

# Halibut User Manual

---

Halibut is a free (MIT-licensed) documentation production system, able to generate multiple output formats from the same input data. This document is its user manual.

This manual is copyright 2004-2007 Simon Tatham. All rights reserved. You may distribute this documentation under the MIT licence. See appendix A for the licence text in full.

# Contents

---

Chapter 1: Introduction to Halibut . . . . .	6
1.1 Output formats supported by Halibut . . . . .	6
1.2 Features supported by Halibut . . . . .	6
Chapter 2: Running Halibut . . . . .	7
2.1 Command-line options . . . . .	7
Chapter 3: Halibut input format . . . . .	11
3.1 The basics . . . . .	11
3.2 Simple inline formatting commands . . . . .	12
3.2.1 \e: Emphasising text . . . . .	12
3.2.2 \c and \cw: Displaying computer code inline . . . . .	12
3.2.3 \q: Quotation marks . . . . .	13
3.2.4 \- and \_: Non-breaking hyphens and spaces . . . . .	14
3.2.5 \date: Automatic date generation . . . . .	14
3.2.6 \w: WWW hyperlinks . . . . .	15
3.2.7 \u: Specifying arbitrary Unicode characters . . . . .	16
3.2.8 \k and \K: Cross-references to other sections . . . . .	16
3.2.9 \#: Inline comments . . . . .	17
3.3 Paragraph-level commands . . . . .	17
3.3.1 \c: Displaying whole paragraphs of computer code . . . . .	17
3.3.2 \b, \n, \dt, \dd, \lcont: Lists . . . . .	18
3.3.2.1 \b: Bulleted lists . . . . .	19
3.3.2.2 \n: Numbered lists . . . . .	19
3.3.2.3 \dt and \dd: Description lists . . . . .	20
3.3.2.4 Continuing list items into further paragraphs . . . . .	21
3.3.3 \rule: Horizontal rules . . . . .	22

3.3.4 \quote: Indenting multiple paragraphs as a long quotation . . . . .	22
3.3.5 \C, \H, \S, \A, \U: Chapter and section headings . . . . .	23
3.3.6 \copyright, \title, \versionid: Miscellaneous blurb commands . . . . .	24
3.3.7 \#: Whole-paragraph comments . . . . .	25
3.4 Creating a bibliography . . . . .	25
3.5 Creating an index . . . . .	26
3.5.1 Simple indexing . . . . .	26
3.5.2 Special cases of indexing . . . . .	27
3.5.3 Fine-tuning the index . . . . .	27
3.5.4 Indexing terms that differ only in case . . . . .	29
3.6 Configuring Halibut . . . . .	29
3.7 Defining macros . . . . .	31
Chapter 4: Halibut output formats . . . . .	33
4.1 Plain text . . . . .	33
4.1.1 Output file name . . . . .	33
4.1.2 Indentation and line width . . . . .	33
4.1.3 Configuring heading display . . . . .	34
4.1.4 Configuring the characters used . . . . .	35
4.1.5 Miscellaneous configuration options . . . . .	36
4.1.6 Default settings . . . . .	37
4.2 HTML . . . . .	37
4.2.1 Controlling the output file names . . . . .	38
4.2.2 Controlling the splitting into HTML files . . . . .	39
4.2.3 Including pieces of your own HTML . . . . .	40
4.2.4 Configuring heading display . . . . .	41
4.2.5 Configuring standard text . . . . .	41
4.2.6 Configuring the characters used . . . . .	42
4.2.7 Miscellaneous options . . . . .	43
4.2.8 Generating MS Windows HTML Help . . . . .	44

4.2.9 Default settings	46
4.3 Windows Help	47
4.3.1 Output file name	47
4.3.2 Configuring the characters used	47
4.3.3 Miscellaneous configuration options	48
4.3.4 Default settings	48
4.4 Unix man pages	49
4.4.1 Output file name	49
4.4.2 Configuring headers and footers	49
4.4.3 Configuring heading display	49
4.4.4 Configuring the characters used	50
4.4.5 Default settings	51
4.5 GNU info	51
4.5.1 Controlling the output filenames	51
4.5.2 Indentation and line width	52
4.5.3 Configuring heading display	52
4.5.4 Controlling the characters used	52
4.5.5 Miscellaneous configuration options	53
4.5.6 Default settings	54
4.6 Paper formats	54
4.6.1 PDF	54
4.6.2 PostScript	55
4.6.3 Configuring layout and measurements	55
4.6.3.1 Page properties	55
4.6.3.2 Vertical spacing	55
4.6.3.3 Indentation	56
4.6.3.4 Headings	56
4.6.3.5 Contents and index	56
4.6.3.6 Fonts	57
4.6.3.7 Miscellaneous	58

4.6.4 Configuring the characters used . . . . .	58
4.6.5 Default settings for paper formats . . . . .	59
Appendix A: Halibut Licence . . . . .	61
Appendix B: Halibut man page . . . . .	62
B.1 NAME . . . . .	62
B.2 SYNOPSIS . . . . .	62
B.3 DESCRIPTION . . . . .	62
B.4 OPTIONS . . . . .	62
B.5 MORE INFORMATION . . . . .	64
B.6 BUGS . . . . .	64
Index . . . . .	65

# Chapter 1: Introduction to Halibut

---

Halibut is a multi-format documentation processing system.

What that means is that you write your document once, in Halibut's input format, and then the Halibut program processes it into several output formats which all contain the same text. So, for example, if you want your application to have a Windows help file, and you also want the same documentation available in HTML on your web site, Halibut can do that for you.

## 1.1 Output formats supported by Halibut

Currently Halibut supports the following output formats:

- Plain ASCII text.
- HTML.
- Unix `man` page format.
- GNU `info` format.
- PDF.
- PostScript.
- Old-style Windows Help ( `.HLP` ).

(By setting suitable options, the HTML output can also be made suitable for feeding to the newer-style Windows HTML Help compiler.)

## 1.2 Features supported by Halibut

Here's a list of Halibut's notable features.

- Halibut automatically assigns sequential numbers to your chapters, sections and subsections, and keeps track of them for you. You supply a *keyword* for each section, and then you can generate cross-references to that section using the keyword, and Halibut will substitute the correct section number. Also, in any output format where it makes sense, the cross-references will be hyperlinks to that section of the document.
- Halibut has some support for Unicode: you can include arbitrary Unicode characters in your document, and specify fallback text in case any output format doesn't support that character.
- Halibut's indexing support is comprehensive and carefully designed. It's easy to use in the simple case, but has powerful features that should make it possible to maintain a high-quality and useful index.

## Chapter 2: Running Halibut

---

In the simplest case, running Halibut is very easy. You provide a set of input files on its command line, and it produces a set of output files.

```
$ halibut intro.but gettingstarted.but reference.but index.but
```

This will generate a large set of output files:

- `output.txt` will be a plain text version of the input document.
- `output.hlp` and `output.cnt` will be an old-style Windows Help version of the same thing. (Most of the text is in `output.hlp`; `output.cnt` contains additional contents data used by the Windows help topic selector. If you lose the latter, the former should still be usable, but it will look less modern.)

Note that to do this Halibut does not require any external software such as a Help compiler. It *directly* generates old-style Windows Help files, and therefore it doesn't need to be run on Windows to do so: it can generate them even when run from an automated script on a Unix machine.

- `output.1` will be a Unix man page.
- The set of files `*.html` will contain an HTML version of the document. If you have configured Halibut to generate more than one HTML file (the default), then the file `Contents.html` will be the topmost one that users should be directed to initially. If you have configured Halibut to generate a single file, it will be called `Manual.html`.
- `output.info`, and some additional files `output.info-1`, `output.info-2` etc., will be files suitable for use with GNU `info`.
- `output.pdf` will be a printable PDF manual.
- `output.ps` will be a printable PostScript manual.

### 2.1 Command-line options

Halibut supports command-line options in case you don't want to use all of Halibut's output formats, or you want to configure the names of your output files, or you want to supply additional configuration on the command line. The supported options are listed here.

Firstly, there are options which indicate which of the output formats you want Halibut to generate:

```
--text[=filename]
```

Specifies that you want to generate plain text output. You can optionally specify a file name

(e.g. `--text=myfile.txt`), in which case Halibut will change the name of the output file as well.

`--html[=filename]`

Specifies that you want to generate HTML output. You can optionally specify a file name (e.g. `--html=myfile.html`), in which case Halibut will change the name of the output file as well. Specifying a file name here will also cause the HTML to be output in *only* one file, instead of the usual behaviour of producing multiple files with links between them. If you want to produce multiple files and configure their names, you will need to use the more complete file name configuration directives given in section 4.2.1 (although you may want to do so on the command line, using the `-C` option).

`--xhtml[=filename]`

Synonym for `--html`.

`--winhelp[=filename]`

Specifies that you want to generate old-style Windows Help output. You can optionally specify a file name (e.g. `--winhelp=myfile.hlp`), in which case Halibut will change the name of the output file as well.

Your output file name should end with `.hlp`; if it doesn't, Halibut will append it. Halibut will also generate a contents file (ending in `.cnt`) alongside the file name you specify.

`--whlp[=filename]`

Synonym for `--winhelp`.

`--hlp[=filename]`

Synonym for `--winhelp`.

`--man[=filename]`

Specifies that you want to generate man page output. You can optionally specify a file name (e.g. `--man=myfile.5`), in which case Halibut will change the name of the output file as well.

`--info[=filename]`

Specifies that you want to generate GNU `info` output. You can optionally specify a file name (e.g. `--info=myfile.info`), in which case Halibut will change the name of the output file as well.

Unless the `info` output format is configured not to (see section 4.5), Halibut will divide the `info` output into many small files. The extra files will have numeric suffixes on their names; so, for example, `output.info` might be accompanied by additional files `output.info-1`, `output.info-2` and so on.

`--pdf[=filename]`

Specifies that you want to generate PDF output. You can optionally specify a file name (e.g. `--pdf=myfile.pdf`), in which case Halibut will change the name of the output file as well.



`--ps[=filename]`

Specifies that you want to generate PostScript output. You can optionally specify a file name (e.g. `--ps=myfile.ps`), in which case Halibut will change the name of the output file as well.

If you do not specify any of the above options, Halibut will simply produce *all* of its output formats.

Also, there is an option which allows you to specify an arbitrary `\cfg` configuration directive (see section 3.6):

`-Cconfig-directive:value[:value...]`

The text following `-C` is expected to be a colon-separated list of strings. (If you need a literal colon, you can escape it with a backslash: `\:`. If you need a literal *backslash*, you can do the same: `\\`.) These strings are used as the parts of a `\cfg` directive. So, for example, the option

`-Ctext-section-align:2:leftplus`

will translate into the configuration directive

`\cfg{text-section-align}{2}{leftplus}`

(Note that your shell may also take an interest in backslashes, particularly under Unix. You may find that the backslash with which you escape a colon must be doubled in order to make the shell pass it to Halibut at all, and to pass a doubled backslash to Halibut you might have to type four backslashes on your shell command line. This is not part of Halibut's own behaviour, and it cannot do anything about it.)

Configuration directives created in this way take effect after all other input has been processed. (In most cases, this has the effect of overriding any other instances of the directive in the input.)

The options which set the output file names actually work by implicitly generating these configuration directives. When you specify `--text=myfile.txt`, for example, Halibut treats it identically to `--text -Ctext-filename:myfile.txt`. The Windows Help and man page formats work similarly. HTML is slightly different, since it also arranges for single-file output if you pass a filename to `--html`; so `--html=myfile.html` is equivalent to `--html -Chtml-single-filename:myfile.html -Chtml-leaf-level:0`. (See chapter 4 for explanations of all these configuration directives.)

In addition to these, there are also a few other options:

`--input-charset=charset`

Changes the default assumed character set for all input files from ASCII to something else. (`-Cinput-charset` cannot be used for this, as `-C` directives are processed after all other input, so wouldn't affect any files.)

Any `\cfg{input-charset}` directives within input files override this option.

See section 3.6 for more information about the input character set.

`--list-charsets`

List character sets known to Halibut.

`--help`

Print a brief help message and exit immediately. (Don't confuse this with `--winhelp`!)

`--version`

Print information about Halibut's version number and exit immediately.

`--licence`

Display Halibut's licence (see also appendix A) and exit immediately.

`--license`

US English alternative spelling of `--licence`.

`--precise`

Report column numbers as well as line numbers when reporting errors in the Halibut input files.

## Chapter 3: Halibut input format

---

This chapter describes the format in which you should write documents to be processed by Halibut.

### 3.1 The basics

Halibut's input files mostly look like ordinary ASCII text files; you can edit them with any text editor you like.

Writing paragraphs of ordinary text is very simple: you just write ordinary text in the ordinary way. You can wrap a paragraph across more than one line using line breaks in the text file, and Halibut will ignore this when it rewraps the paragraph for each output format. To separate paragraphs, use a blank line (i.e. two consecutive line breaks). For example, a fragment of Halibut input looking like this:

```
This is a line of text.  
This is another line of text.
```

This line is separated from the previous one by a blank line.

will produce two paragraphs looking like this:

```
This is a line of text. This is another line of text.
```

```
This line is separated from the previous one by a blank line.
```

The first two lines of the input have been merged together into a single paragraph, and the line break in the input file was treated identically to the spaces between the individual words.

Halibut is designed to have very few special characters. The only printable characters in Halibut input which will not be treated exactly literally in the output are the backslash (\) and the braces ({ and }). If you do not use these characters, *everything* else you might type in normal ASCII text is perfectly safe. If you do need to use any of those three characters in your document, you will have to precede each one with a backslash. Hence, for example, you could write

```
This \\ is a backslash, and these are \{braces\}.
```

and Halibut would generate the text

```
This \ is a backslash, and these are {braces}.
```

If you want to write your input file in a character set other than ASCII, you can do so by using the `\cfg{input-charset}` command. See section 3.6 for details of this.

## 3.2 Simple inline formatting commands

Halibut formatting commands all begin with a backslash, followed by a word or character identifying the command. Some of them then use braces to surround one or more pieces of text acted on by the command. (In fact, the `\\`, `\{` and `\}` sequences you met in section 3.1 are themselves formatting commands.)

This section describes some simple formatting commands you can use in Halibut documents. The commands in this section are *inline* commands, which means you can use them in the middle of a paragraph. Section 3.3 describes some *paragraph* commands, which affect a whole paragraph at a time.

Many of these commands are followed by a pair of braces surrounding some text. In all cases, it is perfectly safe to have a line break (in the input file) within those braces; Halibut will treat that exactly the same as a space. For example, these two paragraphs will be treated identically:

```
Here is some \e{emphasised  
text}.
```

```
Here is some \e{emphasised text}.
```

### 3.2.1 `\e`: Emphasising text

Possibly the most obvious piece of formatting you might want to use in a document is *emphasis*. To emphasise text, you use the `\e` command, and follow it up with the text to be emphasised in braces. For example, the first sentence in this paragraph was generated using the Halibut input

```
Possibly the most obvious piece of formatting you might want  
to use in a document is \e{emphasis}.
```

### 3.2.2 `\c` and `\cw`: Displaying computer code inline

Halibut was primarily designed to produce software manuals. It can be used for other types of document as well, but software manuals are its speciality.

In software manuals, you often want to format text in a way that indicates that it is something you might see displayed verbatim on a computer screen. In printed manuals, this is typically done by setting that text in a font which is obviously fixed-width. This provides a visual cue that the text being displayed is code, and it also ensures that punctuation marks are clearly separated and shown individually (so that a user can copy the text accurately and conveniently).

Halibut provides *two* commands for this, which are subtly different. The names of those commands are `\c` ('code') and `\cw` ('weak code'). You use them just like `\e`, by following them with some text in braces. For example, this...

```
This sentence contains some \c{code} and some \cw{weak code}.
```

... produces this:

```
This sentence contains some code and some weak code.
```

The distinction between code and weak code is mainly important when producing plain text output. Plain text output is typically viewed in a fixed-width font, so there is no need (and no

way) to change font in order to make the order of punctuation marks clear. However, marking text as code is also *sometimes* done to provide a visual distinction between it and the text around it, so that the reader knows where the literal computer text starts and stops; and in plain text, this cannot be done by changing font, so there needs to be an alternative way.

So in the plain text output format, things marked as code (`\c`) will be surrounded by quote marks, so that it's obvious where they start and finish. Things marked as weak code (`\cw`) will not look any different from normal text.

I recommend using weak code for any application where it is *obvious* that the text is literal computer input or output. For example, if the text is capitalised, that's usually good enough. If I talk about the Pentium's EAX and EDX registers, for example, you don't need quotes to notice that those are special; so I would write that in Halibut as 'the Pentium's `\cw{EAX}` and `\cw{EDX}` registers'. But if I'm talking about the Unix command `man`, which is an ordinary English word in its own right, a reader might be slightly confused if it appeared in the middle of a sentence undecorated; so I would write that as 'the Unix command `\c{man}`'.

In summary:

- `\c` means 'this text *must* be visually distinct from the text around it'. Halibut's various output formats will do this by changing the font if possible, or by using quotes if not.
- `\cw` means 'it would be nice to display this text in a fixed-width font if possible, but it's not essential'.

In really extreme cases, you might want Halibut to use quotation marks even in output formats which can change font. In section 3.2.5, for example, I mention the special formatting command `\.`. If that appeared at the end of a sentence *without* the quotes, then the two adjacent full stops would look pretty strange even if they were obviously in different fonts.

For this, Halibut supports the `\cq` command, which is exactly equivalent to using `\q` to provide quotes and then using `\cw` inside the quotes. So in the paragraph above, for example, I wrote

the special formatting command `\cq{\.\.}`.

and I could equivalently have written

the special formatting command `\q{\cw{\.\.}}`.

There is a separate mechanism for displaying computer code in an entire paragraph; see section 3.3.1 for that one.

### 3.2.3 `\q`: Quotation marks

Halibut's various output formats don't all use the same conventions for displaying text in ordinary quotation marks ('like these'). Some output formats have access to proper matched quote characters, whereas others are restricted to using plain ASCII. Therefore, it is not ideal to use the ordinary ASCII double quote character in your document (although you can if you like).

Halibut provides the formatting command `\q` to indicate quoted text. If you write

Here is some `\q{text in quotes}`.

then Halibut will print

Here is some ‘text in quotes’.

and in every output format Halibut generates, it will choose the best quote characters available to it in that format. (The quote characters to use can be configured with the `\cfg` command.)

You can still use the ordinary quote characters of your choice if you prefer; or you could even use the `\u` command (see section 3.2.7) to generate Unicode matched quotes (single or double) in a way which will automatically fall back to the normal ASCII one if they aren't available. But I recommend using the built-in `\q` command in most cases, because it's simple and does the best it can everywhere.

If you're using the `\c` or `\cw` commands to display literal computer code, you will probably want to use literal ASCII quote characters, because it is likely to matter precisely which quote character you use. In fact, Halibut actually *disallows* the use of `\q` within either of `\c` and `\cw`, since this simplifies some of the output formats.

### 3.2.4 `\-` and `\_`: Non-breaking hyphens and spaces

If you use an ordinary hyphen in the middle of a word (such as ‘built-in’), Halibut's output formats will feel free to break a line after that hyphen when wrapping paragraphs. This is fine for a word like ‘built-in’, but if you were displaying some literal computer code such as the Emacs command `M-x psychoanalyze-pinhead`, you might prefer to see the whole hyphenated word treated as an unbreakable block. In some cases, you might even want to prevent the *space* in that command from becoming a line break.

For these purposes, Halibut provides the commands `\-` and `\_`, which generate a non-breaking hyphen and a non-breaking space respectively. So the above Emacs command might be written as

the Emacs command `\c{M\_x\_psychoanalyze\_pinhead}`

Unfortunately, some of Halibut's output formats do not support non-breaking hyphens, and others don't support *breaking* hyphens! So Halibut cannot promise to honour these commands in all situations. All it can do is make a best effort.

### 3.2.5 `\date`: Automatic date generation

Sometimes you might want your document to give an up-to-date indication of the date on which it was run through Halibut.

Halibut supplies the `\date` command to do this. In its simplest form, you simply say

This document was generated on `\date`.

and Halibut generates something like

This document was generated on Thu 31 Jan 2013 02:35:28 EST.

You can follow the `\date` command directly with punctuation (as in this example, where it is immediately followed by a full stop), but if you try to follow it with an alphabetic or numeric character (such as writing `\dateZ`) then Halibut will assume you are trying to invoke the name

of a macro command you have defined yourself, and will complain if no such command exists. To get round this you can use the special ‘\.’ do-nothing command. See section 3.7 for more about general Halibut command syntax and ‘\.’.

If you would prefer the date to be generated in a specific format, you can follow the `\date` command with a format specification in braces. The format specification will be run through the standard C function `strftime`, so any format acceptable to that function is acceptable here as well. I won't document the format here, because the details vary from computer to computer (although there is a standard core which should be supported everywhere). You should look at your local system's manual for `strftime` for details.

Here's an example which generates the date in the international standard ISO 8601 format:

This document was generated on `\date{%Y-%m-%d %H:%M:%S}`.

And here's some sample output from that command:

This document was generated on 2013-01-31 02:35:28.

### 3.2.6 \w: WWW hyperlinks

Since one of Halibut's output formats is HTML, it's obviously useful to be able to provide links to arbitrary web sites in a Halibut document.

This is done using the `\W` command. `\W` expects to be followed by *two* sets of braces. In the first set of braces you put a URL; in the second set you put the text which should be a hyperlink. For example, you might write

Try searching on `\W{http://www.google.com/}{Google}`.

and Halibut would generate

Try searching on Google.

Note that hyperlinks, like the non-breaking commands discussed in section 3.2.4, are *discretionary*: if an output format does not support them then they will just be left out completely. So unless you're *only* intending to use the HTML output format, you should avoid storing vital content in the URL part of a `\W` command. The Google example above is reasonable (because most users are likely to be able to find Google for themselves even without a convenient hyperlink leading straight there), but if you really need to direct users to a specific web site, you will need to give the URL in actual displayed text (probably displayed as code as well). However, there's nothing to stop you making it a hyperlink *as well* for the convenience of HTML readers.

The `\W` command supports a piece of extra syntax to make this convenient for you. You can specify `\c` or `\cw` *between* the first and second pairs of braces. For example, you might write

Google is at `\W{http://www.google.com/}\cw{www.google.com}`.

and Halibut would produce

Google is at `www.google.com`.

If you want the link text to be an index term as well, you can also specify `\i` or `\ii`; this has to come before `\c` or `\cw` if both are present. (See section 3.5 for more about indexing.)

### 3.2.7 \u: Specifying arbitrary Unicode characters

Halibut has extensive support for Unicode and character set conversion. You can specify any (reasonably well known) character set for your input document, and Halibut will convert it all to Unicode as it reads it in. See section 3.6 for more details of this.

If you need to specify a Unicode character in your input document which is not supported by the input character set you have chosen, you can use the `\u` command to do this. `\u` expects to be followed by a sequence of hex digits; so that `\u0041`, for example, denotes the Unicode character `0x0041`, which is the capital letter A.

If a Unicode character specified in this way is not supported in a particular *output* format, you probably don't just want it to be omitted. So you can put a pair of braces after the `\u` command containing fallback text. For example, to specify an amount of money in euros, you might write this:

```
This is likely to cost \u20AC{EUR\_}2500 at least.
```

Halibut will render that as a Euro sign *if available*, and the text 'EUR ' if not. In the output format you're currently reading in, the above input generates this:

This is likely to cost EUR 2500 at least.

If you read it in other formats, you may see different results.

### 3.2.8 \k and \K: Cross-references to other sections

Section 1.2 mentions that Halibut numbers the sections of your document automatically, and can generate cross-references to them on request. `\k` and `\K` are the commands used to generate those cross-references.

To use one of these commands, you simply follow it with a pair of braces containing the keyword for the section in question. For example, you might write something like

```
\K{input-xref} expands on \k{intro-features}.
```

and Halibut would generate something like

Section 3.2.8 expands on section 1.2.

The keywords `input-xref` and `intro-features` are section keywords used in this manual itself. In your own document, you would have supplied a keyword for each one of your own sections, and you would provide your own keywords for the `\k` command to work on.

The difference between `\k` and `\K` is simply that `\K` starts the cross-reference text with a capital letter; so you would use `\K` at the beginning of a sentence, and `\k` everywhere else.

In output formats which permit it, cross-references act as hyperlinks, so that clicking the mouse on a cross-reference takes you straight to the referenced section.

The `\k` commands are also used for referring to entries in a bibliography (see section 3.4 for more about bibliographies), and can also be used for referring to an element of a numbered list by its number (see section 3.3.2.2 for more about numbered lists).

See section 3.3.5 for more about chapters and sections.



### 3.2.9 \#: Inline comments

If you want to include comments in your Halibut input, to be seen when reading it directly but not copied into the output text, then you can use `\#` to do this. If you follow `\#` with text in braces, that text will be ignored by Halibut.

For example, you might write

```
The typical behaviour of an antelope \#{do I mean
gazelle?} is...
```

and Halibut will simply leave out the aside about gazelles, and will generate nothing but

The typical behaviour of an antelope is...

This command will respect nested braces, so you can use it to comment out sections of Halibut markup:

```
This function is \#{very, \e{very}} important.
```

In this example, the comment lasts until the final closing brace (so that the whole ‘very, *very*’ section is commented out).

The `\#` command can also be used to produce a whole-paragraph comment; see section 3.3.7 for details of that.

## 3.3 Paragraph-level commands

This section describes Halibut commands which affect an entire paragraph, or sometimes even *more* than one paragraph, at a time.

### 3.3.1 \c: Displaying whole paragraphs of computer code

Section 3.2.2 describes a mechanism for displaying computer code in the middle of a paragraph, a few words at a time.

However, this is often not enough. Often, in a computer manual, you really want to show several lines of code in a display paragraph.

This is also done using the `\c` command, in a slightly different way. Instead of using it in the middle of a paragraph followed by braces, you can use it at the start of each line of a paragraph. For example, you could write

```
\c #include <stdio.h>
\c
\c int main(int argc, char **argv) {
\c     printf("hello, world\n");
\c     return 0;
\c }
```

and Halibut would generate

```
#include <stdio.h>

int main(int argc, char **argv) {
```

```

    printf("hello, world\n");
    return 0;
}

```

Note that the above paragraph makes use of a backslash and a pair of braces, and does *not* need to escape them in the way described in section 3.1. This is because code paragraphs formatted in this way are a special case; the intention is that you can just copy and paste a lump of code out of your program, put ‘\c ’ at the start of every line, and simply *not have to worry* about the details - you don't have to go through the whole block looking for characters to escape.

Since a backslash inside a code paragraph generates a literal backslash, this means you cannot use any other Halibut formatting commands inside a code paragraph. In particular, if you want to emphasise a particular word in the paragraph, you can't do that using \e (section 3.2.1) in the normal way.

Therefore, Halibut provides an alternative means of emphasis in code paragraphs. Each line beginning with \c can optionally be followed by a single line beginning with \e, indicating the emphasis in that line. The emphasis line contains the letters b and i (for ‘bold’ and ‘italic’, although some output formats might render i as underlining instead of italics), positioned to line up under the parts of the text that you want emphasised.

For example, if you wanted to do syntax highlighting on the above C code by highlighting the preprocessor command in italic and the keywords in bold, you might do it like this:

```

\c #include <stdio.h>
\e iiiiiiiiiiiiiiiiiiiiii
\c
\c int main(int argc, char **argv) {
\e bbb          bbb          bbbb
\c     printf("hello, world\n");
\c     return 0;
\e     bbbbbb
\c }

```

and Halibut would generate:

```

#include <stdio.h>

int main(int argc, char **argv) {
    printf("hello, world\n");
    return 0;
}

```

Note that not every \c line has to be followed by a \e line; they're optional.

Also, note that highlighting within a code paragraph is *discretionary*. Not all of Halibut's output formats can support it (plain text, in particular, has no sensible way to do it). Unless you know you are using a restricted range of output formats, you should use highlighting in code paragraphs *only* as a visual aid, and not rely on it to convey any vital semantic content.

### 3.3.2 \b, \n, \dt, \dd, \lcont: Lists

Halibut supports bulleted lists, numbered lists and description lists.

### 3.3.2.1 \b: Bulleted lists

To create a bulleted list, you simply prefix each paragraph describing a bullet point with the command `\b`. For example, this Halibut input:

```
Here's a list:
```

```
\b One.
```

```
\b Two.
```

```
\b Three.
```

would produce this Halibut output:

```
Here's a list:
```

- One.
- Two.
- Three.

### 3.3.2.2 \n: Numbered lists

Numbered lists are just as simple: instead of `\b`, you use `\n`, and Halibut takes care of getting the numbering right for you. For example:

```
Here's a list:
```

```
\n One.
```

```
\n Two.
```

```
\n Three.
```

This produces the Halibut output:

```
Here's a list:
```

1. One.
2. Two.
3. Three.

The disadvantage of having Halibut sort out the list numbering for you is that if you need to refer to a list item by its number, you can't reliably know the number in advance (because if you later add another item at the start of the list, the numbers will all change). To get round this, Halibut allows an optional keyword in braces after the `\n` command. This keyword can then be referenced using the `\k` or `\K` command (see section 3.2.8) to provide the number of the list item. For example:

```
Here's a list:
```

```
\n One.  
  
\n{this-one} Two.  
  
\n Three.  
  
\n Now go back to step \k{this-one}.
```

This produces the following output:

Here's a list:

1. One.
2. Two.
3. Three.
4. Now go back to step 2.

The keyword you supply after `\n` is allowed to contain escaped special characters (`\\`, `\{` and `\}`), but should not contain any other Halibut markup. It is intended to be a word or two of ordinary text. (This also applies to keywords used in other commands, such as `\B` and `\C`).

### 3.3.2.3 `\dt` and `\dd`: Description lists

To write a description list, you prefix alternate paragraphs with the `\dt` (‘described thing’) and `\dd` (description) commands. For example:

```
\dt Pelican  
  
\dd This is a large bird with a big beak.  
  
\dt Panda  
  
\dd This isn't.
```

This produces the following output:

Pelican

This is a large bird with a big beak.

Panda

This isn't.

If you really want to, you are allowed to use `\dt` and `\dd` without strictly interleaving them (multiple consecutive `\dts` or consecutive `\dds`, or a description list starting with `\dd` or ending with `\dt`). This is probably most useful if you are listing a sequence of things with `\dt`, but only some of them actually need `\dd` descriptions. You should *not* use multiple consecutive `\dds` to provide a multi-paragraph definition of something; that's what `\lcont` is for, as explained in section 3.3.2.4.

### 3.3.2.4 Continuing list items into further paragraphs

All three of the above list types assume that each list item is a single paragraph. For a short, snappy list in which each item is likely to be only one or two words, this is perfectly sufficient; but occasionally you will find you want to include several paragraphs in a single list item, or even to nest other types of paragraph (such as code paragraphs, or other lists) inside a list item.

To do this, you use the `\lcont` command. This is a command which can span *multiple* paragraphs.

After the first paragraph of a list item, include the text `\lcont{`. This indicates that the subsequent paragraph(s) are a *continuation* of the list item that has just been seen. So you can include further paragraphs, and eventually include a closing brace `}` to finish the list continuation. After that, you can either continue adding other items to the original list, or stop immediately and return to writing normal paragraphs of text.

Here's a (long) example.

Here's a list:

```
\n One. This item is followed by a code paragraph:
```

```
\lcont{
```

```
\c code
```

```
\c paragraph
```

```
}
```

```
\n Two. Now when I say \q{two}, I mean:
```

```
\lcont{
```

```
\n Two, part one.
```

```
\n Two, part two.
```

```
\n Two, part three.
```

```
}
```

```
\n Three.
```

The output produced by this fragment is:

Here's a list:

1. One. This item is followed by a code paragraph:

code

paragraph

2. Two. Now when I say ‘two’, I mean:

1. Two, part one.
  2. Two, part two.
  3. Two, part three.
3. Three.

This syntax might seem a little bit inconvenient, and perhaps counter-intuitive: you might expect the enclosing braces to have to go around the *whole* list item, rather than everything except the first paragraph.

`\lcont` is a recent addition to the Halibut input language; previously, *all* lists were required to use no more than one paragraph per list item. So it's certainly true that this feature looks like an afterthought because it *is* an afterthought, and it's possible that if I'd been designing the language from scratch with multiple-paragraph list items in mind, I would have made it look different.

However, the advantage of doing it this way is that no enclosing braces are required in the *common* case: simple lists with only one paragraph per item are really, really easy to write. So I'm not too unhappy with the way it turned out; it obeys the doctrine of making simple things simple, and difficult things possible.

Note that `\lcont` can only be used on `\b`, `\n` and `\dd` paragraphs; it cannot be used on `\dt`.

### 3.3.3 `\rule`: Horizontal rules

The command `\rule`, appearing on its own as a paragraph, will cause a horizontal rule to be drawn, like this:

Some text.

`\rule`

Some more text.

This produces the following output:

Some text.

---

Some more text.

### 3.3.4 `\quote`: Indenting multiple paragraphs as a long quotation

Quoting verbatim text using a code paragraph (section 3.3.1) is not always sufficient for your quoting needs. Sometimes you need to quote some normally formatted text, possibly in multiple paragraphs. This is similar to HTML's `<BLOCKQUOTE>` command.

To do this, you can use the `\quote` command. Like `\lcont`, this is a command which expects to enclose at least one paragraph and possibly more. Simply write `\quote{` at the beginning of your quoted section, and `}` at the end, and the paragraphs in between will be formatted to indicate that they are a quotation.

(This very manual, in fact, uses this feature a lot: all of the examples of Halibut input followed by Halibut output have the output quoted using `\quote`.)

Here's some example Halibut input:

```
In \q{Through the Looking Glass}, Lewis Carroll wrote:

\quote{

\q{The question is,} said Alice, \q{whether you \e{can} make
words mean so many different things.}

\q{The question is,} said Humpty Dumpty, \q{who is to be
master - that's all.}

}
```

So now you know.

The output generated by this is:

```
In ‘Through the Looking Glass’, Lewis Carroll wrote:

    ‘The question is,’ said Alice, ‘whether you can make words mean so many different
    things.’

    ‘The question is,’ said Humpty Dumpty, ‘who is to be master - that's all.’

So now you know.
```

### 3.3.5 \C, \H, \S, \A, \U: Chapter and section headings

Section 1.2 mentions that Halibut numbers the sections of your document automatically, and can generate cross-references to them on request; section 3.2.8 describes the \k and \K commands used to generate the cross-references. This section describes the commands used to set up the sections in the first place.

A paragraph beginning with the \C command defines a chapter heading. The \C command expects to be followed by a pair of braces containing a keyword for the chapter; this keyword can then be used with the \k and \K commands to generate cross-references to the chapter. After the closing brace, the rest of the paragraph is used as the displayed chapter title. So the heading for the current chapter of this manual, for example, is written as

```
\C{input} Halibut input format
```

and this allows me to use the command \k{input} to generate a cross-reference to that chapter somewhere else.

The keyword you supply after one of these commands is allowed to contain escaped special characters (\, \{ and \}), but should not contain any other Halibut markup. It is intended to be a word or two of ordinary text. (This also applies to keywords used in other commands, such as \B and \n).

The next level down from \C is \H, for ‘heading’. This is used in exactly the same way as \C, but section headings defined with \H are considered to be part of a containing chapter, and will be numbered with a pair of numbers. After \H comes \S, and if necessary you can then move on to \S2, \S3 and so on.

For example, here's a sequence of heading commands. Normally these commands would be separated at least by blank lines (because each is a separate paragraph), and probably also by body text; but for the sake of brevity, both of those have been left out in this example.

```
\C{foo} Using Foo
\H{foo-intro} Introduction to Foo
\H{foo-running} Running the Foo program
\S{foo-inter} Running Foo interactively
\S{foo-batch} Running Foo in batch mode
\H{foo-trouble} Troubleshooting Foo
\C{bar} Using Bar instead of Foo
```

This would define two chapters with keywords `foo` and `bar`, which would end up being called Chapter 1 and Chapter 2 (unless there were other chapters before them). The sections `foo-intro`, `foo-running` and `foo-trouble` would be referred to as Section 1.1, Section 1.2 and Section 1.3 respectively; the subsections `foo-inter` and `foo-batch` would be Section 1.2.1 and Section 1.2.2. If there had been a `\S2` command within one of those, it would have been something like Section 1.2.1.1.

If you don't like the switch from `\H` to `\S`, you can use `\S1` as a synonym for `\S` and `\S0` as a synonym for `\H`. Chapters are still designated with `\C`, because they need to be distinguished from other types of chapter such as appendices. (Personally, I like the `\C,\H,\S` notation because it encourages me to think of my document as a hard disk :-)

You can define an appendix by using `\A` in place of `\C`. This is no different from a chapter except that it's given a letter instead of a number, and cross-references to it will say 'Appendix A' instead of 'Chapter 9'. Subsections of an appendix will be numbered 'A.1', 'A.2', 'A.2.1' and so on.

If you want a particular section to be referred to as something other than a 'chapter', 'section' or 'appendix', you can include a second pair of braces after the keyword. For example, if you're writing a FAQ chapter and you want cross-references between questions to refer to 'question 1.2.3' instead of 'section 1.2.3', you can write each section heading as

```
\S{question-about-fish}{Question} What about fish?
```

(The word 'Question' should be given with an initial capital letter. Halibut will lower-case it when you refer to it using `\k`, and will leave it alone if you use `\K`.)

This technique allows you to change the designation of *particular* sections. To make an overall change in what *every* section is called, see section 3.6.

Finally, the `\U` command defines an *unnumbered* chapter. These sometimes occur in books, for specialist purposes such as 'Bibliography' or 'Acknowledgements'. `\U` does not expect a keyword argument, because there is no sensible way to generate an automatic cross-reference to such a chapter anyway.

### 3.3.6 `\copyright`, `\title`, `\versionid`: Miscellaneous blurb commands

These three commands define a variety of special paragraph types. They are all used in the same way: you put the command at the start of a paragraph, and then just follow it with normal text, like this:

```
\title My First Manual
```



The three special paragraph types are:

`\title`

This defines the overall title of the entire document. This title is treated specially in some output formats (for example, it's used in a `<TITLE>` tag in the HTML output), so it needs a special paragraph type to point it out.

`\copyright`

This command indicates that the paragraph attached to it contains a copyright statement for the document. This text is displayed inline where it appears, exactly like a normal paragraph; but in some output formats it is given additional special treatment. For example, Windows Help files have a standard slot in which to store a copyright notice, so that other software can display it prominently.

`\versionid`

This command indicates that the paragraph contains a version identifier, such as those produced by CVS (of the form `$Id: thingy.but,v 1.6 2004/01/01 16:47:48 simon Exp $`). This text will be tucked away somewhere unobtrusive, so that anyone wanting to (for example) report errors to the document's author can pick out the version IDs and send them as part of the report, so that the author can tell at a glance which revision of the document is being discussed.

### 3.3.7 `\#`: Whole-paragraph comments

Section 3.2.9 describes the use of the `\#` command to put a short comment in the middle of a paragraph.

If you need to use a *long* comment, Halibut also allows you to use `\#` without braces, to indicate that an entire paragraph is a comment, like this:

Here's a (fairly short) paragraph which will be displayed.

```
\# Here's a comment paragraph which will not be displayed, no  
matter how long it goes on. All I needed to indicate this was  
the single \# at the start of the paragraph; I don't need one  
on every line or anything like that.
```

Here's another displayed paragraph.

When run through Halibut, this produces the following output:

Here's a (fairly short) paragraph which will be displayed.

Here's another displayed paragraph.

## 3.4 Creating a bibliography

If you need your document to refer to other documents (research papers, books, websites, whatever), you might find a bibliography feature useful.

You can define a bibliography entry using the `\B` command. This looks very like the `\C` command and friends: it expects a keyword in braces, followed by some text describing the document being referred to. For example:

```
\B{freds-book} \q{The Taming Of The Mongoose}, by Fred Bloggs.  
Published by Paperjam & Notoner, 1993.
```

If this bibliography entry appears in the finished document, it will look something like this:

[1] ‘The Taming Of The Mongoose’, by Fred Bloggs. Published by Paperjam & Notoner, 1993.

I say ‘if’ above because not all bibliography entries defined using the `\B` command will necessarily appear in the finished document. They only appear if they are referred to by a `\k` command (see section 3.2.8). This allows you to (for example) maintain a single Halibut source file with a centralised database of *all* the references you have ever needed in any of your writings, include that file in every document you feed to Halibut, and have it only produce the bibliography entries you actually need for each particular document. (In fact, you might even want this centralised source file to be created automatically by, say, a Perl script from BibTeX input, so that you can share the same bibliography with users of other formatting software.)

If you really want a bibliography entry to appear in the document even though no text explicitly refers to it, you can do that using the `\nocite` command:

```
\nocite{freds-book}
```

Normally, each bibliography entry will be referred to (in citations and in the bibliography itself) by a simple reference number, such as [1]. If you would rather use an alternative reference notation, such as [Fred1993], you can use the `\BR` (‘Bibliography Rewrite’) command to specify your own reference format for a particular book:

```
\BR{freds-book} [Fred1993]
```

The keyword you supply after `\B` is allowed to contain escaped special characters (`\\`, `\{` and `\}`), but should not contain any other Halibut markup. It is intended to be a word or two of ordinary text. (This also applies to keywords used in other commands, such as `\n` and `\C`).

## 3.5 Creating an index

Halibut contains a comprehensive indexing mechanism, which attempts to be reasonably easy to use in the common case in spite of its power.

### 3.5.1 Simple indexing

In normal usage, you should be able to add index terms to your document simply by using the `\i` command to wrap one or two words at a time. For example, if you write

```
The \i{hippopotamus} is a particularly large animal.
```

then the index will contain an entry under ‘hippopotamus’, pointing to that sentence (or as close to that sentence as the output format sensibly permits).

You can wrap more than one word in `\i` as well:

```
We recommend using a \i{torque wrench} for this job.
```

## 3.5.2 Special cases of indexing

If you need to index a computer-related term, you can use the special case `\i\c` (or `\i\cw` if you prefer):

The `\i\c{grep}` command is what you want here.

This will cause the word ‘grep’ to appear in code style, as if the `\i` were not present and the input just said `\c{grep}`; the word will also appear in code style in the actual index.

If you want to simultaneously index and emphasise a word, there's another special case `\i\e`:

This is what we call a `\i\e{paper jam}`.

This will cause the words ‘paper jam’ to be emphasised in the document, but (unlike the behaviour of `\i\c`) they will *not* be emphasised in the index. This different behaviour is based on an expectation that most people indexing a word of computer code will still want it to look like code in the index, whereas most people indexing an emphasised word will *not* want it emphasised in the index.

(In fact, *no* emphasis in the text inside `\i` will be preserved in the index. If you really want a term in the index to appear emphasised, you must say so explicitly using `\IM`; see section 3.5.3.)

Sometimes you might want to index a term which is not explicitly mentioned, but which is highly relevant to the text and you think that somebody looking up that term in the index might find it useful to be directed here. To do this you can use the `\I` command, to create an *invisible* index tag:

If your printer runs out of toner, `\I{replacing toner cartridge}`here is what to do:

This input will produce only the output ‘If your printer runs out of toner, here is what to do’; but an index entry will show up under ‘replacing toner cartridge’, so that if a user thinks the obvious place to start in the index is under R for ‘replacing’, they will find their way here with a minimum of fuss.

(It's worth noting that there is no functional difference between `\i{foo}` and `\I{foo}foo`. The simple `\i` case is only a shorthand for the latter.)

Finally, if you want to index a word at the start of a sentence, you might very well not want it to show up with a capital letter in the index. For this, Halibut provides the `\ii` command, for ‘index (case-)insensitively’. You use it like this:

`\ii{Lions}` are at the top of the food chain in this area.

This is equivalent to `\I{lions}Lions`; in other words, the text will say ‘Lions’, but it will show up in the index as ‘lions’. The text inside `\ii` is converted entirely into lower case before being added to the index data.

## 3.5.3 Fine-tuning the index

Halibut's index mechanism as described so far still has a few problems left:

- In a reasonably large index, it's often difficult to predict which of several words a user will think of first when trying to look something up. For example, if they want to know how

to replace a toner cartridge, they might look up ‘replacing’ or they might look up ‘toner cartridge’. You probably don’t really want to have to try to figure out which of those is more likely; instead, what you’d like is to be able to effortlessly index the same set of document locations under *both* terms.

- Also, you may find you’ve indexed the same concept under multiple different index terms; for example, there might be several instances of `\i{frog}` and several of `\i{frogs}`, so that you’d end up with two separate index entries for what really ought to be the same concept.
- You might well not want the word ‘grep’ to appear in the index without explanation; you might prefer it to say something more verbose such as ‘grep command’, so that a user encountering it in the index has some idea of what it is *without* having to follow up the reference. However, you certainly don’t want to have to write `\I{\cw{grep} command}\c{grep}` every time you want to add an index term for this! You wanted to write `\i\c{grep}` as shown in the previous section, and tidy it all up afterwards.

All of these problems can be cleaned up by the `\IM` (for ‘Index Modification’) command. `\IM` expects to be followed by one or more pairs of braces containing index terms as seen in the document, and then a piece of text (not in braces) describing how it should be shown in the index.

So to rewrite the `grep` example above, you might do this:

```
\IM{grep} \cw{grep} command
```

This will arrange that the set of places in the document where you asked Halibut to index ‘grep’ will be listed under ‘grep command’ rather than just under ‘grep’.

You can specify more than one index term in a `\IM` command; so to merge the index terms ‘frog’ and ‘frogs’ into a single term, you might do this:

```
\IM{frog}{frogs} frog
```

This will arrange that the single index entry ‘frog’ will list *all* the places in the document where you asked Halibut to index either ‘frog’ or ‘frogs’.

You can use multiple `\IM` commands to replicate the same set of document locations in more than one index entry. For example:

```
\IM{replacing toner cartridge} replacing toner cartridge
\IM{replacing toner cartridge} toner cartridge, replacing
```

This will arrange that every place in the document where you have indexed ‘replacing toner cartridge’ will be listed both there *and* under ‘toner cartridge, replacing’, so that no matter whether the user looks under R or under T they will still find their way to the same parts of the document.

In this example, note that although the first `\IM` command *looks* as if it’s a tautology, it is still necessary, because otherwise those document locations will *only* be indexed under ‘toner cartridge, replacing’. If you have *no* explicit `\IM` commands for a particular index term, then Halibut will assume a default one (typically `\IM{foo} foo`, although it might be `\IM{foo} \c{foo}` if you originally indexed using `\i\c`); but as soon as you specify an explicit `\IM`, Halibut discards its default implicit one, and you must then specify that one

explicitly as well if you wanted to keep it.

### 3.5.4 Indexing terms that differ only in case

The *tags* you use to define an index term (that is, the text in the braces after `\i`, `\I` and `\IM`) are treated case-insensitively by Halibut. So if, as in this manual itself, you need two index terms that differ only in case, doing this will not work:

The `\i\c{\c}` command defines computer code.

The `\i\c{\C}` command defines a chapter.

Halibut will treat these terms as the same, and will fold the two sets of references into one combined list (although it will warn you that it is doing this). The idea is to ensure that people who forget to use `\ii` find out about it rather than Halibut silently generating a bad index; checking an index for errors is very hard work, so Halibut tries to avoid errors in the first place as much as it can.

If you do come across this situation, you will need to define two distinguishable index terms. What I did in this manual was something like this:

The `\i\c{\c}` command defines computer code.

The `\I{\C-upper}\c{\C}` command defines a chapter.

`\IM{\C-upper} \c{\C}`

The effect of this will be two separate index entries, one reading `\c` and the other reading `\C`, pointing to the right places.

## 3.6 Configuring Halibut

Halibut uses the `\cfg` command to allow you to configure various aspects of its functionality.

The `\cfg` command expects to be followed by at least one pair of braces, and usually more after that. The first pair of braces contains a keyword indicating what aspect of Halibut you want to configure, and the meaning of the one(s) after that depends on the first keyword.

The current list of configuration keywords in the main Halibut code is quite small. Here it is in full:

`\cfg{chapter} {new chapter name}`

This tells Halibut that you don't want to call a chapter a chapter any more. For example, if you give the command `\cfg{chapter} {Book}`, then any chapter defined with the `\C` command will be labelled 'Book' rather than 'Chapter', both in the section headings and in cross-references. This is probably most useful if your document is not written in English.

Your replacement name should be given with a capital letter. Halibut will leave it alone if it appears at the start of a sentence (in a chapter title, or when `\K` is used), and will lower-case it otherwise (when `\k` is used).

`\cfg{section} {new section name}`

Exactly like `chapter`, but changes the name given to subsections of a chapter.

`\cfg{appendix}{new appendix name}`

Exactly like `chapter`, but changes the name given to appendices.

`\cfg{contents}{new contents name}`

This changes the name given to the contents section (by default ‘Contents’) in back ends which generate one.

`\cfg{index}{new index name}`

This changes the name given to the index section (by default ‘Index’) in back ends which generate one.

`\cfg{input-charset}{character set name}`

This tells Halibut what character set you are writing your input file in. By default, it is assumed to be US-ASCII (meaning *only* plain ASCII, with no accented characters at all).

You can specify any well-known name for any supported character set. For example, `iso-8859-1`, `iso8859-1` and `iso_8859-1` are all recognised, `GB2312` and `EUC-CN` both work, and so on. (You can list character sets known to Halibut with by invoking it with the `--list-charsets` option; see section 2.1.)

This directive takes effect immediately after the `\cfg` command. All text after that until the end of the input file is expected to be in the new character set. You can even change character set several times within a file if you really want to.

When Halibut reads the input file, everything you type will be converted into Unicode from the character set you specify here, will be processed as Unicode by Halibut internally, and will be written to the various output formats in whatever character sets they deem appropriate.

`\cfg{quotes}{open-quote}{close-quote}[{open-quote}{close-quote...}]`

This specifies the quote characters which should be used. You should separately specify the open and close quote marks; each quote mark can be one character (`\cfg{quotes}{`}{'}`), or more than one (`\cfg{quotes}{<}{>}`).

`\cfg{quotes}` can be overridden by configuration directives for each individual backend (see chapter 4); it is a convenient way of setting quote characters for all backends at once.

All backends use these characters in response to the `\q` command (see section 3.2.3). Some (such as the text backend) use them for other purposes too.

You can specify multiple fallback options in this command (a pair of open and close quotes, each in their own braces, then another pair, then another if you like), and Halibut will choose the first pair which the output character set supports (Halibut will always use a matching pair). (This is to allow you to configure quote characters once, generate output in several different character sets, and have Halibut constantly adapt to make the best use of the current encoding.) For example, you might write

`\cfg{quotes}{\u201c}{\u201d}{ " }{ " }`

and Halibut would use the Unicode matched double quote characters if possible, and fall

back to ASCII double quotes otherwise. If the output character set were to contain U+201C but not U+201D, then Halibut would fall back to using the ASCII double quote character as *both* open and close quotes. (No known character set is that silly; I mention it only as an example.)

`\cfg{quotes}` (and the backend-specific versions) apply to the *entire* output; it's not possible to change quote characters partway through the output.

In addition to these configuration commands, there are also configuration commands provided by each individual output format. These configuration commands are discussed along with each output format, in chapter 4.

The default settings for the above options are:

```
\cfg{chapter}{Chapter}
\cfg{section}{Section}
\cfg{appendix}{Appendix}
\cfg{contents}{Contents}
\cfg{index}{Index}
\cfg{input-charset}{ASCII}
```

The default for `\cfg{input-charset}` can be changed with the `--input-charset` option; see section 2.1. The default settings for `\cfg{quotes}` are backend-specific; see chapter 4.

## 3.7 Defining macros

If there's a complicated piece of Halibut source which you think you're going to use a lot, you can define your own Halibut command to produce that piece of source.

In section 3.2.7, there is a sample piece of code which prints a Euro sign, or replaces it with 'EUR' if the Euro sign is not available:

```
This is likely to cost \u20AC{EUR\_}2500 at least.
```

If your document quotes a *lot* of prices in Euros, you might not want to spend all your time typing that out. So you could define a macro, using the `\define` command:

```
\define{eur} \u20AC{EUR\_}
```

Your macro names may include Roman alphabetic characters (a-z, A-Z) and ordinary Arabic numerals (0-9), but nothing else. (This is general syntax for all of Halibut's commands, except for a few special ones such as `\_` and `\-` which consist of a single punctuation character only.)

Then you can just write ...

```
This is likely to cost \eur 2500 at least.
```

... except that that's not terribly good, because you end up with a space between the Euro sign and the number. (If you had written `\eur2500`, Halibut would have tried to interpret it as a macro command called `eur2500`, which you didn't define.) In this case, it's helpful to use the special `\.` command, which is defined to do nothing at all! But it acts as a separator between your macro and the next character:

```
This is likely to cost \eur\.2500 at least.
```

This way, you will see no space between the Euro sign and the number (although, of course, there will be space between ‘EUR’ and the number if the Euro sign is not available, because the macro definition specifically asked for it).



## Chapter 4: Halibut output formats

---

This chapter describes each of Halibut's current output formats. It gives some general information about the format, and also describes all the configuration directives which are specific to that format.

### 4.1 Plain text

This output format generates the document as a single plain text file. No table of contents or index is generated.

The precise formatting of the text file can be controlled by a variety of configuration directives. They are listed in the following subsections.

#### 4.1.1 Output file name

```
\cfg{text-filename}{filename}
```

Sets the output file name in which to store the text file. This directive is implicitly generated if you provide a file name parameter after the command-line option `--text` (see section 2.1).

#### 4.1.2 Indentation and line width

This section describes the configuration directives which control the horizontal dimensions of the output text file: how much paragraphs are indented by and how long the lines are.

```
\cfg{text-width}{width}
```

Sets the width of the main part of the document, in characters. This width will be used for wrapping paragraphs and for centring titles (if you have asked for titles to be centred - see section 4.1.3). This width does *not* include the left indentation set by `\cfg{text-indent}`; if you specify an indent of 8 and a width of 64, your maximum output line length will be 72.

```
\cfg{text-indent}{indent}
```

Sets the left indentation for the document. If you set this to zero, your document will look like an ordinary text file as someone with a text editor might have written it; if you set it above zero, the text file will have a margin down the left in the style of some printed manuals, and you can then configure the section numbers to appear in this margin (see section 4.1.3).

```
\cfg{text-indent-code}{indent}
```

Specifies how many extra characters of indentation (on top of the normal left indent) should be given to `code` paragraphs.

`\cfg{text-list-indent}{indent}`

Specifies how many extra spaces should be used to indent the bullet or number in a bulleted or numbered list. The actual body of the list item will be indented by this much *plus* the value configured by `\cfg{text-listitem-indent}`.

`\cfg{text-listitem-indent}{indent}`

Specifies how many extra spaces should be used to indent the body of a list item, over and above the number configured in `\cfg{text-list-indent}`.

`\cfg{text-indent-preamble}{boolean}`

When this is set to `true`, the document preamble (i.e. any paragraphs appearing before the first chapter heading) will be indented to the level specified by `\cfg{text-indent}`. If this setting is `false`, the document preamble will not be indented at all from the left margin.

### 4.1.3 Configuring heading display

The directives in this section allow you to configure the appearance of the title, chapter and section headings in your text file.

Several of the directives listed below specify the alignment of a heading. These alignment options have three possible values:

`left`

Align the heading to the very left of the text file (column zero).

`leftplus`

Align the section title to the left of the main display region (in other words, indented to the level specified by `\cfg{text-indent}`). The section *number* is placed to the left of that (so that it goes in the margin if there is room).

`centre`

Centre the heading.

Also, several of the directives below specify how a title should be underlined. The parameter to one of these directives should be either blank (`{ }`) or a piece of text which will be repeated to produce the underline. So you might want to specify, for example, `\text-title-underline{=}` but `\text-chapter-underline{-}`.

You can also specify more than one underline setting, and Halibut will choose the first one that the output character set supports. So, for example, you could write `\text-chapter-underline{\u203e}{-}`, and Halibut would use the Unicode ‘OVERLINE’ character where possible and fall back to the ASCII minus sign otherwise.

`\cfg{text-title-align}{alignment}`

Specifies the alignment of the overall document title: `left`, `leftplus` or `centre`.

`\cfg{text-title-underline}{underline-text}`

Specifies how the overall document title should be underlined.

`\cfg{text-chapter-align}{alignment}`

Specifies the alignment of chapter and appendix headings.

`\cfg{text-chapter-underline}{underline-text}`

Specifies how chapter and appendix headings should be underlined.

`\cfg{text-chapter-numeric}{boolean}`

If this is set to `true`, then chapter headings will not contain the word ‘Chapter’ (or whatever other word you have defined in its place - see section 3.3.5 and section 3.6); they will just contain the chapter *number*, followed by the chapter title. If you set this to `false`, chapter headings will be prefixed by ‘Chapter’ or equivalent.

`\cfg{text-chapter-suffix}{text}`

This specifies the suffix text to be appended to the chapter number, before displaying the chapter title. For example, if you set this to ‘: ’, then the chapter title might look something like ‘Chapter 2: Doing Things’.

`\cfg{text-section-align}{level}{alignment}`

Specifies the alignment of section headings at a particular level. The *level* parameter specifies which level of section headings you want to affect: 0 means first-level headings (`\H`), 1 means second-level headings (`\S`), 2 means the level below that (`\S2`), and so on. The *alignment* parameter is treated just like the other alignment directives listed above.

`\cfg{text-section-underline}{level}{underline-text}`

Specifies how to underline section headings at a particular level.

`\cfg{text-section-numeric}{level}{boolean}`

Specifies whether section headings at a particular level should contain the word ‘Section’ or equivalent (if `false`), or should be numeric only (if `true`).

`\cfg{text-section-suffix}{level}{text}`

Specifies the suffix text to be appended to section numbers at a particular level, before displaying the section title.

#### 4.1.4 Configuring the characters used

`\cfg{text-charset}{character set name}`

This tells Halibut what character set the output should be in. Any Unicode characters representable in this set will be output verbatim; any other characters will not be output and their fallback text (if any) will be used instead.

The character set names are the same as for `\cfg{input-charset}` (see section 3.6). However, unlike `\cfg{input-charset}`, this directive affects the *entire* output; it's not possible to switch encodings halfway through.

`\cfg{text-bullet}{text}[\{text...\}]`

This specifies the text which should be used as the bullet in bulleted lists. It can be one character (`\cfg{text-bullet}{-}`), or more than one (`\cfg{text-bullet}{(*)}`).

Like `\cfg{quotes}` (see section 3.6), you can specify multiple possible options after this command, and Halibut will choose the first one which the output character set supports. For example, you might write `\cfg{text-bullet}{\u2022}{\u00b7}{*}`, in which case Halibut would use the Unicode ‘BULLET’ character where possible, fall back to the ISO-8859-1 ‘MIDDLE DOT’ if that wasn’t available, and resort to the ASCII asterisk if all else failed.

`\cfg{text-rule}{text}[\{text...\}]`

This specifies the text which should be used for drawing horizontal rules (generated by `\rule`; see section 3.3.3). It can be one character, or more than one. The string you specify will be repeated to reach the required width, so you can specify something like ‘==’ to get a rule that looks like ==-==.

Like `\cfg{text-bullet}`, you can specify multiple fallback options in this command.

`\cfg{text-quotes}{open-quote}{close-quote}[\{open-quote}{close-quote...\}]`

This specifies a set of quote characters for the text backend, overriding any defined by `\cfg{quotes}`. It has the same syntax (see section 3.6).

In this backend, these quotes will also be used to mark text enclosed in the `\c` command (see section 3.2.2).

`\cfg{text-emphasis}{start-emph}{end-emph}[\{start-emph}{end-emph...\}]`

This specifies the characters which should be used to surround emphasised text (written using the `\e` command; see section 3.2.1).

You should separately specify the start-emphasis and end-emphasis text, each of which can be more than one character if you want. Also, like `\cfg{text-quotes}`, you can specify multiple pairs of fallback options in this command, and Halibut will always use a matching pair.

## 4.1.5 Miscellaneous configuration options

`\cfg{text-list-suffix}{text}`

This text is appended to the number on a numbered list item (see section 3.3.2.2). So if you want to label your lists as ‘1)’, ‘2)’ and so on, then you would write `\cfg{text-list-suffix}{)}`.

`\cfg{text-versionid}{boolean}`

If this is set to `true`, version ID paragraphs (defined using the `\versionid` command - see section 3.3.6) will be included at the bottom of the text file. If it is set to `false`, they will be omitted completely.

### 4.1.6 Default settings

The default settings for Halibut's plain text output format are:

```
\cfg{text-filename}{output.txt}

\cfg{text-width}{68}
\cfg{text-indent}{7}
\cfg{text-indent-code}{2}
\cfg{text-list-indent}{1}
\cfg{text-listitem-indent}{3}
\cfg{text-indent-preamble}{false}

\cfg{text-title-align}{centre}
\cfg{text-title-underline}{\u2550}{=}

\cfg{text-chapter-align}{left}
\cfg{text-chapter-underline}{\u203e}{-}
\cfg{text-chapter-numeric}{false}
\cfg{text-chapter-suffix}{: }

\cfg{text-section-align}{0}{leftplus}
\cfg{text-section-underline}{0}{}
\cfg{text-section-numeric}{0}{true}
\cfg{text-section-suffix}{0}{ }

\cfg{text-section-align}{1}{leftplus}
\cfg{text-section-underline}{1}{}
\cfg{text-section-numeric}{1}{true}
\cfg{text-section-suffix}{1}{ }

... and so on for all section levels below this ...

\cfg{text-charset}{ASCII}
\cfg{text-bullet}{\u2022}{-}
\cfg{text-rule}{\u2500}{-}
\cfg{text-quotes}{\u2018}{\u2019}{`}{'}
\cfg{text-emphasis}{_}{_}

\cfg{text-list-suffix}{.}
\cfg{text-versionid}{true}
```

## 4.2 HTML

This output format generates an HTML version of the document. By default, this will be in multiple files, starting with `Contents.html` and splitting the document into files by chapter and/or subsection. You can configure precisely how the text is split between HTML files using the configuration commands described in this section. In particular, you can configure Halibut to output one single HTML file instead of multiple ones.

Configuration directives with an `xhtml-` prefix are synonyms for those with an `html-` prefix.

## 4.2.1 Controlling the output file names

`\cfg{html-contents-filename}{filename}`

Sets the output file name in which to store the top-level contents page. Since this is the first page a user ought to see when beginning to read the document, a good choice in many cases might be `index.html` (although this is not the default, for historical reasons).

`\cfg{html-index-filename}{filename}`

Sets the file name in which to store the document's index.

`\cfg{html-template-filename}{template}`

Provides a template to be used when constructing the file names of each chapter or section of the document. This template should contain at least one *formatting command*, in the form of a per cent sign followed by a letter. (If you need a literal per cent sign, you can write `%%`.)

The formatting commands used in this template are:

`%N`

Expands to the visible title of the section, with white space removed. So in a chapter declared as `\C{fish} Catching Fish`, this formatting command would expand to `CatchingFish`.

`%n`

Expands to the type and number of the section, without white space. So in chapter 1 this would expand to `Chapter1`; in section A.4.3 it would expand to `SectionA.4.3`, and so on. If the section has no number (an unnumbered chapter created using `\U`), this directive falls back to doing the same thing as `%N`.

`%b`

Expands to the number of the section, in a format suitable for an HTML fragment name. The first character of the section type is prepended to the section number. So in chapter 1 this would expand to `C1`; in section A.4.3 it would expand to `SA.4.3`, and so on. If the section has no number (an unnumbered chapter created using `\U`), this directive falls back to doing the same thing as `%N`.

`%k`

Expands to the internal keyword specified in the section title. So in a chapter declared as `\C{fish} Catching Fish`, this formatting command would expand to `fish`. If the section has no keyword (an unnumbered chapter created using `\U`), this directive falls back to doing the same thing as `%N`.

These formatting directives can also be used in the `\cfg{html-template-fragment}` configuration directive (see section 4.2.7).

`\cfg{html-single-filename}{filename}`

Sets the file name in which to store the entire document, if Halibut is configured (using `\cfg{html-leaf-level}{0}`) to produce a single self-contained file. Both this

directive *and* `\cfg{html-leaf-level}{0}` are implicitly generated if you provide a file name parameter after the command-line option `--html` (see section 2.1).

## 4.2.2 Controlling the splitting into HTML files

By default, the HTML output from Halibut is split into multiple files. Each file typically contains a single chapter or section and everything below it, unless subsections of that chapter are themselves split off into further files.

Most files also contain a contents section, giving hyperlinks to the sections in the file and/or the sections below it.

The configuration directives listed below allow you to configure the splitting into files, and the details of the contents sections.

`\cfg{html-leaf-level}{depth}`

This setting indicates the depth of section which should be given a ‘leaf’ file (a file with no sub-files). So if you set it to 1, for example, then every chapter will be given its own HTML file, plus a top-level contents file. If you set this to 2, then each chapter *and* each `\H` section will have a file, and the chapter files will mostly just contain links to their sub-files.

If you set this option to zero, then the whole document will appear in a single file. If you do this, Halibut will call that file `Manual.html` instead of `Contents.html` by default.

This option is automatically set to zero if you provide a file name parameter after the command-line option `--html` (see section 2.1), because you have specified a single file name and so Halibut assumes you want the whole document to be placed in that file.

You can also specify the special name `infinity` (or `infinite` or `inf`) if you want to ensure that *every* section and subsection ends up in a separate file no matter how deep you go.

`\cfg{html-contents-depth}{level}{depth}`

This directive allows you to specify how deep any contents section in a particular level of file should go.

The *level* parameter indicates which level of contents section you are dealing with. 0 denotes the main contents section in the topmost file `Contents.html`; 1 denotes a contents section in a chapter file; 2 is a contents section in a file containing a `\H` heading, and so on.

The *depth* parameter indicates the maximum depth of heading which will be shown in this contents section. Again, 1 denotes a chapter, 2 is a `\H` heading, 3 is a `\S` heading, and so on.

So, for example: `\cfg{html-contents-depth}{1}{3}` instructs Halibut to put contents links in chapter files for all sections down to `\S` level, but not to go into any more detail than that.

For backwards compatibility, the alternative syntax `\cfg{html-contents-depth-level}{depth}` is also supported.

```
\cfg{html-leaf-contains-contents}{boolean}
```

If you set this to `true`, then each leaf file will contain its own contents section which summarises the text within it.

```
\cfg{html-leaf-smallest-contents}{number}
```

Contents sections in leaf files are not output at all if they contain very few entries (on the assumption that it just isn't worth bothering). This directive configures the minimum number of entries required in a leaf contents section to make Halibut bother generating it at all.

### 4.2.3 Including pieces of your own HTML

The directives in this section allow you to supply pieces of verbatim HTML code, which will be included in various parts of the output files.

Note that none of Halibut's usual character set translation is applied to this code; it is assumed to already be in a suitable encoding for the target HTML files.

```
\cfg{html-head-end}{HTML text}
```

The text you provide in this directive is placed at the end of the `<HEAD>` section of each output HTML file. So this is a good place to put, for example, a link to a CSS stylesheet.

```
\cfg{html-local-head}{HTML text}
```

This configuration directive is local: you specify it within a document section, and it acts on that section only.

The text you provide in this directive is placed at the end of the `<HEAD>` section of whichever output HTML file contains the section in which the directive was placed. You can specify this directive multiple times in multiple sections if you like.

This directive is particularly useful for constructing MacOS on-line help, which is mostly normal HTML but which requires a special `<META NAME="AppleTitle">` tag in the topmost source file. You can arrange this by placing this configuration directive in the preamble or the introduction section, something like this:

```
\cfg{html-local-head}{<meta name="AppleTitle"  
content="MyApp Help">}
```

```
\cfg{html-body-tag}{HTML text}
```

The text you provide in this directive is used in place of the `<BODY>` tag in each output file. So if you wanted to define a background colour, for example, you could write `\cfg{html-body-tag}{<body bg="#123456">}`.

```
\cfg{html-body-start}{HTML text}
```

The text you provide in this directive is placed at the beginning of the `<BODY>` section of each output HTML file. So if you intend your HTML files to be part of a web site with a standard house style, and the style needs a header at the top of every page, this is where you can add that header.



```
\cfg{html-body-end} {HTML text}
```

The text you provide in this directive is placed at the end of the <BODY> section of each output HTML file, before any address section. So if you intend your HTML files to be part of a web site with a standard house style, and the style needs a footer at the bottom of every page, this is where you can add that footer.

```
\cfg{html-address-start} {HTML text}
```

The text you provide in this directive is placed at the beginning of the <ADDRESS> section at the bottom of each output HTML file. This might be a good place to put authors' contact details, for example.

```
\cfg{html-address-end} {HTML text}
```

The text you provide in this directive is placed at the end of the <ADDRESS> section at the bottom of each output HTML file, after the version IDs (if present).

```
\cfg{html-navigation-attributes} {HTML attributes}
```

The text you provide in this directive is included inside the <P> tag containing the navigation links at the top of each page ('Previous' / 'Contents' / 'Next'). So if you wanted the navigation links to have a particular CSS style, you could write `\cfg{html-navigation-attributes} {class="foo"}`, and the navigation-links paragraph would then begin with the tag `<p class="foo">`.

#### 4.2.4 Configuring heading display

```
\cfg{html-chapter-numeric} {boolean}
```

If this is set to `true`, then chapter headings will not contain the word 'Chapter' (or whatever other word you have defined in its place - see section 3.3.5 and section 3.6); they will just contain the chapter *number*, followed by the chapter title. If you set this to `false`, chapter headings will be prefixed by 'Chapter' or equivalent.

```
\cfg{html-chapter-suffix} {text}
```

This specifies the suffix text to be appended to the chapter number, before displaying the chapter title. For example, if you set this to `' : '`, then the chapter title might look something like 'Chapter 2: Doing Things'.

```
\cfg{html-section-numeric} {level} {boolean}
```

Specifies whether section headings at a particular level should contain the word 'Section' or equivalent (if `false`), or should be numeric only (if `true`). The *level* parameter specifies which level of section headings you want to affect: 0 means first-level headings (\H), 1 means second-level headings (\S), 2 means the level below that (\S2), and so on.

```
\cfg{html-section-suffix} {level} {text}
```

Specifies the suffix text to be appended to section numbers at a particular level, before displaying the section title.

#### 4.2.5 Configuring standard text

These directives let you fine-tune the names Halibut uses in places such as the navigation bar to

refer to various parts of the document, and other standard pieces of text, for instance to change them to a different language.

```
\cfg{html-preamble-text}{text}
```

```
\cfg{html-contents-text}{text}
```

```
\cfg{html-index-text}{text}
```

Text used to refer to the preamble (i.e., any paragraphs before the first chapter heading), contents, and index respectively, in the navigation bar, contents, and index.

(html-contents-text and html-index-text override the cross-format configuration keywords contents and index (see section 3.6, if both appear. They are legacy keywords preserved for backwards compatibility; you should generally use contents and index.)

```
\cfg{html-title-separator}{text}
```

If multiple headings are used in a file's <TITLE> tag, this text is used to separate them.

```
\cfg{html-index-main-separator}{text}
```

Separator between index term and references in the index.

```
\cfg{html-index-multiple-separator}{text}
```

Separator between multiple references for a single index term in the index.

```
\cfg{html-pre-versionid}{text}
```

```
\cfg{html-post-versionid}{text}
```

Text surrounding each output version ID paragraph.

```
\cfg{html-nav-prev-text}{text}
```

```
\cfg{html-nav-next-text}{text}
```

The text used for the ‘previous page’ and ‘next page’ links on the navigation bar.

```
\cfg{html-nav-separator}{text}
```

Separator between links in the navigation bar.

## 4.2.6 Configuring the characters used

Unlike the other backends, HTML does not have a single `\cfg{html-charset}` directive, as there are several levels of character encoding to consider.

The character set names are the same as for `\cfg{input-charset}` (see section 3.6). However, unlike `\cfg{input-charset}`, these directives affect the *entire* output; it's not possible to switch encodings halfway through.

```
\cfg{html-output-charset}{character set name}
```

The character encoding of the HTML file to be output. Unicode characters in this encoding's repertoire are included literally rather than as HTML entities.

`\cfg{html-restrict-charset} {character set name}`

Only Unicode characters representable in this character set will be output; any others will be omitted and use their fallback text, if any. Characters not in ‘html-output-charset’ will be represented as HTML numeric entities.

`\cfg{html-quotes} {open-quote} {close-quote} [{open-quote} {close-quote...}]`

Specifies the quotation marks to use, overriding any `\cfg{quotes}` directive. You can specify multiple fallback options. Works exactly like the `\cfg{text-quotes}` directive (see section 4.1.4).

## 4.2.7 Miscellaneous options

`\cfg{html-version} {version}`

Identifies the precise version of HTML that is output. This affects the declaration within the HTML, and also has minor effects on the body of the HTML so that it is valid for the declared version. The available variants are:

`html3.2`

W3C HTML 3.2

`html4`

W3C HTML 4.01 Strict

`iso-html`

ISO/IEC 15445:2000

`xhtml1.0transitional`

W3C XHTML 1.0 Transitional

`xhtml1.0strict`

W3C XHTML 1.0 Strict

`\cfg{html-template-fragment} {template} [{template} {...}]`

This directive lets you specify a template, with exactly the same syntax used in `\cfg{html-template-filename}` (see section 4.2.1), to be used for the anchor names (`<A NAME="...">`) used to allow URLs to refer to specific sections within a particular HTML file. So if you set this to ‘%k’, for example, then each individual section in your document will be addressable by means of a URL ending in a # followed by your internal section keyword.

If more than one template is specified, anchors are generated in all the specified formats; Halibut's own cross-references are generated with the first template.

Characters that are not permitted in anchor names are stripped. If there are no valid characters left, or a fragment is non-unique, Halibut starts inventing fragment names and suffixes as appropriate.

Note that there are potentially fragment names that are not controlled by this mechanism, such as index references.

```
\cfg{html-versionid}{boolean}
```

If this is set to `true`, version ID paragraphs (defined using the `\versionid` command - see section 3.3.6) will be included visibly in the `<ADDRESS>` section at the bottom of each HTML file. If it is set to `false`, they will only be included as HTML comments.

```
\cfg{html-rellinks}{boolean}
```

If this is set to `true`, machine-readable relational links will be emitted in each HTML file (`<LINK REL="next">` and so on within the `<HEAD>` section) providing links to related files. The same set of links are provided as in the navigation bar (with which this should not be confused).

Some browsers make use of this semantic information, for instance to allow easy navigation through related pages, and to prefetch the next page. (Search engines can also make use of it.) However, many browsers ignore this markup, so it would be unwise to rely on it for navigation.

The use and rendering of this information is entirely up to the browser; none of the other Halibut options for the navigation bar will have any effect.

```
\cfg{html-suppress-navlinks}{boolean}
```

If this is set to `true`, the usual navigation links within the *body* of each HTML file (near the top of the rendered page) will be suppressed.

```
\cfg{html-suppress-address}{boolean}
```

If this is set to `true`, the `<ADDRESS>` section at the bottom of each HTML file will be omitted completely. (This will therefore also cause version IDs not to be included visibly.)

```
\cfg{html-author}{text}
```

The text supplied here goes in a `<META name="author">` tag in the output HTML files, so that browsers which support this can automatically identify the author of the document.

```
\cfg{html-description}{text}
```

The text supplied here goes in a `<META name="description">` tag in the output HTML files, so that browsers which support this can easily pick out a brief description of the document.

## 4.2.8 Generating MS Windows HTML Help

The HTML files output from Halibut's HTML back end can be used as input to the MS Windows HTML Help compiler. In order to do this, you also need some auxiliary files: a project file, and (probably) a contents file and an index file. Halibut can optionally generate those as well.

To enable the generation of MS HTML Help auxiliary files, use the following configuration directives:

`\cfg{html-mshtmlhelp-project}{filename}`

Instructs Halibut to output an HTML Help project file with the specified name. You will almost certainly want the filename to end in the extension `.hhp` (although Halibut will not enforce this). If you use this option, you must also use the `html-mshtmlhelp-chm` option to specify the desired name of the compiled help file.

`\cfg{html-mshtmlhelp-chm}{filename}`

Specifies the desired name of the compiled HTML Help file. You will almost certainly want this to have the extension `.chm` (although Halibut will not enforce this). The name you specify here will be written into the help project file. If you specify this option, you must also use the `html-mshtmlhelp-project` option to request a help project file in the first place.

`\cfg{html-mshtmlhelp-contents}{filename}`

Instructs Halibut to output an HTML Help contents file with the specified name, and refer to it in the help project file. You will almost certainly want the filename to end in the extension `.hhc` (although Halibut will not enforce this). This option will be ignored if you have not also specified a help project file.

Creating a contents file like this causes the HTML Help viewer to display a contents tree in the pane to the left of the main text window. You can choose to generate an HTML Help project without this feature, in which case the user will still be able to navigate around the document by using the ordinary internal links in the HTML files themselves just as if it were a web page. However, using a contents file is recommended.

`\cfg{html-mshtmlhelp-index}{filename}`

Instructs Halibut to output an HTML Help index file with the specified name, and refer to it in the help project file. You will almost certainly want the filename to end in the extension `.hhk` (although Halibut will not enforce this). This option will be ignored if you have not also specified a help project file.

Specifying this option suppresses the generation of an HTML-based index file (see `\cfg{html-index-filename}` in section 4.2.1).

Creating an index file like this causes the HTML Help viewer to provide a list of index terms in a pane to the left of the main text window. You can choose to generate an HTML Help project without this feature, in which case a conventional HTML index will be generated instead (assuming you have any index terms at all defined) and the user will still be able to use that. However, using an index file is recommended.

Halibut will not output an index file at all, or link to one from the help project file, if your document contains no index entries.

If you use the above options, Halibut will output a help project file which you should be able to feed straight to the command-line MS HTML Help compiler (`HHC.EXE`), or load into the MS HTML Help Workshop (`HHW.EXE`).

You may also wish to alter other HTML configuration options to make the resulting help file look more like a help file and less like a web page. A suggested set of additional configuration options for HTML Help is as follows:

- `\cfg{html-leaf-level}{infinite}`, because HTML Help works best with lots of small files ('topics') rather than a few large ones. In particular, the contents and index mechanisms can only reference files, not subsections within files.
- `\cfg{html-leaf-contains-contents}{false}`, to suppress the contents list above the main text of each bottom-level file.
- `\cfg{html-suppress-navlinks}{true}`, because HTML Help has its own navigation facilities and it looks a bit strange to duplicate them.
- `\cfg{html-suppress-address}{true}`, because the `<ADDRESS>` section makes less sense in a help file than it does on a web page.

## 4.2.9 Default settings

The default settings for Halibut's HTML output format are:

```
\cfg{html-contents-filename}{Contents.html}
\cfg{html-index-filename}{IndexPage.html}
\cfg{html-template-filename}{%n.html}
\cfg{html-single-filename}{Manual.html}

\cfg{html-leaf-level}{2}
\cfg{html-leaf-contains-contents}{false}
\cfg{html-leaf-smallest-contents}{4}
\cfg{html-contents-depth}{0}{2}
\cfg{html-contents-depth}{1}{3}
... and so on for all section levels below this ...

\cfg{html-head-end}{}
\cfg{html-body-tag}{<body>}
\cfg{html-body-start}{}
\cfg{html-body-end}{}
\cfg{html-address-start}{}
\cfg{html-address-end}{}
\cfg{html-navigation-attributes}{}

\cfg{html-chapter-numeric}{false}
\cfg{html-chapter-suffix}{: }

\cfg{html-section-numeric}{0}{true}
\cfg{html-section-suffix}{0}{ }

\cfg{html-section-numeric}{1}{true}
\cfg{html-section-suffix}{1}{ }

... and so on for all section levels below this ...

\cfg{html-preamble-text}{Preamble}
\cfg{html-contents-text}{Contents}
\cfg{html-index-text}{Index}
\cfg{html-title-separator}{ - }
```

```

\cfg{html-index-main-separator}{: }
\cfg{html-index-multiple-separator}{, }
\cfg{html-pre-versionid}{[ ]}
\cfg{html-post-versionid}{[ ]}
\cfg{html-nav-prev-text}{Previous}
\cfg{html-nav-next-text}{Next}
\cfg{html-nav-separator}{ | }

\cfg{html-output-charset}{ASCII}
\cfg{html-restrict-charset}{UTF-8}
\cfg{html-quotes}{\u2018}{\u2019}{" }{" }

\cfg{html-version}{html4}
\cfg{html-template-fragment}{%b}
\cfg{html-versionid}{true}
\cfg{html-rellinks}{true}
\cfg{html-suppress-navlinks}{false}
\cfg{html-suppress-address}{false}
\cfg{html-author}{}
\cfg{html-description}{}

```

## 4.3 Windows Help

This output format generates data that can be used by the Windows Help program WINHLP32.EXE. There are two actual files generated, one ending in .hlp and the other ending in .cnt.

Note that as of 2006, MS is discontinuing the Windows Help format in favour of the newer HTML Help format (.chm files). Halibut is not currently able to generate .chm files directly, but its HTML back end can write out project files suitable for use as input to the MS HTML Help compiler. See section 4.2.8 for more information on this.

Currently, the Windows Help output is hardcoded to be in the ‘Win1252’ character set. (If anyone knows how character sets are encoded in Windows Help files, we’d appreciate help.)

The Windows Help output format supports the following configuration directives:

### 4.3.1 Output file name

```
\cfg{winhelp-filename}{filename}
```

Sets the output file name in which to store the man page. This directive is implicitly generated if you provide a file name parameter after the command-line option `--winhelp` (see section 2.1).

Your output file name should end with .hlp; if it doesn’t, Halibut will append it. Halibut will also generate a contents file (ending in .cnt) alongside the file name you specify.

### 4.3.2 Configuring the characters used

```
\cfg{winhelp-bullet}{text}[{text}...]
```

Specifies the text to use as the bullet in bulleted lists. You can specify multiple fallback options. Works exactly like the `\cfg{text-bullet}` directive (see section 4.1.4).

```
\cfg{winhelp-quotes}{open-quote}{close-quote}[{open-quote}{close-quote...}]
```

Specifies the quotation marks to use, overriding any `\cfg{quotes}` directive. You can specify multiple fallback options. Works exactly like the `\cfg{text-quotes}` directive (see section 4.1.4).

### 4.3.3 Miscellaneous configuration options

```
\cfg{winhelp-contents-titlepage}{title}
```

Sets the text used to describe the help page containing the blurb (see section 3.3.6) and table of contents.

```
\cfg{winhelp-section-suffix}{text}
```

Specifies the suffix text to be appended to section numbers, before displaying the section title. (Applies to all levels.)

```
\cfg{winhelp-list-suffix}{text}
```

This text is appended to the number on a numbered list item, in exactly the same way as `\cfg{text-list-suffix}` (see section 4.1.4).

```
\cfg{winhelp-topic}{topic-name}
```

This directive defines a Windows Help topic name in the current section. Topic names can be used by the program invoking `WINHELP.EXE` to jump straight to a particular section. So you can use this for context-sensitive help.

For example, if you used this directive in a particular section:

```
\cfg{winhelp-topic}{savingfiles}
```

then a Windows application could invoke Windows Help to jump to that particular section in the help file like this:

```
WinHelp(hwnd, "mydoc.hlp", HELP_COMMAND,  
        (DWORD)"JI(`',`savingfiles')");
```

You can use this configuration directive many times, in many different subsections of your document, in order to define a lot of different help contexts which you can use in this way.

### 4.3.4 Default settings

The default settings for the Windows Help output format are:

```
\cfg{winhelp-filename}{output.hlp}
```

```
\cfg{winhelp-bullet}{\u2022}{-}
```

```
\cfg{winhelp-quotes}{\u2018}{\u2019}{"}{"}
```

```
\cfg{winhelp-contents-titlepage}{Title page}
```

```
\cfg{winhelp-section-suffix}{: }
```

```
\cfg{winhelp-list-suffix}{.}
```

and no `\cfg{winhelp-topic}` directives anywhere.



## 4.4 Unix man pages

This output format generates a Unix man page. That is to say, it generates `nroff` input designed to work with the `-mandoc` macro package.

The available configuration options for this format are as follows:

### 4.4.1 Output file name

```
\cfg{man-filename} {filename}
```

Sets the output file name in which to store the man page. This directive is implicitly generated if you provide a file name parameter after the command-line option `--man` (see section 2.1).

### 4.4.2 Configuring headers and footers

```
\cfg{man-identity} {text} {text...}
```

This directive is used to generate the initial `.TH` directive that appears at the top of a man page. It expects to be followed by some number of brace pairs containing text, which will be used in the headers and footers of the formatted output.

A traditional order for the arguments appears to be:

1. The name of the program.
2. The (numeric) manual section.
3. The date that the man page was written.
4. The name of any containing suite of which the program is a part.
5. The name of the author of the man page.

For example, a typical man page might contain

```
\cfg{man-identity} {make-foo} {1} {June 2003} {foo-utils} {Fred  
Bloggs}
```

### 4.4.3 Configuring heading display

```
\cfg{man-headnumbers} {boolean}
```

If this is set to `true`, then section headings in the man page will have their section numbers displayed as usual. If set to `false`, the section numbers will be omitted. (man pages traditionally have section names such as ‘SYNOPSIS’, ‘OPTIONS’ and ‘BUGS’, and do not typically number them, so `false` is the setting which conforms most closely to normal man style.)

```
\cfg{man-mindepth} {depth}
```

If this is set to a number greater than 0, then section headings *higher* than the given depth will not be displayed. If it is set to zero, all section headings will be displayed as normal.

The point of this is so that you can use the same Halibut input file to generate a quick-reference man page for a program, *and* to include that man page as an appendix in your

program's full manual. If you are to include the man page as an appendix, then the internal headings within the page will probably need to be at `\H` or `\S` level; therefore, when you format that input file on its own to create the man page itself, you will need to have defined a `\C` and possibly a `\H` heading beforehand, which you don't want to see displayed.

Here's an example. You might have a file `appendix.but`, which simply says

```
\A{manpages} \cw{man} pages for the Foo tool suite

\cfg{man-mindepth}{2}
```

Then you have a file `make-foo.but`, and probably others like it as well, each of which looks something like this:

```
\cfg{man-identity}{make-foo}{1}{June 2003}{foo-utils}{Fred
Bloggs}

\H{man-foo} \cw{man} page for \c{make-foo}

\S{man-foo-name} NAME

\c{make-foo} - create Foo files for the Foo tool suite

\S{man-foo-synopsis} SYNOPSIS

... and so on ...
```

So when you're generating your main manual, you can include `appendix.but` followed by `make-foo.but` and any other man pages you have, and your man pages will be formatted neatly as part of an appendix. Then, in a separate run of Halibut, you can just do

```
halibut appendix.but make-foo.but
```

and this will generate a man page `output.1`, in which the headings 'man pages for the Foo tool suite' and 'man page for make-foo' will not be displayed because of the `man-mindepth` directive. So the first visible heading in the output man page will be 'NAME', exactly as a user would expect.

#### 4.4.4 Configuring the characters used

```
\cfg{man-charset}{character set}
```

Specifies what character set the output should be in, similarly to `\cfg{text-charset}` (see section 4.1.4).

```
\cfg{man-bullet}{text}[{text}...]
```

Specifies the text to use as the bullet in bulleted lists. You can specify multiple fallback options. Works exactly like the `\cfg{text-bullet}` directive (see section 4.1.4).

```
\cfg{man-rule}{text}[{text...}]
```

This specifies the text which should be used for drawing horizontal rules (generated by `\rule`; see section 3.3.3) when the manual page is rendered into text. It should only be one character long, but otherwise it works like the `\cfg{text-rule}` directive (see

section 4.1.4).

```
\cfg{man-quotes}{open-quote}{close-quote}[open-quote]{close-quote...}]
```

Specifies the quotation marks to use, overriding any `\cfg{quotes}` directive. You can specify multiple fallback options. Works exactly like the `\cfg{text-quotes}` directive (see section 4.1.4).

## 4.4.5 Default settings

The default settings for the man page output format are:

```
\cfg{man-filename}{output.1}
```

```
\cfg{man-identity}{}
```

```
\cfg{man-headnumbers}{false}
```

```
\cfg{man-mindepth}{0}
```

```
\cfg{man-charset}{ASCII}
```

```
\cfg{man-bullet}{\u2022}{o}
```

```
\cfg{man-rule}{\u2500}{-}
```

```
\cfg{man-quotes}{\u2018}{\u2019}{"}{"}
```

## 4.5 GNU info

This output format generates files which can be used with the GNU `info` program.

There are typically multiple output files: a primary file whose name usually ends in `.info`, and one or more subsidiary files whose names have numbers on the end, so that they end in `.info-1`, `.info-2` and so on. Alternatively, this output format can be configured to output a single large file containing the whole document.

The `info` output format supports the following configuration directives:

### 4.5.1 Controlling the output filenames

```
\cfg{info-filename}{filename}
```

Sets the output file name in which to store the `info` file. This directive is implicitly generated if you provide a file name parameter after the command-line option `--info` (see section 2.1).

The suffixes `-1`, `-2`, `-3` and so on will be appended to your output file name to produce any subsidiary files required.

Note that `info` files refer to their own names internally, so these files cannot be renamed after creation and remain useful.

```
\cfg{info-max-file-size}{bytes}
```

Sets the preferred maximum file size for each subsidiary file. As a special case, if you set this to zero, there will be no subsidiary files and the whole document will be placed in a single self-contained output file. (However, note that this file can still not be renamed usefully.)

The preferred maximum file size is only a guideline. Halibut may be forced to exceed it if a single section of the document is larger than the maximum size (since individual `info` nodes may not be split between files).

## 4.5.2 Indentation and line width

`\cfg{info-width}{width}`

Sets the width of the main part of the document, in characters. Works exactly like the `\cfg{text-width}` directive (see section 4.1.2).

`\cfg{info-indent-code}{indent}`

Specifies the extra indentation for code paragraphs. Works exactly like the `\cfg{text-indent-code}` directive (see section 4.1.2).

`\cfg{info-index-width}{width}`

Specifies how much horizontal space to leave in the index node for the text of index terms, before displaying the sections the terms occur in.

`\cfg{info-list-indent}{indent}`

Specifies the extra indentation before the bullet or number in a list item. Works exactly like the `\cfg{text-list-indent}` directive (see section 4.1.2).

`\cfg{info-listitem-indent}{indent}`

Specifies the additional indentation before the body of a list item. Works exactly like the `\cfg{text-listitem-indent}` directive (see section 4.1.2).

## 4.5.3 Configuring heading display

`\cfg{info-section-suffix}{text}`

Specifies the suffix text to be appended to each section number before displaying the section title. For example, if you set this to ‘: ’, then a typical section title might look something like ‘Section 3.1: Something Like This’.

`\cfg{info-underline}{text} [{text} ...]`

Specifies the text to be used to underline section titles. Works very much like the `\cfg{text-chapter-underline}` directive (see section 4.1.3). You can specify more than one option, and Halibut will choose the first one supported by the character set.

## 4.5.4 Controlling the characters used

`\cfg{info-charset}{character set}`

Specifies what character set the output should be in, similarly to `\cfg{text-charset}` (see section 4.1.4).

`\cfg{info-bullet}{text} [{text} ...]`

Specifies the text to use as the bullet in bulleted lists. You can specify multiple fallback options. Works exactly like the `\cfg{text-bullet}` directive (see section 4.1.4).

`\cfg{info-rule}{text}[\{text}...]`

Specifies the text used to draw horizontal rules. You can specify multiple fallback options. Works exactly like the `\cfg{text-rule}` directive (see section 4.1.4).

`\cfg{info-quotes}{open-quote}{close-quote}[\{open-quote}{close-quote}...]`

Specifies the quotation marks to use, overriding any `\cfg{quotes}` directive. You can specify multiple fallback options. Works exactly like the `\cfg{text-quotes}` directive (see section 4.1.4).

`\cfg{info-emphasis}{start-emph}{end-emph}[\{start-emph}{end-emph}...]`

Specifies how to display emphasised text. You can specify multiple fallback options. Works exactly like the `\cfg{text-emphasis}` directive (see section 4.1.4).

## 4.5.5 Miscellaneous configuration options

`\cfg{info-list-suffix}{text}`

Specifies the text to append to the item numbers in a numbered list. Works exactly like the `\cfg{text-list-suffix}` directive (see section 4.1.5).

`\cfg{info-dir-entry}{section}{short name}{long name}[\{keyword}]`

Constructs an `INFO-DIR-ENTRY` section and places it in the header of the Info file. This mechanism is used to automatically generate the `dir` file at the root of a Unix system's info collection.

The parameters to this directive are:

*section*

Specifies the section of the `dir` file in which you want your document referenced. For example, 'Development', or 'Games', or 'Miscellaneous'.

*short name*

Specifies a short name for the directory entry, which will appear at the start of the menu line.

*long name*

Specifies a long name for the directory entry, which will appear at the end of the menu line.

*keyword*

This parameter is optional. If it is present, then the directory entry will cause a jump to a particular subsection of your document, rather than starting at the top. The subsection will be the one referred to by the given keyword (see section 3.3.5 for details about assigning keywords to document sections).

For example, in a document describing many game programs, the configuration directive

```
\cfg{info-dir-entry}{Games}{Chess}{Electronic chess
game}{chess}
```

might produce text in the `dir` file looking something like this:

Games

\* Chess: (mygames)Chapter 3.      Electronic chess game

if the output file were called `mygames.info` and the keyword `chess` had been used to define Chapter 3 of the document.

## 4.5.6 Default settings

The default settings for the `info` output format are:

```
\cfg{info-filename}{output.info}
\cfg{info-max-file-size}{65536}

\cfg{info-width}{70}
\cfg{info-indent-code}{2}
\cfg{info-index-width}{40}
\cfg{info-list-indent}{1}
\cfg{info-listitem-indent}{3}

\cfg{info-section-suffix}{: }
\cfg{info-underline}{\u203e}{-}

\cfg{info-charset}{ASCII}
\cfg{info-bullet}{\u2022}{-}
\cfg{info-rule}{\u2500}{-}
\cfg{info-quotes}{\u2018}{\u2019}{`}{'}
\cfg{info-emphasis}{_}{_}

\cfg{info-list-suffix}{.}

and no \cfg{info-dir-entry} directives.
```

## 4.6 Paper formats

These output formats (currently PDF and PostScript) generate printable manuals. As such, they share a number of configuration directives.

### 4.6.1 PDF

This output format generates a printable manual in PDF format. In addition, it uses some PDF interactive features to provide an outline of all the document's sections and clickable cross-references between sections.

There is one configuration option specific to PDF:

```
\cfg{pdf-filename}{filename}
```

Sets the output file name in which to store the PDF file. This directive is implicitly generated if you provide a file name parameter after the command-line option `--pdf` (see section 2.1).

The default settings for the PDF output format are:

```
\cfg{pdf-filename}{output.pdf}
```

## 4.6.2 PostScript

This output format generates a printable manual in PostScript format. This should look exactly identical to the PDF output (see section 4.6.2), and uses `pdfmark` to arrange that if converted to PDF it will contain the same interactive features.

There is one configuration option specific to PostScript:

```
\cfg{ps-filename}{filename}
```

Sets the output file name in which to store the PostScript file. This directive is implicitly generated if you provide a file name parameter after the command-line option `--ps` (see section 2.1).

The default settings for the PostScript output format are:

```
\cfg{ps-filename}{output.ps}
```

## 4.6.3 Configuring layout and measurements

All measurements are in PostScript points (72 points to the inch).

### 4.6.3.1 Page properties

```
\cfg{paper-page-width}{points}
```

```
\cfg{paper-page-height}{points}
```

Specify the absolute limits of the paper.

```
\cfg{paper-left-margin}{points}
```

```
\cfg{paper-top-margin}{points}
```

```
\cfg{paper-right-margin}{points}
```

```
\cfg{paper-bottom-margin}{points}
```

Specify the margins. Most text appears within these margins, except:

- Section numbers, which appear in the left margin.
- The footer (containing page numbers), which appears in the bottom margin.

### 4.6.3.2 Vertical spacing

```
\cfg{paper-base-leading}{points}
```

Specifies the amount of space between lines of text within a paragraph. (So, if the font size is 12pt and there is 2pt of leading, there will be 14pt between successive baselines.)

```
\cfg{paper-base-para-spacing}{points}
```

Specifies the amount of vertical space between paragraphs. (The vertical space between paragraphs does *not* include `paper-base-leading`.)

### 4.6.3.3 Indentation

`\cfg{paper-list-indent}{points}`

Specifies the indentation of the bullet or number in a bulleted or numbered list, similarly to `\cfg{text-list-indent}` (see section 4.1.2).

`\cfg{paper-listitem-indent}{points}`

Specifies the *extra* indentation for the body of a list item, over and above the amount configured in `\cfg{paper-list-indent}`.

`\cfg{paper-quote-indent}{points}`

Specifies the amount of indentation for a level of quoting. Used for `\quote` (see section 3.3.4) and code quotes with `\c` (see section 3.2.2).

### 4.6.3.4 Headings

`\cfg{paper-chapter-top-space}{points}`

Specifies the space between the top margin and the top of the chapter heading. (Each chapter begins on a new page.)

`\cfg{paper-chapter-underline-thickness}{points}`

Specifies the vertical thickness of the black rule under chapter headings.

`\cfg{paper-chapter-underline-depth}{points}`

Specifies the distance between the base of the chapter heading and the *base* of the underlying rule.

`\cfg{paper-sect-num-left-space}{points}`

Specifies the distance between the left margin and the *right* of section numbers (which are in the left margin).

### 4.6.3.5 Contents and index

`\cfg{paper-contents-index-step}{points}`

`\cfg{paper-contents-margin}{points}`

`\cfg{paper-leader-separation}{points}`

Specifies the horizontal spacing between dots in *leaders* (the dotted lines that appear between section headings and page numbers in the table of contents).

`\cfg{paper-footer-distance}{points}`

Specifies the distance between the bottom margin and the *base* of the footer (which contains page numbers).

`\cfg{paper-index-columns}{columns}`

Specifies the number of columns the index should be divided into.



`\cfg{paper-index-gutter}{points}`

Specifies the amount of horizontal space between index columns.

`\cfg{paper-index-minsep}{points}`

Specifies the minimum allowable horizontal space between an index entry and its page number. If the gap is smaller, the page number is moved to the next line.

#### 4.6.3.6 Fonts

The directives in this section control which fonts Halibut uses for various kinds of text. Directives for setting the font normally take three font names, the first of which is used for normal text, the second for emphasised text, and the third for code. Any fonts which aren't specified are left unchanged. Fonts are named using their PostScript names.

Halibut intrinsically knows about some fonts, and these fonts are also built into all PDF and most PostScript implementations. These are:

- Times-Roman
- Times-Italic
- Times-Bold
- Times-BoldItalic
- Helvetica
- Helvetica-Oblique
- Helvetica-Bold
- Helvetica-BoldOblique
- Courier
- Courier-Oblique
- Courier-Bold
- Courier-BoldOblique

These fonts can be used without further formality. To use any other font, Halibut needs at least to know its measurements, which are provided in an Adobe Font Metrics (AFM) file. Halibut can also embed Type 1 fonts in its PDF and PostScript output if provided with font file in either hexadecimal (PFA) or IBM PC (PFB) format. To provide an AFM, PFA, or PFB file to Halibut, simply name it on Halibut's command line. If a PFA or PFB file is specified, the corresponding AFM file must come first.

Font sizes are specified in PostScript points (72 to the inch).

`\cfg{paper-title-fonts}{normal-font}[{emph-font}][{code-font}]}`

Specifies the fonts to use for text in the document title.

`\cfg{paper-title-font-size}{points}`

Specifies the font size of the document title.

```
\cfg{paper-chapter-fonts}{normal-font}[{emph-font}][{code-font}]
```

Specifies the fonts to use for text in chapter titles.

```
\cfg{paper-chapter-font-size}{points}
```

Specifies the font size of chapter titles.

```
\cfg{paper-section-fonts}{level}{normal-font}[{emph-font}][{code-font}]
```

Specifies the fonts to use for text in section headings at the *level* specified.

```
\cfg{paper-section-font-size}{level}{points}
```

Specifies the font size of section headings at the *level* specified.

```
\cfg{paper-base-fonts}{normal-font}[{emph-font}][{code-font}]
```

Specifies the fonts to use for text in the body text.

```
\cfg{paper-base-font-size}{points}
```

Specifies the font size of body text.

```
\cfg{paper-code-fonts}{bold-font}[{italic-font}][{normal-font}]
```

Specifies the fonts to use for text in code paragraphs. The *bold-font* is used for bold text, the *italic-font* for emphasised text, and the *normal-font* for normal code.

```
\cfg{paper-code-font-size}{points}
```

Specifies the font size of text in code paragraphs.

```
\cfg{paper-pagenum-font-size}{points}
```

Specifies the font size to use for page numbers.

#### 4.6.3.7 Miscellaneous

```
\cfg{paper-rule-thickness}{points}
```

Specifies the vertical thickness of the rule produced by the `\rule` command (see section 3.3.3). (Note that no extra space is reserved for thicker rules.)

#### 4.6.4 Configuring the characters used

```
\cfg{paper-bullet}{text}[{text}...]
```

Specifies the text to use as the bullet in bulleted lists. You can specify multiple fallback options. Works exactly like the `\cfg{text-bullet}` directive (see section 4.1.4).

```
\cfg{paper-quotes}{open-quote}{close-quote}[{open-quote}]{close-quote...}
```

Specifies the quotation marks to use, overriding any `\cfg{quotes}` directive. You can specify multiple fallback options. Works exactly like the `\cfg{text-quotes}` directive (see section 4.1.4).

## 4.6.5 Default settings for paper formats

The default page size corresponds to  $210 \times 297$  mm, i.e., A4 paper.

```
\cfg{paper-page-width}{595}
\cfg{paper-page-height}{842}

\cfg{paper-left-margin}{72}
\cfg{paper-top-margin}{72}
\cfg{paper-right-margin}{72}
\cfg{paper-bottom-margin}{108}

\cfg{paper-base-leading}{1}
\cfg{paper-base-para-spacing}{10}

\cfg{paper-list-indent}{6}
\cfg{paper-listitem-indent}{18}
\cfg{paper-quote-indent}{18}

\cfg{paper-chapter-top-space}{72}
\cfg{paper-chapter-underline-thickness}{3}
\cfg{paper-chapter-underline-depth}{14}
\cfg{paper-sect-num-left-space}{12}

\cfg{paper-contents-index-step}{24}
\cfg{paper-contents-margin}{84}
\cfg{paper-leader-separation}{12}
\cfg{paper-footer-distance}{32}
\cfg{paper-index-columns}{2}
\cfg{paper-index-gutter}{36}
\cfg{paper-index-minsep}{18}

\cfg{paper-base-fonts}{Times-Roman}{Times-Italic}{Courier}
\cfg{paper-base-font-size}{12}
\cfg{paper-code-fonts}{Courier-Bold}{Courier-Oblique}{Courier}
\cfg{paper-code-font-size}{12}
\cfg{paper-title-fonts}{Helvetica-Bold}
    {Helvetica-BoldOblique}{Courier-Bold}
\cfg{paper-title-font-size}{24}
\cfg{paper-chapter-fonts}{Helvetica-Bold}
    {Helvetica-BoldOblique}{Courier-Bold}
\cfg{paper-chapter-font-size}{20}
\cfg{paper-section-fonts}{0}{Helvetica-Bold}
    {Helvetica-BoldOblique}{Courier-Bold}
\cfg{paper-section-font-size}{0}{16}
\cfg{paper-section-fonts}{1}{Helvetica-Bold}
    {Helvetica-BoldOblique}{Courier-Bold}
\cfg{paper-section-font-size}{1}{14}
\cfg{paper-section-fonts}{2}{Helvetica-Bold}
    {Helvetica-BoldOblique}{Courier-Bold}
\cfg{paper-section-font-size}{2}{13}
```

*... and so on for all section levels below this ...*

```
\cfg{paper-pagenum-font-size}{12}
```

```
\cfg{paper-rule-thickness}{1}
```

```
\cfg{paper-bullet}{\u2022}{-}
```

```
\cfg{paper-quotes}{\u2018}{\u2019}{'}{'}
```

## Appendix A: Halibut Licence

---

Halibut is copyright (c) 1999-2007 Simon Tatham.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Halibut contains font metrics derived from the *Font Metrics for PDF Core 14 Fonts*, which carry the following copyright notice and licence:

Copyright (c) 1985, 1987, 1989, 1990, 1991, 1992, 1993, 1997 Adobe Systems Incorporated.  
All Rights Reserved.

This file and the 14 PostScript(R) AFM files it accompanies may be used, copied, and distributed for any purpose and without charge, with or without modification, provided that all copyright notices are retained; that the AFM files are not distributed without this file; that all modifications to this file or any of the AFM files are prominently noted in the modified file(s); and that this paragraph is not modified. Adobe Systems has no responsibility or obligation to support the use of the AFM files.

## Appendix B: Halibut man page

---

### B.1 NAME

`halibut` – multi-format documentation formatting tool

### B.2 SYNOPSIS

```
halibut [options] file1.but [file2.but ...]
```

### B.3 DESCRIPTION

`halibut` reads the given set of input files, assembles them into a document, and outputs that document in one or more formats.

The available command-line options can configure what formats Halibut should output in, and can also configure other things about the way Halibut works.

### B.4 OPTIONS

The command-line options supported by `halibut` are:

`--text[=filename]`

Makes Halibut generate an output file in plain text format. If the optional *filename* parameter is supplied, the output text file will be given that name. Otherwise, the name of the output text file will be as specified in the input files, or `output.txt` if none is specified at all.

`--html[=filename]`

Makes Halibut generate one or more output files in HTML format. If the optional *filename* parameter is supplied, there will be precisely one HTML output file with that name, containing the whole document. Otherwise, there may be one or more than one HTML file produced as output; this, and the file names, will be as specified in the input files, or given a set of default names starting with `Contents.html` if none is specified at all.

`--winhelp[=filename]`

Makes Halibut generate an output file in Windows Help format. If the optional *filename* parameter is supplied, the output help file will be given that name. Otherwise, the name of the output help file will be as specified in the input files, or `output.hlp` if none is specified at all.

The output help file must have a name ending in `.hlp`; if it does not, `.hlp` will be added. A secondary contents file will be created alongside the main help file, with the

same name except that it will end in `.cnt` (for example `output.cnt`, if the main file is `output.hlp`).

`--man[=filename]`

Makes Halibut generate an output file in Unix man page format. If the optional *filename* parameter is supplied, the output man page will be given that name. Otherwise, the name of the output man page will be as specified in the input files, or `output.1` if none is specified at all.

`--info[=filename]`

Makes Halibut generate an `info` file. If the optional *filename* parameter is supplied, the output `info` file will be given that name. Otherwise, the name of the output `info` file will be as specified in the input files, or `output.info` if none is specified at all.

By default, unless configured otherwise using the `\cfg{info-max-file-size}{0}` directive, the `info` output will be split into multiple files. The main file will have the name you specify; the subsidiary files will have suffixes `-1`, `-2` etc.

`--pdf[=filename]`

Makes Halibut generate an output file in PDF format. If the optional *filename* parameter is supplied, the PDF output file will be given that name. Otherwise, the name of the output PDF file will be as specified in the input files, or `output.pdf` if none is specified at all.

`--ps[=filename]`

Makes Halibut generate an output file in PostScript format. If the optional *filename* parameter is supplied, the PostScript output file will be given that name. Otherwise, the name of the output PostScript file will be as specified in the input files, or `output.ps` if none is specified at all.

`-Cword:word[:word...]`

Adds a configuration directive to the input processed by Halibut. Using this directive is exactly equivalent to appending an extra input file to the command line which contains the directive `\cfg{word}{word}{word...}`.

`--input-charset=charset`

Changes the assumed character set for input files from the default of ASCII.

`--list-charsets`

Makes Halibut list character sets known to it.

`--precise`

Makes Halibut report the column number as well as the line number when it encounters an error in an input file.

`--help`

Makes Halibut display a brief summary of its command-line options.

`--version`

Makes Halibut report its version number.

`--licence`

Makes Halibut display its licence (MIT).

## **B.5 MORE INFORMATION**

For more information on Halibut, including full details of the input file format, look in the full manual. If this is not installed locally on your system, you can also find it at the Halibut web site:

<http://www.chiark.greenend.org.uk/~sgtatham/halibut/>

## **B.6 BUGS**

This man page isn't terribly complete.



# Index

---

\A command	23	\cfg{html-author}	44
<ADDRESS>	41, 44	\cfg{html-body-end}	41
Adobe Font Metrics	57	\cfg{html-body-start}	40
AFM files	57	\cfg{html-body-tag}	40
alignment	34	\cfg{html-chapter-numeric}	41
<A NAME="...">	43	\cfg{html-chapter-suffix}	41
A4 paper	59	\cfg{html-charset}, lack of	42
appendices, renaming	24, 29	\cfg{html-contents-depth}	39
appendix	24	\cfg{html-contents-filename}	38
appendix configuration directive	30	\cfg{html-contents-text}	42
AppleTitle, <META> tag	40	\cfg{html-description}	44
ASCII	30	\cfg{html-head-end}	40
ASCII quote characters	14	\cfg{html-index-filename}	38
author, of document	44, 49	\cfg{html-index-main-separator}	42
%b	38	\cfg{html-index-multiple-separator}	42
background colour	40	\cfg{html-index-text}	42
backslash	11	\cfg{html-leaf-contains-contents}	40
\B command	26	\cfg{html-leaf-level}	39
\b command	19	\cfg{html-leaf-smallest-contents}	40
bibliography	16, 25	\cfg{html-local-head}	40
blank line	11	\cfg{html-mshtmlhelp-chm}	45
<BLOCKQUOTE>	22	\cfg{html-mshtmlhelp-contents}	45
blurb commands	24	\cfg{html-mshtmlhelp-index}	45
<BODY>	40, 41	\cfg{html-mshtmlhelp-project}	45
braces	11	\cfg{html-navigation-attributes}	41
\BR command	26	\cfg{html-nav-next-text}	42
bullet	36, 47, 50, 52, 58		
bulleted list, indentation	34, 52, 56		
bulleted lists	19		
\C command	23		
\c command	12, 17		
-C command-line option	9		
centre	34		
\cfg{appendix}	30		
\cfg{chapter}	29		
\cfg command	9, 29		
\cfg{contents}	30		
\cfg{html-address-end}	41		
\cfg{html-address-start}	41		



<code>\cfg{paper-page-height}</code>	55	<code>\cfg{text-section-suffix}</code>	35
<code>\cfg{paper-pagenum-font-size}</code>	58	<code>\cfg{text-section-underline}</code>	35
<code>\cfg{paper-page-width}</code>	55	<code>\cfg{text-title-align}</code>	34
<code>\cfg{paper-quote-indent}</code>	56	<code>\cfg{text-title-underline}</code>	35
<code>\cfg{paper-quotes}</code>	58	<code>\cfg{text-versionid}</code>	36
<code>\cfg{paper-right-margin}</code>	55	<code>\cfg{text-width}</code>	33
<code>\cfg{paper-rule-thickness}</code>	58	<code>\cfg{winhelp-bullet}</code>	47
<code>\cfg{paper-section-fonts}</code>	58	<code>\cfg{winhelp-contents-titlepage}</code>	48
<code>\cfg{paper-section-font-size}</code>	58	<code>\cfg{winhelp-filename}</code>	47
<code>\cfg{paper-sect-num-left-space}</code>	56	<code>\cfg{winhelp-list-suffix}</code>	48
<code>\cfg{paper-title-fonts}</code>	57	<code>\cfg{winhelp-quotes}</code>	48
<code>\cfg{paper-title-font-size}</code>	57	<code>\cfg{winhelp-section-suffix}</code>	48
<code>\cfg{paper-top-margin}</code>	55	<code>\cfg{winhelp-topic}</code>	48
<code>\cfg{pdf-filename}</code>	54	<code>\cfg{xhtml-anything}</code>	37
<code>\cfg{ps-filename}</code>	55	chapter configuration directive	29
<code>\cfg{quotes}</code>	30	chapter headings	23, 49
<code>\cfg{section}</code>	29	chapter headings, configuring display	29, 34, 41
<code>\cfg{text-bullet}</code>	36	chapter keywords, syntax of	23
<code>\cfg{text-chapter-align}</code>	35	chapter numbering	23
<code>\cfg{text-chapter-numeric}</code>	35	chapters, renaming	24, 29
<code>\cfg{text-chapter-suffix}</code>	35	character set	9, 16, 30, 35
<code>\cfg{text-chapter-underline}</code>	35	character sets, enumerating	10
<code>\cfg{text-charset}</code>	35	.chm files	44
<code>\cfg{text-emphasis}</code>	36	citation	26
<code>\cfg{text-filename}</code>	33	code	12, 17
<code>\cfg{text-indent}</code>	33	code paragraphs	17
<code>\cfg{text-indent-code}</code>	33	code paragraphs, indentation	33, 52
<code>\cfg{text-indent-preamble}</code>	34	<code>\#</code> command	17, 25
<code>\cfg{text-list-indent}</code>	34	<code>\-</code> command	14
<code>\cfg{text-listitem-indent}</code>	34	<code>\.</code> command	31
<code>\cfg{text-list-suffix}</code>	36	<code>\_</code> command	14
<code>\cfg{text-quotes}</code>	36	command line	7
<code>\cfg{text-rule}</code>	36	command-line options	7
<code>\cfg{text-section-align}</code>	35	commands, general syntax of	31
<code>\cfg{text-section-numeric}</code>	35	commands, paragraph-level	17
		comments	17, 25
		computer code	12, 17
		configuration directives	33
		configuring	29
		configuring heading display	29, 34, 41

contact details	41	formatting commands, paragraph-level	17
‘Contents’	41	general syntax of formatting commands	31
contents, depth	39	GNU info	51
contents file	39	gutter	57
Contents.html	39	\H command	23
context-sensitive help	48	<HEAD>	40, 44
continuing list items	21	headers	40, 49
\copyright command	25	heading keywords, syntax of	23
copyright statement	25	headings	23, 49
\cq command	13	headings, configuring display	29, 34, 41
cross-references	16	--help command-line option	10
CSS	40	Help compiler, lack of need for	7
\cw command	12	Help topic	48
date	14	Help, Windows	7, 47
\date command	14	--hlp command-line option	8
\dd command	20	horizontal dimensions	33
default settings	31, 37, 46, 48, 51, 54, 55	horizontal rules	22, 36, 50, 53
\define command	31	house style	40, 41
definition lists	20	HTML	7, 15, 37, 40
depth of contents	39	html-address-end configuration	41
description lists	20	directive	41
description, of document	44	html-address-start configuration	41
dir file	53	directive	44
display paragraph	17	html-author configuration directive	41
doing nothing	31	html-body-end configuration directive	41
\dt command	20	html-body-start configuration	40
\e command	12	directive	40
embedding fonts	57	html-body-tag configuration directive	40
emphasis	12	html-chapter-numeric	41
emphasis in code paragraphs	18	configuration directive	41
END-INFO-DIR-ENTRY	53	html-chapter-suffix configuration	41
escaping, special characters	11	directive	42
fallback text	16, 35	html-charset configuration directive,	42
FAQs, writing	24	lack of	8, 39
file name, output	33, 38, 47, 49, 54, 55	--html command-line option	8, 39
fine-tuning the index	27	html-contents-depth configuration	39
fixed-width font	12	directive	38
fonts	57	html-contents-filename	38
fonts, embedding	57	configuration directive	42
font size	57, 58	html-contents-text configuration	42
fonts, Type 1	57	directive	
footers	41, 49		
formatting command	38		
formatting commands, general syntax of	31		
formatting commands, inline	12		

html-description configuration directive	44	html-rellinks configuration directive	44
HTML entities	42	html-restrict-charset configuration directive	43
html-head-end configuration directive	40	html-section-numeric configuration directive	41
HTML Help	44	html-section-suffix configuration directive	41
html-index-filename configuration directive	38	html-single-filename configuration directive	38
html-index-main-separator configuration directive	42	html-suppress-address configuration directive	44
html-index-multiple-separator configuration directive	42	html-suppress-navlinks configuration directive	44
html-index-text configuration directive	42	html-template-filename configuration directive	38
html-leaf-contains-contents configuration directive	40	html-template-fragment configuration directive	43
html-leaf-level configuration directive	39	html-title-separator configuration directive	42
html-leaf-smallest-contents configuration directive	40	html-version configuration directive	43
html-local-head configuration directive	40	html-versionid configuration directive	44
html-mshtmlhelp-chm configuration directive	45	hyperlinks	15, 16
html-mshtmlhelp-contents configuration directive	45	hyphens, non-breaking	14
html-mshtmlhelp-index configuration directive	45	\i\c combination	27
html-mshtmlhelp-project configuration directive	45	\I command	27
html-navigation-attributes configuration directive	41	\i command	26
html-nav-next-text configuration directive	42	\i\cw combination	27
html-nav-prev-text configuration directive	42	\i\c combination	27
html-nav-separator configuration directive	42	\ii command	27
html-output-charset configuration directive	42	\IM command	28
html-post-versionid configuration directive	42	indentation	33
html-preamble-text configuration directive	42	indenting multiple paragraphs	22
html-pre-versionid configuration directive	42	index	26
html-quotes configuration directive	43	index terms	52
		info	51
		info-bullet configuration directive	52
		info-charset configuration directive	52
		--info command-line option	8, 51
		INFO-DIR-ENTRY	53
		info-dir-entry configuration directive	53
		info-emphasis configuration directive	53

info-filename configuration directive	51	list, indentation	34, 56
info-indent-code configuration directive	52	list items, continuing	21
info-index-width configuration directive	52	lists	18
info-list-indent configuration directive	52	lists, bulleted	19
info-listitem-indent configuration directive	52	lists, description	20
info-list-suffix configuration directive	53	lists, nested	21
info-max-file-size configuration directive	51	lists, numbered	16, 19, 36, 48, 53
info-quotes configuration directive	53	MacOS on-line help	40
info-rule configuration directive	53	macros	31
info-section-suffix configuration directive	52	man-bullet configuration directive	50
info-underline configuration directive	52	man-charset configuration directive	50
info-width configuration directive	52	--man command-line option	8, 49
inline formatting commands	12	man-filename configuration directive	49
--input-charset command-line option	9	man-headnumbers configuration directive	49
input-charset configuration directive	30	man-identity configuration directive	49
invisible index tag	27	man-mindepth configuration directive	49
ISO 8601	15	man page	7, 49
%k	38	man-quotes configuration directive	51
\K command	16	man-rule configuration directive	50
\k command	16, 26	Manual.html	39
keywords	16	margin	33
keywords, syntax of	23	maximum file size	51
\lcont command	21	measurements	55
leaders	56	merging index terms	28
leaf file	39	<META NAME="AppleTitle">	40
left	34	<META> tags	44
leftplus	34	Microsoft HTML Help	44
length of lines	33, 52	MS HTML Help	44
--licence command-line option	10	%N	38
line breaks	11, 12	%n	38
line length	33, 52	navigation links	41, 44
linking to web sites	15	\n command	19
<LINK> tags	44	nested lists	21
--list-charsets command-line option	10	'Next'	41
		\nocite command	26
		non-breaking hyphens	14
		non-breaking spaces	14
		NOP	31
		nroff	49
		numbered list, indentation	34, 52, 56

numbered lists	16, 19, 36, 48, 53	paper-index-minsep configuration directive	57
numbering, of sections	23	paper-leader-separation configuration directive	56
options, command-line	7	paper-left-margin configuration directive	55
output.cnt	7	paper-list-indent configuration directive	56
output file name	33, 38, 47, 49, 54, 55	paper-listitem-indent configuration directive	56
output files	7	paper-page-height configuration directive	55
output formats	7, 33	paper-pagenum-font-size configuration directive	58
output.hlp	7	paper-page-width configuration directive	55
output.txt	7	paper-quote-indent configuration directive	56
page footers	41, 49	paper-quotes configuration directive	58
page headers	40, 49	paper-right-margin configuration directive	55
page numbers	58	paper-rule-thickness configuration directive	58
paper-base-fonts configuration directive	58	paper-section-fonts configuration directive	58
paper-base-font-size configuration directive	58	paper-section-font-size configuration directive	58
paper-base-leading configuration directive	55	paper-sect-num-left-space configuration directive	56
paper-base-para-spacing configuration directive	55	paper-title-fonts configuration directive	57
paper-bottom-margin configuration directive	55	paper-title-font-size configuration directive	57
paper-bullet configuration directive	58	paper-top-margin configuration directive	55
paper-chapter-fonts configuration directive	58	paragraph-level formatting commands	17
paper-chapter-font-size configuration directive	58	paragraphs of ordinary text	11
paper-chapter-top-space configuration directive	56	paragraphs, wrapping	11, 14
paper-chapter-underline-depth configuration directive	56	PDF	54
paper-chapter-underline-thickness configuration directive	56	--pdf command-line option	8, 54
paper-code-fonts configuration directive	58	pdf-filename configuration directive	54
paper-code-font-size configuration directive	58	pdfmark	55
paper-contents-index-step configuration directive	56	PFA files	57
paper-contents-margin configuration directive	56	PFB files	57
paper-footer-distance configuration directive	56	plain text	7, 33
paper-index-columns configuration directive	56		
paper-index-gutter configuration directive	57		

points	55, 57	text-chapter-align configuration directive	35
Portable Document Format	54	text-chapter-numeric configuration directive	35
PostScript	55	text-chapter-suffix configuration directive	35
preamble	34	text-chapter-underline configuration directive	35
--precise command-line option	10	text-charset configuration directive	35
'Previous'	41	--text command-line option	7, 33
--ps command-line option	9, 55	text-emphasis configuration directive	36
ps-filename configuration directive	55	text-filename configuration directive	33
\q command	13	text-indent-code configuration directive	33
quotation	22	text-indent configuration directive	33
quotation marks	13	text-indent-preamble configuration directive	34
\quote command	22	text-list-indent configuration directive	34
quotes configuration directive	30	text-listitem-indent configuration directive	34
renaming info files	51	text-list-suffix configuration directive	36
renaming sections	24, 29	text, plain	7, 33
replicating index terms	27	text-quotes configuration directive	36
rewriting index terms	28	text-rule configuration directive	36
\rule command	22, 36, 50	text-section-align configuration directive	35
rules, horizontal	22, 36, 50, 53	text-section-numeric configuration directive	35
running Halibut	7	text-section-suffix configuration directive	35
\S command	23	text-section-underline configuration directive	35
section configuration directive	29	text-title-align configuration directive	34
section headings	23, 49	text-title-underline configuration directive	35
section headings, configuring display	29, 34, 41	text-versionid configuration directive	36
section keywords	16	text width	33, 52
section keywords, syntax of	23	text-width configuration directive	33
section numbering	23	.TH directive	49
section numbers	16, 49		
sections, renaming	24, 29		
spaces, non-breaking	14		
special characters	11		
special paragraph types	24		
\S2, \S3 commands etc.	24		
START-INFO-DIR-ENTRY	53		
strftime	15		
stylesheet	40		
sub-file	39		
suffix text, in section titles	35, 48		
syntax highlighting	18		
syntax of general formatting commands	31		
syntax of keywords	23		
template	38, 43		
text-bullet configuration directive	36		



<code>\title</code> command	25
Type 1 fonts	57
<code>\U</code> command	23
<code>\u</code> command	16
underlining	34
Unicode	16, 30
Unicode matched quotes	14
unnumbered chapter	24
URL	15
verbatim	12
verbatim HTML	40
<code>--version</code> command-line option	10
<code>\versionid</code> command	25, 36, 44
version ID paragraph	42
version IDs	25, 36, 44
<code>\W</code> command	15
weak code	12
web sites	15
<code>--whlp</code> command-line option	8
width, of text	33, 52
Win1252	47
Windows Help	7, 47
Windows HTML Help	44
<code>winhelp-bullet</code> configuration directive	47
<code>--winhelp</code> command-line option	8, 47
<code>winhelp-contents-titlepage</code> configuration directive	48
<code>winhelp-filename</code> configuration directive	47
<code>winhelp-list-suffix</code> configuration directive	48
<code>winhelp-quotes</code> configuration directive	48
<code>winhelp-section-suffix</code> configuration directive	48
<code>winhelp-topic</code> configuration directive	48
wrapping paragraphs	11, 14
writing FAQs	24
WWW hyperlinks	15
<code>xhtml-anything</code> configuration directives	37
<code>--xhtml</code> command-line option	8