

The Markup Document Preparation System
(version 1.0.7a)
2 Oct 2012

Denis M. Wilson
Aberdeen
Scotland

ABSTRACT

The `-markup` system is a package of troff document formatting commands intended to be more comprehensive than the `-ms` macros. In addition it attempts to be more structured (using some ideas from LaTeX), and also to do some fancy things not available with most *troff* macro packages. It is intended to make design flexible, and will ultimately support a number of document styles, and make it relatively simple to write new ones. It works only with *groff* from version 1.20.

Table of Contents

Chapter 1. Preliminaries	3	7.2. The Verbatim Package	21
1.1. Introduction	3	7.3. The Boxes Package	22
Chapter 2. Document Structure	4	7.4. The Dotted line package	22
2.1. Overall structure	4	7.5. Insert package	23
2.2. The “base” Packages	5	7.6. The Drop Package	24
Chapter 3. Document Sizes	5	7.7. The Colour Package	24
Chapter 4. Document Units	7	Chapter 8. Moving material	25
4.1. Page Control Package	7	8.1. Cross References	25
4.2. Section Headings	8	8.2. Table of Contents	26
4.3. Paragraphs	9	8.3. Keeps and Floats	26
4.4. Font Control Package	10	8.4. The Index package	27
4.5. The Lucida Fonts Package	11	Chapter 9. Classes	28
4.6. The Predefined Strings Package.	13	9.1. Classes	28
Chapter 5. Presentations and Displays.	13	9.2. The Article Class	28
5.1. Two column package	13	9.3. The Letter Class	29
5.2. Support for tables	13	9.4. The Booklet class	30
5.3. Support for equations	14	Chapter 10. Customised packages	31
5.4. The Display Package	15	10.1. Address labels	31
5.5. The List package	16	Chapter 11. Internals	32
5.6. The Wide paper package	18	11.1. The Error Package	32
Chapter 6. Utilities	18	11.2. Debugging	33
6.1. The Space Package	18	11.3. The Stack Package	33
6.2. Date Package	19	11.4. The Environment Package	33
6.3. String Manipulation	19	11.5. The Diversion Package	34
Chapter 7. Special effects	20	Acknowledgements	35
7.1. Included and Other Pictures	20	Index	36

Chapter 1

Preliminaries

1.1 Introduction

The `-markup` system is designed to make structured document markup using *groff* simple and flexible. It offers the same kind of facilities as (for example) the `-ms` package, but with several differences

- The system consists of a number of packages. It was originally intended to make these independent of each other, but with only a few exceptions this could not be done, and they work hand-in-hand.
- Some packages offer “fancy” facilities not usually available with *troff* macros, such as side insertions, balanced two-column mode, and others.
- The source is fully commented, and places where style parameters can be changed are marked. Each package contains the user documentation in a form which can be extracted and formatted as in this document. At some time in the future, maintenance documentation may also be included, and also documentation explaining how to tune packages for individual requirements.

The package has been used in reality for several major projects.

The *raison d'être* of the package was the trouble the author found in maintaining many different macro packages; the chief problems being different user interfaces, and propagating improvements in one package into the others. This document is typeset using the system it describes.

There is no support for *nroff*(1), and the system is designed with POSTSCRIPT output assumed.

The `-U` option of *groff* is required as the `.sy` and `.open` requests are used in an unavoidable way.

Directories

Several strings are defined to assist locating files in the system:

`MARKUP-DIR` where the installed macro packages and classes are held;
`EPS-DIR` where EPS files are held;
`IST-DIR` where index style files are held.

1.1.1 Conventions

There are some points which should be noted when creating text for the *markup* system:

- Blank lines may be used freely to improve readability of the source, as they are ignored except where otherwise stated.
- There are as a matter of principle *no* default units for length values (vertical and horizontal sizes, distances and position). Units *must* be used. It is assumed the user has enough familiarity with basic *troff*(1) to know what these are.

1.1.2 Commands

The following is the command to load a package. If a package has been loaded already it is not loaded a second time. The required optional packages should be requested before the `.BeginDocument` command. Base packages (see below) are loaded on startup.

```
.UsePackage package [ package ... ]
```

This loads each argument package in the order given. Those customised to change something in another package must be loaded afterwards. A package is loaded once. There may be as many `.UsePackage` commands as required.

Some packages are always loaded. These are known as *base* packages.

Chapter 2 Document Structure

2.1 Overall structure

The document should start with

```
.DocumentClass <class>
```

where `<class>` is one of a number of document styles, such as *letter*, *book*, *article*, etc. It should come first; in any case no output text is permitted before it.

The user may then incorporate a number of optional packages as required, and adjust any documented parameters if needed. There should be no printed text in this section, which is called the *preamble*.

Then should follow

```
.BeginDocument
The text of the document
.EndDocument
```

Any text after the `.EndDocument` command is ignored with a warning.

If the above structure is not adhered to, error messages are generated, or the document will be malformed.

`.DocumentClass class [options]`

This invokes the appropriate class package, which determines the overall style of the document, and also reads in any additional packages specified as other arguments.

The `options` argument is reserved for future extensions.

In addition, the base name of the *file containing this command* is used as the root of auxiliary files with appropriate extensions where required. Thus for example if the input file is named `shares.mkp` then the table of contents file would be `shares.toc`. The other auxiliary files currently used are the cross-reference file, with extension `.ref`, and the index files, with extensions `.idx`, `.ind`, `.ilg`.

`.BeginDocument`

This command must come before any output text. All that may precede it are definitions, including changing any of the standard settings. If any printed text occurs here a warning is issued. No settings, such as paper size, point size etc., are actually made until this command is executed.

`.EndDocument`

This command must come at the end of the document. Such actions as performing checks that all structures started are also finished, outputting left-over floats, index pages if required and so on are done here. Any text after this will be ignored, with a warning.

2.2 The “base” Packages

Each package is in a file of its own, in a subdirectory `MARKUP` of the directory where *groff* macros are usually found. Each file has an extension `.pkg`. The packages in Table 1 on the next page are automatically included when `-markup` is used (other packages are not). For the benefit of those who wish to write new packages, each package’s internal names begin either with the package name, or a related name shown in the table; users should avoid redefining these names.

Chapter 3 Document Sizes

This package controls the size and spacing of many global items in the document. Its principal use is to set paper and page sizes, but it also sets some values which are related to these. Most of it is for internal use, but there are a few user-level commands.

Package	Brief description	Reserved prefix
Document structure		
Doc	Overall document structure	doc-
Boxes	Draw boxes round text, etc.	box-
Colour	Add colour.	colr-
Contents	Generate table of contents	cont-
Crossref	Cross referencing	cref-
Date	Format dates	date-
Display	Displayed material	disp-
Div	Diversion management	div-
Dotted	Fill line with dots	dot-
Drop	Dropped capitals package	drop-
Eqn	Support for equations	eqn-
Env	Environment definitions	env-
Error	Error and warning messages	error-
Fonts	Font and font-style control	font-
Headings	Headings and subheadings for sections	head-
Index	Generate index entries	idx-
Insert	Put rectangular insertions into text	insert-
Keep	Floating and other keeps	keep-
Lists	Enumerated and itemised lists	list-
Page	Page headers and footers	page-
Para	Paragraphs and indented sections	para-
Pics	Include pictorial material	pic-
Predef	Generally useful strings	pre-
Sizes	Sets sizes for paper and point size	size-
Space	Controls spacing (internal use)	space-
Stack	Controls nesting (internal use)	stack-
Strings	String manipulation	str-
Tbl	Support for tables	tbl-
Twocol	Invoke two-column mode	mcol-
Verb	Typeset verbatim text	verb-

Table 1

.A4Paper

Sets the size of the sheet to be used as A4. This is the default.

.A5Paper

Sets the sheet size to be A5.

Other paper sizes are not yet defined.

.SetPointSize size

Sets the overall document text point size to be *size*, which must be one of 10, 11 or 12. The default is 12-point, as this package is used principally with POSTSCRIPT output, whose font design size seems to be universally 12-point.

The paper and point sizes combine to set suitable parameters. Currently only the combinations (A4, 10- and 12-point), (A5, 10- and 12-point) are defined. They were defined with the appearance of family URWPa11adi0L (similar to Pa1atino) in mind.

Chapter 4

Document Units

4.1 Page Control Package

This package supplies commands for page headers and footers. The *header* is the stuff inserted automatically at the top of each page, additional to the user's text. Similarly a *footer* is inserted at the bottom.

4.1.1 Header styles

The header style can be blank, empty (i.e. no space) or have one line of text. The header style is defined by a string variable `page-opt` which is set to the standard value after each header has been completed. The allowable values are:

`blank` The header is the usual width, but with nothing printed;

`empty` It is of zero width (starting at the top paper margin);

`ord` The header is normal. The normal header consists of one line of text, in a three-part title style, the parts of which are determined by string variables. These can be set by the user, or the whole line can be done differently, as it is done by a command.

Anything other than normal must be set for each page.

4.1.2 Commands

`.SetPageHeader name`

`.SetPageFooter name`

These set the macros to print the header and footer contents; the `name` argument is the name of the command to be installed.

`.SetFooterOpt style [dist]`

This controls the style and width of the page footer on the current page. The style is reset to `ord` at the start of each page. The available styles are:

`ord` The footer style is set to ordinary, with the footer title printed. In this case the argument `dist` is added to the vertical position where the footer is sprung (it may be negative). The default values are `ord 0`.

`blank` The footer is blank, with `dist` as the width: it must be positive — the minimum possible is 1 (one unit of device resolution), 1V.

`.HideFooter`

This removes the page footer; nothing will be printed at the bottom of the page. Useful in a number of circumstances — e.g. when a page has displayed material or has to be left completely blank.

`.RevealFooter`

The footer is set to normal, although in future it may be set to the footer style prevailing at the last `.HideFooter`.

The use of these commands is deprecated, and if there is a better way it will become documented.

.NewPage [args]

Starts a new page. It should be used only at the end of a completed section. By default it causes a break. The arguments come in any order, and may be

nobreak Do not cause a break.

number Number pages from this one starting at *number*. *number* may be relative, and must be decimal.

.DoublePageSpread

If this is placed on a page, the next page will space after the heading to the corresponding position. Usually used on an even-numbered page after a major heading to even the top of text on opposing pages.

.RunningHeader name

Defines *name* to be the name of a string to be part of the running header; it is placed in the left or right part of the three-part title. The string is defined using the *.ds* mechanism of *troff*. Since it is up to a class to make use of this, this is probably not the appropriate place to have this command. By default it is not used.

.BeginFullPage

.EndFullpage

These delimit a segment of text starting on a new page (if the current one is not empty), with no headers or footers, and with the full paper width available. A new environment is used. It is normally used for specially set or displayed material, such as including a whole-page picture. *.EndFullPage* restores the environment, but does not start a new page automatically, nor is the footer restored until the next page.

.OneSided

If a call of this is made then left and right pages are treated the same.

4.2 Section Headings

This package supplies numbered, unnumbered and centred headings for sections of a document. The naming convention is that **Section** in a name is a numbered section, and is never centred or right-adjusted; **Heading** in a name means an unnumbered heading, which will be centred if the name additionally ends in **C**, or right adjusted if the name ends in **R**. Any of these may be preceded by a number of prefixes **Sub** to denote subsections or subheadings. The current defined is 2, giving three levels.

The other convention is that each argument is on a separate line; thus arguments which contain spaces must be enclosed in quotes. For example:

**This is a two-line
right-adjusted subsubheading**

is generated by

```
.SubSubHeadingR "This is a two-line" "right-adjusted subsubheading"
```

4.2.1 Chapters

This package has a rudimentary chapter heading command `.Chapter`. This automatically increases the number of section levels, by adding an extra one at the start. Numbered sections are displayed with the chapter number prepended. The first `.Chapter` must precede any of the `.[Sub]Section` commands.

4.2.2 Section ranges

If the string register `head-ranges` is defined, then the default right page footer contains the range of the numbered (sub)sections appearing on that page.

4.2.3 Placing headings in the contents

Sections and headings can be entered into a table of contents automatically by using the number register `head-contents`, as in

```
.nr head-contents 0-2
```

The first argument of a section heading is placed in the table of contents, provided its level is at least that of the absolute value of this number register. If its value is positive, unnumbered headings are placed there too. The top level is numbered 1. The default value of this register is zero. See the `Contents` package on page 26.

4.2.4 Appendices

If the command `.Appendix` is used at the top level (at `.Section` or `.Chapter` level), then the top-level numbering is started at A then B and so on. The title style is the same as for the ordinary top-level (if chapters are used, the word `Chapter` is replaced by `Appendix`).

4.2.5 Redefine headings package

If this small optional package (`HeadRedef.pkg`) is included, all the sectioning commands are redefined so that if the first argument is `-c` then the first line of the heading is placed in the table of contents.

4.3 Paragraphs

4.3.1 Paragraphing Commands

This package defines commands for paragraphing and simple displays, such as paragraphs with hanging tags, and indented sections. More sophisticated requirements are met by the `Display` and `List` packages.

```
.Para [ left ]
```

Starts a new paragraph. If the optional argument `left` is given, then it will be a non-indented paragraph. If the optional argument is a number then the indentation will be that amount.

```
.IP [ tag [ width ] ]
```

```
.IPx [ tag [ width ] ]
```

This starts an indented paragraph. If one argument is given, then it is used as a hanging tag for the paragraph. If there is no room, the paragraph will start on the next line. If a second argument is given, this is the width of the indent, and will stay in force until a

sequence of `.IP[x]` commands is terminated by an ordinary `.Para` command, or one of the sectioning or heading commands is given. The default is restored at the end of the sequence (e.g. at `.Para`). `.IPx` is identical to `.IP` but omits the space before the paragraph.

`.TP [width]`

`.TPx [width]`

Have the same effect as `.IP` and `.IPx` respectively, but the tag is the following line and the optional argument is the indent width.

`.RS [lmargin [rmargin]]`

Starts a relatively indented section, with left and right indents which are given by the `lmargin` and `rmargin` arguments respectively. There are defaults.

`.RE`

Ends a relatively indented section. Relatively indented sections may be nested.

4.3.2 Default parameters

These are all *number* registers.

`ParaDefaultTagWidth`

the default indent for `.IP` and `.TP`.

`ParaIndent`

the indentation of indented (ordinary) paragraphs.

`ParaRelIndent`

the default left indent of relatively indented sections. The default right indent is 0.

`ParaSep`

the spacing between paragraphs (of all kinds).

4.4 Font Control Package

All the font-style setting commands take 0, 1, 2 or 3 arguments. The effect of the differing number of arguments is:

- 0 The style is set globally (and nested). It is cancelled by the `.P` command.
- 1 The argument is set in the style.
- 2 The first is set in the style and the second is appended in the prevailing style, with no separating space.
- 3 The second is in the specified style, with the first and third prefixed and suffixed in the prevailing style.

Within each part, fonts may be changed, provided each such change is done as `\f[font]text\f[]` where `font` is the font or style and `text` is the text to be set in the new font. Note that such changes may *not* be nested.

`.R [a [b [c]]]`

Sets style *roman*, i.e. normal text face.

`.I [a [b [c]]]`

Sets style *italic*.

`.B [a [b [c]]]`

Sets style *bold*.

`.BI [a [b [c]]]`

Sets style *bold italic*.

`.CW [a [b [c]]]`

Sets text in a fixed-width font. The font currently used is *LucidaSans-Typewriter*, if it can be found, otherwise *Courier*. With this choice, the text is set slightly smaller, as the x-height of this face is quite large.

`.CAL [a [b [c]]]`

Sets text in a calligraphic font. The default face is *ZapfChanceryMediumItalic*, a fancy font, too commonly used but universally available. If the change is in-line, the text is set larger, as this font is quite a small one; the use is usually purely for effect. This command is also called `.ZC` for backwards compatibility (this alias may disappear).

`.P` This command returns to the previous style or font. Do not use the command `R` for this.

`.SC [a [b [c]]]`

Sets in small caps. This is often done by reducing the size slightly, but here it is done by reducing the height of the small-caps letters — a “fake small caps” style. For example the word `SMALLCAPS` is obtained by typing `.SC SmallCaps`. If expert fonts with real small caps are available, this can be redefined to use them.

Since `SC` is not a font-face, the style cannot be set globally.

`.MakeSC name text`

This defines a string named `name` to be the small-caps version of the `text`. This is more efficient for frequently occurring words in small caps, such as acronyms. The size used is that at the time of use of the defined string, not at the time of definition.

`.SwitchFamily family`

This changes to a new typeface family, saving the previous in a fully nestable fashion.

`.RestoreFamily`

This sets the family to that in force before the matching `.SwitchFamily`.

These two commands will probably move to another package.

`.TheFamily`

This command interpolates the name of the font family used for the whole document. Currently it works only for `POSTSCRIPT`, or for the devices `ascii` or `latin1`, where it gives the device name. *WARNING*: this may be replaced by a better method. For example, the installation may prefer to have font names *Times*, *Garamond*, and so on, as long font names are allowed

4.4.1 Settable parameters

The fixed-width typeface for the `.CW` command can be over-ridden by using `.ds font-tt face` and `.ds font-tt-size num` where `face` is the (usually) Roman face of the appropriate font family, and `num` is the percentage size of the font (or empty for its normal size). The calligraphic font can be changed by defining `.ds font-cal face` where `face` is a suitable font (calligraphic fonts usually come in one style).

4.5 The Lucida Fonts Package

The Lucida collection contains a wide variety of faces which work well together, as they are based on a common style. It utilises a family called “Lucida”. The collection contains some expert fonts. Although most of the faces are a commercial product, it was for the author an

exercise in extending the range of styles that this collection provides, beyond the traditional R, I, B, BI styles of *troff*.

Several of the commands in the `Fonts` package are redefined in order to make use of the variety of faces. True small caps are available.

4.5.1 The Defined Styles

Style	Description	Font
Ordinary styles		
R	Roman	LucidaBright
I	Italic	LucidaBright-Italic
B	Bold	LucidaBright-Demi
BI	Bold italic	LucidaBright-DemiItalic
Extra styles		
O	Oblique (a slanted variation of Roman)	LucidaBright-Oblique
SC	Small caps (includes old-style digits)	LucidaBrightSmallcaps
SCB	Small caps bold (includes old-style digits)	LucidaBrightSmallcaps-Demi
Sans Serif group		
SR	Roman Sans	LucidaSans
SI	Italic Sans	LucidaSans-Italic
SB	Bold Sans	LucidaSans-Demi
SEB	Extra Bold Sans	LucidaSans-Bold
SBI	Bold italic Sans	LucidaSans-DemiItalic
SEBI	Extra bold italic Sans	LucidaSans-BoldItalic
Fixed Width, serif		
TSR	Fixed width Roman, serif	LucidaSans-Typewriter
TSI	Fixed width italic, serif	LucidaSans-TypewriterOblique
TSB	Fixed width Bold, serif	LucidaSans-TypewriterBold
TSBI	Fixed width bold italic, serif	LucidaSans-TypewriterBoldOblique
Fixed Width, Sans-serif		
TR	fixed width roman, sans	LucidaTypewriter
TI	Fixed width italic, sans	LucidaTypewriterOblique
TB	fixed width bold, sans	LucidaTypewriterBold
TBI	fixed width bold italic, sans	LucidaTypewriterBoldOblique
Special fonts for effect		
CAL	Calligraphic	LucidaCalligraphy-Italic
HAND	Handwriting (italic)	LucidaHandwriting-Italic
BK	Black letter	LucidaBlackletter
Informal, casual text		
CASR	Casual roman (an informal face)	LucidaCasual
CASI	Casual italic	LucidaCasual-Italic

For each style X there is a corresponding command `.X` which sets the style as the `.R` command.

4.6 The Predefined Strings Package

This package simply defines some strings for user convenience. They are shown in the accompanying table.

String	Printed
bullet	•
dots	...
pound	£
pounds	£
TeX	T _E X
postscript	POSTSCRIPT

Chapter 5 Presentations and Displays

5.1 Two column package

This package enables two-column mode to be used. The two column mode may be started and finished anywhere; the end of two-column mode produces balanced columns (unlike most other two-column mode packages in *troff*).

This package uses some global values from the `Sizes` package.

5.1.1 Commands

`.TwoColumn`

Finish off single column mode and start two-column mode.

`.OneColumn`

End two-column mode and start one-column mode. If the amount of two column material does not fill the rest of the page, it will be split into two roughly equal columns. The result is not perfect, but consideration is being given to ways of improving it. The column balancing does work if there are no additional interline spaces in the two-column text.

5.2 Support for tables

The *tbl* preprocessor may be used to format tables in a manner described in the *tbl* documentation. Tables are delimited by the commands `.TS` and `.TE`.

Boxed tables are a problem if there is a possibility that they may be split across a page. Some solutions are these:

- If the table is small enough to fit on a page, then the `Keep` package may be used.

- If the table is too large, then this package offers support for multi-paged tables, which may also be boxed. It is good practice to arrange for a multi-paged table to have a running header. This package plagiarises the `-ms` package in its method of doing this, by directing the first 1 or more formatted table entries to be used as the header at the start of the table on each page. The user should arrange for the header to be formatted as if it were part of the table, simply by writing it as if it were the initial part of the table. After the lines constituting the header put the command `.TH`. The part of the table preceding this will be saved and used for each page. It is also necessary to indicate to the package that this is to be done; the `.TS` command must have the single argument `H`.

Warning: if a table with a header is boxed, then the use of the `nokeep` option with formatting options will cause the box to be misplaced when there is extra space inserted before or after the end of the header. This is an artefact of `gtbl`.

5.2.1 Commands

`.TS [H]`

Start the table. The `H` argument indicates that the part of the table up to `.TH` is used as a header for multi-paged tables.

`.TH`

This command indicates that the previously formatted part of the table is to be used as a header for the table on each page it spreads across. The `H` option to `.TS` must have been given.

`.TE`

End the table.

`.T&`

Change the table format mid-stream. This is done entirely by the `tbl` preprocessor, and nothing extra is done by this package. It is here merely to avoid spurious warnings.

`.TableStyle pre post`

Specify the distance before and after the table (default both zero).

5.3 Support for equations

The `eqn` preprocessor may be used to format equations either inline or displayed. This package adds some features to displayed equations, which lie between the commands `.EQ` and `.EN`. See the documentation of `eqn(1)` for details of the `eqn` language.

5.3.1 Commands

`.EQ`

Start a displayed equation. Any arguments are ignored.

`.EN`

Finish a displayed equation, and do layout according to the specified style. A warning is given if the number (if any) does not fit the space. *Caveat:* Some styles may fail for subsequent equations in a *mark-lineup* sequence, as `eqn(1)` apparently makes equations using *lineup* the same width as the one using *mark*.

`.EquationStyle args ...`

This command sets the style of equation display. Its arguments have the form

-key *value*

The arguments can be in any order. Any key that is omitted leaves its previous value unchanged. The keys are as follows:

Key	Type	Description	Default
-pre	<i>number</i>	Space before equation	0
-post	<i>number</i>	Space after equation	0
-lab	<i>string</i>	The macro to generate a number or label	<i>null</i>
-side	<i>string</i>	The side on which equation is numbered (left, right)	<i>right</i>
-align	<i>string</i>	The position of the equation (left, indent, right, centre)	<i>left</i>
-indent	<i>number</i>	The amount of indenting (with -align indent)	0

The lab macro must assign the equation number or label to the string register eqn-num.

5.4 The Display Package

This package supports the printing of displayed material. Displays may be nested.

5.4.1 Commands

.BeginDisplay name [caption]

.EndDisplay name

These commands delimit a segment of text which is to be “displayed”. The style of display is given by the argument name, and it will be followed by the caption caption (if given).

Blank lines are not ignored in displays.

A few display styles are predefined, but it is easy to define new ones.

.DefinedDisplay name params ...

This defines a new display called name which can be used as the target of .{Begin|End}Display. There are 11 parameters:

- 1 The space before the display
- 2 The space after the display
- 3 Left indent
- 4 Right indent
- 5 If filling to be on (1) or off (0)
- 6 Font or style (string)
- 7 Make vertical blank space this amount
- 8 The tab stop (if used) — all the same
- 9 If text to be centred (0), right- (-1) or left-adjusted (1)
- 10 The pointsize (may be relative)
- 11 The vertical spacing (may be relative)

Note that if tabs, ps and vs are the empty string they will not be changed.

5.4.2 Predefined displays

These three are currently defined; with usage, more will likely follow.

Text The text is printed in a fixed-width font, is unfilled, and slightly indented on the left.

There is a little preceding and following space. In addition, vertical blank space is printed (unlike the rest of this system), although one or more consecutive blank lines is displayed as a half-line.

This is useful for preformatted ASCII or LATIN1 text.

Program This is similar to **Text**, but also sets tabs at intervals for displaying programs indented with tabs.

Emph The text is printed in an emphasised font, has a little space on all four sides, and is filled. Useful for, say, a quotation from an important document or speech.

5.5 The List package

This is useful for generating numbered and tagged lists, which may be nested. Each list is indented relative to the containing list. A list has several properties: the indent, the width of the label or tag, the distance separating the tag from the indented text, and several vertical space separating distances.

There are two kinds of lists: enumerated, where the tags are an automatically incremented number sequence; and itemised, where the tags are a default string or user-specified.

.BeginItemise [args]

.EndItemise

These delimit a list whose tags are fixed strings, normally a bullet. This type of list may be nested to depth 3, the default tags varying with the depth.

For the meaning of **args** see **.BeginList**.

.BeginEnumerate [args]

.EndEnumerate

These commands delimit a section of text with automatically numbered items. If one of these environments is nested inside another, a new set of numbers is started. The style of number printing changes: the outermost is roman numbering, then lower case letters, and so on.

For the meaning of **args** see **.BeginList**.

.Item [-] [string]

This starts off a new item in the list, and generates the tag.

In an itemised list, the **-** must be omitted, and the optional **string** replaces the default tag.

In an enumerated list, the **string** is appended to the number, and If the first argument is a single ascii hyphen (**-**) then the number is not advanced. If you need **-** as an appendage in an enumerated list, use **\&-**.

It is safe to use font changes in the tag of **Item**, so long as they are not nested. The recommended method is using **\f[I] . . . \f[]**, for example.

.Itemx [-] [string]

This is the same as **.Item**, but the inter-item spacing is suppressed.

.BeginList name

.EndList name

These delimit a list environment called **name**. It is an error if the **name** on an **.EndList** command does not match a **.BeginList** in a nested fashion. How to define such a list is documented below. Meanwhile note that the other list styles are applications of this one.

5.5.1 Arguments

The `args` option is a list of options of the form `-opt [val]`. Each option changes one of the default values for the list. They may be in any order.

Option	Number of arguments	Type	Meaning
<code>-ts</code>	1	number	Space before list
<code>-is</code>	1	number	Space between items
<code>-w</code>	1	number	Width of space for inserting tag
<code>-sep</code>	1	number	Separation between tag and text
<code>-e</code>	0	-	Make the list an enumerated list
<code>-tag</code>	1	string	The default tag (number format for enum list)
<code>-pre</code>	1	string	String inserted before tag
<code>-post</code>	1	string	String appended to tag
<code>-bs</code>	1	number	Space after list
<code>-B</code>	0	-	Break to a new line if tag is too wide
<code>-N</code>	0	-	The tag may be taken from the next line.

The `-N` option causes the tag to be taken from the next line only for an itemised list, and the argument to `.Item` is empty.

5.5.2 Defining a new list type

To define a list whose name is for example `ZIP`, define a string `list-params-ZIP` whose value is the space-separated concatenation of the values given for the first 9 arguments in the above table — as if they were the arguments to a user-defined command. For the `-e` item, use 0 for an itemised list, 1 for an enumerated list. The commands `.{Begin|End}List ZIP` are then available. The `-N` and `-B` facilities still have to be given as arguments to `.BeginList`.

5.5.3 Special lists

There are several extra examples of the list style:

Null This is an itemised list in which all the properties are zero or empty. In principle all the other lists could be defined in terms of this, although currently the method described above is used.

Bold The tags are in a bold font.

FixedWidth The tags are in a constant-width font.

5.5.4 Independent numbering

`.SetCounter name [num]`

This command is provided for cases where the `.BeginEnumerate` command is not appropriate; e.g. for entries in a table. It sets two strings `name` and `name-` which on subsequent uses will display the numbers 1, 2, `name-` displays only the most recent number. If the `num` argument is given, the numbering will start at that. *Note:* for use in a table it is necessary to use it twice: before the start of the table, and at the beginning of the table data.

`.SkipCounter name num`

This advances the value of a counter defined by `.SetCounter` by the amount `num`, without generating any output.

5.6 The Wide paper package

This package simply redefines the text and margins to increase the widths of text and margins to make more use of the paper area. It is designed for those who either do not like to destroy trees or who need to see as much as possible on one sheet.

Chapter 6 Utilities

6.1 The Space Package

The `.sp` command in *troff* has two uses:

- (1) To leave an amount of vertical space (or remove it); and
- (2) To move to a position on the page (used with the `|` operator).

As they are so different, this package distinguishes between them. There is an advantage to the use of these, in that they may be redefined in particular environments, e.g. to change the length of a completed diversion when it is output.

`.Space dist`

This acts by default identically to the `.sp` command. If called by `'Space` it will not cause a break. In future, this command may give an amount of space which may stretch or shrink to improve page layout, e.g. to avoid ragged bottoms.

`.ExtraSpace num`

This command is for the user who needs extra space inserted. The space will not shrink or stretch. Its most likely use is in the final tuning of a finished document.

`.Position dist`

This moves to the absolute position `dist` on the page. It does not cause a break.

6.1.1 Filling Modes

These commands are documented here for convenience.

`.FillOff`

`.FillOn`

These commands should occur in matched pairs (which can be nested). `.FillOff` turns off filling, whereas `.FillOn` restores the filling mode prior to the invocation of its matching `.FillOff`.

6.2 Date Package

6.2.1 Commands

There is one command

`.DateStrings year month day [dw]`

which takes a date in numerical form, and generates a number of strings which may be used to print the date in a wide variety of formats. The arguments are

`year` the year. It is taken literally. Older documents using this package having the year less than 4 digits will have to be corrected.

`month` the month, a number from 1 to 12;

`day` the day of the month, a number from 1 to 31;

`dw` the day of the week, as a number 1 to 7, with 1 representing Sunday. This argument is optional; if omitted it will be worked out, but if given it is believed.

The command

`.TodaysDate`

sets up the strings for today's date, using the `.DateStrings` command, and *gtroff's* internal registers.

6.2.2 Defined strings

All names of days and months are in English.

`date-month` the month name in full, e.g. February

`date-smonth` the short name of the month, e.g. Mar

`date-year` the year (exactly as given);

`date-daynum` the day of the month as a 1- or 2-digit string;

`date-dayname` the name of the day in full, e.g. Wednesday;

`date-sdayname` the short name of the day, e.g. Wed;

`date-suf` the ordinal suffix of the day number e.g. rd for 23rd;

`LongDate` example: Wednesday 31st October 2012

`ShortDate` example: Wed 31st Oct, 2012

`PlainDate` example: 31-Oct-2012

6.3 String Manipulation

This package has some string manipulation commands. In some cases the name of a command is not in the convention of the `-markup` system, but is in lower case, to reflect the similarity to certain functions of the C-language library.

6.3.1 Commands

`.ForEach string command`

This command executes `command` with each character of `string` in turn as its argument.

`.toupper name char`

This defines the string `name` to be the upper-case version of `char` if appropriate, else sets it to the empty string.

.tolower name char

This does the same as `toupper` but converts to lower-case.

.strchr reg string char

This defines the number register to be the first position in `string` of the character `char`, or `-1` if the character does not occur in the string. Strings are indexed from 0.

.strrchr reg string char

This defines the number register to be the last position in `string` of the character `char`, or `-1` if the character does not occur in the string.

.basename name string

This sets the string `name` to be the portion of `string` with everything up to the last `/` removed, then everything from the last `.` removed also.

.StringVal name1 name2

The arguments are string names. The command defines `name1` to have the value of `name2`. If this value is undefined the number register `str-err` is set to non-zero, no error message is printed.

Chapter 7

Special effects

7.1 Included and Other Pictures

This package imports an illustration and places it at a given point and at a given size. The only form of file currently dealt with is a POSTSCRIPT file.

Currently the picture must be a POSTSCRIPT EPS file and be properly structured, with a `%%BoundingBox:` comment. If the output device is not POSTSCRIPT a box of the right size and position is drawn instead.

7.1.1 Commands

.Picture [opts] file [width [height]]

This reads in the file which is assumed to contain a graphic in some known format, and places it within the output document at a position depending on the `opts` argument. There are two groups of options: `display` and `inline`; the `display` options all start on the next line and leave the current position at the line after the end of the picture. The `inline` option leaves the current point the same, regardless of where the picture is drawn. The `display` options are:

–C the picture is centred; this is the default.

- L the picture’s margin is left adjusted;
- R the picture is right adjusted;
- I *d* the picture is indented by the distance *d*;

The inline options are:

- TL *x y* the picture is positioned so that its top left corner is at point (x,y) .

The other inline options are similar with TL being replaced by TR (top right), BL (bottom left) and BR (bottom right). (*note*: (x,y) is absolute with respect to the top and left margins of the *paper*; this may change).

If no option is given, the picture is centred.

If `height` is not given then the picture will be uniformly scaled and will have width and height determined by `width`. If `width` and `height` are given then the scaling will possibly be non-uniform. If `width` is replaced by `–h` then the picture is scaled to have height `height`.

All distances must have units given; although having a superficial resemblance to the PSPIC command of *groff*, there are no default units, as a matter of principle.

7.1.2 Support for ‘pic’

.PS height width
.PE

These commands are used to delimit input to *pic*(1) which generates the width and height (what does it do with user-supplied values?). In this preliminary version, no extra spacing or adjustment is done.

7.1.3 Picture packing

The next few commands are a rather rudimentary collection for placing included pictures side by side on the page. They are aligned so that the bottom edges of the pictures are on the same baseline. Each picture is also outlined with a thin black line.

.InitPic

This initialises internal storage for holding the picture information.

.SetPic width file

This store up the information for the picture contained in the file *file*. *width* is the desired width of the picture (the height is deduced from the contents of the file, using equal scaling).

.PlacePic

This draws the stored pictures. The left and right margins are flush with the edges of the line, and they are spaced by equal gaps; there is no check that there are at least two pictures. Afterwards the current position is after the pictures.

7.2 The Verbatim Package

.BeginVerbatim

This starts a section of the input which will be reproduced exactly, with no processing. It is especially useful for displaying segments of *troff* code. It is useful in conjunction with a display or indenting package.

.EndVerbatim

Ends the segment of verbatim text begun with `.BeginVerbatim`. *Warning*: if this command is misspelt, the rest of the document will be wrong.

7.3 The Boxes Package

This package consists of miscellaneous commands to draw boxes, either round text or a word, or draw a plain box, where, for example a picture could be pasted in to the final document.

7.3.1 Commands**.BeginBox t b l r w**

Draw a box round the following text, until `.EndBox`, The text is formatted normally, except for being slightly narrowed to accommodate the box. The user may supply extra adjustments at the top, bottom left and right (`t`, `b`, `l`, `r` arguments), or set the thickness of the box outline (`w` argument). The arguments are all optional, but the value 0 for any one of them will be replaced by the default. the defaults are *not* changed. The line thickness may be correct only for the `-Tps` output, at present. These commands may not be nested.

.EndBox

Terminates the text to be boxed.

.BoxWord word

Draws `word` inline, with a neat box round it. The box takes into account the letter shapes, and in general gives a better result than the `.BX` command of `-ms`. There is a restriction: the word may not contain a space. Use `\0` instead.

.MakeBox width height

Draw a box at the current position, finishing at the bottom left of the box. It may be used inside a diversion. If used in a diversion, and the diversion is interpolated, the user has to space to the bottom of the box, using `.Space`.

.DefBox [-a] name width height [dx [dy]]

This defines a string `name` which when interpolated will produce a box of size `width`×`height` at the current point, returning to the current point. The box will be shown `dx` units to the right of the current point and `dy` units downwards (defaulting to 0). If the `-a` argument is used, the horizontal position is *after* the box by an extra `dx` units.

.Cartouche radius width height

This command draws a cartouche shape at the current point. The current point afterwards is probably not well-defined. A cartouche is a rectangle (of width `width` and height `height`), but the corners are rounded — a quarter circle with radius `radius`.

.Box width height

This command draws a box *downwards* from the current point, with width `width` and height `height`.

The `MakeBox` command is probably more satisfactory.

7.4 The Dotted line package

This package will eventually do various kinds of dotted and dashed lines. Some of these will be pure *troff* constructs; others will use the particular facilities of an output device, such as POSTSCRIPT. A package for constructing “forms” is under consideration, of which this will be a part.

`.Dotted word [char [space [endstr]]]`

This prints the word `word` on a line by itself, and fills the rest of the line with dots. If the optional second argument is given, then it is the character to use instead of dots. If the optional third argument is given, then it is the spacing between dots. If the optional fourth argument is given, then it is the string to print instead of the right hand dot. If any of the optional arguments are empty, then the default setting is not changed. The default spacing is reasonable. With given settings, different lines filled used this command are guaranteed to have the dots lined up (unless things change such as the line length).

It should be noted that the “dot” may in fact be a string.

`.SetDotDefaults [char space endstr]`

With no arguments, this command sets the standard default “dot” character, spacing, and final character/string for the `.Dotted` command. With three arguments, it changes the defaults to the values specified.

7.5 Insert package

This package enables simple rectangular insertions at the right or left of the document text.

This package does not work inside diversions at present, but that is being fixed.

7.5.1 Commands

`.BeginInsert [name]`

This starts a section of material to be saved for insertion as described above. If the optional argument `name` is given, then it will be saved using this name. *Note:* in this package the words `left`, `right`, `box` and `keep` are reserved so cannot be used for names. It is up to the user to determine the width of the text; as it is processed in its own environment, which has the standard settings for the document. Line length, etc. may be changed (and will be restored to standard values on the next use).

This is an example of an insertion placed on the right of a page.

`.EndInsert`

This terminates the material to be inserted. It is stored under the given name (or anonymously if no name is given).

`.InsertSep top bot side`

This specifies the spacing to be used round the insertion, in *goff* units. These remain set unless changed. There are default values.

`.PlaceInsert left|right [name] [box] [keep]`

This places the saved material in a rectangular space on the left or right. The arguments may be in any order. If the name is not specified then the insertion is defined by the most recent `.BeginInsert` without a name. A small amount of space is placed round the material (see `.InsertSep`). The insertion occurs after the line which would contain the word immediately preceding the command.

If the optional argument `box` is used it indicates only that an addition spacing adjustment is used. This is advised if the saved material consists only of a box outline, as box horizontals are drawn on the baseline.

If the optional argument `keep` is used, then the diversion used to store the placed material is not removed, as it would be normally. It may then be used in another `.PlaceInsert` command. To remove the diversion, use `.RemoveInsert [name]`. WARNING: this

has not yet been implemented.

.CentreHole width height

This leaves a rectangular hole in the text. This command is experimental, and is not advised if the text is not “dense”, apart from the fact that it would not look good. It is currently the user’s responsibility to fill the hole, but that should be done by this package in future (this command may disappear in favour of a new option (`centre`) to `centre` `.PlaceInsert`).

7.6 The Drop Package

This package offers support for various styles of initial dropped capitals.

.Drop word lines

This is intended for use at the beginning of a non-indented paragraph; `word` is the first word. The initial letter is enlarged and dropped into the surrounding text and takes `lines` lines. Currently the initial is both enlarged and stretched upwards. In future the style may be tunable. Currently it does not work inside a diversion, but that is being fixed.

.Yinit word

This is a customised version of `.Drop` using the `yinit` font from T_EX (by Yannis Haralambous). The initial letter is set at 10-point in this font, which must be available in POSTSCRIPT form.

.AugInit word

This is another customised version of `.Drop`, using the scalable `AugsburgInitials` illuminated capitals font.

7.7 The Colour Package

This original version of this package was written before colour was added to troff and applied only to POSTSCRIPT output. It has been rewritten using troff’s colour facilities, but the commands defined here are retained for backward compatibility.

In this package colours are specified in red, green and blue components, each a real number in the range $[0..1]$.

7.7.1 Commands

.FillWithColour R G B

This draws a rectangle filling the whole page with the colour RGB. Place it carefully if you do not wish it to cover up previous output.

.DefColour name R G B

This defines `name` to be a string which when interpolated **changes the current colour** to be that with the given RGB components. The colour range must be ended (on the same page) by the string `e-c`.

.ColourBox R G B width height

This draws a solid rectangle starting and finishing at the current point. The colour is given by the RGB arguments, and `width` and `height` give the dimensions.

Chapter 8

Moving material

8.1 Cross References

This enables references to be made to other parts of a document. Formatting may be required more than once to obtain all the references correctly.

8.1.1 Defining references

There is one command

`.CrossRef name type`

This defines the string `name` to have a value which may be referenced in other parts of the document. These names and values are stored in a file (with extension `.ref`) for future passes.

The argument `type` says what kind of value is used. Currently those predefined are

`p`, `page`.

Use the page number.

`s`, `section`.

Use the section number. They may have a number appended, meaning that subsections only up to that level are used (number not yet implemented).

The user can define new types; all that is required is for the new type (*type say*) to have a corresponding macro

```
.cref-def-type
```

which must assign the required value to the string variable `cref-val`.

The reasons for this indirect approach are: (a) flexibility; (b) in case the material is diverted (to get the value from the place the diversion is output rather than the current place).

8.1.2 Using references

There are a variety of ways of using references, from simple unadorned values (including in-lined values), to fancy as in "this page", "previous page" and so on. There are therefore several commands for using cross references.

If the `name` is not defined a warning is displayed, and a square symbol is placed in the text. If the value is incorrect, a warning is also displayed. The formatting must be re-run to get the reference correct (sometimes more than once).

`.UseCrossRef name`

`*[UseCrossRef1 name]`

These insert the value associated with `name` by `.CrossRef` into the text. The second is the inline version of the first.

`.Ref [pre] name [post]`

This is replaced by the value that `name` refers to (if defined), prefixed by `pre` if present, and suffixed by `post` if present.

`.PageRef name [text]`

This generates the appropriate one of “page *n*”; “this page”; “the previous page”; or “the next page”. `text` if given is appended (it is usually punctuation).

`.SetCrossRef name refname`

Defines the string `name` to have the value of the string `refname` which should be defined by one of the commands of this package somewhere in the document.

8.1.3 Deprecated commands

These are commands retained from earlier incarnations of this package, retained for legacy reasons.

`.MakeCrossRef label value`

This defines the string `label` to have the value `value`, and also makes the label available as a string register for use in other parts of the document. The value supplied is currently up to the user; a common one is the page number `\n[%]`.

`.MakePageRef name`

This is the same as `.MakeCrossRef`, but takes one argument (the name), the value being the current page number.

8.2 Table of Contents

This package has facilities for collecting information for a table of contents, which is normally near the beginning of the document. Then it is usually necessary to reformat it (possibly more than once) to obtain correct page numbers. The table may be elsewhere in the document, the most common alternative being at the end.

The material to be placed in the contents is stored in a file whose name ends in `.toc`.

`.MakeTocEntry ref entry.`

This identifies `ref` with the string `entry` which is the one to appear in the table of contents, along with the page number where this command appears. Warnings are issued if any entry changes; several runs may be required before the page numbers are correct.

`.TableOfContents`

Table of contents may be created by use of the command where the contents are to be placed. If this is anywhere other than at the end, the formatting run may have to be repeated; warnings are given. There is a simple version supplied with the package, but classes may dress it up.

8.3 Keeps and Floats

This package enables material to be kept together on one page, by changing page if necessary. In this initial version, keeps cannot be used within any material which has to be diverted, such as within two-column mode.

The usual reason for a keep is to print a self-contained item such as a display or table, where there are no natural text breaks. If an arbitrary text segment is kept, note that there will be breaks before and after, and there may be partial words in the wrong place.

8.3.1 Commands

.BeginKeep [float]

Starts a section of text to be kept. If the optional argument `float` is given, then the text to be kept is stored for insertion wherever room can be found, while normal text processing continues. The kept material in both cases is output immediately if there is room, and at the top of the next page otherwise (provided no other keeps have to be output first). With a non-floated keep, the order of text is unchanged.

.EndKeep

Ends a section of material to be kept.

8.4 The Index package

The *Index* package is designed to be used in conjunction with the *makeindex* public domain program, and so the conventions are closely tied to those of that program.

During index creation the file *root.idx* is created, where *root* is the same file basename as is used for cross references and contents. If the index is also printed in the same run, the file *root.ind* and *root.ilg* are also created. These may all be removed later.

8.4.1 Index creation commands

.Index args ...

This command simply concatenates its arguments and writes them with page information to the *index* file. Spaces are inserted where the user puts a space and also between arguments. Multiple spaces are compressed. If the page number has to stand out, e.g. as for the main occurrence of an index entry, then end the argument list with `|B` (for bold emphasis) or `|I` (for italic). The `B` or `I` may be replaced by any *groff* font style or face.

The output has to be post-processed by the program *makeindex*, and therefore certain characters are used for special effects:

- ! the character used to separate subitems;
- @ The character used to specify what is printed (as opposed to the item, which is used for sorting — if they are different);
- | The character used to encapsulate page numbers;
- (The character used to begin an explicit page range;
-) The character used to end an explicit page range;
- # The “quote” character, used to protect any of the above special characters.

For more details, see the documentation for the *makeindex* program.

.MakeIndex [style]

This command may be used anywhere after `.BeginDocument` and acts simply as a message to instruct `.EndDocument` to close the index file, run *makeindex* and typeset the result. It is done this way in case of inserted material which may contain index entries, such as floating keeps.

style

This argument gives the style of the index; there is a default. Currently there is just one style, `markup`.

If for testing or other reason, it is desired to print the index now, the command `.idx-print` may be used.

Chapter 9

Classes

9.1 Classes

Table 3 on this page lists the classes that so far have been implemented. A later section describes some example user packages.

Package	Brief description	Reserved prefix
Article	Articles	art-
Letter	Multiple letters	let-
Booklet	Handbooks and newsletters	bkl-

Table 3

9.2 The Article Class

This is one of the predefined document classes. It's main features are a title, an abstract, one or more authors and their institutions. All of these are optional. They should appear just after `.BeginDocument`.

9.2.1 Commands

`.Title`

This command takes any number of arguments. Each will be printed centred on a separate line as part of the document title. Currently no part of the title appears anywhere else in the document, but in future it might be available as part of a page header or footer.

`.Author name [institution]`

An author's name and institution. There may be several authors, each given by a `Author` command.

`.EndAuthors [left | centre | right] [row | col]`

This marks the end of the author list; if not present, the authors will not be displayed. The arguments specify whether the individual authors have their entries left-, centre- or right-adjusted, and whether the list of authors is displayed side-by-side (`row|col` argument) or one under the other. More authors followed by `.EndAuthors` may follow. The default positioning is side-by-side, each centred in its own space.

`.BeginAbstract`

and

.EndAbstract

together limit an abstract, which will be set in a slightly different way, preceded by a heading "ABSTRACT".

9.3 The Letter Class

This class is designed to simplify customising a letter style, with its often rather peculiar layout. It is probably best used with a package which pre-customises many of the individual style settings. There is support for logos and banner headings, multiple letters, default signatory, postscripts and dates. The overall format of a letter is

```
.BeginLetter
[ .Addressee ]
[ .Sender ]
.Salutation
. . . (text of letter)
.Sign
[ .PostScript ]
.EndLetter
. . . (more letters)
```

There are lots of hooks and handles for customising letters; these are not described in the user documentation, but are provided for those who wish to make a customised letter package.

.BeginLetter**.EndLetter**

These delimit a single letter. There may be any number of them in one document. A letter is started with a banner heading at page 1. If a particular letter requires more than one page then each except the last has a "turn over" message at the bottom (by default).

.SetDate date

This command sets the string to be placed in the date position to *date*. If the argument is *today* then the date used is "today" — the day that the letter is formatted. It is in the PlainDate style. If the user wants a different style, then use the Date package. If the argument is *none* then the date is omitted. *Note*: this should be placed before *.Sender* as the latter incorporates the current date, which is "today" by default.

.Salutation name

This is obligatory after *.BeginLetter*, and generates the "Dear ..." at the beginning of the letter, using the argument *name*. The *.Postscript* command is not implemented.

.Sign [n|f|s|-b *bye*] [*name*]

This is obligatory after the text of the letter and before *.EndLetter*. It generates the message "Yours sincerely" by default. If one of the arguments is a single letter *n*, *f* or *s* then the the farewell is none, "faithfully" or "sincerely" respectively. If an argument is *-b* then the next argument *bye* is used as the signing-off phrase. If a name is given, it is placed in brackets after room for a signature: use this for formal letters.

.LetterExtra

If there is material after the signature, but before *.EndLetter*, then use this; it completes the signature and the user can insert any material. A new page or extra space is not inserted, such details are up to the requirements of the user.

.Banner

This produces the banner at the top of a letter. The default is a .5 inch blank space.

.Sender name address ...

This may occur *either* once before the first `.BeginLetter`, in which case it will be used for all letters, *or* it may occur once at most within each letter. It formats the name and address of the sender (at the top right of the letter, under the banner), followed by the date if specified. It may have any number or arguments, including none; each is printed on a separate line.

.SenderName name

This sets a name to be used as the name in `.Sign` if none is given; use for formal letters only.

.Addressee [name] add ...

This typesets the name and address of the person to whom the letter is being sent at the top left, under the banner. It is not obligatory, but should appear after `.BeginLetter` and before `.Salutation`. Each argument is printed on a separate line.

.NoTurn

This suppresses the “turn” message at the foot of all but the last page of a letter. It affects all subsequent letters if it occurs outside a letter, or only that letter if placed after `.BeginLetter`.

9.4 The Booklet class

This is a class designed for making informal booklets, such as newsletters and handbooks. It is used mainly with customised packages which manipulate sizes and styles to suit.

9.4.1 Commands

.TwoPartTitle left right

This command prints a title in two parts, on the left and right respectively, with a horizontal line joining them. Either part may be empty (then the joining line is extended to the text margin). No additional spacing is inserted by the command.

.Wrap space header pageopt footeropt

This command acts as a “wrapper” for included “raw” article files. It is followed by a command `.so file`.

The first page of the file is decorated as follows: If the `pageopt` argument is “newpage”, then the article will start on a new page, and the `header` will be used; it is a name indicating which of a set of fancy strings is used in the “running header”; these are described in those packages which use them. This argument may, however, have the special values

empty

There will be no page header but an amount of space at the top of the page can be introduced by the `space` argument.

blank

There is a blank page header of the standard width.

– There is an ordinary header, but the running header is not changed.

The `footeropt` can be one of `footer` or `nofooter`, indicating that a page footer is required or not respectively (on the new page, if taken).

If the space argument is non-zero, then this amount of space is inserted before the article; usually used between articles on the same page.

`.Auth author [date]`

This identifies the author of an article (with optional date of article). It is designed to go at the end of the article.

Chapter 10

Customised packages

Most of these have been done for the author's own applications, but are available in a separate document.

Packages of the Booklet Class

10.1 Address labels

This is a subpackage of the `Booklet` class, and is designed to print addresses on sheets of self-adhesive labels. The input is a series of addresses, one address line per line, with the following markup:

1. The address must be preceded by the line
`.de lab-label`
2. The address must be followed by the lines
`..`
`.lab-print`
3. The addressee(s) must be first, and be the arguments to the command `.lab-name` — for example
`.lab-name "John" "Mary Smith"`
4. The set of addresses must be between the following lines
`.lab-begin`
`.lab-end`

These can be done by hand but it is usually easier to write a script (e.g. for `awk(1)`) to convert from whatever address database is used.

A useful string register to set is `lab-debug`. If defined this will cause the label borders to be drawn.

Note: there is an as-yet untraced bug which causes these outlines to be misplaced on the last column of the *first* page only.

`.UseLabels make`

As the dimensions of self-adhesive brands vary widely, this command is defined to allow the brand to be selected. If a brand is not known, it is easy enough to define a new brand,

say a brand `xxx`. A string called `lab-xxx` with 12 numerical values is then defined.

1. Number of rows of labels;
2. Number of columns of labels;
3. Left margin — all makes seem to have same right and left margins;
4. Right margin;
5. Top margin — top and bottom margins are usually the same, too;
6. Bottom margin;
7. The horizontal distance from one label to the next;
8. The width of a label;
9. The vertical distance from one label to the one below;
10. The height of a label;
11. The minimum margin allowed on a label around the print;
12. The radius of the rounded corners.

See the source of this package for the meaning. Note that the outer labels may overlap the non-printable area of the medium; this is not yet taken into account.

.UsePrinter printer

This command sets the printable area of the paper for a known printer, and is used because printers vary widely on how near the edges can be marked. To define a printer, say `xxx`, define the string `lab-xxx` to have the following 4 space separated numerical parts:

1. Left unprintable width;
2. Top unprintable width;
3. Length of printable horizontal line;
4. Length of printable vertical line.

Chapter 11 Internals

11.1 The Error Package

This package contains error and warning message commands, which will cause output to the standard error file.

.Error args ...

This prints its arguments and terminates formatting. The first argument is assumed to be the name of the command calling it. The file and line number are also printed. If the number register `error-verbose` is positive, then a call backtrace is printed if it has the 1-bit set, and if the 2-bit is set then trap positions are printed.

.Warning args ...

This does the same as `.Error` but does not exit, and the word `Error` is replaced in the output by `Warning`. If the number register `error-verbose` has the 4-bit set then then extra diagnostics are produced as for `.Error`; this should be used with care as the if there are lots of warnings the diagnostic output will be cluttered.

.SimpleWarning message...

This prints the arguments on standard error, but without the file and line number. The first argument is not treated specially.

11.2 Debugging

This package is mainly for package and class writers who want to print out number register values.

.Debug [string [reg ...]]

This prints a message on standard error, one line per register. The given string is attached (together with a number) as identification to each value. The registers are given by name.

.debug-print-mac name

This prints the command `name` in-line in the output text. For example, here is the value of the command `Box` at this point in formatting:

```
.nr box-boxw \${1}
.nr box-boxh \${2}
\!p 0 \n[box-boxh]u \n[box-boxw]u 0 0 -\n[box-boxh]u -\n[box-boxw]u 0'
```

11.3 The Stack Package

This simulates a stack for general package use. Numbers and strings use a common stack. It is intended mainly for class and package authors. *Troff* has some stacks — the diversion and environment stacks, but nested items also need a stack, hence this one. It is tested for emptiness at the end of the document.

.Stackn name

This stacks the value of number register `name`. Note: the argument is the name, not the value.

.UnStackn name

This removes a value from the stack and assigns it to number register `name`.

This will work only if the format assigned to the register is the default decimal digit format.

.Stacks name**.UnStacks name**

These are similar to `.Stackn` and `.UnStackn`, but deal with string registers.

.PrintStack

If the stack is not empty, then it is printed on standard error. This is a debugging aid.

11.4 The Environment Package

This package was written before the `.envc` command was added to troff. With its use more environmental parameters are saved, and this may affect legacy documents.

This package builds on *troff*'s environments to make them more useful, and to supply some error checking. *WARNING*: This package is experimental, and although it will not disappear, its commands may change, when the best ways of using them are discovered.

The basic problem with “raw” environments is that they start with the *default troff settings*, which almost never what is required.

.NewEnv name

This defines a new environment, with the given *name*. The attributes within this environment are those for the document, *not* the default *troff* values. The environment is not entered. It is an error if the named environment already exists.

.RenewEnv name

The existing environment has its attributes restored to the standard values for the document; it is not entered.

.CopyEnv name

The named environment must already exist, and will have the the *current* attributes copied into it.

.BeginEnv name

The named environment is entered; it must already exist.

.EndEnv name

This exits from the current environment to the containing one. It is an error if the current environment is not the one named.

.BeginStandardEnv

.EndStandardEnv

These commands delimit text where the standard settings may be changed, such as point size, font, line length, etc. Initially the settings are the standard document settings. After *.EndStandardEnv*, the previous environment is restored.

11.5 The Diversion Package

The purpose of this package is to give support for diversions to packages which require them. It is mainly for the authors of packages and classes.

The problem with diversions is that only one diversion trap is allowed. This package allows the use of several diversion traps within one diversion.

There are no breaks at the start and end of diversions defined in this package; it is the user's responsibility to put them where required. Neither is the diversion interpolated. After all uses of the diversion, it should be removed, as there is a restriction imposed in this package that a diversion may not have an existing name.

11.5.1 Commands

.BeginDiversion name

This starts a diversion with the given *name*. There is a restriction in this package that there must not be an existing diversion also called *name*, whether defined by this command or otherwise.

The user should be aware that there may be a partially formed line which may end up in the diversion. Use some kind of command which finishes off a section (such as *.Para*), if this is undesirable.

.EndDiversion

Finishes the current diversion. There is no check that it is the same as the one started with the matching `.BeginDiversion` as there already is a diversion stack within *gtroff*. There is no break, as there are circumstances when that would be the wrong thing to do.

.SetDiversionTrap position name

Sets a trap to be sprung at `position` within the current diversion. The argument `name` is the user's command to be called.

The trap is appended to a priority queue of traps for the diversion. The trap that is sprung first is the one nearest the current position. After use, it is removed, and next one set (as only one can be active at a time).

Acknowledgements

- James Clark, for his brilliant implementation of the *troff* family of programs, and whose macro code I have frequently borrowed;
- Leslie Lamport, for inventing LaTeX, the source of many of my ideas.

Index

A

A4paper, 6
 A5paper, 6
 address labels, 31
 appendices, 9
 Appendix, 9
 AugsburgInitials, 24

B

basename, 20
 BeginFullPage, 8
 BeginInsert, 23
 BeginVerbatim, 21
 Box, 22
 bullet (•), 13

C

Cartouche, 22
 Class
 Article, 28
 Letter, 29
 class
 Article, 28
 Booklet, 30
 Letter, 29
 Contents, 26
 contents, 26
 adding to table, 26
 printing, 26
 Cross references, 25
 CrossRef, 25

D

Date Package, 19
 date strings
 date-daynum, 19
 date-month, 19
 date-sdayname, 19
 date-smonth, 19
 date-year, 19
 LongDate, 19
 PlainDate, 19
 ShortDate, 19
 Debug package, 33
 DefineDisplay, 15

display
 Emph, 16
 Program, 16
 Text, 15
 diversions, 34
 Documentation
 maintenance, 3
 tuning packages, 3
 user
 contained in package, 3
 dots (...), 13
 Dotted package, 22
 DoublePageSpread, 8
 Drop package, 24

E

EndDocument
 empty stack, 5
 finish diversions, 5
 finish environments, 5
 output floats, 5
 print index, 5
 EndFullPage, 8
 EndInsert, 23
 EndKeep, 27
 EndVerbatim, 22
 Environment package, 33
 EPS file, 20
 Error package, 32

F

file
 extension
 .idx, 27
 .idx, .ind, .ilg, 5
 .ilg, 27
 .ind, 27
 .ref, 5, 25
 .toc, 26
 root name, 5
 FillWithColour, 24
 font style commands
 argument conventions, 10
 ForEach, 19

G

groff, 1

H

Haralambous, Yannis, *yinit* font, 24
head-contents, 9
head-ranges, 9
heading, alignment, 8
HideFooter, 7

I

index
 special effect characters, 27
index entry
 in floating keep, 27
index style
 default, 27
InitPic, 21
Insert package, 23
InsertSep, 23
IP, 9
IPx, 9

K

Keep package, 26

L

lab-begin, 31
lab-end, 31
lab-label, 31
lab-name, 31
lab-print, 31
labels, address, 31
Letter
 customising, 29
lists
 Bold, 17
 Enumerate, 16
 FixedWidth, 17
 Itemise, 16
 Null, 17
LucidaSans-Typewriter, 11

M

makeindex, 27
-markup

Preliminaries, 3

ms
 BX, 22
-ms, 1, 3

N

names within packages, 5
NewPage, 8

O

obsolete commands
 ZC, 11
OneColumn, 13
OneSided, 8

P

packages
 base, 5
 Boxes, 5, 22
 Colour, 5, 24
 Contents, 5, 26
 Crossref, 5, 25
 Date, 5, 19
 Display, 5, 15
 Div, 5
 Doc, 4, 5
 Dotted, 5, 22
 Drop, 5, 24
 Env, 5
 Eqn, 5, 14
 Error, 5, 32
 Fonts, 5, 10
 Headings, 5
 Index, 5, 27
 Insert, 5, 23
 Keep, 5, 26
 Lists, 5, 16
 Page, 5, 7
 Para, 5, 9
 Pics, 5, 20
 Predef, 5, 13
 Sizes, 5
 Space, 18
 Strings, 5, 19
 Tbl, 5, 13
 Twocol, 5, 13
 Verb, 5, 21

- customised
 - Labels, 31
 - Wide, 18
- optional
 - Debug, 33
 - Headings, 8
 - HeadRedef, 9
 - Lucida, 11
- system base
 - Div, 34
 - Env, 33
 - Space, 5
 - Stack, 5, 33
- Page package, 7
- Para, 9
- Pics package, 20
- Picture, 20
- PlaceInsert, 23
- PlacePic, 21
- POSTSCRIPT, 11, 20, 22, 24
 - devps, 22
- postscript (POSTSCRIPT), 13
- pound (£), 13
- pounds (£), 13
- preamble, 4

R

- RE, 10
- references, 25
- RevealFooter, 7
- RS, 10
- RunningHeader, 8

S

- Section
 - Cross references package, 25
 - Document structure, 4
 - Font control package, 10
 - Indexing, 27
 - String manipulation package, 19
 - Support for equations, 14
 - Support for tables, 13
 - The List package, 16
- section headings, 8
- SetFooterOpt, 7
- SetPageFooter, 7
- SetPageHeader, 7
- SetPic, 21

- SetPointSize, 6
- Sizes package, 13
- small caps, 11
- strchr, 20
- strings
 - bullet (•), 13
 - dots (...), 13
 - pound (£), 13
 - pounds (£), 13
 - TeX (T_EX), 13
- StringVal, 20
- strchr, 20
- Style parameter
 - changing, 3

T

- tables
 - boxed, 13
 - header, 14
 - multi-paged, 14
- TeX (T_EX), 13
- tolower, 20
- toupper, 19
- TP, 10
- TPx, 10
- Twocol package, 13
- TwoColumn, 13

U

- UseLabels, 31
- UsePrinter, 32
- User commands
 - A4Paper, 6
 - A5Paper, 6
 - Addressee, 30
 - Appendix, 9
 - AugInit, 24
 - Auth, 31
 - Author, 28
 - BI, 11
 - B, 10
 - Banner, 30
 - BeginAbstract, 28
 - BeginBox, 22
 - BeginDisplay, 15
 - BeginDiversion, 34
 - BeginDocument, 5, 27

BeginEnumerate, 16
BeginEnv, 34
BeginFullPage, 8
BeginInsert, 23
BeginItemise, 16
BeginKeep, 27
BeginLetter, 29
BeginList, 16
BeginStandardEnv, 34
BeginVerbatim, 21
BoxWord, 22
Box, 22
CAL, 11
CW, 11
Cartouche, 22
Chapter, 9
ColourBox, 24
CopyEnv, 34
CrossRef, 25
DateStrings, 19
DefBox, 22
DefColour, 24
DefineDisplay, 15
DocumentClass, 5
Dotted, 23
DoublePageSpread, 8
Drop, 24
EN, 14
EQ, 14
EndAbstract, 29
EndAuthors, 28
EndBox, 22
EndDisplay, 15
EndDiversion, 35
EndDocument, 5, 27
EndEnumerate, 16
EndEnv, 34
EndFullPage, 8
EndInsert, 23
EndItemise, 16
EndKeep, 27
EndLetter, 29
EndList, 16
EndStandardEnv, 34
EndVerbatim, 22
EquationStyle, 14
Error, 32
ExtraSpace, 18
FillOff, 18
FillOn, 18
FillWithColour, 24
ForEach, 19
Heading, 8
HideFooter, 7
IP, 9
IPx, 9
I, 10
Index, 27
InitPic, 21
InsertSep, 23
Item, 16
Itemx, 16
LetterExtra, 29
MakeBox, 22
MakeCrossRef, 26
MakeIndex, 27
MakePageRef, 26
MakeSC, 11
MakeTocEntry, 26
NewEnv, 34
NewPage, 8
NoTurn, 30
OneColumn, 13
OneSided, 8
PE, 21
PS, 21
P, 11
PageRef, 26
Para, 9
Picture, 20
PlaceInsert, 23
PlacePic, 21
PlainDate, 29
PrintStack, 33
RE, 10
RS, 10
R, 10
Ref, 26
RenewEnv, 34
RestoreFamily, 11
RevealFooter, 7
RunningHeader, 8
SC, 11
Salutation, 29
Section, 8
SenderName, 30

Sender, 30
SetCounter, 17
SetCrossRef, 26
SetDate, 29
SetDiversionTrap, 35
SetDotDefaults, 23
SetFooterOpt, 7
SetPageFooter, 7
SetPageHeader, 7
SetPic, 21
SetPointSize, 6
Sign, 29
SimpleWarning, 33
SkipCounter, 17
Space, 18
Stackn, 33
Stacks, 33
StringVal, 20
SwitchFamily, 11
T&, 14
TE, 14
TH, 14
TP, 10
TPx, 10
TS, 14
TableOfContents, 26
TableStyle, 14
TheFamily, 11
Title, 28
TodaysDate, 19
TwoColumn, 13
TwoPartTitle, 30
UnStackn, 33
UnStacks, 33
UseCrossRef1, 25
UseCrossRef, 25
UseLabels, 31
UsePackage, 4, 29

UsePrinter, 32
Warning, 33
Wrap, 30
Yinit, 24
basename, 20
debug-print-mac, 33
debug, 33
e-c, 24
idx-print, 28
lab-begin, 31
lab-end, 31
lab-label, 31
lab-name, 31
lab-print, 31
strchr, 20
strrchr, 20
tolower, 20
toupper, 19

V

variables
 ParaDefaultTagWidth, 10
 ParaIndent, 10
 ParaRelIndent, 10
 ParaSep, 10
Verb package, 21

W

Wilson, Denis M., 1

Y

yinit font, by Yannis Haralambous, 24

Z

ZapfChanceryMediumItalic, 11