it seems better never to make such breaks.

For Info and plaintext, quotation characters are usually output around the output of the \texttt{\@code} command and related commands (e.g., \texttt{\@kbd}, \texttt{\@command}) except in typewriter-like contexts such as the \texttt{\example} environment (see Section 7.4 [\example], page 67) and \texttt{\@code} itself, etc. To control which quoting characters are inserted by \texttt{texi2any} in the output of ‘\texttt{\@code}’, etc., see the \texttt{OPEN_QUOTE_SYMBOL} and \texttt{CLOSE_QUOTE_SYMBOL} customization variables (see Section 19.4.7 [Other Customization Variables], page 182).

### 6.1.3 \texttt{\@kbd\{keyboard-characters\}}

Use the \texttt{\@kbd} command for characters of input to be typed by users. For example, to refer to the characters \texttt{M-a}, write:

\begin{verbatim}
\@kbd{M-a}
\end{verbatim}

and to refer to the characters \texttt{M-x shell}, write:

\begin{verbatim}
\@kbd{M-x shell}
\end{verbatim}

By default, the \texttt{\@kbd} command produces a different font (slanted typewriter instead of normal typewriter, where the output format allows), so users can distinguish the characters that they are supposed to type from those that the computer outputs.

Since the usage of \texttt{\@kbd} varies from manual to manual, you can control the font switching with the \texttt{\@kbdinputstyle} command. This command has no effect on Info output. Write this command at the beginning of a line with a single word as an argument, one of the following:

- ‘\texttt{code}’ Always use the same font for \texttt{\@kbd} as \texttt{\@code}.
- ‘\texttt{example}’ Use the distinguishing font for \texttt{\@kbd} only in \texttt{\example} and similar environments.
- ‘\texttt{distinct}’ (the default) Always use the distinguishing font for \texttt{\@kbd}.

You can embed another \texttt{-command} inside the braces of a \texttt{\@kbd} command. Here, for example, is the way to describe a command that would be described more verbosely as “press the ‘r’ key and then press the RETURN key”:

\begin{verbatim}
\@kbd{r \@key{RET}}
\end{verbatim}

This produces: \texttt{r \texttt{\@key{RET}}}. (The present manual uses the default for \texttt{\@kbdinputstyle}.)

You also use the \texttt{\@kbd} command if you are spelling out the letters you type; for example:

\begin{verbatim}
To give the \texttt{\@code{logout}} command,
\end{verbatim}
type the characters \texttt{\kbd{logout} \key{RET}}.

This produces:

To give the \texttt{logout} command, type the characters \texttt{logout RET}.

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

\section*{6.1.4 \texttt{key\{key-name\}}}

Use the \texttt{\key{key-name}} command for the conventional name for a key on a keyboard, as in:

\begin{verbatim}
\key{RET}
\end{verbatim}

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

For example, to produce \texttt{C-x ESC} and \texttt{M-TAB} you would type:

\begin{verbatim}
\kbd{C-x \key{ESC}}
\kbd{M-\key{TAB}}
\end{verbatim}

Here is a list of the recommended names for keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPC</td>
<td>Space</td>
</tr>
<tr>
<td>RET</td>
<td>Return</td>
</tr>
<tr>
<td>LFD</td>
<td>Linefeed</td>
</tr>
<tr>
<td>TAB</td>
<td>Tab</td>
</tr>
<tr>
<td>BS</td>
<td>Backspace</td>
</tr>
<tr>
<td>ESC</td>
<td>Escape</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete</td>
</tr>
<tr>
<td>SHIFT</td>
<td>Shift</td>
</tr>
<tr>
<td>CTRL</td>
<td>Control</td>
</tr>
<tr>
<td>META</td>
<td>Meta</td>
</tr>
</tbody>
</table>

There are subtleties to handling words like ‘meta’ or ‘ctrl’ that are names of modifier keys. When mentioning a character in which the modifier key is used, such as \texttt{Meta-a}, use the \texttt{\kbd} command alone; do not use the \texttt{\key} command; but when you are referring to the modifier key in isolation, use the \texttt{\key} command. For example, write ‘\texttt{\kbd{Meta-a}}’ to produce \texttt{Meta-a} and ‘\texttt{\key{META}}’ to produce \texttt{META}.

\section*{6.1.5 \texttt{samp\{text\}}}

Use the \texttt{samp} command to indicate text that is a literal example or ‘sample’ of a sequence of characters in a file, string, pattern, etc. Enclose the text in braces. The argument appears within single quotation marks; in addition, it is printed in a fixed-width font.

To match \texttt{samp\{foo\}} at the end of the line,

\begin{verbatim}
use the regexp \samp{foo$}.
\end{verbatim}

produces
To match ‘foo’ at the end of the line, use the regexp ‘foo$’.

Any time you are referring to single characters, you should use @samp unless @kbd or @key is more appropriate. Also, you may use @samp for entire statements in C and for entire shell commands—in this case, @samp often looks better than @code. Basically, @samp is a catchall for whatever is not covered by @code, @kbd, @key, @command, etc.

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:

\begin{quote}
In English, the vowels are @samp{a}, @samp{e}, @samp{i}, @samp{o}, @samp{u}, and sometimes @samp{y}.
\end{quote}

This produces:

In English, the vowels are ‘a’, ‘e’, ‘i’, ‘o’, ‘u’, and sometimes ‘y’.

6.1.6 \verb{chartextchar}

Use the \verb command to print a verbatim sequence of characters.

Like \LaTeX’s \verb command, the verbatim text can be quoted using any unique delimiter character. Enclose the verbatim text, including the delimiters, in braces. Text is printed in a fixed-width font:

\begin{quote}
How many \verb{[@]}-escapes does one need to print this
\verb{.@a @b.@c.} string or \verb{+@’e?’{}!’\+} this?
\end{quote}

produces

How many @-escapes does one need to print this
a b.c. string or @’e?’{}!’ this?

This is in contrast to @samp (see the previous section), @code, and similar commands; in those cases, the argument is normal Texinfo text, where the three characters @{} are special, as usual. With \verb, nothing is special except the delimiter character you choose.

The delimiter character itself may appear inside the verbatim text, as shown above. As another example, ‘\verb{...}’ prints a single (fixed-width) period.

It is not reliable to use \verb inside other Texinfo constructs. In particular, it does not work to use \verb in anything related to cross-referencing, such as section titles or figure captions.

6.1.7 \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 normal variables in computer programs. These are specific names, so @code is correct for them. For example, the Emacs Lisp variable texinfo-tex-command is not a metasyntactic variable; it is properly formatted using @code.

Do not use \var for environment variables either; @env is correct for them (see the next section).
The effect of @var in the Info file is to change the case of the argument to all uppercase. In the printed manual, the argument is output in slanted type.¹

For example,

To delete file @var{filename},

type @samp{rm @var{filename}}.

produces

To delete file filename, type ‘rm filename’.

(Note that @var may appear inside @code, @samp, @file, etc.)

Write a metasyntactic variable all in lowercase without spaces, and use hyphens to make it more readable. Thus, the Texinfo source for the illustration of how to begin a Texinfo manual looks like this:

```
\input texinfo
@@settitle @var{name-of-manual}
```

This produces:

```
\input texinfo
@settitle name-of-manual
```

In some documentation styles, metasyntactic variables are shown with angle brackets, for example:

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

However, that is not the style that Texinfo uses.

### 6.1.8 @env{environment-variable}

Use the @env command to indicate environment variables, as used by many operating systems, including GNU. Do not use it for metasyntactic variables; use @var for those (see the previous section).

@env is equivalent to @code in its effects. For example:

```
The @env{PATH} environment variable ...
```

produces

```
The PATH environment variable ...
```

### 6.1.9 @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 file name suffixes. Do not use @file for symbols in a programming language; use @code.

@file is equivalent to @code in its effects. 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.
```

¹ In \TeX output, @var currently uses a slanted typewriter font in code contexts such as @code or @example. We plan to change this in the next release to use a variable-width, slanted roman font in all contexts. To avoid this change, set the ‘txicodevaristt’ flag using @set; specify ‘@clear txicodevaristt’ to make this change now (see Section 15.5.1 [@set @value], page 137). Note that this flag does nothing in \LaTeXX output.
6.1.10 \texttt{@command\{command-name\}}

Use the \texttt{@command} command to indicate command names, such as \texttt{ls} or \texttt{cc}.

\texttt{@command} is equivalent to \texttt{@code} in its effects. For example:

\begin{quote}
   \texttt{The command \texttt{@command\{ls\}} lists directory contents.}
\end{quote}

produces

\begin{quote}
   The command \texttt{ls} lists directory contents.
\end{quote}

You should write the name of a program in the ordinary text font, rather than using \texttt{@command}, if you regard it as a new English word, such as ‘Emacs’ or ‘Bison’.

When writing an entire shell command invocation, as in ‘\texttt{ls -l’}, you should use either \texttt{@samp} or \texttt{@code} at your discretion.

6.1.11 \texttt{@option\{option-name\}}

Use the \texttt{@option} command to indicate a command-line option; for example, \texttt{-l} or \texttt{--version} or \texttt{--output=filename}.

\texttt{@option} is equivalent to \texttt{@code} in its effects. For example:

\begin{quote}
   \texttt{The option \texttt{@option\{-l\}} produces a long listing.}
\end{quote}

produces

\begin{quote}
   The option \texttt{-l} produces a long listing.
\end{quote}

6.1.12 \texttt{@dfn\{term\}}

Use the \texttt{@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 does not deserve \texttt{@dfn}. The command selects a slanted font in the printed manual, and generates double quotation marks in the Info file. For example:

\begin{quote}
   Getting rid of a file is called \texttt{@dfn\{deleting\}} it.
\end{quote}

produces

\begin{quote}
   Getting rid of a file is called \texttt{deleting} it.
\end{quote}

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

6.1.13 \texttt{@abbr\{abbreviation[, meaning]\}}

You can use the \texttt{@abbr} command for general abbreviations. The abbreviation is given as the single argument in braces, as in ‘\texttt{@abbr\{Comput.\}}’. As a matter of style, or for particular abbreviations, you may prefer to omit periods, as in ‘\texttt{@abbr\{Mr\} Stallman\}’.

\texttt{@abbr} accepts an optional second argument, intended to be used for the meaning of the abbreviation.

If the abbreviation ends with a lowercase letter and a period, and is not at the end of a sentence, and has no second argument, remember to use the \texttt{\textbackslash .} command (see Section 11.3.3 [Ending a Sentence], page 101) to get the correct spacing. However, you do not have to use \texttt{\textbackslash .} within the abbreviation itself; Texinfo automatically assumes periods within the abbreviation do not end a sentence.
In output formats with an appropriate tag, such as HTML and DocBook, this tag is used. Otherwise, the first argument is printed as-is; if the second argument is present, it is printed in parentheses after the abbreviation. For instance:
\@abbr{Comput. J., Computer Journal}
produces:
Comput. J. (Computer Journal)

For abbreviations consisting of all capital letters, you may prefer to use the \@acronym command instead. See the next section for more on the usage of these two commands.

### 6.1.14 \@acronym{\textit{acronym[, meaning]}}

You can use the \@acronym command for abbreviations written in all capital letters, such as ‘NASA’. The abbreviation is given as the single argument in braces, as in ‘\@acronym{NASA}’. As a matter of style, or for particular acronyms, you may prefer to use periods, as in ‘\@acronym{N.A.S.A.}’.

\@acronym accepts an optional second argument, intended to be used for the meaning of the acronym.

If the acronym is at the end of a sentence, and if there is no second argument, remember to use the @. or similar command (see Section 11.3.3 [Ending a Sentence], page 101) to get the correct spacing.

In \TeX, the acronym is printed in slightly smaller font. In the Info output, the argument is printed as-is. In either format, and in \LaTeX output, if the second argument is present, it is printed in parentheses after the acronym. In HTML and DocBook the appropriate tag is used.

For instance (since GNU is a recursive acronym, we use \@acronym recursively):
\@acronym{GNU, \@acronym{GNU}’s Not Unix}
produces:
GNU (GNU’s Not Unix)

In some circumstances, it is conventional to print family names in all capitals. Don’t use \@acronym for this, since a name is not an acronym. Use \@sc instead (see Section 6.2.2 [Smallcaps], page 63).

\@abbr and \@acronym are closely related commands: they both signal to the reader that a shortened form is being used, and possibly give a meaning. When choosing whether to use these two commands, please bear the following in mind.

- In common English usage, acronyms are a subset of abbreviations: they include pronounceable words like ‘NATO’, ‘radar’, and ‘snafu’; some sources also include syllable acronyms like ‘Usenet’, hybrids like ‘SIGGRAPH’, and unpronounceable initialisms like ‘FBI’.
- In Texinfo, an acronym (but not an abbreviation) should consist only of capital letters and periods, no lowercase.
- In \TeX, an acronym (but not an abbreviation) is printed in a slightly smaller font.
- It usually turns out to be quite difficult and/or time-consuming to consistently use \@acronym for all sequences of uppercase letters. Furthermore, it looks strange for some acronyms to be in the normal font size and others to be smaller. Thus, a simpler
approach you may wish to consider is to avoid `@acronym` and just typeset everything as normal text in all capitals: ‘GNU’, producing the output ‘GNU’.

- In general, it’s not essential to use either of these commands for all abbreviations; use your judgment. Text is perfectly readable without them.

### 6.1.15 `@indicateurl{uniform-resource-locator}`

Use the `@indicateurl` command to indicate a uniform resource locator on the World Wide Web. This is purely for markup purposes and does not produce a link you can follow (use the `@url` or `@uref` command for that, see Section 5.13 [@url], page 50). `@indicateurl` is useful for URLs which do not actually exist. For example:

For example, the URL might be `@indicateurl{http://example.org/path}`.

which produces:

For example, the URL might be ‘http://example.org/path’.

The output from `@indicateurl` is usually like that of `@samp` (see Section 6.1.5 [@samp], page 57).

### 6.1.16 `@email{email-address[, displayed-text]}`

Use the `@email` command to indicate an electronic mail address. It takes one mandatory argument, the address, and one optional argument, the text to display (the default is the address itself).

In Info, the address is shown in angle brackets, preceded by the text to display if any. In printed output, the angle brackets are omitted. In HTML and DocBook output, `@email` produces a ‘mailto’ link. In HTML, a ‘mailto’ link usually brings up a mail composition window. For example:

Send bug reports to `@email{bug-texinfo@@gnu.org}`, suggestions to the `@email{bug-texinfo@@gnu.org, same place}`.

produces

Send bug reports to bug-texinfo@gnu.org, suggestions to the same place.

### 6.2 Emphasizing Text

Usually, Texinfo changes the font to mark words in the text according to the category the words belong to; an example is the `@code` command. Most often, this is the best way to mark words. However, sometimes you will want to emphasize text without indicating a category. Texinfo has two commands to do this. Also, Texinfo has several commands that specify the font in which text will be output. These commands have no effect in Info and only one of them, the `@r` command, has any regular use.

#### 6.2.1 `@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**. In the Info output, `@emph` surrounds the text with underscores (‘_’), and `@strong` puts asterisks around the text.
For example,

```
@strong{Caution:} @samp{rm * .[^.]*}
```

removes @emph{all} files in the directory.

produces the following:

**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.

**Caution**: Do not use `@strong` with the word ‘Note’ followed by a space; Info will mistake the combination for a cross-reference. Use a phrase such as Please notice or Caution instead, or the optional argument to `@quotation`—‘Note’ is allowable there.

### 6.2.2 `@sc{text}`: The Small Caps Font

Use the `@sc` command to set text in a SMALL CAPS FONT (where possible). Write the text you want to be in small caps between braces in lowercase, like this:

```
Richard @sc{Stallman} a commencé le projet GNU.
```

This produces:

Richard STALLMAN a commencé le projet GNU.

As shown here, we recommend reserving `@sc` for special cases where you want typographic small caps; family names are one such, especially in languages other than English, though there are no hard-and-fast rules about such things.

TEX typesets any uppercase letters between the braces of an `@sc` command in full-size capitals; only lowercase letters are printed in the small caps font. In the Info output, the argument to `@sc` is printed in all uppercase. In HTML, the argument is uppercased and the output marked with the `<small>` tag to reduce the font size, since HTML cannot easily represent true small caps. In L\TeX, a command setting small caps fonts is output.

Overall, we recommend using standard upper- and lowercase letters wherever possible.

### 6.2.3 Fonts for Printing

`Texinfo` provides one command to change the size of the main body font in printed output for a document: `@fonttextsize`. It has no effect in other output. It takes a single argument on the remainder of the line, which must be either ‘10’ or ‘11’. For example:

```
@fonttextsize 10
```

The effect is to reduce the body font to a 10 pt size (the default is 11 pt). Fonts for other elements, such as sections and chapters, are reduced accordingly. This should only be used in conjunction with `@smallbook` (see Section E.6 [@smallbook], page 275) or similar, since 10 pt fonts on standard paper (8.5x11 or A4) are too small. One reason to use this command is to save pages, and hence printing cost, for physical books.

`Texinfo` does not at present have commands to switch the font family to use, or more general size-changing commands.

`Texinfo` also provides a number of font commands that specify font changes in the printed manual and (where possible) in the HTML and DocBook output. They have no effect in Info. All the commands apply to a following argument surrounded by braces.

```
@b selects bold face;
```
@i selects an *italic* font;

@r selects a roman font, which is the usual font in which text is printed. It may or may not be seriffed.

@sansserif selects a **sans serif** font;

@slanted selects a **slanted** font;

@t selects the **fixed-width**, typewriter-style font used by @code;

The @r command can be useful in example-like environments, to write comments in the standard roman font instead of the fixed-width font. This looks better in printed output.

For example,

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

produces

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

The @t command can occasionally be useful for producing output in a typewriter font where that is supported, but no distinction with quotation marks is needed in Info or plain text. (Compare @t{foo} producing foo with @code{foo} producing foo.) Here are some possible reasons for using @t instead of @code:

− The argument is a single character
− There are already quotes of some kind enclosing the argument
− It’s evident from context or the argument itself that the argument could be computer code (e.g. name of a Usenet newsgroup)

In general, the other font commands are unlikely to be useful; they exist primarily to make it possible to document the functionality of specific font effects, such as in \TeX{} and related packages.
7 Quotations and Examples

Quotations and examples 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 the output.

In Texinfo, you always begin a quotation or example by writing an @-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 @example by itself at the beginning of a line and end the example by writing @end example on a line by itself, at the beginning of that line, and with only one space between the @end and the example.

7.1 Block Enclosing Commands

Here is a summary of commands that enclose blocks of text, also known as environments. They’re explained further in the following sections.

@quotation
Indicate text that is quoted. The text is filled, indented (from both margins), and printed in a roman font by default.

@indentedblock
Like @quotation, but the text is indented only on the left.

@example
Illustrate code, commands, and the like. The text is printed in a fixed-width font, and indented but not filled.

@lisp
Like @example, but specifically for illustrating Lisp code. The text is printed in a fixed-width font, and indented but not filled.

@verbatim
Mark a piece of text that is to be printed verbatim; no character substitutions are made and all commands are ignored, until the next @end verbatim. The text is printed in a fixed-width font, and not indented or filled. Extra spaces and blank lines are significant, and tabs are expanded.

@display
Display illustrative text. The text is indented but not filled, and no font is selected (so, by default, the font is roman).

@format
Like @display (the text is not filled and no font is selected), but the text is not indented.

@smallquotation
@smallindentedblock
@smallexample
@smalllisp
@smalldisplay
@smallformat
These @small... commands are just like their non-small counterparts, except that they output text in a smaller font size, where possible.
@flushleft
Text is not filled, but is set flush with the left or right margin, respectively.

@raggedright
Text is filled, but only justified on the left, leaving the right margin ragged.

@cartouche
Highlight text, often an example or quotation, by drawing a box with rounded corners around it.

The @exdent command is used within the above constructs to undo the indentation of a line.

The @noindent command may be used after one of the above constructs (or at the beginning of any paragraph) to prevent the following text from being indented as a new paragraph.

7.2 @quotation: Block Quotations

The text of a quotation is processed like normal text (regular font, text is filled) except that:

- the left margin is closer to the center of the page, so the whole of the quotation is indented; the right margin may also be closer to the center of the page
- the first lines of paragraphs are indented no more than other lines; and
- an @author command may be given to specify the author of the quotation.

This is an example of text written between a @quotation command and an @end quotation command. A @quotation command is most often used to indicate text that is excerpted from another (real or hypothetical) printed work.

Write a @quotation command as text on 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 quotation. The @end quotation line will likewise disappear from the output.

@quotation takes one optional argument, given on the remainder of the line. This text, if present, is included at the beginning of the quotation in bold or otherwise emphasized, and followed with a ‘:’. For example:

@quotation Note
This is a foo.
@end quotation

produces

Note: This is a foo.

If the @quotation argument is one of these English words (case-insensitive):

Caution Important Note Tip Warning

then the DocBook output uses corresponding special tags (<note>, etc.) instead of the default <blockquote>.

If the author of the quotation is specified in the @quotation block with the @author command, a line with the author name is displayed after the quotation:

@quotation
People sometimes ask me if it is a sin in the Church of Emacs to use vi. Using a free version of vi is not a sin; it is a penance. So happy hacking.

@author Richard Stallman
@end quotation

produces

People sometimes ask me if it is a sin in the Church of Emacs to use vi. Using a free version of vi is not a sin; it is a penance. So happy hacking.

—Richard Stallman

7.3 @indentedblock: Indented text blocks

The @indentedblock environment is similar to @quotation, except that text is only indented on the left (and there is no optional argument for an author). Thus, the text font remains unchanged, and text is gathered and filled as usual, but the left margin is increased. For example:

This is an example of text written between an @indentedblock command and an @end indentedblock command. The @indentedblock environment can contain any text or other commands desired.

This is written in the Texinfo source as:

@indentedblock
This is an example ...
@end indentedblock

7.4 @example: Example Text

The @example environment is used to indicate computer input or output that is not part of the running text. If you want to embed code fragments within sentences, use the @code command or its relatives instead (see Section 6.1.2 [@code], page 55).

Write an @example command at the beginning of a line by itself. Mark the end of the block with @end example. For example,

@example
  cp foo @var{dest1}; \
  cp foo @var{dest2}
@end example

produces

  cp foo dest1; \
  cp foo dest2

The output uses a fixed-width font and is indented. Each line in the input file is a line in the output; that is, the source text is not filled. Extra spaces and blank lines are significant. Texinfo commands are expanded; if you want the output to be the input verbatim, use the @verbatim environment instead (see Section 7.5 [@verbatim], page 68).

Examples are often, logically speaking, “in the middle” of a paragraph, and the text that continues afterwards should not be indented, as in the example above. The @noindent
command prevents a piece of text from being indented as if it were a new paragraph (see Section 7.12 [@noindent], page 72).

If you wish to use the normal roman font for a code comment, you can use the @r command (see Section 6.2.3 [Fonts], page 63).

You may optionally give arguments to the @example command, separated by commas if there is more than one. In the HTML output, any such arguments are output as class names, prefixed by the string ‘user-’. This may be useful for adding syntax highlighting to manuals for code samples.

We recommend that when you give multiple arguments to @example, you use the first argument to specify the language of the code (e.g. ‘C’, ‘lisp’, ‘Cplusplus’). You may find uses for other arguments, such as providing a formatting hint or marking code samples for extraction and further processing, but for now nothing definitive is recommended. Perhaps this will change in future Texinfo releases.

Caution: Do not use tabs in the lines of an example! (Or anywhere else in Texinfo, except in verbatim environments.) \TeX treats tabs as single spaces, and that is not what they look like.

### 7.5 @verbatim: Literal Text

Use the @verbatim environment for printing of text that may contain special characters or commands that should not be interpreted, such as computer input or output (@example interprets its text as regular Texinfo commands). This is especially useful for including automatically generated files in a Texinfo manual.

In general, the output will be just the same as the input. No character substitutions are made, e.g., all spaces and blank lines are significant, including tabs. The text is typeset in a fixed-width font, and not indented or filled.

Write a @verbatim command at the beginning of a line by itself. This line will disappear from the output. Mark the end of the verbatim block with an @end verbatim command, also written at the beginning of a line by itself. The @end verbatim will also disappear from the output.

For example:

```verbatim
{[tab]@command with strange characters: @'e expand[tab]me }
@end verbatim
```

(where [tab] stands for a literal tab character). This produces:

```verbatim
{     @command with strange characters: @'e expand me }
```

Since the lines containing @verbatim and @end verbatim produce no output, typically you should put a blank line before the @verbatim and another blank line after the @end
verbatim. Blank lines between the beginning \verbatim and the ending \endverbatim will appear in the output.

You can get a “small” verbatim by enclosing the \verbatim in an \smallformat environment, as shown here:

\verbatim
\smallformat
\verbatim
... still verbatim, but in a smaller font ...
\verbatim
\endverbatim
\end smallformat

Finally, a word of warning: it is not reliable to use \verbatim inside other Texinfo constructs.

See also Section 17.3 [verbatiminclude], page 153.

7.6 @lisp: Marking a Lisp Example

The @lisp command was used for Lisp code:

\lisp
Example lisp code
@end lisp

This is now synonymous with the following:

@example lisp
Example lisp code
@end example

Use @lisp 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 Texinfo file. Then you can use the Texinfo file as a Lisp library.

7.7 @display: Examples Using the Text Font

The @display command begins another kind of environment, where the font is left unchanged, not switched to typewriter as with @example. Each line of input still produces a line of output, and the output is still indented.

This is an example of text written between a @display command and an @end display command. The @display command indents the text, but does not fill it.

7.8 @format: Examples Using the Full Line Width

The @format command is similar to @display, except it leaves the text unindented. Like @display, it does not select the fixed-width font. Thus,

@format
This is an example of text written between a @code{@format} command and an @code{@end format} command. As you can see from this example, the @code{@format} command does not fill the text.
@end format
produces
This is an example of text written between a @format command and an @end format command. As you can see from this example, the @format command does not fill the text.

7.9 @exdent: Undoing a Line’s Indention

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. Do not use braces around the text. The text on an @exdent line is also printed in the roman font where the output format allows this.

@example
This line follows an @@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 @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.

@exdent does not have an effect in all output formats.

7.10 @flushleft and @flushright

The @flushleft and @flushright commands line up the 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 @flushleft and @flushright commands are ended by @end flushleft and @end flushright commands on lines of their own.
For example,

@flushleft
This text is
written flushleft.
@end flushleft

produces

This text is
written flushleft.

@flushright produces the type of indentation often used in the return address of letters. For example,

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

produces

Here is an example of text written
flushright. The @flushright command
right justifies every line but leaves the
left end ragged.

7.11 @raggedright: Ragged Right Text

The @raggedright fills text as usual, but the text is only justified on the left; the right
margin is ragged. The command is written on a line of its own, without braces. The
@raggedright command is ended by @code{@end raggedright} on a line of its own. This command
only has an effect in output formats where text is justified on the left, but not in output
formats where text is always set ragged right, such as Info or HTML.

The @raggedright command can be useful with paragraphs containing lists of commands
with long names, when it is known in advance that justifying the text on both margins will
make the paragraph look bad.

An example (from elsewhere in this manual):

@raggedright
Commands for double and single angle quotation marks:
@code{@@guillemetleft@{@}}, @code{@@guillemetright@{@}},
@code{@@guillemotleft@{@}}, @code{@@guillemotright@{@}},
@code{@@guilsinglleft@{@}}, @code{@@guilsinglright@{@}}.
@end raggedright

produces

Commands for double and single angle quotation marks: @guillemetleft{},
@guillemetright{}, @guillemotleft{}, @guillemotright{}, @guilsinglleft{},
@guilsinglright{}.
7.12 `@noindent`: Omitting Indentation

An example or other inclusion can break a paragraph into segments. Ordinarily, the formatters indent text that follows an example as a new paragraph. You can prevent this on a case-by-case basis by writing `@noindent` at the beginning of a line, preceding the continuation text. You can also disable indentation for all paragraphs globally with `@paragraphindent` (see Section E.3 [@paragraphindent], page 274).

Here is an example showing how to eliminate the normal indentation of the text after an `@example`, a common situation:

```latex
@example
This is an example
@end example
@noindent
This line is not indented. As you can see, the beginning of the line is fully flush left with the line that follows after it.
```

produces:

```
This is an example

This line is not indented. As you can see, the beginning of the line is fully flush left with the line that follows after it.
```

The standard usage of `@noindent` is just as above: at the beginning of what would otherwise be a paragraph, to eliminate the indentation that normally happens there. It can either be followed by text or be on a line by itself. There is no reason to use it in other contexts, such as in the middle of a paragraph or inside an environment (see Chapter 7 [Quotations and Examples], page 65).

You can control the number of blank lines in the Info file output by adjusting the input as desired: a line containing just `@noindent` does not generate a blank line, and neither does an `@end` line for an environment.

Do not put braces after a `@noindent` command; they are not used, since `@noindent` is a command used outside of paragraphs (see Section A.1 [Command Syntax], page 214).

7.13 `@indent`: Forcing Indentation

To complement the `@noindent` command (see the previous section), Texinfo provides the `@indent` command to force a paragraph to be indented. For instance, this paragraph (the first in this section) is indented using an `@indent` command.

And indeed, the first paragraph of a section is the most likely place to use `@indent`, to override the normal behavior of no indentation there (see Section E.3 [@paragraphindent], page 274). It can either be followed by text or be on a line by itself.

As a special case, when `@indent` is used in an environment where text is not filled, it produces a paragraph indentation space in the \TeX{} output. (These environments are where
a line of input produces a line of output, such as `@example` and `@display`; for a summary of all environments, see Section 7.1 [Block Enclosing Commands], page 65. (Do not put braces after an `@indent` command; they are not used, since `@indent` is a command used outside of paragraphs (see Section A.1 [Command Syntax], page 214).

7.14 `@cartouche`: Rounded Rectangles

Where the output format allows, the `@cartouche` command draws a box with rounded corners around its contents. In HTML, a normal rectangle is drawn. 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 by a cartouche for emphasis.

For example,`
@cartouche
@example
% pwd
/usr/local/share/emacs
@end example
@end cartouche`
surrounds the two-line example with a box with rounded corners, in the printed manual.

The output from the example looks like this (if you're reading this in Info, you'll see the `@cartouche` had no effect):

```
% pwd
/usr/local/share/emacs
```

A cartouche is output on a single page in printed output, similarly to `@group` (see Section 12.9 [@group], page 115).

7.15 `@small...` Block Commands

In addition to the regular `@example` and similar commands, Texinfo has “small” example-style commands. These are `@smallquotation`, `@smallindentedblock`, `@smalldisplay`, `@smallexample`, `@smallformat`, and `@smalllisp`.

In most output formats, the `@small...` commands are equivalent to their non-small companion commands.

In printed output, however, the `@small...` commands typeset text in a smaller font than the non-small example commands. Thus, for instance, code examples can contain longer lines and still fit on a page without needing to be rewritten.

Mark the end of a `@small...` block with a corresponding `@end small...`. For example, pair `@smallexample` with `@end smallexample`.

Here is an example of the font used by the `@smallexample` command (in most output formats, the output will be the same as usual):

```
... to make sure that you have the freedom to
distribute copies of free software (and charge for
this service if you wish), that you receive source
code or can get it if you want it, that you can
change the software or use pieces of it in new free
programs; and that you know you can do these things.

The @small... commands use the same font style as their normal counterparts: 
@smallexample and @smalllisp use a fixed-width font, and everything else uses the
regular font. They also have the same behavior in other respects—whether filling is done
and whether margins are narrowed.

As a general rule, it’s better to just use the regular commands (such as @example instead
of @smallexample), unless you have a good reason for it.
8 Lists and Tables

Texinfo has several ways of making lists and tables. Lists can be bulleted or numbered; two-column tables can highlight the items in the first column; multi-column tables are also supported.

8.1 Introducing Lists

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

Numbered lists and tables begin with the appropriate @-command at the beginning of a line, and end with the corresponding @end command on a line by itself. The table and itemized-list commands also require that you write formatting information on the same line as the beginning @-command.

Begin an enumerated list, for example, with an @enumerate command and end the list with an @end enumerate command. Begin an itemized list with an @itemize command, followed on the same line by a formatting command such as @bullet, and end the list with an @end itemize command.

Precede each element of a list with an @item or @itemx command.

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

- Itemized lists with and without bullets.
- Enumerated lists, using numbers or letters.
- Two-column tables with highlighting.

Here is an enumerated list with the same items:

1. Itemized lists with and without bullets.
2. Enumerated lists, using numbers or letters.
3. Two-column tables with highlighting.

And here is a two-column table with the same items and their @-commands:

@itemize Itemized lists with and without bullets.
@enumerate Enumerated lists, using numbers or letters.
@table @ftable @vtable Two-column tables, optionally with indexing.
8.2 @itemize: Making an Itemized List

The @itemize command produces a sequence of “items”, each starting with a bullet or other mark inside the left margin, and generally indented.

Begin an itemized list by writing @itemize at the beginning of a line. Follow the command, on the same line, with a character or a Texinfo command that generates a mark. Usually, you will use @bullet after @itemize, but you can use @minus, or any command or character that results in a single character in the Info file. (When you write the mark command such as @bullet after an @itemize command, you may omit the ‘{‘.) If you don’t specify a mark command, the default is @bullet. If you don’t want any mark at all, but still want logical items, use @w{} (in this case the braces are required).

After the @itemize, write your items, each starting with @item. Text can follow on the same line as the @item. The text of an item can continue for more than one paragraph.

There should be at least one @item inside the @itemize environment. If none are present, texi2any gives a warning. If you just want indented text and not a list of items, use @indentedblock; see Section 7.3 [@indentedblock], page 67.

Index entries and comments that are given before an @item including the first, are automatically moved (internally) to after the @item, so the output is as expected. Historically this has been a common practice.

Usually, you should put a blank line between items. This puts a blank line in the Info file. (TEX inserts the proper vertical space in any case.) Except when the entries are very brief, these blank lines make the list look better.

Here is an example of the use of @itemize, followed by the output it produces. @bullet produces an ‘*’ in Info and a round dot in other output formats.

@example
@itemize @bullet
@item
Some text for foo.
@item
Some text for bar.
@end itemize
@end example

This produces:

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

Itemized lists may be embedded within other itemized lists. Here is a list marked with dashes embedded in a list marked with bullets:
@itemize @bullet
@item
First item.

@itemize @minus
@item
Inner item.
@item
Second inner item.
@end itemize

@item
Second outer item.
@end itemize

This produces:
- First item.
  - Inner item.
  - Second inner item.
- Second outer item.

8.3 @enumerate: Making a Numbered or Lettered List

@enumerate is like @itemize (see Section 8.2 [@itemize], page 76), except that the labels on the items are successive integers or letters instead of bullets.

Write the @enumerate command at the beginning of a line. The command does not require an argument, but accepts either a number or a letter as an option. Without an argument, @enumerate starts the list with the number ‘1’. With a numeric argument, such as ‘3’, the command starts the list with that number. With an upper- or lowercase letter, such as ‘a’ or ‘A’, the command starts the list with that letter.

Write the text of the enumerated list in the same way as an itemized list: write a line starting with @item at the beginning of each item in the enumeration. It is ok to have text following the @item, and the text for an item can continue for several paragraphs.

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

Here is an example of @enumerate without an argument:

@enumerate
@item
Underlying causes.
@item
Proximate causes.
@end enumerate

This produces:
1. Underlying causes.
2. Proximate causes.

Here is an example with an argument of 3:

\begin{enumerate}
\item Predisposing causes.
\item Precipitating causes.
\item Perpetuating causes.
\end{enumerate}

This produces:
3. Predisposing causes.
4. Precipitating causes.
5. Perpetuating causes.

Here is a summary:

a. @enumerate
   Without an argument, produce a numbered list, with the first item numbered 1.
b. @enumerate unsigned-integer
   With an (unsigned) numeric argument, start a numbered list with that number. You
can use this to continue a list that you interrupted with other text.
c. @enumerate upper-case-letter
   With an uppercase letter as argument, start a list in which each item is marked by a
letter, beginning with that uppercase letter.
d. @enumerate lower-case-letter
   With a lowercase letter as argument, start a list in which each item is marked by a
letter, beginning with that lowercase letter.

You can also nest enumerated lists, as in an outline.

8.4 Making a Two-column Table
@table is similar to @itemize (see Section 8.2 [@itemize], page 76), but allows you to specify
a name or heading line for each item. The @table command is used to produce two-column
tables, and is especially useful for glossaries, explanatory exhibits, and command-line option
summaries.

8.4.1 Using the @table Command
Use the @table command to produce a two-column table. This command is typically used
when you have a list of items and a brief text with each one, such as a list of definitions.
Write the \texttt{@table} command at the beginning of a line, after a blank line, and follow it on the same line with an argument that is an ‘indicating’ command, such as \texttt{@code}, \texttt{@samp}, \texttt{@var}, \texttt{@option}, or \texttt{@kbd} (see Section 6.1 [Indicating], page 54). This command will be applied to the text in the first column. For example, \texttt{@table @code} will cause the text in the first column to be output as if it had been the argument to a \texttt{@code} command.

You may use the \texttt{@asis} command as an argument to \texttt{@table}. \texttt{@asis} is a command that does nothing: if you use this command after \texttt{@table}, the first column entries are output without added highlighting (“as is”).

The \texttt{@table} command works with other commands besides those explicitly mentioned here. However, you can only use predefined Texinfo commands that take an argument in braces. You cannot reliably use a new command defined with \texttt{@macro}, although an \texttt{@alias} (for a suitable predefined command) is acceptable. See Chapter 16 [Defining New Texinfo Commands], page 143.

Begin each table entry with an \texttt{@item} command at the beginning of a line. Write the text for the first column on the same line as the \texttt{@item} command. Write the text for the second column on the line following the \texttt{@item} line and on subsequent lines. You may write as many lines of supporting text as you wish, even several paragraphs. But only the text on the same line as the \texttt{@item} will be placed in the first column (including any footnotes). You do not need to type anything for an empty second column.

Normally, you should put a blank line between table entries. This puts a blank line in the Info file, which looks better unless the entries are very brief.

End the table with a line consisting of \texttt{@end table}. Follow the end of the table by a blank line for consistent formatting across output formats.

For example, the following table highlights the text in the first column with the \texttt{@samp} command:

\begin{verbatim}
@table @samp
@item foo
  This is the text for
  @samp{foo}.

@item bar
  Text for @samp{bar}.
@end table
\end{verbatim}

This produces:

\begin{verbatim}
‘foo’  This is the text for ‘foo’.
‘bar’  Text for ‘bar’.
\end{verbatim}

If you want to list two or more named items with a single block of text, use the \texttt{@itemx} command. (See Section 8.4.3 [\texttt{@itemx}], page 80.)

The \texttt{@table} command (see Section 8.4.1 [\texttt{@table}], page 78) is not supported inside \texttt{@display}. Since \texttt{@display} is line-oriented, it doesn’t make sense to use them together. If you want to indent a table, try \texttt{@quotation} (see Section 7.2 [\texttt{@quotation}], page 66) or \texttt{@indentedblock} (see Section 7.3 [\texttt{@indentedblock}], page 67).
8.4.2 @ftable and @vtable

The @ftable and @vtable commands are the same as the @table command except that @ftable automatically enters each of the items in the first column of the table into the index of functions and @vtable automatically enters each of the items in the first column of the table into the index of variables. This simplifies the task of creating indices. Only the items on the same line as the @item or @itemx commands are indexed, and they are indexed in exactly the form that they appear on that line. See Chapter 10 [Indices], page 90, for more information about indices.

Begin a two-column table using @ftable or @vtable by writing the @-command at the beginning of a line, followed on the same line by an argument that is a Texinfo command such as @code, exactly as you would for a @table command; and end the table with an @end ftable or @end vtable command on a line by itself.

See the example for @table in the previous section.

8.4.3 @itemx: Second and Subsequent Items

Use the @itemx command inside a table when you have two or more first column entries for the same item, each of which should appear on a line of its own.

Use @item for the first entry, and @itemx for all subsequent entries; @itemx must always follow an @item command, with no blank line intervening.

The @itemx command works exactly like @item except that it does not generate extra vertical space above the first column text in some output formats. If you have multiple consecutive @itemx commands, do not insert any blank lines between them.

For example,

@table @code
@item upcase
@itemx downcase
These two functions accept a character or a string as argument, and return the corresponding uppercase (lowercase) character or string.
@end table

This produces:

upcase
downcase
These two functions accept a character or a string as argument, and return the corresponding uppercase (lowercase) character or string.

(Note also that this example illustrates multi-line supporting text in a two-column table.)

8.5 @multitable: Multi-column Tables

@multitable allows you to construct tables with any number of columns, with each column having any width you like.

You define the column widths on the @multitable line itself, and write each row of the actual table following an @item command, with columns separated by a @tab command. Finally, @end multitable completes the table. Details in the sections below.
8.5.1 Multitable Column Widths

You can define the column widths for a multitable in two ways: as fractions of the line length; or with a prototype row. Mixing the two methods is not supported. In either case, the widths are defined entirely on the same line as the `@multitable` command.

1. To specify column widths as fractions of the line length, write `@columnfractions` and the decimal numbers (presumably less than 1; a leading zero is allowed and ignored) after the `@multitable` command, as in:

   `@multitable @columnfractions .33 .33 .33`

   The fractions need not add up exactly to 1.0, as these do not. This allows you to produce tables that do not need the full line length.

   When using `@columnfractions`, the leftmost column may appear slightly wider than you might expect, relative to the other columns. This is due to spacing between columns being included in the width of the other columns.

2. To specify a prototype row, write the longest entry for each column enclosed in braces after the `@multitable` command. For example:

   `@multitable {some text for column one} {for column two}`

   The first column will then have the width of the typeset ‘some text for column one’, and the second column the width of ‘for column two’.

   The prototype entries need not appear in the table itself.

   Although we used simple text in this example, the prototype entries can contain Texinfo commands; markup commands such as `@code` are particularly likely to be useful.

8.5.2 Multitable Rows

After the `@multitable` command defining the column widths (see the previous section), you begin each row in the body of a multitable with `@item`, and separate the column entries with `@tab`. Line breaks are not special within the table body, and you may break input lines in your source file as necessary.

You can also use `@headitem` instead of `@item` to produce a heading row. The TEX output for such a row is in bold, and the HTML and DocBook output uses the `<thead>` tag. In Info, the heading row is followed by a separator line made of dashes (’-‘ characters).

The command `@headitemfont` can be used in templates when the entries in a `@headitem` row need to be used in a template. It is a synonym for `@b`, but using `@headitemfont` avoids any dependency on that particular font style, in case we provide a way to change it in the future.

Here is a complete example of a multi-column table (the text is from The GNU Emacs Manual, see Section “Splitting Windows” in The GNU Emacs Manual):

   `@multitable @columnfractions .15 .45 .4`
   `@headitem Key @tab Command @tab Description`
   `@item C-x 2`
   `@tab @code{split-window-vertically}`
   `@tab Split the selected window into two windows, with one above the other.`
   `@item C-x 3`
@tab @code{split-window-horizontally}
@tab Split the selected window into two windows positioned side by side.
@item C-Mouse-2
@tab
@tab In the mode line or scroll bar of a window, split that window.
@end multitable

produces:

<table>
<thead>
<tr>
<th>Key</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-x 2</td>
<td>split-window-vertically</td>
<td>Split the selected window into two windows, with one above the other.</td>
</tr>
<tr>
<td>C-x 3</td>
<td>split-window-horizontally</td>
<td>Split the selected window into two windows positioned side by side.</td>
</tr>
<tr>
<td></td>
<td>C-Mouse-2</td>
<td>In the mode line or scroll bar of a window, split that window.</td>
</tr>
</tbody>
</table>
9 Special Displays

The commands in this chapter allow you to write text that is specially displayed (output format permitting), outside of the normal document flow.

One set of such commands is for creating “floats”, that is, figures, tables, and the like, set off from the main text, possibly numbered, captioned, and/or referred to from elsewhere in the document. Images are often included in these displays.

Another group of commands is for creating footnotes in Texinfo.

9.1 Floats

A float is a display which is set off from the main text. It is typically labeled as being a “Figure”, “Table”, “Example”, or some similar type.

A float is so-named because, in principle, it can be moved to the bottom or top of the current page, or to a following page, in the printed output. (Floating does not make sense in other output formats.) In every output format except for \LaTeX, however, this floating is unfortunately not yet implemented. Instead, the floating material is simply output at the current location, more or less as if it were an \texttt{@group} (see Section 12.9 [\texttt{@group}], page 115).

9.1.1 \texttt{@float [type][,label]: Floating Material}

To produce floating material, enclose the material you want to be displayed separate between \texttt{@float} and \texttt{@end float} commands, on lines by themselves.

Floating material often uses \texttt{@image} to display an already-existing graphic (see Section 9.2 [Images], page 85), or \texttt{@multitable} to display a table (see Section 8.5 [Multi-column Tables], page 80). However, the contents of the float can be anything. Here’s an example with simple text:

\begin{verbatim}
@float Figure,fig:ex1
  This is an example float.
@end float
\end{verbatim}

And the output:

This is an example float.

Figure 9.1

As shown in the example, \texttt{@float} takes two arguments (separated by a comma), \texttt{type} and \texttt{label}. Both are optional.

\begin{description}
  \item[type] Specifies the sort of float this is; typically a word such as “Figure”, “Table”, etc. If this is not given, and \texttt{label} is, any cross-referencing will simply use a bare number.
  \item[label] Specifies a cross-reference label for this float. If given, this float is automatically given a number, and will appear in any \texttt{@listoffloats} output (see Section 9.1.3 [\texttt{@listoffloats}], page 85). Cross references to \texttt{label} are allowed. For example, \texttt{\textasciitilde see \texttt{@ref{fig:ex1}}} will produce see Figure 9.1.
\end{description}

On the other hand, if \texttt{label} is not given, then the float will not be numbered and consequently will not appear in the \texttt{@listoffloats} output or be cross-referenceable.
Ordinarily, you specify both type and label, to get a labeled and numbered float.

In the \LaTeX output, code loading the float package is output in the preamble if \texttt{@float} are present. A \texttt{@float} with type ‘figure’ or ‘table’ (case insensitive) is already defined by the package. Other float types lead to the definition of a new float environment, with names based on the \texttt{@float} type with anything else than letters and ‘-’ removed.

In Texinfo, all floats are numbered in the same way: with the chapter number (or appendix letter), a period, and the float number, which simply counts 1, 2, 3, \ldots, and is reset at each chapter. Each float type is counted independently.

Floats within an \texttt{@unnumbered}, or outside of any chapter, are simply numbered consecutively from 1.

These numbering conventions are not, at present, changeable.

### 9.1.2 \texttt{@caption} & \texttt{@shortcaption}

You may write a \texttt{@caption} anywhere within a \texttt{@float} environment, to define a caption for the float. It is not allowed in any other context. \texttt{@caption} takes a single argument, enclosed in braces. Here’s an example:

```latex
\begin{float}
An example float, with caption.
\caption{Caption for example float.}
\end{float}
```

The output is:

An example float, with caption.

Caption for example float.

\texttt{@caption} can appear anywhere within the float; it is not processed until the \texttt{@end float}. The caption text is usually a sentence or two, but may consist of several paragraphs if necessary.

In the output, the caption always appears below the float; this is not currently changeable. It is preceded by the float type and/or number, as specified to the \texttt{@float} command (see the previous section).

The \texttt{@shortcaption} command likewise may be used only within \texttt{@float}, and takes a single argument in braces. The short caption text is used instead of the caption text in a list of floats (see the next section). Thus, you can write a long caption for the main document, and a short title to appear in the list of floats. For example:

```latex
\begin{float}
... as above ...
\shortcaption{Text for list of floats.}
\end{float}
```

The text for \texttt{@shortcaption} may not contain comments (\texttt{@c}), verbatim text (\texttt{@verb}), environments such as \texttt{@example}, footnotes (\texttt{@footnote}), multiple paragraphs or other complex constructs. The same constraints apply to \texttt{@caption} unless there is a \texttt{@shortcaption}. In \LaTeX a multi-paragraph \texttt{@caption} will lead to an error when formatted, unless there is a \texttt{@shortcaption}. 
9.1.3 \texttt{@listoffloats}: Tables of Contents for Floats

You can write a \texttt{@listoffloats} command to generate a list of floats for a given float type (see Section 9.1.1 \texttt{@float}, page 83), analogous to the document’s overall table of contents. Typically, it is written in its own \texttt{unnumbered} node to provide a heading and structure, rather like \texttt{@printindex} (see Section 10.5 [Printing Indices & Menus], page 93).

\texttt{@listoffloats} takes one optional argument, the float type. Here’s an example:

\begin{verbatim}
@node List of Figures
@unnumbered List of Figures
@listoffloats Figure
\end{verbatim}

Without any argument, \texttt{@listoffloats} generates a list of floats for which no float type was specified, i.e., no first argument to the \texttt{@float} command (see Section 9.1.1 \texttt{@float}, page 83).

Here’s what the output from \texttt{@listoffloats} looks like, given the example figure earlier in this chapter:

Figure 9.1: ................................................................. 83

Usually, each line in the list of floats contains the float type (if any), the float number, and the caption, if any—the \texttt{shortcaption} argument, if it was specified, else the \texttt{caption} argument. The page number may also be included, depending on output format.

Unnumbered floats (those without cross-reference labels) are omitted from the list of floats.

The formatting of \texttt{@listoffloats} depends on the output format. In Info, for example, a \texttt{@listoffloats} is formatted as a menu.

In \LaTeX\ output, \texttt{\listoffigures} is output for the ‘\texttt{figure}’ (case insensitive) float type, \texttt{\listoftables} is output for the ‘\texttt{table}’ (case insensitive) float type. For other float types, a specific \texttt{\listof} command is output.

9.2 Inserting Images

You can insert an image given in an external file with the \texttt{@image} command. Although images can be used anywhere, including the middle of a paragraph, we describe them in this chapter since they are most often part of a displayed figure or example.

9.2.1 Image Syntax

Here is the synopsis of the \texttt{@image} command:

\begin{verbatim}
@image{filename[, width[, height[, alttext[, extension]]]]}
\end{verbatim}

The \texttt{filename} argument is mandatory, and must not have an extension, because the different processors support different formats:

- \TeX\ (DVI output) reads the file \texttt{filename.eps} (Encapsulated PostScript format).
- \TeX\ (PDF output) reads \texttt{filename.pdf}, \texttt{filename.png}, \texttt{filename.jpg}, or \texttt{filename.jpeg} (in that order). It also tries uppercase versions of the extensions. The PDF format does not support EPS images, so such must be converted first.
- In Info, \texttt{filename.txt} is included verbatim (more or less as if it were in \texttt{@verbatim}). The Info output may also include a reference to \texttt{filename.png} or \texttt{filename.jpg}. (See below.)
In HTML, a reference to `filename.png`, `filename.jpg`, `filename.jpeg` or `filename.gif` (in that order) is output. If none of those exist, it gives an error, and outputs a reference to `filename.jpg` anyway.

In DocBook, references to `filename.eps`, `filename.gif`, `filename.jpeg`, `filename.jpg`, `filename.pdf`, `filename.png` and `filename.svg` are output, for every file found. Also, `filename.txt` is included verbatim, if present. (The subsequent DocBook processor is supposed to choose the appropriate one.)

For \LaTeX{}, `filename` without any extension is used; the subsequent \LaTeX{} processor is supposed to choose the appropriate image type.

For Info and HTML output, the optional fifth argument `extension` to `@image` is used for the file extension, if it is specified and the file is found. Any leading period should be included in `extension`. For example:

```
@image{foo,,,,.xpm}
```

If you want to install image files for use by Info readers too, we recommend putting them in a subdirectory like `foo-figures` for a package `foo`. Copying the files into `$(infodir)/foo-figures/` should be done in your `Makefile`.

The `width` and `height` arguments are described in the next section.

If an image is the first thing in a paragraph and followed by more text, then you should precede the `@image` command with `@indent` or `@noindent` to indicate the beginning of paragraph formatting. This is especially important for \TeX{} output to get correct paragraph indentation.

Use `@center` to center an image (see Section 2.8.3 `[titlefont @center @sp]`, page 20).

For HTML output, the `alt` attribute for inline images is set to the optional `alttext` (fourth) argument to `@image`, if supplied. If not supplied, the full file name of the image being displayed is used. The `alttext` is processed as Texinfo text, so special characters such as `"`, and `<` and `&` are escaped in the HTML output; also, you can get an empty `alt` string with `@-` (a command that produces no output; see Section 12.3 `[@- @hyphenation]`, page 114).

For Info output, the `alttext` string is also processed as Texinfo text and output. In this case, `\` is escaped as `\` and `"` as `\"`; no other escapes are done.

In Info output, a reference to the binary image file is written (trying `filename` suffixed with `extension`, `extension`, `.png`, or `.jpg`, in that order) if one exists. The `.txt` file is also literally included, if one exists. This way, Info readers which can display images (such as the Emacs Info browser, running under X) can do so, whereas Info readers which can only use text (such as the standalone Info reader) can display the textual version.

### 9.2.2 Image Scaling

The optional `width` and `height` arguments to the `@image` command (see the previous section) specify the size to which to scale the image. They are only taken into account in printed output.

In \TeX{}, if neither is specified, the image is presented in its natural size (given in the file); if only one is specified, the other is scaled proportionately; and if both are specified, both are respected, thus likely distorting the original image by changing its aspect ratio.

The `width` and `height` may be specified using any valid \TeX{} dimension, namely:

- pt point (72.27pt = 1in)
pc    pica (1pc = 12pt)
bp    big point (72bp = 1in)
in    inch
cm    centimeter (2.54cm = 1in)
mm    millimeter (10mm = 1cm)
dd    didot point (1157dd = 1238pt)
cc    cicero (1cc = 12dd)
sp    scaled point (65536sp = 1pt)

For example, the following will scale a file ridt.eps to one inch vertically, with the width
scaled proportionately:
@image{ridt,,1in}

For @image to work with TeX, the file epsf.tex must be installed somewhere that TeX
can find it. (The standard location is texmf/tex/generic/dvips/epsf.tex, where texmf
is a root of your TeX directory tree.) This file is included in the Texinfo distribution and is
also available from ftp://tug.org/tex/epsf.tex, among other places.

@image can be used within a line as well as for displayed figures. Therefore, if you intend
it to be displayed, be sure to leave a blank line before the command, or the output will run
into the preceding text.

Image scaling is presently implemented only in printed output, not in any other sort of
output.

9.3 Footnotes

A footnote is for a reference that documents or elucidates the primary text.\(^1\)

Footnotes are distracting; use them sparingly at most, and it is best to avoid them
completely. Standard bibliographical references are usually better placed in a bibliography
at the end of a document instead of in footnotes throughout.

9.3.1 Footnote Commands

In Texinfo, 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. Footnotes may be of any length (they will be broken across pages if necessary),
but are usually short. The template is:

ordinary text@footnote{text of footnote}

As shown here, the @footnote command should come right after the text being footnoted,
with no intervening space; otherwise, the footnote marker might end up starting a line.

For example, this clause is followed by a sample footnote\(^2\); in the Texinfo source, it looks
like this:

...a sample footnote@footnote{Here is the sample

---

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.
footnote.}; in the Texinfo source...

As you can see, this source includes two punctuation marks next to each other; in this case, ‘.;’ is the sequence. This is normal (the first ends the footnote and the second belongs to the sentence being footnoted), so don’t worry that it looks odd. (Another style, perfectly acceptable, is to put the footnote after punctuation belonging to the sentence, as in ‘;@footnote{...’)

In printed output formats, the reference mark for a footnote is a small, superscripted number; the text of the footnote appears 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)’. The reference mark is followed by a cross-reference link to the footnote text if footnotes are put in separate nodes (see Section 9.3.2 [Footnote Styles], page 88).

In the HTML output, footnote references are generally marked with a small, superscripted number which is rendered as a hypertext link to the footnote text.

Footnotes cannot be nested, and cannot appear in section headings of any kind or other “unusual” places.

A final tip: footnotes in the argument of an @item command for an @table must be entirely on the same line as the @item (as usual). See Section 8.4 [Two-column Tables], page 78.

9.3.2 Footnote Styles

Online formats have two footnote styles, which determine where the text of the footnote is located, the ‘end’ and ‘separate’ footnote style.

‘end’ For Info, in the ‘end’ 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 the Info output for a single footnote in the end-of-node style:

-------- Footnotes --------

(1) Here is a sample footnote.

In HTML, when the footnote style is ‘end’, or if the output is not split, footnotes are put at the end of each output file.

‘separate’ For Info, in the ‘separate’ node 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 which you use to reach the footnote node.

The name of the node with the footnotes 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: texinfo.info Node: Overview-Footnotes, Up: Overview

(1) The first syllable of "Texinfo" is pronounced like "speck", not "hex". . . .

In HTML, when the footnote style is ‘separate’, and the output is split, footnotes are placed in a separate file.

Unless your document has long and important footnotes (as in, say, Gibbon’s Decline and Fall . . .), we recommend the ‘end’ style, as it is simpler for readers to follow.

Use the @footnotestyle command to specify the 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

Write a @footnotestyle command in the Texinfo file preamble.
10 Indices

Using Texinfo, 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 other formats, links to the index entries location or to the associated nodes are generated.

Texinfo provides several predefined kinds of indices: 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. Lastly, you can define your own new indices.

10.1 Predefined Indices

Texinfo provides six predefined indices. Here are their nominal meanings, abbreviations, and the corresponding index entry commands:

- ‘cp’ \(\text{(cindex)}\) Concept index, for general concepts.
- ‘fn’ \(\text{(findex)}\) Function index, for function and function-like names (such as entry points of libraries).
- ‘ky’ \(\text{(kindex)}\) Keystroke index, for keyboard commands.
- ‘pg’ \(\text{(pindex)}\) Program index, for names of programs.
- ‘tp’ \(\text{(tindex)}\) Data type index, for type names (such as structures defined in header files).
- ‘vr’ \(\text{(vindex)}\) Variable index, for variable names (such as library global variables).

Not every manual needs all of these, and most manuals use only two or three at most. The present manual, for example, has two indices: a concept index and an @-command index. (The latter is actually the function index but is called a command index in the chapter heading.)

You are not required to use the predefined indices strictly 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 \texttt{@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.

On the other hand, it is best not to stray too far from the meaning of the predefined indices. Otherwise, in the event that your text is combined with other text from other manuals, the index entries will not match up. Instead, define your own new index (see Section 10.7 \[New Indices\], page 95).

We recommend having a single index in the final document whenever possible, however many source indices you use, since then readers have only one place to look. Two or more source indices can be combined into one output index by using the \texttt{@synindex} or \texttt{@syncodeindex} commands (see Section 10.6 \[Combining Indices\], page 94).

Index entry commands based on the the two letter index names are also valid, for example \texttt{@cpindex} can be used instead of \texttt{cindex}. 
10.2 Defining the Entries of an Index

The data to make an index come from many individual indexing commands scattered throughout the Texinfo source file. Each command says to add one entry to a particular index; after formatting, the index 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, on the rest of the line, by the entry.

For example, this section begins with the following five entries for the concept index:

@cindex Defining indexing entries  
@cindex Index entries, defining  
@cindex Entries for an index  
@cindex Specifying index entries  
@cindex Creating index entries

Each predefined index has its own indexing command—@cindex for the concept index, @findex for the function index, and so on, as listed in the previous section.

Index entries should precede the visible material that is being indexed. For instance:

@cindex hello  
Hello, there!

Among other reasons, that way following indexing links (in whatever context) ends up before the material, where readers want to be, instead of after.

In Info, an index is usually formatted as a menu. Try to avoid using a colon in index entries, as this may confuse some Info readers. See Section 3.8.4 [Menu Parts], page 35, for more information about the structure of a menu entry.

By default, entries for a concept index are printed in a roman font and entries for the other indices are printed in a typewriter font. You may change the way part of an entry is printed with the usual Texinfo commands, such as @file for file names (see Chapter 6 [Marking Text], page 54), and @r for the normal roman font (see Section 6.2.3 [Fonts], page 63).

You may specify an explicit sort key for an index entry using @sortas following either the index command or the text of the entry. For example: '@findex @sortas{} \ @r{(literal in @code{@@math})}' sorts the index entry this produces under backslash.

You may choose to ignore certain characters in index entries for the purposes of sorting. The characters that you can currently choose to ignore are ‘\’, ‘-’, ‘<’ and ‘@’, which are ignored by giving as an argument to the @set command, respectively, txiindexbackslashignore, txiindexhyphenignore, txiindexlessthanignore and txiindexatsignignore. For example, specifying ‘@set txiindexbackslashignore’ causes the ‘\mathopsup’ entry in the index for this manual to be sorted as if it were ‘mathopsup’, so that it appears among the other entries beginning with ‘M’. This avoids the need to provide explicit sort keys for index entries containing these characters.

When using these options, it is possible to get an index entry with an empty sort key. To avoid this, specify a @sortas directive in the index entry. For example:

@set txiindexbackslashignore  
@findex @sortas{} \  

10.3 Advanced Indexing Commands

Texinfo provides several further commands for indexing.

First, you can create *multilevel* index entries, allowing you to group many related subtopics under the same higher-level topic. You do this by separating the parts of such an entry with the `@subentry` command. Such commands might look like this:

```
@cindex Superhumans @subentry villains
@cindex Superhumans @subentry heroes
```

You may have up to three levels in an entry:

```
@cindex coffee makers @subentry electric @subentry pink
@cindex coffee makers @subentry electric @subentry blue
```

You can use the `@sortas` command mentioned earlier with any or all of the three parts of an entry to cause them to sort differently than they would by default.

Second, you may provide an index entry that points to another, using the `@seeentry` ("see entry") command. For example:

```
@cindex Indexes @seeentry{Indices}
```

Such an entry should be unique in your document; the idea is to redirect the reader to the other entry where they will find all the information they are looking for.

Finally, you may provide a “see also” entry using the `@seealso` command. These entries go along with regular entries, and are grouped together with them in the final printed index. For example:

```
@cindex Coffee
@cindex Coffee @subentry With milk and sugar
@cindex Coffee @subentry With doughnuts
@cindex Coffee @subentry Decaffeinated
@cindex Coffee @seealso{Tea}
```

When using all three of these advanced commands, *do not* place a comma between the different parts of the index text. The `texindex` program, which sorts the index entries and generates the indexing formatting commands, takes care of placing commas in the correct places for you.

Do not interrupt an index or `@subentry` entry by `@sortas` or advanced commands with braces; place the commands with braces at the beginning or at the end of the entry.

These features are the most useful with printed documents, and when translating Texinfo to DocBook.

10.4 Making Index Entries

Concept index entries consist of text. The best way to write an index is to devise entries which are terse yet clear. If you can do this, the index usually looks better if the entries are written just as they would appear in the middle of a sentence, that is, capitalizing only proper names and acronyms that always call for uppercase letters. This is the case convention we use in most GNU manuals’ indices.

If you don’t see how to make an entry terse yet clear, make it longer and clear—not terse and confusing. If many of the entries are several words long, the index may look better
if you use a different convention: capitalize the first word of each entry. Whichever case
convention you use, use it consistently.

In any event, do not ever capitalize a case-sensitive name such as a C or Lisp function
name or a shell command; that would be a spelling error. Entries in indices other than the
concept index are symbol names in programming languages, or program names; these names
are usually case-sensitive, so likewise use upper- and lowercase as required.

It is a good idea to make index entries unique wherever feasible. That way, people using
the printed output or online completion of index entries don’t see undifferentiated lists.
Consider this an opportunity to make otherwise-identical index entries be more specific, so
readers can more easily find the exact place they are looking for. The advanced indexing
features described in Section 10.3 [Advanced Indexing], page 92, can help with this, as well.

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, one reader may think it obvious that the two-letter
names for indices should be listed under “Indices, two-letter names”, since “Indices” are 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 readers.

Like typesetting, the construction of an index is a skilled art, the subtleties of which may
not be appreciated until you need to do it yourself.

10.5 Printing Indices and Menus

@printindex takes one argument, a two-letter index abbreviation. You must include the
@printindex command at the place in the document where you want the index to appear.
This does not happen automatically just because you use @cindex or other index-entry
generating commands in the Texinfo file; those just cause the raw data for the index to be
accumulated.

You should precede the @printindex command with a suitable section or chapter
command (usually @appendix or @unnumbered) to supply the chapter heading and put
the index into the table of contents. Precede the chapter heading with an @node line as
usual.

For example:

@node Variable Index
@unnumbered Variable Index

@printindex vr

@node Concept Index
@unnumbered Concept Index

@printindex cp

The text ‘Index’ needs to appear in the name of the node containing the index for the
index to be found by Info readers.
If you have more than one index, we recommend placing the concept index last.

Other details of index output in output formats:

- As part of the process of creating a printed manual with \TeX, you run a program called \texttt{texindex} (see Chapter 18 [Hardcopy with \TeX], page 154) to sort the raw data to produce a sorted index file. The sorted index file is what is actually used to print the index.

  \texttt{@printindex} reads the corresponding sorted index file and produces a traditional two-column index, with index terms and page numbers.

- In Info output, \texttt{@printindex} produces a special menu containing the line number of the entry, relative to the start of the node. Info readers can use this to go to the exact line of an entry, not just the containing node. (Older Info readers will just go to the node.) Here’s an example:

  * First index entry: Top. (line 7)

- In plain text output, \texttt{@printindex} formatting is usually similar to a menu in Info, showing the line number of each entry relative to the start of the file.

- In HTML output, \texttt{@printindex} produces links to the index entries.

- In DocBook and \LaTeX output, it simply records the index to be printed.

### 10.6 Combining Indices

Sometimes you will want to combine two disparate indices such as functions and concepts, perhaps because you have few enough entries that a separate index would look silly.

You could put functions into the concept index by writing \texttt{@cindex} commands for them instead of \texttt{@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 as part of 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 need to enclose every one of them between the braces of \texttt{@code}.

#### 10.6.1 \texttt{@syncodeindex}: Combining Indices Using \texttt{@code}

When you want to combine functions and concepts into one index, you should index the functions with \texttt{@findex} and index the concepts with \texttt{@cindex}, and use the \texttt{@syncodeindex} command to redirect the function index entries into the concept index.

The \texttt{@syncodeindex} command takes two arguments; they are the name of the index to redirect, and the name of the index to redirect it to. The template looks like this:

\texttt{\@syncodeindex from to}

For this purpose, the indices are given two-letter names:

- ‘cp’ Concept index
- ‘fn’ Function index
- ‘ky’ Key index
Write a `@syncodeindex` command before or shortly after the end-of-header line at the beginning of a Texinfo file. For example, to merge a function index with a concept index, write the following:

```
@syncodeindex fn cp
```

This causes all entries designated for the function index to merge in with the concept index instead.

To merge both a variable index and a function index into a concept index, write the following:

```
@syncodeindex vr cp
@syncodeindex fn cp
```

The `@syncodeindex` command puts all the entries from the ‘from’ index (the redirected index) into the `@code` font, overriding whatever default font is used by the index to which the entries are now directed. This way, if you direct function names from a function index into a concept index, all the function names are printed in the `@code` font as you would expect.

### 10.6.2 `@synindex`: Combining Indices

The `@synindex` command is nearly the same as the `@syncodeindex` command, except that it does not put the ‘from’ index entries into the `@code` font; rather it puts them in the roman font. Thus, you use `@synindex` when you merge a concept index into a function index.

See Section 10.5 [Printing Indices & Menus], page 93, for information about printing an index at the end of a book or creating an index menu in an Info file.

### 10.7 Defining New Indices

In addition to the predefined indices (see Section 10.1 [Predefined Indices], page 90), you may use the `@defindex` and `@defcodeindex` commands to define new indices. These commands create new indexing `@-commands with which you mark index entries. The `@defindex` command is used like this:

```
@defindex name
```

New index names are usually two-letter words, such as ‘au’. For example:

```
@defindex au
```

This defines a new index, called the ‘au’ index. At the same time, it creates a new indexing command, `@auindex`, that you can use to make index entries. Use this new indexing command just as you would use a predefined indexing command.

For example, here is a section heading followed by a concept index entry and two ‘au’ index entries.

```
@section Cognitive Semantics
@cindex kinesthetic image schemas
@auindex Johnson, Mark
```
@auindex Lakoff, George

(Evidently, ‘au’ serves here as an abbreviation for “author”.)

Texinfo constructs the new indexing command by concatenating the name of the index with ‘index’; thus, defining an ‘xy’ index leads to the automatic creation of an @xyindex command.

Use the @printindex command to print the index, as you do with the predefined indices. For example:

@node Author Index
@unnumbered Author Index

@printindex au

The @defcodeindex command is like the @defindex command, except that, in the printed output, it prints entries in an @code font by default instead of in a roman font.

You should define new indices before the end-of-header line of a Texinfo file, and (of course) before any @synindex or @syncodeindex commands (see Section 2.5 [Texinfo File Header], page 12).

As mentioned earlier (see Section 10.1 [Predefined Indices], page 90), we recommend having a single index in the final document whenever possible (no matter how many source indices you use), since then readers have only one place to look.

When creating an index, TeX creates a file whose extension is the name of the index (see [Names of index files], page 156). Therefore you should avoid using index names that collide with extensions used for other purposes, such as ‘.aux’ or ‘.xml’. texi2any already reports an error if a new index conflicts well-known extension name.
11 Special Insertions

Texinfo provides several commands for inserting characters that have special meaning in
Texinfo, such as braces, and for other graphic elements that do not correspond to simple
characters you can type.

These are:

- The Texinfo special characters: '\texttt{@} {} , \# &'.
- Whitespace within and around a sentence.
- Accents.
- Dots and bullets.
- The \texttt{T\LaTeX} logo and the copyright symbol.
- The euro and pounds currency symbols.
- The degrees symbol.
- The minus sign.
- Mathematical expressions.
- Glyphs for examples of programming: evaluation, macros, errors, etc.
- Footnotes.

11.1 Special Characters: Inserting @ {} , \# &

'@' and curly braces are the basic special characters in Texinfo. To insert these characters so
they appear in text, you must put an '@' in front of these characters to prevent Texinfo from
misinterpreting them. Alphabetic commands are also provided.

The other characters (comma, backslash, hash, ampersand) are special only in restricted
contexts, as explained in the respective sections.

11.1.1 Inserting '@' with @{} and @atchar{}

@{} produces a single '@' character in the output. Do not put braces after an @{} command.

@atchar{} also produces a single '@' character in the output. It does need following
braces, as usual for alphabetic commands. In inline conditionals (see Section 15.4 [Inline
Conditionals], page 136), it can be necessary to avoid using the literal '@' character in the
source (and may be clearer in other contexts).

11.1.2 Inserting '{' with @{ @} and @1 rbracechar{}

@{} produces a single '{' in the output, and @} produces a single '}'. Do not put braces after
either an @{ or an @} command.

@lbracechar{} and @rbracechar{} also produce single '{' and '}' characters in the
output. They do need following braces, as usual for alphabetic commands. In inline
conditionals (see Section 15.4 [Inline Conditionals], page 136), it can be necessary to avoid
using literal brace characters in the source (and may be clearer in other contexts).
11.1.3 Inserting ‘,’ with \texttt{@comma{}}

Ordinarily, a comma ‘,’ is a normal character that can be simply typed in your input where you need it.

However, Texinfo uses the comma as a special character only in one context: to separate arguments to those Texinfo commands, such as \texttt{@node} (see Section 3.3 [Node Line Requirements], page 29), \texttt{@acronym} (see Section 6.1.14 [acronym], page 61) and \texttt{@xref} (see Chapter 5 [Cross References], page 43), as well as user-defined macros (see Section 16.1 [Defining Macros], page 143), which take more than one argument.

Since a comma character would confuse Texinfo’s parsing for these commands, you must use the command ‘\texttt{@comma{}}’ instead if you want to pass an actual comma. Here are some examples:

\begin{verbatim}
@acronym{ABC, A Bizarre @comma{}}
@xref{Comma,, The @comma{} symbol}
@mymac{One argument@comma{} containing a comma}
\end{verbatim}

Although ‘\texttt{@comma{}}’ can be used nearly anywhere, there is no need for it anywhere except in this unusual case.

(Incidentally, the name ‘\texttt{@comma{}}’ lacks the ‘\texttt{char}’ suffix used in its companion commands only for historical reasons. It didn’t seem important enough to define a synonym.)

11.1.4 Inserting ‘\’ with \texttt{@backslashchar{}}

Ordinarily, a backslash ‘\’ is a normal character in Texinfo that can be simply typed in your input where you need it. The result is to typeset the backslash from the typewriter font.

However, Texinfo uses the backslash as a special character in one restricted context: to delimit formal arguments in the bodies of user-defined macros (see Section 16.1 [Defining Macros], page 143).

Due to the vagaries of macro argument parsing, it is more reliable to pass an alphabetic command that produces a backslash instead of using a literal ‘\’. Hence \texttt{@backslashchar{}}. Here is an example macro call:

\begin{verbatim}
@mymac{One argument@backslashchar{} with a backslash}
\end{verbatim}

Texinfo documents may also use ‘\’ as a command character inside \texttt{@math} (see Section 11.7 [Inserting Math], page 105). In this case, ‘\backslash’ or ‘\texttt{backslash}’ produce a “math” backslash (from the math symbol font), while \texttt{@backslashchar{}} produces a typewriter backslash as usual.

Although ‘\texttt{@backslashchar{}}’ can be used nearly anywhere, there is no need for it except in these unusual cases.

11.1.5 Inserting ‘#’ with \texttt{@hashchar{}}

Ordinarily, a hash ‘#’ is a normal character in Texinfo that can be simply typed in your input where you need it. The result is to typeset the hash character from the current font.

This character has many other names, varying by locale, such as “number sign”, “pound”, and “octothorp”. It is also sometimes called “sharp” or “sharp sign” since it vaguely resembles the musical symbol by that name. In situations where Texinfo is used, “hash” is the most common in our experience.
However, Texinfo uses the hash character as a special character in one restricted context: to introduce the so-called \texttt{#line} directive and variants (see Section 16.6 [External Macro Processors], page 150).

So, in order to typeset an actual hash character in such a place (for example, in a program that needs documentation about \texttt{#line}), it’s necessary to use \texttt{@hashchar{}} or some other construct. Here’s an example:

\begin{verbatim}
@hashchar{} 10 "example.c"
\end{verbatim}

Although ‘@hashchar{}’ can be used nearly anywhere, there is no need for it anywhere except this unusual case.

11.1.6 Inserting ‘&’ with @& and @ampchar{}

Ordinarily, an ampersand ‘&’ is a normal character in Texinfo that can be simply typed in your input where you need it. The result is to typeset the ampersand character.

However, the ampersand character is treated specially in just one restricted context. In the argument to a definition command (see Chapter 13 [Definition Commands], page 117), an ampersand followed by a series of letters may be typeset specially when processing with \TeX\footnote{This syntax is used for Emacs Lisp keywords. See Section “A Sample Function Description” in GNU Emacs Lisp Reference Manual.} (see Section 13.7 [Def Cmd Conventions], page 127). For example:

\begin{verbatim}
@defun foo integer1 &optional integer2 &rest integers
  @code{foo} described here.
@end defun
\end{verbatim}

may have ‘&optional’ and ‘&rest’ formatted in bold, even without any @-command involved.

So, in order to typeset an ampersand in such a context, it’s necessary to use @& or some other construct. Here’s an example of a C++ function taking a reference as a parameter:

\begin{verbatim}
@deftypefn Function int foo (const std::vector<int>@& @var{bar})
  Documentation of @code{foo}.
@end deftypefn
\end{verbatim}

This gives the output

\begin{verbatim}
int foo (const std::vector<int>& bar)  [Function]
  Documentation of foo.
\end{verbatim}

Although ‘@&’ and ‘@ampchar{}’ can be used nearly anywhere, there is no need for them anywhere except this unusual case.

11.2 Inserting Quote Characters

As explained in the early section on general Texinfo input conventions (see Section 2.1 [Conventions], page 9), Texinfo source files use the ASCII character ‘ (96 decimal) to produce a left quote (‘), and ASCII ’ (39 decimal) to produce a right quote (’). Doubling these input characters (‘‘ and ’’) produces double quotes (“ and ”). These are the conventions used by \TeX.

This works all right for text. However, in examples of computer code, readers are especially likely to cut and paste the text verbatim—and, unfortunately, some document
viewers will mangle these characters. (The free PDF reader xpdf works fine, but other PDF readers, both free and nonfree, have problems.)

If this is a concern for you, Texinfo provides these two commands:

\@codequoteundirected on-off
causes the output for the ' character in code environments to be the undirected single quote, like this: '

\@codequotebacktick on-off
causes the output for the ' character in code environments to be the backtick character (standalone grave accent), like this: `.

If you want these settings for only part of the document, @codequote... off will restore the normal behavior, as in @codequoteundirected off.

These settings affect \code, \example, \kbd, \samp, \verb, and \verbatim. See Section 6.1.1 [Useful Highlighting], page 54.

This feature can also be controlled by using \set to change the values of the corresponding variables txicodequoteundirected and txicodequotebacktick.

### 11.3 Inserting Space

The following sections describe commands that control spacing of various kinds within and after sentences.

#### 11.3.1 Multiple Spaces

Ordinarily, multiple whitespace characters (space, tab, and newline) are collapsed into a single space.

Occasionally, you may want to produce several consecutive spaces, either for purposes of example (e.g., what your program does with multiple spaces as input), or merely for purposes of appearance in headings or lists. Texinfo supports three commands: \@SPACE, \@TAB, and \@NL, all of which insert a single space into the output. (Here, \@SPACE represents an ‘@’ character followed by a space, i.e., ‘@ ’, \TAB represents an actual tab character, and \NL represents an ‘@’ character and end-of-line, i.e., when ‘@’ is the last character on a line.)

For example,

```
Spacey@ @ @ @
example.
```

produces

```
Spacey example.
```

Do not follow any of these commands with braces.

To produce a non-breakable space, see Section 12.6 [@tie], page 115.

#### 11.3.2 Not Ending a Sentence

When a period, exclamation point or question mark is at the end of a sentence, slightly more space is inserted after it in a typeset manual.

Usually, Texinfo can determine automatically when a period ends a sentence. However, special commands are needed in some circumstances. Use the @: command after a period,
question mark, exclamation mark or colon that should not be followed by extra space. This is necessary in the following situations:

1. After a period that ends a lowercase abbreviation which is not at the end of a sentence.
2. When a parenthetical remark in the middle of a sentence (like this one!) ends with a period, exclamation point or question mark, : should be used after the right parenthesis. Similarly for right brackets and right quotes (both single and double).

For example:

`'foo vs. : bar (or?) : baz'`,

The first line below shows the output, and for comparison, the second line shows the spacing when the `:` commands were not used.

foo vs. bar (or?) baz
foo vs. bar (or?) baz

If you look carefully, you will see a bit of extraneous space after the `vs.` and `(or?)`.

It may help you to remember what : does by imagining that it stands for an invisible lower-case character that stops a word ending in a period.

A few Texinfo commands force normal interword spacing, so that you don’t have to insert : where you otherwise would. These are the code-like highlighting commands, @var, @abbr, and @acronym (see Section 6.1.1 [Useful Highlighting], page 54). For example, in `@code{foo. bar}` the period is not considered to be the end of a sentence, and no extra space is inserted.

: has no effect on the HTML or DocBook output.

### 11.3.3 Ending a Sentence

As mentioned above, Texinfo normally inserts additional space after the end of a sentence. It uses the same heuristic for this as TeX: a sentence ends with a period, exclamation point, or question mark, either preceded or followed by optional closing punctuation, and then whitespace, and not preceded by a capital letter.

Use . instead of a period, ! instead of an exclamation point, and ? instead of a question mark at the end of a sentence that does end with a capital letter. Do not put braces after any of these commands. For example:

Give it to M.I.B. and to M.E.W. Also, give it to R.J.C.

Give it to M.I.B. and to M.E.W. Also, give it to R.J.C.

The output follows. In printed output and Info, you can see the desired extra whitespace after the ‘W’ in the first line.

Give it to M.I.B. and to M.E.W. Also, give it to R.J.C.

Give it to M.I.B. and to M.E.W. Also, give it to R.J.C.

In other output formats, . is equivalent to a simple ‘.’; likewise for ! and ?.

The “closing punctuation” mentioned above is defined as a right parenthesis (‘)’, right bracket (‘]’), or right quote, either single or double (‘‘’ and ‘’’’; the many possible additional Unicode right quotes are not included). These characters can be thought of as invisible with respect to whether a given period ends a sentence. (This is the same rule as TeX.) For instance, the periods in ‘foo.) Bar’ and ‘foo.’’ Bar’ do end sentences.
The meanings of @ and @., etc. in Texinfo are designed to work well with the Emacs sentence motion commands (see Section “Sentences” in The GNU Emacs Manual). It may help to imagine that the ‘@’ in ‘@.’, etc., is an invisible lower-case letter ‘a’ which makes an upper-case letter before it immaterial for the purposes of deciding whether the period ends the sentence.

A few Texinfo commands are not considered as being an abbreviation, even though they may end with a capital letter when expanded, so that you don’t have to insert @ and companions. Notably, this is the case for code-like highlighting commands, @var arguments ending with a capital letter, @LaTeX, and @TeX. For example, that sentence ended with ‘... @code{@@TeX}.’; @, was not needed. Similarly, in ... @var{VARNAME}. Text the period after VARNAME ends the sentence; there is no need to use @.

11.3.4 @frenchspacing val: Control Sentence Spacing

In American typography, it is traditional and correct to put extra space at the end of a sentence. This is the default in Texinfo (implemented in Info and printed output, not in other output formats). In French typography (and others), this extra space is wrong; all spaces are uniform.

Therefore Texinfo provides the @frenchspacing command to control the spacing after punctuation. It reads the rest of the line as its argument, which must be the single word ‘on’ or ‘off’ (always these words, regardless of the language of the document). Here is an example:

```latex
@frenchspacing on
This is text. Two sentences. Three sentences. French spacing.

@frenchspacing off
This is text. Two sentences. Three sentences. Non-French spacing.
```

produces:

This is text. Two sentences. Three sentences. French spacing.
This is text. Two sentences. Three sentences. Non-French spacing.

@frenchspacing also affects the output after @., @!, and @? (see Section 11.3.3 [Ending a Sentence], page 101).

@frenchspacing has no effect on the HTML or DocBook output.

11.3.5 @dmn{dimension}: Format a Dimension

You can use the @dmn command to format a dimension with just enough space for proper typesetting inserted in printed output. In other output formats, the formatting commands insert no space at all.

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 8.27 in wide.

Not everyone uses this style. Some people prefer ‘8.27 in.’ or ‘8.27 inches’. In these cases, however, you need to use @tie (see Section 12.6 [@tie], page 115) or @w (see Section 12.5
Chapter 11: Special Insertions

[[@w, page 114]) so that no line break can occur between the number and the dimension. Also, if you write a period after an abbreviation within a sentence (as with the ‘in.’ above), you should write ‘@:’ after the period to avoid inserting extra whitespace in printed output, as shown here. See Section 11.3.2 [Not Ending a Sentence], page 100.

11.4 Inserting Accents

Here is a table with the commands Texinfo provides for inserting floating accents. They all need an argument, the character to accent, which can either be given in braces as usual (@'{e}), or, as a special case, the braces can be omitted, in which case the argument is the next character (@'e). This is to make the source as convenient as possible to type and read, since accented characters are very common in some languages.

If the command is alphabetic, such as @dotaccent, then there must be a space between the command name and argument if braces are not used. If the command is non-alphabetic, such as @', then there must not be a space; the argument is the very next character.

Exception: the argument to @tieaccent must be enclosed in braces (since it is two characters instead of one).

In Info and plain text output, accent constructs are output as the true accented characters if the document encoding supports the required characters, unless the option --disable-encoding is given to texi2any (see Section 14.2 [@documentencoding], page 131). ASCII transliterations are used if the encoded characters are not output.

<table>
<thead>
<tr>
<th>Command</th>
<th>Output</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>@&quot;o</td>
<td>ö</td>
<td>umlaut accent</td>
</tr>
<tr>
<td>@'o</td>
<td>ó</td>
<td>acute accent</td>
</tr>
<tr>
<td>@,{c}</td>
<td>ç</td>
<td>cedilla accent</td>
</tr>
<tr>
<td>@=o</td>
<td>ô</td>
<td>macron/overbar accent</td>
</tr>
<tr>
<td>@^o</td>
<td>ò</td>
<td>circumflex accent</td>
</tr>
<tr>
<td>@'o</td>
<td>ò</td>
<td>grave accent</td>
</tr>
<tr>
<td>@&quot;o</td>
<td>õ</td>
<td>tilde accent</td>
</tr>
<tr>
<td>@dotaccent{o}</td>
<td>ó</td>
<td>overdot accent</td>
</tr>
<tr>
<td>@H{o}</td>
<td>Ő</td>
<td>long Hungarian umlaut</td>
</tr>
<tr>
<td>@ogonek{a}</td>
<td>å</td>
<td>ogonek</td>
</tr>
<tr>
<td>@ringaccent{c}</td>
<td>õ</td>
<td>ring accent</td>
</tr>
<tr>
<td>@tieaccent{oo}</td>
<td>öö</td>
<td>tie-after accent</td>
</tr>
<tr>
<td>@u{o}</td>
<td>û</td>
<td>breve accent</td>
</tr>
<tr>
<td>@ubaraccent{o}</td>
<td>û</td>
<td>underbar accent</td>
</tr>
<tr>
<td>@udotaccent{o}</td>
<td>û</td>
<td>underdot accent</td>
</tr>
<tr>
<td>@v{o}</td>
<td>û</td>
<td>caron/hack/check accent</td>
</tr>
</tbody>
</table>

This table lists the Texinfo commands for inserting other characters commonly used in languages other than English.

<table>
<thead>
<tr>
<th>Command</th>
<th>output</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>@exclamdown{}</td>
<td>ï</td>
<td>upside-down !</td>
</tr>
<tr>
<td>@questiondown{}</td>
<td>¿</td>
<td>upside-down ?</td>
</tr>
<tr>
<td>@aa{} @AA{}</td>
<td>å Å</td>
<td>a,A with circle</td>
</tr>
<tr>
<td>@ae{} @AE{}</td>
<td>æ Å</td>
<td>ae,AE ligatures</td>
</tr>
<tr>
<td>@dh{} @DH{}</td>
<td>ð Ð</td>
<td>Icelandic eth</td>
</tr>
<tr>
<td>@dotless{i}</td>
<td>i</td>
<td>dotless i</td>
</tr>
</tbody>
</table>
11. Inserting Quotation Marks

Use doubled single-quote characters to begin and end quotations: ‘‘...’’. Two single quotes are converted to left- and right-hand doubled quotation marks, “like this”.

You may occasionally need to produce two consecutive single quotes; for example, in documenting a computer language such as Maxima where ‘’ is a valid command. You can do this with the input ‘‘@w{}’’; the empty @w command stops the combination into the double-quote characters.

The left quote character (‘, ASCII code 96) used in Texinfo is a grave accent in ANSI and ISO character set standards. We use it as a quote character because that is how TeX is set up, by default.

Texinfo supports several other quotation marks used in languages other than English. Below is a table with the commands Texinfo provides for inserting quotation marks.

<table>
<thead>
<tr>
<th>Command</th>
<th>Glyph</th>
<th>Unicode name (point)</th>
</tr>
</thead>
<tbody>
<tr>
<td>@quotedblleft{} `&quot;</td>
<td>&quot;</td>
<td>Left double quotation mark (U+201C)</td>
</tr>
<tr>
<td>@quotedblright{} `'&quot;</td>
<td>&quot;</td>
<td>Right double quotation mark (U+201D)</td>
</tr>
<tr>
<td>@quoteleft{} `</td>
<td>'</td>
<td>Left single quotation mark (U+2018)</td>
</tr>
<tr>
<td>@quoteright{} `</td>
<td>'</td>
<td>Right single quotation mark (U+2019)</td>
</tr>
<tr>
<td>@quotedblbase{} `&quot;</td>
<td>&quot;</td>
<td>Double low-9 quotation mark (U+201E)</td>
</tr>
<tr>
<td>@quotesinglebase{} `</td>
<td>'</td>
<td>Single low-9 quotation mark (U+201A)</td>
</tr>
<tr>
<td>@guillemetleft{} «</td>
<td>«</td>
<td>Left-pointing double angle quotation mark (U+00AB)</td>
</tr>
<tr>
<td>@guillemetright{} »</td>
<td>»</td>
<td>Right-pointing double angle quotation mark (U+00BB)</td>
</tr>
<tr>
<td>@guilsingleleft{} ‹</td>
<td>‹</td>
<td>Single left-pointing angle quotation mark (U+2039)</td>
</tr>
<tr>
<td>@guilsingleright{} ›</td>
<td>›</td>
<td>Single right-pointing angle quotation mark (U+203A)</td>
</tr>
</tbody>
</table>

For the double angle quotation marks, Adobe and \LaTeX\ glyph names are also supported: @guillemotleft and @guillemotright. These names are incorrect; a “guillemot” is a bird species (a type of auk).

The standard \TeX\ fonts support the usual quotation marks used in English (the ones produced with single and doubled ASCII single quotes). For the other quotation marks, \TeX\ uses European Computer Modern (EC) fonts (ecrm1000 and other variants). These fonts are freely available, of course; you can download them from http://ctan.org/pkg/ec, among other places.
The free EC fonts are bitmap fonts created with Metafont. Especially for on-line viewing, Type 1 (vector) versions of the fonts are preferable; these are available in the CM-Super font package (http://ctan.org/pkg/cm-super).

Both distributions include installation instructions.

Traditions for quotation mark usage vary to a great extent between languages (https://en.wikipedia.org/wiki/Quotation_mark). Texinfo does not provide commands or configurations for typesetting quotation marks according to the numerous traditions. Therefore, you have to choose the commands appropriate for the language of your manual. Sometimes aliases (see Section 16.4 [alias], page 149) can simplify the usage and make the source code more readable. For example, in German, @quotedblbase is used for the left double quote, and the right double quote is the glyph produced by @quotedblleft, which is counterintuitive. Thus, in this case the following aliases would be convenient:

@alias lgqq = quotedblbase
@alias rgqq = quotedblleft

11.6 @sub and @sup: Inserting Subscripts and Superscripts

You can insert subscripts and superscripts with the @sub and @sup commands. For example:

```
@sub{below}@sup{above}
```

produces:

```
here^{below}_^{above}
```

In Info and plain text, @sub{text} is currently output as ‘_{text}’ and @sup{text} as ‘^{text}’, including the literal braces (to mark the beginning and end of the “script” text to the reader).

When the output format (and display program) permit (printed output, HTML), the superscript is set above the subscript when both commands are given consecutively.

For subscripts and superscripts in mathematical expressions, it is better to use T\TeX’s ‘\_' and ‘^’ characters. See the next section.

11.7 @math and @displaymath: Formatting Mathematics

You can write a mathematical expression, equation or formula using the @math command. Write the T\TeX math notation between braces, like this:

```
@math{\partial_\alpha \partial^\alpha A^\beta = \mu_0 J^\beta}
```

@math is formatted inline when used inside a paragraph, like \( \partial_\alpha \partial^\alpha A^\beta = \mu_0 J^\beta \) in this example. The @math command has no special effect on the Info output, merely outputting the contents verbatim.

For printed output, @math switches into T\TeX “math mode”. In that context, ‘\_’ must be used instead of ‘@’ for plain T\TeX math control sequences for symbols, functions, and so on.

By default, the HTML output is only emphasized. texi2any provides three options for displaying properly formatted mathematics for HTML. You can select these with the HTML-MATH variable (see Section 19.4.3 [HTML Customization Variables], page 174). With HTML-MATH set to ‘12h’, texi2any attempts to use the latex2html program to produce image files for mathematical material. With the ‘t4h’ setting, texi2any attempts to use the tex4ht program. With the ‘mathjax’ setting, texi2any inserts references in the output.
files to MathJax scripts to format the material. The MathJax option requires JavaScript to be enabled in the browser to work. See also Section 19.4.4 [MathJax Customization Variables], page 180, Section 19.4.5 [latex2html Customization Variables], page 181, and Section 19.4.6 [tex4ht Customization Variables], page 181.

For displayed equations, you can use the @displaymath command. Example:

\begin{displaymath}
f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{x-\mu}{\sigma} \right)^2}
\end{displaymath}

which produces:

\[ f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{x-\mu}{\sigma} \right)^2} \]

In general, the contents of @math or @displaymath should be plain \TeX only, with no interspersed Texinfo commands. If you do use any Texinfo commands, then you should mark these with '@' as usual, rather than '\' (e.g. '\var' rather than '\val'), but we do not guarantee which Texinfo commands will work.

Although @sub and @sup may work inside math mode in some contexts, it is better to use \TeX’s ‘\textsubscript’ and ‘\textsuperscript’ characters to denote subscripts and superscripts within mathematical expressions.

\LaTeX-specific code will only work when the output format is \LaTeX, and with the HTML_MATH options (although tex4ht needs T4H_MATH_CONVERSION to be set to ‘latex’; see Section 19.4.6 [tex4ht Customization Variables], page 181).

Due to the conflict with Texinfo’s @sup command, you can access the plain \TeX command \sup as \mathopsup instead, in the unlikely occurrence that you want to do this (but only when processing with \TeX, not when outputting \LaTeX nor with any of the HTML_MATH options).

11.8 Glyphs for Text

Texinfo has support for a few additional glyphs that are commonly used in printed text but not available in ASCII. Of course, there are many thousands more. It is possible to use Unicode characters as-is as far as texi2any is concerned, but \TeX is not so lucky.

11.8.1 @TeX{} (\TeX) and @LaTeX{} (\LaTeX)

Use the @TeX{} command to generate ‘\TeX’. In a printed manual, this is a special logo that is different from three ordinary letters. In other output formats, it just looks like ‘\TeX’.

Similarly, use the @LaTeX{} command to generate ‘\LaTeX’, which is even more special in printed manuals (and different from the incorrect La@TeX{}). In other output formats, the result is just ‘La\TeX’.

The spelling of these commands is unusual for Texinfo, in that they use both uppercase and lowercase letters.
11.8.2 \copyright{} (©)
Use the \copyright{} command to generate the copyright symbol, ‘©’. Where possible, this is a ‘c’ inside a circle; otherwise this is ‘(C)’.

Legally, it’s not necessary to use the copyright symbol; the English word ‘Copyright’ suffices, according to international treaty.

11.8.3 \_registers{} (®)
Use the \_registers{} command to generate the registered symbol, ‘®’. Where possible, this is an ‘R’ inside a circle; otherwise this is ‘(R)’.

11.8.4 @dots (...) and \enddots (...) 
An ellipsis (a sequence of dots) would be spaced wrong when typeset as a string of periods, so a special command is used in Texinfo: use the \dots{} command to generate a normal ellipsis, which is three dots in a row, appropriately spaced . . . like so. To emphasize: do not simply write three periods in the input file; that could work in some output formats, but would produce the wrong amount of space between the periods in printed manuals.

The \enddots{} command generates an end-of-sentence ellipsis, which also has three dots, but with different spacing afterwards, . . . Look closely to see the difference.

Here is an ellipsis: . . . Here are three periods in a row: ...

In printed (and usually HTML) output, the three periods in a row are much closer together than the dots in the ellipsis.

11.8.5 \bullet (•)
Use the \bullet{} command to generate a large round dot, or the closest possible thing to one. An asterisk can also be used. Here is a bullet: •.

When you use \bullet{} in \itemize{}, you do not need to type the braces, because \itemize{} supplies them. (see Section 8.2 \itemize{}, page 76).

11.8.6 \euro{}: Euro Currency Symbol
Use the \euro{} command to generate ‘€’. Where possible, this is the symbol for the Euro currency. Otherwise, the word ‘Euro’ is used.

The Euro symbol does not exist in the standard \TeX\ fonts (which were designed before the Euro was legislated into existence). Therefore, \TeX\ uses an additional font, named feymr10 (along with other variables). It is freely available, of course; you can download it from http://ctan.org/pkg/euromsym, among other places. The distribution includes installation instructions.

11.8.7 \pounds{} (£): Pounds Sterling
Use the \pounds{} command to generate ‘£’. Where possible, this is the symbol for the pounds sterling British currency. Otherwise, it is ‘#’.

11.8.8 \textdegree{} (°): Degrees Symbol
Use the \textdegree{} command to generate ‘°’. Where possible, this is the normal symbol for degrees. Otherwise, it is an ‘o’.
11.8.9 @minus (−): Inserting a Minus Sign

Use the @minus{}{} command to generate a minus sign. Where the character encoding and font used in the output allow it, the symbol is the customary length for a minus sign—a little longer than a hyphen, shorter than an em-dash:

‘−’ is a minus sign generated with ‘@minus{}{}’,

‘−’ is a hyphen generated with the character ‘−’,

‘—’ is an em-dash for text.

If you actually want to typeset some math that does a subtraction, it is better to use @math, as in @math{a-b} (see Section 11.7 [Inserting Math], page 105).

11.8.10 @geq (≥) and @leq (≤): Inserting Relations

Use the @geq{} and @leq{} commands to generate greater-than-or-equal and less-than-equal-signs, ‘≥’ and ‘≤’. When those symbols are not available, the ASCII sequences ‘>=’ and ‘<=’ are output.

11.9 Glyphs for Programming

In Texinfo, code is often illustrated in examples that are delimited by @example and @end example. In such examples, you can indicate the results of evaluation or an expansion using ‘⇒’ or ‘→’. Likewise, there are commands to insert glyphs to indicate printed output, error messages, equivalence of expressions, the location of point in an editor, and GUI operation sequences.

The glyph-insertion commands do not need to be used within an example, but most often they are. All glyph-insertion commands are followed by empty braces.

11.9.1 Glyphs Summary

Here is a summary of the glyph commands:

<table>
<thead>
<tr>
<th>Glyph</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>⇒</td>
<td>@result{} indicates the result of an expression.</td>
</tr>
<tr>
<td>→</td>
<td>@expansion{} indicates the results of a macro expansion.</td>
</tr>
<tr>
<td>−</td>
<td>@print{} indicates printed output.</td>
</tr>
<tr>
<td></td>
<td>@error{} indicates the following text is an error message.</td>
</tr>
<tr>
<td>≡</td>
<td>@equiv{} indicates the exact equivalence of two forms.</td>
</tr>
<tr>
<td>⋆</td>
<td>@point{} shows the location of point.</td>
</tr>
<tr>
<td>A → B</td>
<td>@clicksequence{A @click{} B} indicates a GUI operation sequence: first A, then clicking B, or choosing B from a menu, or otherwise selecting it.</td>
</tr>
</tbody>
</table>

11.9.2 @result{} (⇒): Result of an Expression

Use the @result{} command to indicate the result of evaluating an expression.

The @result{} command is displayed as ‘⇒’, either a double stemmed arrow or (when that is not available) the ASCII sequence ‘=>’.
Thus, the following,

```
(cdr '(1 2 3))
⇒ (2 3)
```

may be read as “(cdr '(1 2 3)) evaluates to (2 3)”.

### 11.9.3 \@expansion{} (\rightarrow\,): 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 ‘\rightarrow\,’ either a long arrow with a flat base or (when that is not available) the ASCII sequence ‘\Rightarrow’.

For example, the following
```
@example lisp
(third '(a b c))
@expansion{} (car (cdr (cdr '(a b c))))
@end example
```

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.

### 11.9.4 \@print{} (\dash\,): Indicating Generated Output

Sometimes an expression will generate output during its execution. You can indicate such displayed output with the \@print{} command.

The \@print{} command is displayed as ‘\dash\,’ either a horizontal dash butting against a vertical bar or (when that is not available) the ASCII sequence ‘\-\,\,\,\,’.

In the following example, the printed text is indicated with ‘\dash\,’ and the value of the expression follows on the last line.

```
(progn (print 'foo) (print 'bar))
\dash foo
\dash bar
⇒ bar
```

In a Texinfo source file, this example is written as follows:
```
@example lisp
(progn (print 'foo) (print 'bar))
@print{} foo
@print{} bar
@end example
```
11.9.5 \texttt{@error{}} (\texttt{error}): Indicating an Error Message

A piece of code may cause an error when you evaluate it. You can designate the error message with the \texttt{@error{}} command.

The \texttt{@error{}} command is displayed as \texttt{`error'}, either the word `error' in a box in the printed output, the word error followed by an arrow in other formats or (when no arrow is available) \texttt{`error-->'}.

Thus,

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

produces

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

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

\begin{verbatim}
Wrong type argument: integer-or-marker-p, x
\end{verbatim}

The word \texttt{`error'} itself is not part of the error message.

11.9.6 \texttt{@equiv{}} (\equiv): Indicating Equivalence

Sometimes two expressions produce identical results. You can indicate the exact equivalence of two forms with the \texttt{@equiv{}} command. The \texttt{@equiv{}} command is displayed as \texttt{`equiv'}, either a standard mathematical equivalence sign (three parallel horizontal lines) or (when that is not available) as the ASCII sequence \texttt{``==''}.

Thus,

\begin{verbatim}
@example lisp
(make-sparse-keymap) @equiv{} (list 'keymap)
@end example
\end{verbatim}

produces

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

This indicates that evaluating \texttt{(make-sparse-keymap)} produces identical results to evaluating \texttt{(list 'keymap)}.

11.9.7 \texttt{@point{}} (\star): 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 \texttt{`@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 \texttt{@point{}} command is displayed as \texttt{`*'}, either a pointed star or (when that is not available) the ASCII sequence \texttt{``-!'}. 
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 Texinfo source file, the example is written like this:

```
@example
---------- Buffer: foo ----------
This is the contents of foo.
---------- Buffer: foo ----------

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

### 11.9.8 Click Sequences

When documenting graphical interfaces, it is necessary to describe sequences such as ‘Click on ‘File’, then choose ‘Open’, then . . .’. Texinfo offers commands `@clicksequence` and `click` to represent this, typically used like this:

```
... @clicksequence{File @click{} Open} ...
```

which produces:

```
... File → Open ...
```

The `@click` command produces a right arrow by default; this glyph is also available independently via the command `@arrow{}`.

You can change the glyph produced by `@click` with the command `@clickstyle`, which takes a command name as its single argument on the rest of the line, much like `@itemize` and friends (see Section 8.2 [@itemize], page 76). The command should produce a glyph, and the usual empty braces `{}` are omitted. Here’s an example:

```
@clickstyle @result
... @clicksequence{File @click{} Open} ...
```

now produces:

```
... File ⇒ Open ...
```
11.10 Inserting Unicode: @U

The command @U{hex} inserts a representation of the Unicode character U+hex. For example, @U{0132} inserts the Dutch ‘IJ’ ligature (‘IJ’).

The hex value should be at least four hex digits; leading zeros are not added. In general, hex must specify a valid normal Unicode character; e.g., U+10FFFF (the very last code point) is invalid by definition, and thus cannot be inserted this way.

@U is useful for inserting occasional glyphs for which Texinfo has no dedicated command, while allowing the Texinfo source to remain purely 7-bit ASCII for maximum portability.

This command has many limitations—the same limitations as inserting Unicode characters in UTF-8 or another binary form. First and most importantly, TeX knows nothing about most of Unicode. Supporting specific additional glyphs upon request is possible, but it’s not viable for texinfo.tex to support whole additional scripts (Japanese, Urdu, . . .). The @U command does nothing to change this. If the specified character is not supported in TeX, an error is given. LaTeX output has more possibilities regarding UTF-8, but could require extra code to load fonts and declare how UTF-8 characters are output. (See Section 14.2 [documentencoding], page 131.)

In HTML and DocBook, the output from @U is always an entity reference of the form ‘&#xhex;’, as in ‘&#x0132;’ for the example above. This should work even when an HTML document uses some other encoding (say, Latin 1) and the given character is not supported in that encoding.

In Info and plain text, if the output encoding is not UTF-8, the output is the ASCII sequence ‘U+hex’, as in the six ASCII characters ‘U+0132’ for the example above.
12 Forcing and Preventing Breaks

Line and page breaks can sometimes occur in the ‘wrong’ place in one or another form of output. It’s up to you to ensure that text looks right in all the output formats.

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 normally occur.

You can use the break, break prevention, or pagination commands to fix problematic line and page breaks.

12.1 Break Commands

The break commands create or allow line and paragraph breaks:

- \texttt{@*} Force a line break.
- \texttt{@sp n} Skip \(n\) blank lines.
- \texttt{@-} Insert a discretionary hyphen.
- \texttt{@hyphenation{hy-phen-a-ted words}}
  Define hyphen points in 	extit{hy-phen-a-ted words}.

These commands hold text together on a single line:

- \texttt{@w{text}} Prevent \textit{text} from being split and hyphenated across two lines.
- \texttt{@tie{}} Insert a normal interword space at which a line break may not occur.

The pagination commands apply only to printed output, since other output formats do not have pages.

- \texttt{@page} Start a new page.
- \texttt{@group} Hold text together that must appear on one page.
- \texttt{@need \textit{mils}}
  Start a new page if not enough space on this one.

12.2 @* and @/: Generate and Allow Line Breaks

The \texttt{@*} command forces a line break in all output formats. The \texttt{@/} command allows a line break (printed manual only).

Here is an example with \texttt{@*}:

This sentence is broken \texttt{@*}into two lines.

produces

This sentence is broken
into two lines.

The \texttt{@/} command can be useful within long URLs or other identifiers where \TeX{} can’t find a good place to break. \TeX{} will automatically break URLs at the natural places (see Section 5.13.2 [URL Line Breaking], page 51), so only use \texttt{@/} if you need it. \texttt{@/} has no effect on the other output formats.
12.3 \@- and \@hyphenation: Hyphenation in Printed Output

Although \TeX's hyphenation algorithm is generally pretty good, it does miss useful hyphenation points from time to time. (Or, far more rarely, insert an incorrect hyphenation.) So, for documents with an unusual vocabulary or when fine-tuning for a printed edition, you may wish to specify hyphenation points explicitly. Texinfo supports two commands for this:

\@-

Insert a discretionary hyphen, i.e., a place where a word can be broken across lines with a hyphen. This is especially useful when you notice that an overfull hbox is due to \TeX missing a hyphenation (see Section 18.8 [Overfull hboxes], page 162). \TeX will not insert any hyphenation points itself into a word containing \@-.

\@hyphenation{hyphenated words}

Give hyphenation points for certain words. For example:

\@hyphenation{man-u-script man-u-scripts}

As shown, you put a ‘-’ at each hyphenation point. \TeX only uses the specified hyphenation points when the words match exactly, so give all necessary variants, such as plurals.

Non-printed output is not hyphenated, so none of these commands have any effect in other output formats.

12.4 \allowcodebreaks: Control Line Breaks in \code

Ordinarily, \TeX considers breaking lines at ‘-’ and ‘_’ characters within \code and related commands (see Section 6.1.2 [\code], page 55), more or less as if they were “empty” hyphenation points.

This is necessary since many manuals, especially for Lisp-family languages, must document very long identifiers. On the other hand, some manuals don’t have this problem, and you may not wish to allow a line break after the underscore in, for example, \texttt{SIZE\_MAX}, or even worse, after any of the four underscores in \texttt{\_\_typeof\_\_}.

So Texinfo provides this command:

\allowcodebreaks false

to prevent from breaking at ‘-’ or ‘_’ within \code. You can go back to allowing such breaks with \allowcodebreaks true. Write these commands on lines by themselves.

These commands can be given anywhere in the document. For example, you may have just one problematic paragraph where you need to turn off the breaks, but want them in general, or vice versa.

This command has no effect except in \TeX output.

12.5 \@w{text}: Prevent Line Breaks

\@w{text} outputs text, while prohibiting line breaks within text.

Thus, you can use \@w to produce a non-breakable space, fixed at the width of a normal interword space:

\@w{ } \@w{ } \@w{ } indentation.
produces:

    indentation.

The space from \texttt{\{\ }, as well as being non-breakable, also will not stretch or shrink. Sometimes that is what you want, for instance if you’re doing manual indenting. However, usually you want a normal interword space that does stretch and shrink (in the printed output); for that, see the \texttt{\{tie\} command in the next section.

In printed output, you can also use the \texttt{\{w\} command to prevent a long name or phrase being automatically hyphenated, for example if it happens to fall near the end of a line.

You can also use \texttt{\{w\} to avoid unwanted keyword expansion in source control systems. For example, to literally write \texttt{$Id$} in your document, use \texttt{\{w\}{$Id$}. This trick isn’t effective for output files in some output formats, though.

### 12.6 \{tie\}: Inserting an Unbreakable Space

The \texttt{\{tie\} command produces a normal interword space at which a line break may not occur. Always write it with following (empty) braces, as usual for commands used within a paragraph. Here’s an example:

    \TeX{} was written by Donald E.\{tie\}Knuth.

produces:

    \TeX{} was written by Donald E. Knuth.

There are two important differences between \texttt{\{tie\}} and \texttt{\{w\}}:

- The space produced by \texttt{\{tie\}} will stretch and shrink slightly along with the normal interword spaces in the paragraph; the space produced by \texttt{\{w\}} will not vary.
- \texttt{\{tie\}} allows hyphenation of the surrounding words, while \texttt{\{w\}} inhibits hyphenation of those words (for \TeX{}nical reasons, namely that it produces an ‘\hbox’).

### 12.7 \{sp \textit{n}\}: Insert Blank Lines

A line beginning with and containing only \texttt{\{sp \textit{n}\} generates \textit{n} blank lines of space. \texttt{\{sp \}} also forces a paragraph break. For example,

    \{sp 2

generates two blank lines.

The \texttt{\{sp \}} command is most often used in the title page.

### 12.8 \{page\}: Start a New Page

A line containing only \texttt{\{page\} starts a new page in a printed manual. In other formats, without the concept of pages, it starts a new paragraph. A \texttt{\{page\} command is often used in the \texttt{\{titlepage\} section of a Texinfo file to start the copyright page.

### 12.9 \{group\}: Prevent Page Breaks

The \texttt{\{group\} command is used to hold an example together on one page. Use it inside an \texttt{\{example\} or similar construct to begin an unsplittable vertical group, which will appear entirely on one page in the printed output. Terminate the group by a line containing only \texttt{\{end group\}. This command has an effect in \TeX{} output only.
The `@group` and `@end group` commands need to be inside the `@example` and `@end example` commands, thus:

```
@example
@group
...
@end group
@end example
```

Although `@group` would make sense conceptually in a wide variety of contexts, its current implementation works only within `@example` and variants, and within `@display`, `@format`, `@flushleft` and `@flushright`. See Chapter 7 [Quotations and Examples], page 65. (What all these commands have in common is that each line of input produces a line of output.)

### 12.10 `@need` mils: 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. Do not use braces around the argument \( n \). The `@need` command has no effect on other output formats since they are not paginated.

This paragraph is preceded by a `@need` command that starts a new page in printed output if fewer than 800 mils (eight-tenths inch) remain on the page. It looks like this:

```
@need 800
This paragraph is preceded by ...
```

The `@need` command is useful for preventing orphans: single lines at the bottoms of printed pages.
13 Definition Commands

The @deffn command and the other definition commands enable you to describe functions, variables, macros, commands, user options, special forms and other entities in a uniform format.

In the output, the name of the entity is shown with any arguments, along with the entity category —‘Function’, ‘Variable’, or whatever. Underneath, the body of the definition is indented. The name of the entity is also entered into the appropriate index.

All the definition commands follow a similar format. This chapter starts by describing @deffn before detailing all the other definition commands.

13.1 The Template for a Definition

@deffn category name arguments...
body-of-definition
@end deffn

The @deffn command is used for definitions of entities that resemble functions—entities that may take arguments. Write the @deffn command at the beginning of a line and follow it on the same line by the category of the entity, the name of the entity itself, and its arguments (if any). Then write the body of the definition on succeeding lines. Finally, end the definition with an @end deffn command written on a line of its own.

For example,

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

produces

forward-word count
This command moves point forward count words (or backward if count is negative). ...

Capitalize the category name like a title. If the name of the category contains spaces, as in the phrase ‘Interactive Command’, enclose it in braces. For example:

@deffn {Interactive Command} isearch-forward
...
@end deffn

Otherwise, the second word will be mistaken for the name of the entity. As a general rule, when any of the arguments in the heading line except the last one are more than one word, you need to enclose them in braces. This may also be necessary if the text contains commands, for example, ‘{declaraci@'on}’ if you are writing in Spanish.

The category is output in a different location for different output formats. For example, in the Info file, the category appears at the beginning of the first line of the definition. With TpX output, the category is printed next to the right margin.

@deffn enters names into the index of functions.
Three predefined, specialized variations of @ndefn (~defun, ~defmac, and ~defspec) specify the category for you: “Function”, “Macro”, and “Special Form” respectively. (In Lisp, a special form is an entity much like a function.) Similarly, the general ~defvr command is accompanied by several specialized variations for describing particular kinds of variables.

13.2 Definition Command Continuation Lines

The heading line of a definition command can get very long. Therefore, Texinfo has a special syntax allowing them to be continued over multiple lines of the source file: a lone '@' at the end of each line to be continued. Here’s an example:

@defun fn-name @
  arg1 arg2 arg3
This is the basic continued defun.
@end defun

produces:

\begin{verbatim}
fn-name arg1 arg2 arg3
  This is the basic continued defun.
\end{verbatim}

As you can see, the continued lines are combined, as if they had been typed on one source line.

Although this example only shows a one-line continuation, continuations may extend over any number of lines, in the same way; put an '@' at the end of each line to be continued.

In general, any number of spaces or tabs before the '@' continuation character are collapsed into a single space. There is one exception: the Texinfo processors will not fully collapse whitespace around a continuation inside braces. For example:

@deffn {\texttt{Category \@ \texttt{Name}}} ...

The output (not shown) has excess space between ‘Category’ and ‘Name’. To avoid this, elide the unwanted whitespace in your input, or put the continuation '@' outside braces.

'@' does not function as a continuation character in any other context. Ordinarily, '@' followed by a whitespace character (space, tab, newline) produces a normal interword space (see Section 11.3.1 [Multiple Spaces], page 100).

13.3 Optional and Repeated Arguments

Some entities take optional or repeated arguments. One convention for indicating these uses square brackets and ellipses: an argument enclosed within square brackets is optional, and an argument followed by an ellipsis is optional and may be repeated more than once.

Thus, [optional-arg] means that optional-arg is optional and 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 definition line of an example of an imaginary (complicated) special form:

\begin{verbatim}
foobar (var [from to [inc]]) body...
\end{verbatim}

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. These arguments are
grouped with the argument \texttt{var} into a list, to distinguish them from \textit{body}, which includes all remaining elements of the form.

In a Texinfo source file, this \texttt{@defspec} line is written like this:

\begin{verbatim}
@defspec foobar (var [from to [inc]]) body@dots{}
\end{verbatim}

\section*{13.4 Omitting the Space After a Definition Name}

As a matter of style or as demanded by the syntax of a programming language, you may wish to omit any space after a name in a definition, occurring before an opening bracket. To do this, set the '\texttt{txidefnamenospace}' flag (see Section 15.5.1 [\texttt{@set @value}], page 137). For example, this input

\begin{verbatim}
@set txidefnamenospace
@deffn Builtin index (string, substring) @dots{}
@end deffn
\end{verbatim}

produces the following:

\begin{verbatim}
index(string, substring) [Builtin]
\end{verbatim}

\section*{13.5 \texttt{@deffnx}, et al.: Two or More ‘First’ Lines}

To create two or more ‘first’ or header lines for a definition, follow the first \texttt{@deffn} line by a line beginning with \texttt{@deffnx}.

For example,

\begin{verbatim}
@deffn {Interactive Command} isearch-forward
@deffnx {Interactive Command} isearch-backward
These two search commands are similar except ...
@end deffn
\end{verbatim}

produces

\begin{verbatim}
isearch-forward [Interactive Command]
isearch-backward [Interactive Command]
These two search commands are similar except ...
\end{verbatim}

Each definition command has an ‘x’ form: \texttt{@defunx}, \texttt{@defvrx}, \texttt{@deftypefunx}, etc.

The ‘x’ forms work similarly to \texttt{@itemx} (see Section 8.4.3 [\texttt{@itemx}], page 80).

\section*{13.6 The Definition Commands}

This section describes all the definition commands of Texinfo.

\subsection*{13.6.1 Functions and Similar Entities}

This section describes the commands for describing functions and similar entities with simple arguments:

\begin{verbatim}
@deffn category name arguments...
\end{verbatim}

The \texttt{@deffn} command is the general definition command for functions, interactive commands, and similar entities that may take simple 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 \texttt{@deffn} command is written at the beginning of a line and is followed on the same line by the category of entity being described, the name of this particular entity, and its arguments, if any. Terminate the definition with \texttt{@end deffn} on a line of its own.

For example, here is a definition:

\begin{verbatim}
@deffn Command forward-char nchars
    Move point forward \texttt{@var{nchars}} characters.
@end deffn
\end{verbatim}

This shows a rather terse definition for a “command” named \texttt{forward-char} with one argument, \texttt{nchars}.

Where the output format allows, \texttt{@deffn} uses a typewriter font for \texttt{name}, and a slanted font for the rest of the arguments, as would be produced by \texttt{@var}.

Within the text of the description, write an argument name explicitly with \texttt{@var} to refer to the value of the argument. In the example above, we used ‘\texttt{@var{nchars}}’ in this way.

In the extremely unusual case when an argument name contains ‘\texttt{--}’, or another character sequence which is treated specially (see Section 2.1 [Conventions], page 9), use \texttt{@code} around the special characters. This avoids the conversion to typographic en-dashes and em-dashes.

\begin{verbatim}
@defun name arguments... 
    The \texttt{@defun} command is the definition command for functions with simple arguments. \texttt{@defun} is equivalent to ‘@texttt{@deffn Function ...}’. Terminate the definition with \texttt{@end defun} on a line of its own.
@end defun
\end{verbatim}

\begin{verbatim}
@defmac name arguments... 
    The \texttt{@defmac} command is the definition command for macros. \texttt{@defmac} is equivalent to ‘@texttt{@deffn Macro ...}’ and works like \texttt{@defun}.
@end defmac
\end{verbatim}

\begin{verbatim}
@defspec name arguments... 
    The \texttt{@defspec} command is the definition command for special forms. (In Lisp, a special form is an entity much like a function; see Section “Special Forms” in GNU Emacs Lisp Reference Manual.) \texttt{@defspec} is equivalent to ‘@texttt{@deffn {Special Form} ...}’ and works like \texttt{@defun}.
@end defspec
\end{verbatim}

All these commands create entries in the index of functions.

13.6.2 Functions in Typed Languages

The \texttt{@deftypefn} command and its variants are generic commands for describing functions. They are particularly suitable for languages in which you must declare types of variables and functions, such as C and C++.

\begin{verbatim}
@deftypefn category data-type name arguments... 
    The \texttt{@deftypefn} command is the general definition command for functions and similar entities that may take arguments and that could be typed. The \texttt{@deftypefn} command is written at the beginning of a line and is followed on the same line by the category of entity being described, information on the returned value, the name of this particular entity, and its arguments, if any.
@end deftypefn
\end{verbatim}
For example,

```latex
@deftypefn {Library Function} int foobar @
  (int @var{foo}, float @var{bar})
...
@end deftypefn
```

produces:

```latex
int foobar (int foo, float bar) [Library Function]
...
```

This means that `foobar` is a “library function” that returns an `int`, and its arguments are `foo` (an `int`) and `bar` (a `float`). ‘Library Function’ has to be enclosed in braces to make it a single argument.

When using `@deftypefn` command and variations, you should mark parameter names with `@var` to distinguish these from data type names, keywords, and other parts of the literal syntax of the programming language. Where the output format allows, all output on the definition line is in a typewriter font by default. `@var` uses an appropriate font where it occurs.

If you are describing a procedure in a language that has packages, such as Ada, you might consider using `@deftypefn` in the following manner:

```latex
@deftypefn stacks private push @
  (@var{s}:in out stack; @
   @var{n}:in integer)
...
@end deftypefn
```

(In these examples the `@deftypefn` arguments are shown using continuations (see Section 13.2 [Def Cmd Continuation Lines], page 118), but could be on a single line.)

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`.) Output:

```latex
private push (s:in out stack; n:in integer) [stacks]
...
```

`@deftypefn` and variants could be used when it is convenient to interpret the arguments list as literal computer code, marking argument names with `@var`. It is not necessary for any type names to appear, and you can give an empty return type as `{}`. For example, for documenting the R language, you could use input like

```latex
@deftypefn stats {} t.test (@var{x}, @var{y} = NULL,@
  @w{@var{alternative} = c("two.sided", "less", "greater")},@ 
  @w{@var{mu} = 0}, @w{@var{paired} = FALSE},@
  @w{@var{var.equal} = FALSE},@ 
  @w{@var{conf.level} = 0.95}, @dots{})
...
```
Here the \texttt{\textbackslash w} command is used to prevent some line breaks.

The information on return values may be used to name the return variables. For example, in Perl, returned scalars may be given the \texttt{$result} and \texttt{$status} name in the following manner:

\begin{verbatim}
@deftypefn {Sub} {(@var{$result}, @var{$status} =)} @
    process (@var{$input})
... Set @var{$status} to 0 in case of failure ...
@end deftypefn
\end{verbatim}

\begin{verbatim}
($result, $status) = process ($input)
... Set $status to 0 in case of failure ...
\end{verbatim}

@deftypefn creates an entry in the index of functions for \textit{name}.

\texttt{\textbackslash deftypefun data-type name arguments...}

The \texttt{\textbackslash deftypefun} command is the specialized definition command for functions.

The command is equivalent to ‘\texttt{\textbackslash deftypefn Function ...\textbackslash}’.

@deftypefun creates an entry in the index of functions for \textit{name}.

Ordinarily, the return type is printed on the same line as the function name and arguments, as shown above. In source code, GNU style for typed functions is to put the return type on a line by itself. So Texinfo provides an option to do that: \texttt{\textbackslash deftypefnnennewline on}.

This affects the generic functions only—not untyped functions with simple arguments such as \texttt{\textbackslash deffn}, not typed variables, etc. Specifically, it affects the commands in this section, and the analogous commands for object-oriented languages, namely \texttt{\textbackslash deftypemethod} (see Section 13.6.6.2 [Object-Oriented Methods], page 126).

Specifying \texttt{\textbackslash deftypefnnennewline off} reverts to the default.

13.6.3 Variables and Similar Entities

Here are the commands for defining variables and similar entities:

@defvr category name

The \texttt{\textbackslash 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 \texttt{\textbackslash defvr} command at the beginning of a line and follow it on the same line by the category of the entity and the name of the entity.
We recommend capitalizing the category name like a title. If the name of the category contains spaces, as in the name “User Option”, enclose it in braces. Otherwise, the second word will be mistaken for the name of the entity. For example,

```lisp
@defvr {User Option} fill-column
  This buffer-local variable specifies the maximum width of filled lines.
...
@end defvr
```
Terminate the definition with `@end defvr` on a line of its own.

`@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:

```lisp
@defvar kill-ring
...
@end defvar
```

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

**@defopt name**

The `@defopt` command is the definition command for user options, i.e., variables intended for users to change according to taste; Emacs has many such (see Section “Variables” in *The GNU Emacs Manual*). `@defopt` is equivalent to `@defvr {User Option} ...`.

### 13.6.4 Variables in Typed Languages

Variables in typed languages are handled in a manner similar to functions in typed languages. See Section 13.6.2 [Typed Functions], page 120. 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 on the same line by the category of the entity being described, the data type, and the name of this particular entity.

For example:

```lisp
@deftypevr {Global Flag} int enable
...
@end deftypevr
```
produces the following:
int enable  
[Global Flag]

@deftypevar data-type name

The @deftypevar command is the specialized definition command for variables in typed languages. @deftypevar is equivalent to ‘@deftypevr Variable ...’.

These commands create entries in the index of variables.

13.6.5 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 on the same line 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 ...
@end deftp

@deftp creates an entry in the index of data types.

13.6.6 Object-Oriented Programming

Here are the commands for formatting descriptions about abstract objects, such as are used in object-oriented programming. A class is a defined type of abstract 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.

13.6.6.1 Object-Oriented Variables

These commands allow you to define different sorts of variables in object-oriented programming languages.

@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. For instance:

@defcv {Class Option} Window border-pattern ...
@end defcv

produces:
border-pattern [Class Option of Window]

@defcv creates an entry in the index of variables.

@deftypecv category class data-type name
The @deftypecv command is the definition command for typed class variables in object-oriented programming. It is analogous to @defcv with the addition of the data-type parameter to specify the type of the instance variable.

@deftypecv {Class Option} Window int border-pattern
... 
@end deftypecv
produces:

int border-pattern [Class Option of Window]
...

@deftypecv 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} ...’. For instance:

@defivar Window border-pattern
...
@end defivar
produces:

border-pattern [Instance Variable of Window]
...

@defivar creates an entry in the index of variables.

@deftypeivar class data-type name
The @deftypeivar command is the definition command for typed instance variables in object-oriented programming. It is analogous to @defivar with the addition of the data-type parameter to specify the type of the instance variable.

@deftypeivar Window int border-pattern
...
@end deftypeivar
produces:

int border-pattern [Instance Variable of Window]
...

@deftypeivar creates an entry in the index of variables.
13.6.6.2 Object-Oriented Methods

These commands allow you to define different sorts of function-like entities resembling methods in object-oriented programming languages. These entities take arguments, as functions do, but are associated with particular classes of objects.

@defop category class name arguments...

The @defop command is the definition command for these method-like entities with simple arguments.

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 on the same line 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, if any.

@defop creates an entry, such as ‘expose on windows’, in the index of functions.

@deftypeop category class data-type name arguments...

The @deftypeop command is the generic definition command for operations in object-oriented programming. It is particularly suitable for typed object-oriented languages. It is similar to @defop with the addition of the data-type parameter to specify information on the return value of the method, for example the return type.

@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
    ...
@end defmethod
Chapter 13: Definition Commands

illustrates the definition for a method called \texttt{bar-method} of the class \texttt{bar-class}. The method takes an argument.

\@define\texttt{method} creates an entry in the index of functions.

\@deftype\texttt{method} \texttt{class data-type name arguments}...

The \@deftype\texttt{method} command is the definition command for methods in object-oriented languages, in particular for typed languages such as C++ and Java. It is similar to the \@define\texttt{method} command with the addition of the \texttt{data-type} parameter to specify information on the return value of the method, for example the return type.

The commands with information on return values are affected by the \@deftype\texttt{fnnewline} option (see Section 13.6.2 [Functions in Typed Languages], page 120).

13.7 Conventions for Writing Definitions

A manual need not and should not contain more than one definition for a given name. An appendix containing a summary should use \@table rather than the definition commands.

When you write a definition using \@defn, \@defun, or one of the other definition commands, please take care to use arguments that indicate the meaning, as with the \texttt{count} argument to the \texttt{forward-word} function. Also, if the name of an argument contains the name of a type, such as \texttt{integer}, take care that the argument actually is of that type.

\textit{Fonts.} As Texinfo is a semantic language, you should nearly never need to specify font styles with explicit font commands in definitions (see Section 6.2.3 [Fonts], page 63). However, you may be need to work around problems for particular output formats and/or constructs. Here are some possibilities:

- Explicitly marking a Lisp keyword like ‘\texttt{&keyword}’ with \@r\@b{&keyword}, producing \&keyword.

  Note such keywords in definition arguments are (at present) rendered in bold in TeX, but this formatting is not done in any other output format.

- ‘\@r\@slanted{argument}’, producing argument, similar to \@var but avoiding uppercase its argument in Info output.

  In this and the previous point, \@r resets the font from that being used in the definition line context (slanted or typewriter) to a roman, upright style.

- You could use \@t (or even ‘\@r\@t{...’), using \@r to reset font styles) to mark literal syntax on a definition line where text would otherwise be output in the slanted roman style. \@code would be inappropriate here as it adds quotation marks in Info output.

Some degree of trial and error may be needed to get the result you want. As ever, how nested font commands combine depends on the output format, so should be avoided where possible.

Hopefully, such usages are kept to a minimum. One possibility is to wrap these in \@macro (see Chapter 16 [Defining New Texinfo Commands], page 143), allowing these usages to be easily changed in the future.
13.8 A Sample Function Definition

Here is a definition from Section “Calling Functions” in *The GNU Emacs Lisp Reference Manual*, using the `@defun` command. The name of the function, `apply`, follows immediately after the `@defun` command and it is followed, on the same line, by the parameter list.

```
apply function &rest arguments
```

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)
  ⇒ list
(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 Texinfo source file, this example should look like this:

```
@defun apply function @r{@b{&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}.
```
@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)
@end example

An interesting example of using @code{apply} is found in the description of @code{mapcar}.
@end defun

In this manual, this function is listed in the Command and Variable Index under @code{apply}.
14 Internationalization

Texinfo has some support for writing in languages other than English, although this area still needs considerable work. (If you are the one helping to translate the fixed strings written to documents, see Section 19.5 [Internationalization of Document Strings], page 188.)

For a list of the various accented and special characters Texinfo supports, see Section 11.4 [Inserting Accents], page 103.

14.1 \@documentlanguage \texttt{ll}\undertilde{cc}: Set the Document Language

The \@documentlanguage command declares the current document locale. Write it on a line by itself, near the beginning of the file.

\@documentlanguage \texttt{ll} \undertilde{cc}

Include a two-letter ISO 639-2 language code (\texttt{ll}) following the command name, optionally followed by an underscore and two-letter ISO 3166 two-letter country code (\texttt{cc}). If you have a multilingual document, the intent is to be able to use this command multiple times, to declare each language change. If the command is not used at all, the default is \texttt{en\_US} for US English.

As with GNU Gettext (see Gettext), if the country code is omitted, the main dialect is assumed where possible. For example, \texttt{de} is equivalent to \texttt{de\_DE} (German as spoken in Germany).

For Info and other online output, this command changes the translation of various document strings such as “see” in cross-references (see Chapter 5 [Cross References], page 43), “Function” in defuns (see Chapter 13 [Definition Commands], page 117), and so on. Some strings, such as “Node:”, “Next:”, “Menu:”, etc., are keywords in Info output, so are not translated there; they are translated in other output formats. In DocBook output \@documentlanguage sets the language for following sections.

For \LaTeX{}, this command causes code to load the ‘\texttt{babel}’ package to be output in the preamble, and \texttt{selectlanguage} to be output.

For \TeX{}, this command causes a file \texttt{txi-locale.tex} to be read (if it exists). If \@documentlanguage argument contains the optional \texttt{\_cc} suffix, this is tried first. For example, with \@documentlanguage \texttt{de\_DE}, \TeX{} first looks for \texttt{txi-de\_DE.tex}, then \texttt{txi-de.tex}.

Such a \texttt{txi-\_} file is intended to redefine the various English words used in \TeX{} output, such as ‘Chapter’, ‘See’, and so on. We are aware that individual words like these cannot always be translated in isolation, and that a very different strategy would be required for ideographic (among other) scripts. Help in improving Texinfo’s language support is welcome.

\@documentlanguage also changes \TeX{}’s current hyphenation patterns, if the \TeX{} program being run has the necessary support included. This will generally not be the case for \texttt{tex} itself, but will usually be the case for up-to-date distributions of the extended \TeX{} programs \texttt{etex} (DVI output) and \texttt{pdftex} (PDF output). \texttt{texi2dvi} will use the extended \TeX{}Xs if they are available (see [Format with \texttt{texi2dvi}], page 154).

Since the lists of language codes and country codes are updated relatively frequently, we don’t attempt to list them here. The valid language codes are on the official home page for ISO 639, \url{http://www.loc.gov/standards/iso639-2/}. The country codes and the official web site for ISO 3166 can be found via \url{https://en.wikipedia.org/wiki/ISO_3166}. 
14.2 @documentencoding enc: Set Input Encoding

In the default case, the input and output document encoding are assumed to be UTF-8, the vast global character encoding, expressed in 8-bit bytes. UTF-8 is compatible with 7-bit ASCII. It is recommended to use UTF-8 encoding for Texinfo manuals.

The @documentencoding command declares the input document encoding, and also affects the encoding of the output. Write it on a line by itself, with a valid encoding specification following, near the beginning of the file if your document encoding is not the default encoding.

@documentencoding enc

UTF-8 should always be the best choice for the encoding. Texinfo still supports additional encodings, mainly for compatibility with older manuals:

**US-ASCII**  Character encoding based on the English alphabet.

**ISO-8859-1**

**ISO-8859-15**

**ISO-8859-2**  These specify the pre-UTF-8 standard encodings for Western European (the first two) and Eastern European languages (the third), respectively. ISO 8859-15 replaces some little-used characters from 8859-1 (e.g., precomposed fractions) with more commonly needed ones, such as the Euro symbol (€).

A full description of the encodings is beyond our scope here; one useful reference is [http://czyborra.com/charsets/iso8859.html](http://czyborra.com/charsets/iso8859.html).

**koi8-r**  This was a commonly used encoding for the Russian language before UTF-8.

**koi8-u**  This was a commonly used encoding for the Ukrainian language before UTF-8.

In Info output, a so-called ‘Local Variables’ section (see Section “File Variables” in *The GNU Emacs Manual*) is output including the output encoding. This allows Info readers to set the encoding appropriately. It looks like this:

```
Local Variables:
  coding: UTF-8
End:
```

By default, for Info and plain text output, `texi2any` outputs accent constructs and special characters (such as 'e) as the actual UTF-8 sequence or 8-bit character in the output encoding where possible. If this is not possible, or if the option `--disable-encoding` is given, an ASCII transliteration is used instead.

In HTML output, a `<meta>` tag is output, in the `<head>` section of the HTML, that specifies the output encoding. Web servers and browsers cooperate to use this information so the correct encoding is used to display the page, if supported by the system. That looks like this:

```
<meta http-equiv="Content-Type" content="text/html;
      charset=utf-8">  
```

In HTML and LATEX output, if the option `--enable-encoding` is given to `texi2any`, accent constructs and special characters, such as 'e or `, are output as the actual UTF-8 sequence or 8-bit character in the output encoding where possible. Otherwise, HTML entities are used for those characters in HTML, and LATEX macros are used in LATEX.
In DocBook output, if the encoding is different from UTF-8, an encoding attribute is added to the XML declaration. If the option --enable-encoding is given to texi2any, accent constructs such as @'e are output as the actual 8-bit or UTF-8 character in the output encoding where possible. Otherwise XML entities are used for those constructs.

In \TeX output, the characters which are supported in the standard Computer Modern fonts are output accordingly. For example, this means using constructed accents rather than precomposed glyphs. Using a missing character generates a warning message, as does specifying an unimplemented encoding.

Although modern \TeX systems support nearly every script in use in the world, this wide-ranging support is not available in texinfo.tex, and it’s not feasible to duplicate or incorporate all that effort.

In \LaTeX output, code loading the `inputenc` package is output based on the encoding. This, by itself, does not ensure that all the characters from the input document can be subsequently output. The fonts used in the default case should cover the specific Texinfo glyphs, but not all the possible encoded characters. You may need to load different fonts in the preamble and use `\DeclareUnicodeCharacter` with a UTF-8 encoding. For example:

```latex
\DeclareUnicodeCharacter{017B}{.Z}
```

Cross-references between Info files in different character encodings with non-ASCII characters in node names fail. We strongly recommend using UTF-8 only as the encoding for manuals with non-ASCII characters in the destinations of cross-references.
15 Conditionally Visible Text

The conditional commands allow you to use different text for different output formats, or for general conditions that you define. For example, you can use them to specify different text for the printed manual and the Info output.

The conditional commands comprise the following categories.

- Commands specific to an output format (Info, \TeX, HTML, \ldots).
- Commands specific to any output format excluding a given one (e.g., not Info, not \TeX, \ldots).
- ‘Raw’ formatter text for any output format, passed straight through with minimal (but not zero) interpretation of @-commands.
- Format-independent variable substitutions, and testing if a variable is set or clear.

15.1 Conditional Commands

Texinfo has an @if format environment for each output format, to allow conditional inclusion of text for a particular output format.

@ifinfo begins segments of text that should be ignored when not producing Info output, in particular in printed output. The segment of text appears only in the Info file and, for historical compatibility, the plain text output.

The environments for the other formats are analogous:

@ifdocbook ... @end ifdocbook
  Text to appear only in the DocBook output.
@ifhtml ... @end ifhtml
  Text to appear only in the HTML output.
@iflatex ... @end iflatex
  Text to appear only in the \LaTeX output.
@ifplaintext ... @end ifplaintext
  Text to appear only in the plain text output.
@iftex ... @end iftex
  Text to appear only in the printed manual.
@ifxml ... @end ifxml
  Text to appear only in the XML output.

The @if... and @end if... commands must appear on lines by themselves in your source file. The newlines following the commands are (more or less) treated as whitespace, so that the conditional text is flowed normally into a surrounding paragraph.

The @if... constructs are intended to conditionailize normal Texinfo source; see Section 15.3 [Raw Formatter Commands], page 135, for using underlying format commands directly.

Here is an example showing all these conditionals:

@iftex
  This text will appear only in the printed manual.
@end iftex
@ifinfo
However, this text will appear only in Info and plain text.
@end ifinfo
@ifhtml
And this text will only appear in HTML.
@end ifhtml
@iflatex
Moreover, this text will only appear in \LaTeX{}. 
@end iflatex
@ifplaintext
Whereas this text will only appear in plain text.
@end ifplaintext
@ifxml
Notwithstanding that this will only appear in XML.
@end ifxml
@ifdocbook
Nevertheless, this will only appear in DocBook.
@end ifdocbook

The preceding example produces the following line:

This text will appear only in the printed manual.

Notice that you only see one of the input lines, depending on which version of the manual
you are reading.

In complex documents, you may want Texinfo to issue an error message in some condi-
tionals that should not ever be processed. The \texttt{\@errormsg{text}} command will do this; it
takes one argument, the text of the error message.

We mention \texttt{\@errormsg{}} here even though it is not strictly related to conditionals, since
in practice it is most likely to be useful in that context. Technically, it can be used anywhere.

See Section 16.6 [External Macro Processors], page 150, for a caveat regarding the line
numbers which \texttt{\@errormsg{} em}its in \TeX{}.

15.2 Conditional Not Commands

You can specify text to be included in any output format other than a given one with the
\texttt{\@ifnot...} environments:

\begin{verbatim}
@ifnotdocbook ... @end ifnotdocbook
@ifnothtml ... @end ifnothtml
@ifnotinfo ... @end ifnotinfo
@ifnotlatex ... @end ifnotlatex
@ifnotplaintext ... @end ifnotplaintext
@ifnottext ... @end ifnottext
@ifnotxml ... @end ifnotxml
\end{verbatim}

The \texttt{\@ifnot...} command and the \texttt{\@end} command must appear on lines by themselves in
your actual source file.

If the output file is being made in the given format, the region is \textit{ignored}. Otherwise, it
is included.
Chapter 15: Conditionally Visible Text

There is one exception (for historical compatibility): @ifnotinfo text is omitted for both Info and plain text output, not just Info. To specify text which appears only in Info and not in plain text, use @ifnotplaintext, like this:

@ifinfo
@ifnotplaintext
This will be in Info, but not plain text.
@end ifnotplaintext
@end ifinfo

The regions delimited by these commands are ordinary Texinfo source as with @iftex, not raw formatter source as with @tex (see Section 15.3 [Raw Formatter Commands], page 135).

15.3 Raw Formatter Commands

The @if... conditionals just described must be used only with normal Texinfo source. For instance, most features of plain \TeX{} will not work within @iftex. The purpose of @if... is to provide conditional processing for Texinfo source, not provide access to underlying formatting features. For that, Texinfo provides so-called raw formatter commands. They should only be used when truly required (most documents do not need them).

The first raw formatter command is @tex. You can enter plain \TeX{} completely, and use \verb|\}| in the \TeX{} commands, by delineating a region with the @tex and @end tex commands. All plain \TeX{} commands and category codes are restored within a @tex region. The sole exception is that the @ character still introduces a command, so that @end tex can be recognized. Texinfo processors will not output material in such a region unless \TeX{} output is being produced.

In complex cases, you may wish to define new \TeX{} macros within @tex. You must use \verb|\gdef| to do this, not \verb|\def|, because @tex regions are processed in a \TeX{} group. If you need to make several definitions, you may wish to set \verb|\globaldefs=1| (its value will be restored to zero as usual when the group ends at @end tex, so it won’t cause problems with the rest of the document).

As an example, here is a displayed equation written in plain \TeX{}:

@tex
$$ \chi^2 = \sum_{i=1}^N \frac{(y_i - (a + b x_i))}{\sigma_i}^2 $$
@end tex

The output of this example will appear only in a printed manual. If you are reading this in a format not generated by \TeX{}, you will not see the equation that appears in the printed manual.

\[ \chi^2 = \sum_{i=1}^N \left( \frac{y_i - (a + bx_i)}{\sigma_i} \right)^2 \]

Analogously, you can use @html ... @end html for a region of raw HTML, @docbook ... @end docbook for a region of raw DocBook, @latex ... @end latex for a region of raw \LaTeX{}, and @xml ... @end xml for a region of raw XML.
The behavior of newlines in raw regions is unspecified.

In all cases, in raw processing, @ retains the same meaning as in the remainder of the document. Thus, the Texinfo processors must recognize and even execute, to some extent, the contents of the raw regions, regardless of the final output format. Therefore, specifying changes that globally affect the document inside a raw region leads to unpredictable and generally undesirable behavior. For example, using the @kbdinputstyle command inside a raw region is undefined. The remedy is simple: don’t do that.

15.4 Inline Conditionals: @inline, @inlineifelse, @inlineraw

Texinfo provides a set of conditional commands with arguments given within braces:

@inlinefmt{format, text}

Process the Texinfo text if format output is being generated.

@inlinefmtifelse{format, then-text, else-text}

Process the Texinfo then-text if format output is being generated; otherwise, process else-text.

@inlineraw{format, text}

Similar, but for raw text (see Section 15.3 [Raw Formatter Commands], page 135).

The supported format names are:

docbook html info latex plaintext tex xml

For example,

@inlinefmt{html, @emph{HTML-only text}}

is nearly equivalent to

@ifhtml
@emph{HTML-only text}
@end ifhtml

except that no whitespace is added, as happens in the latter (environment) case.

In these commands, whitespace is ignored after the comma separating the arguments, as usual, but is not ignored at the end of text.

To insert a literal at sign, left brace, or right brace in one of the arguments, you must use the alphabetic commands @atchar{} (see Section 11.1.1 [Inserting an Atsign], page 97), and @bracechar{} or @bracechar{} (see Section 11.1.2 [Inserting Braces], page 97), or the parsing will become confused.

With @inlinefmtifelse, it is also necessary to use @comma{} to avoid mistaking a ‘,’ in the text for the delimiter. With @inlinefmt and @inlineraw, @comma{} is not required (though it’s fine to use it), since these commands always have exactly two arguments.

For TEx, the processed text cannot contain newline-delimited commands. Text to be ignored (i.e., for non-TEx) can, though.

Two other @inline... conditionals complement the @ifset and @ifclear commands; see the next section.
15.5 Flags: @set, @clear, conditionals, and @value

You can direct the Texinfo formatting commands to format or ignore parts of a Texinfo file with the @set, @clear, @ifset, and @ifclear commands. Here is a summary:

@set flag [value]
Set the variable flag, to the optional value if specified.

@clear flag
Undefine the variable flag, whether or not it was previously defined.

@ifset flag
If flag is set, text through the next @end ifset command is formatted. If flag is clear, text through the following @end ifset command is ignored.

@inlineifset{flag, text}
Brace-delimited version of @ifset.

@ifclear flag
If flag is set, text through the next @end ifclear command is ignored. If flag is clear, text through the following @end ifclear command is formatted.

@inlineifclear{flag, text}
Brace-delimited version of @ifclear.

15.5.1 @set and @value

You use the @set command to specify a value for a flag, which is later expanded by the @value command.

A flag (aka variable) name is an identifier starting with an alphanumeric, ‘-‘, or ‘_’. Subsequent characters, if any, may not be whitespace, ‘@’, braces, angle brackets, or any of ‘~`^+|’; other characters, such as ‘%’, may work. However, it is best to use only letters and numerals in a flag name, not ‘-‘ or ‘_’ or others—they will work in some contexts, but not all, due to limitations in \TeX.

The value is the remainder of the input line, and can contain anything. However, unlike most other commands which take the rest of the line as a value, @set need not appear at the beginning of a line.

Write the @set command like this:

@set foo This is a string.

This sets the value of the flag foo to “This is a string.”.

The Texinfo processors then replace a @value{flag} command with the string to which flag is set. Thus, when foo is set as shown above, the Texinfo processors convert this:

@value{foo}

to this:

This is a string.

You can write a @value command within a paragraph; but you must write a @set command on a line of its own.

If you write the @set command like this:

@set foo
without specifying a string, the value of foo is the empty string.

If you clear a previously set flag with @clear flag, a subsequent @value{flag} command will report an error.

For example, if you set foo as follows:

@set howmuch very, very, very

then the processors transform

It is a @value{howmuch} wet day.

into

It is a very, very, very wet day.

If you write

@clear howmuch

then the processors transform

It is a @value{howmuch} wet day.

into

It is a {No value for "howmuch"} wet day.

@value cannot be reliably used as the argument to an accent command (see Section 11.4 [Inserting Accents], page 103). For example, this fails:

@set myletter a
@'@value{myletter}

15.5.2 @ifset and @ifclear

When a flag is set, the Texinfo formatting commands format text between subsequent pairs of @ifset flag and @end ifset commands. When the flag is cleared, the Texinfo formatting commands do not format the text. @ifclear operates analogously.

Write the conditionally formatted text between @ifset flag and @end ifset commands, like this:

@ifset flag
  conditional-text
@end ifset

For example, you can create one document that has two variants, such as a manual for a ‘large’ and ‘small’ model:

You can use this machine to dig up shrubs without hurting them.

@set large

@ifset large
  It can also dig up fully grown trees.
@end ifset

Remember to replant promptly ...

In the example, the formatting commands will format the text between @ifset large and @end ifset because the large flag is set.
When flag is cleared, the Texinfo formatting commands do not format the text between @ifset flag and @end ifset; that text is ignored and does not appear in the output.

For example, if you clear the flag of the preceding example by writing an @clear large command after the @set large command (but before the conditional text), then the Texinfo formatting commands ignore the text between the @ifset large and @end ifset commands. In the formatted output, that text does not appear; you see only the lines that say, “You can use this machine to dig up shrubs without hurting them. Remember to replant promptly...”.

If a flag is cleared with a @clear flag command, then the formatting commands format text between subsequent pairs of @ifclear and @end ifclear commands. But if the flag is set with @set flag, then the formatting commands do not format text between an @ifclear and an @end ifclear command; rather, they ignore that text. An @ifclear command looks like this:

@ifclear flag

15.5.3 @inlineifset and @inlineifclear

@inlineifset and @inlineifclear provide brace-delimited alternatives to the @ifset and @ifclear forms, similar to the other @inline... Commands (see Section 15.4 [Inline Conditionals], page 136). The same caveats about argument parsing given there apply here too.

@inlineifset{var, text}
   Process the Texinfo text if the flag var is defined.

@inlineifclear{var, text}
   Process the Texinfo text if the flag var is not defined.

Except for the syntax, their general behavior and purposes is the same as with @ifset and @ifclear, described in the previous section.

15.5.4 @value Example

You can use the @value command to minimize the number of places you need to change when you record an update to a manual. See Section C.1 [GNU Sample Texts], page 243, for the full text of an example of using this to work with Automake distributions.

This example is adapted from The GNU Make Manual.

1. Set the flags:

   @set EDITION 0.35 Beta
   @set VERSION 3.63 Beta
   @set UPDATED 14 August 1992
   @set UPDATE-MONTH August 1992

2. Write text for the @copying section (see Section 2.7.1 [@copying], page 17):
3. Write text for the title page, for people reading the printed manual:

```latex
@titlepage
@title GNU Make
@subtitle A Program for Directing Recompilation
@subtitle Edition @value{EDITION}, ...
@subtitle @value{UPDATE-MONTH}
@page
@end titlepage
```

(On a printed cover, a date listing the month and the year looks less fussy than a date listing the day as well as the month and year.)

4. Write text for the Top node, for people reading the Info file:

```latex
@ifnottex
@node Top
@top Make

This is Edition @value{EDITION},
last updated @value{UPDATED},
of @cite{The GNU Make Manual},
for @code{make}, version @value{VERSION}.
@end ifnottex
```

After you format the manual, the @value constructs have been expanded, so the output contains text like this:

This is Edition 0.35 Beta, last updated 14 August 1992,

When you update the manual, you change only the values of the flags; you do not need to edit the three sections.

15.6 Testing for Texinfo Commands: @ifcommanddefined, @ifcommandnotdefined

Occasionally, you may want to arrange for your manual to test if a given Texinfo command is available and (presumably) do some sort of fallback formatting if not. There are conditionals @ifcommanddefined and @ifcommandnotdefined to do this. For example:

@ifcommanddefined node
Chapter 15: Conditionally Visible Text

Good, @samp{@@node} is defined.
@end ifcommanddefined

will output the expected ‘Good, ‘@node’ is defined.’.

This conditional will also consider any new commands defined by the document via @macro, @alias, @definfoenclose, and @def(code)index (see Chapter 16 [Defining New Texinfo Commands], page 143) to be true. Caveat: the T\TeX implementation reports internal T\TeX commands, in addition to all the Texinfo commands, as being “defined”; the texi2any implementation is reliable in this regard, however.

You can check the NEWS file in the Texinfo source distribution and linked from the Texinfo home page (http://www.gnu.org/software/texinfo) to see when a particular command was added.

These command-checking conditionals themselves were added in Texinfo 5.0, released in 2013—decades after Texinfo’s inception. In order to test if they themselves are available, the predefined flag txicommandconditionals can be tested, like this:

    @ifset txicommandconditionals
    @ifcommandnotdefined foobarnode
    (Good, @samp{@@foobarnode} is not defined.)
    @end ifcommandnotdefined
    @end ifset

Since flags (see the previous section) were added early in the existence of Texinfo, there is no problem with assuming they are available.

We recommend avoiding these tests whenever possible—which is usually the case. For many software packages, it is reasonable for all developers to have a given version of Texinfo (or newer) installed, and thus no reason to worry about older versions. (It is straightforward for anyone to download and install the Texinfo source; it does not have any problematic dependencies.)

The issue of Texinfo versions does not generally arise for end users. With properly distributed packages, users need not process the Texinfo manual simply to build and install the package; they can use preformatted Info (or other) output files. This is desirable in general, to avoid unnecessary dependencies between packages (see Section “Releases” in GNU Coding Standards).

15.7 Conditional Nesting

Texinfo requires that for a failing conditional, the ignored text must be properly nested with respect to that failing conditional. Here’s an example:

    @ifset somevar
    @ifset anothervar
    Both somevar and anothervar are set.
    @end ifset
    @ifclear anothervar
    Somevar is set, anothervar is not.
    @end ifclear
    @end ifset

If ‘somevar’ is not set, the whole block is skipped.
To allow the processors to reliably determine which commands to consider for nesting purposes, all conditional commands must be on lines by themselves, with no text (even spaces) before or after.
16 Defining New Texinfo Commands

Texinfo provides several ways to define new commands (in all cases, it’s not recommended to try redefining existing commands):

- A Texinfo macro allows you to define a new Texinfo command as any sequence of text and/or existing commands (including other macros). The macro can have any number of parameters—text you supply each time you use the macro.
  Incidentally, these macros have nothing to do with the @defmac command, which is for documenting macros in the subject area of the manual (see Section 13.1 [Def Cmd Template], page 117).
- `@alias` defines a new name for an existing command.
- `@definfoenclose` allows you to define new commands with customized output for some non-printed output formats. This command is deprecated.

Most generally of all (not just for defining new commands), it is possible to invoke any external macro processor and have Texinfo recognize so-called #line directives for error reporting.

If you want to do simple text substitution, @set and @value is the simplest approach (see Section 15.5 [@set @clear @value], page 137).

16.1 Defining Macros

You use the Texinfo @macro command to define a macro, like this:

```
@macro macroname{param1, param2, ...}
  text ... \param1\ ... 
@end macro
```

The parameters param1, param2, ... correspond to arguments supplied when the macro is subsequently used in the document (described in the next section).

In principle, macroname should consist of alphanumerics, and (except as the first character) ‘-’. The ‘-’ character is excluded so that macros can be called inside @math without a following space (see Section 11.7 [Inserting Math], page 105). However, for a macro to work consistently with \TeX, macroname must consist entirely of letters: no digits, hyphens, or other special characters. So, we recommend using only letters. Texinfo commands should not be redefined as macros.

If a macro needs no parameters, you can define it either with an empty list (`@macro foo {}`) or with no braces at all (`@macro foo`).

The definition or body of the macro can contain most Texinfo commands, including macro invocations. However, a macro definition that defines another macro does not work in \TeX due to limitations in the design of @macro.

In the macro body, instances of a parameter name surrounded by backslashes, as in `\param1\` in the example above, are replaced by the corresponding argument from the macro invocation. You can use parameter names any number of times in the body, including zero.

To get a single `\` in the macro expansion, use `\\`. Any other use of `\` in the body yields a warning.
The newline characters after the @macro line and before the @end macro line are ignored, that is, not included in the macro body. All other whitespace is treated according to the usual Texinfo rules.

To allow a macro to be used recursively, that is, in an argument to a call to itself, you must define it with ‘@rmacro’, like this:

@rmacro rmac {arg}
a\arg\b
@end rmacro

...  
rmac{1@rmac{text}2}

This produces the output ‘a1atextb2b’. With ‘@macro’ instead of ‘@rmacro’, an error message is given.

You can undefine a macro foo with @unmacro foo. It is not an error to undefine a macro that is already undefined. For example:

@unmacro foo

16.2 Invoking Macros

After a macro is defined (see the previous section), you can invoke (use) it in your document like this:

@macroname {arg1, arg2, ...}

and the result will be more or less as if you typed the body of macroname at that spot. For example:

@macro foo {p, q}
Together: p & q.
@end macro
@foo{a, b}

produces:

Together: a & b.

Thus, the arguments and parameters are separated by commas and delimited by braces; any whitespace after (but not before) a comma is ignored. The braces are required in the invocation even when the macro takes no arguments, consistent with other Texinfo commands. For example:

@macro argless {}
No arguments here.
@end macro
@argless{}

produces:

No arguments here.

Passing macro arguments containing commas requires care, since commas also separate the arguments. To include a comma character in an argument, the most reliable method is to use the @comma{} command. For texi2any, you can also prepend a backslash character, as in ‘\,’, but this does not work with TpX.
It’s not always necessary to worry about commas. To facilitate use of macros, two rules for automatic quoting are implemented:

1. If a macro takes only one argument, all commas in its invocation are quoted by default. For example:

   @macro TRYME{\text}
   @strong{\text}: TRYME:
   @end macro

   @TRYME{A nice feature, though it can be dangerous.}

   will produce the following output

   **TRYME: A nice feature, though it can be dangerous.**

   And indeed, it can. Namely, there is no control on the number of arguments passed to one-argument macros, so be careful when you invoke them.

2. If a macro invocation includes another command (including a recursive invocation of itself), any commas in the nested command invocation(s) are quoted by default. For example, in

   @say{@strong{Yes, I do}, person one}

   the comma after ‘Yes’ is implicitly quoted. Here’s another example, with a recursive macro:

   @rmacro cat{a,b}
   \\a\\b\
   @end rmacro

   @cat{@cat{foo, bar}, baz}

   will produce the string ‘foobaz’.

3. Otherwise, a comma should be explicitly quoted, as above, for it to be treated as a part of an argument.

   The backslash itself can be quoted in macro arguments with another backslash. For example:

   @macname {\\bleh}

   will pass the argument ‘\bleh’ to *macname*.

   texi2any also recognizes ‘\{’ and ‘\}’ sequences for curly braces, but these are not recognized by the implementation in *T\TeX*. There should, however, rarely be a need for these, as they are only needed when a macro argument contains unbalanced braces.

   If a macro is defined to take exactly one argument, it can be invoked without any braces, taking all of the line after the macro name as the argument. For example:

   @macro bar {p}
   Twice: \p\ & \p\.
   @end macro
   @bar aah

   produces:

   Twice: aah & aah.
In these arguments, there is no escaping of special characters, so each ‘\’ stands for itself.

If a macro is defined to take more than one argument, but is called with only one (in braces), the remaining arguments are set to the empty string, and no error is given. For example:

```texinfo
@macro addtwo {p, q}
Both: \p\q
@end macro
```

produces simply:

```
Both: a.
```

### 16.3 Macro Details and Caveats

By design, macro expansion should not happen in the following contexts:

- `@macro` and `@unmacro` lines;
- `@if...` lines, including `@ifset` and similar;
- `@set`, `@clear`, `@value`;
- `@clickstyle` lines;
- `@end` lines when there is no macro expansion in the block command.

Unfortunately, TEX may do some expansion in these situations, possibly yielding errors.

Also, quite a few macro-related constructs cause problems with TEX; some of the caveats are listed below. Thus, if you get macro-related errors when producing the printed version of a manual, you might try expanding the macros with `texi2any` by invoking `texi2dvi` with the ‘-E’ option (see [Format with `texi2dvi`], page 154). Or, more reliably, eschew Texinfo macros altogether and use a language designed for macro processing, such as M4 (see Section 16.6 [External Macro Processors], page 150).

- As mentioned earlier, macro names must consist entirely of letters.
- It is not advisable to redefine any TEX primitive, plain, or Texinfo command name as a macro. Unfortunately, this is a large and open-ended set of names, and the possible resulting errors are unpredictable.
- Arguments to macros taking more than one argument cannot cross lines.
- Macros containing a command which must be on a line by itself, such as a conditional, cannot be invoked in the middle of a line. Similarly, macros containing line-oriented commands or text, such as `@example` environments, may behave unpredictably in TEX.
- If you have problems using conditionals within a macro, an alternative is to use separate macro definitions inside conditional blocks. For example, instead of

```texinfo
@macro Mac
@if\text for TeX output\end if\text for not TeX output
@end\text for TeX output
```

```texinfo
@macro Mac
@if\text for TeX output\endif\text for not TeX output
@end\text for TeX output
```
you can do the following instead:

@iftex
@macro Mac
text for TeX output
@end macro
@end iftex

@ifnottex
@macro Mac
text for not TeX output
@end macro
@end ifnottex

- Texinfo commands in the expansion of a macro in the text of an index entry may end up being typeset as literal text (including an “@” sign), instead of being interpreted with their intended meaning.
- White space is ignored at the beginnings of lines.
- Macros can’t be reliably used in the argument to accent commands (see Section 11.4 [Inserting Accents], page 103).
- The backslash escape for commas in macro arguments does not work; @comma{} must be used.
- Ending a macro body with ‘@c’ may cause text following the macro invocation to be ignored as a comment in texi2any. This is not the case when processing with T\TeX. This was often done to “comment out” an unwanted newline at the end of a macro body, but this is not necessary any more, as the final newline before ‘@end macro’ is not included in the macro body anyway.
- In general, you can’t arbitrarily substitute a macro (or @value) call for Texinfo command arguments, even when the text is the same. Texinfo is not M4 (or even plain T\TeX). It might work with some commands, it fails with others. Best not to do it at all. For instance, this fails:

@macro offmacro
off
@end macro
@headings @offmacro

This looks equivalent to @headings off, but for T\TeXnical reasons, it fails with a mysterious error message (namely, ‘Paragraph ended before @headings was complete’).

- Macros cannot define macros in the natural way. To do this, you must use conditionals and raw T\TeX. For example:

@ifnottex
@macro ctor {name, arg}
@macro \name\ something involving \arg\ somehow
@end macro
@end macro

@iftex
@macro Mac
text for TeX output
@end macro
@end iftex
The following limitations are by design:

- If you want to pass an argument with the Texinfo command \[, (to produce a cedilla, see Section 11.4 [Inserting Accents], page 103), you have to use \emph{value} or another workaround. Otherwise, the comma may be taken as separating the arguments. With \texttt{texi2any}, the comma can be escaped by a backslash. With \texttt{TEX} another workaround need to be used, therefore we recommend using such a workaround.

For example,

\begin{verbatim}
@macro mactwo{argfirst, argsecond}
  \argfirst\+\argsecond.
@end macro
@set fc Fran\text{"}cois
@mactwo{@value{fc},}
\end{verbatim}
produces:

François+

- \texttt{verbatim} and macros do not mix; for instance, you can’t start a verbatim block inside a macro and end it outside (see Section 7.5 [verbatim], page 68). Starting any environment inside a macro and ending it outside may or may not work, for that matter.

- Macros that completely define macros are ok, but it’s not possible to have incompletely nested macro definitions. That is, \texttt{macro} and \texttt{end macro} (likewise for \texttt{rmacro}) must be correctly paired. For example, you cannot start a macro definition within a macro, and then end that nested definition outside the macro.

In the \texttt{makeinfo} implementation before Texinfo 5.0, ends of lines from expansion of a \texttt{macro} definition did not end an \texttt{-command line-delimited argument} (\texttt{chapter}, \texttt{center}, etc.). This is no longer the case. For example:

\begin{verbatim}
@macro twolines{}
  aaa
  bbb
@end macro
@center twolines{}
\end{verbatim}

In the current \texttt{texi2any}, this is equivalent to:

\begin{verbatim}
@center aaa
  bbb
\end{verbatim}

with just ‘aaa’ as the argument to \texttt{center}. In the earlier implementation, it would have been parsed as this:

\begin{verbatim}
@center aaa bbb
\end{verbatim}
16.4 ‘@alias new=existing’

The ‘@alias’ command defines a new command to be just like an existing one. This is useful for defining additional markup names, thus preserving additional semantic information in the input even though the output result may be the same.

Write the ‘@alias’ command on a line by itself, followed by the new command name, an equals sign, and the existing command name. Whitespace around the equals sign is optional and ignored if present. Thus:

    @alias new = existing

For example, if your document contains citations for both books and some other media (movies, for example), you might like to define a macro @moviecite{} that does the same thing as an ordinary @cite{} but conveys the extra semantic information as well. You’d do this as follows:

    @alias moviecite = cite

Macros do not always have the same effect as aliases, due to vagaries of argument parsing. Also, aliases are much simpler to define than macros. So the command is not redundant.

Unfortunately, it’s not possible to alias Texinfo environments; for example, @alias lang=example is an error.

Aliases must not be recursive, directly or indirectly.

It is not advisable to redefine any TeX primitive, plain TeX, or Texinfo command name as an alias. Unfortunately this is a very large set of names, and the possible resulting errors from TeX are unpredictable.

Alias identifiers should be the same as for macro names, that is alphanumerics and (except as the first character) ‘-‘. However, with TeX, letters only should be used. So, we recommend using only letters.

16.5 @definfoenclose: Customized Highlighting

An @definfoenclose command may be used to define a highlighting command for online output formats. A command defined using @definfoenclose marks text by enclosing it in strings that precede and follow the text.

In practice, there is little use for this command, and we do not recommend you use it. Support for @definfoenclose may be removed in future releases of Texinfo.

Write a @definfoenclose command at the beginning of a line followed by three comma-separated arguments. The first argument to @definfoenclose is the @-command name (without the @); the second argument is the start delimiter string; and the third argument is the end delimiter string. The latter two arguments enclose the highlighted text in the output.

A delimiter string may contain spaces. Neither the start nor end delimiter is required. If you do not want a start delimiter but do want an end delimiter, you must follow the command name with two commas in a row; otherwise, the end delimiter string you intended will naturally be (mis)interpreted as the start delimiter string.

An enclosure command defined this way takes one argument in braces, since it is intended for new markup commands (see Chapter 6 [Marking Text], page 54).
For example, you can write:

```
@definfoenclose phoo,//,\\
```

near the beginning of a Texinfo file to define \texttt{@phoo} as an Info and HTML formatting command that inserts ‘//’ before and ‘\\’ after the argument to \texttt{@phoo}. You can then write \texttt{@phoo{bar}} wherever you want ‘//bar\\’ highlighted in Info and HTML.

For \LaTeX{} formatting, you could write

```
@iftex
@alias phoo = i
@end iftex
```

to define \texttt{@phoo} as a command that causes \LaTeX{} to typeset the argument to \texttt{@phoo} in italics.

Each definition applies to its own formatter: one for \LaTeX{}, the other for online formats. The \LaTeX{} definitions need to be in ‘@iftex’. \texttt{@definfoenclose} command need not be within ‘@ifinfo’ unless you want to use different definitions for different online output formats. \texttt{@definfoenclose} defined commands have no effect in DocBook and \LaTeX{} output, their argument is output as-is. An \texttt{@alias} could also be used for these formats.

\texttt{@definfoenclose} definitions must not be recursive, directly or indirectly.

### 16.6 External Macro Processors: Line Directives

Texinfo macros (and its other text substitution facilities) work fine in straightforward cases. If your document needs unusually complex processing, however, their fragility and limitations can be a problem. In this case, you may want to use a different macro processor altogether, such as M4 (see Section “Preprocessor features” in GNU M4) or CPP (see The C Preprocessor).

With one exception, Texinfo does not need to know whether its input is “original” source or preprocessed from some other source file. Therefore, you can arrange your build system to invoke whatever programs you like to handle macro expansion or other preprocessing needs. Texinfo does not offer built-in support for any particular preprocessor, since no one program seemed likely to suffice for the requirements of all documents.

The one exception is line numbers in error messages. In that case, the line number should refer to the original source file, whatever it may be. There’s a well-known mechanism for this: the so-called ‘#line’ directive. Texinfo supports this.

#### 16.6.1 ‘#line’ Directive

An input line such as this:

```
#line 100 "foo.ptexi"
```

indicates that the next line was line 100 of the file \texttt{foo.ptexi}, and so that’s what an error message should refer to. Both M4 (see Section “Preprocessor features” in GNU M4) and CPP (see Section “Line Control” in The C Preprocessor, and Section “Preprocessor Output” in The C Preprocessor) can generate such lines.

The \texttt{texi2any} program recognizes these lines by default, except within \texttt{@verbatim} blocks (see Section 7.5 [\texttt{@verbatim}], page 68). Their recognition can be turned off completely with \texttt{CPP\_LINE\_DIRECTIVES} (see Section 19.4.7 [Other Customization Variables], page 182), though there is normally no reason to do so.

For those few programs (M4, CPP, Texinfo) which need to document ‘#line’ directives and therefore have examples which would otherwise match the pattern, the command
@hashchar{} can be used (see Section 11.1.5 [Inserting a Hashsign], page 98). The example line above looks like this in the source for this manual:

```
@hashchar{}line 100 "foo.ptexi"
```

The @hashchar{} command was added to Texinfo in 2013. If you don’t want to rely on it, you can also use @set and @value to insert the literal ‘#’:

```
@set hash #
@value{hash}line 1 "example.c"
```

Or, if suitable, a @verbatim environment can be used instead of @example. As mentioned above, #line-recognition is disabled inside verbatim blocks.

### 16.6.2 ‘#line’ and TEX

As mentioned, texi2any recognizes the ‘#line’ directives described in the previous section. However, texinfo.tex does not and cannot. Therefore, such a line will be incorrectly typeset verbatim if \TeX\ sees it. The solution is to use texi2any’s macro expansion options before running \TeX\. There are three approaches:

- If you run texi2dvi or its variants (see [Format with texi2dvi], page 154), you can pass -E and texi2dvi will run texi2any first to expand macros and eliminate ‘#line’.
- If you run texi2any, you can specify --no-ifinfo --iftex -E somefile.out, and then give somefile.out to texi2dvi in a separate command.
- Or you can run texi2any --dvi --Xopt -E. (Or --pdf instead of --dvi.) texi2any will then call texi2dvi -E.

One last caveat regarding use with \TeX: since the #line directives are not recognized, the line numbers emitted by the @errormsg{} command (see Section 15.1 [Conditional Commands], page 133), or by \TeX\ itself, are the (incorrect) line numbers from the derived file which \TeX\ is reading, rather than the preprocessor-specified line numbers.

### 16.6.3 ‘#line’ Syntax Details

Syntax details for the ‘#line’ directive: the ‘#’ character can be preceded or followed by whitespace, the word ‘line’ is optional, and the file name can be followed by a whitespace-separated list of integers (these are so-called “flags” output by CPP in some cases). For those who like to know the gory details, the actual (Perl) regular expression which is matched is this:

```
/\s*#\s*(line)? \(\d+\)(\([^{*]}+\)\)(\<\d+\)*?\s*$\)/
```

As far as we’ve been able to tell, the trailing integer flags only occur in conjunction with a file name, so that is reflected in the regular expression.

As an example, the following is a syntactically valid ‘#line’ directive, meaning line 1 of /usr/include/stdio.h:

```
# 1 "/usr/include/stdio.h" 2 3 4
```

Unfortunately, the quoted file name (“...”) has to be optional, because M4 (especially) can often generate ‘#line’ directives within a single file. Since the ‘line’ is also optional, the result is that lines might match which you wouldn’t expect, e.g.,

```
# 1
```

The possible solutions are described above (see Section 16.6.1 [‘#line’ Directive], page 15).
17 Include Files

When a Texinfo processor sees an @include command in a Texinfo file, it processes the contents of the file named by the @include and incorporates them into the output files being created. Include files thus let you keep a single large document as a collection of conveniently small parts.

17.1 How to Use Include Files

To include another file within a Texinfo 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 buffers.texi

@value{var} references are expanded on the @include line. Other than that, the only @-commands allowed are @&{, @{ and associated @-commands such as @atchar{).

An included file should simply be a segment of text that you expect to be included as is into the overall or outer Texinfo file; it should not contain the standard beginning and end parts of a Texinfo file. In particular, you should not start an included file with a line saying \texttt{\input texinfo}; if you do, that text is inserted into the output file literally. Likewise, you should not end an included file with a @bye command; nothing after @bye is formatted.

In the long-ago past, you were required to write an @setfilename line at the beginning of an included file, but no longer. Now, it does not matter whether you write such a line. If an @setfilename line exists in an included file, it is ignored.

GNU Emacs Texinfo mode provides texinfo-multiple-files-update to update node pointers and master menu with multiple include files. See Section D.5.3 [Update Multiple Files], page 255.

17.2 Sample File with @include

Here is an example of an outer Texinfo file with @include files within it:

\input texinfo @c -*-texinfo-*- 

@settitle Include Example

@node Top 
@top Include Example 
@include foo.texi 
@include bar.texi 
@include concept-index.texi 
@bye 

An included file, such as foo.texi, might look like this:

@node First 
@chapter First Chapter 

Contents of first chapter ...
The full contents of `concept-index.texi` might be as simple as this:

@node Concept Index
@unnumbered Concept Index

@printindex cp

The outer Texinfo source file for *The GNU Emacs Lisp Reference Manual* is named `elisp.texi`. This outer file has already contained a master menu with 417 entries and a list of 41 @include files.

### 17.3 @verbatiminclude file: Include a File Verbatim

You can include the exact contents of a file in the document with the @verbatiminclude command:

@verbatiminclude filename

The contents of `filename` is printed in a verbatim environment (see Section 7.5 [@verbatim], page 68). Generally, the file is printed exactly as it is, with all special characters and white space retained. No indentation is added; if you want indentation, enclose the @verbatiminclude within @example (see Section 7.4 [@example], page 67).

@value{var} references are expanded on the @verbatiminclude line. This makes it possible to include files in other directories within a distribution, for instance:

@verbatiminclude @value{top_srcdir}/NEWS

(You still have to get `top_srcdir` defined in the first place.)

Other than that, the only @-commands allowed are @®, @{, @} and associated @-commands such as @atchar{}.

For a method on printing the file contents in a smaller font size, see the end of the section on @verbatim.
18 Formatting and Printing with T\TeX

Running the `texi2dvi` or `texi2pdf` command is the simplest way to create printable output. These commands are installed as part of the Texinfo package.

In more detail, three major shell commands are used to print formatted output from a Texinfo manual: one converts the Texinfo source into something printable, a second sorts indices, and a third actually prints 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 GNU Emacs (or some other computing environment).

If you are using GNU Emacs, you can use commands provided by Texinfo mode instead of shell commands. In addition to the three commands to format a file, sort the indices, and print the result, Texinfo mode offers key bindings for commands to recenter the output buffer, show the print queue, and delete a job from the print queue.

In the United States, documents are most often printed on 8.5 inch by 11 inch pages (216 mm by 280 mm); this is the default size. But you can also print for 7 inch by 9.25 inch pages (178 mm by 235 mm, the `@smallbook` size; or on A4 or A5 size paper (`@afourpaper`, `@afivepaper`). See Section E.6 [`@smallbook`], page 275, and Section E.7 [A4 Paper], page 275.

Details are in the following sections.

18.1 Use T\TeX

The typesetting program called T\TeX is used to format a Texinfo document for printable output. T\TeX is a very powerful typesetting program and, when used correctly, does an exceptionally good job. It is not included in the Texinfo package, being a vast suite of software in itself.

T\TeX is a document formatter that is used by the FSF for its documentation. It is the easiest way to get printed output (e.g., PDF and PostScript) for Texinfo manuals. TeX is freely redistributable, and you can get it over the Internet or on physical media. See [http://tug.org/texlive](http://tug.org/texlive).

18.2 Format with `texi2dvi` or `texi2pdf`

The `texi2dvi` program takes care of all the steps for producing a T\TeX DVI file from a Texinfo document. Similarly, `texi2pdf` produces a PDF file\(^1\).

To run `texi2dvi` or `texi2pdf` on an input file `foo.texi`, do this (where `prompt$` is your shell prompt):

```
prompt$ texi2dvi foo.texi
prompt$ texi2pdf foo.texi
```

As shown in this example, the file names given to `texi2dvi` and `texi2pdf` must include any extension, such as `.
texi`.

For a list of all the options, run `texi2dvi --help`. Some of the options are discussed below.

\(^1\) PDF stands for ‘Portable Document Format’. It was invented by Adobe Systems for document interchange, based on their PostScript language.
With the --pdf option, texi2dvi produces PDF output instead of DVI, by running pdftex instead of tex. Alternatively, the command texi2pdf is an abbreviation for running `texi2dvi --pdf`. The command pdftexi2dvi is also provided as a convenience for AUC-T\TeX{} (see \texttt{AUC-T\TeX}), as it prefers to merely prepend `pdf` to DVI producing tools to have PDF producing tools.

With the --dvipdf option, texi2dvi produces PDF output by running T\TeX{} and then a DVI-to-PDF program: if the DVIPDF environment variable is set, that value is used, else the first program extant among dvipdfmx, dvipdfm, dvipdf, dvi2pdf, dvitopdf. This method generally supports CJK typesetting better than pdftex.

With the --ps option, texi2dvi produces PostScript instead of DVI, by running \texttt{tex} and then \texttt{dvips} (see \texttt{Dvips}). (Or the value of the DVIPS environment variable, if set.)

texi2dvi can also be used to process \LaTeX{} files. Normally texi2dvi is able to guess the input file language by its contents and file name extension; however, if it guesses wrong you can explicitly specify the input language using --language=lang command line option, where lang is either `latex` or `texinfo`.

One useful option to texi2dvi is --command=cmd'. This inserts cmd on a line by itself at the start of the file in a temporary copy of the input file, before running T\TeX{}. With this, you can specify different printing formats, such as \texttt{@smallbook} (see Section E.6 [\texttt{@smallbook}], page 275), \texttt{@afourpaper} (see Section E.7 [\texttt{A4 Paper}], page 275), or \texttt{@pagesizes} (see Section E.8 [\texttt{@pagesizes}], page 276), without actually changing the document source. (You can also do this on a site-wide basis with \texttt{texinfo.cnf}; see Section 18.7 [Preparing for T\TeX{}], page 161).

The option -E (equivalently, -e and --expand) does Texinfo macro expansion using texi2any instead of the T\TeX{} implementation (see Section 16.3 [Macro Details], page 146). Each implementation has its own limitations and advantages. If this option is used, no line in the source file may begin with the string \texttt{@c _texi2dvi} or the string \texttt{@c (_texi2dvi)}.

texi2dvi takes the --build=mode option to specify where the T\TeX{} compilation takes place, and, as a consequence, how auxiliary files are treated. The build mode can also be set using the environment variable \texttt{TEXI2DVI_BUILD_MODE}. The valid values for mode are:

\begin{itemize}
  \item \texttt{local} Compile in the current directory, leaving all the auxiliary files around. This is the traditional TeX use.
  \item \texttt{tidy} Compile in a local *.t2d directory, where the auxiliary files are left. Output files are copied back to the original file.
\end{itemize}

Using the 'tidy' mode brings several advantages:

\begin{itemize}
  \item the current directory is not cluttered with plethora of temporary files.
  \item clutter can be even further reduced using --build-dir=dir: all the *.t2d directories are stored there.
  \item clutter can be reduced to zero using, e.g., --build-dir=/tmp/$USER.t2d or --build-dir=$HOME/.t2d.
  \item the output file is updated after every successful T\TeX{} run, for sake of concurrent visualization of the output. In a 'local' build the viewer stops during the whole T\TeX{} run.
  \item if the compilation fails, the previous state of the output file is preserved.
\end{itemize}
- PDF and DVI compilation are kept in separate subdirectories preventing any possibility of auxiliary file incompatibility.

On the other hand, because ‘tidy’ compilation takes place in another directory, occasionally \TeX won’t be able to find some files (e.g., when using \texttt{\textbackslash graphicspath}): in that case, use -I to specify the additional directories to consider.

‘clean’ Same as ‘tidy’, but remove the auxiliary directory afterwards. Every compilation therefore requires the full cycle.

texi2dvi will use etex if it is available, because it runs faster in some cases, and provides additional tracing information when debugging texinfo.tex. Nevertheless, this extended version of \TeX is not required, and the DVI output is identical.

texi2dvi attempts to detect auxiliary files output by \TeX, either by using the \texttt{-recorder} option, or by scanning for ‘\textbackslash openout’ in the log file that a run of \TeX produces. You may control how texi2dvi does this with the TEXI2DVI\_USE\_RECORDER environment variable. Valid values are:

‘yes’ use the \texttt{-recorder} option, no checks.
‘no’ scan for ‘\textbackslash openout’ in the log file, no checks.
‘yesmaybe’ check whether \texttt{-recorder} option is supported, and if yes use it, otherwise check for tracing ‘\textbackslash openout’ in the log file is supported, and if yes use it, else it is an error.

‘nomaybe’ same as ‘yesmaybe’, except that the ‘\textbackslash openout’ trace in log file is checked first.

The default is ‘nomaybe’. This environment variable is provided for troubleshooting purposes, and may change or disappear in the future.

18.3 Format with tex/texindex

You can do the basic formatting of a Texinfo file with the shell command tex followed by the name of the Texinfo file. For example:

tex foo.texi
\TeX will produce a DVI file as well as several auxiliary files containing information for indices, cross-references, etc. The DVI file (for Devise Independent file) can be printed on virtually any device, perhaps after a further conversion (see the previous section).

The tex formatting command itself does not sort the indices; it writes an output file of unsorted index data. To generate a printed index after running the tex command, you first need a sorted index to work from. The texindex command sorts indices. (texi2dvi, described in the previous section, runs tex and texindex as necessary.)

tex outputs unsorted index files under names following a standard convention: the name of your main input file with any ‘.texi’ or similar extension replaced by the two letter index name. For example, the raw index output files for the input file foo.texi would be, by default, foo.cp, foo.vr, foo.fn, foo.tp, foo.pg and foo.ky. Those are exactly the arguments to give to texindex.
Instead of specifying all the unsorted index file names explicitly, it’s typical to use ‘??’ as shell wildcards and give the command in this form:

```
texindex foo.??
```

This command will run `texindex` on all the unsorted index files, including any two letter indices that you have defined yourself using `@defindex` or `@defcodeindex`. You can safely run `texindex foo.??` even if there are files with two letter extensions that are not index files, such as `foo.el`. The `texindex` command reports but otherwise ignores such files.

For each file specified, `texindex` generates a sorted index file whose name is made by appending ‘s’ to the input file name; for example, `foo.cps` is made from `foo.cp`. The `@printindex` command looks for a file with that name (see Section 10.5 [Printing Indices & Menus], page 93). \TeX does not read the raw index output file, and `texindex` does not alter it.

After you have sorted the indices, you need to rerun `tex` on the Texinfo file. This regenerates the output file, this time with up-to-date index entries.

Finally, you may need to run `tex` one more time, to get the page numbers in the cross-references correct.

To summarize, this is a five-step process. (Alternatively, it’s a one-step process: run `texi2dvi`; see the previous section.)

1. Run `tex` on your Texinfo file. This generates a DVI file (with undefined cross-references and no indices), and the raw index files (with two letter extensions).
2. Run `texindex` on the raw index files. This creates the corresponding sorted index files (with three letter extensions).
3. Run `tex` again on your Texinfo file. This regenerates the DVI file, this time with indices and defined cross-references, but with page numbers for the cross-references from the previous run, generally incorrect.
4. Sort the indices again, with `texindex`.
5. Run `tex` one last time. This time the correct page numbers are written for the cross-references.

To generate PDF, you can run the `pdftex` program instead of plain `tex`. That is, run `pdftex foo.texi` instead of `tex foo.texi` in the examples above.

### 18.3.1 Formatting Partial Documents

Sometimes you may wish to print a document while you know it is incomplete, or to print just one chapter of a document. In such a case, the usual auxiliary files that \TeX creates and warnings \TeX gives about undefined cross-references are just nuisances. You can avoid them with the `@novalidate` command, which you must give before any sectioning or cross-reference commands.

Thus, the beginning of your file would look approximately like this:

```
\input texinfo
@novalidate
...
```

`@novalidate` also turns off validation in `texi2any`, just like its `--no-validate` option (see Section 19.1 [Invoking `texi2any`], page 164).
Furthermore, you need not run `texindex` each time after you run `tex`. The `tex` formatting command simply uses whatever sorted index files happen to exist from a previous use of `texindex`. If those are out of date, that is usually ok while you are creating or debugging a document.

**18.3.2 Details of `texindex`**

In Texinfo version 6, released in 2015, the `texindex` program was completely reimplemented. The principal functional difference is that index entries beginning with a left brace or right brace (’{’ resp. ’}’) can work properly. For example, these simple index entries are processed correctly, including the “index initial” shown in the index:

```latex
cindex {}
cindex @
...
printindex cp
```

Although not a matter of functionality, readers may be interested to know that the new `texindex` is a literate program ([https://en.wikipedia.org/wiki/Literate_programming](https://en.wikipedia.org/wiki/Literate_programming)) using Texinfo for documentation and (portable) `awk` for code. A single source file, `texindex/ti.twjr` in this case, produces the runnable program, a printable document, and an online document.

The system is called TexiWeb Jr. and was created by Arnold Robbins, who also wrote the new `texindex`. Not coincidentally, he is also the long-time maintainer of `gawk` (GNU Awk, see [The GNU Awk User’s Guide](https://www.gnu.org/software/gawk/manual/)). The file `texindex/Makefile.am` shows example usage of the system.

**18.4 Print with `lpr` from Shell**

The way to print a DVI file depends on your system installation. Two common ones are `dvips foo.dvi -o` to make a PostScript file first and then print that, and `lpr -d foo.dvi` to print a DVI file directly.

For example, the following commands will (probably) suffice to sort the indices, format, and print this manual using the `texi2dvi` shell script (see [Format with `texi2dvi`], page 154).

```bash
texi2dvi texinfo.texi
dvips texinfo.dvi -o
lpr texinfo.ps
```

Depending on the `lpr` setup on your machine, you might be able to combine the last two steps into `lpr -d texinfo.dvi`.

You can also generate a PDF file by running `texi2dvi` instead of `texi2dvi`; a PDF is often directly printable. Or you can generate a PCL file by using `dvilj` instead of `dvips`, if you have a printer that prefers that format.

`lpr` is a standard program on Unix systems, but it is usually absent on MS-DOS/MS-Windows. If so, just create a PostScript or PDF or PCL file, whatever is most convenient, and print that in the usual way for your machine (e.g., by sending to the appropriate port, usually ‘PRN’).
18.5 Formatting and Printing with Emacs

GNU Emacs can be used for formatting and printing with \TeX\, from an Emacs Shell. Texinfo mode also provides predefined key commands for formatting and printing.

18.5.1 Printing From an Emacs Shell

You can give formatting and printing commands from a shell within GNU Emacs, just like any other shell command. To create a shell within Emacs, type \texttt{M-x shell} (see Section “Shell” in The GNU Emacs Manual). In this shell, you can format and print the document. See Chapter 18 [Format and Print Hardcopy], page 154, for details.

You can switch to and from the shell buffer while \texttt{tex} is running and do other editing. If you are formatting a long document on a slow machine, this can be very convenient.

For example, you can use \texttt{texi2dvi} from an Emacs shell. Here is one way to use \texttt{texi2dvi} to format and print \textit{Using and Porting GNU CC} from a shell within Emacs:

\begin{verbatim}
texi2pdf gcc.texi
lpr gcc.pdf
\end{verbatim}

See the next section for more information about formatting and printing in Texinfo mode.

18.5.2 Formatting and Printing in Texinfo Mode

Texinfo mode provides several predefined key commands for \TeX\ formatting and printing. These include commands for sorting indices, looking at the printer queue, killing the formatting job, and recentering the display of the buffer in which the operations occur.

\begin{verbatim}
C-c C-t C-b
M-x texinfo-tex-buffer
Run \texttt{texi2dvi} on the current buffer.

C-c C-t C-r
M-x texinfo-tex-region
Run \TeX\ on the current region.
If @-commands related to printed output are between the start-of-header and end-of-header lines, then \texttt{texinfo-tex-region} will format the region accordingly. For example, if you write the @smallbook command between the start-of-header and end-of-header lines, \texttt{texinfo-tex-region}, will format the region in “small” book size.

C-c C-t C-i
M-x texinfo-texindex
Sort the indices of a Texinfo file formatted with \texttt{texinfo-tex-region}.

C-c C-t C-p
M-x texinfo-tex-print
Print a DVI file that was made with \texttt{texinfo-tex-region} or \texttt{texinfo-tex-buffer}.

C-c C-t C-q
M-x tex-show-print-queue
Show the print queue.
\end{verbatim}
C-c C-t C-d
M-x texinfo-delete-from-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 (texinfo-show-tex-print-queue).

C-c C-t C-k
M-x tex-kill-job
   Kill the currently running TeX job started by either texinfo-tex-region or
   texinfo-tex-buffer, or any other process running in the Texinfo shell buffer.

C-c C-t C-x
M-x texinfo-quit-job
   Quit a TeX formatting job that has stopped because of an error by sending an
   x to it. When you do this, TeX preserves a record of what it did in a .log file.

C-c C-t C-l
M-x tex-recenter-output-buffer
   Redisplay the shell buffer in which the TeX 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):
   C-c C-t C-b   Run texi2dvi on the buffer.
   C-c C-t C-p   Print the DVI file.
   C-c C-t C-q   Display the printer queue.

The Texinfo mode TeX formatting commands start a subshell in Emacs called the
*tex-shell*. The texinfo-tex-command, texinfo-texindex-command, and tex-dvi
print-command commands are all run in this shell.

You can watch the commands operate in the ‘*tex-shell*’ buffer, and you can switch
to and from and use the ‘*tex-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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>texinfo-texi2dvi-command</td>
<td>&quot;texi2dvi&quot;</td>
</tr>
<tr>
<td>texinfo-tex-command</td>
<td>&quot;tex&quot;</td>
</tr>
<tr>
<td>texinfo-texindex-command</td>
<td>&quot;texindex&quot;</td>
</tr>
<tr>
<td>texinfo-delete-from-print-queue-command</td>
<td>&quot;lprm&quot;</td>
</tr>
<tr>
<td>texinfo-tex-trailer</td>
<td>&quot;@bye&quot;</td>
</tr>
<tr>
<td>tex-start-of-header</td>
<td>&quot;%**start&quot;</td>
</tr>
<tr>
<td>tex-end-of-header</td>
<td>&quot;%**end&quot;</td>
</tr>
<tr>
<td>tex-dvi-print-command</td>
<td>&quot;lpr -d&quot;</td>
</tr>
<tr>
<td>tex-show-queue-command</td>
<td>&quot;lpq&quot;</td>
</tr>
</tbody>
</table>

You can change the values of these variables with the M-x set-variable command (see
   Section “Examining and Setting Variables” in The GNU Emacs Manual), or with your
   .emacs initialization file (see Section “Init File” in The GNU Emacs Manual).

   Beginning with version 20, GNU Emacs offers a user-friendly interface, called Customize,
   for changing values of user-definable variables. See Section “Easy Customization Interface”
18.5.3 Using the Local Variables List

Yet another way to apply the \TeX formatting command to a Texinfo file is to put that command in a local variables list at the end of the Texinfo file. You can then specify the \texttt{tex} or \texttt{texi2dvi} commands as a compile-command and have Emacs run it by typing \texttt{M-x customize} command. This creates a special shell called the \texttt{*compilation*} buffer in which Emacs runs the compile command. For example, at the end of the \texttt{gdb.texi} file, after the \texttt{@bye}, you could put the following:

\begin{verbatim}
Local Variables:
  compile-command: "texi2dvi gdb.texi"
End:
\end{verbatim}

This technique is most often used by programmers who also compile programs this way; see Section “Compilation” in \textit{The GNU Emacs Manual}.

18.6 \TeX Formatting Requirements Summary

Every Texinfo file that is to be input to \TeX must begin with a \texttt{\input} command:

\texttt{\input texinfo}

This instructs \TeX to load the macros it needs to process a Texinfo file.

Every Texinfo file must end with a line that terminates \TeX’s processing and forces out unfinished pages:

\texttt{@bye}

Strictly speaking, these two lines are all a Texinfo file needs to be processed successfully by \TeX.

Usually, however, the beginning includes a \texttt{@settitle} command to define the title of the printed manual, a title page, a copyright page, permissions, and a table of contents. Besides \texttt{@bye}, the end of a file usually includes indices. (Not to mention that most manuals contain a body of text as well.)

For more information, see:

- Section 2.5.3 \texttt{@settitle}, page 14.
- Section E.1 \texttt{@setchapternewpage}, page 269.
- Section E.2 \texttt{[Headings]}, page 270.
- Section 2.8 \texttt{[Titlepage & Copyright Page]}, page 18.
- Section 10.5 \texttt{[Printing Indices & Menus]}, page 93.
- Section 2.9 \texttt{[Contents]}, page 22.

18.7 Preparing for \TeX

\TeX needs to find the \texttt{texinfo.tex} file that the \texttt{\input texinfo} command on the first line reads. The \texttt{texinfo.tex} file tells \TeX how to handle \texttt{@}-commands; it is included in all standard GNU distributions. The latest version released for general use is available from the usual GNU servers and mirrors:

\texttt{http://ftp.gnu.org/gnu/texinfo/texinfo.tex}
http://ftpmirror.gnu.org/texinfo/texinfo.tex

The latest development version is available from the Texinfo source repository:
http://git.savannah.gnu.org/cgit/texinfo.git/plain/doc/texinfo.tex

texinfo.tex is essentially a standalone file, so, if you need or want to try a newer version
than came with your system, it nearly always suffices to download it and put it anywhere
that \TeX will find it. You can replace any existing texinfo.tex with a newer version (of
course saving the original in case of disaster).

Also, you should install epsf.tex, if it is not already installed from another distribution.
More details are at the end of the description of the @image command (see Section 9.2 [Images], page 85).

To use quotation marks other than those used in English, you’ll need to have the
European Computer Modern fonts (e.g., ecrm1000) and (for PDF output) CM-Super fonts
(see Section 11.5 [Inserting Quotation Marks], page 104).

To use the @euro command, you’ll need the ‘feym*’ fonts (e.g., feymr10). See
Section 11.8.6 [@euro], page 107.

All of the above files should be installed by default in a reasonable \TeX installation.

Optionally, you may create a file texinfo.cnf for site configuration. When processing a
Texinfo file, \TeX looks for this file in its search path, which includes the current directory
and standard installation directories. You can use this file for local conventions. For example,
if texinfo.cnf contains the line ‘@afourpaper’ (see Section E.7 [A4 Paper], page 275), then
all Texinfo documents will be processed with that page size in effect. If you have nothing to
put in texinfo.cnf, you do not need to create it.

You can set the TEXINPUTS environment variable to allow \TeX to find texinfo.cnf.
(This also works for texinfo.tex and any other file \TeX might read). For example, if you
are using a Bourne shell-compatible shell (sh, bash, ksh, . . .), your .profile file could
contain the lines:

TEXINPUTS=.:/home/me/mylib:
export TEXINPUTS

These settings would cause \TeX first to look for an \input file in the current directory,
indicated by the ‘.’, then in a hypothetical user ‘me’’s mylib directory, and finally in the
system directories. (A leading, trailing, or doubled ‘:’ indicates searching the system
directories at that point.)

18.8 Overfull “hboxes”

\TeX is sometimes unable to typeset a line within the normal margins. This most often
occurs when \TeX comes upon what it interprets as a long word that it cannot hyphenate,
such as an electronic mail network address or a very long identifier. When this happens,
\TeX prints an error message like this:

Overfull \hbox (20.76302pt too wide)

(In \TeX, lines are in “horizontal boxes”, hence the term, “hbox”. ‘\hbox’ is a \TeX primitive
not used in the Texinfo language.)

\TeX also provides the line number in the Texinfo source file and the text of the offending
line, which is marked at all the places that \TeX considered hyphenation. See Section D.10.3
[Debugging with \TeX], page 265, for more information about typesetting errors.
If the Texinfo 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.

If you have many overfull boxes and/or an antipathy to rewriting, you can coerce \TeX into greatly increasing the allowable interword spacing, thus (if you’re lucky) avoiding many of the bad line breaks, like this:

\begin{verbatim}
@tex
\global\emergencystretch = .9\hsize
@end tex
\end{verbatim}

(You should adjust the fraction as needed.) This huge value for \emergencystretch cannot be the default, since then the typeset output would generally be of noticeably lower quality; its default value is \texttt{.15\hsize}. \hsize is the \TeX dimension containing the current line width.

For any overfull boxes you do have, \TeX will print a large, ugly, black rectangle beside the line that contains the overfull hbox unless told otherwise. This is so you will notice the location of the problem if you are correcting a draft.

To prevent such a monstrosity from marring your final printout, write the following in the beginning of the Texinfo file on a line of its own, before the \texttt{@titlepage} command:

\begin{verbatim}
@finalout
\end{verbatim}
19 texi2any: The Translator for Texinfo

texi2any is the generic translator for Texinfo that can produce different output formats and is highly customizable. It supports these formats:

- Info (by default, or with --info),
- HTML (with --html),
- EPUB 3 (with --epub3),
- plain text (with --plaintext),
- LATEX (with --latex),
- DocBook (with --docbook),
- Texinfo XML (with --xml).

makeinfo is an alias for texi2any. By default, both texi2any and makeinfo generate Info output; indeed, there are no differences in behavior based on the name.

Beside these default formats, command line options to texi2any can change many aspects of the output. Beyond that, initialization files provide even more control over the final output—nearly anything not specified in the Texinfo input file. Initialization files are written in Perl, like the main program, and anything which can be specified on the command line can also be specified within a initialization file.

19.1 Invoking texi2any from a Shell

To process a Texinfo file, invoke texi2any followed by the name of the Texinfo file. Also select the format you want to output with the appropriate command line option (default is Info). Thus, to create the Info file for Bison, type the following to the shell:

texi2any --info bison.texi

You can specify more than one input file name; each is processed in turn. If an input file name is '-', standard input is read.

The texi2any program accepts many options. Perhaps the most basic are those that change the output format. By default, texi2any outputs Info.

Each command line option is either a long name preceded by '--' or a single letter preceded by '-'. You can use abbreviations for the long option names as long as they are unique.

For example, you could use the following shell command to create an Info file for bison.texi in which lines are filled to only 68 columns:

texi2any --fill-column=68 bison.texi

You can write two or more options in sequence, like this:

texi2any --no-split --fill-column=70 ...

(This would keep the Info file together as one possibly very long file and would also set the fill column to 70.)

The options are (approximately in alphabetical order):

--commands-in-node-names

This option now does nothing, but remains for compatibility. (It used to ensure that @-commands in node names were expanded throughout the document, especially @value. This is now done by default.)
--conf-dir=dir
Prepend dir to the directory search list for finding customization files that may be loaded with --init-file (see below). The dir value can be a single directory, or a list of several directories separated by the usual path separator character (‘:’ on Unix-like systems, ‘;’ on Windows).

--css-include=file
When producing HTML, literally include the contents of file, which should contain Cascading Style Sheets specifications, in the ‘<style>’ block of the HTML output. If file is ‘-’, read standard input. See Section 21.3 [HTML CSS], page 204.

--css-ref=url
When producing HTML, add a ‘<link>’ tag to the output which references a cascading style sheet at url. This allows using standalone style sheets.

-D var
-D 'var value'
Cause the Texinfo variable var to be defined. This is equivalent to @set var in the Texinfo file (see Section 15.5 [@set @clear @value], page 137).

The argument to the option is always one word to the shell; if it contains internal whitespace, the first word is taken as the variable name and the remainder as the value. For example, -D 'myvar someval' is equivalent to @set myvar someval.

--docbook
Generate DocBook output (rather than Info).

--document-language=lang
Use lang to translate Texinfo keywords which end up in the output document. The default is the locale specified by the @documentlanguage command if there is one, otherwise English (see Section 14.1 [@documentlanguage], page 130).

--dvi
Generate a TeX DVI file using texi2dvi, rather than Info (see Section 19.3 [texi2any Printed Output], page 171).

--dvipdf
Generate a PDF file using texi2dvi --dvipdf, rather than Info (see Section 19.3 [texi2any Printed Output], page 171).

--enable-encoding
--disable-encoding
By default, or with --enable-encoding, output accented and special characters in Info and plain text output based on the document encoding. With --disable-encoding, 7-bit ASCII transliterations are output.

By default, or with --disable-encoding, output accented and special characters in HTML, XML and DocBook using XML entities. With --enable-encoding, output accented characters in HTML, XML and DocBook output and special characters in HTML output based on the document encoding. See Section 14.2 [@documentencoding], page 131, and Section 11.4 [Inserting Accents], page 103.

--epub3
Generate EPUB 3 output.
--error-limit=limit
-e limit Report limit errors before aborting (on the assumption that continuing would be useless); default 100.

--fill-column=width
-f width Specify the maximum number of columns in a line; this is the right-hand edge of a line. Paragraphs that are filled will be filled to this width. (Filling is the process of breaking up and connecting lines so that lines are the same length as or shorter than the number specified as the fill column. Lines are broken between words.) The default value is 72.

--footnote-style=style
-s style Set the footnote style to style: either ‘end’ for the end node style (the default) or ‘separate’ for the separate node style. The value set by this option overrides the value set in a Texinfo file by a @footnotestyle command (see Section 9.3.2 [Footnote Styles], page 88).

In Info, when the footnote style is ‘separate’, texi2any makes a new node containing the footnotes found in the current node. When the footnote style is ‘end’, texi2any places the footnote references at the end of the current node.

In HTML, when the footnote style is ‘end’, or if the output is not split, footnotes are put at the end of the output. If set to ‘separate’, and the output is split, they are placed in a separate file.

--force
-F Ordinarily, if the input file has errors, the output files are not created. With this option, they are preserved.

--help
-h Print a message with available options and basic usage, then exit successfully.

--html
Generate HTML output (rather than Info). By default, the HTML output is split into one output file per node, and the split output is written into a subdirectory based on the name of the input file. See Chapter 21 [Generating HTML], page 203.

-1 dir Append dir to the directory search list for finding files that are included using the @include command. By default, texi2any searches only the current directory. If dir is not given, the current directory is appended. The dir value can be a single directory or a list of several directories separated by the usual path separator character (‘:’ on Unix-like systems, ‘;’ on Windows).

--ifdocbook
--ifhtml
--ifinfo
--iflatex
--ifplaintext
--iftex
--ifxml For the given format, process ‘@ifformat’ and ‘@format’ commands, and do not process ‘@ifnotformat’, regardless of the format being output. For instance, if --iftex is given, then ‘@iftex’ and ‘@tex’ blocks will be read, and ‘@ifnottex’ blocks will be ignored.
--no-ifdocbook
--no-ifhtml
--no-ifinfo
--no-iflatex
--no-ifplaintext
--no-iftex
--no-ifxml

For the given format, do not process ‘@if format’ and ‘@format’ commands, and
do process ‘@ifnot format’, regardless of the format being output. For instance,
if --no-ifhtml is given, then ‘@ifhtml’ and ‘@html’ blocks will not be read,
and ‘@ifnothtml’ blocks will be.

--info
Generate Info output. By default, if the output file contains more than about
300,000 bytes, it is split into shorter subfiles of about that size. The name of
the output file and any subfiles is determined by the input file name, or by
@setfilename, if present (see [Setting the Output File Name], page 13). See
Section 20.2 [Tag and Split Files], page 199.

--init-file=file
Load file as code to modify the behavior and output of the generated manual.
It is customary to use the .pm or the .init extensions for these customization
files, but that is not enforced; the file name can be anything. The --conf-dir
option (see above) can be used to add to the list of directories in which these
customization files are searched for.

--internal-links=file
In HTML mode, output a tab-separated file containing three columns: the
internal link to an indexed item or item in the table of contents, the name of the
index (or table of contents) in which it occurs, and the term which was indexed
or entered. The items are in the natural sorting order for the given element.
This dump can be useful for post-processors.

--latex
Generate LaTeX output.

--macro-expand=file
-E file Output the Texinfo source, with all Texinfo macros expanded, to file. Normally,
the result of macro expansion is used internally by texi2any and then discarded.

--no-headers
Do not include menus or node separator lines in the output.
When generating Info, this is the same as using --plaintext, resulting in
a simple plain text file. Furthermore, output is to standard output unless
overridden with -o. (This behavior is for backward compatibility.)
When generating HTML, and output is split, also output navigation links only at
the beginning of each file. If output is not split, do not include navigation links
at the top of each node at all. See Chapter 21 [Generating HTML], page 203.

--node-files
--no-node-files
With --node-files, when generating HTML, create redirection files for anchors
and any nodes not already output with the file name corresponding to the node
name (see Section 21.6.2 [HTML Xref Node Name Expansion], page 208). This makes it possible for section- and chapter-level cross-manual references to succeed (see Section 21.6.6 [HTML Xref Configuration], page 212).

If the output is split, this is enabled by default. If the output is not split, --node-files enables the creation of the redirection files, in addition to the monolithic main output file. --no-node-files suppresses the creation of redirection files in any case. This option has no effect with any output format other than HTML. See Chapter 21 [Generating HTML], page 203.

--no-validate
--no-pointer-validate
Suppress the pointer-validation phase of texi2any—a dangerous thing to do. This can also be done with the @novalidate command (see Section 18.3.1 [Formatting Partial Documents], page 157).

If you do not suppress pointer validation, texi2any will check the validity of cross-references and menu entries in the Texinfo file, as well as node pointers if they are given explicitly.

--no-warn
Suppress warning messages (but not error messages).

--number-footnotes
--no-number-footnotes
With --no-number-footnotes, suppress automatic footnote numbering. By default, footnotes are numbered sequentially within a node, i.e., the current footnote number is reset to 1 at the start of each node.

--number-sections
--no-number-sections
With --number-sections (the default), output chapter, section, and appendix numbers as in printed manuals. This works only with hierarchically structured manuals. You should specify --no-number-sections if your manual is not normally structured.

--output=file
-o file
Specify that the output should be directed to file. This overrides any file name specified in a @setfilename command found in the Texinfo source. If neither @setfilename nor this option are specified, the input file name is used to determine the output name. See [Setting the Output File Name], page 13.

If file is ‘-’, output goes to standard output and ‘--no-split’ is implied.

If file is a directory or ends with a ‘/’ the usual rules are used to determine the output file name (namely, use @setfilename or the input file name) but the files are written to the file directory. For example, ‘texi2any -o bar/ foo.texi’, with or without --no-split, will write bar/foo.info, and possibly other files, under bar/.

When generating HTML and output is split, file is used as the name for the directory into which all files are written. For example, ‘texi2any -o bar --html foo.texi’ will write bar/index.html, among other files.
When generating EPUB a container directory for the files and directories needed for the EPUB format is created, as well as the EPUB output file. If \textit{file} corresponds to a directory, the container directory is placed within \textit{file}. The EPUB output file is never put in this directory. If \textit{file} corresponds to a file, it is used for the EPUB output file name.

\texttt{--output-indent=\textit{val}}

This option now does nothing, but remains for compatibility. (It used to alter indentation in XML/DocBook output.)

\texttt{-P \textit{path}}

Prepend \textit{path} to the directory search list for @include. If \textit{path} is not given, the current directory is prepended. See \texttt{--I} above.

\texttt{--paragraph-indent=\textit{indent}}

\texttt{-p \textit{indent}}

Set the paragraph indentation style to \textit{indent}. The value set by this option overrides the value set in a Texinfo file by an @paragraphindent command (see Section E.3 [@paragraphindent], page 274). The value of \textit{indent} is interpreted as follows:

\texttt{\textit{asis}}

Preserve any existing indentation (or lack thereof) at the beginnings of paragraphs.

\texttt{\textit{0} or \textit{none}}

Delete any existing indentation.

\texttt{\textit{num}}

Indent each paragraph by \textit{num} spaces.

The default is to indent by two spaces, except for paragraphs following a section heading, which are not indented.

\texttt{--pdf}

Generate a PDF file using \texttt{texi2dvi --pdf}, rather than Info (see Section 19.3 [\texttt{texi2any} Printed Output], page 171).

\texttt{--plaintext}

Output a plain text file (rather than Info): do not include menus or node separator lines in the output. This results in a straightforward plain text file that you can (for example) send in email without complications, or include in a distribution (for example, an INSTALL file).

With this option the output goes to standard output by default, instead of to a file with a name based on the input file name or @setfilename; this can be overridden with \texttt{-o}.

\texttt{--ps}

Generate a PostScript file using \texttt{texi2dvi --ps}, rather than Info (see Section 19.3 [\texttt{texi2any} Printed Output], page 171).

\texttt{--set-customization-variable \textit{var}=\textit{value}}

\texttt{-c \textit{var}=\textit{value}}

Set the customization variable \textit{var} to \textit{value}. The = is optional, but both \textit{var} and \textit{value} must be quoted to the shell as necessary so the result is a single word. Many aspects of \texttt{texi2any} behavior and output may be controlled by customization variables, beyond what can be set in the document by @-commands and with other command line switches. See Section 19.4 [Customization Variables], page 172.
--split=how
--no-split

When generating Info, by default large output files are split into smaller subfiles, of approximately 300k bytes. When generating HTML, by default each output file contains one node (see Chapter 21 [Generating HTML], page 203). --no-split suppresses this splitting of the output.

Alternatively, --split=how may be used to specify at which level the HTML output should be split. The possible values for how are:

‘chapter’ The output is split at @chapter and other sectioning @-commands at this level (@appendix, etc.).

‘section’ The output is split at @section and similar.

‘node’ The output is split at every node. This is the default.

Plain text output can be split similarly to HTML. This may be useful for extracting sections from a Texinfo document and making them available as separate files.

--split-size=num

Keep Info files to at most num characters if possible; default is 300,000. (However, a single node will never be split across Info files.)

--transliterate-file-names

Enable transliteration of non-ASCII characters in node names for the purpose of file name creation. See Section 21.6.4 [HTML Xref 8-bit Character Expansion], page 211.

-U var

Cause var to be undefined. This is equivalent to @clear var in the Texinfo file (see Section 15.5 [@set @clear @value], page 137).

--verbose

Cause texi2any to display messages saying what it is doing. Normally, texi2any only outputs messages if there are errors or warnings.

--version

-V

Print the version number, then exit successfully.

--Xopt str

Pass str (a single shell word) to texi2dvi; may be repeated (see Section 19.3 [texi2any Printed Output], page 171).

--xml

Generate Texinfo XML output (rather than Info).

19.2 Environment Variables Recognized by texi2any

texi2any also reads the environment variable TEXINFO_OUTPUT_FORMAT to determine the output format, if not overridden by a command line option. The value should be one of:

docbook dvi dvipdf epub3 html info latex pdf plaintext ps xml

If not set or otherwise specified, Info output is the default.
TEXINFO_OUTPUT_FORMAT may take some other special values, which do not correspond to any command-line settable output format (see Section 19.4.2 [Customization Variables and Options], page 173).

The customization variable of the same name is also read; if set, that overrides an environment variable setting, but not a command-line option. See Section 19.4.2 [Customization Variables and Options], page 173.

You can control texi2any’s use of Perl extension modules by setting the TEXINFO_XS environment variable. These modules are compiled native code that the interpreted Perl code can use. Ideally, these extension modules should just work, and the only noticeable difference they should make is that texi2any finishes running sooner. However, you can use this environment variable for the purposes of troubleshooting: for example, if you have problems with the output of texi2any varying depending on whether the extension modules are in use.

The following values of TEXINFO_XS are recognized by texi2any:

- **‘default’**  The default behavior. Try to load extension modules, and silently fall back to the interpreted Perl implementations if this fails.

- **‘warn’**  Try to load extension modules, and if this fails, give a warning message before falling back to the interpreted Perl implementations.

- **‘debug’**  Try to load extension modules, printing many messages while doing so.

- **‘omit’**  Do not use extension modules.

Set TEXINFO_XS_PARSER to ‘0’ to disable the use of the native code implementation of the parser module. This is the part of texi2any that converts Texinfo input into an internal tree format used for further processing into output formats. This may be useful for working around bugs or incompatibilities between the native code implementation and the implementation in pure Perl code.

### 19.3 texi2any Printed Output

To justify the name Texinfo-to-any, texi2any has basic support for creating printed output in the various formats: TEX DVI, PDF, and PostScript. This is done via the simple method of executing the texi2dvi program when those output formats are requested, after checking the validity of the input to give users the benefit of texi2any’s error checking. If you don’t want such error checking, perhaps because your manual plays advanced TEX tricks together with texinfo.tex, just invoke texi2dvi directly.

The output format options for this are --dvi, --dvipdf, --pdf, and --ps. See [Format with texi2dvi], page 154, for more details on these options and general texi2dvi operation. In addition, the --verbose, --silent, and --quiet options are passed on if specified; the -1 and -o options are likewise passed on with their arguments, and --debug without its argument.

The only option remaining that is related to the texi2dvi invocation is --Xopt. Here, just the argument is passed on and multiple --Xopt options accumulate. This provides a way to construct an arbitrary command line for texi2dvi. For example, running

```
texi2any --Xopt -t --Xopt @a4paper --pdf foo.texi
```
is equivalent to running

```
texi2dvi -t @a4paper --pdf foo.texi
```
except for the validity check.

Although one might wish that other options to texi2any would take effect, they don’t. For example, running `texi2any --no-number-sections --dvi foo.texi` still results in a DVI file with numbered sections. (Perhaps this could be improved in the future, if requests are received.)

The actual name of the command that is invoked is specified by the TEXI2DVI customization variable (see Section 19.4.7 [Other Customization Variables], page 182). As you might guess, the default is ‘texi2dvi’.

texi2any itself does not generate any normal output when it invokes texi2dvi, only diagnostic messages.

### 19.4 Customization Variables

**Warning:** These customization variable names and meanings may change in any Texinfo release. We always try to avoid incompatible changes, but we cannot absolutely promise, since needs change over time.

Many aspects of the behavior and output of texi2any may be modified by modifying so-called *customization variables*. These fall into a few general categories:

- Those associated with @-commands; for example, @documentlanguage.
- Those associated with command-line options; for example, the customization variable SPLIT is associated with the --split command-line option, and TEXINFO_OUTPUT_FORMAT allows specifying the output format.
- Those associated with customizing the HTML output.
- Other ad hoc variables.

Customization variables may set on the command line using --set-customization-variable ’var value’ (quoting the variable/value pair to the shell) or --set-customization-variable var=value (using =). A special value is ‘undef’, which sets the variable to this special “undefined” Perl value.

The sections below give the details for each of these.

#### 19.4.1 Customization Variables for @-Commands

Each of the following @-commands has an associated customization variable with the same name (minus the leading @):

- @afivepaper
- @afourpaper
- @afourlatex
- @afourwide
- @allowcodebreaks
- @bsixpaper
- @contents
- @clickstyle
- @codequotebacktick
- @codequoteundirected
- @deftypedefnnewline
- @documentdescription
- @documentencoding
- @documentlanguage
- @evenfooting
- @evenfootingmarks
- @exampleindent
- @firstparagraphindent
- @fonttextsize
- @footnotestyle
- @frenchspacing
Setting such a customization variable to a value ‘foo’ is similar to executing `@cmd foo`. It is not exactly the same, though, since any side effects of parsing the Texinfo source are not redone. Also, some variables do not take Texinfo code when generating particular formats, but an argument that is already formatted. This is the case, for example, for HTML for `documentdescription`.

Note that if `texi2any` is invoked to process the file with \TeX (e.g., with the `--pdf` option), then these customization variables may not be passed on to \TeX.

19.4.2 Customization Variables and Options

The following table gives the customization variables associated with some command line options. See Section 19.1 [Invoking `texi2any`], page 164, for the meaning of the options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>--enable-encoding</td>
<td>ENABLE_ENCODING</td>
</tr>
<tr>
<td>--document-language</td>
<td>documentlanguage</td>
</tr>
<tr>
<td>--error-limit</td>
<td>ERROR_LIMIT</td>
</tr>
<tr>
<td>--fill-column</td>
<td>FILLCOLUMN</td>
</tr>
<tr>
<td>--footnote-style</td>
<td>footnotestyle</td>
</tr>
<tr>
<td>--force</td>
<td>FORCE</td>
</tr>
<tr>
<td>--internal-links</td>
<td>INTERNAL_LINKS</td>
</tr>
<tr>
<td>--macro-expand</td>
<td>MACRO_EXPAND</td>
</tr>
<tr>
<td>--headers</td>
<td>HEADERS, FORMAT_MENU</td>
</tr>
<tr>
<td>--no-warn</td>
<td>NO_WARN</td>
</tr>
<tr>
<td>--no-validate</td>
<td>novalidate</td>
</tr>
<tr>
<td>--number-footnotes</td>
<td>NUMBER_FOOTNOTES</td>
</tr>
<tr>
<td>--number-sections</td>
<td>NUMBER_SECTIONS</td>
</tr>
<tr>
<td>--node-files</td>
<td>NODE_FILES</td>
</tr>
<tr>
<td>--output</td>
<td>OUTFILE, SUBDIR</td>
</tr>
<tr>
<td>--paragraph-indent</td>
<td>paragraphindent</td>
</tr>
<tr>
<td>--silent</td>
<td>SILENT</td>
</tr>
<tr>
<td>--split</td>
<td>SPLIT</td>
</tr>
<tr>
<td>--split-size</td>
<td>SPLIT_SIZE</td>
</tr>
<tr>
<td>--transliterate-file-names</td>
<td>TRANSLITERATE_FILE_NAMES</td>
</tr>
<tr>
<td>--verbose</td>
<td>VERBOSERE</td>
</tr>
</tbody>
</table>

Setting such a customization variable to a value ‘foo’ is essentially the same as specifying the `--opt=foo` if the option takes an argument, or `--opt` if not.

In addition, the customization variable `TEXINFO_OUTPUT_FORMAT` allows specifying what `texi2any` outputs, either one of the usual output formats that can be specified with options, or various other forms:
Chapter 19: texi2any: The Translator for Texinfo

'docbook'
'dvi'
'dvipdf'
'epub3'
'html'
'info'
'pdf'
'plaintext'
'ps'
'xml' These correspond to the command-line options (and TEXINFO_OUTPUT_FORMAT environment variable values) of the same name. See Section 19.1 [Invoking texi2any], page 164.

'debugtree' Instead of generating a regular output format, output a text representation of the tree obtained by parsing the input texinfo document.

'parse' Do only Texinfo source parsing; there is no output.

'plaintexinfo' Output the Texinfo source with all the macros, @include and @value{} expanded. This is similar to setting --macro-expand, but instead of being output in addition to the normal conversion, output of Texinfo is the main output.

'rawtext' Output raw text, with minimal formatting. For example, footnotes are ignored and there is no paragraph filling. This is used by the parser for file names and copyright text in HTML comments, for example.

'structure' Do only Texinfo source parsing and determination of the document structure; there is no output.

'texinfosxml' Output the document in TexinfoSXML representation, a syntax for writing XML data using Lisp S-expressions.

'textcontent' Output the text content only, stripped of commands; this is useful for spell checking or word counting, for example. The trivial detexinfo script setting this is in the util directory of the Texinfo source as an example. It’s one line:

```
exec texi2any -c TEXINPUT_OUTPUT_FORMAT=textcontent "$@
```

19.4.3 HTML Customization Variables

This table gives the customization variables which apply to HTML output only. A few other customization variables apply to both HTML and other output formats; see Section 19.4.7 [Other Customization Variables], page 182.

AVOID_MENU_REDUNDANCY
If set, and the menu entry and menu description are the same, then do not print the menu description; default false.
Chapter 19: texi2any: The Translator for Texinfo

AFTER_BODY_OPEN
If set, the corresponding text will appear at the beginning of each HTML file; default unset.

AFTER_SHORT_TOC_LINES
AFTER_TOC_LINES
If set, the corresponding text is output after the short table of contents for
AFTER_SHORT_TOC_LINES and after the table of contents for AFTER_TOC_LINES; otherwise, a default string is used. At the time of writing, a \texttt{</div>} element is closed.

In general, you should set BEFORE_SHORT_TOC_LINES if AFTERSHORT_TOC_LINES is set, and you should set BEFORE_TOC_LINES if AFTER_TOC_LINES is set.

BASEFILENAME_LENGTH
The maximum length of a base file name; default 245. Changing this would make cross-manual references to such long node names invalid (see Section 21.6.1 [HTML Xref Link Basics], page 207).

BEFORE_SHORT_TOC_LINES
BEFORE_TOC_LINES
If set, the corresponding text is output before the short table of contents for
BEFORE_SHORT_TOC_LINES and before the table of contents for BEFORE_TOC_LINES, otherwise a default string is used. At the time of writing, a \texttt{<div ...> element is opened.

In general you should set AFTER_SHORT_TOC_LINES if BEFORE_SHORT_TOC_LINES is set, and you should set AFTER_TOC_LINES if BEFORE_TOC_LINES is set.

BIG_RULE
Rule used after and before the top element and before special elements, but not for footers and headers; default \texttt{<hr>.

BODYTEXT
The text appearing in \texttt{<body>}. By default, sets the HTML \texttt{lang} attribute to the document language (see Section 14.1 [@documentlanguage], page 130).

CASE_INSENSITIVE_FILENAMES
Construct output file names as if the filesystem were case insensitive (see Section 21.2 [HTML Splitting], page 204); default false.

CHAPTER_HEADER_LEVEL
Header formatting level used for chapter level sectioning commands; default ‘2’.

CHECK_HTMLXREF
Check that manuals which are the target of external cross-references (see Section 5.6 [Four and Five Arguments], page 46) are present in \texttt{htmlxref.cnf} (see Section 21.6.6 [HTML Xref Configuration], page 212); default false.

COMPLEX_FORMAT_IN_TABLE
If set, use tables for indentation of complex formats; default false.

CONTENTS_OUTPUT_LOCATION
If set to ‘after\_top’, output the contents at the end of the @top section. If set to ‘inline’, output the contents where the @contents and similar @-commands
are located. If set to ‘**separate_element**’ output the contents in separate elements, either at the end of the document if not split, or in a separate file. If set to ‘**after_title**’ the tables of contents are output after the title; default ‘**after_top**’.

**CONVERT_TO_LATEX_IN_MATH**
If set, try to convert any Texinfo @-commands inside @math and @displaymath to \( \text{La} \text{TeX} \), before converting the @math or @displaymath to HTML. Default **undef**. If undefined, set if HTML_MATH is set.

**COPIABLE_LINKS**
If set, output copiable links for the definition commands (see Chapter 13 [Definition Commands], page 117) and table commands (see Section 8.4 [Two-column Tables], page 78) where an index entry is defined. A link appears as a ‘¶’ sign that appears when you hover the mouse pointer over the heading text.

**DATE_IN_HEADER**
Put the document generation date in the header; off by default.

**DEF_TABLE**
If set, a `<table>` construction for @deffn and similar @-commands is used (looking more like the \( \text{TeX} \) output), instead of definition lists; default false.

**DEFAULT_RULE**
Rule used between element, except before and after the top element, and before special elements, and for footers and headers; default `<hr>`.

**DO_ABOUT**
If set to 0 never do an About special element; if set to 1 always do an About special element; default 0.

**EXTERNAL_CROSSREF_SPLIT**
For cross-references to other manuals, this determines if the other manual is considered to be split or monolithic. By default, it is set based on the value of SPLIT. See Section 21.6 [HTML Xref], page 207, and see Section 21.6.6 [HTML Xref Configuration], page 212.

**EXTERNAL_DIR**
Base directory for external manuals; default none. It is better to use the general external cross-reference mechanism (see Section 21.6.6 [HTML Xref Configuration], page 212) than this variable.

**EXTERNAL_CROSSREF_EXTENSION**
File extension for cross-references to other manuals. If unset, based on EXTENSION.

**EXTRA_HEAD**
Additional text appearing within `<head>`; default unset.

**FOOTNOTE_END_HEADER_LEVEL**
Header formatting level used for the footnotes header with the ‘end’ footnotestyle; default ‘4’. See Section 9.3.2 [Footnote Styles], page 88.

**FOOTNOTE_SEPARATE_HEADER_LEVEL**
Header formatting level used for the footnotes header with the ‘separate’ footnotestyle; default ‘4’. See Section 9.3.2 [Footnote Styles], page 88.
FRAMES If set, a file describing the frame layout is generated, together with a file with the short table of contents; default false.

FRAMESET_DOCTYPE Same as DOCTYPE, but for the file containing the frame description.

HEADER_IN_TABLE Use tables for header formatting rather than a simple <div> element; default false.

HTML_MATH Method to use to render @math. This can be unset, set to ‘mathjax’ (see Section 19.4.4 [MathJax Customization Variables], page 180), set to ‘12h’, which uses latex2html (see Section 19.4.5 [latex2html Customization Variables], page 181), or set to ‘t4h’, which uses tex4ht (see Section 19.4.6 [tex4ht Customization Variables], page 181). In the default case, setting HTML_MATH also sets CONVERT_TO_LATEX_IN_MATH.

HTML_ROOT_ELEMENT_ATTRIBUTES Use that string for the <html> HTML document root element. Default undefined.

HTMLXREF_FILE Set the file name used for cross-references to other manuals. If not defined, htmlxref.cnf is used (see Section 21.6.6 [HTML Xref Configuration], page 212). Not defined in the default case. If TEST is set, HTMLXREF_MODE is set to the default and HTMLXREF_FILE is not defined, information on cross-references to other manuals is not used.

If HTMLXREF_MODE is set to ‘file’ the file name is directly used as the source of information, otherwise the file name is searched for in directories, and all the files found are used (see Section 21.6.6 [HTML Xref Configuration], page 212).

HTMLXREF_MODE How cross-references to other manuals information is determined. If set to ‘none’, no information is used. If set to ‘file’, the information is determined from a file path, htmlxref.cnf in the default case, or the value of HTMLXREF_FILE. If not defined (the default) or set to any other value, search in directories and use all the files (see Section 21.6.6 [HTML Xref Configuration], page 212).

ICONS Use icons for the navigation panel; default false.

IMAGE_LINK_PREFIX If set, the associated value is prepended to the image file links; default unset.

INDEX_ENTRY_COLON Symbol used between the index entry and the associated node or section; default ‘:’.

INFO_JS_DIR (Experimental.) Add a JavaScript browsing interface to the manual. The value of the variable is the directory to place the code for this interface, so you would run the program as e.g. `texi2any --html -c INFO_JS_DIR=js manual.texi` to place files in a ‘js’ directory under the output. This provides some of the
functionality of the Info browsers in a web browser, such as keyboard navigation and index lookup. This only works with non-split HTML output.

The interface should provide an acceptable fallback in functionality if JavaScript or web browser features are not available. However, please be cautious when using this option, in case you do make your documentation harder to access for some of your users.

**IGNORE_REF_TO_TOP_NODE_UP**
- Ignore references to `TOP_NODE_UP`, the up node for the Top node.

**INLINE_CSS_STYLE**
- Put CSS directly in HTML elements rather than at the beginning of the output; default false.

**JS_WEBLABELS**
  **JS_WEBLABELS_FILE**
- Specify how to use a JavaScript license web labels page to give licensing information and source code for any JavaScript used in the HTML files for the manual. (See [https://www.gnu.org/licenses/javascript-labels.html](https://www.gnu.org/licenses/javascript-labels.html)).
  - With the value 'generate' (the default), generate a labels page at `JS_WEBLABELS_FILE`, and link to it in the HTML output files. Only do this if actually referencing JavaScript files (either with `HTML_MATH` set to 'mathjax', or when using the experimental JS browsing interface when `INFO_JS_DIR` is set).
  - With this setting, `JS_WEBLABELS_FILE` must be a relative file name.
  - With the value 'reference', link to the labels file given by `JS_WEBLABELS_FILE` in the output, and do not generate a labels file. This setting is useful if you separately maintain a single labels file for a larger website that includes your manual.
  - With 'omit', neither generate nor link to a labels file.

**MAX_HEADER_LEVEL**
- Maximum header formatting level used (higher header formatting level numbers correspond to lower sectioning levels); default '4'.

**MENU_ENTRY_COLON**
- Symbol used between the menu entry and the description; default ‘:’.

**MENU_SYMBOL**
- Symbol used in front of menu entries when node names are used for menu entries formatting; default is undefined and set to `&bull;` if `USE_NUMERIC_ENTITY` is not set, and to `&#8217;` if set.

**MONOLITHIC**
- Output only one file including the table of contents. Set by default, but only relevant when the output is not split.

**NO_CSS**
- Do not use CSS; default false. See Section 21.3 [HTML CSS], page 204.

**NO_CUSTOM_HTML_ATTRIBUTE**
- Do not output HTML with custom attributes in elements; default false.
NO_NUMBER FOOTNOTE_SYMBOL
Symbol used for footnotes if NUMBER FOOTNOTES is false. Default is *.

PRE_BODY_CLOSE
If set, the given text will appear at the footer of each HTML file; default unset.

PROGRAM_NAME_IN_ABOUT
Used when an About element is output. If set, output the program name and miscellaneous related information in About special element; default false.

PROGRAM_NAME_IN FOOTER
If set, output the program name and miscellaneous related information in the page footers; default false.

SECTION_NAME_IN_TITLE
If set, when output is split, use the argument of the chapter structuring command (e.g., @chapter or @section) in the <title> instead of the argument to @node.

SHORT_TOC_LINK_TO_TOC
If set, the cross-references in the Short table of contents links to the corresponding Table of Contents entries, if a Table of Contents is output; default true.

SHOW_BUILTIN_CSS_RULES
Output the built-in default CSS rules on the standard output and exit.

SHOW_TITLE
If set, output the title at the beginning of the document; default ‘undef’. If set to ‘undef’, setting NO_TOP_NODE_OUTPUT also sets SHOW_TITLE for HTML.

SIMPLE_MENU
If set, use a simple preformatted style for the menu, instead of breaking down the different parts of the menu; default false. See Section 3.8.4 [Menu Parts], page 35.

TOC_LINKS
If set, links from headings to toc entries are created; default false.

TOP_FILE
This file name may be used for the top-level file. The extension is set appropriately, if necessary. This is used to override the default, and is, in general, only taken into account when output is split, and for HTML.

TOP_NODE_FILE_TARGET
File name used for the Top node in cross-references; default is index.html.

TOP_NODE_UP_URL
A URL used for Top node up references; the default is undef, in that case no Top node Up reference is generated. For more about the Top node pointers, see Section 3.4 [First Node], page 30. For overriding the Up pointer name in case TOP_NODE_UP_URL is set and for other formats, see TOP_NODE_UP in Section 19.4.7 [Other Customization Variables], page 182.

USE_ACCESSKEY
Use accesskey in cross-references; default true.
Chapter 19: texi2any: The Translator for Texinfo

USE_ISO
Use entities for doubled single-quote characters (see Section 11.5 [Inserting Quotation Marks], page 104), and ‘---’ and ‘--’ (see Section 2.1 [Conventions], page 9); default true.

USE_LINKS
Generate <link> elements in the HTML <head> output; default true.

USE_NODE_DIRECTIONS
If true, use nodes to determine where next, up and prev link to in node headers. If false, use sections. If undefined, use USE_NODES value. Default is undefined. Note that this setting does not determine the link string only where the links points to, see Section 5.5 [xrefautomaticsectiontitle], page 45, for the link string customization. If nodes and sections are systematically associated, this customization has no practical effect.

USE_REL_REV
Use rel in cross-references; default true.

USE_TITLEPAGE_FOR_TITLE
Use the full @titlepage as the title, not a simple title string; default true. Only relevant if SHOW_TITLE is set.

USE_XML_SYNTAX
Use XML/XHTML compatible syntax.

VERTICAL_HEAD_NAVIGATION
If set, a vertical navigation panel is used; default false.

WORDS_IN_PAGE
When output is split by nodes, specifies the approximate minimum page length at which a navigation panel is placed at the bottom of a page. To avoid ever having the navigation buttons at the bottom of a page, set this to a sufficiently large number. The default is 300.

XREF_USE_FLOAT_LABEL
If set, for the float name in cross-references, use the float label instead of the type followed by the float number (see Section 9.1.1 [@float], page 83). The default is off.

XREF_USE_NODE_NAME_ARG
Only relevant for cross-reference commands with no cross reference name (second argument). If set to 1, use the node name (first) argument in cross-reference @-commands for the text displayed as the hyperlink. If set to 0, use the node name if USE_NODES is set, otherwise the section name. If set to ‘undef’, use the first argument in preformatted environments, otherwise use the node name or section name depending on USE_NODES. The default is ‘undef’.

19.4.4 MathJax Customization Variables
This table lists the customization variables which can be used when MathJax is being used, which will be the case when HTML_MATH is set to ‘mathjax’.
MATHJAX_SCRIPT
URL of the MathJaX component file (e.g. tex-svg.js) you are using. texi2any provides a default value for this variable, but you are encouraged to host this file yourself on your website so that you are not dependent on others’ hosting.

MATHJAX_SOURCE
A URL of the full source code in its preferred form for modification, or instructions for obtaining such source code, for the component file named by MATHJAX_SCRIPT. ‘Preferred form for modification’ means that this should not be in a ‘minified’ form. Used in the license labels page (see Section 19.4.3 [HTML Customization Variables], page 174, under JS_WEBLABELS).

Again, texi2any provides a default value for this variable, but you are encouraged to host the source code for MathJaX and its dependencies yourself. This is in order to make the source code available reliably, and to reduce you and your users’ dependence on others’ distribution systems.

19.4.5 latex2html Customization Variables
This table lists the customization variables which can be used when latex2html is being used to convert @math, @displaymath, @latex and @tex sections for HTML. These customization variables are relevant only if HTML_MATH is set to ‘l2h’.

To actually convert @tex sections, --iftex should be used, and to actually convert @latex sections, --iflatex should be used.

L2H_CLEAN
If set, the intermediate files generated in relation with latex2html are removed; default true.

L2H_FILE
If set, the given file is used as latex2html’s init file; default unset.

L2H_HTML_VERSION
The HTML version used in the latex2html call; default unset.

L2H_L2H
The program invoked as latex2html; default is latex2html.

L2H_SKIP
If set to a true value, the actual call to latex2html is skipped; previously generated content is reused instead. If set to 0, the cache is not used at all. If set to ‘undef’, the cache is used for as many \TeX fragments as possible and for any remaining the command is run. The default is ‘undef’.

L2H_TMP
Set the directory used for temporary files. None of the file name components in this directory name may start with ‘.’; otherwise, latex2html will fail (because of dvips). The default is the empty string, which means the current directory.

19.4.6 tex4ht Customization Variables
This table lists the customization variables which can be used when tex4ht is being used to convert @math, @displaymath, @tex and @latex sections for HTML. These customization variables are relevant only if HTML_MATH is set to ‘t4h’.

To actually convert @tex sections, --iftex should be used, and to actually convert @latex sections, --iflatex should be used.
Chapter 19: texi2any: The Translator for Texinfo

T4H_LATEX_CONVERSION
If set, the conversion type used for @latex sections. Possibilities are ‘latex’, ‘tex’ and ‘texi’. Set to ‘latex’ if not defined.

T4H_MATH_CONVERSION
If set, the conversion type used for @math and @displaymath. Possibilities are ‘latex’, ‘tex’ and ‘texi’. Set to ‘tex’ if not defined.

T4H_TEX_CONVERSION
If set, the conversion type used for @tex sections. Possibilities are ‘latex’, ‘tex’ and ‘texi’. Set to ‘tex’ if not defined.

19.4.7 Other Customization Variables
This table gives the remaining customization variables, which apply to multiple formats, or affect global behavior, or otherwise don’t fit into the categories of the previous sections.

ASCII_PUNCTUATION
For Info output. When set, use plain ASCII characters to represent quotation marks, rather than the directional quotation marks available in UTF-8. Also use ASCII for a few other constructs; for example, the copyright sign (@copyright{}, becoming ‘(C)’), and the bullet symbol (@bullet{}, becoming ‘*’).

CHECK_NORMAL_MENU_STRUCTURE
Warn if the nodes pointers (either explicitly or automatically set) are not consistent with the order of node menu entries.

CLOSE_QUOTE_SYMBOL
When a closing quote is needed, use this character; default &#8217; in DocBook. Undefined in the default case in HTML and set to &rsquo; if USE_NUMERIC_ENTITY is not set, to &#8217; if set, and to a quote character if --enable-encoding is set and the output encoding includes that character.

The default for Info is set the same as for OPEN_QUOTE_SYMBOL, except that the Unicode code is a closing quote (see below).

COMMAND_LINE_ENCODING
Encoding used to decode command-line arguments. Default is based on the locale encoding. This may affect file names inserted into output files or error messages printed by the program.

Note that some file and directory names from the command line may not be decoded immediately, and may not be decoded at all.

CPP_LINE_DIRECTIVES
Recognize #line directives in a “preprocessing” pass (see Section 16.6 [External Macro Processors], page 150); on by default.

DEBUG
If set, debugging output is generated; default is off (zero).

DOC_ENCODING_FOR_INPUT_FILE_NAME
If set, use the input Texinfo document encoding information for the encoding of input file names, such as file names specified as @include or @verbatiminclude arguments. If unset, use the locale encoding instead. Default is set, except on MS-Windows where the locale encoding is used by default.
Chapter 19: \texttt{texi2any}: The Translator for Texinfo

Note that this is for file names only; the default encoding or \texttt{@documentencoding} is always used for the encoding of file content (see Section 14.2 \texttt{@documentencoding}, page 131).

The \texttt{INPUT\_FILE\_NAME\_ENCODING} variable overrides this variable.

\textbf{DOC\_ENCODING\_FOR\_OUTPUT\_FILE\_NAME}

If set, use the input Texinfo document encoding information for the encoding of output file names, such as files specified with \texttt{--output}. If unset, use the locale encoding instead. Default is unset, so files names are encoded using the current locale.

Note that this is for file names only; \texttt{OUTPUT\_ENCODING\_NAME} is used for the encoding of file content.

The \texttt{OUTPUT\_FILE\_NAME\_ENCODING} variable overrides this variable.

\textbf{DOCTYPE}

For DocBook, HTML, XML. Specifies the \texttt{SystemLiteral}, the entity's system identifier. This is a URI which may be used to retrieve the entity, and identifies the canonical DTD for the document. The default value is different for each of HTML, DocBook and XML.

\textbf{DUMP\_TEXI}

For debugging. If set, no conversion is done, only parsing and macro expansion. If the option \texttt{--macro-expand} is set, the Texinfo source is also expanded to the corresponding file. Default false.

\textbf{DUMP\_TREE}

For debugging. If set, the tree constructed upon parsing a Texinfo document is output to standard error; default false.

\textbf{EPUB\_CREATE\_CONTAINER\_FILE}

If set to 0, do not generate the EPUB output file. Default is set to 1.

\textbf{EPUB\_KEEP\_CONTAINER\_FOLDER}

If set, keep the directory containing the directories and files needed for EPUB. The EPUB output file is a ZIP archive of this directory. Default is not defined. Set if not defined and \texttt{TEST} or \texttt{DEBUG} is set. See Section 21.5.1 [EPUB Output File and Directory], page 206.

\textbf{EXTENSION}

The extension added to the output file name. The default is different for each output format.

\textbf{FORMAT\_MENU}

If set to `menu`, output Texinfo menus. This is the default for Info. `sectiontoc` is the default setting for HTML, where instead of the contents of \texttt{@menu} blocks being output, a list of subordinate sections is output in each node. If set to `nomenu`, do not output menus.

This variable is set to `nomenu` when generating DocBook, or when \texttt{--no-headers} is specified.

\textbf{IGNORE\_SPACE\_AFTER\_BRACED\_COMMAND\_NAME}

If set, spaces are ignored after an \texttt{@}-command that takes braces. Default true, matching the \TeX behavior.
INDEX_SPECIAL_CHARS_WARNING
If set, warn about ‘:’ in index entry, as not all Info readers may be able to process
these. For Info and plaintext only. Default false, because parsing problems there
don’t prevent navigation; readers can still relatively easily find their way to the
node in question.

INFO_SPECIAL_CHARS_QUOTE
If set, whenever there are problematic characters for Info output in places such as
node names or menu items, surround the part of the construct where they appear
with quoting characters, as described in Appendix F [Info Format Specification],
page 278. Default is set for Info and unset for plaintext. See Section 3.3 [Node
Line Requirements], page 29.

INFO_SPECIAL_CHARS_WARNING
If set, warn about problematic constructs for Info output (such as the string
‘::’)) in node names, menu items, and cross-references. If not defined, set unless
INFO_SPECIAL_CHARS_QUOTE is set. Default is set for Info and not defined for
plaintext. Similar warnings in index entries are covered by INDEX_SPECIAL_
CHARS_WARNING.

INPUT_FILE_NAME_ENCODING
Encoding used for input file names. This variable overrides any encoding from
the document or current locale. Normally, you do not need to set this variable,
but it can be used if file names are in a certain character encoding on a filesystem.
An alternative is to set DOC_ENCODING_FOR_INPUT_FILE_NAME to ‘0’ to use the
locale encoding. See also OUTPUT_FILE_NAME_ENCODING.

LOCALE_ENCODING
Locale encoding obtained from the system. You should not need to explicitly
set this variable.

MAX_MACRO_CALL_NESTING
The maximal number of recursive calls of @-commands defined through @rmacro;
default 100000. The purpose of this variable is to avoid infinite recursions.

MESSAGE_ENCODING
Encoding used to encode messages output by texi2any. Default is based on
the locale encoding.

It is also used for command-line argument passed to commands called from
texi2any. For example, latex2html will be called from texi2any if HTML_MATH
is set to ‘l2h’.

NO_TOP_NODE_OUTPUT
If set do not output the Top node content. The Top node is still parsed, but
the content is discarded. Not set in the default case for HTML. Set in the
default case for EPUB. If SHOW_TITLE is ‘undef’, setting NO_TOP_NODE_OUTPUT
also sets SHOW_TITLE for HTML.

Setting NO_TOP_NODE_OUTPUT, which removes the Top node and adds a title page
corresponds more to the formatting of a book. Setting NO_TOP_NODE_OUTPUT
to false, with SHOW_TITLE remaining ‘undef’, and false, corresponds more to a
document setup for browsing, with a direct access to the information at the Top
node.

For DocBook, NO_TOP_NODE_OUTPUT is set to true. Setting NO_TOP_NODE_OUTPUT
to false causes the Top node content to be output. It is not recommended to
output the Top node in DocBook as the title and copying informations are
always output. This option is kept for DocBook for compatibility, as before
2022 the Top node was output in DocBook. It could be removed in the future.

NO_USE_SETFILENAME
If set, do not use @setfilename to set the document name; instead, base the
output document name only on the input file name. The default is false.

NODE_NAME_IN_INDEX
If set, use node names in index entries, otherwise prefer section names; default
true.

NODE_NAME_IN_MENU
If set, use node names in menu entries, otherwise prefer section names; default
true.

OPEN_QUOTE_SYMBOL
When an opening quote is needed, e.g., for ‘@samp’ output, use the specified
character; default &#8216; for DocBook. Undefined in the default case in HTML
and set to &lsquo; if USE_NUMERIC_ENTITY is not set, to &#8217; if set, and to
a quote character if --enable-encoding is set and the output encoding includes
that character.

For Info, the default depends on the enabled document encoding. If --disable-
encoding is set or the document encoding is not UTF-8, ‘’ is used. This
character usually appears as an undirected single quote on modern systems.
Otherwise, the Info output uses a Unicode left quote.

OUTPUT_ENCODING_NAME
Normalized encoding name used for output files. Should be a usable charset name
in HTML, typically one of the preferred IANA encoding names. By default, if an
input encoding is set (typically through @documentencoding), this information
is used to set the output encoding name, otherwise the output encoding is based
on the default encoding. See Section 14.2 [documentencoding], page 131.

OUTPUT_FILE_NAME_ENCODING
Encoding used for output file names. This variable overrides any encoding from
the document or current locale.

Normally, you do not need to set this variable, but it can be used if file names
should be created in a certain character encoding on a filesystem. See also
INPUT_FILE_NAME_ENCODING.
Chapter 19: texi2any: The Translator for Texinfo

PACKAGE
PACKAGE_VERSION
PACKAGE_AND_VERSION
PACKAGE_URL
PACKAGE_NAME

The implementation’s short package name, package version, package name and
version concatenated, package URL, and full package name, respectively. By
default, these variables are all set through Autoconf, Automake, and configure.

PREFIX

The output file prefix, which is prepended to some output file names. By
default it is set by @setfilename or from the input file (see [Setting the Output
File Name], page 13). How this value is used depends on the value of other
customization variables or command line options, such as whether the output is
split. The default is unset.

PROGRAM

Name of the program used. By default, it is set to the name of the program
launched, with a trailing ‘.pl’ removed.

SORT_ELEMENT_COUNT

If set, the name of a file to which a list of elements (nodes or sections, de-
pending on the output format) is dumped, sorted by the number of lines they
contain after removal of @-commands; default unset. This is used by the pro-
gram texi-elements-by-size in the util/ directory of the Texinfo source
distribution (see [texi-elements-by-size], page 241).

SORT_ELEMENT_COUNT_WORDS

When dumping the elements-by-size file (see preceding item), use word counts
instead of line counts; default false.

TEST

If set to true, some variables which are normally dynamically generated anew
for each run (date, program name, version) are set to fixed and given values.
This is useful to compare the output to a reference file, as is done for the tests.
The default is false.

TEXI2DVI

Name of the command used to produce PostScript, PDF, and DVI; default
‘texi2dvi’. See Section 19.3 [texi2any Printed Output], page 171.

TEXI2HTML

Generate HTML and try to be as compatible as possible with texi2html; default
false.

TEXINFO_DTD_VERSION

For XML. Version of the DTD used in the XML output preamble. The default
is set based on a variable in configure.ac.

TEXTCONTENT_COMMENT

For stripped text content output (i.e., when TEXINFO_OUTPUT_FORMAT is set to
textcontent). If set, also output comments. Default false.

TOP_NODE_UP

Up node for the Top node; default ‘(dir)’. This node name is supposed to be
already formatted for the output format. In HTML can be used in attribute, so
should not contain any element. Used for HTML output only if `TOP_NODE_UP_URL` is set to override the URL, see `TOP_NODE_UP_URL` in Section 19.4.3 [HTML Customization Variables], page 174.

**TREE_TRANSFORMATIONS**

The associated value is a comma separated list of transformations that can be applied to the Texinfo tree prior to outputting the result. If more than one is specified, the ordering is irrelevant; each is always applied at the necessary point during processing.

By default, the tree transformations ‘move_index_entries_after_items’ and ‘relate_index_entries_to_table_entries’ are executed for HTML and DocBook output. Here’s an example of updating the master menu in a document:

```
texi2any \
  -c TREE_TRANSFORMATIONS=regenerate_master_menu \
  -c TEXINFO_OUTPUT_FORMAT=plaintexinfo \n  mydoc.texi \n  -o /tmp/out
```

(Caveat: Since ‘plaintexinfo’ output expands Texinfo macros and conditionals, it’s necessary to remove any such differences before installing the updates in the original document. This may be remedied in a future release.)

The following transformations are currently supported (many are used in the pod2texi utility distributed with Texinfo; see Section 19.6 [Invoking pod2texi], page 189):

‘complete_tree_nodes_menus’

Add menu entries or whole menus for nodes associated with sections of any level, based on the sectioning tree.

‘complete_tree_nodes_missing_menu’

Add whole menus for nodes associated with sections without menu. The menus are based on the sectioning tree.

‘fill_gaps_in_sectioning’

Adds empty @unnumbered... sections in a tree to fill gaps in sectioning. For example, an @unnumberedsec will be inserted if a @chapter is followed by a @subsection.

‘insert_nodes_for_sectioning_commands’

Insert nodes for sectioning commands lacking a corresponding node.

‘move_index_entries_after_items’

In @enumerate and @itemize, move index entries appearing just before an @item to just after the @item. Comment lines between index entries are moved too. As mentioned, this is always done for HTML and DocBook output.

‘regenerate_master_menu’

Update the Top node master menu, either replacing the (first) @detailmenu in the Top node menu, or creating it at the end of the Top node menu.
relate_indexEntries_to_table_entries

In @table, @vtable and @ftable, reassociate the index entry information from an index @-command appearing right after an @item line with the first element of the @item. Remove the index @-command from the tree.

simple_menu

Mostly the same as SIMPLE_MENU: use a simple preformatted style for the menu. It differs from setting SIMPLE_MENU in that SIMPLE_MENU only has an effect in HTML output.

USE_NODES

Preferentially use nodes to decide where elements are separated. If set to false, preferentially use sectioning to decide where elements are separated. The default is true.

USE_NUMERIC_ENTITY

For HTML, XML and DocBook. If set, use numeric entities instead of named entities. By default, set to true for DocBook output.

USE_UP_NODE_FOR_ELEMENT_UP

Fill in up sectioning direction with node direction when there is no sectioning up direction. In practice this can only happen when there is no @top section. Not set by default.

USE_SETFILENAME_EXTENSION

Default is on for Info, off for other output. If set, use exactly what @setfilename gives for the output file name, including the extension. You should not need to explicitly set this variable.

USE_UNIDECODE

If set to false, do not use the Text::Unidecode Perl module to transliterate more characters; default true.

19.5 Internationalization of Document Strings

texi2any writes fixed strings into the output document at various places: cross-references, page footers, the help page, alternate text for images, and so on. The string chosen depends on the value of the documentlanguage at the time of the string being output (see Section 14.1 [@documentlanguage], page 130, for the Texinfo command interface).

The Gettext framework is used for those strings (see Gettext). The libintl-perl package is used as the gettext implementation; more specifically, the pure Perl implementation is used, so Texinfo can support consistent behavior across all platforms and installations, which would not otherwise be possible. libintl-perl is included in the Texinfo distribution and always installed, to ensure that it is available if needed. It is also possible to use the system gettext (the choice can be made at build-time).

The Gettext domain ‘texinfo_document’ is used for the strings. Translated strings are written as Texinfo, and may include @-commands. In translated strings, the varying parts of the string are not usually denoted by %s and the like, but by ‘{arg_name}’. (This convention is common for gettext in Perl and is fully supported in GNU Gettext; see
Section “Perl Format Strings” in GNU Gettext.) For example, in the following, ‘{section}’ will be replaced by the section name:

    see {section}

These Perl-style brace format strings are used for two reasons: first, changing the order of printf arguments is only available since Perl 5.8.0; second, and more importantly, the order of arguments is unpredictable, since @-command expansion may lead to different orders depending on the output format.

The expansion of a translation string is done like this:

1. First, the string is translated. The locale is documentlanguage.documentencoding.

   If the documentlanguage has the form ‘ll_CC’, that is tried first, and then just ‘ll’.

   To cope with the possibility of having multiple encodings, a special use of the us-ascii locale encoding is also possible. If the ‘ll’ locale in the current encoding does not exist, and the encoding is not us-ascii, then us-ascii is tried.

   The idea is that if there is a us-ascii encoding, it means that all the characters in the charset may be expressed as @-commands. For example, there is a fr.us-ascii locale that can accommodate any encoding, since all the Latin 1 characters have associated @-commands. On the other hand, Japanese has only a translation ja.utf-8, since there are no @-commands for Japanese characters.

   The us-ascii locales are not needed much now that UTF-8 is used for most documents. Note that accented characters are required to be expressed as @-commands in the us-ascii locales, which may be inconvenient for translators.

2. Next, the string is expanded as Texinfo, and converted. The arguments are substituted; for example, ‘arg_name’ is replaced by the corresponding actual argument.

   In the following example, ‘{date}’, ‘{program_homepage}’ and ‘{program}’ are the arguments of the string. Since they are used in @uref, their order is not predictable. ‘{date}’, ‘{program_homepage}’ and ‘{program}’ are substituted after the expansion:

   Generated on @emph{{date}} using
   @uref{{program_homepage}, @emph{{program}}).

   This approach is admittedly a bit complicated. Its usefulness is that it supports having translations available in different encodings for encodings which can be covered by @-commands, and also specifying how the formatting for some commands is done, independently of the output format—yet still be language-dependent. For example, the ‘@pxref’ translation string can be like this:

    see {node_file_href} section '{section}' in @cite{book}

   which allows for specifying a string independently of the output format, while nevertheless with rich formatting it may be translated appropriately in many languages.

19.6 Invoking pod2texi: Convert Pod to Texinfo

The pod2texi program translates Perl Pod documentation file(s) to Texinfo. There are two basic modes of operation: generating a standalone manual from each input Pod, or (if --base-level=1 or higher is given) generating Texinfo subfiles suitable for use with @include.
The *pod2texi* program may be useful outside of the rest of Texinfo; thus, the invocation of *pod2texi* is documented in the Pod language using the man page format to follow the convention used in Perl standalone programs, with a version on the web [http://www.gnu.org/software/texinfo/manual/pod2texi.html](http://www.gnu.org/software/texinfo/manual/pod2texi.html) and a version included below. The version included in the manual is also an example of *pod2texi* use, as it is converted from Pod using *pod2texi*.

### 19.6.1 pod2texi

**pod2texi NAME**

pod2texi - convert Pod to Texinfo

**pod2texi SYNOPSIS**

    pod2texi [OPTION]... POD...

**pod2texi DESCRIPTION**

Translate Pod file(s) to Texinfo. There are two basic modes of operation. First, by default, each Pod is translated to a standalone Texinfo manual.

Second, if **--base-level** is set higher than 0, each Pod is translated to a file suitable for @include, and one more file with a main menu and all the @include is generated.

**pod2texi OPTIONS**

- **--appendix-sections**
  
  Use appendix sectioning commands (@appendix, ...) instead of the default numbered sectioning Texinfo @-commands (@chapter, @section, ...).

- **--base-level=NUM|NAME**
  
  Sets the level of the head1 commands. It may be an integer or a Texinfo sectioning command (without the @): 1 corresponds to the @chapter/@unnumbered level, 2 to the @section level, and so on. The default is 0, meaning that head1 commands are still output as chapters, but the output is arranged as a standalone manual.

  If the level is not 0, the Pod file is rendered as a fragment of a Texinfo manual suitable for @include. In this case, each Pod file has an additional sectioning command covering the entire file, one level above the --base-level value. Therefore, to make each Pod file a chapter in a large manual, you should use section as the base level.

  For an example of making Texinfo out of the Perl documentation itself, see contrib/perldoc-all in the Texinfo source distribution.

- **--debug=NUM**
  
  Set debugging level to NUM.

- **--headings-as-sections**
  
  Use headings commands (@heading, ...) instead of the default numbered sectioning Texinfo @-commands (@chapter, @section, ...). The sectioning command covering the entire file output for each Pod file if --base-level is not 0 is a numbered command.
Chapter 19: texi2any: The Translator for Texinfo

- **help**
  Display help and exit.

- **menus**
  Output node menus. If there is a main manual, its Top node menu is always output, since a master menu is generated. Other nodes menus are not output in the default case.

- **output=NAME**
  Name for the first manual, or the main manual if there is a main manual. Default is to write to standard output.

- **no-section-nodes**
  Use anchors for sections instead of nodes.

- **no-fill-section-gaps**
  Do not fill sectioning gaps with empty @unnumbered files. Ordinarily, it’s good to keep the sectioning hierarchy intact.

- **preamble=STR**
  Insert STR as top boilerplate before menu and includes. If STR is set to -, read the top boilerplate from the standard input. The default top boilerplate is a minimal beginning for a Texinfo document.

- **setfilename=STR**
  Use STR in top boilerplate before menu and includes for @setfilename. No @setfilename is output in the default case.

- **subdir=NAME**
  If there is a main manual with include files (each corresponding to an input Pod file), then those include files are put in directory NAME.

- **unnumbered-sections**
  Use unnumberered sectioning commands (@unnumbered, ...) instead of the default numbered sectioning Texinfo @-commands (@chapter, @section, ...).

- **top=TOP**
  Name of the @top element for the main manual. May contain Texinfo code.

- **version**
  Display version information and exit.

**pod2texi SEE ALSO**


**pod2texi COPYRIGHT AND LICENSE**

Copyright 2012-2022 Free Software Foundation, Inc.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

There is NO WARRANTY, to the extent permitted by law.
19.7 texi2html: Ancestor of texi2any

Conceptually, the texi2html program is the parent of today’s texi2any program. texi2html was developed independently, originally by Lionel Cons in 1998; at the time, makeinfo could not generate HTML. Many other people contributed to texi2html over the years.

The present texi2any uses little of the actual code of texi2html, and has quite a different basic approach to the implementation (namely, parsing the Texinfo document into a tree), but still, there is a family resemblance.

By design, texi2any supports nearly all the features of texi2html in some way. However, we did not attempt to maintain strict compatibility, so no texi2html executable is installed by the Texinfo package. An approximation can be run with an invocation like this (available as util/texi2html in the Texinfo source):

texi2any --set-customization-variable TEXI2HTML=1 ...

but, to emphasize, this is not a drop-in replacement for the previous texi2html. Here are the biggest differences:

- Most blatantly, the command line options of texi2html are now customization variables, for the most part. A table of approximate equivalents is given below.
- The program-level customization API is very different in texi2any.
- Indices cannot be split.
- Translated strings cannot be customized; we hope to introduce this feature in texi2any in the future.

Aside from the last, we do not intend to reimplement these differences. Therefore, the route forward for authors is alter manuals and build processes as necessary to use the new features and methods of texi2any. The texi2html maintainers (one of whom is the principal author of texi2any) do not intend to make further releases.

Here is the table showing texi2html options and corresponding texi2any customization variables.

---toc-links TOC_LINKS
--short-ext EXTENSION set to ‘htm’
--prefix PREFIX
--def-table DEF_TABLE
--html-xref-prefix EXTERNAL_DIR
--l2h HTML_MATH set to ‘l2h’
--l2h-12h L2H_L2H
--l2h-skip L2H_SKIP
--l2h-tmp L2H_TMP
--l2h-file L2H_FILE
--l2h-clean L2H_CLEAN
--use-nodes USE_NODES
Finally, any texi2html users seeking more detailed information can check the draft file doc/texi2oldapi.texi in the Texinfo source repository. It consists mainly of very rough notes, but may still be useful to some.
# 20 Creating and Installing Info Files

This chapter gives some information on the Info output and describes how to install Info files. For the creation of Info files with `texi2any`, see Chapter 19 [Generic Translator `texi2any`], page 164, and with Emacs, Section D.6 [Info Formatting], page 257. See Section 1.3 [Info Files], page 4, for general information about the file format. See Appendix F [Info Format Specification], page 278, for a detailed technical specification of the Info format.

## 20.1 Installing an Info File

Info files are usually kept in the `info` directory. You can read Info files using the standalone Info program or the Info reader built into Emacs. (See Info, for an introduction to Info.)

### 20.1.1 The Directory File `dir`

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`. (You can find the location of this file within Emacs by typing `C-h i` to enter Info and then typing `C-x C-f` to see the location of the `info` directory.)

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.
  * Texinfo: (texinfo). With one source file, make either a printed manual using \TeX{} or an Info file.

...  

Each of these menu entries points to the ‘Top’ node of the Info file that is named in parentheses. (The menu entry does not need to specify the ‘Top’ node, since Info goes to the ‘Top’ node if no node name is mentioned. See Section 3.8.6 [Nodes in Other Info Files], page 36.)

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 line for the ‘Top’ node of the Emacs manual looks like this in Info:

```plaintext
File: emacs Node: Top, Up: (DIR), Next: Distrib
```

In this case, the `dir` file name is written in uppercase letters—it can be written in either upper- or lowercase. This is not true in general, it is a special case for `dir`.

See the `util/dir-example` file in the Texinfo distribution for a large sample `dir` file.

### 20.1.2 Listing a New Info File

To add a new Info file to your system, you must write a menu entry to add to the menu in the `dir` file in the `info` directory. For example, if you were adding documentation for GDB, you would write the following new entry:

* GDB: (gdb). The source-level C debugger.
Chapter 20: Creating and Installing Info Files

The first part of the menu entry is the menu entry name, followed by a colon. The second part is the name of the Info file, in parentheses, followed by a period. The third part is the description.

The name of an Info file often has a .info extension. Thus, the Info file for GDB might be called either gdb or gdb.info. The Info reader programs automatically try the file name both with and without .info; so it is better to avoid clutter and not to write '.info' explicitly in the menu entry. For example, the GDB menu entry should use just 'gdb' for the file name, not 'gdb.info'.

20.1.3 Info Files in Other Directories

If an Info file is not in the info directory, there are three ways to specify its location:

1. Write the file name in the dir file as the second part of the menu.
2. Specify the Info directory name in the INFOPATH environment variable in your .profile or .cshrc initialization file. (Only you and others who set this environment variable will be able to find Info files whose location is specified this way.)
3. If you are using Emacs, list the name of the file in a second dir file, in its directory; and then add the name of that directory to the Info-directory-list variable in your personal or site initialization file.

This variable tells Emacs where to look for dir files (the files must be named dir). Emacs merges the files named dir from each of the listed directories. (In Emacs version 18, you can set the Info-directory variable to the name of only one directory.)

For example, to reach a test file in the /home/bob/info directory, you could add an entry like this to the menu in the standard dir file:

```
```

In this case, the absolute file name of the info-test file is written as the second part of the menu entry.

If you don’t want to edit the system dir file, you can tell Info where to look by setting the INFOPATH environment variable in your shell startup file. This works with both the Emacs and standalone Info readers.

Emacs uses the INFOPATH environment variable to initialize the value of Emacs’s own Info-directory-list variable. The standalone Info reader merges any files named dir in any directory listed in the INFOPATH variable into a single menu presented to you in the node called '(dir)Top'.

However you set INFOPATH, if its last character is a colon (on MS-DOS/MS-Windows systems, use a semicolon instead), this is replaced by the default (compiled-in) path. This gives you a way to augment the default path with new directories without having to list all the standard places. For example (using sh syntax):

```
INFOPATH=/home/bob/info:
export INFOPATH
```

will search /home/bob/info first, then the standard directories. Leading or doubled colons are not treated specially.

---

1 On MS-DOS/MS-Windows systems, Info will try the .inf extension as well.
When you create your own dir file for use with Info-directory-list or INFOPATH, it's easiest to start by copying an existing dir file and replace all the text after the '* Menu:' with your desired entries. That way, the punctuation and special \texttt{CTRL-\_} characters that Info needs will be present.

As one final alternative, which works only with Emacs Info, you can change the Info-directory-list variable. For example:

\begin{verbatim}
(add-hook 'Info-mode-hook '(lambda ()
  (add-to-list 'Info-directory-list
    (expand-file-name "~/info"))))
\end{verbatim}

\section*{20.1.4 Installing Info Directory Files}

When you install an Info file onto your system, you can use the program \texttt{install-info} to update the Info directory file dir. Normally the makefile for the package runs \texttt{install-info}, just after copying the Info file into its proper installed location.

In order for the Info file to work with \texttt{install-info}, you include the commands \texttt{@dircategory} and \texttt{@direntry} in the Texinfo source file. Use \texttt{@direntry} to specify the menu entries to add to the Info directory file. Use \texttt{@dircategory} to specify a category for the manual, which determines which part of the Info directory to put it in. See Section 2.6 [Directory Category], page 16.

Here is how these commands are used in this manual:

\begin{verbatim}
@dircategory Texinfo documentation system
@direntry
  * Texinfo: (texinfo). The GNU documentation format.
  * install-info: (texinfo)Invoking install-info. ...
...
@end direntry
\end{verbatim}

Here's what this produces in the Info file:

\begin{verbatim}
INFO-DIR-SECTION Texinfo documentation system
START-INFO-DIR-ENTRY
  * Texinfo: (texinfo). The GNU documentation format.
  * install-info: (texinfo)Invoking install-info. ...
...
END-INFO-DIR-ENTRY
\end{verbatim}

The \texttt{install-info} program sees these lines in the Info file, and that is how it knows what to do.

Always use the \texttt{@direntry} and \texttt{@dircategory} commands near the beginning of the Texinfo input, before the first \texttt{@node} command. If you use them later on in the input, \texttt{install-info} will not notice them.

\texttt{install-info} will automatically reformat the description of the menu entries it is adding. As a matter of convention, the description of the main entry (above, 'The GNU documentation format') should start at column 32, starting at zero (as in \texttt{what-cursor-position} in Emacs). This will make it align with most others. Description for individual utilities best start in column 48, where possible. For more information about formatting see the \texttt{--calign}, \texttt{--align}, and \texttt{--max-width} options in Section 20.1.5 [Invoking \texttt{install-info}], page 197.
If you use `@dircategory` more than once in the Texinfo source, each usage specifies the ‘current’ category; any subsequent `@direntry` commands will add to that category.

Each ‘Invoking’ node for every program installed should have a corresponding `@direntry`. This lets users easily find the documentation for the different programs they can run, as with the traditional `man` system.

### 20.1.5 Invoking `install-info`

`install-info` inserts menu entries from an Info file into the top-level `dir` file in the Info system (see the previous sections for an explanation of how the `dir` file works). `install-info` also removes menu entries from the `dir` file. It’s most often run as part of software installation, or when constructing a `dir` file for all manuals on a system. Synopsis:

```
install-info [option...] [info-file [dir-file]]
```

If `info-file` or `dir-file` are not specified, the options (described below) that define them must be. There are no compile-time defaults, and standard input is never used. `install-info` can read only one Info file and write only one `dir` file per invocation.

If `dir-file` (however specified) does not exist, `install-info` creates it if possible (with no entries).

If any input file is compressed with `gzip` (see `Gzip`), `install-info` automatically uncompressed it for reading. And if `dir-file` is compressed, `install-info` also automatically leaves it compressed after writing any changes. If `dir-file` itself does not exist, `install-info` tries to open `dir-file.gz`, `dir-file.xz`, `dir-file.bz2`, `dir-file.lz`, and `dir-file.lzma`, in that order.

Options:

--add-once

Specifies that the entry or entries will only be put into a single section.

--align=column

Specifies the column that the second and subsequent lines of menu entry’s description will be formatted to begin at. The default for this option is ‘35’. It is used in conjunction with the ‘--max-width’ option. `column` starts counting at 1.

--append-new-sections

Instead of alphabetizing new sections, place them at the end of the DIR file.

--calign=column

Specifies the column that the first line of menu entry’s description will be formatted to begin at. The default for this option is ‘33’. It is used in conjunction with the ‘--max-width’ option. When the name of the menu entry exceeds this column, entry’s description will start on the following line. `column` starts counting at 1.

--delete

Delete the entries in `info-file` from `dir-file`. The file name in the entry in `dir-file` must be `info-file` (except for an optional ‘.info’ in either one). Don’t insert any new entries. Any empty sections that result from the removal are also removed.

--debug

Report what is being done.
--description=text
    Specify the explanatory portion of the menu entry. If you don’t specify a
description (either via ‘--entry’, ‘--item’ or this option), the description is
taken from the Info file itself.

--dir-file=name
    Specify file name of the Info directory file. This is equivalent to using the dir-file
argument.

--dry-run
    Same as ‘--test’.

--entry=text
    Insert text as an Info directory entry; text should have the form of an Info menu
item line plus zero or more extra lines starting with whitespace. If you specify
more than one entry, they are all added. If you don’t specify any entries, they
are determined from information in the Info file itself.

--help
    Display a usage message with basic usage and all available options, then exit
successfully.

--info-file=file
    Specify Info file to install in the directory. This is equivalent to using the info-file
argument.

--info-dir=dir
    Specify the directory where the directory file dir resides. Equivalent to
‘--dir-file=dir/dir’.

--infodir=dir
    Same as ‘--info-dir’.

--item=text
    Same as ‘--entry=text’. An Info directory entry is actually a menu item.

--keep-old
    Do not replace pre-existing menu entries. When ‘--remove’ is specified, this
option means that empty sections are not removed.

--max-width=column
    Specifies the column that the menu entry’s description will be word-wrapped at.
    column starts counting at 1.

--maxwidth=column
    Same as ‘--max-width’.

--menuentry=text
    Same as ‘--name’.

--name=text
    Specify the name portion of the menu entry. If the text does not start with an
    asterisk ‘*’, it is presumed to be the text after the ‘*’ and before the parentheses
    that specify the Info file. Otherwise text is taken verbatim, and is taken as
    defining the text up to and including the first period (a space is appended if
necessary). If you don’t specify the name (either via ‘--entry’, ‘--item’ or this option), it is taken from the Info file itself. If the Info does not contain the name, the basename of the Info file is used.

--no-indent
Suppress formatting of new entries into the dir file.

--quiet
--silent
Suppress warnings, etc., for silent operation.

--remove
Same as ‘--delete’.

--remove-exactly
Also like ‘--delete’, but only entries if the Info file name matches exactly; .info and/or .gz suffixes are not ignored.

--section=sec
Put this file’s entries in section sec of the directory. If you specify more than one section, all the entries are added in each of the sections. If you don’t specify any sections, they are determined from information in the Info file itself. If the Info file doesn’t specify a section, the menu entries are put into the Miscellaneous section.

--section regex sec
Same as ‘--regex=regex --section=sec --add-once’.
install-info tries to detect when this alternate syntax is used, but does not always guess correctly. Here is the heuristic that install-info uses:
1. If the second argument to --section starts with a hyphen, the original syntax is presumed.
2. If the second argument to --section is a file that can be opened, the original syntax is presumed.
3. Otherwise the alternate syntax is used.

When the heuristic fails because your section title starts with a hyphen, or it happens to be a file that can be opened, the syntax should be changed to ‘--regex=regex --section=sec --add-once’.

--regex=regex
Put this file’s entries into any section that matches regex. If more than one section matches, all of the entries are added in each of the sections. Specify regex using basic regular expression syntax, more or less as used with grep, for example.

--test
Suppress updating of the directory file.

--version
Display version information and exit successfully.

20.2 Tag Files and Split Files
Info files always contain a tag table, to be able to jump to nodes quickly. Info files can be nonsplit (also called unsplit) or split.
If the Info file contains less than about 300,000 characters the file should be nonsplit. In that case, the tag table should appear at the end of the Info file. If the Texinfo file contains more than about 300,000 characters, Texinfo processors split the large Info file into shorter indirect subfiles of about 300,000 characters each. With texi2any, splitting may be prevented by --no-split, and the default size of 300,000 characters may be modified with --split-size (see Section 19.1 [Invoking texi2any], page 164).

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 output file name, specified by the @setfilename command or the input file name. The shortened version of the original file continues to have the name specified by @setfilename or the input file name.

At one stage in writing this document, for example, the Info file was saved as the file test-texinfo and that file looked like this:

```
Info file: test-texinfo, -*-Text-*-
produced by texinfo-format-buffer
from file: new-texinfo-manual.texinfo

Indirect:
test-texinfo-1: 102
test-texinfo-2: 50422
test-texinfo-3: 101300

Tag table:
(Indirect)
Node: overview?104
Node: info file?1271
Node: printed manual?4853
Node: conventions?6855
...
```

(But test-texinfo had far more nodes than are shown here.) Each of the split-off, indirect files, test-texinfo-1, test-texinfo-2, and test-texinfo-3, is listed in this file after the line that says ‘Indirect:’. The tag table is listed after the line that says ‘Tag table:’.

In the list of indirect files, the number following the file name records the cumulative number of bytes in the preceding indirect files, not counting the file list itself, the tag table, or any permissions text in the first file. In the tag table, the number following the node name records the location of the beginning of the node, in bytes from the beginning of the (unsplit) output.

If you are using texinfo-format-buffer to create Info files, you may want to run the Info-validate command. (The texi2any command does such a good job on its own, you do not need Info-validate.) 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 with `texinfo-format-buffer` and how to validate the structure of the nodes, see Section D.10.4.1 [Using `Info-validate`], page 267.

### 20.3 Info Format FAQ

Here are some questions that have been frequently asked on the project mailing lists and elsewhere.

Why when I run `info foo` do I get the same output as `man foo`?
Check that the Info manuals are installed. Not all GNU/Linux distributions install all the Info manuals by default. This is regrettable, as often the Info manual provides more information than the provided man page.

Why not use HTML instead of Info?
Manuals are rarely written in the Info format itself, but are generated from Texinfo source by the `texi2any` program. `texi2any` can generate HTML as well as Info from Texinfo source. You can also access and download HTML manuals from the GNU website ([https://www.gnu.org/manual/manual.html](https://www.gnu.org/manual/manual.html)). Additionally, some GNU/Linux distributions provide packages for the installation of HTML manuals.

Info still has some advantages over HTML for locally installed documentation. The Info browsers support index lookup and support for links between locally installed manuals in multiple locations. It’s important to have documentation installed locally on your machine rather than relying on the Internet; this makes accessing documentation more reliable, and ensures it matches installed versions of packages. It’s hoped that support for browsing locally installed Texinfo documentation in some form of HTML can be improved in the future.

Why provide the command-line `info` program when the Emacs Info reader is better?
The Emacs Info reader can display images, and fully supports using a mouse. There are not many differences among the Info readers besides that. Command-line `info` can be configured to change the default key bindings, as well as change the color of cross-references and search results, and to enable mouse scrolling support. You should not need to be experienced with the Emacs editor to use `info`. (Some recommend another program called `pinfo`, but this lacks in important features like index lookup.)

Some prefer to be able to scroll through the entire manual at once (as happens with man pages), rather than being limited to a single ‘node’ of content at once. Of course, there is no accounting for taste, but a single, unstructured block of text becomes more awkward as a manual becomes longer and more complicated. (However, if you really want to, you can view an info manual all at once in the `less` pager with `info foo | less`.)

Why do I have ‘see’ appearing in front of all of my links?
By default, Emacs Info mode either changes the marker ‘*note’ for cross-references to ‘see’, or hides it completely. Command-line `info` does the same if `hide-note-references` is set. Unfortunately, there is no way to do this reliably, as both the `@pxref` and `@ref` commands in Texinfo output this marker in the Info output, and the ‘see’ text is only appropriate for `@pxref`.
Yes, but how do I get a plain link, with no extra markup?
You can’t. Info is a plain text format that is displayed mostly as-is in the viewers, and without the ‘*note’ text there would be nothing to mark text as a link. Additionally, in printed output there is no such thing as a plain link, as the page number of the target would have to be printed somewhere.
If you really want a plain link in HTML output without affecting other output formats, you could create a macro with conditional definitions for each output format.

Why do lines containing links appear mysteriously short?
This due to Emacs (or info with hide-note-references set to ‘0n’) hiding ‘*note’ strings, as mentioned above.

Can the Info format be extended to support fonts, colors or reflowable text?
Any extension would be incompatible with existing Info-viewing programs. Extra markup added to Info files would be displayed to the user, making files ugly and unreadable.
When Texinfo files are processed into Info files, information about which Texinfo commands were used to mark up text is lost. Moreover, it is not possible to reliably detect whether text can be reflowed (e.g. a paragraph of prose), or whether line breaks should be kept (e.g. a code sample, or poem).
Info’s core purpose is to display documentation on text terminals. If you want more, you are recommended to use the HTML output from texi2any instead.
21 Generating HTML

texi2any generates Info output by default, but given the --html option, it will generate HTML, for web browsers and other programs. This chapter gives some details on such HTML output.

texi2any has many user-definable customization variables with which you can influence the HTML output. See Section 19.4 [Customization Variables], page 172. You can also write so-called initialization files, or init files for short, to modify almost every aspect of HTML output formatting. Initialization files contain code and are loaded by --init-file (see Section 19.1 [Invoking texi2any], page 164).

Some initialization files are maintained with Texinfo and installed in the default case. For example, chm.pm produces the intermediate compressed HTML Help format files that can be subsequently converted to the CHM format. As another example, highlight_syntax.pm uses source-highlight for syntax highlighting of @example blocks using the first argument to specify the language of the code (see Section 7.4 [@example], page 67). See GNU Source-highlight.

The draft documentation of texi2any HTML output adaptation using customization files is in a separate manual. See GNU Texinfo texi2any Output Customization.

21.1 HTML Translation

The HTML generated by texi2any generates standard HTML output. The output is intentionally quite plain for maximum portability and accessibility.

You can customize the output via CSS (see Section 21.3 [HTML CSS], page 204) or other means (see Section 19.4 [Customization Variables], page 172). If you cannot accomplish a reasonable customization, feel free to report that.

**Navigation bar:** By default, a navigation bar is inserted at the start of each node, analogous to Info output. If the ‘--no-headers’ option is used, the navigation bar is only inserted at the beginning of split files. Header <link> elements in split output support Info-like navigation with browsers which implement this feature.

**Raw HTML:** texi2any will include segments of Texinfo source between @ifhtml and @end_ifhtml in the HTML output (but not any of the other conditionals, by default). Source between @html and @end_html is passed without change to the output (i.e., suppressing the normal escaping of input ‘<’, ‘>’ and ‘&’ characters which have special significance in HTML). See Section 15.1 [Conditional Commands], page 133.

**Standards:** It is intentionally not our goal, and not even always possible, to pass through every conceivable validation test without any diagnostics. Different validation tests have different goals, often about pedantic enforcement of some standard or another. Our overriding goal is to help users, not blindly comply with standards.

Please report output from an error-free run of texi2any which has practical browser or EPUB reader portability problems as a bug (see Section 1.1 [Reporting Bugs], page 2).

In practice, the HTML produced by texi2any is slowly adjusted over time towards the latest HTML standard, while also trying to keep compatibility with earlier produced HTML. We use transitional markup and try to be slow enough to give time for the diverse HTML readers to adjust (and for standards to reincorporate useful features that were dropped...).
21.2 HTML Splitting

When splitting output at nodes (which is the default), texi2any writes HTML output into (basically) one output file per Texinfo source @node.

Each output file name is the node name with spaces replaced by ‘-’s and special characters changed to ‘_’ followed by their code point in hex (see Section 21.6 [HTML Xref], page 207). This is to make it portable and easy to use as a file name. In the unusual case of two different nodes having the same name after this treatment, they are written consecutively to the same file, with HTML anchors so each can be referred to independently.

If texi2any is run on a system which does not distinguish case in file names, nodes which are the same except for case (e.g., ‘index’ and ‘Index’) will also be folded into the same output file with anchors. You can also pretend to be on a case insensitive filesystem by setting the customization variable CASE_INSENSITIVE_FILENAMES.

It is also possible to split at chapters or sections with --split (see Section 19.1 [Invoking texi2any], page 164). In that case, the file names are constructed after the name of the node associated with the relevant sectioning command. Also, unless --no-node-files is specified, a redirection file is output for every node in order to more reliably support cross-references to that manual (see Section 21.6 [HTML Xref], page 207).

When splitting, the HTML output files are written into a subdirectory. The subdirectory name is derived from the base name (that is, any extension is removed), with _html postpended. For example, HTML output for gcc.texi would be written into a subdirectory named ‘gcc_html/’. The subdirectory name is based on @setfilename or on the input file name (see [Setting the Output File Name], page 13).

In any case, the top-level output file within the directory is always named ‘index.html’.

Monolithic output (--no-split) is named according to @setfilename, if present (with any ‘.info’ extension replaced with ‘.html’), --output (the argument is used literally), or based on the input file name as a last resort (see [Setting the Output File Name], page 13).

21.3 HTML CSS

Cascading Style Sheets (CSS) is an Internet standard for influencing the display of HTML documents: see http://www.w3.org/Style/CSS/.

By default, some CSS code is included to better implement the appearance of some Texinfo environments. For example:

pre.display { font-family:inherit }

The above tells the web browser to use the same font as the main document inside ‘<pre>’ elements used for @display environments. By default, the HTML ‘<pre>’ command uses a monospaced font.

Please see the reference above for a full explanation of CSS.

You can influence the CSS in the HTML output with two texi2any options: --css-include=file and --css-ref=url.

The option --css-ref=url adds to each output HTML file a ‘<link>’ tag referencing a CSS at the given url. This allows using external style sheets.

The option --css-include=file includes the contents file in the HTML output, as you might expect. However, the details are somewhat tricky, as described in the following, to provide maximum flexibility.
The CSS file first line may be a ‘@charset’ directive. If present, this directive is used to determine the encoding of the CSS file. The line is not copied into the output.

The CSS file may begin with so-called ‘@import’ directives, which link to external CSS specifications for browsers to use when interpreting the document. Again, a full description is beyond our scope here, but we’ll describe how they work syntactically, so we can explain how they are handled.

There can be more than one ‘@import’, but they have to come first in the file, with only whitespace and comments interspersed, no normal definitions. Comments in CSS files are delimited by ‘/* ... */’, as in C. An ‘@import’ directive must be in one of these two forms:

```plaintext
@import url(http://example.org/foo.css);
@import "http://example.net/bar.css";
```

The crucial characters are the ‘@’ at the beginning and the semicolon terminating the directive. When reading the CSS file, any such ‘@’-directive is simply copied into the output, as follows:

- If `file` contains only normal CSS declarations, it is included after the default CSS, thus overriding it.
- If `file` begins with ‘@import’ specifications (see below), then the ‘import’-s are included first (they have to come first, according to the standard), and then the default CSS is included. If you need to override the default CSS from an ‘@import’, you can do so with the ‘! important’ CSS construct, as in:
  ```plaintext
  pre.example { font-size: inherit ! important }
  ```
- If `file` contains both ‘@import’ and inline CSS specifications, the ‘@import’-s are included first, then default CSS, and lastly the inline CSS from `file`.
- Any @-directive other than ‘@import’ and ‘@charset’ is treated as a CSS declaration, meaning the default CSS is included and then the rest of the file is prepended.

If the CSS file is malformed or erroneous, the output is unspecified. The meaning of the CSS file is not interpreted in any way; the special ‘@’ and ‘;’ characters are looked for the text is blindly copied into the output. Comments in the CSS file may or may not be included in the output.

In addition to the possibilities offered by CSS, `texi2any` has many user-definable customization variables with which you can influence the HTML output. See Section 19.4 [Customization Variables], page 172.

21.4 @documentdescription: Summary Text

When producing HTML output for a document, a ‘<meta>’ element is written in the ‘<head>’ to give some idea of the content of the document. By default, this `description` is the title of the document, taken from the `@settitle` command (see Section 2.5.3 [settitle], page 14). To change this, use the `@documentdescription` environment, as in:

```plaintext
@documentdescription
descriptive text.
@end documentdescription
```

This will produce the following output in the ‘<head>’ of the HTML:

```plaintext
<meta name=description content="descriptive text.">
```
21.5 Generating EPUB

EPUB is a format designed for reading electronic books on portable devices. texi2any generated EPUB 3.2 in 2022. An EPUB file is a ZIP archive container, holding informative files as well as the manual rendered in HTML.

The generation of the EPUB file depends on the Archive::Zip Perl module being installed. This dependency is checked at runtime. In the default case, trying to output EPUB without this dependency raises an error. However, if the EPUB output file is not generated, with the customization variable EPUB_CREATE_CONTAINER_FILE set to 0, it is not an error if Archive::Zip is not installed.

The texi2any tests related to EPUB generation do not require the installation of Archive::Zip, as they set EPUB_CREATE_CONTAINER_FILE to 0 and keep the directory containing the files and directories needed for the EPUB file by setting the EPUB_KEEP_CONTAINER_FOLDER customization variable to 1.

21.5.1 Container Directory and Output Files

A directory containing the files and directories needed for the EPUB format is created when outputting EPUB. The name of this container directory is derived from the base name of the input file (that is, any extension is removed), with _epub_package postpended. If an output directory is specified, with --output, or with the SUBDIR customization variable, the container directory is created in that directory instead of the current directory. At the beginning of a new run, the container directory and all its contents are removed. The container directory is also removed after the final EPUB file has been generated in the default case.

The HTML files produced from the Texinfo manual are output in subdirectories of the container directory. Image files referred to from the Texinfo manual, if found, are copied to subdirectories of the container directory.

The EPUB output file is a ZIP archive of the container directory. The file name is derived from the base name, with the .epub extension postpended. If an output file is specified, with --output, or with the OUTFILE customization function, this file name is used instead. The output EPUB file is never placed in the directory specified by --output or SUBDIR; only the container directory is placed there, as explained just above.

The EPUB output file is not generated if the customization variable EPUB_CREATE_CONTAINER_FILE is set to 0. The container directory is left after the final EPUB file has been generated if EPUB_KEEP_CONTAINER_FOLDER is set.

See Section 19.1 [Invoking texi2any], page 164.

21.5.2 EPUB Cross-References

The EPUB format does not support references from an EPUB file to another EPUB file. Therefore any references to “external” Texinfo manuals should resolve to an external URL. texi2any produces these links with HTML cross-reference configuration (see Section 21.6.6 [HTML Xref Configuration], page 212). Since the links in the resulting EPUB are incorrect if no information is found for the cross-references, texi2any issues a warning by default for missing cross-references information. If these warnings are unwanted, set CHECK_HTMLXREF to 0.
21.5.3 HTML Generated for EPUB

The HTML generated for EPUB is XHTML conformant, UTF-8 encoded, and formatted without the usual HTML navigation headers and footers. Most of these features are enabled with customization variables, such as `USE_XML_SYNTAX` or `OUTPUT_FILE_NAME_ENCODING`. Some features of printed output are used in EPUB. In particular, the Top node does not appear in the EPUB output, while a title page is generated. This is obtained by setting `NO_TOP_NODE_OUTPUT`.

The `OUTFILE` and `SUBDIR` customization variables values correspond initially to the EPUB directory container and/or the EPUB output file (see Section 21.5.1 [EPUB Output File and Directory], page 206). These customization variables values are undefined or reset to the locations in the container directory where the XHTML files are output for the HTML generation. It is mentioned here because resetting customization variables is unusual; however, the variables reset are used internally for the conversion, and should not interact with any customization set by the user.

See Section 19.4.3 [HTML Customization Variables], page 174.

21.6 HTML Cross-references

Cross-references between Texinfo manuals in HTML format become standard HTML `<a>` links. This section describes the algorithm used, so that Texinfo can cooperate with other programs, such as `texi2html`, by writing mutually compatible HTML files.

This algorithm may or may not be used for links within HTML output for a Texinfo file. Since no issues of compatibility arise in such cases, we do not need to specify this.

We try to support references to such “external” manuals in both monolithic and split forms. A monolithic (mono) manual is entirely contained in one file, and a split manual has a file for each node. (See Section 21.2 [HTML Splitting], page 204.)

The algorithm was primarily devised by Patrice Dumas in 2003–04.

21.6.1 HTML Cross-reference Link Basics

For our purposes, an HTML link consists of four components: a host name, a directory part, a file part, and a target part. We always assume the `http` protocol. For example:

```
http://host/dir/file.html#target
```

The information to construct a link comes from the node name and manual name in the cross-reference command in the Texinfo source (see Chapter 5 [Cross References], page 43), and from external information (see Section 21.6.6 [HTML Xref Configuration], page 212).

We now consider each part in turn.

The host is hardwired to be the local host. This could either be the literal string ‘localhost’, or, according to the rules for HTML links, the ‘http://localhost/’ could be omitted entirely.

The dir and file parts are more complicated, and depend on the relative split/mono nature of both the manual being processed and the manual that the cross-reference refers to. The underlying idea is that there is one directory for Texinfo manuals in HTML, and a given manual is either available as a monolithic file `manual.html`, or a split subdirectory `manual_html/*.html`. Here are the cases:
• If the present manual is split, and the referent manual is also split, the directory is 
  ‘./referent_html/’ and the file is the expanded node name (described later).
• If the present manual is split, and the referent manual is mono, the directory is ‘./’
  and the file is referent.html.
• If the present manual is mono, and the referent manual is split, the directory is referent_ 
  html/ and the file is the expanded node name.
• If the present manual is mono, and the referent manual is also mono, the directory is 
  ./ (or just the empty string), and the file is referent.html.

Another rule, that only holds for file names, is that base file names are truncated to 245 
characters, to allow for an extension to be appended and still comply with the 255-character 
limit which is common to many filesystems. Although technically this can be changed with 
the BASEFILENAME_LENGTH customization variable (see Section 19.4.7 [Other Customization 
Variables], page 182), doing so would make cross-manual references to such nodes invalid.

Any directory part in the file name argument of the source cross reference command is 
ignored. Thus, \texttt{@xref{,,,../foo}} and \texttt{@xref{,,,foo}} both use ‘foo’ as the manual name. 
This is because any such attempted hardwiring of the directory is very unlikely to be useful 
for all the output formats that use the manual name.

Finally, the \textit{target} part is always the expanded node name.

Whether the present manual is split or mono is determined by user option; \texttt{texi2any} 
defaults to split, with the \texttt{--no-split} option overriding this.

Whether the referent manual is split or mono, however, is another bit of the external 
information (see Section 21.6.6 [HTML Xref Configuration], page 212). By default, \texttt{texi2any} 
uses the same form of the referent manual as the present manual.

Thus, there can be a mismatch between the format of the referent manual that the 
generating software assumes, and the format it’s actually present in. See Section 21.6.5 
[HTML Xref Mismatch], page 211.

\subsection*{21.6.2 HTML Cross-reference Node Name Expansion}

As mentioned in the previous section, the key part of the HTML cross reference algorithm 
is the conversion of node names in the Texinfo source into strings suitable for XHTML 
identifiers and file names. The restrictions are similar for each: plain ASCII letters, numbers, 
and the ‘-’ and ‘_’ characters are all that can be used. (Although HTML anchors can contain 
most characters, XHTML is more restrictive.)

Cross-references in Texinfo can refer either to nodes, anchors (see Section 5.11 [\texttt{@anchor}], 
page 49) or float labels (see Section 9.1.1 [\texttt{@float}], page 83). However, anchors and float 
labels are treated identically to nodes in this context, so we’ll continue to say “node” names 
for simplicity.

A special exception: the Top node (see Section 2.10 [The Top Node], page 23) is always 
mapped to the file \texttt{index.html}, to match web server software. However, the HTML \textit{target} 
is ‘Top’. Thus (in the split case):

\texttt{@xref{Top,,, emacs, The GNU Emacs Manual}.} 
⇒ \texttt{<a href="./emacs_html/index.html#Top">}

1. The standard ASCII letters (a-z and A-Z) are not modified. All other characters may 
be changed as specified below.
2. The standard ASCII numbers (0-9) are not modified except when a number is the first character of the node name. In that case, see below.

3. Multiple consecutive space, tab and newline characters are transformed into just one space.

4. Leading and trailing spaces are removed.

5. After the above has been applied, each remaining space character is converted into a ‘-’ character.

6. Other ASCII 7-bit characters are transformed into ‘._00xx’, where xx is the ASCII character code in (lowercase) hexadecimal. This includes ‘.’, which is mapped to ‘._005f’.

7. If the node name does not begin with a letter, the literal string ‘g.t’ is prefixed to the result. (Due to the rules above, that string can never occur otherwise; it is an arbitrary choice, standing for “GNU Texinfo”.) This is necessary because XHTML requires that identifiers begin with a letter.

For example:

```
@node A node --- with _’%
⇒ A-node-_002d_002d_002d-with-_005f_0027_0025
```

Example translations of common characters:

- ‘_’ \⇒ ‘._005f’
- ‘-’ \⇒ ‘._002d’
- ‘A node’ \⇒ ‘A-node’

On case-folding computer systems, nodes differing only by case will be mapped to the same file. In particular, as mentioned above, Top always maps to the file index.html. Thus, on a case-folding system, Top and a node named ‘Index’ will both be written to index.html.

Fortunately, the targets serve to distinguish these cases, since HTML target names are always case-sensitive, independent of operating system.

### 21.6.3 HTML Cross-reference Command Expansion

Node names may contain @-commands (see Section 3.3 [Node Line Requirements], page 29). This section describes how they are handled.

First, comments are removed.

Next, any @value commands (see Section 15.5.1 [@set @value], page 137) and macro invocations (see Section 16.2 [Invoking Macros], page 144) are fully expanded.

Then, for the following commands, the command name and braces are removed, and the text of the argument is recursively transformed:

- @asis
- @b
- @cite
- @code
- @command
- @dfn
- @dmn
- @dotless
- @emph
- @env
- @file
- @i
- @indicateurl
- @kbd
- @key
- @samp
- @sansserif
- @sc
- @slanted
- @strong
- @sub
- @sup
- @t
- @U
- @var
- @verb
- @w

For @sc, any letters are capitalized.

In addition, the following commands are replaced by constant text, as shown below. If any of these commands have non-empty arguments, as in @TeX{bad}, it is an error, and the
result is unspecified. In this table, ‘(space)’ means a space character and ‘(nothing)’ means the empty string. The notation ‘U+hhhh’ means Unicode code point hhhh (in hex, as usual).

There are further transformations of many of these expansions to yield the final file or other target name, such as space characters to ‘-’, etc., according to the other rules.

@newline (space)
@space (space)
@tab (space)
@! ‘!’
@* (space)
@– (nothing)
@. ‘.’
@: (nothing)
@? ‘?’
@@ ‘@’
@{ ‘{’
@} ‘}’
@LaTeX ‘LaTeX’
@TeX ‘TeX’
@arrow U+2192
@bullet U+2022
@comma ‘,’
@copyright U+00A9
@dots U+2026
@enddots ‘…’
@equiv U+2261
@error ‘error--->’
@euro U+20AC
@exclamdown U+00A1
@expansion U+21A6
@geq U+2265
@leq U+2264
@minus U+2212
@ordf U+00AA
@ordm U+00BA
@point U+2605
@pounds U+00A3
@print U+22A3
@questiondown U+00BF
@registered U+00AE
@result U+21D2
@textdegree U+00B0
@tie (space)

Quotation mark @-commands (@quotedblright{} and the like), are likewise replaced by their Unicode values. Normal quotation characters (e.g., ASCII ‘ and ’) are not altered. See Section 11.5 [Inserting Quotation Marks], page 104.
Any \texttt{@acronym}, \texttt{@abbr}, \texttt{@email}, and \texttt{@image} commands are replaced by their first argument. (For these commands, all subsequent arguments are optional, and ignored here.) See Section 6.1.14 \texttt{@acronym}, page 61, and Section 6.1.16 \texttt{@email}, page 62, and Section 9.2 \texttt{[Images]}, page 85.

Accents are handled according to the next section.

Any other command is an error, and the result is unspecified.

\section{21.6.4 HTML Cross-reference 8-bit Character Expansion}

Usually, characters other than plain 7-bit ASCII are transformed into the corresponding Unicode code point(s) in Normalization Form C, which uses precomposed characters where available. (This is the normalization form recommended by the W3C and other bodies.) This holds when that code point is \texttt{0xffff} or less, as it almost always is.

These will then be further transformed by the rules above into the string `\_\_\texttt{hhhh}`, where \texttt{hhhh} is the code point in hex.

For example, combining this rule and the previous section:

\begin{verbatim}
@node @b{A} @TeX{} @u{B} @point{}@enddots{}
⇒ A-TeX-B\_0306--\_2605\_002e\_002e\_002e
\end{verbatim}

Notice: 1) \texttt{@enddots} expands to three periods which in turn expands to three `\_002e’’s; 2) \texttt{@u{B}} is a ‘B’ with a breve accent, which does not exist as a pre-accented Unicode character, therefore expands to ‘\_\_\texttt{B\_0306}’ (B with combining breve).

When the Unicode code point is above \texttt{0xffff}, the transformation is `\_\_\_\texttt{xxxxxx}', that is, two leading underscores followed by six hex digits. Since Unicode has declared that their highest code point is \texttt{0x10ffff}, this is sufficient. (We felt it was better to define this extra escape than to always use six hex digits, since the first two would nearly always be zeros.)

This method works fine if the node name consists mostly of ASCII characters and contains only few 8-bit ones. But if the document is written in a language whose script is not based on the Latin alphabet (for example, Ukrainian), it will create file names consisting almost entirely of `\_\_\texttt{xxxx}' notations, which is inconvenient and all but unreadable. To handle such cases, \texttt{texi2any} offers the \texttt{--transliterate-file-names} command line option. This option enables \texttt{transliteration} of node names into ASCII characters for the purposes of file name creation and referencing. The transliteration is based on phonetic principles, which makes the generated file names more easily understandable.

For the definition of Unicode Normalization Form C, see Unicode report UAX#15, \url{http://www.unicode.org/reports/tr15/}. Many related documents and implementations are available elsewhere on the web.

\section{21.6.5 HTML Cross-reference Mismatch}

As mentioned earlier (see Section 21.6.1 \texttt{[HTML Xref Link Basics]}, page 207), the generating software may need to guess whether a given manual being cross-referenced is available in split or monolithic form—and, inevitably, it might guess wrong. However, when the referent manual is generated, it is possible to handle at least some mismatches.

In the case where we assume the referent is split, but it is actually available in mono, the only recourse would be to generate a \texttt{manual.html/} subdirectory full of HTML files which redirect back to the monolithic \texttt{manual.html}. Since this is essentially the same as a split manual in the first place, it’s not very appealing.
On the other hand, in the case where we assume the referent is mono, but it is actually available in split, it is possible to use JavaScript to redirect from the putatively monolithic manual.html to the different manual_html/node.html files. Here’s an example:

```javascript
function redirect() {
    switch (location.hash) {
    case "#Node1":
        location.replace("manual_html/Node1.html#Node1"); break;
    case "#Node2":
        location.replace("manual_html/Node2.html#Node2"); break;
    ...
    default:;
    }
}
```

Then, in the `<body>` tag of manual.html:

```html
<body onLoad="redirect();">
```

Once again, this is something the software which generated the referent manual has to do in advance, it’s not something the software generating the cross-reference in the present manual can control.

### 21.6.6 HTML Cross-reference Configuration: htmlxref.cnf

texi2any reads a file named htmlxref.cnf to gather information for cross-references to other manuals in HTML output. It is looked for in the following directories:

- `./` (the current directory)
- `./.texinfo/` (under the current directory)
- `~/.texinfo/` (where `~` is the current user’s home directory)
- `sysconfdir/texinfo/` (where `sysconfdir` is the system configuration directory specified at compile-time, e.g., `/usr/local/etc`)
- `datadir/texinfo/` (likewise specified at compile time, e.g., `/usr/local/share`)

All files found are used, with earlier entries overriding later ones. The Texinfo distribution includes a default file which handles many GNU manuals; it is installed in the last of the above directories, i.e., `datadir/texinfo/htmlxref.cnf`.

The HTMLXREF_MODE customization variable can be set to modify how the files are found. For instance, if set to ‘none’, no external information is used. HTMLXREF_FILE sets the file name to something else than htmlxref.cnf. see Section 19.4.3 [HTML Customization Variables], page 174.

The file is line-oriented. Lines consisting only of whitespace are ignored. Comments are indicated with a ‘#’ at the beginning of a line, optionally preceded by whitespace. Since ‘#’ can occur in URLs (like almost any character), it does not otherwise start a comment.
Each non-blank non-comment line must be either a variable assignment or manual information.

A variable assignment line looks like this:

\[ \text{varname} = \text{varvalue} \]

Whitespace around the '=' is optional and ignored. The varname should consist of letters; case is significant. The varvalue is an arbitrary string, continuing to the end of the line. Variables are then referenced with '${\text{varname}}$'; variable references can occur in the varvalue.

A manual information line looks like this:

\[ \text{manual keyword urlprefix} \]

with manual the short identifier for a manual, keyword being one of: mono, node, section, chapter, and urlprefix described below. Variable references can occur only in the urlprefix. For example (used in the canonical htmlxref.cnf):

\[
\begin{align*}
G &= \text{http://www.gnu.org} \\
GS &= ${G}/software \\
\text{hello mono} &= ${GS}/hello/manual/hello.html \\
\text{hello chapter} &= ${GS}/hello/manual/html\_chapter/ \\
\text{hello section} &= ${GS}/hello/manual/html\_section/ \\
\text{hello node} &= ${GS}/hello/manual/html\_node/
\end{align*}
\]

If the keyword is mono, urlprefix gives the host, directory, and file name for manual as one monolithic file.

If the keyword is node, section, or chapter, urlprefix gives the host and directory for manual split into nodes, sections, or chapters, respectively.

When available, texi2any will use the “corresponding” value for cross-references between manuals. That is, when generating monolithic output (--no-split), the mono URL will be used, when generating output that is split by node, the node URL will be used, etc. However, if a manual is not available in that form, anything that is available can be used. Here is the search order for each style:

\[
\begin{align*}
\text{node} \Rightarrow \text{node, section, chapter, mono} \\
\text{section} \Rightarrow \text{section, chapter, node, mono} \\
\text{chapter} \Rightarrow \text{chapter, section, node, mono} \\
\text{mono} \Rightarrow \text{mono, chapter, section, node}
\end{align*}
\]

These section- and chapter-level cross-manual references can succeed only when the target manual was created using --node-files; this is the default for split output.

If you have additions or corrections to the htmlxref.cnf distributed with Texinfo, please email bug-texinfo@gnu.org as usual. You can get the latest version from http://ftpmirror.gnu.org/texinfo/htmlxref.cnf.
Appendix A  @-Command Details

Here are the details of @-commands: information about their syntax, a list of commands, and information about where commands can appear.

A.1  @-Command Syntax

Texinfo has the following types of @-command:

1. Brace commands
   These commands start with @ followed by a letter or a word, followed by an argument within braces. For example, the command @dfn indicates the introductory or defining use of a term; it is used as follows: ‘In Texinfo, @-commands are @dfn{mark-up} commands.’

2. Line commands
   These commands occupy an entire line. The line starts with @, followed by the name of the command (a word); for example, @center or @cindex. If no argument is needed, the word is followed by the end of the line. If there is an argument, it is separated from the command name by a space. Braces are not used.

3. Block commands
   These commands are written at the start of a line, with general text on following lines, terminated by a matching @end command on a line of its own. For example, @example, then the lines of a coding example, then @end example. Some of these block commands take arguments as line commands do; for example, @enumerate A opening an environment terminated by @end enumerate. Here ‘A’ is the argument.

4. Symbol insertion commands with no arguments
   These commands start with @ followed by a word followed by a left and right-brace. These commands insert special symbols in the document; they do not take arguments. Some examples: @dots{} ⇒ ‘...’, @equiv{} ⇒ ‘≡’, @TeX{} ⇒ ‘\LaTeX’, and @bullet{} ⇒ ‘•’.

5. Non-alphabetic commands
   The names of commands in all of the above categories consist of alphabetic characters, almost entirely in lower-case. Unlike those, the non-alphabetic commands consist of an @ followed by a punctuation mark or other character that is not part of the Latin alphabet. Non-alphabetic commands are almost always part of text within a paragraph. The non-alphabetic commands include @@, @{}, @., @SPACE, and most of the accent commands.

6. Miscellaneous commands
   There are a handful of commands that don’t fit into any of the above categories; for example, the obsolete command @refill, which is always used at the end of a paragraph immediately following the final period or other punctuation character. @refill takes no argument and does not require braces. Likewise, @tab used in a @multitable block does not take arguments, and is not followed by braces.
Thus, the alphabetic commands fall into classes that have different argument syntaxes. You cannot tell to which class a command belongs by the appearance of its name, but you can tell by the command’s meaning: if the command stands for a glyph, it is in class 4 and does not require an argument; if it makes sense to use the command among other text as part of a paragraph, the command is in class 1 and must be followed by an argument in braces. The non-alphabetic commands, such as @:, are exceptions to the rule; they do not need braces.

The purpose of having different syntax for commands is to make Texinfo files easier to read, and also to help the GNU Emacs paragraph and filling commands work properly.

A.2 @-Command List

Here is an alphabetical list of the @-commands in Texinfo. Square brackets, [ ], indicate optional arguments; an ellipsis, ‘...’, indicates repeated text.

@ \textit{whitespace}

An @ followed by a space, tab, or newline produces a normal, stretchable, interword space. See Section 11.3.1 [Multiple Spaces], page 100.

@! Produce an exclamation point that ends a sentence (usually after an end-of-sentence capital letter). See Section 11.3.3 [Ending a Sentence], page 101.

@" Generate an umlaut or acute accent, respectively, over the next character, as in ¨ o and ´ o. See Section 11.4 [Inserting Accents], page 103.

@\texttt{\&}}

@\texttt{\&ampchar{}}

Generate an ampersand. See Section 11.1.6 [Inserting an Ampersand], page 99.

@* Force a line break. See Section 12.2 [Line Breaks], page 113.

@,\{c\} Generate a cedilla accent under c, as in ç. See Section 11.4 [Inserting Accents], page 103.

@- Insert a discretionary hyphenation point. See Section 12.3 [@- @hyphenation], page 114.

@. Produce a period that ends a sentence (usually after an end-of-sentence capital letter). See Section 11.3.3 [Ending a Sentence], page 101.

@/ Produces no output, but allows a line break. See Section 12.2 [Line Breaks], page 113.

@: Tell printed output processors to refrain from inserting extra whitespace after an immediately preceding period, question mark, exclamation mark, or colon. See Section 11.3.2 [Not Ending a Sentence], page 100.

@= Generate a macron (bar) accent over the next character, as in ö. See Section 11.4 [Inserting Accents], page 103.

@? Produce a question mark that ends a sentence (usually after an end-of-sentence capital letter). See Section 11.3.3 [Ending a Sentence], page 101.
Appendix A: @-Command Details

@atchar{}
Insert an at sign, ‘@’. See Section 11.1.1 [Inserting an Atsign], page 97.

@backslashchar{}
Insert a backslash, ‘\’; @backslashchar{} works anywhere, while \ works only inside @math. See Section 11.1.4 [Inserting a Backslash], page 98, and Section 11.7 [Inserting Math], page 105.

@^ @`
Generate a circumflex (hat) or grave accent, respectively, over the next character, as in ô and è. See Section 11.4 [Inserting Accents], page 103.

@lbracechar{}
Insert a left brace, ‘{’. See Section 11.1.2 [Inserting Braces], page 97.

@rbracechar{}
Insert a right brace, ‘}’. See Section 11.1.2 [Inserting Braces], page 97.

@~
Generate a tilde accent over the next character, as in Ñ. See Section 11.4 [Inserting Accents], page 103.

@AA{} @aa{}
Generate the uppercase and lowercase Scandinavian A-ring letters, respectively: Á, á. See Section 11.4 [Inserting Accents], page 103.

@abbr{abbreviation}
Indicate a general abbreviation, such as ‘Comput.’. See Section 6.1.13 [@abbr], page 60.

@acronym{acronym}
Indicate an acronym in all capital letters, such as ‘NASA’. See Section 6.1.14 [@acronym], page 61.

@AE{} @ae{}
Generate the uppercase and lowercase AE ligatures, respectively: Æ, æ. See Section 11.4 [Inserting Accents], page 103.

@afivepaper
Change page dimensions for the A5 paper size. See Section E.7 [A4 Paper], page 275.

@afourlatex
@afourpaper
@afourwide
Change page dimensions for the A4 paper size. See Section E.7 [A4 Paper], page 275.

@alias new=existing
Make the command ‘@new’ a synonym for the existing command ‘@existing’. See Section 16.4 [@alias], page 149.
@allowcodebreaks true-false

Control breaking at ‘-’ and ‘_’ in printed output. See Section 12.4 [@allowcodebreaks], page 114.

@anchor{name}

Define name as the current location for use as a cross-reference target. See Section 5.11 [@anchor], page 49.

@appendix title

Begin an appendix. The title appears in the table of contents. See Section 4.4 [@unnumbered @appendix], page 39.

@appendixsec title

@appendixsection title

Begin an appendix section within an appendix. The section title appears in the table of contents. @appendixsection is a longer spelling of the @appendixsec command. See Section 4.7 [@unnumberedsec @appendixsec @heading], page 40.

@appendixsubsec title

Begin an appendix subsection. The title appears in the table of contents. See Section 4.9 [@unnumberedsubsec @appendixsubsec @subheading], page 40.

@appendixsubsubsec title

Begin an appendix subsubsection. The title appears in the table of contents. See Section 4.10 [@subsubsection], page 41.

@arrow{}

Generate a right arrow glyph: ‘→’. Used by default for @click. See Section 11.9.8 [Click Sequences], page 111.

@asis

Keep the argument as is. Used following @table, @ftable, and @vtable to print the table’s first column without highlighting (“as is”). See [@asis], page 79.

@author author

Set a manual author in the title page. See Section 2.8.2 [@title @subtitle @author], page 19. Set a quotation author in @quotation. See Section 7.2 [@quotation], page 66.

@b{text}

Set text in a bold font, if possible. See Section 6.2.3 [Fonts], page 63.

@bullet{}

Generate a large round dot, •, or the closest possible thing to one. Often used with @table. See Section 11.8.5 [@bullet], page 107.

@bsixpaper

Change page dimensions for the B6 paper size. See Section E.7 [A4 Paper], page 275.

@bye

Stop processing a file. The processors do not see anything in the input file following @bye. See Section 2.12 [Ending a File], page 26.

@c comment

Begin a comment in Texinfo. The rest of the line does not appear in any output. A synonym for @comment. DEL also starts a comment. See Section 2.2 [Comments], page 10.
\caption
Define the full caption for a \float. See Section 9.1.2 [\caption \shortcaption], page 84.
\cartouche
Highlight an example or quotation by drawing a box with rounded corners around it, if possible. Pair with \end\cartouche. See Section 7.14 [\cartouche], page 73.
\center line-of-text
Center the line of text following the command. See Section 2.8.3 [\titlefont \center \sp], page 20.
\centerchap line-of-text
Like \chapter, but centers the chapter title. See Section 4.3 [\chapter], page 38.
\chapheading title
Print an unnumbered chapter-like heading, but omit from the table of contents. See Section 4.5 [\majorheading \chapheading], page 39.
\chapter title
Begin a numbered chapter. The chapter title appears in the table of contents. See Section 4.3 [\chapter], page 38.
\cindex entry
Add entry to the index of concepts. See Section 10.4 [Defining the Entries of an Index], page 92.
\cite{reference}
Highlight the name of a book or other reference that has no companion Info file. See Section 5.14 [\cite], page 52.
\clear flag
Unset flag, preventing the Texinfo formatting commands from formatting text between subsequent pairs of \ifset flag and \end ifset commands, and preventing \value{flag} from expanding to the value to which flag is set. See Section 15.5 [\set \clear \value], page 137.
\click{}
Represent a single “click” in a GUI. Used within \clicksequence. See Section 11.9.8 [Click Sequences], page 111.
\clicksequence{action \click{} action}
Represent a sequence of clicks in a GUI. See Section 11.9.8 [Click Sequences], page 111.
\clickstyle @cmd
Execute @cmd for each \click; the default is @arrow. The usual following empty braces on @cmd are omitted. See Section 11.9.8 [Click Sequences], page 111.
\code{sample-code}
Indicate an expression, a syntactically complete token of a program, or a program name. See Section 6.1.2 [\code], page 55.
\codequotebacktick on-off
\codequotedirected on-off
Control output of ` and ' in code examples. See Section 11.2 [Inserting Quote Characters], page 99.
@comma{}  
Insert a comma ‘,’ character; only needed when a literal comma would be taken as an argument separator. See Section 11.1.3 [Inserting a Comma], page 98.

@command{command-name}  
Indicate a command name, such as ls. See Section 6.1.10 [@command], page 60.

@comment comment  
Begin a comment in Texinfo. The rest of the line does not appear in any output. A synonym for @c. See Section 2.2 [Comments], page 10.

@contents  
Print a complete table of contents or specify that a table of content should be output, for formats that may output a table of contents. See Section 2.9 [Generating a Table of Contents], page 22.

@copying  
Specify copyright holders and copying conditions for the document. Pair with @end copying. See Section 2.7.1 [@copying], page 17.

@copyright{}  
Generate the copyright symbol ©. See Section 11.8.2 [@copyright], page 107.

@defcodeindex index-name  
Define a new index and its indexing command. Print entries in an @code font. See Section 10.7 [Defining New Indices], page 95.

@defcv category class name  
@defcvx 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 Chapter 13 [Definition Commands], page 117.

@deffn category name arguments...  
@deffnx 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 Chapter 13 [Definition Commands], page 117.

@defindex index-name  
Define a new index and its indexing command. Print entries in a roman font. See Section 10.7 [Defining New Indices], page 95.

@definfoenclose newcmd, before, after  
Create a new command @newcmd for online formats that marks text by enclosing it in strings that precede and follow the text. See Section 16.5 [@definfoenclose], page 149.

@defivar class instance-variable-name  
@defivarx class instance-variable-name  
Format a description for an instance variable in object-oriented programming. The command is equivalent to ‘@defcv {Instance Variable} ...’. See Chapter 13 [Definition Commands], page 117.
Appendix A: @-Command Details

@defmac macroname arguments...
@defmacx macroname arguments...
Format a description for a macro; equivalent to ‘@deffn Macro ...’. See Chapter 13 [Definition Commands], page 117.

@defmethod class method-name arguments...
@defmethodx class method-name arguments...
Format a description for a method in object-oriented programming; equivalent to ‘@defop Method ...’. See Chapter 13 [Definition Commands], page 117.

@defop category class name arguments...
@defopx category class name arguments...
Format a description for an operation in object-oriented programming. @defop takes as arguments the name of the category of operation, the name of the operation’s class, the name of the operation, and its arguments, if any. See Chapter 13 [Definition Commands], page 117, and Section 13.6.6 [Abstract Objects], page 124.

@defopt option-name
@defoptx option-name
Format a description for a user option; equivalent to ‘@defvr {User Option} ...’. See Chapter 13 [Definition Commands], page 117.

@defspect special-form-name arguments...
@defspectx special-form-name arguments...
Format a description for a special form; equivalent to ‘@deffn {Special Form} ...’. See Chapter 13 [Definition Commands], page 117.

@deftp category name-of-type attributes...
@deftpx category name-of-type attributes...
Format a description for a data type; its arguments are the category, the name of the type (e.g., ‘int’), and then the names of attributes of objects of that type. See Chapter 13 [Definition Commands], page 117, and Section 13.6.5 [Data Types], page 124.

@deftypecv category class data-type name
@deftypecvx category class data-type name
Format a description for a typed class variable in object-oriented programming. See Chapter 13 [Definition Commands], page 117, and Section 13.6.6 [Abstract Objects], page 124.

@deftypefn category data-type name arguments...
@deftypefnx category 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 category of entity being described, the type, the name of the entity, and its arguments, if any. See Chapter 13 [Definition Commands], page 117.

@deftypefnnewline on-off
Specifies whether return types for @deftypefn and similar are printed on lines by themselves; default is off. See Section 13.6.2 [Functions in Typed Languages], page 120.
@deftypefun data-type function-name arguments...
@deftypefunx data-type function-name arguments...
Format a description for a function in a typed language. The command is equivalent to ‘@deftypefn Function ...’. See Chapter 13 [Definition Commands], page 117.

@deftypeivar class data-type variable-name
@deftypeivarx class data-type variable-name
Format a description for a typed instance variable in object-oriented programming. See Chapter 13 [Definition Commands], page 117, and Section 13.6.6 [Abstract Objects], page 124.

@deftypemethod class data-type method-name arguments...
@deftypemethodx class data-type method-name arguments...
Format a description for a typed method in object-oriented programming. See Chapter 13 [Definition Commands], page 117.

@deftypeop category class data-type name arguments...
@deftypeopx category class data-type name arguments...
Format a description for a typed operation in object-oriented programming. See Chapter 13 [Definition Commands], page 117, and Section 13.6.6 [Abstract Objects], page 124.

@deftypevar data-type variable-name
@deftypevarx data-type variable-name
Format a description for a variable in a typed language. The command is equivalent to ‘@deftypevr Variable ...’. See Chapter 13 [Definition Commands], page 117.

@deftypevr category data-type name
@deftypevrx category data-type name
Format a description for anything like a variable in a typed language—an entity that records a value. Takes as arguments the category of entity being described, the type, and the name of the entity. See Chapter 13 [Definition Commands], page 117.

@defun function-name arguments...
@defunx function-name arguments...
Format a description for a function; equivalent to ‘@deffn Function ...’. See Chapter 13 [Definition Commands], page 117.

@defvar variable-name
@defvarx variable-name
Format a description for a variable; equivalent to ‘@defvr Variable ...’. See Chapter 13 [Definition Commands], page 117.

@defvr category name
@defvrx 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 Chapter 13 [Definition Commands], page 117.
@detailmenu
Mark the (optional) detailed node listing in a master menu. See Section 2.10.1 [Master Menu Parts], page 24.

@dfn{term}
Indicate the introductory or defining use of a term. See Section 6.1.12 [@dfn], page 60.

@DH{ }
@dh{}
Generate the uppercase and lowercase Icelandic letter eth, respectively: Ð, ð. See Section 11.4 [Inserting Accents], page 103.

@dircategory dirpart
Specify a category for the manual. See Section 2.6 [Directory Category], page 16.

@direntry
Begin the Info directory menu entry for this file. Pair with @end direntry. See Section 20.1.4 [Installing Dir Entries], page 196.

@display
Begin a kind of example. Like @example (indent text, do not fill), but do not select a new font. Pair with @end display. See Section 7.7 [@display], page 69.

@displaymath
Format a block of math in “display” format. See Section 11.7 [Inserting Math], page 105.

@dmn{dimension}
Format a unit of measure, as in 12 pt. See Section 11.3.5 [@dmn], page 102.

@docbook
Enter DocBook completely. Pair with @end docbook. See Section 15.3 [Raw Formatter Commands], page 135.

@documentdescription
Set the document description text, included in the HTML output. Pair with @end documentdescription. See Section 21.4 [@documentdescription], page 205.

@documentencoding enc
Declare the input encoding to be enc. See Section 14.2 [@documentencoding], page 131.

@documentlanguage CC
Declare the document language as the two-character ISO-639 abbreviation CC. See Section 14.1 [@documentlanguage], page 130.

@dotaccent{c}
Generate a dot accent over the character c, as in ó. See Section 11.4 [Inserting Accents], page 103.

@dotless{i-or-j}
Generate dotless i (‘ı’) and dotless j (‘ȷ’). See Section 11.4 [Inserting Accents], page 103.

@dots{}
Generate an ellipsis, ‘...’. See Section 11.8.4 [@dots], page 107.

@email{address[, displayed-text]}
Indicate an electronic mail address. See Section 6.1.16 [@email], page 62.
@<emph{text}> Emphasize text. See Section 6.2 [Emphasizing Text], page 62.

@end environment Ends environment, as in ‘@end example’. See [@-commands], page 9.

@enddots{} Generate an end-of-sentence ellipsis, like this: . . . See Section 11.8.4 [@dots], page 107.

@enumerate [number-or-letter] Begin a numbered list, using @item for each entry. Optionally, start list with number-or-letter. Pair with @end enumerate. See Section 8.3 [@enumerate], page 77.

@env{environment-variable} Indicate an environment variable name, such as PATH. See Section 6.1.8 [@env], page 59.

@equiv{} Indicate to the reader the exact equivalence of two forms with a glyph: ‘≡’. See Section 11.9.6 [@equiv], page 110.

@error{} Indicate to the reader with a glyph that the following text is an error message: ‘error’. See Section 11.9.5 [@error], page 110.

@errmsg{msg} Report msg as an error to standard error, and exit unsuccessfully. Texinfo commands within msg are expanded to plain text. See Chapter 15 [Conditionals], page 133, and Section 16.6 [External Macro Processors], page 150.

@euro{} Generate the Euro currency sign. See Section 11.8.6 [@euro], page 107.

@evenfooting [left] @| [center] @| [right] Specify page footings resp. headings for even-numbered (left-hand) pages. See Section E.2.3 [How to Make Your Own Headings], page 271.

@everyfooting [left] @| [center] @| [right] Specify page footings resp. headings for every page. Not relevant to Info. See Section E.2.3 [How to Make Your Own Headings], page 271.

@example Begin an example. Indent text, do not fill, and select fixed-width font. Pair with @end example. @example accepts optional arguments, separated by commas. It is recommended to set the first argument to the language of the example code. See Section 7.4 [@example], page 67.

@exampleindent indent Indent example-like environments by indent number of spaces (perhaps 0). See Section E.5 [@exampleindent], page 275.

@exclamdown{} Generate an upside-down exclamation point. See Section 11.4 [Inserting Accents], page 103.
@exdent line-of-text
Remove any indentation a line might have. See Section 7.9 [@exdent], page 70.

@expansion{}
Indicate the result of a macro expansion to the reader with a special glyph: ‘↦’. See Section 11.9.3 [@expansion], page 109.

@file{filename}
Highlight the name of a file, buffer, node, directory, etc. See Section 6.1.9 [@file], page 59.

@finalout
Prevent TeX from printing large black warning rectangles beside over-wide lines. See Section 18.8 [Overfull hboxes], page 162.

@findex entry
Add entry to the index of functions. See Section 10.4 [Defining the Entries of an Index], page 92.

@firstparagraphindent word
Control indentation of the first paragraph after section headers according to word, one of ‘none’ or ‘insert’. See Section E.4 [@firstparagraphindent], page 274.

@float
Environment to define floating material. Pair with @end float. See Section 9.1 [Floats], page 83.

@flushleft
@flushright
Do not fill text; left (right) justify every line while leaving the right (left) end ragged. Leave font as is. Pair with @end flushleft (@end flushright). See Section 7.10 [@flushleft @flushright], page 70.

@fonttextsize 10-11
Change the size of the main body font in the printed output. See Section 6.2.3 [Fonts], page 63.

@footnote{text-of-footnote}
Enter a footnote, for a reference that documents or elucidates the primary text. Footnote text is printed at the bottom of the page in printed output. In other formats, footnote text can be output in the same node, in a separate node, or simply be marked as being footnote text. See Section 9.3 [Footnotes], page 87.

@footnotestyle style
Specify a footnote style, either ‘end’ for the end node style or ‘separate’ for the separate style. In the separate style, footnotes are put in a separate node or file. See Section 9.3.2 [Footnote Styles], page 88.

@format
Begin a kind of example. Like @display, but do not indent. Pair with @end format. See Section 7.4 [@example], page 67.

@frenchspacing on-off
Control spacing after punctuation. See Section 11.3.4 [@frenchspacing], page 102.
@ftable formatting-command

Begin a two-column table, using @item for each entry. Automatically enter each of the items in the first column into the index of functions. Pair with @end ftable. The same as @table, except for indexing. See Section 8.4.2 [@ftable @vtable], page 80.

@geq{}

Generate a greater-than-or-equal sign, ‘≥’. See Section 11.8.10 [@geq @leq], page 108.

@group

Disallow page breaks within following text. Pair with @end group. Ignored in Info. See Section 12.9 [@group], page 115.

@guillemetleft{}
@guillemetright{}
@guillemotleft{}
@guillemotright{}
@guilsinglleft{}
@guilsinglright{}

Double and single angle quotation marks: « » ⟨ ⟩. @guillemotleft and @guillemotright are synonyms for @guillemetleft and @guillemetright. See Section 11.5 [Inserting Quotation Marks], page 104.

@H{c}

Generate the long Hungarian umlaut accent over c, as in õ.

@hashchar{}

Insert a hash ‘#’ character; only needed when a literal hash would introduce #line directive. See Section 11.1.5 [Inserting a Hashsign], page 98, and Section 16.6 [External Macro Processors], page 150.

@heading title

Print an unnumbered section-like heading, but omit from the table of contents. See Section 4.7 [@unnumberedsec @appendixsec @heading], page 40.

@headings on-off-single-double

Turn page headings on or off, and/or specify single-sided or double-sided page headings for printing. See Section E.2.1 [@headings], page 270.

@headitem

Begin a heading row in a multitable. See Section 8.5.2 [Multitable Rows], page 81.

@headitemfont{text}

Set text in the font used for multitable heading rows; mostly useful in multitable templates. See Section 8.5.2 [Multitable Rows], page 81.

@html

Enter HTML completely. Pair with @end html. See Section 15.3 [Raw Formatter Commands], page 135.

@hyphenation{hyphenated words}

Explicitly define hyphenation points. See Section 12.3 [@- @hyphenation], page 114.

@i{text}

Set text in an italic font, when possible. See Section 6.2.3 [Fonts], page 63.
@ifclear txivar
If the Texinfo variable txivar is not set, format the following text. Pair with
@end ifclear. See Section 15.5 [@set @clear @value], page 137.

@ifcommanddefined txicmd
@ifcommandnotdefined txicmd
If the Texinfo code `@txicmd' is (not) defined, format the following text. Pair with
the corresponding @end ifcommand... See Section 15.6 [Testing for Texinfo
Commands], page 140.

@ifdocbook
@ifhtml
@ifinfo
@iflatex
@ifplaintext
@ifxml
Begin text that will appear only in the given output format. @ifinfo output
appears in both Info and (for historical compatibility) plain text output. Pair
with @end ifdocbook resp. @end ifhtml... See Chapter 15 [Conditionals],
page 133.

@ifnotdocbook
@ifnothtml
@ifnotlatex
@ifnotplaintext
@ifnottex
@ifnotxml
Begin text to be ignored in one output format but not the others. @ifnothtml
text is omitted from HTML output, etc. Pair with the corresponding @end
ifnotformat. See Chapter 15 [Conditionals], page 133.

@ifnotinfo
Begin text to appear in output other than Info and (for historical compatibility)
plain text. Pair with @end ifnotinfo. See Chapter 15 [Conditionals], page 133.

@ifset txivar
If the Texinfo variable txivar is set, format the following text. Pair with @end
ifset. See Section 15.5 [@set @clear @value], page 137.

@iftex
Begin text to appear only in the T\TeX output. Pair with @end iftex. See
Chapter 15 [Conditionals], page 133.

@ignore
Begin text that will not appear in any output. Pair with @end ignore. See
Section 2.2 [Comments and Ignored Text], page 10.

@image{filename, [width], [height], [alt], [ext]}
Include graphics image in external filename scaled to the given width and/or
height, using alt text and looking for `filename.ext' in HTML. See Section 9.2
[Images], page 85.

@include filename
Read the contents of Texinfo source file filename. See Chapter 17 [Include Files],
page 152.
Appendix A: @-Command Details

@indent  Insert paragraph indentation. See Section 7.13 [@indent], page 72.

@indentedblock
Indent a block of arbitrary text on the left. Pair with @end indentedblock. See Section 7.3 [@indentedblock], page 67.

@indicateurl{indicateurl}
Indicate text that is a uniform resource locator for the World Wide Web. See Section 6.1.15 [@indicateurl], page 62.

@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 5.12 [@inforef], page 50.

@inlinefmt{fmt, text}
Insert text only if the output format is fmt. See Section 15.4 [Inline Conditionals], page 136.

@inlinefmtifelse{fmt, text, else-text}
Insert text if the output format is fmt, else else-text.

@inlineifclear{var, text}
@inlineifset{var, text}
Insert text only if the Texinfo variable var is (not) set.

@inlineraw{fmt, raw-text}
Insert text as in a raw conditional, only if the output format is fmt.

\input macro-definitions-file
Use the specified macro definitions file. This command is used only in the first line of a Texinfo file to cause \TeX to make use of the texinfo macro definitions file. The \ in \input is used instead of an @ because \TeX does not recognize @ until after it has read the definitions file. See Section 2.5 [Texinfo File Header], page 12.

@insertcopying
Insert the text previously defined with the @copying environment. See Section 2.7.2 [@insertcopying], page 18.

@item  Indicate the beginning of a marked paragraph for @itemize and @enumerate; indicate the beginning of the text of a first column entry for @table, @ftable, and @vtable. See Chapter 8 [Lists and Tables], page 75.

@itemize mark-generating-character-or-command
Begin an unordered list: indented paragraphs with a mark, such as @bullet, inside the left margin at the beginning of each item. Pair with @end itemize. See Section 8.2 [@itemize], page 76.

@itemx  Like @item in @table, @ftable, and @vtable, but do not generate extra vertical space above the item text. Thus, when several items have the same description, use @item for the first and @itemx for the others. See Section 8.4.3 [@itemx], page 80.
\@kbd{keyboard-characters}
Indicate characters of input to be typed by users. See Section 6.1.3 [\@kbd], page 56.

\@kbdinputstyle style
Specify when @kbd should use a font distinct from @code according to style: code, distinct, example. See Section 6.1.3 [\@kbd], page 56.

\@key{key-name}
Indicate the name of a key on a keyboard. See Section 6.1.4 [\@key], page 57.

\@kindex entry
Add entry to the index of keys. See Section 10.4 [Defining the Entries of an Index], page 92.

\@L{}
\@l{}
Generate the uppercase and lowercase Polish suppressed-L letters, respectively: Ŝ,ª.

\@LaTeX{}
Generate the L\TeX logo. See Section 11.8.1 [\@TeX \@LaTeX], page 106.

\@latex
Enter L\TeX completely. Pair with \@end latex. See Section 15.3 [Raw Formatter Commands], page 135.

\@leq{}
Generate a less-than-or-equal sign, ‘\(\leq\)’. See Section 11.8.10 [\@geq \@leq], page 108.

\@lisp
Begin an example of Lisp code. Indent text, do not fill, and select fixed-width font. Pair with \@end lisp. See Section 7.6 [\@lisp], page 69.

\@listoffloats
Produce a table-of-contents-like listing of @floats. See Section 9.1.3 [\@listoffloats], page 85.

\@lowersections
Change subsequent chapters to sections, sections to subsections, and so on. See Section 4.12 [\@raisesections and \@lowersections], page 42.

\@macro macroname {params}
Define a new Texinfo command @macroname{params}. Pair with \@end macro. See Section 16.1 [Defining Macros], page 143.

\@majorheading title
Print an unnumbered chapter-like heading, but omit from the table of contents. This generates more vertical whitespace before the heading than the @chapheading command. See Section 4.5 [\@majorheading \@chapheading], page 39.

\@math{mathematical-expression}
Format a mathematical expression. See Section 11.7 [Inserting Math], page 105.

\@menu
Mark the beginning of a menu of nodes. No effect in a printed manual. Pair with \@end menu. See Section 3.8 [Menus], page 33.

\@microtype on-off
Turn microtype on or off. See Section E.9 [Microtypography], page 276.
Appendix A: @-Command Details

@minus{} Generate a minus sign, ‘−’. See Section 11.8.9 [@minus{}], page 108.

@multitable column-width-spec
Begin a multi-column table. Begin each row with @item or @headitem, and separate columns with @tab. Pair with @end multitable. See Section 8.5.1 [Multitable Column Widths], page 81.

@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 12.10 [@need], page 116.

@node name, [next], [previous], [up]
Begin a new node. Only the first argument is mandatory. See Section 3.1 [Writing a Node], page 27.

@noindent Prevent text from being indented as if it were a new paragraph. See Section 7.12 [@noindent], page 72.

@novalidate Suppress validation of node references and omit creation of auxiliary files with \TeX. Use before any sectioning or cross-reference commands. See [Pointer Validation], page 168.

@OE{} @oe{} Generate the uppercase and lowercase O-with-slash letters, respectively: Ø, ø. See Section 11.4 [Inserting Accents], page 103.

@ogonek{c} Generate an ogonek diacritic under the next character, as in à. See Section 11.4 [Inserting Accents], page 103.

@option{option-name}
Indicate a command-line option, such as -l or --help. See Section 6.1.11 [@option], page 60.

@ordf{} @ordm{} Generate the feminine and masculine Spanish ordinals, respectively: a, o. See Section 11.4 [Inserting Accents], page 103.

@page Start a new page in a printed manual. See Section 12.8 [@page], page 115.

@pagesize [width] [, height]
Change page dimensions. See [pagesize], page 276.

@paragraphindent indent
Indent paragraphs by indent number of spaces (perhaps 0); preserve source file indentation if indent is asis. See Section E.3 [@paragraphindent], page 274.
@part title
Begin a group of chapters or appendixes; included in the tables of contents. See Section 4.11 [@part], page 41.

@pindex entry
Add entry to the index of programs. See Section 10.4 [Defining the Entries of an Index], page 92.

@point{}
Indicate the position of point in a buffer to the reader with a glyph: ‘⋆’. See Section 11.9.7 [@point], page 110.

@pounds{}
Generate the pounds sterling currency sign, ‘£’. See Section 11.8.7 [@pounds], page 107.

@print{}
Indicate printed output to the reader with a glyph: ‘−|’. See Section 11.9.4 [@print], page 109.

@printindex index-name
Generate the index for index-name. See Section 10.5 [Printing Indices & Menus], page 93.

@pxref{node, [entry], [node-title], [info-file], [printed-manual]}
Make a reference to be used within parentheses. Starts with a lowercase ‘see’ in a printed manual. The first argument is mandatory, except for references to whole manuals. To refer to another manual as a whole, the printed-manual and/or the info-file are the only required arguments. See Section 5.10 [@pxref], page 49.

@questiondown{}
Generate an upside-down question mark. See Section 11.4 [Inserting Accents], page 103.

@quotation
Narrow the margins to indicate text that is quoted from another work. Takes optional argument specifying prefix text. Pair with @end quotation. See Section 7.2 [@quotation], page 66.

@quotedblleft{}
@quotedblright{}
@quoteleft{}
@quoteright{}
@quotedblbase{}
@quotesinglbase{}
Produce various quotation marks: “ ” ’ ” . . See Section 11.5 [Inserting Quotation Marks], page 104.

@r{text}
Set text in the regular roman font, if possible. See Section 6.2.3 [Fonts], page 63.

@raggedright
Fill text; left justify every line while leaving the right end ragged. Leave font as is. Pair with @end raggedright. See Section 7.11 [@raggedright], page 71.
@raisesections
Change subsequent sections to chapters, subsections to sections, and so on. See Section 4.12 [Raise/lower sections], page 42.

@ref{node, [entry], [node-title], [info-file], [printed-manual]}
Make a plain reference that does not start with any special text. Follow command with a punctuation mark. The first argument is mandatory, except for references to whole manuals. To refer to another manual as a whole, the printed-manual and/or the info-file are the only required arguments. See Section 5.9 [@ref], page 48.

@registeredsymbol{}
Generate the legal symbol ®. See Section 11.8.3 [@registeredsymbol], page 107.

@result{}
Indicate the result of an expression to the reader with a special glyph: ‘⇒’. See Section 11.9.2 [@result], page 108.

@ringaccent{c}
Generate a ring accent over the next character, as in ó. See Section 11.4 [Inserting Accents], page 103.

@samp{text}
Indicate a literal example of a sequence of characters, in general. See Section 6.1.5 [@samp], page 57.

@sansserif{text}
Set text in a sans serif font if possible. See Section 6.2.3 [Fonts], page 63.

@sc{text}
Set text in a small caps font if possible, and uppercase in Info. See Section 6.2.2 [Smallcaps], page 63.

@section title
Begin a section within a chapter. The section title appears in the table of contents. Within @chapter and @appendix, the section title is numbered; within @unnumbered, the section is unnumbered. See Section 4.6 [@section], page 39.

@seealso{index-text}
Use in an index entry to refer the reader to another relevant index entry. See Section 10.3 [Advanced Indexing], page 92.

@seeentry{index-text}
Use in an index entry to redirect the reader to another index entry. See Section 10.3 [Advanced Indexing], page 92.

@set txivar [string]
Define the Texinfo variable txivar, optionally to the value string. See Section 15.5 [@set cleartexivar], page 137.

@setchapternewpage on-off-odd
Specify whether chapters start on new pages, and if so, whether on odd-numbered (right-hand) new pages. See Section E.1 [@setchapternewpage], page 269.
@setfilename info-file-name
Provide a name to be used for the output files. This command is ignored for \TeX\ formatting. See Section 2.5.2 [setfilename], page 13.

@settitle title
Specify the title for page headers in a printed manual, and the default document title for HTML. See Section 2.5.3 [settitle], page 14.

@shortcaption
Define the short caption for a @float. See Section 9.1.2 [caption \shortcaption], page 84.

@shortcontents
Print a short table of contents, with chapter-level entries only, or specify that a short table of contents should be output. For formats that may output a short table of contents. See Section 2.9 [Generating a Table of Contents], page 22.

@shorttitlepage title
Generate a minimal title page. See Section 2.8.1 [titlepage], page 19.

@slanted{text}
Set text in a slanted font if possible. See Section 6.2.3 [Fonts], page 63.

@smallbook
In printed output, use a 7 by 9.25 inch format rather than the regular 8.5 by 11 inch format. See Section E.6 [smallbook], page 275.

@smalldisplay
Begin a kind of example. Like @display, but use a smaller font size where possible. Pair with @end smalldisplay. See Section 7.15 [small], page 73.

@smallexample
Begin an example. Like @example, but use a smaller font size where possible. Pair with @end smallexample. See Section 7.15 [small], page 73.

@smallformat
Begin a kind of example. Like @format, but use a smaller font size where possible. Pair with @end smallformat. See Section 7.15 [small], page 73.

@smallindentedblock
Like @indentedblock, but use a smaller font size where possible. Pair with @end smallindentedblock. See Section 7.15 [small], page 73.

@smalllisp
Begin an example of Lisp code. Same as @smallexample. Pair with @end smalllisp. See Section 7.15 [small], page 73.

@smallquotation
Like @quotation, but use a smaller font size where possible. Pair with @end smallquotation. See Section 7.15 [small], page 73.

@sortas {key}
Used in the arguments to index commands to give a string by which the index entry should be sorted. See Section 10.2 [Indexing Commands], page 91.
@sp \text{n} \quad \text{Skip \text{n} blank lines. See Section 12.7 [@sp], page 115.}

@ss{} \quad \text{Generate the German sharp-S es-zet letter, \ss{}. See Section 11.4 [Inserting Accents], page 103.}

@strong \{text\} \quad \text{Emphasize \text{text} more strongly than @emph. See [Emphasizing Text], page 62.}

@sub \{text\} \quad \text{Set \text{text} as a subscript. See Section 11.6 [Inserting Subscripts and Superscripts], page 105.}

@subentry \quad \text{Use in an index entry to separate parts of a multi-level entry. See Section 10.3 [Advanced Indexing], page 92.}

@subheading \text{title} \quad \text{Print an unnumbered subsection-like heading, but omit from the table of contents of a printed manual. See Section 4.9 [@unnumberedsubsec @appendixsubsec @subheading], page 40.}

@subsection \text{title} \quad \text{Begin a subsection within a section. The subsection title appears in the table of contents. Same context-dependent numbering as @section. See Section 4.8 [@subsection], page 40.}

@subsubsection \text{title} \quad \text{Print an unnumbered subsubsection-like heading, but omit from the table of contents of a printed manual. See Section 4.10 [@subsubsection], page 41.}

@subsubsection \text{title} \quad \text{Begin a subsubsection within a subsection. The subsubsection title appears in the table of contents. Same context-dependent numbering as @section. See Section 4.10 [@subsubsection], page 41.}

@subtitle \text{title} \quad \text{Set a subtitle for the title page. See Section 2.8.2 [@title @subtitle @author], page 19.}

@summarycontents \quad \text{Print or specify a short table of contents. Synonym for @shortcontents. See Section 2.9 [Generating a Table of Contents], page 22.}

@sup \{text\} \quad \text{Set \text{text} as a superscript. See Section 11.6 [Inserting Subscripts and Superscripts], page 105.}

@syncodeindex \text{from-index to-index} \quad \text{Merge the index named in the first argument into the index named in the second argument, formatting the entries from the first index with @code. See Section 10.6 [Combining Indices], page 94.}

@synindex \text{from-index to-index} \quad \text{Merge the index named in the first argument into the index named in the second argument. Do not change the font of \text{from-index} entries. See Section 10.6 [Combining Indices], page 94.}
\t{text}  Set text in a fixed-width, typewriter-like font, if possible. See Section 6.2.3 [Fonts], page 63.

@tab  Separate columns in a row of a multitable. See Section 8.5.2 [Multitable Rows], page 81.

@table formatting-command  Begin a two-column table (description list), using @item for each entry. Write each first column entry on the same line as @item. First column entries are printed in the font resulting from formatting-command. Pair with @end table. See Section 8.4 [Making a Two-column Table], page 78. Also see Section 8.4.2 [@ftable @vtable], page 80, and Section 8.4.3 [@itemx], page 80.

@TeX{}  Generate the \TeX\ logo. See Section 11.8.1 [@TeX @LaTeX], page 106.

@tex  Enter \TeX completely. Pair with @end tex. See Section 15.3 [Raw Formatter Commands], page 135.

@textdegree{}  Generate the degree symbol. See Section 11.8.8 [@textdegree], page 107.

@thischapter  @thischaptername  @thischapternum  @thissection  @thissectionname  @thissectionnum  @thisfile  @thispage  @thistitle  

Only allowed in a heading or footing. Stands for, respectively, the number and name of the current chapter (in the format ‘Chapter 1: Title’), the current chapter name only, the current chapter number only, the number and name of the current section, the current section name only, the current section number only, the file name, the current page number, and the title of the document. See Section E.2.3 [How to Make Your Own Headings], page 271.

@TH{}  @th{}  Generate the uppercase and lowercase Icelandic letter thorn, respectively: Þ, þ. See Section 11.4 [Inserting Accents], page 103.

@tie{}  Generate a normal interword space at which a line break is not allowed. See Section 12.6 [@tie], page 115.

@tieaccent{cc}  Generate a tie-after accent over the next two characters cc, as in ‘ōō’. See Section 11.4 [Inserting Accents], page 103.

@tindex entry  Add entry to the index of data types. See Section 10.4 [Defining the Entries of an Index], page 92.
@title title
Set the title for the title page. See Section 2.8.2 [@title @subtitle @author], page 19.

@titlefont{text}
Print text in a larger than normal font, if possible. See Section 2.8.3 [@titlefont @center @sp], page 20.

@titlepage
Begin the title page. Write the command on a line of its own, paired with @end titlepage. The title page is not output, in the default case, in online formats. See Section 2.8.1 [@titlepage], page 19.

@today{}
Insert the current date, in ‘1 Jan 1900’ style. See Section E.2.3 [How to Make Your Own Headings], page 271.

@top title
Mark the topmost @node in the file, which must be defined on the line immediately preceding the @top command. The title is formatted as a chapter-level heading. In \TeX the @top command is merely a synonym for @unnumbered.

@U{hex}
Output a representation of Unicode character U+hex. See Section 11.10 [Inserting Unicode], page 112.

@u{c}
@ubaraccent{c}
@udotaccent{c}
Generate a breve, underbar, or underdot accent, respectively, over or under the character c, as in ö, ò, ő. See Section 11.4 [Inserting Accents], page 103.

@unmacro macroname
Undefine the macro @macroname if it has been defined. See Section 16.1 [Defining Macros], page 143.

@unnumbered title
Begin a chapter that appears without chapter numbers of any kind. The title appears in the table of contents. See Section 4.4 [@unnumbered @appendix], page 39.

@unnumberedsec title
Begin a section that appears without section numbers of any kind. The title appears in the table of contents. See Section 4.7 [@unnumberedsec @appendixsec @heading], page 40.

@unnumberedsubsec title
Begin an unnumbered subsection. The title appears in the table of contents. See Section 4.9 [@unnumberedsubsec @appendixsubsec @subheading], page 40.

@unnumberedsubsubsec title
Begin an unnumbered subsubsection. The title appears in the table of contents. See Section 4.10 [@subsubsection], page 41.

@uref{url[, displayed-text][, replacement]}
@url{url[, displayed-text][, replacement]}
Define a cross-reference to an external uniform resource locator, e.g., for the World Wide Web. See Section 5.13 [@url], page 50.
@urefbreakstyle style
Specify how @uref/@url should break at special characters: after, before, none. See Section 5.13 [@url], page 50.

@v{c}
Generate check accent over the character c, as in ô. See Section 11.4 [Inserting Accents], page 103.

@value{txivar}
Insert the value, if any, of the Texinfo variable txivar, previously defined by @set. See Section 15.5 [@set @clear @value], page 137.

@var{metasyntactic-variable}
Highlight a metasyntactic variable, which is something that stands for another piece of text. See Section 6.1.7 [@var], page 58.

@verb{delim literal delim}
Output literal, delimited by the single character delim, exactly as is (in the fixed-width font), including any whitespace or Texinfo special characters. See Section 6.1.6 [@verb], page 58.

@verbatim
Output the text of the environment exactly as is (in the fixed-width font). Pair with @end verbatim. See Section 7.5 [@verbatim], page 68.

@verbatiminclude filename
Output the contents of filename exactly as is (in the fixed-width font). See Section 17.3 [@verbatiminclude], page 153.

@vindex entry
Add entry to the index of variables. See Section 10.4 [Defining the Entries of an Index], page 92.

@vskip 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’.) See Section 2.8.4 [Copyright], page 21.

@vtable formatting-command
Begin a two-column table, using @item for each entry. Automatically enter each of the items in the first column into the index of variables. Pair with @end vtable. The same as @table, except for indexing. See Section 8.4.2 [@table @vtable], page 80.

@w{text}
Disallow line breaks within text. See Section 12.5 [@w], page 114.

@xml
Enter XML completely. Pair with @end xml. See Section 15.3 [Raw Formatter Commands], page 135.

@xref{node, [entry], [node-title], [info-file], [printed-manual]}
Make a reference that starts with ‘See’ in a printed manual. Follow command with a punctuation mark. The first argument is mandatory, except for references to whole manuals. To refer to another manual as a whole, the printed-manual and/or the info-file are the only required arguments. See Section 5.8 [@xref], page 48.
@xrefautomaticsectiontitle on-off

By default, use the section title instead of the node name in cross references, including in node headers in HTML. See Section 5.5 [Three Arguments], page 45.

A.3 @-Command Contexts

Here we describe approximately which @-commands can be used in which contexts. It is not exhaustive or meant to be a complete reference. Discrepancies between the information here and the Texinfo processors implementations are most likely to be resolved in favor of the implementations.

By general text below, we mean anything except sectioning and other such outer-level document commands, such as @section, @node, and @setfilename.

@c, @comment and @if ... @end if conditional commands may appear anywhere (except the conditionals must still be on lines by themselves). @caption and @shortcaption may only appear in @float but may contain general text. @footnote content likewise.

@-commands with braces marking text (such as @strong, @sc, @asis) may contain raw formatter commands such as @html but no other block commands (other commands terminated by @end) and may not be split across paragraphs, but may otherwise contain general text.

In addition to the block command restriction, on @center, @exdent and @item in @table lines, @-commands that makes only sense in a paragraph are not accepted, such as @indent.

In addition to the above, sectioning commands cannot contain @anchor, @footnote or @verb.

In addition to the above, remaining commands (@node, @anchor, @printindex, @ref, @math, @cindex, @url, @image, and so on) cannot contain cross-reference commands (@ref, @xref, @pxref and @inforef).

For precise and complete information, we suggest looking into the test suite in the sources, which exhaustively tries combinations.

A.4 Obsolete @-Commands

Here are Texinfo @-commands which are obsolete or have been removed completely. This section is for historical purposes.

@refill This command used to refill and indent the paragraph after all the other processing has been done. It is no longer needed, since all formatters now automatically refill as needed, but you may still see it in the source to some manuals, as it does no harm.

@setcontentsaftertitlepage

In the past, the contents commands were sometimes placed at the end of the file, after any indices and just before the @bye, but we no longer recommend this. This command could be used by a user printing a manual, to force the contents to be printed after the title page (after the ‘@end titlepage’ line) even if the @contents command was at the end of the manual.

@setshortcontentsaftertitlepage

This placed the short table of contents after the ‘@end titlepage’ command even if the @shortcontents command was at the end.
Appendix B  Tips and Hints

Here are some tips for writing Texinfo documentation:

- Write in the present tense, not in the past or the future.
- Write actively! For example, write “We recommend that . . .” rather than “It is recommended that . . .”.
- Use 70 or 72 as your fill column. Longer lines are hard to read.
- Include a copyright notice and copying permissions.
- Design your manual so that it can be read sequentially, as far as possible. People tire of flipping back and forth to find information that should be presented to them as they need it.

Index, Index, Index!

Write many index entries, in different ways. Readers like indices; they are helpful and convenient.

Although it is easiest to write index entries as you write the body of the text, some people prefer to write entries afterwards. In either case, write an entry before the paragraph to which it applies. This way, an index entry points to the first page of a paragraph that is split across pages.

Here are more index-related hints we have found valuable:

- Write each index entry differently, so each entry refers to a different place in the document.
- Write index entries only where a topic is discussed significantly. For example, it is not useful to index “debugging information” in a chapter on reporting bugs. Someone who wants to know about debugging information will certainly not find it in that chapter.
- Consistently capitalize the first word of every concept index entry, or else consistently use lowercase. Terse entries often call for lowercase; longer entries for capitalization. Whichever case convention you use, please use one or the other consistently! Mixing the two styles looks bad.
- Always capitalize or use uppercase for those words in an index for which this is proper, such as names of countries or acronyms. Always use the appropriate case for case-sensitive names, such as those in C or Lisp.
- Write the indexing commands that refer to a whole section immediately after the section command, and write the indexing commands that refer to a paragraph before that paragraph.

In the example that follows, a blank line comes after the index entry for “Leaping”:

@section The Dog and the Fox
@cindex Jumping, in general
@cindex Leaping

@cindex Dog, lazy, jumped over
@cindex Lazy dog jumped over
@cindex Fox, jumps over dog
@cindex Quick fox jumps over dog
The quick brown fox jumps over the lazy dog.
(Note that the example shows entries for the same concept that are written in different ways—‘Lazy dog’, and ‘Dog, lazy’—so readers can look up the concept in different ways.)

Blank Lines

- Insert a blank line between a sectioning command and the first following sentence or paragraph, or between the indexing commands associated with the sectioning command and the first following sentence or paragraph, as shown in the tip on indexing. It makes the source easier to read.
- Always insert a blank line before a @table command and after an @end table command; but never insert a blank line after an @table command.

For example,

```
Types of fox:

@table @samp
@item Quick
Jump over lazy dogs.

@item Brown
Also jump over lazy dogs.
@end table

@noindent
On the other hand, ...
```

Insert blank lines before and after @itemize ... @end itemize and @enumerate ... @end enumerate in the same way.

Complete Phrases

Complete phrases are easier to read than . . .

- Write entries in an itemized list as complete sentences; or at least, as complete phrases. Incomplete expressions . . . awkward . . . like this.
- Write the prefatory sentence or phrase for a multi-item list or table as a complete expression. Do not write “You can set;”; instead, write “You can set these variables;”. The former expression sounds cut off.

Editions, Dates and Versions

Include edition numbers, version numbers, and dates in the @copying text (for people reading the Texinfo file, and for the legal copyright in the output files). Then use @insertcopying in the @titlepage section for people reading the printed output (see [Short Sample], page 11).

It is easiest to handle such version information using @set and @value. See Section 15.5.4 [@value Example], page 139, and Section C.1 [GNU Sample Texts], page 243.

Definition Commands

Definition commands are @deffn, @defun, @defmac, and the like, and enable you to write descriptions in a uniform format.
Appendix B: Tips and Hints

- Write just one definition command for each entity you define with a definition command. The automatic indexing feature creates an index entry that leads the reader to the definition.
- Use @table ... @end table in an appendix that contains a summary of functions, not @deffn or other definition commands.

Capitalization
- Capitalize “Texinfo”; it is a name. Do not write the ‘x’ or ‘i’ in uppercase.
- Capitalize “Info”; it is a name.
- Write $\TeX$ using the @TeX{} command. Note the uppercase ‘T’ and ‘X’. This command causes the formatters to typeset the name according to the wishes of Donald Knuth, who wrote $\TeX$. (Likewise @LaTeX{} for $\LaTeX$.)

Spaces
Do not use spaces to format a Texinfo file, except inside of @example ... @end example and other literal environments and commands.

For example, \TeX fills the following:

@kbd{C-x v}
@kbd{M-x vc-next-action}

Perform the next logical operation
on the version-controlled file

so it looks like this:

\texttt{C-x v M-x vc-next-action} Perform the next logical operation on the version-controlled file corresponding to the current buffer.

In this case, the text should be formatted with @table, @item, and @itemx, to create a table.

@code, @samp, @var, and ‘---’
- Use @code around Lisp symbols, including command names. For example,

\texttt{The main function is \code{vc-next-action}, \ldots}

- Avoid putting letters such as ‘s’ immediately after an @code. Such letters look bad.
- Use @var around meta-variables. Do not write angle brackets around them.
- Use three hyphens in a row, ‘---’, to indicate a long dash. The Info formatter reduces three hyphens to two; a long dash is typeset in other output formats.

Periods Outside of Quotes
Place periods and other punctuation marks \textit{outside} of quotations, unless the punctuation is part of the quotation. This practice goes against some publishing conventions in the United States, but enables the reader to distinguish between the contents of the quotation and the whole passage.

For example, you should write the following sentence with the period outside the end quotation marks:

Evidently, ‘au’ is an abbreviation for ‘``author’’.
since ‘au’ does not serve as an abbreviation for ‘author.’ (with a period following the word).

**Introducing New Terms**

- Introduce new terms so that a reader who does not know them can understand them from context; or write a definition for the term.
  
  For example, in the following, the terms “check in”, “register” and “delta” are all appearing for the first time; the example sentence should be rewritten so they are understandable.
  
  The major function assists you in checking in a file to your version control system and registering successive sets of changes to it as deltas.

- Use the `@dfn` command around a word being introduced, to indicate that the reader should not expect to know the meaning already, and should expect to learn the meaning from this passage.

**Program Invocation Nodes**

You can invoke programs such as Emacs, GCC, and `gawk` from a shell. The documentation for each program should contain a section that describes this. Unfortunately, if the node names and titles for these sections are all different, they are difficult for users to find.

So, there is a convention to name such sections with a phrase beginning with the word ‘Invoking’, as in ‘Invoking Emacs’; this way, users can find the section easily.

**ANSI C Syntax**

When you use `@example` to describe a C function’s calling conventions, use the ANSI C syntax, like this:

```c
void dld_init (char *var{path});
```

And in the subsequent discussion, refer to the argument values by writing the same argument names, again highlighted with `@var`.

Avoid the obsolete style that looks like this:

```c
#include <dld.h>
dld_init (path)
  char *path;
```

Also, it is best to avoid writing `#include` above the declaration just to indicate that the function is declared in a header file. The practice may give the misimpression that the `#include` belongs near the declaration of the function. Either state explicitly which header file holds the declaration or, better yet, name the header file used for a group of functions at the beginning of the section that describes the functions.

**Node Length**

Keep nodes (sections) to a reasonable length, whatever reasonable might be in the given context. Don’t hesitate to break up long nodes into subnodes and have an extensive tree structure; that’s what it’s there for. Many times, readers will probably try to find a single specific point in the manual, using search, indexing, or just plain guessing, rather than reading the whole thing from beginning to end.
You can use the `texi-elements-by-size` utility to see a list of all nodes (or sections) in the document, sorted by size (either lines or words), to find candidates for splitting. It’s in the `util/` subdirectory of the Texinfo sources.

**Bad Examples**

Here are several examples of bad writing to avoid:

In this example, say, “... you must `check in` the new version.” That flows better.

When you are done editing the file, you must perform a `check in`.

In the following example, say, “... makes a unified interface such as VC mode possible.”

SCCS, RCS and other version-control systems all perform similar functions in broadly similar ways (it is this resemblance which makes a unified control mode like this possible).

And in this example, you should specify what ‘it’ refers to:

If you are working with other people, it assists in coordinating everyone’s changes so they do not step on each other.

**And Finally . . .**

- Pronounce TeX as if the ‘X’ were a Greek ‘chi’, as the last sound in the name ‘Bach’. But pronounce Texinfo as in ‘speck’: “teckinfo”.
- Write notes for yourself at the very end of a Texinfo file after the `bye`. None of the processors process text after the `bye`; it is as if the text were within `ignore` . . . `end ignore`. 
Appendix C  Sample Texinfo Files

This appendix includes texts to be used in GNU manuals.

C.1 GNU Sample Texts

Following is a sample Texinfo document with the full texts that should be used (adapted as necessary) in GNU manuals.

As well as the legal texts, it also serves as a practical example of how many elements in a GNU system can affect the manual. If you’re not familiar with all these different elements, don’t worry. They’re not required and a perfectly good manual can be written without them. They’re included here nonetheless because many manuals do (or could) benefit from them.

See [Short Sample], page 11, for a minimal example of a Texinfo file.

Here are some notes on the example:

• The version.texi in the @include command is maintained automatically by Automake (see Section “Texinfo” in GNU Automake). It sets the ‘VERSION’, ‘UPDATED’ and ‘UPDATED-MONTH’ values used elsewhere. If your distribution doesn’t use Automake, but you do use Emacs, you may find the time-stamp.el package helpful (see Section “Time Stamps” in The GNU Emacs Manual).

• The @symcodeindex command reflects the recommendation to use only one index where possible, to make it easier for readers to look up index entries.

• The @dircategory specify a category for the manual. It is used for constructing the Info directory. See Section 2.6 [Directory Category], page 16, which includes a variety of recommended category names. See Section 20.1.4 [Installing Dir Entries], page 196.

• The ‘Invoking’ node is a GNU standard to help users find the basic information about command-line usage of a given program. See Section “Manual Structure Details” in GNU Coding Standards.

• This sample shows how to includes the FDL in a manual using the @include command. The fdl.texi file is available in the Texinfo and other GNU source distributions. It is also available on the GNU website (at https://www.gnu.org/licenses/fdl-1.3.html) along with guidance for using it.

• If the FSF is not the copyright holder, then use the appropriate name.

• For documents that express your personal views, feelings or experiences, it is more appropriate to use a license permitting only verbatim copying, rather than the FDL. See Section C.2 [Verbatim Copying License], page 245.

Here is the sample document:

\input texinfo @c -*-texinfo--
@comment %**start of header
@include version.texi
@settitle GNU Sample @value{VERSION}
@symcodeindex pg cp
@comment %**end of header
@copying
This manual is for GNU Sample (version @value{VERSION}, @value{UPDATED}),


which is an example in the Texinfo documentation.

Copyright © 2016 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

This manual is for GNU Sample (version %VERSION%, %UPDATED%).
This is a sample manual. There is no sample program to invoke, but if there were, you could see its basic usage and command line options here.

@node GNU Free Documentation License
@appendix GNU Free Documentation License
@include fdl.texi

@node Index
@unnumbered Index

@printindex cp
@bye

C.2 Verbatim Copying License

For software manuals and other documentation, it is critical to use a license permitting free redistribution and updating, so that when a free program is changed, the documentation can be updated as well.

On the other hand, for documents that express your personal views, feelings or experiences, it is more appropriate to use a license permitting only verbatim copying.

Here is a sample text for such a license permitting verbatim copying only. This is just the license text itself. For a complete sample document, see the previous sections.

@copying
This document is a sample for allowing verbatim copying only.

Copyright @copyright{} 2016 Free Software Foundation, Inc.

@quotation
Permission is granted to make and distribute verbatim copies of this entire document without royalty provided the copyright notice and this permission notice are preserved.
@end quotation
@end copying

C.3 All-permissive Copying License

For software manuals and other documentation, it is important to use a license permitting free redistribution and updating, so that when a free program is changed, the documentation can be updated as well.

On the other hand, for small supporting files, short manuals (under 300 lines long) and rough documentation (README files, INSTALL files, etc.), the full FDL would be overkill. They can use a simple all-permissive license.

Here is a sample text for such an all-permissive license. This is just the license text itself. For a complete sample document, see the previous sections.
Copyright © 2016 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved.
Appendix D Using Texinfo Mode

You may edit a Texinfo file with any text editor you choose. A Texinfo file is no different from any other ASCII file. However, GNU Emacs comes with a special mode, called Texinfo mode, that provides Emacs commands and tools to help ease your work.

D.1 Texinfo Mode Overview

Texinfo mode provides special features for working with Texinfo files. You can:

- Insert frequently used @-commands.
- Automatically create \texttt{@node} lines.
- Show the structure of a Texinfo 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.

Perhaps the two most helpful features are those for inserting frequently used @-commands and for creating node pointers and menus.

D.2 The Usual GNU Emacs Editing Commands

In most cases, the usual Text mode commands work the same in Texinfo mode as they do in Text mode. Texinfo mode adds new editing commands and tools to GNU Emacs’ general purpose editing features. The major difference concerns filling. In Texinfo mode, the paragraph separation variable and syntax table are redefined so that Texinfo commands that should be on lines of their own are not inadvertently included in paragraphs. Thus, the \texttt{M-q} (\texttt{fill-paragraph}) command will refill a paragraph but not mix an indexing command on a line adjacent to it into the paragraph.

In addition, Texinfo mode sets the page-delimiter variable to the value of \texttt{texinfo-chapter-level-regexp}; by default, this is a regular expression matching the commands for chapters and their equivalents, such as appendices. With this value for the page delimiter, you can jump from chapter title to chapter title with the \texttt{C-x J} (\texttt{forward-page}) and \texttt{C-x [} (\texttt{backward-page}) commands and narrow to a chapter with the \texttt{C-x n p} (\texttt{narrow-to-page}) command. (See Section “Pages” in The GNU Emacs Manual, for details about the page commands.)

GNU Emacs automatically enters Texinfo mode when you visit a file with a .texinfo, .texi or .txi extension. Also, Emacs switches to Texinfo mode when you visit a file that has ‘\texttt{-*-texinfo-*-}’ in its first line. If ever you are in another mode and wish to switch to Texinfo mode, type \texttt{M-x texinfo-mode}.

Like all other Emacs features, you can customize or enhance Texinfo mode as you wish. In particular, the keybindings are very easy to change. The keybindings described here are the default or standard ones.
D.3 Inserting Frequently Used Commands

Texinfo 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:

- `C-c C-c c`  
  `M-x texinfo-insert-@code`  
  Insert `@code{}` and put the cursor between the braces.

- `C-c C-c d`  
  `M-x texinfo-insert-@dfn`  
  Insert `@dfn{}` and put the cursor between the braces.

- `C-c C-c e`  
  `M-x texinfo-insert-@end`  
  Insert `@end` and attempt to insert the correct following word, such as `‘example’` or `‘table’`. (This command does not handle nested lists correctly, but inserts the word appropriate to the immediately preceding list.)

- `C-c C-c i`  
  `M-x texinfo-insert-@item`  
  Insert `@item` and put the cursor at the beginning of the next line.

- `C-c C-c k`  
  `M-x texinfo-insert-@kbd`  
  Insert `@kbd{}` and put the cursor between the braces.

- `C-c C-c n`  
  `M-x texinfo-insert-@node`  
  Insert `@node` and a comment line listing the sequence for the ‘Next’, ‘Previous’, and ‘Up’ nodes. Leave point after the `@node`.

- `C-c C-c o`  
  `M-x texinfo-insert-@noindent`  
  Insert `@noindent` and put the cursor at the beginning of the next line.

- `C-c C-c r`  
  `M-x texinfo-insert-dwim-@ref`  
  This function and binding were added in Emacs 27.1. Inserts one of `@pxref{}`, `@xref{}`, or `@ref{}` based on the text around point; calling it near an unclosed preceding open parenthesis results in `@pxref{}`, at the beginning of a sentence or at `(point-min)` yields `@xref{}`, any other location (including inside a word), will result in `@ref{}`. A numeric argument says how many words the braces should surround. Puts the cursor between the braces.

- `C-c C-c s`  
  `M-x texinfo-insert-@samp`  
  Insert `@samp{}` and put the cursor between the braces.

- `C-c C-c t`  
  `M-x texinfo-insert-@table`  
  Insert `@table` followed by a SPC and leave the cursor after the SPC.
C-c C-c v
M-x texinfo-insert-@var
  Insert @var{} and put the cursor between the braces.

C-c C-c x
M-x texinfo-insert-@example
  Insert @example and put the cursor at the beginning of the next line.

C-c C-c {
M-x texinfo-insert-braces
  Insert {} and put the cursor between the braces.

C-c }
C-c ]
M-x up-list
  Move from between a pair of braces forward past the closing brace. Typing C-c ]
  is easier than typing C-c }, which is, however, more mnemonic; hence the two
  keybindings. (Also, you can move out from between braces by typing C-f.)

To put a command such as @code{...} around an existing word, position the cursor in
front of the word and type C-u 1 C-c C-c c. This makes it easy to edit existing plain text.
The value of the prefix argument tells Emacs how many words following point to include
between braces—‘1’ for one word, ‘2’ for two words, and so on. Use a negative argument to
enclose the previous word or words. If you do not specify a prefix argument, Emacs inserts
the @-command string and positions the cursor between the braces. This feature works only
for those @-commands that operate on a word or words within one line, such as @kbd and
@var.

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 texinfo.el.

C-c C-c C-d (texinfo-start-menu-description) is an insert command that works
differently from the other insert commands. It inserts a node’s section or chapter title in
the space for the description in a menu entry line. (A menu entry has three parts, the
entry name, the node name, and the description. Only the node name is required, but a
description helps explain what the node is about. See Section 3.8.4 [The Parts of a Menu],
page 35.)

To use texinfo-start-menu-description, position point in a menu entry line and type
C-c C-c C-d. The command looks for and copies the title that goes with the node name,
and inserts the title as a description; it positions point at the beginning of the inserted text
so you can edit it. The function does not insert the title if the menu entry line already
contains a description.

This command is only an aid to writing descriptions; it does not do the whole job. You
must edit the inserted text since a title tends to use the same words as a node name but a
useful description uses different words.
D.4 Showing the Sectioning Structure of a File

You can show the sectioning structure of a Texinfo file by using the \texttt{C-c C-s} command (\texttt{texinfo-show-structure}). This command lists the lines that begin with the @-commands for \texttt{@chapter}, \texttt{@section}, and the like. It 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 \texttt{C-c C-c} command (\texttt{occur-mode-goto-occurrence}), to jump to the corresponding spot in the Texinfo file.

\texttt{C-c C-s}  
\texttt{M-x texinfo-show-structure}  
Show the @chapter, @section, and such lines of a Texinfo file.

\texttt{C-c C-c}  
\texttt{M-x occur-mode-goto-occurrence}  
Go to the line in the Texinfo file corresponding to the line under the cursor in the *Occur* buffer.

If you call \texttt{texinfo-show-structure} with a prefix argument by typing \texttt{C-u C-c C-s}, it will list not only those lines with the @-commands for \texttt{@chapter}, \texttt{@section}, and the like, but also the \texttt{@node} lines. You can use \texttt{texinfo-show-structure} with a prefix argument to check whether the ‘Next’, ‘Previous’, and ‘Up’ pointers of an \texttt{@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 by using the \texttt{C-x n n} (\texttt{narrow-to-region}) command and \texttt{texinfo-show-structure} will work on only that region. To see the whole buffer again, use \texttt{C-x n w} (\texttt{widen}). (See Section “Narrowing” in \textit{The GNU Emacs Manual}, for more information about the narrowing commands.)

In addition to providing the \texttt{texinfo-show-structure} command, Texinfo mode sets the value of the page delimiter variable to match the chapter-level @-commands. This enables you to use the \texttt{C-x [} (\texttt{forward-page}) and \texttt{C-x ]} (\texttt{backward-page}) commands to move forward and backward by chapter, and to use the \texttt{C-x n p} (\texttt{narrow-to-page}) command to narrow to a chapter. See Section “Pages” in \textit{The GNU Emacs Manual}, for more information about the page commands.

D.4.1 Using \texttt{texinfo-show-structure}

It is not always easy to keep track of the nodes, chapters, sections, and subsections of a Texinfo file. This is especially true if you are revising or adding to a Texinfo file that someone else has written.

In GNU Emacs, in Texinfo mode, the \texttt{texinfo-show-structure} command lists all the lines that begin with the @-commands that specify the structure: \texttt{@chapter}, \texttt{@section}, \texttt{@appendix}, and so on. With an argument (\texttt{C-u} as prefix argument, if interactive), the command also shows the \texttt{@node} lines. The \texttt{texinfo-show-structure} command is bound to \texttt{C-c C-s} in Texinfo mode, by default.

The lines are displayed in a buffer called the ‘*Occur*’ buffer, indented by hierarchical level. For example, here is a part of what was produced by running \texttt{texinfo-show-structure} on this manual:
Appendix D: Using Texinfo Mode

Lines matching "^\(chapter \| sect\| subs\| subh\| unnum\| major\| chapheading \| heading \| appendix\)"
in buffer texinfo.texi.

... 4177: \chapter Nodes
4198: \heading Two Paths
4231: \section Node and Menu Illustration
4337: \section The \code{@node} Command
4393: \subheading Choosing Node and Pointer Names
4417: \subsection How to Write a \code{@node} Line
4469: \subsection \code{@node} Line Tips

This says that lines 4337, 4393, and 4417 of texinfo.texi begin with the \section, \subheading, and \subsection 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 Texinfo file. See Section “Using Occur” in The GNU Emacs Manual, for more information about occur-mode-goto-occurrence.

The first line in the ‘*Occur*’ window describes the regular expression specified by texinfo-heading-pattern. This regular expression is the pattern that texinfo-show-structure looks for. See Section “Using Regular Expressions” in The GNU Emacs Manual, for more information.

When you invoke the texinfo-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 n (narrow-to-region) command to mark the region. (See Section “Narrowing” in The GNU Emacs Manual.) This is how the example used above was generated. (To see the whole buffer again, use \C-x n w (widen).)

If you call texinfo-show-structure with a prefix argument by typing \C-u \C-c \C-c \C-s, it will list lines beginning with \@node as well as the lines beginning with the @-sign commands for \chapter, \section, and the like.

You can remind yourself of the structure of a Texinfo 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.

D.4.2 Using occur

Sometimes the texinfo-show-structure command produces too much information. Perhaps you want to remind yourself of the overall structure of a Texinfo file, and are overwhelmed by the detailed list produced by texinfo-show-structure. In this case, you can use the occur command directly. To do this, type:

\texttt{M-x occur}

and then, when prompted, type a \texttt{regexp}, a regular expression for the pattern you want to match. (See Section “Regular Expressions” in The GNU Emacs Manual.) 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 ‘$’; for example, ‘catching mistakes$’. 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 Section “Using Occur” in The GNU Emacs Manual, for more information.

D.5 Updating Nodes and Menus

The texi2any command will create an Info file for a hierarchically organized Texinfo file that lacks ‘Next’, ‘Previous’ and ‘Up’ pointers (see Section 3.1 [Writing a Node], page 27). Thus, in general, there is no need for explicit ‘Next’, ‘Previous’, and ‘Up’ pointers. In this setting, menus will be added automatically for nodes without an explicit menu. (See Chapter 19 [Generic Translator texi2any], page 164, for more information about texi2any.)

If you still want explicit pointers, Texinfo 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 Texinfo file after you have worked on it; but you can use them to insert the ‘Next’, ‘Previous’, and ‘Up’ pointers into an @node line that has none and to create menus in a file that has none.

D.5.1 The Updating Commands

You can use the updating commands to:

- insert or update the ‘Next’, ‘Previous’, and ‘Up’ pointers of a node,
- insert or update the menu for a section, and
- create a master menu for a Texinfo source file.

You can also use the commands to update all the nodes and menus in a region or in a whole Texinfo file.

The updating commands work only with conventional Texinfo files, which are structured hierarchically like books. In such files, a structuring command line must follow closely after each @node line, except for the ‘Top’ @node line. (A structuring command line is a line beginning with @chapter, @section, or other similar command.)

You can write the structuring command line on the line that follows immediately after an @node line or else on the line that follows after a single @comment line or a single @ifinfo line. You cannot interpose more than one line between the @node line and the structuring command line; and you may interpose only a @comment line or an @ifinfo line.

Commands which work on a whole buffer require that the ‘Top’ node be followed by a node with a @chapter or equivalent-level command. The menu updating commands will not create a main or master menu for a Texinfo file that has only @chapter-level nodes! The menu updating commands only create menus within nodes for lower level nodes. To create a menu of chapters, you must provide a ‘Top’ node.

The menu updating commands remove menu entries that refer to other Info files since they do not refer to nodes within the current buffer. This is a deficiency. Rather than use
menu entries, you can use cross references to refer to other Info files. None of the updating commands affect cross-references.

Texinfo 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 `texinfo-master-menu` command, is for creating a master menu for a complete file, and optionally, for updating every node and menu in the whole Texinfo file.

The `texinfo-master-menu` command is the primary command:

\textasciitilde{C-c C-u m}

\texttt{M-x texinfo-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, \texttt{C-u}, if interactive), first create or update all the nodes and all the regular menus in the buffer before constructing the master menu. (See Section 2.10 [The Top Node and Master Menu], page 23, for more about a master menu.)

For `texinfo-master-menu` to work, the Texinfo file must have a ‘Top’ node and at least one subsequent node.

After extensively editing a Texinfo file, you can type the following:

\texttt{C-u M-x texinfo-master-menu}

or

\texttt{C-u C-c C-u m}

This updates 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 Texinfo file.

The commands are:

\textasciitilde{C-c C-u C-n}

\texttt{M-x texinfo-update-node}

Insert the ‘Next’, ‘Previous’, and ‘Up’ pointers for the node that point is within (i.e., for the \texttt{@node} line preceding point). If the \texttt{@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, \texttt{C-u}, if interactive), this command updates all \texttt{@node} lines in the region (which is the text between point and mark).

\textasciitilde{C-c C-u C-m}

\texttt{M-x texinfo-make-menu}

Create or update the menu in the node that point is within. With an argument (\texttt{C-u} as prefix argument, if interactive), the command makes or updates menus for the nodes which are either within or a part of the region.

Whenever `texinfo-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.
C-c C-u C-e
M-x texinfo-every-node-update
Insert or update the ‘Next’, ‘Previous’, and ‘Up’ pointers for every node in the buffer.

C-c C-u C-a
M-x texinfo-all-menus-update
Create or update all the menus in the buffer. With an argument (C-u as prefix argument, if interactive), first insert or update all the node pointers before working on the menus.

If a master menu exists, the texinfo-all-menus-update command updates it; but the command does not create a new master menu if none already exists. (Use the texinfo-master-menu command for that.)

When working on a document that does not merit a master menu, you can type the following:

    C-u C-c C-u C-a

or

    C-u M-x texinfo-all-menus-update

This updates all the nodes and menus.

The texinfo-column-for-description variable specifies the column to which menu descriptions are indented. By default, the value is 32 although it can be useful to reduce it to as low as 24. You can set the variable via customization (see Section “Customization” in The GNU Emacs Manual) or with the M-x set-variable command (see Section “Examining and Setting Variables” in The GNU Emacs Manual).

Also, the texinfo-indent-menu-description command may be used to indent existing menu descriptions to a specified column. Finally, if you wish, you can use the texinfo-insert-node-lines command to insert missing @node lines into a file. In particular, you can ignore @node lines altogether in your first draft and then use the texinfo-insert-node-lines command to create @node lines for you. However, we do not recommend this practice. It is better to name the node itself at the same time that you write a segment so you can easily make cross-references. Useful cross-references are an especially important feature of a good Texinfo manual. (See Section D.5.5 [Other Updating Commands], page 256, for more information.)

D.5.2 Updating Requirements

To use the updating commands, you must organize the Texinfo file hierarchically with chapters, sections, subsections, and the like. When you construct the hierarchy of the manual, do not ‘jump down’ more than one level at a time: you can follow the ‘Top’ node with a chapter, but not with a section; you can follow a chapter with a section, but not with a subsection. However, you may ‘jump up’ any number of levels at one time—for example, from a subsection to a chapter.

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:
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.

The menu updating commands create a menu of sections within a chapter, a menu of subsections within a section, and so on. This means that you must have a 'Top' node if you want a menu of chapters.

D.5.3 Update Outer File and Include Files

GNU Emacs Texinfo mode provides the `texinfo-multiple-files-update` command. This command creates or updates 'Next', 'Previous', and 'Up' pointers of included files as well as those in the outer or overall Texinfo file, and it creates or updates a main menu in the outer file. Depending on whether you call it with optional arguments, the command updates only the pointers in the first `@node` line of the included files or all of them.

With `C-u` as a prefix argument, create and insert a master menu in the outer file. With a numeric prefix argument, such as `C-u 2`, 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.

In more details:

`M-x texinfo-multiple-files-update`

Called without any arguments:

- Create or update the 'Next', 'Previous', and 'Up' pointers of the first `@node` line in each file included in an outer or overall Texinfo 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 M-x texinfo-multiple-files-update`

Called with `C-u` as a prefix argument:

- Create or update pointers in the first `@node` line in each included file.
- Create or update the 'Top' level node pointers of the outer file.
- Create and insert a master menu in the outer file. The master menu is made from all the menus in all the included files.


\textit{C-u 8 M-x texinfo-multiple-files-update}  
Called with a numeric prefix argument, such as \texttt{C-u 8}:  
\begin{itemize}
  \item Create or update \textit{all} the `Next`, `Previous`, and `Up` pointers of all the included files.
  \item Create or update \textit{all} the menus of all the included files.
  \item Create or update the `Top` level node pointers of the outer or overall file.
  \item And then create a master menu in the outer file. This is similar to invoking \texttt{texinfo-master-menu} with an argument when you are working with just one file.
\end{itemize}

Note the use of the prefix argument in interactive use: with a regular prefix argument, just \texttt{C-u}, the \texttt{texinfo-multiple-files-update} command inserts a master menu; with a numeric prefix argument, such as \texttt{C-u 8}, the command updates \textit{every} pointer and menu in \textit{all} the files and then inserts a master menu.

\textbf{D.5.4 Include Files Requirements}  
If you plan to use the \texttt{texinfo-multiple-files-update} command, the outer Texinfo file that lists included files within it should contain nothing but the beginning and end parts of a Texinfo file, and a number of \texttt{@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, \texttt{@chapter} 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 a \texttt{@chapter}, an \texttt{@appendix}, or an \texttt{@unnumbered} node. Thus, normally, each included file contains one, and only one, chapter or equivalent-level node.

The outer file should contain only one node, the `Top` node. It should not contain any nodes besides the single `Top` node. The \texttt{texinfo-multiple-files-update} command will not process them.

\textbf{D.5.5 Other Updating Commands}  
In addition to the major updating commands, Texinfo mode possesses several less frequently used updating commands:

\textit{M-x texinfo-insert-node-lines}  
Insert \texttt{@node} lines before the \texttt{@chapter}, \texttt{@section}, and other sectioning commands wherever they are missing throughout a region in a Texinfo file.

With an argument (\texttt{C-u} as prefix argument, if interactive), the command \texttt{texinfo-insert-node-lines} not only inserts \texttt{@node} lines but also inserts the chapter or section titles as the names of the corresponding nodes. In addition, it inserts the titles as node names in pre-existing \texttt{@node} lines that lack names. Since node names should be more concise than section or chapter titles, you must manually edit node names so inserted.

For example, the following marks a whole buffer as a region and inserts \texttt{@node} lines and titles throughout:

\begin{verbatim}
C-x h C-u M-x texinfo-insert-node-lines
\end{verbatim}
This command inserts titles as node names in @node lines; the texinfo-start-menu-description command (see Section D.3 [Inserting], page 248) inserts titles as descriptions in menu entries, a different action. However, in both cases, you need to edit the inserted text.

\texttt{M-x texinfo-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 (\texttt{C-u} as prefix argument, if interactive), the texinfo-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.

\texttt{M-x texinfo-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 novels and other documents that you read through sequentially. (However, in Info, the \texttt{g *} command lets you look through the file sequentially, so sequentially ordered nodes are not strictly necessary.) With an argument (prefix argument, if interactive), the texinfo-sequential-node-update command sequentially updates all the nodes in the region.

### D.6 Formatting for Info

Texinfo mode provides several commands for formatting part or all of a Texinfo file for Info.

#### D.6.1 Running texi2any/makeinfo Within Emacs

The texi2any program provides better error messages than either of the Emacs formatting commands. We recommend it. The texi2any program is independent of Emacs.

You can run texi2any (or makeinfo) in GNU Emacs Texinfo mode by using either the makeinfo-region or the makeinfo-buffer commands. In Texinfo mode, the commands are bound to \texttt{C-c C-m C-r} and \texttt{C-c C-m C-b} by default.

\texttt{C-c C-m C-r}

\texttt{M-x makeinfo-region}

Format the current region for Info.

\texttt{C-c C-m C-b}

\texttt{M-x makeinfo-buffer}

Format the current buffer for Info.

When you invoke makeinfo-region the output goes to a temporary buffer. When you invoke makeinfo-buffer output goes to the file set with \texttt{@setfilename} (see Section 2.5.2 [\texttt{@setfilename}], page 13).

The Emacs makeinfo-region and makeinfo-buffer commands run the texi2any program in a temporary shell buffer. If texi2any finds any errors, Emacs displays the error messages in the temporary buffer.
You can parse the error messages by typing \texttt{C-x ` (\texttt{next-error})}. This causes Emacs to go to and position the cursor on the line in the Texinfo source that \texttt{texi2any} thinks caused the error. See Section “Running \texttt{make} or Compilers Generally” in The GNU Emacs Manual, for more information about using the \texttt{next-error} command.

In addition, you can kill the shell in which the \texttt{texi2any} command is running or make the shell buffer display its most recent output.

\texttt{C-c C-m C-k}  \\
\texttt{M-x makeinfo-kill-job}  \\
\hspace{1em} Kill the current running \texttt{texi2any} (or \texttt{makeinfo}) job (from \texttt{makeinfo-region} or \texttt{makeinfo-buffer}).

\texttt{C-c C-m C-l}  \\
\texttt{M-x makeinfo-recenter-output-buffer}  \\
\hspace{1em} Redisplay the \texttt{texi2any} shell buffer to display its most recent output.

(Note that the parallel commands for killing and recentering a \texttt{TeX} job are \texttt{C-c C-t C-k} and \texttt{C-c C-t C-l}. See Section 18.5.2 [Texinfo Mode Printing], page 159.)

You can specify options for \texttt{texi2any} by setting the \texttt{makeinfo-options} variable with either the \texttt{M-x customize} or the \texttt{M-x set-variable} command, or by setting the variable in your \texttt{.emacs} initialization file.

For example, you could write the following in your \texttt{.emacs} file:

\begin{verbatim}
(setq makeinfo-options
  "--paragraph-indent=0 --no-split
   --fill-column=70 --verbose")
\end{verbatim}

For more information, see [\texttt{texi2any} Options], page 164, as well as “Easy Customization Interface,” “Examining and Setting Variables,” and “Init File” in The GNU Emacs Manual.

\section*{D.6.2 The \texttt{texinfo-format...} Commands}

In GNU Emacs in Texinfo mode, you can format part or all of a Texinfo file with the \texttt{texinfo-format-region} command. This formats the current region and displays the formatted text in a temporary buffer called ‘*Info Region*’.

Similarly, you can format a buffer with the \texttt{texinfo-format-buffer} command. This command creates a new buffer and generates the Info file in it. Typing \texttt{C-x C-s} will save the Info file under the name specified by the \texttt{@setfilename} line which must be near the beginning of the Texinfo file.

\texttt{C-c C-e C-r}  \\
\texttt{texinfo-format-region}  \\
\hspace{1em} Format the current region for Info.

\texttt{C-c C-e C-b}  \\
\texttt{texinfo-format-buffer}  \\
\hspace{1em} Format the current buffer for Info.

The \texttt{texinfo-format-region} and \texttt{texinfo-format-buffer} commands provide you with some error checking, and other functions can provide you with further help in finding formatting errors. These procedures are described in an appendix; see Section D.10 [Catching
Mistakes], page 263. However, the texi2any program provides better error checking (see Section D.6.1 [texi2any in Emacs], page 257).

A peculiarity of the texinfo-format-buffer and texinfo-format-region commands is that they do not indent (nor fill) paragraphs that contain @w or @* commands.

### D.7 Printing

Typesetting and printing a Texinfo file is a multi-step process in which you first create a file for printing (called a DVI file), and then print the file. Optionally, you may also create indices. To do this, you must run the texindex command after first running the tex typesetting command; and then you must run the tex command again. Or else run the texi2dvi command which automatically creates indices as needed (see [Format with texi2dvi], page 154).

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 texinfo-tex-region and related commands for this purpose. Use the texinfo-tex-buffer command to format all of a buffer.

\[ C-c C-t C-b \]
\[ M-x \text{texinfo-tex-buffer} \]
\[ \text{Run texi2dvi on the buffer. In addition to running \LaTeX\ on the buffer, this} \]
\[ \text{command automatically creates or updates indices as needed.} \]

\[ C-c C-t C-r \]
\[ M-x \text{texinfo-tex-region} \]
\[ \text{Run \LaTeX\ on the region.} \]

\[ C-c C-t C-i \]
\[ M-x \text{texinfo-texindex} \]
\[ \text{Run texindex to sort the indices of a Texinfo file formatted with \text{texinfo-tex-region}. The \text{texinfo-tex-region} command does not run texindex automatically; it only runs the \text{tex} typesetting command. You must run the} \]
\[ \text{\text{texinfo-tex-region} command a second time after sorting the raw index files} \]
\[ \text{with the \text{texindex} command. (Usually, you do not format an index when} \]
\[ \text{you format a region, only when you format a buffer. Now that the \text{texi2dvi} command exists, there is little or no need for this command.)} \]

\[ C-c C-t C-p \]
\[ M-x \text{texinfo-tex-print} \]
\[ \text{Print the file (or the part of the file) previously formatted with \text{texinfo-tex-buffer} or \text{texinfo-tex-region}.} \]

For texinfo-tex-region or texinfo-tex-buffer to work, the file must start with a `\input texinfo` line and must include a @settitle line. The file must end with @bye on a line by itself. (When you use texinfo-tex-region, you must surround the @settitle line with start-of-header and end-of-header lines.)

See Chapter 18 [Hardcopy with \TeX\], page 154, for a description of the other \TeX\ related commands, such as tex-show-print-queue.
D.8 Texinfo Mode Summary

In Texinfo mode, each set of commands has default keybindings that begin with the same keys. All the commands that are custom-created for Texinfo mode begin with \texttt{C-c}. The keys are somewhat mnemonic.

Insert Commands

The insert commands are invoked by typing \texttt{C-c} twice and then the first letter of the \texttt{@-command} to be inserted. (It might make more sense mnemonically to use \texttt{C-c C-i}, for 'custom insert', but \texttt{C-c C-c} is quick to type.)

- \texttt{C-c C-c c} Insert \texttt{@code}.
- \texttt{C-c C-c d} Insert \texttt{@dfn}.
- \texttt{C-c C-c e} Insert \texttt{@end}.
- \texttt{C-c C-c i} Insert \texttt{@item}.
- \texttt{C-c C-c n} Insert \texttt{@node}.
- \texttt{C-c C-c s} Insert \texttt{@samp}.
- \texttt{C-c C-c v} Insert \texttt{@var}.
- \texttt{C-c \{} Insert braces.
- \texttt{C-c \}} Move out of enclosing braces.
- \texttt{C-c C-c C-d} Insert a node's section title in the space for the description in a menu entry line.

Show Structure

The \texttt{texinfo-show-structure} command is often used within a narrowed region.

- \texttt{C-c C-s} List all the headings.

The Master Update Command

The \texttt{texinfo-master-menu} command creates a master menu; and can be used to update every node and menu in a file as well.

- \texttt{C-c C-u m} M-x \texttt{texinfo-master-menu}
  Create or update a master menu.

- \texttt{C-u C-c C-u m} With \texttt{C-u} as a prefix argument, first create or update all nodes and regular menus, and then create a master menu.

Update Pointers

The update pointer commands are invoked by typing \texttt{C-c C-u} and then either \texttt{C-n} for \texttt{texinfo-update-node} or \texttt{C-e} for \texttt{texinfo-every-node-update}.

- \texttt{C-c C-u C-n} Update a node.
- \texttt{C-c C-u C-e} Update every node in the buffer.
Update Menus
Invoke the update menu commands by typing \texttt{C-c C-u} and then either \texttt{C-m} for \texttt{texinfo-make-menu} or \texttt{C-a} for \texttt{texinfo-all-menus-update}. To update both nodes and menus at the same time, precede \texttt{C-c C-u C-a} with \texttt{C-u}.

\texttt{C-c C-u C-m} Make or update a menu.

\texttt{C-c C-u C-a} Make or update all menus in a buffer.

\texttt{C-u C-c C-u C-a} With \texttt{C-u} as a prefix argument, first create or update all nodes and then create or update all menus.

Format for Info
The Info formatting commands that are written in Emacs Lisp are invoked by typing \texttt{C-c C-e} and then either \texttt{C-r} for a region or \texttt{C-b} for the whole buffer.

The Info formatting commands that are based on the \texttt{texi2any/makeinfo} program are invoked by typing \texttt{C-c C-m} and then either \texttt{C-r} for a region or \texttt{C-b} for the whole buffer.

Use the \texttt{texinfo-format...} commands:

\texttt{C-c C-e C-r} Format the region.

\texttt{C-c C-e C-b} Format the buffer.

Use \texttt{texi2any/makeinfo}:

\texttt{C-c C-m C-r} Format the region.

\texttt{C-c C-m C-b} Format the buffer.

\texttt{C-c C-m C-l} Recenter the \texttt{texi2any} output buffer.

\texttt{C-c C-m C-k} Kill the \texttt{texi2any} formatting job.

Typeset and Print
The \TeX{} typesetting and printing commands are invoked by typing \texttt{C-c C-t} and then another control command: \texttt{C-r} for \texttt{texinfo-tex-region}, \texttt{C-b} for \texttt{texinfo-tex-buffer}, and so on.

\texttt{C-c C-t C-r} Run \TeX{} on the region.

\texttt{C-c C-t C-b} Run \texttt{texi2dvi} on the buffer.

\texttt{C-c C-t C-i} Run \texttt{texindex}.

\texttt{C-c C-t C-p} Print the DVI file.

\texttt{C-c C-t C-q} Show the print queue.

\texttt{C-c C-t C-d} Delete a job from the print queue.

\texttt{C-c C-t C-k} Kill the current \TeX{} formatting job.

\texttt{C-c C-t C-x} Quit a currently stopped \TeX{} formatting job.

\texttt{C-c C-t C-l} Recenter the output buffer.

Other Updating Commands
The remaining updating commands do not have standard keybindings because they are rarely used.
D.9 Direct Formatting of Info files

In general, there is no need to edit or process Info files. Most of the content of Info file is plain text, but the tag tables require computing offsets for nodes and indirect files (see Section 20.2 [Tag and Split Files], page 199), which is impractical. The Texinfo processors output well-formatted Info from Texinfo input. Also, lot of information from the Texinfo source is lost in the Info file. The Texinfo source is therefore more suitable to analyse a manual, be it for error reporting, for statistics, or to set up translations.

Still, especially with manually written Info files, there are some situations where it may be interesting to process Info files directly, in particular to compute Info files node offsets and to compute split Info files file offsets. Two Emacs commands do that. Info-tagify adds a tag table for a nonsplit file lacking one. This function is also useful for Info file validation (see Section D.10.4 [Running Info-validate], page 266). It is also possible to split a nonsplit Info file with Info-split.

D.9.1 Tagifying a File

Texinfo processors create tag tables automatically. The only exception arise when texinfo-format-buffer is called with a prefix, as \texttt{C-u M-x texinfo-format-buffer}, to create an non split Info file without a tag table (which can be useful in some cases for Info file validation, see Section D.10.4.2 [Unsplit and Tagify], page 267). Adding a tag table is therefore more generally useful for manually written Info files.

To create a tag table for an nonsplit file, visit the Info file you wish to tagify and type:

\texttt{M-x Info-tagify}

(Note the uppercase ‘I’ in \texttt{Info-tagify}.) This creates an Info file with a tag table. A tag table is part of a well formed Info file (see Section 20.2 [Tag and Split Files], page 199). A tag table is also needed to validate or split the Info file.
D.9.2 Splitting a File Manually

You should split a large file or else let the `texinfo-format-buffer` or `makeinfo-buffer` command do it for you automatically. (Generally you will let one of the formatting commands do this job for you. See Section D.6 [Info Formatting], page 257.)

The split-off files are called the indirect subfiles.

Tag tables are created automatically by the formatting command; you only need to create a tag table yourself if you are doing the job manually. See Section D.9.1 [Tagifying], page 262, for information about creating a tag table.

Visit the Info file you wish to tagify and split manually and type the two commands:

```
M-x Info-tagify
M-x Info-split
```

(Note that the ‘I’ in ‘Info’ is uppercase.)

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. See Section 20.2 [Tag and Split Files], page 199.

D.10 Catching Mistakes

Besides mistakes in the content of your documentation, there are two kinds of mistake you can make with Texinfo: you can make mistakes with @-commands, and you can make mistakes with the structure of the nodes and chapters.

For finding problems with @-commands, you can run TpX or a region formatting command on the region that has a problem; indeed, you can run these commands on each region as you write it.

For finding problems with the structure of nodes and chapters, you can use `C-c C-s` (`texinfo-show-structure`) and the related `occur` command and you can use the `M-x Info-validate` command.

D.10.1 texi2any Preferred

The `texi2any` program does an excellent job of catching errors and reporting them—far better than `texinfo-format-region` or `texinfo-format-buffer`. In addition, the various functions for automatically creating and updating node pointers and menus remove many opportunities for human error.

Use `texi2any` (or its Texinfo mode manifestations, `makeinfo-region` and `makeinfo-buffer`) to format your file and check for other errors. This is the best way to work with Texinfo. But if you cannot use `texi2any`, or your problem is very puzzling, then you may want to use the tools described in this section.

D.10.2 Catching Errors with Info Formatting

After you have written part of a Texinfo file, you can use the `texinfo-format-region` or the `makeinfo-region` command to see whether the region formats properly.
Appendix D: Using Texinfo Mode

Most likely, however, you are reading this section because for some reason you cannot use the `makeinfo-region` command; therefore, the rest of this section presumes that you are using `texinfo-format-region`.

If you have made a mistake with an @-command, `texinfo-format-region` will stop processing at or after the error and display an error message. To see where in the buffer 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 where the error occurred (or more precisely, where it was detected).

For example, if you accidentally end a menu with the command `@end menus` with an ‘s’ on the end, instead of with `@end menu`, you will see an error message that says:

```plaintext
@end menus is not handled by texinfo
```

The cursor will stop at the point in the buffer where the error occurs, or not long after it. The buffer will look like this:

```plaintext
---------- Buffer: *Info Region* ----------
* Menu:

* Using texinfo-show-structure:: How to use 'texinfo-show-structure' to catch mistakes.

* Running Info-validate:: How to check for unreferenced nodes.
@end menus
```

The `texinfo-format-region` command sometimes provides slightly odd error messages. For example, the following cross-reference fails to format:

```plaintext
(@xref{Catching Mistakes, for more info.)
```

In this case, `texinfo-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 `texinfo-format-region` fails to detect mistakes. For example, in the following, the closing brace is swapped with the closing parenthesis:

```plaintext
(@xref{Catching Mistakes), for more info.)
```

Formatting produces:

```plaintext
(*Note for more info.: Catching Mistakes)
```

The only way for you to detect this error is to realize that the reference should have looked like this:

```plaintext
(*Note Catching Mistakes::, for more info.)
```

Incidentally, if you are reading this node in Info and type `f RET` (Info-follow-reference), you will generate an error message that says:

```plaintext
No such node: "Catching Mistakes) The only way ...
```
This is because Info perceives the example of the error as the first cross-reference in this node and if you type a RET immediately after typing the Info f command, Info will attempt to go to the referenced node. If you type f catch TAB RET, Info will complete the node name of the correctly written example and take you to the ‘Catching Mistakes’ node. (If you try this, you can return from the ‘Catching Mistakes’ node by typing l (Info-last).)

D.10.3 Debugging with \TeX

You can also catch mistakes when you format a file with \TeX.

Usually, you will want to do this after you have run texinfo-format-buffer (or, better, makeinfo-buffer) on the same file, because texinfo-format-buffer sometimes displays error messages that make more sense than \TeX. (See Section D.10.2 [Debugging with Info], page 263, for more information.)

For example, \TeX was run on a Texinfo file, part of which is shown here:

```
---------- Buffer: texinfo.texi ----------
name of the Texinfo 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.})
---------- Buffer: texinfo.texi ----------
```

(The cross-reference lacks a closing brace.) \TeX produced the following output, after which it stopped:

```
---------- Buffer: *tex-shell* ----------
Runaway argument?
{sorting indices, for more information about sorting
indices.} @ETC.
! Paragraph ended before @xref was complete.
<to be read again>
    @par
1.27
?
---------- Buffer: *tex-shell* ----------
```

In this case, \TeX produced an accurate and understandable error message:

```
Paragraph ended before @xref was complete.
```

`@par' is an internal \TeX command of no relevance to Texinfo. ‘1.27’ means that \TeX detected the problem on line 27 of the Texinfo file. The ‘?’ is the prompt \TeX uses in this circumstance.

Unfortunately, \TeX is not always so helpful, and sometimes you must truly be a Sherlock Holmes to discover what went wrong.

In any case, if you run into a problem like this, you can do one of three things.

1. You can tell \TeX to continue running and ignore just this error by typing RET at the ‘?’ prompt.
2. You can tell \TeX{} to continue running and to ignore all errors as best it can by typing \texttt{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 \TeX{} when it is producing such an avalanche of error messages, type \texttt{C-c c} (or \texttt{C-c C-c}, if you are running a shell inside Emacs).

3. You can tell \TeX{} to stop this run by typing \texttt{x RET} at the ‘?’ prompt.

   If you are running \TeX{} inside Emacs, you need to switch to the shell buffer and line at which \TeX{} offers the ‘?’ prompt.

   Sometimes \TeX{} will format a file without producing error messages even though there is a problem. This usually occurs if a command is not ended but \TeX{} is able to continue processing anyhow. For example, if you fail to end an itemized list with the \texttt{@end itemize} command, \TeX{} will write a DVI file that you can print out. The only error message that \TeX{} will give you is the somewhat mysterious comment:

   \begin{verbatim}
   (@end occurred inside a group at level 1)
   \end{verbatim}

   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 \TeX{} says that it expected to find an \texttt{@end} command somewhere in the file; but that it could not determine where it was needed.

   Another source of notoriously hard-to-find errors is a missing \texttt{@end group} command. If you ever are stumped by incomprehensible errors, look for a missing \texttt{@end group} command first.

   If the Texinfo file lacks header lines, \TeX{} may stop in the beginning of its run and display output that looks like the following. The ‘*’ indicates that \TeX{} is waiting for input.

   \begin{verbatim}
   This is TeX, Version 3.14159 (Web2c 7.0)
   (test.texinfo [1])
   *
   \end{verbatim}

   In this case, simply type \texttt{\end RET} after the asterisk. Then write the header lines in the Texinfo file and run the \TeX{} command again. (Note the use of the backslash, ‘\’. \TeX{} uses ‘\’ instead of ‘\@’; and in this circumstance, you are working directly with \TeX{}, not with Texinfo.)

\section*{D.10.4 Finding Badly Referenced Nodes}

You can use the \texttt{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 \texttt{Info-validate} command works only on Info files, not on Texinfo files.

   The \texttt{texi2any} program validates pointers automatically, so you do not need to use the \texttt{Info-validate} command if you are using \texttt{texi2any}. With the customization variable \texttt{CHECK_NORMAL_MENU_STRUCTURE} set, \texttt{texi2any} will also warn if the nodes pointers (either explicitly or automatically set) are not consistent with the order of node menu entries. \texttt{texi2any} does not check that every ‘Next’ pointer is matched by a ‘Previous’ (in the node where the ‘Next’ points) which points back, since it may be correct for a non standard document structure.
D.10.4.1 Using Info-validate

To use Info-validate, visit the Info file you wish to check and type:

\texttt{M-x Info-validate}

Note that the Info-validate command requires an uppercase 'I'. You may also need to create a tag table before running Info-validate. See Section D.9.1 [Tagifying], page 262.

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:

\texttt{In node "Overview", invalid Next: Texinfo 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 `Texinfo Mode' to our test case but we do not specify a `Previous' for this node. Then we will get the following error message:

\texttt{In node "Texinfo Mode", should have Previous: Overview}

This is because, with a standard document structure, 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 entries and cross-references point to actual nodes.

Info-validate requires a tag table and does not work with files that have been split. (The texinfo-format-buffer command automatically splits large files.) In order to use Info-validate on a large file, you must run texinfo-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.

D.10.4.2 Creating an Unsplit File and Adding a Tag Table

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 300,000 bytes or so), you need to run the texinfo-format-buffer or makeinfo-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 texinfo-format-buffer from splitting a Texinfo file into smaller Info files, give a prefix to the M-x texinfo-format-buffer command:

\texttt{C-u M-x texinfo-format-buffer}

or else

\texttt{C-u C-c C-e C-b}
When you do this, `texinfo-format-buffer` will not split the file and will not create a tag table for it.

After creating an unsplit Info file, you must create a tag table for it (see Section D.9.1 [Tagifying], page 262). Visit the unsplit Info file created by `texinfo-format-buffer` and type:

```
M-x Info-tagify
```

The third step is to validate the Info file:

```
M-x Info-validate
```

(Note the uppercase ‘I’ in `Info-validate`.) In brief, the steps are:

```
C-u M-x texinfo-format-buffer
M-x Info-tagify
M-x Info-validate
```

After you have validated the node structure, you can rerun `texinfo-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 (see Section D.9.2 [Splitting], page 263).

## D.11 Batch Formatting

You can format Texinfo files for Info using `batch-texinfo-format` and Emacs batch mode. You can run Emacs in batch mode from any shell, including a shell inside of Emacs. (See Section “Initial Options” in The GNU Emacs Manual.)

Here is a shell command to format all the files that end in `.texinfo` in the current directory:

```
emacs -batch -funcall batch-texinfo-format *.texinfo
```

Emacs processes all the files listed on the command line, even if an error occurs while attempting to format some of them.

Run `batch-texinfo-format` only with Emacs in batch mode as shown; it is not interactive. It kills the batch mode Emacs on completion.

`batch-texinfo-format` is convenient if you lack `texi2any` and want to format several Texinfo files at once. When you use Batch mode, you create a new Emacs process. This frees your current Emacs, so you can continue working in it. (When you run `texinfo-format-region` or `texinfo-format-buffer`, you cannot use that Emacs for anything else until the command finishes.)
Appendix E Global Document Commands

Here are additional commands which affect the document as a whole. Most of these commands are for customizing the appearance of the printed output. They are generally all given before the Top node, if they are given at all.

E.1 @setchapternewpage: Blank Pages Before Chapters

In an officially bound book, text is usually printed on both sides of the paper, chapters start on right-hand pages, and right-hand pages have odd numbers. But in short reports, text often is printed only on one side of the paper. Also in short reports, chapters sometimes do not start on new pages, but are printed on the same page as the end of the preceding chapter, after a small amount of vertical whitespace.

You can use the @setchapternewpage command with various arguments to specify how chapters should be started in printed output and whether headers should be formatted for printing on one or both sides of the paper (single-sided or double-sided printing).

Write the @setchapternewpage command at the beginning of a line followed by its argument. For example, you would write the following to cause each chapter to start on a fresh odd-numbered page:

```plaintext
@setchapternewpage odd
```

You can specify one of three alternatives with the @setchapternewpage command:

- **@setchapternewpage off**
  - Typeset a new chapter on the same page as the last chapter, after skipping some vertical whitespace. Also, format page headers for single-sided printing.

- **@setchapternewpage on**
  - Start new chapters on new pages and format page headers for single-sided printing. This is the form most often used for short reports or personal printing. This is the default.

- **@setchapternewpage odd**
  - Start new chapters on new, odd-numbered pages (right-handed pages) and typeset for double-sided printing. This is the form most often used for books and manuals.

Texinfo does not have a @setchapternewpage even command, because there is no printing tradition of starting chapters or books on an even-numbered page.

If you don’t like the default headers that @setchapternewpage sets, you can explicit control them with the @headings command. See Section E.2.1 [@headings], page 270.

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. By convention, table of contents and frontmatter pages are numbered with roman numerals and not in sequence with the rest of the document.

The @setchapternewpage has no effect in output formats that do not have pages, such as Info and HTML.

We recommend not including any @setchapternewpage command in your document source at all, since such desired pagination is not intrinsic to the document. For a particular
hard copy run, if you don’t want the default output (no blank pages, same headers on all
pages) use the --texinfo option to texi2dvi to specify the output you want.

E.2 Page Headings

Most printed manuals contain headings along the top of every page except the title
and copyright pages. Some manuals also contain footings. Headings and footings have no
meaning in Info or the other output formats.

Texinfo provides two standard heading formats, one for manuals printed on one side
of each sheet of paper, and the other for manuals printed on both sides of the paper. By
default, nothing is specified for the footing of a Texinfo file, so the footing remains blank.

Texinfo also has several heading and footing commands that you can use to generate
your own heading and footing formats.

In Texinfo, headings and footings are single lines at the tops and bottoms of pages; you
cannot create multiline headings or footings. Each header or footer line is divided into three
parts: a left part, a middle part, and a right part. Any part, or a whole line, may be left
blank. Text for the left part of a header or footer line is set flushleft; text for the middle
part is centered; and, text for the right part is set flushright.

E.2.1 The @headings Command

The @headings command is rarely used. It specifies what kinds of page headings and
footings to print on each page. Usually, this is controlled by the @setchapternewpage
command. You need the @headings command only if the @setchapternewpage command
does not do what you want, or if you want to turn off predefined page headings prior to
defining your own. Write a @headings command immediately after the @end titlepage
command.

You can use @headings as follows:

@headings off
    Turn off printing of page headings.

@headings single
    Turn on page headings appropriate for single-sided printing.

@headings double
    Turn on page headings appropriate for double-sided printing.

@headings singleafter
@headings doubleafter
    Turn on single or double headings, respectively, after the current page is
    output.

@headings on
    Turn on page headings: single if 'setchapternewpage on’, double otherwise.

For example, suppose you write @setchapternewpage off before the @titlepage command
to start a new chapter on the same page as the end of the last chapter. This command
also causes page headers to be typeset for single-sided printing. To cause page headers to
be typeset for double-sided printing, write @headings double after the @end titlepage
command.
You can stop any page headings at all from being generated by writing `@headings off` on a line of its own immediately after the line containing the `@end titlepage` command, like this:

```latex
@end titlepage
@headings off
```

The `@headings off` command overrides the `@end titlepage` command, which would otherwise turn on print page headings.

**E.2.2 Standard Heading Formats**

Texinfo provides two standard heading formats, one for manuals printed on one side of each sheet of paper, and the other for manuals printed on both sides of the paper.

The standard format for single-sided printing consists of a header line in which the left-hand part contains the name of the chapter, the central part is blank, and the right-hand part contains the page number. A single-sided page looks like this:

```
\hline
<table>
<thead>
<tr>
<th>chapter</th>
<th>page number</th>
</tr>
</thead>
<tbody>
<tr>
<td>\hline</td>
<td></td>
</tr>
<tr>
<td>Start of text ...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
```

In the standard double-sided format, the left part of the left-hand (even-numbered) page contains the page number, the central part is blank, and the right part contains the title (specified by the `@settitle` command). The left part of the right-hand (odd-numbered) page contains the name of the chapter, the central part is blank, and the right part contains the page number. An even-numbered page and an odd-numbered page, side by side as in an open book, look like this:

```
\hline
| \hline |
| page number | title |
| \hline |
| Start of text ... |
| \hline |
| \hline |
| chapter | page number |
| \hline |
| More text ... |
| \hline |
```

The chapter name is preceded by the word “Chapter”, the chapter number and a colon. This makes it easier to keep track of where you are in the manual.

Note that on pages where a new chapter starts, some text may be omitted from the heading line.

By default, nothing is specified for the footing of a Texinfo file, so the footing remains blank.

**E.2.3 How to Make Your Own Headings**

You can use the standard headings provided with Texinfo or specify your own. By default, Texinfo has no footers, so if you specify them, the available page size for the main text will be slightly reduced.
Texinfo provides six commands for specifying headings and footings:

- `@everyheading` and `@everyfooting` generate page headers and footers that are the same for both even- and odd-numbered pages.
- `@evenheading` and `@evenfooting` commands generate headers and footers for even-numbered (left-hand) pages.
- `@oddheading` and `@oddfooting` generate headers and footers for odd-numbered (right-hand) pages.

Write custom heading specifications in the Texinfo file immediately after the `@end titlepage` command. You must cancel the predefined heading commands with the `@headings off` command before defining your own specifications.

Here is how to place the chapter name at the left, the page number in the center, and the date at the right of every header for both even- and odd-numbered pages:

```
@headings off
@everyheading @thischapter @| @thispage @| @today{}
```

You need to divide the left part from the central part and the central part from the right part by inserting `@|` between parts. Otherwise, the specification command will not be able to tell where the text for one part ends and the next part begins.

Each part can contain text or @-commands. The text is printed as if the part were within an ordinary paragraph in the body of the page. The @-commands replace themselves with the page number, date, chapter name, or whatever.

Here are the six heading and footing commands:

- `@everyheading left @| center @| right`
- `@everyfooting left @| center @| right`
  The ‘every’ commands specify the format for both even- and odd-numbered pages. These commands are for documents that are printed on one side of each sheet of paper, or for documents in which you want symmetrical headers or footers.

- `@evenheading left @| center @| right`
- `@oddheading left @| center @| right`
- `@evenfooting left @| center @| right`
- `@oddfooting left @| center @| right`
  The ‘even’ and ‘odd’ commands specify the format for even-numbered pages and odd-numbered pages. These commands are for books and manuals that are printed on both sides of each sheet of paper.

Use the ‘@this...’ series of @-commands to provide the names of chapters and sections and the page number. You can use the ‘@this...’ commands in the left, center, or right portions of headers and footers.

Here are the ‘@this...’ commands:

- `@thispage`
  Expands to the current page number.
- `@thissectionname`
  Expands to the name of the current section.
@thissectionnum
Expands to the number of the current section.

@thissection
Expands to the number and name of the current section, in the format ‘Section 1: Title’.

@thischaptername
Expands to the name of the current chapter.

@thischapternum
Expands to the number of the current chapter, or letter of the current appendix.

@thischapter
Expands to the number and name of the current chapter, in the format ‘Chapter 1: Title’.

@thistitle
Expands to the name of the document, as specified by the @settitle command.

@thisfile
For @include files only: expands to the name of the current @include file. If the current Texinfo source file is not an @include file, this command has no effect. This command does not provide the name of the current Texinfo source file unless it is an @include file. (See Chapter 17 [Include Files], page 152, for more information about @include files.) Not expanded in \LaTeXX.

You can also use the @today{} command, which expands to the current date, in ‘1 Jan 1900’ format.

Other @-commands and text are printed in a header or footer just as if they were in the body of a page. It is useful to incorporate text, particularly when you are writing drafts:

@headings off
@everyheading @emphasis{Draft!} @| @thispage @| @thischapter
@everyfooting @| @| Version: 0.27: @today{}

Beware of overlong titles: they may overlap another part of the header or footer and blot it out.

If you have very short chapters and/or sections, several of them can appear on a single page. You can specify which chapters and sections you want @thischapter, @thissection and other such macros to refer to on such pages as follows:

@everyheadingmarks ref
@everyfootingmarks ref

The ref argument can be either top (the @this... commands will refer to the chapter/section at the top of a page) or bottom (the commands will reflect the situation at the bottom of a page). These ‘@every...’ commands specify what to do on both even- and odd-numbered pages.
These ‘@even...’ and ‘@odd...’ commands specify what to do on only even- or odd-numbered pages, respectively. The ref argument is the same as with the ‘@every...’ commands.

Write these commands immediately after the @...contents commands, or after the @end titlepage command if you don’t have a table of contents or if it is printed at the end of your manual. These commands have no effect in \LaTeX.

By default, for \TeX, the @this... commands reflect the situation at the bottom of a page both in headings and in footings.

E.3 @paragraphindent: Controlling Paragraph Indentation

The Texinfo processors may insert whitespace at the beginning of the first line of each paragraph, thereby indenting that paragraph. You can use the @paragraphindent command to specify this indentation. Write a @paragraphindent command at the beginning of a line followed by either ‘asis’ or a number:

@paragraphindent indent

The indentation is according to the value of indent:

asis          Do not change the existing indentation (not implemented in printed output).
none          Omit all indentation.
0             Indent by n space characters in Info output, by n ems in printed output.

The default value of indent is 3. @paragraphindent is ignored for HTML output.

It is best to write the @paragraphindent command before the end-of-header line at the beginning of a Texinfo file, so the region formatting commands indent paragraphs as specified. See [Start of Header], page 15.

E.4 @firstparagraphindent: Indenting After Headings

As you can see in the present manual, the first paragraph in any section is not indented by default. Typographically, indentation is a paragraph separator, which means that it is unnecessary when a new section begins. This indentation is controlled with the @firstparagraphindent command:

@firstparagraphindent word

The first paragraph after a heading is indented according to the value of word:

none          Prevents the first paragraph from being indented (default). This option is ignored by makeinfo if @paragraphindent asis is in effect.
insert        Include normal paragraph indentation. This respects the paragraph indentation set by a @paragraphindent command (see Section E.3 [@paragraphindent], page 274).
\@firstparagraphindent is ignored for HTML and DocBook output.

It is best to write the \@firstparagraphindent command before the end-of-header line at the beginning of a Texinfo file, so the region formatting commands indent paragraphs as specified. See [Start of Header], page 15.

E.5 \@exampleindent: Environment Indenting

The Texinfo processors indent each line of \@example and similar environments. You can use the \@exampleindent command to specify this indentation. Write an \@exampleindent command at the beginning of a line followed by either `asis' or a number:

\begin{quote}
\@exampleindent indent
\end{quote}

The indentation is according to the value of \texttt{indent}:

\begin{itemize}
\item \texttt{asis} Do not change the existing indentation (not implemented in printed output).
\item 0 Omit all indentation.
\item \texttt{n} Indent environments by \texttt{n} space characters in Info output, by \texttt{n} ems in printed output.
\end{itemize}

The default value of \texttt{indent} is 5 spaces in Info, and 0.4in in printed output, which is somewhat less. (The reduction is to help fit more characters onto physical lines in printed manuals.)

It is best to write the \@exampleindent command before the end-of-header line at the beginning of a Texinfo file, so the region formatting commands indent paragraphs as specified. See [Start of Header], page 15.

E.6 \@smallbook: Printing "Small" Books

By default, \TeX typesets pages for printing in an 8.5 by 11 inch format. However, you can direct \TeX or \LaTeX to typeset a document in a 7 by 9.25 inch format that is suitable for bound books by inserting the following command on a line by itself at the beginning of the Texinfo file, before the title page:

\begin{quote}
\@smallbook
\end{quote}

(Since many books are about 7 by 9.25 inches, this command might better have been called the \@regularbooksizes command, but it came to be called the \@smallbook command by comparison to the 8.5 by 11 inch format.)

See Section 18.2 [Format with \texttt{texi2dvi} or \texttt{texi2pdf}], page 154, and Section 18.7 [Preparing for \TeX], page 161, for other ways to format with \@smallbook with \TeX that do not require changing the source file.

E.7 Printing on A4 Paper

You can format a document for printing on ISO 216 A4 paper size with the \@afourpaper command. Write the command on a line by itself near the beginning of the Texinfo file, before the title page.

You may or may not prefer the formatting that results from the command \@afourlatex. There’s also \@afourwide for A4 paper in wide format, \@afivepaper for A5 paper, and \@bsixpaper for B6 paper.
See Section 18.2 [Format with texi2dvi or texi2pdf], page 154, and Section 18.7 [Preparing for \TeX{}], page 161, for other ways to format for different paper sizes that do not require changing the source file.

E.8 \texttt{@pagesizes [width][, height]: Custom Page Sizes}

You can explicitly specify the height and (optionally) width of the main text area on the page with the \texttt{@pagesizes} command. Write this on a line by itself near the beginning of the Texinfo file, before the title page. The height comes first, then the width if desired, separated by a comma. Examples:

\begin{verbatim}
@pagesizes 200mm,150mm
\end{verbatim}

and

\begin{verbatim}
@pagesizes 11.5in
\end{verbatim}

This would be reasonable for printing on B5-size paper. To emphasize, this command specifies the size of the text area, not the size of the paper (which is 250 mm by 177 mm for B5, 14 in by 8.5 in for legal).

To make more elaborate changes, such as changing any of the page margins, you must define a new command in \texttt{texinfo.tex} or \texttt{texinfo.cnf}.

See Section 18.2 [Format with texi2dvi or texi2pdf], page 154, and Section 18.7 [Preparing for \TeX{}], page 161, for other ways to specify \texttt{@pagesizes} that do not require changing the source file.

E.9 Microtypography

Some versions of \TeX{}, namely pdf\TeX{} and Lua\TeX{}, have microtypography features. These involve stretching font glyphs slightly, and allowing text to produce very slightly into the margins. Use of these gives \TeX{} more flexibility in breaking a paragraph into lines, and can improve paragraph appearance by reducing hyphenation and producing a more consistent typographic color.

\texttt{texinfo.tex} uses these features (‘microtype’ for short) when available. You can turn microtype off by specifying ‘@microtype off’ in your input file; likewise, ‘@microtype on’ turns microtype back on.

The fonts used in \LaTeX{} output may not be scalable. If fonts are not scalable, using microtypography could trigger an error when converting the \LaTeX{} output. Therefore microtypography is not turned on in the default case in \LaTeX{} output. It is possible to turn it on with \texttt{@microtype on}. A way to get scalable fonts for the font used in the default \LaTeX{} output is to install the CM-Super font package (\url{http://ctan.org/pkg/cm-super}).

The \texttt{@microtype} command does nothing for other output formats.

E.10 Magnification

You can attempt to direct \TeX{} to typeset pages larger or smaller than usual with the \texttt{\mag} \TeX{} command. Everything that is typeset is scaled proportionally larger or smaller. (\texttt{\mag} stands for “magnification”.) This is not a Texinfo @-command, but is a raw \TeX{} command that is prefixed with a backslash. You have to write this command between \texttt{@tex} and \texttt{@end tex} (see Section 15.3 [Raw Formatter Commands], page 135).
Follow the \texttt{mag} command with an ‘\texttt{=}’ and then a number that is 1000 times the magnification you desire. For example, to print pages at 1.2 normal size, write the following near the beginning of the Texinfo file, before the title page:

\begin{verbatim}
\@tex
  \global\mag=1200
\@end tex
\end{verbatim}

With some printing technologies, you can print normal-sized copies that look better than usual by giving a larger-than-normal master to your print shop. They do the reduction, thus effectively increasing the resolution.

Depending on your system, DVI files prepared with a nonstandard-\texttt{mag} may not print or may print only with certain magnifications. Be prepared to experiment.
Appendix F  Info Format Specification

Here we describe the technical details of the Info format.

In this formal description, the characters `<*>()|=#` are used for the language of the description itself. Other characters are literal. The formal constructs used are typical: `...` indicates a metavariable name, `=` means definition, `*` repetition, `?` optional, `( )` grouping, `|` alternation, and `#` comment.

In general, programs that read Info files should try to be case-insensitive to keywords that occur in the file (for example, `Tag Table` and `Tag table` should be equivalent) in order to support Info-generating programs that use different capitalization.

We specify literal parentheses (those that are part of the Info format) with `<lparen>` and `<rparen>`, meaning the single characters `( )` respectively. Finally, the two-character sequence `^[x]` means the single character `CTRL-x`, for any `x`.

This format definition was written some 25 years after the Info format was first devised. So in the event of conflicts between this definition and actual practice, practice wins. It also assumes some general knowledge of Texinfo; it is meant to be a guide for implementors rather than a rigid technical standard. We may refer back to other parts of this manual for examples and definitions, rather than redundantly spelling out every detail.

F.1 Info Format: A Whole Manual

To begin, an Info manual is either *nonsplit* (contained wholly within a single file) or *split* (across several files).

The syntax for a nonsplit manual is:

```plaintext
<nonsplit info file> =
  <preamble>
  <node>*
  <tag table>?
  <local variables>?
```

When split, there is a *main file*, which contains only pointers to the nodes given in other *subfiles*. The main file looks like this:

```plaintext
<split info main file> =
  <preamble>
  <indirect table>
  <tag table>
  <local variables>?
```

The subfiles in a split manual have the following syntax:

```plaintext
<split info subfile> =
  <preamble>
  <node>*
```

Note that the tag table is not optional for split files, as it is used with the indirect table to deduce which subfile a particular node is in.

Several of the sections in an Info file (such as nodes or tag tables) begin with a sequence:

```plaintext
<separator> = (^L)?^_(^L)?^J
```

That is, a `CTRL-_` character followed by a newline, with optional formfeed characters.
F.2 Info Format: Preamble

The <preamble> is text at the beginning of all output files. It is not intended to be visible by default in an Info viewer, but may be displayed upon user request.

\[<\text{preamble}> = \]
\[<\text{identification}> \quad \# \text{"This is FILENAME, produced by ..."} \]
\[<\text{copying text}> \quad \# \text{Expansion of @copying text.} \]
\[<\text{dir entries}> \quad \# \text{Derived from @dircategory and @direntry.} \]

These pieces are:

<identification line>
An arbitrary string beginning the output file, followed by a blank line.

<copying text>
The expansion of a @copying environment, if the manual has one (see Section 2.7.1 [@copying], page 17).

<dir entries>
The result of any @dircategory and @direntry commands present in the manual (see Section 20.1.4 [Installing Dir Entries], page 196).

F.3 Info Format: Indirect Table

\[<\text{indirect table}> = \]
\[<\text{separator}> \quad \text{Indirect: } \]
\[(<\text{filename}>: <\text{bytepos}>)* \]

The indirect table is written to the main file in the case of split output only. It specifies, as a decimal integer, the starting byte position (zero-based) that the first node of each subfile would have if the subfiles were concatenated together in order, not including the top-level file. The first node of actual content is pointed to by the first entry.

As an example, suppose split output is generated for the GDB manual. The top-level file gdb.info will contain something like this:

\[<\text{separator}> \quad \text{Indirect:} \]
\[\text{gdb.info-1: 1878} \]
\[\text{gdb.info-2: 295733} \]
\[... \]

This tells Info viewers that the first node of the manual occurs at byte 1878 of the file gdb.info-1 (which would be after that file’s preamble.) The first node in the gdb.info-2 subfile would start at byte 295733 if gdb.info-2 were appended to gdb.info-1, including any preamble sections in both files.

Unfortunately, Info-creating programs such as makeinfo have not always implemented these rules perfectly, due to various bugs and oversights. Therefore, robust Info viewers should fall back to searching “nearby” the given position for a node, instead of giving up immediately if the position is not exactly at a node beginning.
F.4 Info Format: Tag Table

\[
<\text{tag table}> = \\
<\text{separator}> \\
\text{Tag Table:} \\
(\langle lparen \text{Indirect} rparen \rangle)? \\
(\text{Node}|\text{Ref}): \langle \text{nodeid} \rangle^?\langle \text{bytepos} \rangle \\
<\text{separator}> \\
\text{End Tag Table}
\]

The ‘(Indirect)’ line appears in the case of split output only.

The tag table specifies the starting byte position of each node and anchor in the file. In the case of split output, it is only written in the main output file.

Each line defines an identifier as either an anchor or a node, as specified. For example, ‘Node: Top^?1647’ says that the node named ‘Top’ starts at byte 1647 while ‘Ref: Overview-Footnote-1^?30045’ says that the anchor named ‘Overview-Footnote-1’ starts at byte 30045. It is an error to define the same identifier both ways.

In the case of nonsplit output, the byte positions simply refer to the location in the output file. In the case of split output, the byte positions refer to an imaginary file created by concatenating all the split files (but not the top-level file). See the previous section.

Here is an example:

\[
^_ \\
\text{Tag Table:} \\
\text{Node: Top^?89} \\
\text{Node: Ch1^?292} \\
^_ \\
\text{End Tag Table}
\]

This specifies a manual with two nodes, ‘Top’ and ‘Ch1’, at byte positions 89 and 292 respectively. Because the ‘(Indirect)’ line is not present, the manual is not split.

Preamble sections or other non-node sections of files do not have a tag table entry.

F.5 Info Format: Local Variables

The local variables section is optional and is currently used to give the encoding information. It may be augmented in the future.

\[
<\text{local variables}> = \\
<\text{separator}> \\
\text{Local Variables:} \\
\text{coding: } <\text{encoding}> \\
\text{End:}
\]

See Section 14.2 [\@documentencoding], page 131.

F.6 Info Format: Regular Nodes

Regular nodes look like this:

\[
<\text{node}> = \\
<\text{separator}>
\]
File: <fn>, Node: <id1>, (Next: <id2>, )? (Prev: <id3>, )? Up: <id4>

<general text, until the next ^_ or end-of-file>

At least one space or tab must be present after each colon and comma, but any number of spaces are ignored. The <id> node identifiers have following format:

```
{id} = (<lparen><infofile><rparen>)?<node-spec>?  
{node-spec} = <nodename> | <del><nodename><del>  
{del} = ^?
```

This <node> defines <id1> in file <fn>, which is typically either 'manualname' or 'manualname.info'. No parenthesized <infofile> component may appear within <id1>.

Each of the identifiers after Next, Prev and Up refer to nodes or anchors within a file. These pointers normally refer within the same file, but ‘(dir)’ is often used to point to the top-level dir file. If an <infofile> component is used then the node name may be omitted, in which case the node identifier refers to the ‘Top’ node within the referenced file.

The Next and Prev pointers are optional. The Up pointer is technically also optional, although most likely this indicates a mistake in the node structuring. Conventionally, the nodes are arranged to form a tree, but this is not a requirement of the format.

Node names containing periods, commas, colons or parentheses can confuse Info readers. If it is necessary to refer to a node whose name contains any of these, the <nodename> should be surrounded by a pair of DEL characters (‘CTRL-?’, character number 127). makeinfo adds these characters when needed in the default case. Note that not all Info readers recognize this syntax. See [Info Node Names Constraints], page 29.

The <general text> of the node can include the special constructs described next.

**F.7 Info Format: Menu**

Conventionally menus appear at the end of nodes, but the Info format places no restrictions on their location.

```
<menu> =  
  * Menu:  
    (<menu entry> | <menu comment>)*
```

The parts of a <menu entry> are also described in Section 3.8.4 [Menu Parts], page 35. They have the same syntax as cross-references, with a leading ‘*’ instead of ‘* (N|n)ote’ (see Section F.10 [Info Format Cross Reference], page 283). Indices extend the menu format to specify the destination line; see Section F.9 [Info Format Printindex], page 282.

A <menu comment> is any line not beginning with ‘*’ that appears either at the beginning of the menu or is separated from a menu entry by one or more blank lines. These comments are intended to be displayed as part of the menu, as-is (see Section 3.8.1 [Writing a Menu], page 33).

**F.8 Info Format: Image**

The @image command results in the following special directive within the Info file (see Section 9.2 [Images], page 85):

```
<image> =
```
The line breaks and indentation in this description are editorial; the whitespace between the different parts of the directive in Info files is arbitrary.

In the strings <image file>, <txt file contents> and <alt text>, ‘”’ is quoted as ‘\n’ and ‘\’ is quoted as ‘\\’. The txt and alt specifications are optional.

The alt value serves the same purpose as in HTML: A prose description of the image. In text-only displays or speech systems, for example, the alt value may be used instead of displaying the (typically graphical) <image file>.

The <txt file contents>, if present, should be taken as an ASCII representation of the image, for possible use on a text-only display.

The format does not prescribe the choice between displaying the <image file>, the <alt text> or the <txt file contents>.

F.9 Info Format: Printindex

An index in Info format is a kind of menu, with an additional directive at the beginning to mark it as an index menu.

<printindex> =
"@H[index]"
* Menu:

@index entry*

The@index entry> items are similar to normal menu entries, but the free-format description is replaced by the line number of where the entries occurs in the text:

@index entry> =
* <entry text>: <node-spec>. <line-spec>

〈line-spec〉 =
〈lparen〉line <lineno〉〈rparen〉

The initial part, ‘<entry text>: <entry node>.’, should be on a single line. The <entry text> is the index term.

Even though <entry text> is followed by a colon, it may itself contain colons, so Info readers should try to include as much of the line as possible in the <entry text>. (However, <entry text> may not contain the DEL characters that may occur in <node-spec>.)

<lineno〉 is an unsigned integer, given relative to the start of the <node-spec>. An optional line break may occur before ‘<line-spec>’.

Here is an example:
"@H[index]"
* Menu:

* thunder: Weather Phenomena. (line 5)
This means that an index entry for ‘thunder’ appears at line 5 of the node ‘Weather Phenomena’.

**F.10 Info Format: Cross-reference**

A general cross-reference in Info format has one of the following two forms:

$$<\text{cross-reference}> =$$

$$* (N|n)ote <id>::$$

$$| * (N|n)ote <label>:<id>(.|,)$$

$$<id> = (<lparen><\text{infofile}><rparen>)?<\text{node-spec}>?$$

$$<label> = <\text{label text}> | <\text{del}<\text{label text}><\text{del}>$$

No space should occur between the ‘*’ character and the following ‘N’ or ‘n’. ‘*Note’ should be used at the start of a sentence, otherwise ‘*note’ should be used. (Some Info readers, such as the one in Emacs, can display ‘*Note’ and ‘*note’ as ‘See’ and ‘see’ respectively.) In both cases, <label text> is descriptive text.

In both forms the <id> refers to a node or anchor, in the same way as a reference in the node information line does (see Section F.6 [Info Format Regular Nodes], page 280). The optional parenthesized ‘<infofile>’ is the name of the manual being referenced, and <node-spec> gives the node or anchor within that manual.

The second form has a descriptive label. A cross-reference in this form should usually be terminated with a comma or period, to make it feasible to find the end of the <id>.

If <label> contains a colon character (:), it should be surrounded with a pair of <del> characters. Likewise, quoting characters may be used for the node name if it contains problematic characters; then a terminating comma or period is not needed. As stated earlier, this quoting mechanism is not supported in all Info-reading programs.

The format does not prescribe how to find other manuals to resolve such references.

Here are some examples:

*note GNU Free Documentation License::

*note Tag table: Info Format Tag Table, for details.

*Note Overview: (make)Top.

*Note `^?:^?: (bash)Bourne Shell Builtins.

*Note alloca.h: (gnulib)`^?alloca.h`^?.

The first shows a reference to a node in the current manual using the short form.

The second also refers to a node in the current manual, namely ‘Info Format Tag Table’; the ‘Tag table’ before the ‘:’ is only a label on this particular reference, and the ‘for details.’ is text belonging to the sentence, not part of the reference.

The third example refers to the node ‘Top’ in another manual, namely ‘make’, with ‘Overview’ being the label for this cross-reference.

The fourth example shows a colon character being quoted in a label, and the fifth example shows a period being quoted in a node name.

See Chapter 5 [Cross References], page 43.
Appendix G  GNU Free Documentation License

Version 1.3, 3 November 2008
https://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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

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

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

1. APPLICABILITY AND DEFINITIONS
This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

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

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

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released
under this License. If a section does not fit the above definition of Secondary then it is
not allowed to be designated as Invariant. The Document may contain zero Invariant
Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover
Texts or Back-Cover Texts, in the notice that says that the Document is released under
this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may
be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a
format whose specification is available to the general public, that is suitable for revising
the document straightforwardly with generic text editors or (for images composed of
pixels) generic paint programs or (for drawings) some widely available drawing editor,
and that is suitable for input to text formatters or for automatic translation to a
variety of formats suitable for input to text formatters. A copy made in an otherwise
Transparent file format whose markup, or absence of markup, has been arranged to
thwart or discourage subsequent modification by readers is not Transparent. An image
format is not Transparent if used for any substantial amount of text. A copy that is
not “Transparent” is called “Opaque”.

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

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

The “publisher” means any person or entity that distributes copies of the Document to
the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either
is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in
another language. (Here XYZ stands for a specific section name mentioned below, such
as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve
the Title” of such a section when you modify the Document means that it remains a
section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that
this License applies to the Document. These Warranty Disclaimers are considered to be
included by reference in this License, but only as regards disclaiming warranties: any
other implication that these Warranty Disclaimers may have is void and has no effect
on the meaning of this License.

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

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

3. COPYING IN QUANTITY

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

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

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

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

4. MODIFICATIONS

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

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

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

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

D. Preserve all the copyright notices of the Document.

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

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

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

H. Include an unaltered copy of this License.

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

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

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

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

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

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

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

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

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

5. COMBINING DOCUMENTS

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

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

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

6. COLLECTIONS OF DOCUMENTS

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

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

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

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

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.
10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free
Documentation License from time to time. Such new versions will be similar in spirit
to the present version, but may differ in detail to address new problems or concerns.
See https://www.gnu.org/licenses/.

Each version of the License is given a distinguishing version number. If the Document
specifies that a particular numbered version of this License “or any later version” applies
to it, you have the option of following the terms and conditions either of that specified
version or of any later version that has been published (not as a draft) by the Free
Software Foundation. If the Document does not specify a version number of this License,
you may choose any version ever published (not as a draft) by the Free Software
Foundation. If the Document specifies that a proxy can decide which future versions
of this License can be used, that proxy’s public statement of acceptance of a version
permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide
Web server that publishes copyrightable works and also provides prominent facilities for
anybody to edit those works. A public wiki that anybody can edit is an example of
such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the
site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published
by Creative Commons Corporation, a not-for-profit corporation with a principal place
of business in San Francisco, California, as well as future copyleft versions of that license
published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of
another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works
that were first published under this License somewhere other than this MMC, and
subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or
invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under
CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is
eligible for relicensing.
ADDENDUM: How to use this License for your documents

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

Copyright (C) year your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts. A copy of the license is included in the section entitled ``GNU
Free Documentation License''.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the
"with...Texts." line with this:

with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.

If you have Invariant Sections without Cover Texts, or some other combination of the
three, merge those two alternatives to suit the situation.

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

This is an alphabetical list of all the @-commands, assorted Emacs Lisp functions, and several variables. To make the list easier to use, the commands are listed without their preceding `@`.

! .......................... 101
! (end of sentence) ............ 101

" ............................ 103
" (umlaut accent) ................. 103

& ............................... 99
& (literal '&') ................... 99

, ................................. 103
, (acute accent) .................. 103

* ................................. 113
* (force line break) .............. 113

, ................................. 103
, (cedilla accent) ................. 103

- ................................. 114
- (discretionary hyphen) ........... 114

. ................................. 101
. (end of sentence) .............. 101

/ ................................. 113
/ (allow line break) .............. 113

: ................................. 113
: (suppress end-of-sentence space) 100

= ................................. 103
= (macron accent) ................. 103

? ................................. 101
? (end of sentence) .............. 101
# Command and Variable Index

<table>
<thead>
<tr>
<th>Command or Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>ak</td>
<td>103</td>
</tr>
<tr>
<td>Aa</td>
<td>103</td>
</tr>
<tr>
<td>abbr</td>
<td>60</td>
</tr>
<tr>
<td>acronym</td>
<td>61</td>
</tr>
<tr>
<td>ae</td>
<td>103</td>
</tr>
<tr>
<td>AE</td>
<td>103</td>
</tr>
<tr>
<td>afivepaper</td>
<td>275</td>
</tr>
<tr>
<td>afourlatex</td>
<td>275</td>
</tr>
<tr>
<td>afourpaper</td>
<td>275</td>
</tr>
<tr>
<td>afourwide</td>
<td>275</td>
</tr>
<tr>
<td>alias</td>
<td>149</td>
</tr>
<tr>
<td>allowcodebreaks</td>
<td>114</td>
</tr>
<tr>
<td>ampchar</td>
<td>90</td>
</tr>
<tr>
<td>anchor</td>
<td>49</td>
</tr>
<tr>
<td>appendix</td>
<td>39</td>
</tr>
<tr>
<td>appendixsec</td>
<td>40</td>
</tr>
<tr>
<td>appendixsection</td>
<td>40</td>
</tr>
<tr>
<td>appendixsubsec</td>
<td>40</td>
</tr>
<tr>
<td>appendixsubsubsec</td>
<td>41</td>
</tr>
<tr>
<td>apply</td>
<td>128</td>
</tr>
<tr>
<td>arrow</td>
<td>111</td>
</tr>
<tr>
<td>asis</td>
<td>79</td>
</tr>
<tr>
<td>atchar</td>
<td>97</td>
</tr>
<tr>
<td>author</td>
<td>19</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>63</td>
</tr>
<tr>
<td>\backslash</td>
<td>98</td>
</tr>
<tr>
<td>backslashchar</td>
<td>98</td>
</tr>
<tr>
<td>bsixpaper</td>
<td>275</td>
</tr>
<tr>
<td>bullet</td>
<td>107</td>
</tr>
<tr>
<td>bye</td>
<td>26</td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>10</td>
</tr>
<tr>
<td>caption</td>
<td>84</td>
</tr>
<tr>
<td>cartouche</td>
<td>73</td>
</tr>
<tr>
<td>center</td>
<td>20</td>
</tr>
<tr>
<td>centerchap</td>
<td>39</td>
</tr>
<tr>
<td>chapheading</td>
<td>39</td>
</tr>
<tr>
<td>chapter</td>
<td>38</td>
</tr>
<tr>
<td>cindex</td>
<td>90</td>
</tr>
<tr>
<td>cite</td>
<td>52</td>
</tr>
<tr>
<td>clear</td>
<td>137</td>
</tr>
<tr>
<td>click</td>
<td>111</td>
</tr>
<tr>
<td>clicksquence</td>
<td>111</td>
</tr>
<tr>
<td>clickstyle</td>
<td>111</td>
</tr>
<tr>
<td>code</td>
<td>55</td>
</tr>
<tr>
<td>codequotebacktick</td>
<td>100</td>
</tr>
<tr>
<td>codequoteforcequote</td>
<td>100</td>
</tr>
<tr>
<td>columnfactions</td>
<td>81</td>
</tr>
<tr>
<td>comma</td>
<td>98</td>
</tr>
<tr>
<td>command</td>
<td>60</td>
</tr>
<tr>
<td>comment</td>
<td>10</td>
</tr>
<tr>
<td>complete_tree_nodesMenus</td>
<td>187</td>
</tr>
</tbody>
</table>

...
direction ........................................... 196
display ............................................. 69
displaymath ......................................... 106
dmn .................................................. 102
docbook ............................................. 135
documentdescription ............................... 205
documentencoding .................................. 131
documentlanguage .................................. 130
dotaccent ............................................ 103
dotless .............................................. 103
dots .................................................. 107

E
email .................................................. 62
\emergencystretch .................................. 163
emph ................................................. 62
end ................................................... 214
enddots ............................................. 107
enumerate .......................................... 77
env ................................................... 59
equiv ................................................ 110
error ............................................... 110
errormsg ........................................... 134
errormsg, and line numbers in \TeX ........................................ 151
euro ................................................. 107
evenfooting ........................................ 272
evenfootingmarks ................................... 274
evenheading ........................................ 272
evenheadingmarks .................................. 274
everyfooting ....................................... 272
everyfootingmarks .................................. 273
everyheading ....................................... 272
everyheadingmarks .................................. 273
example ............................................ 67
exampleindent ..................................... 275
exclamdown ....................................... 103
exdent ............................................. 70
expansion ......................................... 109

F
file .................................................. 59
fill_gaps_in_sectioning ............................. 187
finalout .......................................... 163
findext ............................................ 90
firstparagraphindent .................. 274
float .............................................. 83
flushleft ......................................... 70
flushright ....................................... 70
fnindex ........................................... 90
fn-name .......................................... 118
fonttextsize ...................................... 63
foo ............................................... 99
foobar ........................................... 118, 121
footnote .......................................... 87
footnotestyle .................................... 89
format ............................................ 69
forward-word ...................................... 117
frenchspacing ...................................... 102
ftable ............................................ 80

G
\gdef within \TeX ................................... 135
\geq .............................................. 108
\globaldefs within \TeX ................................... 135
group ............................................. 115
guillemetleft .................................... 104
guillemeright ..................................... 104
guillemotleft ..................................... 104
guillemotright .................................... 104
guilsingleleft .................................... 104
guilsinglright .................................... 104

H
\H (Hungarian umlaut accent) ......................... 103
hashchar .......................................... 98
hbox .............................................. 162
heading .......................................... 40
headings ......................................... 270
headitem ......................................... 81
headitemfont ..................................... 81
html .............................................. 135
hyphenation ...................................... 114

I
i (italic font) ...................................... 64
ifclear ........................................... 139
ifcommanddefined ................................ 140
ifcommandnotdefined ............................. 140
ifdocbook ........................................ 133
ifhtml ........................................... 133
ifinfo ........................................... 133
iflatex ........................................... 133
ifnodocbook ..................................... 134
ifnothtml ........................................ 134
ifnotinfo ........................................ 134
ifnotlatex ....................................... 134
ifnotplaintext .................................. 134
ifnottex ......................................... 134
ifnotxml ......................................... 134
ifplaintext ...................................... 133
ifset ............................................. 138
iftext ............................................ 133
ifxml ............................................ 133
ignore ........................................... 10
image ............................................. 85
include .......................................... 152
indent ............................................ 72
indentedblock .................................... 67
index .............................................. 119
indicateurl ...................................... 62
inforef ........................................... 50
N
need........................................... 116

\newline..................................... 100
next-error................................... 257
node.......................................... 27
noindent...................................... 72
novalidate.................................... 157

O
occurs......................................... 251
occur........................................... 250
occur-mode-goto-occurrence................... 272
oddfooting.................................... 272
oddfootingmarks................................. 274
oddheading..................................... 272
oddfootingmarks................................. 274
oe.............................................. 103
ogonek......................................... 103
option.......................................... 60
ordf............................................ 103
ordm............................................ 103
0.................................................. 103
OE............................................... 103

P
page............................................. 115
page, within @titlepage......................... 19
pagesizes...................................... 276
paragraphindent................................. 274
part............................................. 41
pgindex........................................ 90
phoo............................................ 149
pindex.......................................... 90
point........................................... 110
pounds.......................................... 107
print............................................ 109
printindex...................................... 93
process........................................ 122
push............................................ 121
pxref.......................................... 49

Q
questiondown.................................. 103
quotation...................................... 66
quotedblbase................................. 104
quotedblleft................................. 104
quotedblright................................. 104
quoteleft...................................... 104
quoteright................................. 104
quoting\textbackslash{}base................... 104
Command and Variable Index

R
r (roman font) ........................................... 64
raggedright ............................................. 71
raisesections ........................................... 42
rbracechar ............................................. 97
ref ....................................................... 48
@refill ................................................. 237
regenerate_master_menu .............................. 187
registeredsymbol ........................................ 107
relate_index_entries_to_table_entries ............. 188
result .................................................... 108
ringaccent ............................................. 103
rmacro .................................................... 144

S
samp ...................................................... 57
sansserif ................................................ 64
sc .......................................................... 63
section ................................................... 93
seealso ................................................... 92
seeentry .................................................. 92
set ........................................................ 137
setchapternewpage ..................................... 269
@setcontentsaftertitlepage ......................... 237
setfilename ............................................. 13
@setshortcontentsaftertitlepage .................... 237
settitle ................................................... 14
shortcaption ............................................ 84
shortcontents ........................................... 22
shorttitlepage .......................................... 19
simple_menu ............................................ 188
slanted ................................................. 64
smallbook ............................................... 275
smalldisplay ........................................... 73
smallexample ........................................... 73
smallformat ............................................. 73
smallindentedblock .................................... 73
smalllibp ................................................ 73
smallquotation ......................................... 73
sortas .................................................... 91
sp .......................................................... 115
sp (titlepage line spacing) ......................... 20
<space> ................................................ 100
ss .......................................................... 103
strong ..................................................... 62
sub ........................................................ 105
subentry ................................................... 92
subheading ............................................... 40
subsection ............................................... 40
subsubheading .......................................... 41
subsubsection ........................................... 41
subtitle ................................................... 19
summarycontents ....................................... 22
sup ........................................................ 105
syncodeline ............................................. 94
synindex .................................................. 95

t (typewriter font) .................................... 64
t.test .................................................... 122
tab .......................................................... 81
<tab> ...................................................... 100
table ....................................................... 78
tex .......................................................... 135
texinfo-all-menus-update ............................ 254
texinfo-every-node-update ......................... 254
texinfo-format-buffer ................................. 258
texinfo-format-region ................................. 258
texinfo-indent-menu-description ................. 257
texinfo-insert-braces.................................. 249
texinfo-insert-code .................................... 248
texinfo-insert-dfn ..................................... 248
texinfo-insert-dwim-@ref ............................ 248
texinfo-insert-end ..................................... 248
texinfo-insert-example ............................... 249
texinfo-insert-@item ................................... 248
texinfo-insert-@kbd .................................... 248
texinfo-insert-@node .................................. 248
texinfo-insert-node-lines ......................... 256
texinfo-insert-noindent .............................. 248
texinfo-insert-@samp ................................. 248
texinfo-insert-@table .................................. 248
texinfo-insert-@var .................................... 249
texinfo-make-menu ..................................... 253
texinfo-master-menu .................................. 253
texinfo-multiple-files-update ..................... 255
texinfo-sequential-node-update .................... 257
texinfo-show-structure ................................ 250
texinfo-start-menu-description ........................ 249
texinfo-tex-buffer .................................... 259
texinfo-tex-print ..................................... 259
texinfo-tex-region .................................... 259
texinfo-update-node .................................. 253
textdegree ............................................. 107
TeX .......................................................... 106
th .......................................................... 103
thischapter ............................................. 273
thischaptername ....................................... 273
thischapternum ........................................... 273
thisfile ................................................... 273
thispage ............................................... 272
thissection ............................................ 273
thissectionname ....................................... 272
thissectionnum ........................................... 273
thistitle ............................................... 273
TH .......................................................... 103
tie ........................................................ 115
tieaccent ............................................. 103
tindex ..................................................... 90
title ..................................................... 19
titelfont ................................................ 20
titlepage ............................................... 19
today ...................................................... 273
top ......................................................... 31
tpindex ................................................... 90
**U**

u (breve accent) ........................................ 103
ubaraccent .................................................. 103
udotaccent .................................................. 103
unmacro ....................................................... 144
unnumbered ................................................... 39
unnumberedsec .............................................. 40
unnumberedsubsec ......................................... 40
unnumberedsubsubsec ..................................... 41
up-list ......................................................... 249
uref ............................................................. 50
urefbreakstyle ............................................... 51
\urefurlonlylinktrue ........................................ 52
url .............................................................. 50
\urlcolor ...................................................... 52
U ................................................................. 112

**V**

v (caron) ..................................................... 103
value .......................................................... 137
var ............................................................. 58
verb ........................................................... 58
verbatim ...................................................... 68
verbatiminclude ........................................... 153
vindex .......................................................... 90
vrvindex ....................................................... 90
vskip \TeX vertical skip .................................. 21
vtable ........................................................ 80

**W**

w .............................................................. 114

**X**

xml ............................................................ 135
xref .......................................................... 48
xrefautomaticsectiontitle ................................. 46
General Index

! ......................................................... 103

” ......................................................... 104
" (double low-9 quotation mark) ............ 104
" (undirected double quote character) .... 104

#
‘#line’ directive ............................... 150
   not processing with \TeX ..................... 151
   syntax details ................................. 151

$
$Id expansion, preventing ................. 115

&
‘&\$hex;’, output from @U .................. 112

,
, (single low-9 quotation mark) ............ 104
'' ..................................................... 104

( (dir) as Up node of Top node ............. 30

,
, (single low-9 quotation mark) ............ 104

-- (in image alt string) .................... 86, 114

<
< ..................................................... 104
<< ..................................................... 104

>
> ..................................................... 104
>> ..................................................... 104

? 
i ..................................................... 103

-  
- breakpoint within @code .................. 114

<
i ..................................................... 104
‘’ ..................................................... 104
‘’ ..................................................... 104

@
‘@’ as continuation in definition commands .. 118

8
8-bit characters, in HTML cross-references .. 211

A
\a ..................................................... 103
A4 paper, printing on ....................... 275
A5 paper, printing on ....................... 275
\a ..................................................... 103
\a ..................................................... 103
Abbreviations for keys ....................... 57
Abbreviations, tagging ...................... 60
Abstract of document ....................... 205
Abstract syntax tree 
   representation of documents .............. 7
Accents, inserting ............................ 103
accesskey, 
   customization variable for ............... 179
   in HTML output of menus .................. 34
   in HTML output of nodes .................. 31
<acknowledgements> DocBook tag .......... 39
Acronyms, tagging ............................ 61
Acute accent .................................... 103
Adding a new Info file ...................... 194
Additional output formats ................. 5
--add-once, for install-info ............. 197
Advanced indexing ............................ 92
Advice on writing entries .................. 92
ak ..................................................... 103
\E ..................................................... 103
after, value for @refbreakstyle ........... 51
AFTER_BODY_OPEN .............................. 175
AFTER_SHORT_TOC_LINES ..................... 175
AFTER_TOC_LINES .............................. 175
Aliases, command ............................ 149
--align=column, for install-info .......... 197
Allow line break .............................. 113
All-permissive copying license ............ 245
Alphabetical @-command list ............... 215
Alt attribute for images ................... 86
Ampersand, inserting ....................... 99
### C

- `c var=value` .................................. 169
- `--align=column` for `install-info` ........ 197

### CASE_INSENSITIVE_FILENAMES ............... 175

### CLASSIFICATION ERRORS ...................... 263

### Catching mistakes .......................... 263

### Catcode for comments in TeX ............... 10

### Categories, choosing ........................ 16

### Category codes, of plain TeX ............. 135

- `<caution>` DocBook tag ..................... 66

### Caveats for macro usage .................. 146

### Cedilla accent ............................ 103

### Centimeters .................................. 87

### Chapter structuring ......................... 37

### `chapter>` DocBook tag ................... 39

### CHAPTER_HEADER_LEVEL ..................... 175

### Chapter, formatting one at a time .......... 157

### Character set, declaring .................. 131

### Characterics, printed books or manuals ...... 5

### Characters, basic input .................. 9

### Characters, invalid in node name .......... 29

### ‘@charset’ specification, in CSS files ..... 204

- `check` .................................. 6

### Check accent ................................ 103

### CHECK_HTMLXREF .......................... 175

### CHECK_NORMAL_MENU_STRUCTURE ............ 182

### Checking for badly referenced nodes ...... 266

### Checking for Texinfo commands ........... 140

### Checklist for bug reports ................. 2

### Ciceros ................................... 87

### Circumflex accent ......................... 103

### Click sequences ........................... 111

### CLOSEQUOTE_SYMBOL ....................... 182

### Box with rounded corners .................. 73

### Box, ugly black in hardcopy .............. 163

### Brace-delimited conditional text .......... 136

### Brace-delimited flag conditionals .......... 139

### Braces and argument syntax ................ 215

### Braces, in index entries .................. 158

### Braces, in macro arguments ................ 145

### Braces, inserting .......................... 97

### Braces, when to use ........................ 9

### Breakpoints within URLs ................... 51

### Breaks in a line .......................... 113

### Breaks, within `@code` ..................... 114

### Breve accent ............................... 103

### Buffer formatting and printing ............ 259

### Bugs, reporting ........................... 2

### Bzipped dir files, reading ................. 197

### Badly referenced nodes .................... 266

### BASEFILENAME_LENGTH ....................... 175

### Bastard title page ........................ 19

### Batch formatting for Info .................. 268

### Beebe, Nelson .............................. 4

### before, value for `@urefbreakstyle` ........ 51

### BEFORE_SHORT_TOC_LINES .................. 175

### BEFORE_TOC_LINES .......................... 175

### Beginning line of a Texinfo file .......... 13

### Berry, Karl ............................... 6

### Big points .................................. 87

### BIG_RULE ................................... 175

### Black rectangle in hardcopy ............... 163

### Blank lines ................................ 115

### `<blockquote>` DocBook tag ................. 66

### Body of a macro ............................ 143

### `<body>` text, customizing .................. 175

### BODYTEXT .................................. 175

### Bold font .................................. 63

### Bolio ..................................... 6

### Book characteristics, printed .............. 5

### Book, printing small ....................... 275

### border-pattern of Window ................. 125

### BoTeX ..................................... 6

### Anchors .................................... 49

### Angle quotation marks ...................... 104

### Another Info directory ..................... 195

### `--append-new-sections` for `install-info` .... 197

### Archive: Zip, for EPUB file output .......... 206

### Arguments, repeated and optional .......... 118

### ASCII text output with `--plaintext` ........ 169

### ASCII, source document portability using .... 112

### ASCII_PUNCTUATION ......................... 182

### Aspect ratio of images ..................... 86

### @-commands ................................ 9

### customization variables for ................ 172

### in node names ................................ 29

### list of .................................... 215

### syntax ..................................... 214

### At sign, inserting .......................... 97

### Auk, bird species .......................... 104

### automake, and version info .................. 243

### Automatic pointer creation with `makeinfo` .. 27

### Automatic quoting of commas ................. 144

### for some macros ............................ 144

### Automatically insert nodes, menus .......... 252

### Auxiliary files, omitting ................... 157

### AVOID_MENU_REDUNDANCY ..................... 174
Closing punctuation, and sentence ending ........................................ 111
CM-Super fonts .................................................................................. 104
for LaTeX ......................................................................................... 276
installing .......................................................................................... 162
Code point of Unicode
character, inserting by ................................................................. 117

doxygen\texttt{code}, value for \texttt{#bldinputstyle} ................................ 56
Collapsing whitespace around continuations ................................ 118
Colon in node name ........................................................................... 29
Colon, last in \texttt{INFO\textsc{PATH}} .................................................. 195
\texttt{\small\langle\texttt{colophon}\rangle} DocBook tag ..................................... 39
Colored links, in PDF output .............................................................. 52
Column widths, defining for multitable ............................................. 81
Combining indices ............................................................................. 94
Comma after cross-reference ............................................................ 44
Comma in node name ......................................................................... 29
Comma, in macro arguments .............................................................. 144
Comma, inserting ............................................................................. 98
Command aliases ............................................................................. 149
Command definitions ....................................................................... 119
Command names, indicating .............................................................. 60
Command syntax .............................................................................. 214
\texttt{--command}, for \texttt{texi2dvi} ................................................ 155
\texttt{COMMAND\_LINE\_ENCODING} .................................................. 182
Command-line options of \texttt{texi2html} ........................................... 192
Commands in node names ................................................................. 29
Commands to insert special characters .......................................... 97
Commands using raw Te\LaTeX ......................................................... 135
Commands, inserting them ............................................................... 248
Commands, testing for Texinfo ......................................................... 140
\texttt{--commands\_in\_node\_names} .................................................. 164
Comments ..................................................................................... 10
Comments, in CSS files .................................................................... 205
compatibility, with \texttt{texi2html} ................................................... 186
Compile command for formatting .................................................... 161
\texttt{COMPLEX\_FORMAT\_IN\_TABLE} .................................................. 175
Compressed dir files, reading ............................................................ 197
Computer Modern fonts ................................................................... 132
Conditional commands, inline ........................................................ 136
Conditionally visible text .................................................................. 133
Conditions, nested .......................................................................... 141
Conditions for copying Texinfo ........................................................ 1
\texttt{--conf\_dir=dir} ....................................................................... 165
Configuration, for HTML
cross-manual references ................................................................. 212
Cons, Lionel ..................................................................................... 7, 192
Container directory for EPUB ........................................................... 206
Contents, after title page ................................................................. 237
Contents, table of .............................................................................. 22
\texttt{CONTENTS\_OUTPUT\_LOCATION} .......................................... 175
Contents-like outline of file structure ............................................. 250
Contexts, of \texttt{@-commands} ........................................................ 237
Continuation lines in definition commands .................................. 118
Control keys, specifying ................................................................. 57
Controlling line breaks .................................................................... 113
Conventions for writing definitions ............................................... 127
Conventions, syntactic ..................................................................... 9
\texttt{CONVERT\_TO\_LATEX\_IN\_MATH} .................................................. 176
COPYABLE\_LINKS ............................................................................ 176
Copying conditions .......................................................................... 1
Copying permissions ......................................................................... 16
Copying text, including ..................................................................... 18
Copyright holder for FSF works ....................................................... 17
Copyright page .................................................................................. 21
Copyright symbol ............................................................................ 107
Copyright word, always in English ................................................ 17
Correcting mistakes .......................................................................... 263
Country codes .................................................................................. 130
\texttt{cp (concept) index} .................................................................. 90
\texttt{CPP\_LINE\_DIRECTIVES} .......................................................... 182
Create nodes, menus automatically ............................................... 252
Creating an unsplit file .................................................................... 267
Creating index entries ...................................................................... 91
Creating pointers with \texttt{makeinfo} ............................................ 27
Critical editions ............................................................................... 88
Cross-reference configuration, for HTML ...................................... 212
Cross-reference parts ....................................................................... 43
Cross-reference targets, arbitrary ................................................... 49
Cross-references ............................................................................. 43
Cross-references using \texttt{@inref} ................................................ 50
Cross-references using \texttt{\textbackslash{pxref}} .................................... 49
Cross-references using \texttt{\textbackslash{ref}} ....................................... 48
Cross-references using \texttt{\textbackslash{xref}} ....................................... 48
Cross-references, in HTML output ............................................... 207
Cross-references, in Info format ...................................................... 283
\texttt{.cshrc} initialization file .......................................................... 161
CSS, and HTML output .................................................................... 204
\texttt{--css\_include} ...................................................................... 165
\texttt{--css\_ref} ............................................................................. 165
\texttt{CTRL-\l} .................................................................................. 9
Custom page sizes ............................................................................ 276
Customization variables for \texttt{@-commands} ................................ 172
Customization variables for options .............................................. 173
Customize Emacs package (Development/Docs/Texinfo) .......... 160
Customizing of \LaTeX for Texinfo .................................................... 162

D

\texttt{-D \texttt{var}} ............................................................................. 165
Dash, breakpoint within \texttt{@code} ................................................ 114
Dashes in source ............................................................................... 9
\texttt{DATE\_IN\_HEADER} ................................................................. 176
\texttt{--debug}, for \texttt{install\_info} ................................................ 197
debugging document, with tree representation ................................ 174
Debugging the Texinfo structure .................................................... 263
Debugging with Info formatting ..................................................... 263
Debugging with \LaTeX formatting ................................................... 265
\texttt{DEBUG} .................................................................................. 182
\texttt{\langle\texttt{dedication}\rangle} DocBook tag ..................................... 39
\texttt{DEF\_TABLE} .......................................................................... 176
Default font ...................................................................................... 64
\texttt{DEFAULT\_RULE} .................................................................... 176
Defining indexing entries ............................................................... 91
Defining macros ............................................................................... 143
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining new indices</td>
<td>95</td>
</tr>
<tr>
<td>Defining new Texinfo commands</td>
<td>143</td>
</tr>
<tr>
<td>Definition command headings, continuing</td>
<td>118</td>
</tr>
<tr>
<td>Definition commands</td>
<td>117</td>
</tr>
<tr>
<td>Definition conventions</td>
<td>127</td>
</tr>
<tr>
<td>Definition lists, typesetting</td>
<td>78</td>
</tr>
<tr>
<td>Definition of Info format</td>
<td>278</td>
</tr>
<tr>
<td>Definition template</td>
<td>117</td>
</tr>
<tr>
<td>Definitions grouped together</td>
<td>119</td>
</tr>
<tr>
<td>Degree symbol</td>
<td>107</td>
</tr>
<tr>
<td>--delete, for install-info</td>
<td>197</td>
</tr>
<tr>
<td>Delimiter character, for verbatim</td>
<td>58</td>
</tr>
<tr>
<td>Depth of text area</td>
<td>276</td>
</tr>
<tr>
<td>Description for menu, start</td>
<td>249</td>
</tr>
<tr>
<td>Description of document</td>
<td>205</td>
</tr>
<tr>
<td>--description=text, for install-info</td>
<td>198</td>
</tr>
<tr>
<td>Detailed menu</td>
<td>24</td>
</tr>
<tr>
<td>Details of macro usage</td>
<td>146</td>
</tr>
<tr>
<td>detexinfo</td>
<td>174</td>
</tr>
<tr>
<td>Didot points</td>
<td>87</td>
</tr>
<tr>
<td>Different cross-reference commands</td>
<td>43</td>
</tr>
<tr>
<td>Dimension formatting</td>
<td>102</td>
</tr>
<tr>
<td>Dimensions and image sizes</td>
<td>86</td>
</tr>
<tr>
<td>Dir categories, choosing</td>
<td>16</td>
</tr>
<tr>
<td>dir directory for Info installation</td>
<td>194</td>
</tr>
<tr>
<td>dir file listing</td>
<td>194</td>
</tr>
<tr>
<td>dir file, creating your own</td>
<td>195</td>
</tr>
<tr>
<td>dir files and Info directories</td>
<td>195</td>
</tr>
<tr>
<td>Dir files, compressed</td>
<td>197</td>
</tr>
<tr>
<td>dir, created by install-info</td>
<td>197</td>
</tr>
<tr>
<td>Direct formatting of Info files</td>
<td>262</td>
</tr>
<tr>
<td>--dir-file=name, for install-info</td>
<td>198</td>
</tr>
<tr>
<td>--disable-encoding</td>
<td>165</td>
</tr>
<tr>
<td>Display formatting</td>
<td>69</td>
</tr>
<tr>
<td>Displayed equation, in plain TEX</td>
<td>135</td>
</tr>
<tr>
<td>Displayed equations</td>
<td>106</td>
</tr>
<tr>
<td>distinct, value for @kbdinputstyle</td>
<td>56</td>
</tr>
<tr>
<td>Distorting images</td>
<td>86</td>
</tr>
<tr>
<td>DO_ABOUT</td>
<td>176</td>
</tr>
<tr>
<td>DOC_ENCODING_FOR_INPUT_FILE_NAME</td>
<td>182</td>
</tr>
<tr>
<td>DOC_ENCODING_FOR_OUTPUT_FILE_NAME</td>
<td>183</td>
</tr>
<tr>
<td>--docbook</td>
<td>165</td>
</tr>
<tr>
<td>DocBook and prefatory sections</td>
<td>39</td>
</tr>
<tr>
<td>DocBook output, overview</td>
<td>4</td>
</tr>
<tr>
<td>DocBook, including raw</td>
<td>135</td>
</tr>
<tr>
<td>DOC_TYPE</td>
<td>183</td>
</tr>
<tr>
<td>Document description</td>
<td>205</td>
</tr>
<tr>
<td>Document input encoding</td>
<td>131</td>
</tr>
<tr>
<td>Document language, declaring</td>
<td>130</td>
</tr>
<tr>
<td>Document permissions</td>
<td>16</td>
</tr>
<tr>
<td>Document strings, internationalization of</td>
<td>188</td>
</tr>
<tr>
<td>Document strings, translation of</td>
<td>130</td>
</tr>
<tr>
<td>Document structure, of Texinfo</td>
<td>31</td>
</tr>
<tr>
<td>Document title, specifying</td>
<td>14</td>
</tr>
<tr>
<td>--document-language</td>
<td>165</td>
</tr>
<tr>
<td>documentlanguage customization variable</td>
<td>188</td>
</tr>
<tr>
<td>Dot accent</td>
<td>103</td>
</tr>
<tr>
<td>Dotless i, j</td>
<td>103</td>
</tr>
<tr>
<td>Dots, inserting</td>
<td>107</td>
</tr>
<tr>
<td>Double angle quotation marks</td>
<td>104</td>
</tr>
<tr>
<td>Double guillemets</td>
<td>104</td>
</tr>
<tr>
<td>Double left-pointing angle quotation mark</td>
<td>104</td>
</tr>
<tr>
<td>Double low-9 quotation mark</td>
<td>104</td>
</tr>
<tr>
<td>Double quotation marks</td>
<td>104</td>
</tr>
<tr>
<td>Double right-pointing angle quotation mark</td>
<td>104</td>
</tr>
<tr>
<td>Double structure, of Texinfo documents</td>
<td>31</td>
</tr>
<tr>
<td>Double-colon menu entries</td>
<td>36</td>
</tr>
<tr>
<td>--dry-run, for install-info</td>
<td>198</td>
</tr>
<tr>
<td>DTD, for Texinfo XML</td>
<td>4</td>
</tr>
<tr>
<td>Dumas, Patrice</td>
<td>7, 207</td>
</tr>
<tr>
<td>DUMP_TEXI</td>
<td>183</td>
</tr>
<tr>
<td>DUMP_TREE</td>
<td>183</td>
</tr>
<tr>
<td>--dvi</td>
<td>165</td>
</tr>
<tr>
<td>DVI file</td>
<td>156</td>
</tr>
<tr>
<td>DVI output, overview</td>
<td>3</td>
</tr>
<tr>
<td>DVI, output in</td>
<td>154</td>
</tr>
<tr>
<td>--dvipdf</td>
<td>165</td>
</tr>
<tr>
<td>--dvipdf, for texi2dvi</td>
<td>155</td>
</tr>
<tr>
<td>dvipdfmx</td>
<td>155</td>
</tr>
<tr>
<td>dvips</td>
<td>155</td>
</tr>
<tr>
<td>dvips (program)</td>
<td>3</td>
</tr>
<tr>
<td>δ</td>
<td>103</td>
</tr>
<tr>
<td>D</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>--e limit</td>
<td>166</td>
</tr>
<tr>
<td>--E file</td>
<td>167</td>
</tr>
<tr>
<td>EC fonts</td>
<td>104</td>
</tr>
<tr>
<td>EC fonts, installing</td>
<td>162</td>
</tr>
<tr>
<td>Ellipsis, inserting</td>
<td>107</td>
</tr>
<tr>
<td>Em dash, compared to minus sign</td>
<td>108</td>
</tr>
<tr>
<td>Em dash, producing</td>
<td>9</td>
</tr>
<tr>
<td>Emacs</td>
<td>247</td>
</tr>
<tr>
<td>Emacs shell, format, print from</td>
<td>159</td>
</tr>
<tr>
<td>Emphasizing text</td>
<td>62</td>
</tr>
<tr>
<td>Emphasizing text, font for</td>
<td>62</td>
</tr>
<tr>
<td>En dash, producing</td>
<td>9</td>
</tr>
<tr>
<td>enable</td>
<td>124</td>
</tr>
<tr>
<td>ENABLE_ENCODING</td>
<td>165</td>
</tr>
<tr>
<td>--enable-encoding</td>
<td>165</td>
</tr>
<tr>
<td>Encoding</td>
<td></td>
</tr>
<tr>
<td>input file names</td>
<td>184</td>
</tr>
<tr>
<td>output file names</td>
<td>185</td>
</tr>
<tr>
<td>Encoding, declaring</td>
<td>131</td>
</tr>
<tr>
<td>‘end’ node footnote style</td>
<td>88</td>
</tr>
<tr>
<td>End of header line</td>
<td>15</td>
</tr>
<tr>
<td>End titlepage starts headings</td>
<td>22</td>
</tr>
<tr>
<td>Ending a Sentence</td>
<td>101</td>
</tr>
<tr>
<td>Ending a Texinfo file</td>
<td>26</td>
</tr>
<tr>
<td>Entity reference in HTML et al.</td>
<td>112</td>
</tr>
<tr>
<td>Entries for an index</td>
<td>91</td>
</tr>
<tr>
<td>Entries, making index</td>
<td>92</td>
</tr>
<tr>
<td>--entry=text, for install-info</td>
<td>198</td>
</tr>
<tr>
<td>Enumeration</td>
<td>77</td>
</tr>
<tr>
<td>Environment indentation</td>
<td>275</td>
</tr>
</tbody>
</table>
Environment variable INFOPATH ........................................ 195
Environment variable TEXINFO_OUTPUT_FORMAT ............. 170
Environment variable TEXINPUTS .................................. 162
epsf image format .................................................. 85
espsf.tex ............................................................. 87
espsf.tex, installing ................................................ 162
--epub3 ................................................................ 165
EPUB 3 output, overview .......................................... 3
EPUB Container directory ........................................... 206
EPUB output file ....................................................... 206
EPUB, generating ..................................................... 206
EPUB_CREATE_CONTAINER_FILE .................................. 183
avoiding Archive::Zip dependency .............................. 206
EPUB_KEEP_CONTAINER_FOLDER ................................ 183
Equation, displayed, in plain TEX .............................. 135
Equations, displayed ................................................ 106
Equivalence, indicating .............................................. 110
Error message, indicating ......................................... 110
Error messages, line numbers in .................................. 150
ERROR_LIMIT ........................................................... 166
--error-limit=limit .................................................. 166
Errors, parsing ......................................................... 257
Escaping to HTML ..................................................... 203
Es-zet ................................................................. 103
etex ................................................................. 156
Eth ................................................................. 103
Euro font .............................................................. 107
Euro font, installing ................................................ 162
Euro symbol, and encodings ...................................... 131
Euro symbol, producing .......................................... 107
European A4 paper ................................................... 275
European Computer Modern fonts ................................ 104
installing ............................................................ 162
Evaluation glyph ..................................................... 108
Example indentation ................................................ 275
Example menu ......................................................... 34
example, value for @&#126;inputstyle#126; ..................... 56
Examples in smaller fonts ......................................... 73
Examples, formatting them ....................................... 67
Examples, glyphs for ................................................. 108
Expanding macros .................................................... 144
Expansion of 8-bit characters in HTML cross-references .. 211
Expansion of macros, contexts for ................................ 146
Expansion, indicating ............................................... 109
expansion, of node names in HTML cross-references ... ... 208
Expressions in a program, indicating ......................... 55
EXTENSION .............................................................. 183
External macro processors ........................................ 150
EXTERNAL_CROSSREF_EXTENSION .............................. 176
EXTERNAL_CROSSREF_SPLIT ..................................... 176
EXTERNAL_DIR .......................................................... 176
EXTRA_HEAD ............................................................ 176

F

-f width .............................................................. 166
Family names, in all capitals .................................... 61
Features of Texinfo, adapting to ................................ 140
Feminine ordinal ..................................................... 103
feynman 10 ............................................................ 107
installing .............................................................. 162
file recorder for TeX ............................................... 156
--fill-column=width ............................................... 166
fill TeX dimension .................................................. 21
-F ................................................................ 166
File ending ............................................................ 26
File name collision ................................................... 14
File sectioning structure, showing ......................... 250
FILLCOLUMN ........................................................... 166
Final output ........................................................... 162
Finding badly referenced nodes ......................... ... ... 266
Fine-tuning, and hyphenation .................................. 114
First line of a Texinfo file ....................................... 13
First node .............................................................. 30
First paragraph, suppressing indentation of .......... 274
Fixed-width font ....................................................... 64
Flag conditionals, brace-delimited ..................... 139
Float environment ................................................... 83
Floating accents, inserting ...................................... 103
Floating, not yet implemented .................................. 83
Floats ................................................................. 83
list of .................................................................. 85
making unnumbered ............................................... 83
numbering of ......................................................... 84
Flooding ............................................................... 49
fn (function) index ................................................... 90
Font for multitable heading rows ....................... 81
Font size, reducing .................................................. 63
Fonts for indices ...................................................... 95
Fonts for printing ..................................................... 63
Footings ............................................................... 270
Footnote style, 'end' .................................................. 88
'separate' ............................................................. 88
FOOTNOTE_END_HEADER_LEVEL ............................... 176
FOOTNOTE_SEPARATE_HEADER_LEVEL ................... 176
Footnotes .............................................................. 87
footnotestyle ........................................................ 166
--footnote-style=style ............................................. 166
--force ............................................................... 166
Force line break ..................................................... 113
FORCE ................................................................. 166
Forcing indentation .................................................. 72
Forcing line and page breaks ................................. 113
Form feed character ............................................... 9
Format a dimension ............................................... 102
Format and print hardcopy ...................................... 154
Format and print in Texinfo mode ....................... 159
Format with the compile command .................... 161
Format, print from Emacs shell ............................. 159
FORMATMENU ........................................................... 183
Formats for images ................................................ 85
General Index

Formatting commands .................................. 9
Formatting examples .................................. 67
Formatting for Info ................................. 257
Formatting for printing ............................ 259
Formatting headings and footings ............. 270
Formatting partial documents .................. 157
Formatting requirements .......................... 161
Formatting with \text and \texindex ............ 156
Formulas, mathematical .......................... 105
Four- and five argument forms of
  cross-references ................................ 46
Fox, Brian ........................................... 6
FRAMES ........................................... 177
\FRAMESET\_DOCTYPE .................. 177
Free Documentation License,
  including entire ................................ 243
Free software ..................................... 1
Free Software Directory ......................... 16
French quotation marks .......................... 104
French spacing .................................... 102
Frequently used commands, inserting ..... 248
Frontmatter, text in ............................ 24
Full texts, GNU ................................. 243
Function definitions .............................. 119
Functions, in typed languages .............. 120
Future of Texinfo implementations ........ 8

G
General syntactic conventions .................... 9
Generating EPUB .................................. 206
Generating HTML .................................. 203
Generating menus with indices ................ 93
Generating page headings ......................... 22
Generating plain text files with
  \no\_headers .................................. 167
Generating plain text files with \plaintext 169
German quotation marks .......................... 104
German S ......................................... 103
Global Document Commands .................... 269
Globbing ......................................... 157
Glyphs for programming ........................ 108
Glyphs for text .................................. 106
GNU Emacs ....................................... 247
GNU Emacs shell, format, print from ...... 159
GNU Free Documentation License,
  including entire ................................ 243
GNU sample texts ................................ 243
Going to other Info files' nodes ............ 36
Grave accent ....................................... 103
  standalone .................................... 100
  vs. left quote ................................ 104
Group (hold text together vertically) ...... 115
Grouping two definitions together .......... 119
GUI click sequence ............................... 111
Guillemets ........................................ 104
Guillemots ........................................ 104

H
-h .................................................. 166
Hacek accent ...................................... 103
Hardcopy, printing it ........................... 154
Hash sign, inserting ............................. 98
'hbox', overfull ................................ 162
\<head\> HTML tag, and \<link\> ............. 180
Header for Texinfo files ......................... 12
Header of a Texinfo file ......................... 13
\HEADER\_IN\_TABLE ......................... 177
HEADERS ......................................... 167
Heading row, in table ........................... 81
Headings ......................................... 270
Headings, indentation after ................. 274
Headings, page, begin to appear ........... 22
Height of images ................................ 86
Height of text area .............................. 276
--help, for texi2any ......................... 166
--help, for texindex .......................... 198
help2man ......................................... 6
Hierarchical documents, and menus ....... 34
Highlighting text ................................ 54
Highlighting, customized ..................... 149
Hints ............................................. 238
History of Texinfo ............................... 6
Holder of copyright for FSF works .......... 17
Holding text together vertically .......... 115
href, producing HTML ......................... 150
--html .......................................... 166
HTML cross-references ......................... 207
8-bit character expansion ................... 211
command expansion ......................... 209
configuration .................................. 212
link basics ..................................... 207
mismatch ....................................... 211
node name expansion ......................... 208
HTML output, and encodings ................. 207
HTML output, browser compatibility of ... 203
HTML output, overview ......................... 3
HTML output, split ............................ 204
HTML translation ............................... 203
HTML, and CSS .................................. 204
HTML, including raw ......................... 135
\HTML\_MATH .................................. 177
\HTML\_ROOT\_ELEMENT\_ATTRIBUTES .... 177
htmlxref.cnf .................................. 212
HTMLXREF\_FILE ............................... 177
HTMLXREF\_MODE ............................... 177
http-equiv, and charset specification .... 131
Hungarian umlaut accent ...................... 103
Hurricanes ....................................... 48
Hyphen, breakpoint within \code .......... 114
Hyphen, compared to minus .................. 108
Hyphenation patterns, language-dependent .. 130
Hyphenation, helping \TeX do ............... 114
Hyphenation, preventing ...................... 115
Hyphens in source, two or three in a row .. 9
<table>
<thead>
<tr>
<th>Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>103</td>
</tr>
<tr>
<td>Indices, defining new</td>
<td>166</td>
</tr>
<tr>
<td>Indices, printing and menus</td>
<td>188</td>
</tr>
<tr>
<td>Icelandic</td>
<td>103</td>
</tr>
<tr>
<td>ICONS</td>
<td>177</td>
</tr>
<tr>
<td>If text conditionally visible</td>
<td>133</td>
</tr>
<tr>
<td>--ifdoctool</td>
<td>166</td>
</tr>
<tr>
<td>--ifhtml</td>
<td>166</td>
</tr>
<tr>
<td>--ifinfo</td>
<td>166</td>
</tr>
<tr>
<td>--iflatex</td>
<td>166</td>
</tr>
<tr>
<td>--ifplaintext</td>
<td>166</td>
</tr>
<tr>
<td>--iftext</td>
<td>166</td>
</tr>
<tr>
<td>--ifxml</td>
<td>166</td>
</tr>
<tr>
<td>IGNORE_REF_TO_TOP_NODE_UP</td>
<td>178</td>
</tr>
<tr>
<td>IGNORE_SPACE_AFTER_BRACED_COMMAND_NAME</td>
<td>183</td>
</tr>
<tr>
<td>Ignored before @setfilename</td>
<td>13</td>
</tr>
<tr>
<td>Ignored text</td>
<td>10</td>
</tr>
<tr>
<td>Image formats</td>
<td>85</td>
</tr>
<tr>
<td>IMAGE_LINK_PREFIX</td>
<td>177</td>
</tr>
<tr>
<td>Images</td>
<td>85</td>
</tr>
<tr>
<td>alternate text for</td>
<td>86</td>
</tr>
<tr>
<td>in Info format</td>
<td>281</td>
</tr>
<tr>
<td>scaling</td>
<td>86</td>
</tr>
<tr>
<td>Implementation, texi2any as reference</td>
<td>8</td>
</tr>
<tr>
<td>Implicit pointer creation with makeinfo</td>
<td>27</td>
</tr>
<tr>
<td>‘import’ specifications, in CSS files</td>
<td>205</td>
</tr>
<tr>
<td>&lt;important&gt; DocBook tag</td>
<td>66</td>
</tr>
<tr>
<td>Inches</td>
<td>87</td>
</tr>
<tr>
<td>Include file sample</td>
<td>152</td>
</tr>
<tr>
<td>@include file sample</td>
<td>152</td>
</tr>
<tr>
<td>Include files</td>
<td>152</td>
</tr>
<tr>
<td>Include files requirements</td>
<td>256</td>
</tr>
<tr>
<td>Include files, and section levels</td>
<td>42</td>
</tr>
<tr>
<td>Including a file verbatim</td>
<td>153</td>
</tr>
<tr>
<td>Including permissions text</td>
<td>18</td>
</tr>
<tr>
<td>Indentation undoing</td>
<td>70</td>
</tr>
<tr>
<td>Indentation, forcing</td>
<td>72</td>
</tr>
<tr>
<td>Indentation, omitting</td>
<td>72</td>
</tr>
<tr>
<td>Indented text block</td>
<td>67</td>
</tr>
<tr>
<td>Indenting environments</td>
<td>275</td>
</tr>
<tr>
<td>Indenting paragraphs, control of</td>
<td>274</td>
</tr>
<tr>
<td>Indenting, suppressing of first paragraph</td>
<td>274</td>
</tr>
<tr>
<td>Index entries</td>
<td>91</td>
</tr>
<tr>
<td>Index entries, advice on writing</td>
<td>92</td>
</tr>
<tr>
<td>Index entries, making</td>
<td>92</td>
</tr>
<tr>
<td>Index file names</td>
<td>156</td>
</tr>
<tr>
<td>Index font types</td>
<td>91</td>
</tr>
<tr>
<td>index sorting</td>
<td>91</td>
</tr>
<tr>
<td>INDEX_ENTRY_COLON</td>
<td>177</td>
</tr>
<tr>
<td>INDEX_SPECIAL_CHARS_WARNING</td>
<td>184</td>
</tr>
<tr>
<td>Indexing table entries automatically</td>
<td>80</td>
</tr>
<tr>
<td>Indexing, advanced</td>
<td>92</td>
</tr>
<tr>
<td>Indicating commands, definitions, etc</td>
<td>54</td>
</tr>
<tr>
<td>Indicating evaluation</td>
<td>108</td>
</tr>
<tr>
<td>Indices</td>
<td>90</td>
</tr>
<tr>
<td>Indices, combining them</td>
<td>94</td>
</tr>
<tr>
<td>Indices, defining new</td>
<td>95</td>
</tr>
<tr>
<td>Page</td>
<td>Topic</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>2</td>
<td>Introduction to Texinfo</td>
</tr>
<tr>
<td>29</td>
<td>Invalid characters in node names</td>
</tr>
<tr>
<td>144</td>
<td>Invoking macros</td>
</tr>
<tr>
<td>197</td>
<td>Invoking nodes, including in dir file</td>
</tr>
<tr>
<td>189</td>
<td>Invoking pod2text</td>
</tr>
<tr>
<td>130</td>
<td>ISO 3166 country codes</td>
</tr>
<tr>
<td>130</td>
<td>ISO 639-2 language codes</td>
</tr>
<tr>
<td>64</td>
<td>Italic font</td>
</tr>
<tr>
<td>198</td>
<td>--item=text, for install-info</td>
</tr>
<tr>
<td>76</td>
<td>Itemization</td>
</tr>
<tr>
<td>103</td>
<td>(dotless \i)</td>
</tr>
<tr>
<td>103</td>
<td>jpeg image format</td>
</tr>
<tr>
<td>178</td>
<td>JS_WEUBLABELS</td>
</tr>
<tr>
<td>178</td>
<td>JS_WEUBLABELSFLE</td>
</tr>
<tr>
<td>198</td>
<td>--keep-old, for install-info</td>
</tr>
<tr>
<td>56</td>
<td>Keyboard input</td>
</tr>
<tr>
<td>57</td>
<td>Keys, recommended names</td>
</tr>
<tr>
<td>115</td>
<td>Keyword expansion, preventing</td>
</tr>
<tr>
<td>55</td>
<td>Keywords, indicating</td>
</tr>
<tr>
<td>5</td>
<td>Knuth, Donald</td>
</tr>
<tr>
<td>90</td>
<td>ky (keystroke) index</td>
</tr>
<tr>
<td>103</td>
<td>L</td>
</tr>
<tr>
<td>181</td>
<td>L2H_CLEAN</td>
</tr>
<tr>
<td>181</td>
<td>L2H_FILE</td>
</tr>
<tr>
<td>181</td>
<td>L2H_HTML_VERSION</td>
</tr>
<tr>
<td>181</td>
<td>L2H_L2H</td>
</tr>
<tr>
<td>175</td>
<td>lang, HTML attribute</td>
</tr>
<tr>
<td>130</td>
<td>Language codes</td>
</tr>
<tr>
<td>130</td>
<td>Language, declaring</td>
</tr>
<tr>
<td>155</td>
<td>--language, for teix2dvi</td>
</tr>
<tr>
<td>276</td>
<td>Larger or smaller pages</td>
</tr>
<tr>
<td>167</td>
<td>--latex</td>
</tr>
<tr>
<td>106</td>
<td>I\TeX\ logo</td>
</tr>
<tr>
<td>4</td>
<td>LaTeX output, overview</td>
</tr>
<tr>
<td>135</td>
<td>I\TeX, including raw</td>
</tr>
<tr>
<td>155</td>
<td>I\TeX, processing with teix2dvi</td>
</tr>
<tr>
<td>104</td>
<td>Left quotation marks</td>
</tr>
<tr>
<td>104</td>
<td>Left-pointing angle quotation marks</td>
</tr>
<tr>
<td>276</td>
<td>Legal paper, printing on</td>
</tr>
<tr>
<td>14</td>
<td>Length of file names</td>
</tr>
<tr>
<td>36</td>
<td>Less cluttered menu entry</td>
</tr>
<tr>
<td>188</td>
<td>Libintl-perl Gettext implementation</td>
</tr>
<tr>
<td>1</td>
<td>Libre software</td>
</tr>
<tr>
<td>245</td>
<td>License for all-permissive copying</td>
</tr>
<tr>
<td>245</td>
<td>License for verbatim copying</td>
</tr>
<tr>
<td>2</td>
<td>Limited scope of Texinfo</td>
</tr>
<tr>
<td>113</td>
<td>Line breaks, awkward</td>
</tr>
<tr>
<td>113</td>
<td>Line breaks, controlling</td>
</tr>
<tr>
<td>51</td>
<td>Line breaks, preventing</td>
</tr>
<tr>
<td>81</td>
<td>Line length, column widths as fraction of</td>
</tr>
<tr>
<td>150</td>
<td>Line numbers, in error messages</td>
</tr>
<tr>
<td>115</td>
<td>Line spacing</td>
</tr>
<tr>
<td>180</td>
<td>&lt;link&gt; HTML tag, in &lt;head&gt;</td>
</tr>
<tr>
<td>52</td>
<td>Links, coloring in PDF output</td>
</tr>
<tr>
<td>69</td>
<td>Lisp example</td>
</tr>
<tr>
<td>73</td>
<td>Lisp examples in smaller fonts</td>
</tr>
<tr>
<td>215</td>
<td>List of @-commands</td>
</tr>
<tr>
<td>85</td>
<td>List of floats</td>
</tr>
<tr>
<td>194</td>
<td>Listing a new Info file</td>
</tr>
<tr>
<td>75</td>
<td>Lists and tables, making</td>
</tr>
<tr>
<td>158</td>
<td>Literate programming, with Texinfo and awk</td>
</tr>
<tr>
<td>280</td>
<td>Local variable section, in Info format</td>
</tr>
<tr>
<td>161</td>
<td>Local variables</td>
</tr>
<tr>
<td>131</td>
<td>Local Variables section, for encoding</td>
</tr>
<tr>
<td>130</td>
<td>Locale, declaring</td>
</tr>
<tr>
<td>184</td>
<td>LOCALT_ENCODING</td>
</tr>
<tr>
<td>35</td>
<td>Location of menus</td>
</tr>
<tr>
<td>106</td>
<td>Logos, \TeX</td>
</tr>
<tr>
<td>186</td>
<td>Longest nodes, finding</td>
</tr>
<tr>
<td>266</td>
<td>Looking for badly referenced nodes</td>
</tr>
<tr>
<td>42</td>
<td>Lowering and raising sections</td>
</tr>
<tr>
<td>158</td>
<td>\pr (DVI print command)</td>
</tr>
<tr>
<td>158</td>
<td>\pr-d, replacements on MS-DOS/MS-Windows</td>
</tr>
<tr>
<td>197</td>
<td>Lzip-compressed dir files, reading</td>
</tr>
<tr>
<td>197</td>
<td>LZMA-compressed dir files, reading</td>
</tr>
<tr>
<td>119</td>
<td>Macro definitions, programming-language</td>
</tr>
<tr>
<td>143</td>
<td>Macro definitions, Texinfo</td>
</tr>
<tr>
<td>146</td>
<td>Macro details</td>
</tr>
<tr>
<td>146</td>
<td>Macro expansion, contexts for</td>
</tr>
<tr>
<td>109</td>
<td>Macro expansion, indicating</td>
</tr>
<tr>
<td>144</td>
<td>Macro invocation</td>
</tr>
<tr>
<td>143</td>
<td>Macro names, valid characters in</td>
</tr>
<tr>
<td>150</td>
<td>Macro processors, external</td>
</tr>
<tr>
<td>167</td>
<td>\macroexpand= file</td>
</tr>
<tr>
<td>103</td>
<td>Macron accent</td>
</tr>
<tr>
<td>143</td>
<td>Macros</td>
</tr>
<tr>
<td>144</td>
<td>Macros, undefining</td>
</tr>
<tr>
<td>276</td>
<td>Magnified printing</td>
</tr>
<tr>
<td>62</td>
<td>Mailto link</td>
</tr>
<tr>
<td>164</td>
<td>makeinfo</td>
</tr>
<tr>
<td>257</td>
<td>makeinfo inside Emacs</td>
</tr>
<tr>
<td>164</td>
<td>makeinfo options</td>
</tr>
<tr>
<td>154</td>
<td>Making a printed manual</td>
</tr>
<tr>
<td>262</td>
<td>Making a tag table manually</td>
</tr>
<tr>
<td>43</td>
<td>Making cross-references</td>
</tr>
<tr>
<td>113</td>
<td>Making line and page breaks</td>
</tr>
<tr>
<td>75</td>
<td>Making lists and tables</td>
</tr>
</tbody>
</table>
Man page output, not supported .......................... 6
Man page, reference to ................................ 50
Manual characteristics, printed .......................... 5
Manual, referring to as a whole .......................... 47
Margins on page, not controllable ........................ 276
Marking text within a paragraph .......................... 54
Marking words and phrases ................................. 54
Masculine ordinal ......................................... 103
Master menu ................................................ 24
Math output for HTML ....................................... 105
Mathematical expressions, inserting ....................... 105
MATHJAX_SCRIPT ............................................ 181
MATHJAX_SOURCE ............................................ 181
MAX_HEADER_LEVEL .......................................... 178
MAX_MACRO_CALL_NESTING ................................ 184
--max-width=column, for install-info ................... 198
Menu description, start ................................... 249
Menu entries with two colons ............................... 36
Menu example .............................................. 34
Menu location .............................................. 35
@menu parts ................................................ 35
Menu parts ................................................ 35
Menu writing ................................................ 33
Menu, master ............................................... 24
MENU_ENTRY_COLON .......................................... 178
MENU_SYMBOL ............................................... 178
--menuentry=text, for install-info ...................... 198
Menus ......................................................... 33
Menus generated with indices ............................... 93
Menus, in Info format ..................................... 281
Menus, omitting with --no-headers ......................... 167
Menus, omitting with --plaintext ........................ 169
MESSAGE_ENCODING .......................................... 184
Meta keys, specifying ..................................... 57
<meta> HTML tag, and charset specification .......... 131
<meta> HTML tag, and document description .............. 205
META key .................................................... 57
Meta-syntactic chars for arguments ....................... 118
Methods, object-oriented .................................. 126
Millimeters .................................................. 87
Mils, argument to @need ................................... 116
Minimal requirements for formatting ..................... 161
Minimal Texinfo file (requirements) ...................... 10
Minus sign .................................................. 108
Mismatched HTML cross-reference source and target .... 211
Mistakes, catching .......................................... 263
Mode, using Texinfo ........................................ 247
Monolithic manuals, for HTML cross-references ......... 213
MONOLITHIC ................................................ 178
Monospace font .............................................. 64
Multiple dashes in source .................................. 9
Multiple spaces ............................................. 100
Multitable column widths .................................. 81
Multitable rows ............................................. 81

N

--name=text, for install-info ......................... 198
Names for indices ......................................... 94
Names of index files ....................................... 156
Names of macros, valid characters of .................... 143
Names recommended for keys ................................ 57
NASA, as acronym .......................................... 61
Navigation bar, in HTML output .......................... 203
Navigation footer ........................................... 180
Navigation links, omitting ................................ 167
Navigation panel, bottom of page ......................... 180
Need space at page bottom .................................. 116
Nested footnotes ............................................ 88
Nesting conditions .......................................... 141
New index defining ......................................... 95
New Info file, listing it in dir file ....................... 194
New Texinfo commands, defining ......................... 143
Newlines, avoiding in conditionals ....................... 136
NEWS file for Texinfo ..................................... 141
Next node of Top node ..................................... 30
NLS .......................................................... 6
NO_CSS ....................................................... 178
NO_CUSTOM_HTML_ATTRIBUTE .............................. 178
NO_NUMBER_FOOTNOTE_SYMBOL ............................ 179
NO_TOP_NODE_OUTPUT .................................... 184
NO_USE_SETFILENAME ....................................... 185
NO_WARN ..................................................... 168
Node line requirements ..................................... 29
Node line writing ........................................... 27
@node line writing ......................................... 27
Node name expansion, in HTML cross-references ......... 208
Node names must be unique ................................ 29
Node names, choosing ....................................... 28
Node names, invalid characters in ......................... 29
Node separators, omitting with --no-headers ............. 167
Node separators, omitting with --plaintext .............. 169
Node, defined .............................................. 27
Node, 'Top' .................................................. 23
NODE_FILES ................................................ 167
NODE_NAME_IN_INDEX ...................................... 185
NODE_NAME_IN_MENU ....................................... 185
--node-files .............................................. 167
--node-files, and HTML cross-references ............... 213
Nodes in other Info files .................................. 36
Nodes, catching mistakes ................................... 263
Nodes, checking for badly referenced ..................... 266
Nodes, deleting or renaming ................................ 50
--no-headers .............................................. 167
--no-ifdocbook ........................................... 167
--no-ifhtml .............................................. 167
--no-ifinfo ............................................... 167
--no-iflatex .............................................. 167
--no-ifplaintext ......................................... 167
--no-iftex ................................................. 167
--no-ifxml ............................................... 167
--no-indent, for install-info ............................ 199
Non-breakable space, fixed ........................................... 114
Non-breakable space, variable ...................................... 115
none, value for @refbreakstyle ...................................... 51
--no-node-files .................................................... 167
 nonsplit Info file ................................................. 199
 Non-split manuals, Info format of .......................... 278
--no-number-footnotes ........................................... 168
--no-number-sections ............................................ 168
--no-pointer-validate ............................................ 168
 Normalization Form C, Unicode .................................. 211
--no-split .......................................................... 170
 Not ending a sentence ............................................ 100
<note> DocBook tag .................................................. 66
--no-validate ....................................................... 168, 173
--no-warn ............................................................ 168
 Number sign, inserting ............................................. 98
NUMBER_FOOTNOTES .................................................. 168
NUMBER_SECTIONS .................................................... 168
Numbering of floats ................................................. 84
--number-sections .................................................. 168

O

-o file ......................................................... 168
ø ........................................................................ 103
O'Dea, Brendan ....................................................... 6
Œ ........................................................................ 103
Object-oriented programming ...................................... 124
Oblique font ............................................................ 64
Obtaining TeX .......................................................... 154
Occurrences, listing with @occur ................................ 251
Octotherp, inserting ................................................... 98
œ ........................................................................ 103
Œ ........................................................................ 103
Ogonek diacritic ......................................................... 103
Omitting indentation ................................................... 72
One-argument form of cross-references ..................... 44
OPENQUOTE_SYMBOL ................................................ 185
'\openout' line in log file .......................................... 156
Optional and repeated arguments ............................ 118
Options for makeinfo ............................................... 164
Options for texi2any ............................................... 164
Options of texi2html ............................................... 192
Options, customization variables for ....................... 173
Ordinals, Romance .................................................... 103
Ordinary TeX commands, using ............................... 135
Orphans, preventing .................................................. 116
Other Info files' nodes ................................................. 36
OUTFILE ................................................................. 168
Outline of file structure, showing ............................ 250
Output document strings, ........................................... 250
internationalization of .............................................. 188
output file name ....................................................... 13
Output file splitting .................................................. 170
Output formats .......................................................... 3
Output formats, supporting more ............................... 5
Output, in PDF .......................................................... 154
Output, printed through texi2any .................. 171
--output=file ......................................................... 168
OUTPUT_ENCODING_NAME ........................................... 185
OUTPUT_FILE_NAME_ENCODING ................................ 185
--outputindent ....................................................... 169
Outputting EPUB ....................................................... 206
Outputting HTML ...................................................... 203
Overfull 'hboxes' ...................................................... 162
Overview of Texinfo .................................................. 2
Owner of copyright for FSF works ......................... 17

P

-p indent ............................................................... 169
-P path ................................................................. 169
PACKAGE ............................................................... 186
PACKAGE_AND_VERSION .......................................... 186
PACKAGE_NAME ....................................................... 186
PACKAGE_URL ......................................................... 186
PACKAGE_VERSION .................................................... 186
Page breaks, awkward .............................................. 113
Page breaks, forcing .................................................. 115
Page delimiter in Texinfo mode .............................. 250
Page headings .......................................................... 270
Page numbering ......................................................... 270
Page sizes for books ............................................... 275
Page sizes, customized ............................................... 276
page-delimiter ......................................................... 250
Pages, starting odd .................................................. 269
Paper size, A4 ........................................................... 275
Paragraph indentation control .................................. 274
Paragraph, marking text within ................................ 54
paragraphindent ......................................................... 173
--paragraph-indent=indent ...................................... 169
Parameters to macros ................................................ 143
Parentheses in node names ...................................... 29
Parsing errors .......................................................... 257
Part of file formatting and printing ......................... 259
Part pages .............................................................. 41
Partial documents, formatting ................................... 157
Parts of a cross-reference ........................................ 43
Parts of a master menu ............................................. 24
Parts of a menu ...................................................... 35
Patches, contributing ................................................ 3
PCL file, for printing ............................................... 158
--pdf ................................................................. 169
df image inclusions .................................................. 85
--pdf, for texi2dvi ................................................. 154
PDF output .......................................................... 154
PDF output of URLs ............................................... 52
PDF output, overview ................................................. 4
pdf2tex ................................................................. 157
pdf2tex, and images .................................................. 85
pdf2texi2dvi ......................................................... 154
Period in node name ............................................... 29
Periods, inserting .................................................... 100
Perl extension modules (XS) ...................................... 171
Perl periods, inserting .............................................. 100
Perl Pod, converting to Texinfo ............................... 189
Permissions text, including .................................. 18
Permissions, printed ........................................ 21
**pg** (program) index .................................. 90
Picas ................................................................ 87
Pictures, inserting ........................................... 85
Plain text output with **--plaintext** .................. 169
Plain text output, overview ............................. 3
Plain **T e X** output ..................................... 135
**--plaintext** ............................................. 169
**png image format** ...................................... 85
Pod, converting to **Texinfo** ......................... 189
pod2t e xi .................................................. 189
Point, indicating in a buffer ............................. 110
Pointer creation with **makeinfo** ................. 27
Pointer validation .......................................... 168
  suppressing ......................................... 157
Points (dimension) ......................................... 86
PostScript output, overview ............................ 3
Pounds symbol ............................................. 107
PRE_BODY_CLOSE ........................................... 179
Preamble ...................................................... 14
Preamble, in **Info** format ............................. 279
Predefined names for indices ......................... 94
Preface, etc., and **DocBook** ....................... 39
<preface> **DocBook** tag ................................ 39
PREINDEX .................................................... 186
Preparing for **T e X** .................................. 161
Prev node of Top node .................................. 30
Preventing first paragraph indentation .............. 274
Preventing line and page breaks ...................... 113
Print and format in **Texinfo** mode ............... 159
Print, format from **Emacs** shell .................... 159
Printed book and manual characteristics ............ 5
Printed output, indicating ............................ 109
Printed output, through **texi2any** ............... 171
Printed permissions ...................................... 21
Printing a region or buffer ............................ 259
Printing an index ......................................... 93
Printing cost, reducing .................................. 63
Printing **DVI** files, on
  MS-DOS/MS-Windows .................................. 158
Printing hardcopy ....................................... 154
Problems, catching ...................................... 263
.profile initialization file ............................. 161
Program names, indicating ............................ 60
**PROGRAM** ............................................... 186
**PROGRAM_NAME_IN_ABOUT** ......................... 179
**PROGRAM_NAME_IN_FOOTER** ......................... 179
Programming, glyphs for ................................ 108
Pronunciation of **TeXinfo** ............................ 2
Prototype row, column widths defined by .......... 81
  **--ps** ............................................. 169
  **--ps**, for **texi2dvi** ............................. 155
Q
  **--quiet**, for **install-info** ..................... 199
Quotation characters ("), in source ................ 104
Quotation marks, French ............................... 104
Quotation marks, German .............................. 104
Quotation marks, inserting ........................... 104
Quotations ............................................... 66
Quotations in smaller fonts .......................... 73
Quote characters, inserting .......................... 99
Quoting, automatic for some macros ............... 144
R
  Ragged left, without filling ....................... 70
  Ragged right, with filling ......................... 71
  Ragged right, without filling ..................... 70
  Raising and lowering sections .................... 42
  Raw formatter commands ............................ 135
  Raw **HTML** ......................................... 203
  raw text output ..................................... 174
  Recommended names for keys ....................... 57
  Rectangle, black in hardcopy ...................... 163
  Recursive macro invocations ...................... 144
  Reducing font size .................................. 63
  Reference implementation .......................... 8
  Reference to **@commands** ......................... 215
  References .......................................... 43
  References using **@inforef** ...................... 50
  References using **@ref** ............................ 49
  References using **@xref** ......................... 48
  References using **@pxref** ......................... 48
  Referring to an entire manual ..................... 47
  Referring to other **Info** files ................... 36
  **--regex=regex**, for **install-info** .......... 199
  Region formatting and printing .................... 259
  Region printing in **Texinfo** mode .............. 159
  Registered symbol .................................. 107
  Regular expression, for **‘#line’** ............... 151
  Reid, Brian ........................................... 6
  **--remove**, for **install-info** .................. 199
  **--remove-exactly**, for **install-info** .......... 199
  Repeated and optional arguments .................. 118
  Reporting bugs ...................................... 2
  Required in **texinfo** file ....................... 10
  Requirements for formatting ....................... 161
  Requirements for include files .................... 256
  Requirements for updating commands ............... 254
  Reserved words, indicating ....................... 55
  Restrictions on node names ....................... 29
  Result of an expression ............................ 108
  Return type, own line for .......................... 122
  RGB color specification ............................ 52
  **ri d.t.e ps** ....................................... 87
  Right quotation marks .............................. 104
  Right-pointing angle quotation marks .............. 104
  Ring accent ......................................... 103
  Robbins, Arnold .................................... 158
  Roman font ......................................... 64
<table>
<thead>
<tr>
<th>W</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>@w, for blank items</td>
<td>76</td>
<td>Writing index entries</td>
</tr>
<tr>
<td>&lt;warning&gt; DocBook tag</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>White space in node name</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Whitespace in macros</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>Whitespace, collapsed around continuations</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>Whitespace, controlling in conditionals</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>Whitespace, inserting</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Whole manual, in Info format</td>
<td>278</td>
<td></td>
</tr>
<tr>
<td>Width of images</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>Width of text area</td>
<td>276</td>
<td></td>
</tr>
<tr>
<td>Widths, defining multitable column</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>Wildcards</td>
<td>157</td>
<td></td>
</tr>
<tr>
<td>word counting</td>
<td>174</td>
<td></td>
</tr>
<tr>
<td>Words and phrases, marking them</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>WORDS_IN_PAGE</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>Writing a @node line</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>Writing a $node line</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>xdvi</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>--xml</td>
<td>170</td>
<td>XML DocBook output, overview</td>
</tr>
<tr>
<td>XML Texinfo output, overview</td>
<td>4</td>
<td>XML, including raw</td>
</tr>
<tr>
<td>--Xopt str</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td>XPM image format</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>XREF_USE_FLOAT_LABEL</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>XREF_USE_NODE_NAME_ARG</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>XZ-compressed dir files, reading</td>
<td>197</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Years, in copyright line</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>