# Eplain: Expanded Plain TEX

July 2020
3.11

**Karl Berry**
**Oleg Katsitadze**
**Steven Smith**

This manual documents the Eplain macros, version 3.11, July 2020. Eplain provides functionality for plain TEX that is intended to be useful regardless of how your document is actually formatted.

Most of this manual is in the public domain, like most of the Eplain code. It was originally written by Karl Berry, starting in 1989. Steven Smith wrote the documentation for the commutative diagram macros; this chapter is under the GNU General Public License. Adam Lewenberg has made additions and corrections. Oleg Katsitadze wrote the section on LATEX packages and the chapter on hyperlinks, and updates throughout.

# Short Contents

# Table of Contents

# 1 Introduction

The *Eplain* macro package expands on and extends the definitions in plain TeX. Its home on the web is `https://tug.org/eplain`.

This manual describes the definitions that you, as either an author or a macro writer, might like to use. It doesn't discuss the implementation; see comments in the source code (`xeplain.tex`) for that.

Eplain is not intended to provide typesetting capabilities, as does LaTeX (originally written by Leslie Lamport) and Texinfo (Originally written by Richard Stallman). Instead, it provides definitions that are intended to be useful regardless of the high-level commands that you use when you actually prepare your manuscript.

For example, Eplain does not have a command `\section` to format section headings in an "appropriate" way, such as LaTeX's `\section`. The philosophy of Eplain is that some people will always need or want to go beyond the macro designer's idea of "appropriate". Such canned macros are fine—as long as you are willing to accept the resulting output. If you don't like the results, or if you are trying to match a different format, you have to put in extra work to override the defaults.

On the other hand, almost everyone would like capabilities such as cross-referencing by labels, so that you don't have to put actual page numbers in the manuscript. The author of Eplain is not aware of any generally available macro packages that (1) do not force their typographic style on an author, and yet (2) provide such capabilities.

Besides such generic macros as cross-referencing, Eplain contains another set of definitions: ones that change the conventions of plain TeX's output. For example, math displays in TeX are, by default, centered. If you want your displays to come out left-justified, you have to plow through *The TeXbook* to find some way to do it, and then adapt the code to your own needs. Eplain tries to take care of the messy details of such things, while still leaving the detailed appearance of the output up to you.

Finally, numerous definitions turned out to be useful as Eplain was developed. They are also documented in this manual, on the chance that people writing other macros will be able to use them.

You can send bug reports or suggestions to `tex-eplain@tug.org`. The current version number of Eplain is defined as the macro `\fmtversion` at the end of the source file `eplain.tex`. When corresponding, please refer to it.

To get on this mailing list yourself, email `tex-eplain-request@tug.org` with a message whose body contains a line

```
subscribe you@your.preferred.address
```

or visit `http://lists.tug.org/tex-eplain`.

David Walden had reported his experience with Eplain as a new user. The article is available online at `https://tug.org/pracjourn/2005-4/walden`. An introductory article (written for *TUGboat*) is also available online at `https://tug.org/eplain/misc/tb84katsi.pdf`.

# 2 Installation

Your TEX installation should already contain a version of Eplain (`eplain.tex`) in its main `texmf` tree (usually under `/usr/share/texmf/tex/eplain/` on Unix systems). To install a newer version of Eplain, put the new `eplain.tex` (included in Eplain distributions) in the `tex/eplain/` subdirectory of your local `texmf` tree. The newer version you install in the local tree should override the older one in the main tree.

The location of the local `texmf` tree obviously depends on your operating system and TEX installation. On Unix systems the usual location is `/usr/local/share/texmf/`. If you don't have write permissions for `/usr/local/share/texmf/`, many installations read the `texmf` tree in the user's home directory; `eplain.tex` then should go under `~/texmf/tex/eplain/`. For more information about TEX directory structure, please see `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=tds`.

If you prefer to install `eplain.tex` in a non-standard place, set an environment variable (`TEXINPUTS` for the Web2C port of TEX to Unix) to tell TEX how to find it.

If you want, you can also create a format (`.fmt`) file for Eplain, which will eliminate the time spent reading the macro source file with `\input`. You do this by issuing a sequence of Unix commands something like this:

```
prompt$ touch eplain.aux
prompt$ initex
This is TeX, ...
**&plain eplain
(eplain.tex)
*\dump
... messages ...
```

You must make sure that `eplain.aux` exists *before* you run `initex`; otherwise, warning messages about undefined labels will never be issued.

You then have to install the resulting `eplain.fmt` in your local `texmf` tree or set an environment variable to tell TEX how to find it. For the Web2C port of TEX to Unix, format files are usually installed under `/usr/local/share/texmf/web2c/` or `/var/lib/texmf/web2c/`; the environment variable is `TEXFORMATS`.

# 3 Invoking Eplain

The simplest way to use Eplain is simply to put:

```
\input eplain
```

at the beginning of your input file. The macro file is small enough that reading it does not take an unbearably long time—at least on contemporary machines.

In addition, if a format (`.fmt`) file has been created for Eplain (see the previous section), you can eliminate the time spent reading the macro source file. You do this by responding `&eplain` to TEX's '`**`' prompt. For example:

```
initex
This is TeX, ...
**&eplain myfile
```

Depending on the implementation of TEX which you are using, you might also be able to invoke TEX as `eplain` and have the format file automatically read.

If you write something which you will be distributing to others, you won't know if the Eplain format will be loaded already. If it is, then doing `\input eplain` will waste time; if it isn't, then you must load it. To solve this, Eplain defines the control sequence `\eplain` to be the letter `t` (a convention borrowed from Lisp; it doesn't actually matter what the definition is, only that the definition exists). Therefore, you can do the following:

```
\ifx\eplain\undefined \input eplain \fi
```

where `\undefined` must never acquire a definition.

Eplain consists of several source files:

| | |
|---|---|
| `xeplain.tex` | most of the macros; |
| `arrow.tex` | commutative diagram macros (see Chapter 6 [Arrow theoretic diagrams], page 62), written by Steven Smith; |
| `btxmac.tex` | bibliography-related macros (see Section 4.3 [Citations], page 5); |
| `ifpdf.sty` | sets the switch `\ifpdf`, which can be used to detect pdfTEX in PDF mode (see Section 4.22 [Checking for PDF output], page 34), written by Heiko Oberdiek; |
| `path.sty` | macro for allowing line breaks at punctuation characters within long pathnames, electronic mail addresses, etc., (see Section 4.19 [Paths], page 32), written by Philip Taylor; |
| `texnames.sty` | abbreviations for various TEX-related names (see Section 4.20 [Logos], page 33), edited by Nelson Beebe. |

The file `eplain.tex` is all of these files merged together, with comments removed. The original sources can be found in Eplain source zip archive in your TeX distribution, on CTAN or on Eplain's home page at `https://tug.org/eplain`.

All of these files except `xeplain.tex` can be input individually, if all you want are the definitions in that file.

Also, since the bibliography macros are fairly extensive, you might not want to load them, to conserve TEX's memory. Therefore, if the control sequence `\nobibtex` is defined, then the bibliography definitions are skipped. You must set `\nobibtex` before `eplain.tex` is read, naturally. For example, you could start your input file like this:

```
\let\nobibtex = t
\input eplain
```

By default, `\nobibtex` is undefined, and so the bibliography definitions *are* made.

Likewise, define `\noarrow` if you don't want to include the commutative diagram macros from `arrow.tex`, perhaps because you already have conflicting ones.

If you don't want to read or write an `aux` file at all, for any kind of cross-referencing, define `\noauxfile` before reading `eplain.tex`. This also turns off all warnings about undefined labels.

Eplain conflicts with AMSTEX (to be precise, with `amsppt.sty`): the macros `\cite` and `\ref` are defined by both.

If you want to use AMSTEX's `\cite`, the solution is to define `\nobibtex` before reading Eplain, as described above.

If you have `amsppt.sty` loaded and use `\ref`, Eplain writes a warning on your terminal. If you want to use the AMSTEX `\ref`, do `\let\ref = \amsref` after reading Eplain. To avoid the warning, do `\let\ref = \eplainref` after reading Eplain and before using `\ref`.

Sometimes you may need to run TEX more then once on your `.tex` file in order to produce and typeset indexes, resolve undefined cross-references and/or citations. The shell script `texi2dvi` from the Texinfo documentation system (see `http://www.gnu.org/software/texinfo`) can automate this process: it runs BibTEX, MakeIndex and TEX as many times as needed to complete the compilation process. You will need to set the `LATEX` environment variable to '`tex`'. For example, in a Bourne-compatible shell, the following command will do all the work:

```
prompt$ LATEX=tex texi2dvi file.tex
```

(Despite the name, `texi2dvi` can also produce `.pdf` files; just set '`LATEX=pdftex`'.) See the output from the command `texi2dvi --help` for invoking information and a full list of options.

# 4 User definitions

This chapter describes definitions that are meant to be used directly in a document. When appropriate, ways to change the default formatting are described in subsections.

## 4.1 Diagnostics

Plain TEX provides the `\tracingall` command, to turn on the maximum amount of tracing possible in TEX. The (usually voluminous) output from `\tracingall` goes both on the terminal and into the transcript file. It is sometimes easier to have the output go only to the transcript file, so you can peruse it at your leisure and not obscure other output to the terminal. So, Eplain provides the command `\loggingall`. (For some reason, this command is available in Metafont, but not in TEX.)

It is also sometimes useful to see the complete contents of boxes. `\tracingboxes` does this. (It doesn't affect whether or not the contents are shown on the terminal.)

You can turn off all tracing with `\tracingoff`.

You can also turn logging on and off globally, so you don't have to worry about whether or not you're inside a group at the time of command. These variants are named `\gloggingall` and `\gtracingall`.

Finally, if you write your own help messages (see `\newhelp` in *The TEXbook*), you want a convenient way to break lines in them. This is what TEX's `\newlinechar` parameter is for; however, plain TEX doesn't set `\newlinechar`. Therefore, Eplain defines it to be the character `^^J`.

For example, one of Eplain's own error messages is defined as follows:

```
\newhelp\envhelp{Perhaps you forgot to end the previous^^J%
   environment? I'm finishing off the current group,^^J%
   hoping that will fix it.}%
```

## 4.2 Rules

The default dimensions of rules are defined in chapter 21 of the *The TEXbook*. To sum up what is given there, the "thickness" of rules is 0.4pt by default. Eplain defines three parameters that let you change this dimension: `\hruledefaultheight`, `\hruledefaultdepth`, and `\vruledefaultwidth`. By default, they are defined as *The TEXbook* describes.

But it would be wrong to redefine `\hrule` and `\vrule`. For one thing, some macros in plain TEX depend on the default dimensions being used; for another, rules are used quite heavily, and the performance impact of making it a macro can be noticeable. Therefore, to take advantage of the default rule parameters, you must use `\ehrule` and `\evrule`.

## 4.3 Citations

Bibliographies are part of almost every technical document. To handle them conveniently, you need two things: a program to do the tedious formatting, and a way to cite references by labels, rather than by numbers. The BibTEX program, written by Oren Patashnik, takes care of the first item; the citation commands in LATEX, written to be used with BibTEX, take care of the second. Therefore, Eplain adopts the use of BibTEX, and virtually the same interface as LATEX.

The general idea is that you put citation commands in the text of your document, and commands saying where the bibliography data is. When you run TEX, these commands produce output on the file with the same root name as your document (by default) and the extension `.aux`. BibTEX reads this file. You should put the bibliography data in a file or files with the extension `.bib`. BibTEX writes out a file with the same root name as your document and extension `.bbl`. Eplain reads this file the next time you run your document through TEX. (It takes multiple passes to get everything straight, because usually after seeing your bibliography typeset, you want to make changes in the `.bib` file, which means you have to run BibTEX again, which means you have to run TEX again. . . ) An annotated example of the whole process is given below.

If your document has more than one bibliography—for example, if it is a collection of papers—you can tell Eplain to use a different root name for the `.bbl` file by defining the control sequence `\bblfilebasename`. The default definition is simply `\jobname`.

On the other hand, if your document's bibliography is very simple, you may prefer to create the `.bbl` file yourself, by hand, instead of using BibTEX. An annotated example of this approach is also given below.

See the document *BibTEXing* (whose text is in the file `btxdoc.tex`, which should be in the Eplain distribution you got) for information on how to write your `.bib` files. Both the BibTEX and the Eplain distributions contain several examples, also.

The `\cite` command produces a citation in the text of your document. The exact printed form the citation will take is under your control (see Section 4.3.1 [Formatting citations], page 7). `\cite` takes one required argument, a comma-separated list of cross-reference labels (see Section 4.9 [Cross-references], page 15, for exactly what characters are allowed in such labels). Warning: spaces in this list are taken as part of the following label name, which is probably not what you expect. The `\cite` command also produces a command in the `.aux` file that tells BibTEX to retrieve the given reference(s) from the `.bib` file. `\cite` also takes one optional argument, which you specify within square brackets, as in LATEX. This text is simply typeset after the citations. (See the example below.)

Eplain can create hypertext links for citations pointing to the relevant bibliography entries (see Section 5.3.3 [Citation hyperlinks], page 43).

Another command, `\nocite`, puts the given reference(s) into the bibliography, but produces nothing in the text.

The `\bibliography` command is next. It serves two purposes: producing the typeset bibliography, and telling BibTEX the root names of the `.bib` files. Therefore, the argument to `\bibliography` is a comma separated list of the `.bib` files (without the '`.bib`'). Again, spaces in this list are significant.

You tell BibTEX the particular style in which you want your bibliography typeset with one more command: `\bibliographystyle`. The argument to this is a single filename *style*, which tells BibTEX to look for a file *style*`.bst`. See the document *Designing BibTEX styles* (whose text is in the `btxhak.tex`) for information on how to write your own styles.

Eplain automatically reads the citations from the `.aux` file when your job starts.

If you don't want to see the messages about undefined citations, you can say `\xrefwarningfalse` before making any citations. Eplain automatically does this if the `.aux` file does not exist. You can restore the default by saying `\xrefwarningtrue`.

Here is a TEX input file that illustrates the various commands.

```
\input eplain                   % Reads the .aux file.
Two citations to Knuthian works:
  \cite[note]{surreal,concrete-math}.
\beginsection{References.}\par   % Title for the bibliography.
\bibliography{knuth}            % Use knuth.bib for the labels.
\bibliographystyle{plain}       % Number the references.
\end                            % End of the document.
```

If we suppose that this file was named `citex.tex` and that the bibliography data is in `knuth.bib` (as the above `\bibliography` command says), the following commands do what's required. ('`$` ' represents the shell prompt.)

```
$ tex citex     (produces undefined citation messages)
$ bibtex citex  (read knuth.bib and citex.aux, write citex.bbl)
$ tex citex     (read citex.bbl, still have undefined citations)
$ tex citex     (one more time, to resolve the references)
```

The `texi2dvi` program can help you automate this process (see Chapter 3 [Invoking Eplain], page 3).

For simple documents you might choose to write the `.bbl` file yourself, instead of running BibTEX. For this scenario, the following commands should suffice:

```
$ tex citex     (read citex.bbl, produces undefined citation messages)
$ tex citex     (one more time, to resolve the references)
```

The output looks something like (because we used the `plain` bibliography style):

> Two citations to Knuthian works: [2,1 note].
>
> **References**
>
> [1] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, Massachusetts, 1989.
>
> [2] Donald E. Knuth. *Surreal Numbers*. Addison-Wesley, Reading, Massachusetts, 1974.

See the BibTEX documentation for information on how to write the bibliography databases, and the bibliography styles that are available. (If you want your references printed with names, as in [Knu74], instead of numbered, the bibliography style is `alpha`.)

### 4.3.1 Formatting citations

You may wish to change Eplain's formatting of citations; i.e., the result of your `\cite` commands. By default, the citation labels are printed one after another, separated by commas and enclosed in brackets, using the main text font. Some formats require other styles, such as superscripted labels. You can accommodate such formats by redefining the following macros.

`\printcitestart`
`\printcitefinish`

Eplain expands these macros at the beginning and end of the list of citations for each `\cite` command. By default, they produce a '`[`' and '`]`', respectively.

`\printbetweencitations`

If a `\cite` command has multiple citations, as in `\cite{acp,texbook}`, Eplain expands this macro in between each pair of citations. By default, it produces a comma followed by a space.

`\printcitenote`

This macro takes one argument, which is the optional note to the `\cite` command. If the `\cite` command had no note, this macro isn't used. Otherwise, it should print the note. By default, the note is preceded with a comma and a space.

Here is an example, showing you could produce citations as superscripted labels, with the optional notes in parentheses.

```
\def\printcitestart{\unskip $^\bgroup}
\def\printbetweencitations{,}
\def\printcitefinish{\egroup$}
\def\printcitenote#1{\hbox{\sevenrm\space (#1)}}
```

### 4.3.2 Formatting bibliographies

You may wish to change Eplain's formatting of the bibliography, especially with respect to the fonts that are used. Therefore, Eplain provides the following control sequences:

`\biblabelwidth`

This control sequence represents a `\dimen` register, and its value is the width of the widest label in the bibliography. Although it is unlikely you will ever want to redefine it, you might want to use it if you redefine `\biblabelprint`, below.

`\biblabelprint`

This macro takes one argument, the label to print. By default, the label is put in a box of width `\biblabelwidth`, and is followed by an enspace. When you want to change the spacing around the labels, this is the right macro to redefine.

`\biblabelcontents`

This macro also takes one argument, the label to print. By default, the label is printed using the font `\bblrm` (below), and enclosed in brackets. When you want to change the appearance of the label, but not the spacing around it, this is the right macro to redefine.

`\biblabelprecontents`
`\biblabelpostcontents`

Macros expanded before and after `\biblabelcontents`, respectively. For example, to get left-justified numeric labels (they are right-justified by default):

```
\def\biblabelprecontents{\relax}
\def\biblabelpostcontents{\hss}
```

`\bblrm` The default font used for printing the bibliography.

`\bblem` The font used for printing the titles and other "emphasized" material.

`\bblemph` Typesets its argument using `\bblem`, then inserts an italic correction.

`\bblsc` In some styles, authors' names are printed in a caps-and-small-caps font. In those cases, this font is used.

`\bblnewblock`
This is invoked between each of the parts of a bibliography entry. The default is to leave some extra space between the parts; you could redefine it to start each part on a new line (for example). A part is simply a main element of the entry; for example, the author is a part. (It was LaTeX that introduced the (misleading, as far as I am concerned) term 'block' for this.)

`\biblabelextraspace`
Bibliography entries are typeset with a hanging indentation of `\biblabelwidth` plus this. The default is `.5em`, where the em width is taken from the `\bblrm` font. If you want to change this, you should do it inside `\bblhook`.

`\bblhook` This is expanded before reading the `.bbl` file. By default, it does nothing. You could, for example, define it to set the bibliography fonts, or produce the heading for the references. Two spacing parameters must be changed inside `\bblhook`: `\parskip`, which produces extra space between the items; and `\biblabelextraspace`, which is described above. (By the way, `\hookappend` won't work with `\bblhook`, despite the names. Just use `\def`.)

If you are desperate, of course you can also hand-edit the `.bbl` file that BibTeX produces to do anything you wish.

### 4.3.3 Commands from LaTeX

Because of the historical connection between BibTeX and LaTeX, in practice many bibliography styles and bibliographies use LaTeX commmands that are not part of bibliography handling, per se.

To support this, `btxmac.tex` (and thus Eplain) define the following. In all cases, an existing definition (e.g., from `miniltx.tex`, see Section 4.23 [Loading LaTeX packages], page 34) will not be overwritten. Here is the list:

`\newcommand`
`\renewcommand`
`\providecommand`
Defining new commands in various ways. The Eplain versions do not support the *-form of these; use `miniltx` for that.

`\em`
`\emph`
`\sc`
`\textbf` Selecting fonts.

`\mbox` A horizontal box.

`\newblock`
Starts elements of a bibliography entry.

For full information about these, see the LaTeX manual and sources.

## 4.4 Displays

By default, TEX centers displayed material. (Displayed material is just whatever you put between `$$`'s—it's not necessarily mathematics.) Many layouts would be better served if the displayed material was left-justified. Therefore, Eplain provides the command `\leftdisplays`, which indents displayed material by `\parindent` plus `\leftskip`, plus `\leftdisplayindent`.

You can go back to centering displays with `\centereddisplays`. (It is usually poor typography to have both centered and left-justified displays in a single publication, though.)

`\leftdisplays` also changes the plain TEX commands that deal with alignments inside math displays, `\displaylines`, `\eqalignno`, and `\leqalignno`, to produce left-justified text. You can still override this formatting by inserting `\hfill` glue, as explained in *The TEXbook*.

Eplain defines `\eqnum` and `\eqalignnum` which can be set up to produce either left-aligned or right-aligned equation numbers. `\lefteqnumbers` (`\righteqnumbers`) will define `\eqnum` to expand to `\eqno` (`\leqno`), and `\eqalignnum` to expand to `\eqalignno` (`\leqalignno`). Default is `\righteqnumbers` (right-aligned equation numbers).

### 4.4.1 Formatting displays

If you want some other kind of formatting, you can write a definition of your own, analogous to `\leftdisplays`. You need only make sure that `\leftdisplaysetup` is called at the beginning of every display (presumably by invoking it in TEX's `\everydisplay` parameter).

`\leftdisplays` expands the old value of `\everydisplay` before calling `\leftdisplaysetup`, so that any changes you have made to it won't be lost. That old token list as available as the value of the token register `\previouseverydisplay`.

## 4.5 Time of day

TEX provides the day, month, and year as numeric quantities (unless your TEX implementation is woefully deficient). Eplain provides some control sequences to make them a little more friendly to humans.

`\monthname` produces the name of the current month, abbreviated to three letters.

`\fullmonthname` produces the name of the current month, unabbreviated (in English).

`\timestring` produces the current time, as in '1:14 p.m.'

`\timestamp` produces the current date and time, as in '23 Apr 64 1:14 p.m.'. (Except the spacing is slightly different.)

`\today` produces the current date, as in '23 April 1964'.

## 4.6 Lists

Many documents require lists of items, either numbered or simply enumerated. Plain TEX defines one macro to help with creating lists, `\item`, but that is insufficient in many cases. Therefore, Eplain provides two pairs of commands:

`\numberedlist ... \endnumberedlist`
`\orderedlist ... \endorderedlist`

These commands (they are synonyms) produce a list with the items numbered sequentially, starting from one. A nested `\numberedlist` labels the items with lowercase letters, starting with 'a'. Another nested `\numberedlist` labels the items with roman numerals. Yet more deeply nested numbered lists label items with '`*`'.

`\unorderedlist ... \endunorderedlist`

This produces a list with the items labelled with small black boxes ("square bullets"). A nested `\unorderedlist` labels items with em-dashes. Doubly (and deeper) nested unordered lists label items with '*'s.

The two kinds of lists can be nested within each other, as well.

In both kinds of lists, you begin an item with `\li`. An item may continue for several paragraphs. Each item starts a paragraph.

You can give `\li` an optional argument, a cross-reference label. It's defined to be the "marker" for the current item. This is useful if the list items are numbered. You can produce the value of the label with `\xrefn`. See Section 4.9 [Cross-references], page 15.

Eplain can create hypertext links for the markers produced by `\xrefn` pointing to the relevant list item (see Section 5.3.4 [List hyperlinks], page 43).

You can also say `\listcompact` right after `\numberedlist` or `\unorderedlist`. The items in the list will then not have any extra space between them (see Section 4.6.1 [Formatting lists], page 11). You might want to do this if the items in this particular list are short.

Here is an example:

```
\numberedlist\listcompact
\li The first item.
\li The second item.

The second paragraph of the second item.
\endnumberedlist
```

### 4.6.1 Formatting lists

Several registers define the spacing associated with lists. It is likely that their default values won't suit your particular layout.

`\abovelistskipamount, \belowlistskipamount`

The vertical glue inserted before and after every list, respectively.

`\interitemskipamount`

The vertical glue inserted before each item except the first. `\listcompact` resets this to zero, as mentioned above.

`\listleftindent, \listrightindent`

`\listrightindent` is the amount of space by which the list is indented on the right; i.e., it is added to `\rightskip`. `\listleftindent` is the amount of space, *relative to* `\parindent`, by which the list is indented on the left. Why treat the two parameters differently? Because (a) it is more useful to make the list indentation depend on the paragraph indentation; (b) footnotes aren't formatted right if `\parindent` is reset to zero.

The three vertical glues are inserted by macros, and preceded by penalties: `\abovelistskip` does `\vpenalty\abovelistpenalty` and then `\vskip\abovelistskip`. `\belowlistskip` and `\interitemskip` are analogous.

In addition, the macro `\listmarkerspace` is called to separate the item label from the item text. This is set to `\enspace` by default.

If you want to change the labels on the items, you can redefine these macros: `\numberedmarker` or `\unorderedmarker`. The following registers might be useful if you do:

`\numberedlistdepth, \unorderedlistdepth`

These keep track of the depth of nesting of the two kinds of lists.

`\itemnumber, \itemletter`

These keep track of the number of items that have been seen in the current numbered list. They are both integer registers. The difference is that `\itemnumber` starts at one, and `\itemletter` starts at 97, i.e., lowercase 'a'.

You can also redefine the control sequences that are used internally, if you want to do something radically different: `\beginlist` is invoked to begin both kinds of lists; `\printitem` is invoked to print the label (and space following the label) for each item; and `\endlist` is invoked to end both kinds of lists.

## 4.7 Verbatim listing

It is sometimes useful to include a file verbatim in your document; for example, part of a computer program. The `\listing` command is given one argument, a filename, and produces the contents of that file in your document. `\listing` expands `\listingfont` to set the current font. The default value of `\listingfont` is `\tt`.

You can take arbitrary actions before reading the file by defining the macro `\setuplistinghook`. This is expanded just before the file is input.

If you want to have line numbers on the output, you can say `\let\setuplistinghook = \linenumberedlisting`. The line numbers are stored in the count register `\lineno` while the file is being read. You can redefine the macro `\printlistinglineno` to change how they are printed.

Normally, the `\listing` command will add a final empty line at the end of the output, even if the file does not end in a newline. To suppress this final line, you can say `\let\setuplistinghook = \nolastlinelisting`. This also works with line numbers (say `\def\setuplistinghook{\linenumberedlisting \nolastlinelisting}`), but only if `\printlistinglineno` consists exclusively of boxes at the top level (i.e., any `\kern`s or glue should be wrapped up in a box).

You can use the form feed control character (ASCII code 12, typed as *`CTRL-L`*) in the file to force a page break in the output.

You can produce in-line verbatim text in your document with `\verbatim`. End the text with `|endverbatim`. If you need a '`|`' in the text, double it. If the first character of the verbatim text is a space, use `| `. (`| ` will work elsewhere in the argument, too, but isn't necessary.)

For example:

```
\verbatim| ||\#%&!|endverbatim
```

produces ` |\#%&!`.

Line breaks and spaces in the verbatim text are preserved.

You can change the verbatim escape character from the default '`|`' with `\verbatimescapechar` *`char`*; for example, this changes it to '`@`'.

```
\verbatimescapechar \@
```

The backslash is not necessary in some cases, but is in others, depending on the catcode of the character. The argument to `\verbatimescapechar` is used as `\catcode `*`‘char`*, so the exact rules follow that for `\catcode`.

To reset the category code of all special characters to 12 ("other"), `\verbatim` uses `\uncatcodespecials` (see Section 7.1 [Category codes], page 69). If you make additional characters "special", you should extend `\dospecials` to include those characters, lest they be given special treatment inside verbatim environments. For example,

```
\catcode‘\A=\active
% Try commenting out the following line.
\expandafter\def\expandafter\dospecials\expandafter{\dospecials\do\A}
\verbatimA#$%_^|endverbatim
```

Because `\verbatim` must change the category code of special characters, calling inside a macro definition of your own does not work properly. For example:

```
\def\mymacro{\verbatim &#%|endverbatim}% Doesn't work!
```

To accomplish this, you must change the category codes yourself before making the macro definition. Perhaps `\uncatcodespecials` will help you (see Section 7.1 [Category codes], page 69).

## 4.8 Contents

Producing a table of contents that is both useful and aesthetic is one of the most difficult design problems in any work. Naturally, Eplain does not pretend to solve the design problem. Collecting the raw data for a table of contents, however, is much the same across documents. Eplain uses an auxiliary file with extension `.toc` (and the same root name as your document) to save the information.

### 4.8.1 Writing the `.toc` file

To write an entry for the table of contents, you say `\writetocentry{`*`part`*`}{`*`text`*`}`, where *part* is the type of part this entry is, e.g., '`chapter`', and *text* is the text of the title. `\writetocentry` puts an entry into the `.toc` file that looks like `\toc`*`part`*`entry{`*`text`*`}{`*`page number`*`}` (unless *part* is an integer, see below). The *text* is written unexpanded.

A related command, `\writenumberedtocentry`, takes one additional argument, the first token of which is expanded at the point of the `\writenumberedtocentry`, but the rest of the argument is not expanded. The usual application is when the parts of the document are numbered. On the other hand, the one-level expansion allows you to use the argument for other things as well (author's names in a proceedings, say), and not have accents or other control sequences expanded. The downside is that if you *want* full expansion of the third argument, you don't get it—you must expand it yourself, before you call `\writenumberedtocentry`.

For example:

```
\writenumberedtocentry{chapter}{A $\sin$ wave}{\the\chapno}
\writetocentry{section}{A section title}
```

Supposing `\the\chapno` expanded to '`3`' and that the `\write`'s occurred on pages eight and nine, respectively, the above writes the following to the `.toc` file:

```
\tocchapterentry{A $\sin$ wave}{3}{8}
\tocsectionentry{A section title}{9}
```

A variation on `\writenumberedtocentry` is `\writenumberedtocline`, differing only in the order of the parameters it takes and writes for the `\toc`*part*`entry` control sequences. To continue the previous example:

```
\writenumberedtocline{chapter}{\the\chapno}{A $\sin$ wave}
```

writes the following to the `.toc` file:

```
\tocchapterentry{3}{A $\sin$ wave}{8}
```

Such ordering of the parameters allows the `\toc`*part*`entry` macros to typeset the text of the entry without actually reading it as an argument. This is required for entries which need to change character catcodes, e.g., to produce verbatim text (see Section 4.7 [Verbatim listing], page 12).

Each of `\writetocentry`, `\writenumberedtocentry` and `\writenumberedtocline` processes a numeric *part* argument specially. If you pass *part* expanding to an integer, these macros write into the `.toc` file an entry that starts with `\tocentry{`*part*`}`. Thus, you can define a single `\tocentry` macro which formats all entries for a table of contents. To continue the previous examples:

```
\writenumberedtocentry{1}{A $\sin$ wave}{\the\chapno}
\writenumberedtocline{1}{\the\chapno}{A $\sin$ wave}
\writetocentry{2}{A section title}
```

writes the following to the .toc file:

```
\tocentry{1}{A $\sin$ wave}{3}{8}
\tocentry{1}{3}{A $\sin$ wave}{8}
\tocentry{2}{A section title}{9}
```

### 4.8.2 Reading the `.toc` file

You read the `.toc` file with the command `\readtocfile`. Naturally, whatever `\toc... entry` commands that were written to the file must be defined when `\readtocfile` is invoked. Eplain has minimal definitions for `\tocchapterentry`, `\tocsectionentry`, and `\tocsubsectionentry`, just to prevent undefined control sequence errors in common cases. They aren't suitable for anything but preliminary proofs.

Each of `\writetocentry`, `\writenumberedtocentry` and `\writenumberedtocline` opens the `.toc` file for writing, thereby deleting the information from the previous run.

You should therefore arrange that `\readtocfile` be called *before* the first call to a `\writetoc...` macro. `\readtocfile` does not itself delete the information from the `.toc` file, so that you can call it several times, e.g., to create both a short and normal table of contents. (To produce this in particular, define `\tocsectionentry` to produce nothing while you are reading `.toc` file for a short table of contents (see Section 7.4 [Macro arguments], page 71).)

On the other hand, if you don't want to rewrite the `.toc` file at all, perhaps because you are only running TEX on part of your manuscript, you can set `\rewritetocfilefalse`.

### 4.8.3 Changing the `.toc` file's root name

By default, the `.toc` file has the root `\jobname`. If your document has more than one contents—for example, if it is a collection of papers, some of which have their own contents—you can tell Eplain to use a different root name by defining the control sequence `\tocfilebasename`.

Note that `\writetocentry`, `\writenumberedtocentry` and `\writenumberedtocline` will open the contents file for writing only at the first call, using the value of `\tocfilebasename` at that time. Changing the value of `\tocfilebasename` afterwards will not affect which file gets *written*, although it will affect which file gets *read* by `\readcontentsfile`. In case you need to write several contents files from a single TEX job, use `\definecontentsfile` (see Section 4.8.4 [Alternative contents files], page 15).

### 4.8.4 Alternative contents files

In addition to the usual table of contents, you may want to have a list of figures, list of tables, or other such contents-like list. You can do this with `\definecontentsfile{`*abbrev*`}`. All of the above commands are actually a special case that Eplain predefines with

```
\definecontentsfile{toc}
```

The *abbrev* is used both for the file extension and in the control sequence names.

## 4.9 Cross-references

It is often useful to refer the reader to other parts of your document; but putting literal page, section, equation, or whatever numbers in the text is certainly a bad thing.

Eplain therefore provides commands for symbolic cross-references. It uses an auxiliary file with extension `.aux` (and the same root name as your document) to keep track of the information. Therefore, it takes two passes to get the cross-references right—one to write them out, and one to read them in. Eplain automatically reads the `.aux` file at the first reference; after reading it, Eplain reopens it for writing.

You can control whether or not Eplain warns you about undefined labels. See Section 4.3 [Citations], page 5.

Labels in Eplain's cross-reference commands can use characters of category code eleven (letter), twelve (other), ten (space), three (math shift), four (alignment tab), seven (superscript), or eight (subscript). For example, '`(a1 $&^_`' is a valid label (assuming the category codes of plain TEX), but '`%#\{`' has no valid characters.

You can also do symbolic cross-references for bibliographic citations and list items. See Section 4.3 [Citations], page 5, and Section 4.6 [Lists], page 10.

Eplain can create hypertext links for the cross-references (see Section 5.3.5 [Cross-reference hyperlinks], page 44).

### 4.9.1 Defining generic references

Eplain provides the command `\definexref` for general cross-references. It takes three arguments: the name of the label (see section above for valid label names), the value of the label (which can be anything), and the "class" of the reference—whether it's a section, or theorem, or what. For example:

```
\definexref{sec-intro}{3.1}{section}
```

Of course, the label value is usually generated by another macro using TEX count registers or some such.

`\definexref` doesn't actually define *label*; instead, it writes out the definition to the `.aux` file, where Eplain will read it on the next TEX run.

The *class* argument is used by the `\ref` and `\refs` commands. See the next section.

### 4.9.2 Using generic references

To retrieve the value of the label defined via `\definexref` (see the previous section), Eplain provides the following macros:

`\refn{`*`label`*`}`
`\xrefn{`*`label`*`}`

> `\refn` and `\xrefn` (they are synonyms) produce the bare definition of *label*. If *label* isn't defined, issue a warning, and produce *label* itself instead, in typewriter. (The warning isn't given if `\xrefwarningfalse`.)

`\ref{`*`label`*`}`

> Given the class *c* for *label* (see the description of `\definexref` in the previous section), expand the control sequence `\`*`c`*` word` (if it's defined) followed by a tie. Then call `\refn` on *label*. (Example below.)

`\refs{`*`label`*`}`

> Like `\ref`, but append the letter '`s`' to the `\...word`.

The purpose of the `\...word` macro is to produce the word 'Section' or 'Figure' or whatever that usually precedes the actual reference number.

Here is an example:

```
\def\sectionword{Section}
\definexref{sec-intro}{3.1}{section}
\definexref{sec-next}{3.2}{section}
See \refs{sec-intro} and \refn{sec-next} ...
```

This produces 'See Sections 3.1 and 3.2 . . .'

## 4.10 Page references

Eplain provides two commands for handling references to page numbers, one for definition and one for use.

`\xrdef{label}`
: Define *label* to be the current page number. This produces no printed output, and ignores following spaces.

`\xref{label}`
: Produce the text 'p. *pageno*', which is the usual form for cross-references. The *pageno* is actually *label*'s definition; if *label* isn't defined, the text of the label itself is printed. The 'p. ' prefix is defined by `\xrefpageword`. Its default definition is `p.\thinspace`.

Eplain can create hypertext links for the page references (see Section 5.3.6 [Page reference hyperlinks], page 44).

## 4.11 Equation references

Instead of referring to pages, it's most useful if equation labels refer to equation numbers. Therefore, Eplain reserves a `\count` register, `\eqnumber`, for the current equation number, and increments it at each numbered equation.

Here are the commands to define equation labels and then refer to them:

`\eqdef{label}`
: This defines *label* to be the current value of `\eqnumber`, and, if the current context is not inner, then produces a `\eqnum` command (see Section 4.4 [Displays], page 10). (The condition makes it possible to use `\eqdef` in an `\eqalignno` construction, for example.) The text of the equation number is produced using `\eqprint`. See Section 4.11.1 [Formatting equation references], page 18.

: If *label* is empty, you still get an equation number (although naturally you can't reliably refer to it). This is useful if you want to put numbers on all equations in your document, and you don't want to think up unique labels.

: To refer to the last equation with the empty label, you use the empty label in one of the equation reference macros (see below). This can be handy when you want to refer to an equation shortly after its definition, say, in the sentence following the displayed equation, and do not intend to refer to the equation later. But use this trick with extreme caution: if later you change the text and insert another empty definition between the original definition and the reference, the reference will start to refer to the new empty-labeled equation.

`\eqdefn{label}`
: This is like `\eqdef`, except it always omits the `\eqnum` command. It can therefore be used in places where `\eqdef` can't; for example, in a non-displayed equation. The text of the equation number is not produced, so you can also use it in the (admittedly unusual) circumstance when you want to define an equation label but not print that label.

`\eqref{label}`
: This produces a formatted reference to *label*. If *label* is undefined (perhaps because it is a forward reference), it just produces the text of the label itself. Otherwise, it calls `\eqprint`.

`\eqrefn{label}`
: This produces the cross-reference text for *label*. That is, it is like `\eqref`, except it doesn't call `\eqprint`.

Equation labels can contain the same characters that are valid in general cross-references.

Eplain can create hypertext links for the equation references (see Section 5.3.7 [Equation reference hyperlinks], page 44).

### 4.11.1 Formatting equation references

Both defining an equation label and referring to it should usually produce output. This output is produced with the `\eqprint` macro, which takes one argument, the equation number being defined or referred to. By default, this just produces '`(number)`', where *number* is the equation number. To produce the equation number in a different font, or with different surrounding symbols, or whatever, you can redefine `\eqprint`. For example, the following definition would print all equation numbers in italics. (The extra braces define a group, to keep the font change from affecting surrounding text.)

```
\def\eqprint#1{{\it (#1)}}
```

In addition to changing the formatting of equation numbers, you might want to add more structure to the equation number; for example, you might want to include the chapter number, to get equation numbers like '(1.2)'. To achieve this, you redefine `\eqconstruct`. For example:

```
\def\eqconstruct#1{\the\chapternumber.#1}
```

(If you are keeping the chapter number in a count register named `\chapternumber`, naturally.)

The reason for having both `\eqconstruct` and `\eqprint` may not be immediately apparent. The difference is that `\eqconstruct` affects the text that cross-reference label is defined to be, while `\eqprint` affects only what is typeset on the page. The example just below might help.

Usually, you want equation labels to refer to equation numbers. But sometimes you might want a more complicated text. For example, you might have an equation '(1)', and then have a variation several pages later which you want to refer to as '(1*)'.

Therefore, Eplain allows you to give an optional argument (i.e., arbitrary text in square brackets) before the cross-reference label to `\eqdef`. Then, when you refer to the equation, that text is produced. Here's how to get the example just mentioned:

```
$$...\eqdef{a-eq}$$
...
$$...\eqdef[\eqrefn{a-eq}*]{a-eq-var}$$
In \eqref{a-eq-var}, we expand on \eqref{a-eq}, ...
```

We use `\eqrefn` in the cross-reference text, not `\eqref`, so that `\eqprint` is called only once.

As another example, consider the following requirement: we want to include chapter number in all equation references, and additionally we want to include the part number when referencing an equation from any part other than the one where the equation appears. For example, references to the third equation in chapter 2 of part 1 should be typeset as

'(2.3)' throughout part 1, but as '(I.2.3)' in any other part. Let's assume we have the current chapter and part numbers in count registers `\chapnum` and `\partnum`, respectively.

The idea is to have `\eqconstruct` store the part number of the equation (that is, the part number *at the time of definition*), so that later `\eqprint` can compare the stored number with the current part number (that is, the part number *at the time of reference*). The complicating factor is that internally, the result of `\eqconstruct` is both expanded and written out to the `.aux` file, *and* used to typeset the equation number, so the commands that store the part number should behave correctly in both situations. This is difficult to achieve with expandable commands; therefore, to avoid expansion problems, we are going to use only TEX primitives, which are non-expandable:

```
\newcount\eqpartnum

\def\eqconstruct#1{%
  \global\eqpartnum=\the\partnum\relax
  \number\chapnum.#1%
}

\def\eqprint#1{%
  \setbox0=\hbox{#1}%
  (\ifnum\partnum=\eqpartnum \else
     \uppercase\expandafter{\romannumeral\eqpartnum}.%
   \fi
   \box0)%
}%
```

In `\eqconstruct`, besides constructing the base equation number (e.g., '1.2'), we also store the part number of the equation in the count register `\eqpartnum` (`\the\partnum` is expanded when the equation number is written to the `.aux` file, so the equation label definition in the `.aux` file will contain the actual part number). In `\eqprint`, we need to know the equation's part number before we typeset the base equation number, therefore we first put the argument in a box, thus causing `\eqpartnum` to be set.

### 4.11.2 Subequation references

Eplain also provides for one level of substructure for equations. That is, you might want to define a related group of equations with numbers like '2.1' and '2.2', and then be able to refer to the group as a whole: "... in the system of equations (2)...".

The commands to do this are `\eqsubdef` and `\eqsubdefn`. They take one *label* argument like their counterparts above, and generally behave in the same way. The difference is in how they construct the equation number: instead of using just `\eqnumber`, they also use another counter, `\subeqnumber`. This counter is advanced by one at every `\eqsubdef` or `\eqsubdefn`, and reset to zero at every `\eqdef` or `\eqdefn`.

You use `\eqref` to refer to subequations as well as main equations.

To put the two together to construct the text that the label will produce, they use a macro `\eqsubreftext`. This macros takes two arguments, the "main" equation number (which, because the equation label can be defined as arbitrary text, as described in the previous section, might be anything at all) and the "sub" equation number (which is always just a number). Eplain's default definition just puts a period between them:

```
\def\eqsubreftext#1#2{#1.#2}%
```

You can redefine `\eqsubreftext` to print however you like. For example, this definition makes the labels print as '2a', '2b', and so on.

```
\newcount\subref
\def\eqsubreftext#1#2{%
  \subref = #2           % The space stops a <number>.
  \advance\subref by 96  % 'a' is character code 97.
  #1\char\subref
}
```

Sadly, we must define a new count register, `\subref`, instead of using the scratch count register `\count255`, because '`#1`' might include other macro calls which use `\count255`.

## 4.12 Indexing

Eplain provides support for generating raw material for an index, and for typesetting a sorted index. A separate program must do the actual collection and sorting of terms, because TEX itself has no support for sorting.

Eplain can create hypertext links pointing from the index to the index terms (see Section 5.3.8 [Index hyperlinks], page 44).

Eplain's indexing commands were designed to work with the program MakeIndex (`https://ctan.org/pkg/makeindex`); MakeIndex is also commonly included in prepackaged TEX distributions. It is beyond the scope of this manual to explain how to run MakeIndex, and all of its many options.

The basic strategy for indexing works like this:

1. For a document `foo.tex`, Eplain's indexing commands (e.g., `\idx`; see the section 'Indexing terms' below) write the raw index material to `foo.idx`.
2. MakeIndex reads `foo.idx`, collects and sorts the index, and writes the result to `foo.ind`.
3. Eplain reads and typesets `foo.ind` on a subsequent run of TEX. See the section 'Typesetting an index' below.

The `texi2dvi` program can help you automate this process (see Chapter 3 [Invoking Eplain], page 3).

If your document needs more than one index, each must have its own file. Therefore, Eplain provides the command `\defineindex`, which takes an argument that is a single letter, which replaces '`i`' in the filenames and in the indexing command names described below. For example,

```
\defineindex{m}
```

defines the command `\mdx` to write to the file `foo.mdx`. Eplain simply does `\defineindex{i}` to define the default commands.

Note that MakeIndex does not use the above naming scheme for multiple indexes. Unless instructed otherwise, MakeIndex always writes its output to a file with extension `.ind`. For example, if you define an additional index with the command `\defineindex{j}`, you'll need to run MakeIndex like this:

```
$ makeindex book.jdx -o book.jnd
```

For each index defined with `\defineindex{n}`, Eplain provides a switch `\ifndx` which controls whether indexing commands write index entries to the corresponding index file. However, even when index term writing is disabled, indexing commands still do all other processing of their arguments, including typesetting of proof index terms (see Section 4.12.1.4 [Proofing index terms], page 25.

For example, if you write `\idxfalse` near the beginning of a document `foo.tex` (before the first indexing command), Eplain will not open the default index file (`foo.idx`) and the corresponding indexing commands (`\idx`, `\sidx`, etc.) will not write index entries there. This may be useful for draft compilations of a manuscript, e.g., to avoid the overhead of index file input/output.

### 4.12.1 Indexing terms

Indexing commands in Eplain come in pairs: one command that only writes the index entry to the '`.idx`' file (see above section), and one that also typesets the term being indexed. The former always starts with '`s`' (for "silent"). In either case, the name always includes '`Idx`', where *I* is the index letter, also described above. Eplain defines the index '`i`' itself, so that's what we'll use in the names below.

The silent form of the commands take a subterm as a trailing optional argument. For example, `\sidx{truth}[definition of]` on page 75 makes an index entry that will eventually be typeset (by default) as

truth
  definition of, 75

Also, the silent commands ignore trailing spaces. The non-silent ones do not.

#### 4.12.1.1 Indexing commands

Here are the commands.

- `\sidx{term}[subterm]` makes an index entry for *term*, optionally with subterm *subterm*. `\idx{term}` also produces *term* as output. Example:

  ```
  \sidx{truth}[beauty of]
  The beauty of truth is \idx{death}.
  ```

  Subterms at the second and further levels can also be specified in *subterm*, using the `\idxsubentryseparator` character to separate them. This character is by default '`!`'.

- `\sidxname{First M.}{von Last}[subterm]` makes an index entry for '`von Last, First M.`'. You can change the '`, `' by redefining `\idxnameseparator`. `\idxname{First M.}{von Last}` also produces *First M. von Last* as output. (These commands are useful special cases of `\idx` and `\sidx`.) Example:

  ```
  \sidxname{Richard}{Stark}
  \idxname{Donald}{Westlake} has written many kinds of novels, under
  almost as many names.
  ```

- `\sidxmarked\cs{term}[subterm]` makes an index entry for `term[subterm]`, but *term* will be put in the index as `\cs{term}`, but still sorted as just *term*. `\idxmarked\cs{term}` also typesets `\cs{term}`. This provides for the usual ways of changing the typesetting of index entries. Example:

  ```
  \def\article#1{``#1''}
  ```

```
\sidxmarked\article{Miss Elsa and Aunt Sophie}
Peter Drucker's \idxmarked\article{The Polanyis} is a remarkable
essay about a remarkable family.
```

- `\sidxsubmarked{term}\cs{subterm}` makes an index entry for *term*, *subterm* as usual, but also puts *subterm* in the index as `\cs{term}`. `\idxsubmarked{term}\cs{subterm}` also typesets `term \cs{subterm}`, in the unlikely event that your syntax is convoluted enough to make this useful. Example:

  ```
  \def\title#1{{\sl #1}}
  \sidxsubmarked{Anderson, Laurie}\title{Strange Angels}
  The \idxsubmarked{Anderson}\title{Carmen} is a strange twist.
  ```

The commands above rely on MakeIndex's feature for separating sorting of an index entry's from its typesetting. You can use this directly by specifying an index entry as `sort@typeset`. For example:

```
\sidx{Ap-weight@$A_\pi$-weight}
```

will sort as `Ap-weight`, but print with the proper math. The `@` here is MakeIndex's default character for this purpose. To make an index entry with an `@` in it, you have to escape it with a backslash; Eplain provides no macros for doing this.

After any index command, Eplain runs `\hookaction{afterindexterm}`. Because the index commands always add a whatsit item to the current list, you may wish to preserve a penalty or space past the new item. For example, given a conditional `\if@aftersctnhead` set true when you're at a section heading, you could do:

```
\hookaction{afterindexterm}{\if@aftersctnhead \nobreak \fi}
```

#### 4.12.1.2 Modifying index entries

All the index commands described in the previous section take an initial optional argument before the index term, which modify the index entry's meaning in various ways. You can specify only one of the following in any given command, except that `begin` and `end` can be specified together with `pagemarkup=cs` (separate them with a comma without a following space, like this: `[begin,pagemarkup=defn]`).

These work via MakeIndex's "encapsulation" feature. See Section 4.12.3 [Customizing indexing], page 26, if you're not using the default characters for the MakeIndex operators. The other optional argument (specifying a subterm) is independent of these.

Here are the possibilities:

`begin`
`end` These mark an index entry as the beginning or end of a range. The index entries must match exactly for MakeIndex to recognize them. Example:

```
\sidx[begin]{future}[Cohen, Leonard]
...
\sidx[end]{future}[Cohen, Leonard]
```

will typeset as something like

future,
  Cohen, Leonard, 65–94

`see` This marks an index entry as pointing to another; the real index term is an additional (non-optional) argument to the command. Thus you can anticipate a term readers may wish to look up, yet which you have decided not to index. Example:

```
\sidx[see]{analysis}[archetypal]{archetypal criticism}
```

becomes

analysis,
  archetypal, *see* archetypal criticism

`seealso` Similar to `see` (the previous item), but also allows for normal index entries of the referencing term. The normal index entries have to be created separately—`seealso` does *not* contribute a page number to the index entry. For example, if you have indexed a term on pages 75, 97 and 114, and then add a `seealso` entry for the term:

```
\sidx[seealso]{archetypal criticism}[elements of]{dichotomies}
```

the index will contain

archetypal criticism,
  elements of, 75, 97, 114, *see also* dichotomies

(Aside for the academically curious: The archetypal critical book I took these dichotomous examples from is Laurence Berman's *The Musical Image*, which I happened to co-design and typeset.)

`pagemarkup=`*`cs`*

This puts `\`*`cs`* before the page number in the typeset index, thus allowing you to underline definitive entries, italicize examples, and the like. You do *not* precede the control sequence *cs* with a backslash. (That just leads to expansive difficulties.) Naturally it is up to you to define the control sequences you want to use. Example:

```
\def\defn#1{{\sl #1}}
\sidx[pagemarkeup=defn]{indexing}
```

becomes something like

```
indexing, \defn{75}
```

#### 4.12.1.3 Index entries with special characters

Indexing terms with special characters can become quite cumbersome because you have to keep both TEX and MakeIndex happy at the same time. For example, while '`!`' has no special meaning for TEX, it is a subentry separator for MakeIndex, therefore you'd have to escape occurrences of literal '`!`' in index terms. Things get even more interesting with characters which are special in both TEX and MakeIndex.

This in turn has some implications for the non-silent forms of the indexing commands (see Section 4.12.1 [Indexing terms], page 21), since TEX and MakeIndex use different conventions for escaping characters. For example, this will not typeset the exclamation point correctly within the text, while it will come out right inside the index, after MakeIndex strips the quoting character ('`"`'):

```
\idx{"!}
```

This would have to be rewritten using the silent command:

```
!\sidx{"!}
```

In general, it is a good idea to eschew the non-silent commands whenever index term contains anything unusual.

To understand this keep in mind that indexing commands read the terms verbatim so that the terms can embed almost any character, and that's what gets written into the `.idx` file. The non-silent forms then typeset the term by rescanning the verbatim copy, hence for the non-silent commands the term, besides being a valid MakeIndex input, must also represent a valid TEX input. The silent commands don't have this restriction—their terms only need to become valid TEX input *after* MakeIndex processes the `.idx` file and writes the `.ind` file. This is what makes the non-silent commands less powerful and more troublesome when dealing with special characters.

Here's an example showing that terms for the silent commands can contain almost any character:

```
\sidx[see]{comments}[with %@with \verbatim %"|endverbatim]
  {commenting with \verbatim %"|endverbatim}
```

We didn't have to escape '`%`' in the sort string for MakeIndex, while we had to put it inside the verbatim environment (see Section 4.7 [Verbatim listing], page 12) in the part which MakeIndex will pass back to TEX. Also, we had to escape the '`|`' character because it is special for MakeIndex. If you have trouble understanding the reasons for the different types of escaping used, it is best to examine the `.idx` and `.ind` files resulting from processing the above input.

As was mentioned, index terms can embed "almost any character", so now we'll describe the exceptions.

The following characters are reset to their usual meanings because they are not useful verbatim: multiple consequent spaces are converted into a single space; ASCII tab characters are treated as spaces; ASCII return is treated as end-of-line (this means, among other things, that long terms can be broken across several lines).

You have to be careful with the begin- and end-group characters ('`{`' and '`}`' by default). If they are matched, you don't have to do anything special. For example:

```
\sidx {braces {, }@braces
  \verbatim {"|endverbatim, \verbatim }"|endverbatim}
```

However, if they are not matched you have two problems on hand. The first one is TEX—you have to instruct TEX to use something else as begin- and/or end-group characters. Eplain provides an easy way to do this: just define `\idxargopen` and/or `\idxargclose` to the begin- and end-group characters you are going to use with indexing macros, and use braces inside index terms without any restrictions. Here's an example:

```
\def\idxargopen{`\<}
\def\idxargclose{`\>}
\sidx <left brace "{@left brace \verbatim "{"|endverbatim>
```

In this example we've also dealt with the second problem—braces are MakeIndex's grouping characters as well (by default), so we have escaped unmatched braces with '`"`'.

And the final note: if you need a subentry containing brackets (‘`[`’ and ‘`]`’), avoid the optional argument of `\sidx` and friends, and use instead MakeIndex’s subentry separator to create the subentry with the brackets in it:

```
\sidx{entry!subentry with a bracket []}
```

#### 4.12.1.4 Proofing index terms

As you are reading through a manuscript, it is helpful to see what terms have been indexed, so you can add others, catch miscellaneous errors, etc. (Speaking from bitter experience, I can say it is extremely error-prone to leave all indexing to the end of the writing, since it involves adding many TEX commands to the source files.)

So Eplain puts index terms in the margin of each page, if you set `\indexproofingtrue`. It is `false` by default. The terms are typeset by the macro `\indexproofterm`, which takes a single argument, the term to be typeset. Eplain’s definition of `\indexproofterm` just puts it into an `\hbox`, first doing `\indexprooffont`, which Eplain defines to select the font `cmtt8`. With this definition long terms run off the page, but since this is just for proofreading anyway, it seems acceptable.

On the other hand, we certainly don’t want the index term to run into the text of the page, so Eplain uses the right-hand side of the page rather than the left-hand page (assuming a language read left to right here). So `\ifodd\pageno`, Eplain kerns by `\outsidemargin`, otherwise by `\insidemargin`. If those macros are undefined, `\indexsetmargins` defines them to be one inch plus `\hoffset`.

To get the proofing index entries on the proper page, Eplain defines a new insertion class `\@indexproof`. To unbox any index proofing material, Eplain redefines `\makeheadline` to call `\indexproofunbox` before the original `\makeheadline`. Thus, if you have your own output routine, that redefines or doesn’t use `\makeheadline`, it’s up to you to call `\indexproofunbox` at the appropriate time.

### 4.12.2 Typesetting an index

The command `\readindexfile{i}` reads and typesets the `.ind` file that MakeIndex outputs (from the `.idx` file which the indexing commands in the previous sections write). Eplain defines a number of commands that support the default MakeIndex output.

More precisely, `\readindexfile` reads `\indexfilebasename.`*`index-letter`*`nd`, where the *index-letter* is the argument. `\indexfilebasename` is `\jobname` by default, but if you have different indexes in different parts of a book, you may wish to change it, just as with bibliographies (see Section 4.3 [Citations], page 5).

MakeIndex was designed to work with LATEX; therefore, by default the `.ind` file starts with `\begin{theindex}` and ends with `\end{theindex}`. If no `\begin` has been defined, Eplain defines one to ignore its argument and set up for typesetting the index (see below), and also defines a `\end` to ignore its argument. (In a group, naturally, since there is a primitive `\end`).

Eplain calls `\indexfonts`, sets `\parindent = 0pt`, and does `\doublecolumns` (see Section 4.16 [Multiple columns], page 31) at the `\begin{theindex}`. `\indexfonts` does nothing by default; it’s just there for you to override. (Indexes are usually typeset in smaller type than the main text.)

It ends the setup with `\hookrun{beginindex}`, so you can override anything you like in that hook (see Section 7.6.3 [Hooks], page 74). For example:

```
\hookaction{beginindex}{\triplecolumns}
```

MakeIndex turns each main index entry into an `\item`, subentries into `\subitem`, and subsubentries into `\subsubitem`. By default, the first line of main entries are not indented, and subentries are indented 1em per level. Main entries are preceded by a `\vskip` of `\aboveitemskipamount`, `0pt plus2pt` by default. Page breaks are encouraged before main entries (`\penalty -100`), but prohibited afterwards—Eplain has no provision for "continued" index entries.

All levels do the following:

```
\hangindent = 1em
\raggedright
\hyphenpenalty = 10000
```

Each entry ends with `\hookrun{indexitem}`, so you can change any of this. For example, to increase the allowable rag:

```
\hookaction{indexitem}{\advance\rightskip by 2em}
```

Finally, MakeIndex outputs `\indexspace` between each group of entries in the `.ind` file. Eplain makes this equivalent to `\bigbreak`.

### 4.12.3 Customizing indexing

By default, MakeIndex outputs '`, `' after each term in the index. To change this, you can add the following to your MakeIndex style (`.ist`) file:

```
delim_0 "\\afterindexterm "
delim_1 "\\afterindexterm "
delim_2 "\\afterindexterm "
```

Eplain makes `\afterindexterm` equivalent to `\quad`.

You can also change the keywords Eplain recognizes (see Section 4.12.1.2 [Modifying index entries], page 22):

`\idxrangebeginword`
  'begin'

`\idxrangeendword`
  'end'

`\idxseecmdword`
  'see'

`\idxseealsocmdword`
  'seealso'

You can also change the magic characters Eplain puts into the `.idx` file, in case you've changed them in the `.ist` file:

`\idxsubentryseparator`
  '`!`'

`\idxencapoperator`
  '`|`'

`\idxbeginrangemark`
‘(’

`\idxendrangemark`
‘)’

There is no macro for the `actual` (‘`@`’ by default) character, because it’s impossible to make it expand properly.

You can change the (imaginary) page number that “see also” entries sort as by redefining `\idxmaxpagenum`. This is 99999 by default, which is one digit too many for old versions of MakeIndex.

The words output by Eplain for “see” and “see also” index entries are defined by `\indexseeword` and `\indexseealsowords` respectively. You can change the typeface used for these words by redefining `\seevariant`. And finally, the macros `\indexsee` and `\indexseealso` actually produce the “see ...” entries, so you can redefine them if you want something entirely different. If you do redefine them, make them take two parameters, the term being referenced and the `\idxmaxpagenum` (the latter should normally be ignored). See the example below.

Unfortunately, it is impossible to reliably control the commas produced by MakeIndex in front of “see ...” entries in the `.ind` file, either at MakeIndex level or at Eplain level. However, the `sed` script contained in `trimsee` distributed with Eplain in the `util` directory can be used to filter out these commas from the output of MakeIndex. For example, suppose you want the following style for your “see ...” entries:

> analysis,
>   archetypal (*see* archetypal criticism)
> archetypal criticism,
>   elements of, 75, 97, 114 (*see also* dichotomies)

You would need to redefine these macros in your TEX file:

```
\def\indexsee#1#2{({\seevariant \indexseeword\/ }#1)}
\def\indexseealso#1#2{({\seevariant \indexseealsowords\/ }#1)}
```

and then filter out the commas in front of the “see ...” entries by running the following command to produce the `.ind` file (assuming the name of the `.idx` file is `myfile.idx` and the `trimsee` script is placed in the current directory):

```
prompt$ cat myfile.idx | makeindex | ./trimsee > myfile.ind
```

By default, `trimsee` uses default page list separators and default “see ...” command names. If you set up MakeIndex to use different page list separator or change the names of `\indexsee` and `\indexseealso` commands, it is possible to adjust the `trimsee` script through its command line options, which are the following:

`-i` *is*
Use *is* as a regular expression matching separator before “see ...” commands in the input (default: ‘`, \+`’).

`-o` *os*
Use *os* as a separator to replace *is* before “see ...” commands (default: ‘ ’).

`-s` *see*
Use *see* as a regular expression matching “see ...” commands (default: ‘`\\indexsee`’).

`-h`
`--help`
Print a usage message.

`-v`
`--version`
Print version.

`trimsee` reads input from the standard input, and directs its output to the standard output.

## 4.13 Justification

Eplain defines three commands to conveniently justify multiple lines of text: `\flushright`, `\flushleft`, and `\center`.

They all work in the same way; let's take `\center` as the example. To start centering lines, you say `\center` inside a group; to stop, you end the group. Between the two commands, each end-of-line in the input file also starts a new line in the output file.

The entire block of text is broken into paragraphs at blank lines, so all the TEX paragraph-shaping parameters apply in the usual way. This is convenient, but it implies something else that isn't so convenient: changes to any linespacing parameters, such as `\baselineskip`, will have *no effect* on the paragraph in which they are changed. TEX does not handle linespacing changes within a paragraph (because it doesn't know where the line breaks are until the end of the paragraph).

The space between paragraphs is by default one blank line's worth. You can adjust this space by assigning to `\blanklineskipamount`; this (vertical) glue is inserted after each blank line.

Here is an example:

```
{\center First line.

   Second line, with a blank line before.
}
```

This produces:

First line.

Second line, with a blank line before.

You may wish to use the justification macros inside of your own macros. Just be sure to put them in a group. For example, here is how a title macro might be defined:

```
\def\title{\begingroup\titlefont\center}
\def\endtitle{\endgroup}
```

In addition, Eplain defines `\raggedleft`, analogous to plain TEX's `\raggedright`. This macro is also typically used inside a group, but unlike the environments above, TEX does normal line breaking; that is, ends-of-lines in the input file aren't treated specially. Just like plain's `\raggedright`, it also resets `\spaceskip` and `\xspaceskip` so that interword spacing is uniform. It also sets `\parfillskip` to zero so that last lines of paragraphs are also "ragged left". Finally, `\leftskip`'s new value is taken from a new glue register, `\raggedleft`; its default value is `0pt plus 2em`, the same as `\raggedright`'s `\rightskip`.

Here's an example:

```
{\raggedleft This text will be set ragged left,
although the left margin won't be too ragged by default.
You may well want to increase the value of
{\tt \char`\\raggedleftskip} before calling the macro.
It's necessary to end the paragraph before ending the group
or the setting won't have any effect, so: {\tt \char`\\par}
}
```

Despite `\raggedleft` resetting `\parfillskip` to zero, TeX's line breaking may still prefer to make the last line of a paragraph considerably shorter than the rest, to minimize overall badness. Increasing `\raggedleftskip` may help somewhat, but using `\emergencystretch`, retaining interword stretchability by assigning `\leftskip` directly, or even forcing line breaks may be necessary.

## 4.14 Tables

Eplain provides a single command, `\makecolumns`, to make generating one particular kind of table easier. More ambitious LaTeX styles and macro packages tackle more difficult applications. The `autorows` feature of the Memoir package provides similar functionality to this.

Many tables are homogenous, i.e., all the entries are semantically the same. The arrangement into columns is to save space on the page, not to encode different meanings. In this kind of the table, it is useful to have the column breaks chosen automatically, so that you can add or delete entries without worrying about the column breaks.

`\makecolumns` takes two arguments: the number of entries in the table, and the number of columns to break them into. As you can see from the example below, the first argument is delimited by a slash, and the second by a colon and a space (or end-of-line). The entries for the table then follow, one per line (not including the line with the `\makecolumns` command itself).

`\parindent` defines the space to the left of the table. `\hsize` defines the width of the table. So you can adjust the position of the table on the page by assignments to these parameters, probably inside a group.

You can also control the penalty at a page break before the `\makecolumns` by setting the parameter `\abovecolumnspenalty`. Usually, the table is preceded by some explanatory text. You wouldn't want a page break to occur after the text and before the table, so Eplain sets it to `10000`. But if the table produced by `\makecolumns` is standing on its own, `\abovecolumnspenalty` should be decreased.

If you happen to give `\makecolumns` a smaller number of entries than you really have, some text beyond the (intended) end of the table will be incorporated into the table, probably producing an error message, or at least some strange looking entries. And if you give `\makecolumns` a larger number of entries than you really have, some of the entries will be typeset as straight text, probably also looking somewhat out of place.

Here is an example:

```
% Arrange 6 entries into 2 columns:
\makecolumns 6/2: % This line doesn't have an entry.
one
two
```

```
three
four
five
six
Text after the table.
```

This produces 'one', 'two', and 'three' in the first column, and 'four', 'five', and 'six' in the second.

## 4.15 Margins

TEX's primitives describe the type area in terms of an offset from the upper left corner, and the width and height of the type. Some people prefer to think in terms of the *margins* at the top, bottom, left, and right of the page, and most composition systems other than TEX conceive of the page laid out in this way. Therefore, Eplain provides commands to directly assign and increment the margins.

`\topmargin = `*`dimen`*
`\bottommargin = `*`dimen`*
`\leftmargin = `*`dimen`*
`\rightmargin = `*`dimen`*

> These commands set the specified margin to the *dimen* given. The `=` and the spaces around it are optional. The control sequences here are not TEX registers, despite appearances; therefore, commands like `\showthe\topmargin` will not do what you expect.

`\advancetopmargin by `*`dimen`*
`\advancebottommargin by `*`dimen`*
`\advanceleftmargin by `*`dimen`*
`\advancerightmargin by `*`dimen`*

> These commands change the specified margin by the *dimen* given.

Regardless of whether you use the assignment or the advance commands, Eplain always changes the type area in response, not the other margins. For example, when TEX starts, the left and right margins are both one inch. If you then say `\leftmargin = 2in`, the right margin will remain at one inch, and the width of the lines (i.e., `\hsize`) will decrease by one inch.

When you use any of these commands, Eplain computes the old value of the particular margin, by how much you want to change it, and then resets the values of TEX's primitive parameters to correspond. Unfortunately, Eplain cannot compute the right or bottom margin without help: you must tell it the full width and height of the final output page. It defines two new parameters for this:

`\paperheight`

> The height of the output page; default is 11truein.

`\paperwidth`

> The width of the output page; default is 8.5truein.

If your output page has different dimensions than this, you must reassign to these parameters, as in

```
\paperheight = 11truein
```

```
\paperwidth = 17truein
```

## 4.16 Multiple columns

Eplain provides for double, triple, and quadruple column output: say `\doublecolumns`, `\triplecolumns`, or `\quadcolumns`, and from that point on, the manuscript will be set in columns. To go back to one column, say `\singlecolumn`.

You may need to invoke `\singlecolumn` to balance the columns on the last page of output.

To do a "column eject", i.e., move to the top of the next column, do `\columnfill`. This does not actually force an eject, however: it merely inserts an unbreakable space of (essentially) size `\@normalvsize` minus `\pagetotal` (where `\@normalvsize` is the usual height of the page; to implement multicolumns, Eplain multiplies `\vsize` itself by the number of columns). In most circumstances, a column break will be forced after this space (during the column splitting operation when the whole page is output). Bugs are easily possible, unfortunately.

The columns are separated by the value of the dimen parameter `\gutter`. Default value is two picas. If you want to add vertical material between the columns, use `\gutterbox`. For example, to put a vertical line between columns, define `\gutterbox` as

```
\def\gutterbox{\vbox to \dimen0{\vfil\hbox{\vrule height\dimen0}\vfil}}%
```

The dimension counter `\dimen0` contains the height of the column.

All the `\...columns` macros insert the value of the glue parameter `\abovecolumnskip` before the multicolumn text, and the value of the glue parameter `\belowcolumnskip` after it. The default value for both of these parameters is `\bigskipamount`, i.e., one linespace in plain TeX.

The macros take into account only the insertion classes defined by plain TeX; namely, footnotes and `\topinserts`. If you have additional insertion classes, you will need to change the implementation.

Also, Eplain makes insertions the full page width. There is no provision for column-width insertions.

## 4.17 Footnotes

The most common reference mark for footnotes is a raised number, incremented on each footnote. The `\numberedfootnote` macro provides this. It takes one argument, the footnote text.

If your document uses only numbered footnotes, you could make typing `\numberedfootnote` more convenient with a command such as:

```
\let\footnote = \numberedfootnote
```

After doing this, you can type your footnotes as `\footnote{`*footnote text*`}`, instead of as `\numberedfootnote{`*footnote text*`}`.

Eplain keeps the current footnote number in the count register `\footnotenumber`. So, to reset the footnote number to zero, as you might want to do at, for example, the beginning of a chapter, you could say `\footnotenumber=0`.

Plain TEX separates the footnote marker from the footnote text by an en space (it uses the `\textindent` macro). In Eplain, you can change this space by setting the dimension register `\footnotemarkseparation`. The default is still an en.

You can produce a space between footenotes by setting the glue register `\interfootnoteskip`. The default is zero.

`\parskip` is also set to zero by default before the beginning of each footnote (but not for the text of the footnote).

You can also control footnote formatting in a more general way: Eplain expands the token register `\everyfootnote` before a footnote is typeset, but after the default values for all the parameters have been established. For example, if you want your footnotes to be printed in seven-point type, indented by one inch, you could say:

```
\everyfootnote = {\sevenrm \leftskip = 1in}
```

By default, an `\hrule` is typeset above each group of footnotes on a page. You can control the dimensions of this rule by setting the dimension registers `\footnoterulewidth` and `\footnoteruleheight`. The space between the rule and the first footnote on the page is determined by the dimension register `\belowfootnoterulespace`. If you don't want any rule at all, set `\footenoteruleheight=0pt`, and, most likely, `\belowfootnoterulespace=0pt`. The defaults for these parameters typeset the rule in the same way as plain TEX: the rule is 0.4 points high, 2 true inches wide, with 2.6 points below it.

The space above the rule and below the text on the page is controlled by the glue register `\skip\footins`. The default is a plain TEX `\bigskip`.

Eplain can create hypertext links for the footnote marks (see Section 5.3.9 [Footnote hyperlinks], page 47).

## 4.18 Fractions

Exercise 11.6 of *The TEXbook* describes a macro `\frac` for setting fractions, but `\frac` never made it into plain TEX. So Eplain includes it.

`\frac` typesets the numerator and denominator in `\scriptfont0`, slightly raised and lowered. The numerator and denominator are separated by a slash. The denominator must be enclosed in braces if it's more than one token long, but the numerator need not be. (This is a consequence of `\frac` taking delimited arguments; see page 203 of *The TEXbook* for an explanation of delimited macro arguments.)

For example, `\frac 23/{64}` turns '23/64' into ${}^{23}\!/_{64}$.

## 4.19 Paths

When you typeset long pathnames, electronic mail addresses, or other such "computer" names, you would like TEX to break lines at punctuation characters within the name, rather than trying to find hyphenation points within the words. For example, it would be better to break the email address `letters@alpha.gnu.ai.mit.edu` at the '`@`' or a '`.`', rather than at the hyphenation points in '`letters`' and '`alpha`'.

If you use the `\path` macro to typeset the names, TEX will find these good breakpoints. The argument to `\path` is delimited by any character other than '\' which does not appear in the name itself. '`|`' is often a good choice, as in:

```
\path|letters@alpha.gnu.ai.mit.edu|
```

You can control the exact set of characters at which breakpoints will be allowed by calling `\discretionaries`. This takes the same sort of delimited argument; any character in the argument will henceforth be a valid breakpoint within `\path`. The default set is essentially all the punctuation characters:

```
\discretionaries |~!@$%^&*()_+‘-=#{}[]:";’<>,.?\/|
```

If for some reason you absolutely must use `\` as the delimiter character for `\path`, you can set `\specialpathdelimiterstrue`. (Other delimiter characters can still be used.) TeX then processes the `\path` argument about four times more slowly.

The `\path` macro comes from `path.sty`, written by Nelson Beebe and Philip Taylor, and available at `https://ctan.org/pkg/path`.

## 4.20 Logos

Eplain redefines the `\TeX` macro of plain TeX to end with `\null`, so that the proper spacing is produced when `\TeX` is used at the end of a sentence. The other . . . TeX macros listed here do this, also.

Eplain defines `\AMSLaTeX`, `\AMSTeX`, `\BibTeX` `\eTeX`, `\ExTeX`, `\LAMSTeX`, `\LaTeX`, `\MF`, `\SLiTeX`, `\XeLaTeX`, and `\XeTeX` to produce their respective logos. (Sorry, the logos are not shown here.) Some spelling variants of these are also supported.

Most of these macros come from `texnames.sty`, compiled by Nelson Beebe and available at `http://mirror.ctan.org/info/biblio/texnames.sty` (part of the `biblio` package, `https://ctan.org/pkg/biblio`).

## 4.21 Boxes

The solid rectangle that Eplain uses as a marker in unordered lists (see Section 4.6 [Lists], page 10) is available by itself: just say `\blackbox`.

You can create black boxes of arbitrary size with `\hrule` or `\vrule`.

You can also get unfilled rectangles with `\makeblankbox`. This takes two explicit arguments: the height and depth of the rules that define the top and bottom of the rectangle. (The two arguments are added to get the width of the left and right borders, so that the thickness of the border is the same on all four sides.) It also uses, as implicit arguments, the dimensions of `\box0` to define the dimensions of the rectangle it produces. (The contents of `\box0` are ignored.)

Here is an example. This small raised open box is suitable for putting next to numbers in, e.g., a table of contents.

```
\def\openbox{%
  \ht0 = 1.75pt \dp0 = 1.75pt \wd0 = 3.5pt
  \raise 2.75pt \makeblankbox{.2pt}{.2pt}
}
```

Finally, you can put a box around arbitrary text with `\boxit`. This takes one argument, which must itself be a (TeX) box, and puts a printed box around it, separated by `\boxitspace` white space (3 points by default) on all four sides. For example:

```
\boxit{\hbox{This text is boxed.}}
```

The reason that the argument must be a box is that when the text is more than one line long, TEX cannot figure out the line length for itself. Eplain does set `\parindent` to zero inside `\boxit`, since it is very unlikely you would want indentation there. (If you do, you can always reset it yourself.)

`\boxit` uses `\ehrule` and `\evrule` so that you can easily adjust the thicknesses of the box rules. See Section 4.2 [Rules], page 5.

## 4.22 Checking for PDF output

pdfTEX is an extended TEX that can output both `.dvi` and `.pdf` (Adobe's Portable Document Format) files (see `https://ctan.org/pkg/pdftex`). You might sometimes want to know whether the target format is `.pdf` or `.dvi`. The `\ifpdf` switch can be used to detect pdfTEX in PDF mode:

```
\ifpdf
   This text is produced when pdfTeX is in PDF mode.
\else
   This text is produced when pdfTeX is in DVI mode,
   or when some program other than pdfTeX is used.
\fi
```

Keep in mind that `\ifpdf` is set based on the value of the `\pdfoutput` primitive of pdfTEX at the time Eplain is loaded. If you change the value of `\pdfoutput` after you load Eplain, `\ifpdf` will not reflect the change.

Eplain defines `\ifpdf` by incorporating Heiko Oberdiek's `ifpdf.sty`, which is available at `https://ctan.org/pkg/ifpdf`.

## 4.23 Loading LATEX packages

Eplain provides a limited support for loading LATEX packages (`.sty` files—not `.cls`). This will mostly work for packages which were designed with plain TEX compatibility in mind, which means that most LATEX packages cannot be loaded. The packages which are known to work are listed below (see Section 4.23.3 [Packages known to work], page 36). If you discover a working package which is not in the list, please report it to the Eplain mailing list (see Chapter 1 [Introduction], page 1).

To set up a pseudo-LATEX environment for the packages, Eplain uses `miniltx.tex` (`https://ctan.org/pkg/miniltx`) from the LATEX graphics collection, written by David Carlisle and Sebastian Rahtz. Eplain extends `miniltx.tex` to provide (primarily) support for package options.

### 4.23.1 The `\usepackage` command

`\usepackage` loads a LATEX package. Its syntax is similar to that of LATEX's `\usepackage` command:

```
\usepackage[options]{packages}[version]
```

where *options* is a comma-separated list of package options, *packages* is a comma-separated list of packages to load (without the `.sty` suffix), and *version* is a package version number given as a date in the format '`YYYY/MM/DD`'. If an older version of the package is found, a warning is issued. If several packages are loaded within a single `\usepackage` command, the *options* will be applied to each of the packages. As usual, parameters in square brackets are optional and can be omitted (together with the square brackets).

For example:

```
\usepackage[foo,bar]{pack1,pack2}[2005/08/29]
```

will load packages '`pack1`' and '`pack2`', each with the options '`foo`' and '`bar`', and will check that each of the packages are dated 2005/08/29 or newer.

### 4.23.2 Environment for loading packages

Some packages request that certain commands are executed after all packages have been loaded. In LaTeX, this means that the commands are executed at the beginning of the document, after the so-called *preamble*. Neither plain TeX nor Eplain have a concept of preamble; therefore, Eplain requires that all packages be loaded inside a `\beginpackages...\endpackages` block. For example:

```
\beginpackages
  \usepackage[foo,bar]{pack1}
  \usepackage{pack2}
\endpackages
```

This requirement enables Eplain to execute the "delayed" commands at the end of the `\beginpackages...\endpackages` block.

For the same reason, it is advisable to specify only one such block per document, just like there is only one preamble in LaTeX.

Both the `miniltx.tex` file used by Eplain and some LaTeX packages redefine TeX's primitive `\input` to be a macro. Under plain TeX, users probably expect the primitive `\input`. Therefore, at the beginning of the `\beginpackages...\endpackages` block Eplain saves the meaning of `\input` as `\eplaininput` and restores the original `\input` at the end of the block. This usually means that the primitive `\input` is restored, unless you (or some other macro package you've loaded directly) have redefined it before calling `\beginpackages`. In case you need to access the package-provided `\input`, Eplain saves it as `\packageinput`.

Along the same lines, Eplain restores the catcode of '`@`' at `\endpackages` to whatever it was before (using `\resetatcatcode`, as defined by `miniltx.tex`). This is needed because `miniltx.tex`, read by `\beginpackages`, does not restore the catcode of '`@`', but leaves it as 11 (letter).

Sometimes you may encounter packages which make conflicting redefinitions of `\input`. Common symptoms are TeX spewing incomprehensible error messages or hanging in a loop at a call to `\input`. This sometimes can be cured by restoring `\input` to `\eplaininput` before loading each package. For example:

```
\beginpackages
  \usepackage{pack1}
  \let\input\eplaininput
  \usepackage{pack2}
\endpackages
```

### 4.23.3 Packages known to work

The following table lists packages that had been tested and are known to work with Eplain, and locations where you can find manuals for these packages. Some of the short descriptions of the packages were taken from the documentation for those packages.

autopict (‘`2001/06/04 v1.1j Picture mode autoload file`’)
`https://tug.org/eplain/misc/ltpictur.pdf`

This is the LaTeX “picture mode”, started by `\begin{picture}` and ended by `\end{picture}` (in LaTeX, this package is not explicitly loaded since it is part of the LaTeX kernel). It provides commands to draw simple figures inside your document without resorting to any external tools.

color (‘`1999/02/16 v1.0i Standard LaTeX Color (DPC)`’)
graphics (‘`2001/07/07 v1.0n Standard LaTeX Graphics (DPC,SPQR)`’)
graphicx (‘`1999/02/16 v1.0f Enhanced LaTeX Graphics (DPC,SPQR)`’)
`https://ctan.org/pkg/graphics`

These packages are from the LaTeX graphics collection. (The independent `xcolor` package does not work with Eplain.) They provide commands for changing text/page colors, text rotation and scaling, and much more.

**Warning 1:** If you encounter problems loading one of these packages under pdfTEX (when pdfTEX reads `supp-mis.tex`), the cause may be an outdated `supp-mis.tex` (part of ConTEXt, a typesetting system for TEX) installed on your system. The problem was fixed in `supp-mis.tex` version 2004.10.26. You can obtain up-to-date versions of `supp-mis.tex` and the accompanying `supp-pdf.tex` from `http://mirror.ctan.org/macros/pdftex/graphics`. To convince TEX to use the new files, you have the following options:

1. put the new files in the same directory with your document;
2. overwrite the outdated files installed by your TEX distribution;
3. install the new `supp-mis.tex` and `supp-pdf.tex` files in the relevant subdirectory of your local `texmf` tree (for info on TEX directory structure see `https://tug.org/tds/` and `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=tds`);
4. upgrade your ConTEXt installation.

Note that option 1 is the safest but provides the fix only for your current document. Options 2 and 3 will usually suffice for Eplain but may break ConTEXt. Option 4 is the most general but is more complicated than the first three. Be sure to backup any files you overwrite. Also keep in mind that upgrading your TEX distribution may overwrite files you install in the system `texmf` tree.
**End of warning 1.**

**Warning 2:** If you encounter problems using the `\pagecolor` command from the `color.sty` package under pdfTEX, the cause may be an outdated pdfTEX color and graphics driver `pdftex.def`. The problem was fixed in `pdftex.def` version 0.03p. You can obtain an up-to-date version from `http://ctan.org/pkg/pdftex-def`.
**End of warning 2.**

The `\fcolorbox` macro provided by the `color` package requires the macro `\fbox` to work, but `miniltx` does not provide that. Here is a definition for it that uses Eplain's `\boxit` (see Section 4.21 [Boxes], page 33), thanks to Dan Luecking and Helmut Jarausch:

```
\makeatletter
\def\fbox#1{{%
  \hruledefaultheight=\fboxrule
  \hruledefaultdepth=0pt
  \vruledefaultwidth=\fboxrule
  \let\boxitspace\fboxsep % use miniltx register
  \boxit{\color@begingroup\hbox{#1}\color@endgroup}%
}}
\makeatother
```

The `graphics/graphicx` packages have the option `draft` which instructs `\includegraphics` not to include the graphics but instead typeset a box with the dimensions of the graphics and the name of the graphics file in typewriter type at the center of the box. These packages expect the LaTeX-provided command `\ttfamily` to switch to typewriter type. This command is not defined by `miniltx.tex`, therefore Eplain defines it and makes it equivalent to plain TeX's `\tt`.

See Section 8.1 [Hyperlinks (xhyper.tex)], page 80, for the demonstration of text rotation and graphics inclusion using the `graphicx` package, and using the `color` package to colorize hypertext links.

Klaus Höppner has written a nice introduction to the LaTeX graphics packages and different graphics formats. You can download it from

`https://tug.org/pracjourn/2005-3/hoeppner`

epstopdf (‘`2009/07/16 v2.2 Conversion with epstopdf on the fly (HO)`’)

`https://ctan.org/pkg/epstopdf-pkg`

This package does on-the-fly conversion of Encapsulated PostScript (EPS) graphics into Portable Document Format (PDF) graphics for inclusion with the `\includegraphics` command from the `graphics/graphicx` packages, so that you do not have to explicitly call the `epstopdf` script.

psfrag (‘`1998/04/11 v3.04 PSfrag (MCG)`’)

`https://ctan.org/pkg/psfrag`

PSfrag allows the user to precisely overlay Encapsulated PostScript (EPS) files with arbitrary (La)TeX constructions. In order to accomplish this, the user places a simple text “tag” in the graphics file, as a “position marker” of sorts. Then, using simple (La)TeX commands, the user instructs PSfrag to remove that tag from the figure, and replace it with a properly sized, aligned, and rotated (La)TeX equation.

soul (‘`2003/11/17 v2.4 letterspacing/underlining (mf)`’)

`https://ctan.org/pkg/soul`

This package provides hyphenatable letterspacing (spacing out), underlining, and some derivatives. The package is optimized for LaTeX, but works with plain

TEX—you don't actually need to load it with the `\usepackage` command, just say `\input soul.sty`. If you intend to use the highlighting macros of `soul`, don't forget to load the `color` package.

url (‘`2005/06/27 ver 3.2 Verb mode for urls, etc.`’)

`https://ctan.org/pkg/url`

This package provides a form of `\verbatim` that allows line breaks at certain characters or combinations of characters, accepts reconfiguration, and can usually be used in the argument to another command. It is intended for email addresses, hypertext links, directories/paths, etc., which normally have no spaces.

Eplain can create hypertext links with the `\url` command (see Section 5.3.2 [URL hyperlinks], page 42).

Be sure to get a version dated at least 2005/06/27, as older versions have problems in plain TEX.

### 4.23.4 Packages known not to work

The following packages are known not to work with Eplain:

hyperref `https://ctan.org/pkg/hyperref`

This package depends heavily on LATEX, so that it is essentially unusable outside of LATEX. Eplain provides its own macros for creating hyperlinks; see Chapter 5 [Hyperlinks], page 39.

microtype (‘`2013/05/23 v2.51 Micro-typographical refinements (RS)`’)

`https://ctan.org/pkg/microtype`

pict2e (‘`2005/07/15 v0.2r Improved picture commands (HjG,RN)`’)

`https://ctan.org/pkg/pict2e`

xcolor (‘`2005/06/06 v2.03 LaTeX color extensions (UK)`’)

`https://ctan.org/pkg/xcolor`

# 5 Hyperlinks

This chapter describes the support which Eplain provides for hypertext links (*hyperlinks* for short). Hyperlinks can be created implicitly by the cross-reference, indexing and other macros in Eplain. Macros for constructing explicit hyperlinks are also provided.

## 5.1 Introduction to hyperlinks

The original TEX engine has no built-in support for hyperlinks (a.k.a. hypertext links). Many of the present-day file formats with hyperlinking capabilities did not even exist at the time TEX was written. However, TEX's `\special` primitive can be used to instruct TEX to write special directives into its `.dvi` output file. These directives are not interpreted by TEX in any way; they are intended for programs which process the `.dvi` files produced by TEX, be it printing or converting to other formats, such as `.ps` or `.pdf`.

Another approach is to extend the original TEX engine with the ability to generate one of the hyperlinking formats; TEX's set of primitives can be extended to include hyperlink commands. This is the approach used by the pdfTEX engine, which is capable of producing `.pdf` files directly from the TEX source, skipping the `.dvi` generation and processing step.

It turns out that the sets of commands for different formats are mostly not interchangeable, as each of the file formats has its own quirks and capabilities. And this is where Eplain *hyperlink drivers* come into play.

In order for Eplain to generate proper commands, Eplain has to know two things: which engine or `.dvi` processor you are using, and the set of commands it understands.

The knowledge about the commands that the various processors understand is programmed into Eplain's hyperlink drivers. Eplain provides three drivers: `hypertex` (implementation of the HyperTEX standard, see `http://arxiv.org/hypertex`), and `pdftex` and `dvipdfm` (named after the programs which process the hyperlink commands, pdfTEX and dvipdfm). Therefore, Eplain can only produce HyperTEX commands and hyperlink commands for one of these two programs—except that the extended `dvipdfmx` program can be used as well as the original `dvipdfm`, since they are compatible.

To tell Eplain which `.dvi` processor or extended TEX engine you are using, use the command `\enablehyperlinks`.

For example:

```
\enablehyperlinks
```

instructs Eplain to attempt to automatically detect which driver to use, as follows: if it detects pdfTEX in PDF mode, it loads the `pdftex` driver. If it does not detect pdfTEX in PDF mode, the `hypertex` driver is loaded. The detection is based on the `\ifpdf` switch (see Section 4.22 [Checking for PDF output], page 34).

If necessary, you can explicitly specify the driver name:

```
\enablehyperlinks[dvipdfm]
```

will start producing hyperlinks under the assumption that you are using pdfTEX.

Eplain does not produce any hyperlinks until you explicitly enable them with `\enablehyperlinks`. For one thing, this keeps Eplain backward-compatible with previous releases without hyperlink support. For another, you may be using a program other than pdfTEX or `dvipdfm`, which does not understand their hyperlink commands or the HyperTEX commands.

## Concepts and Terminology

In general, hyperlinks work as follows. You mark some place in your document as a hyperlink destination, associating a *hyperlink label* with that destination. Next, somewhere within your document, you create a hyperlink, using a label to identify the destination you want this link to point to. A hyperlink is a region in the document (which can take many forms, for example, text or a picture); when a user clicks on that region, they will be taken to a place in the document marked by the corresponding destination. The following two sections (Section 5.2 [Explicit hyperlinks], page 40, and Section 5.3 [Implicit hyperlinks], page 41) describe the macros you can use to define destinations and create links pointing to those destinations.

In the rest of this chapter, we will often need to refer to links and destinations jointly, in which case we will use the term *hyperlinks*. We will use the terms *links* and *destinations* in cases when we need to refer specifically to links or destinations.

Hyperlink drivers provide several kinds of links and destinations. We will refer to them as *link types* and *destination types*.

For example, one of the destination types that the `pdftex` driver provides is the '`xyz`' type; when the user follows a link pointing to an '`xyz`' destination, the exact location marked by that destination is displayed. Another destination type provided by the `pdftex` driver is the '`fit`' type; when the user follows a link pointing to a '`fit`' destination, the page containing that destination is zoomed to fit into the window in which the document is displayed.

Similarly, drivers support various link types. For example, with the `pdftex` driver, the usual link type used to refer to destinations in the current document is called '`name`'. You can also create a link pointing to another local document (by using the '`filename`' link type) or to a URL (by using the '`url`' link type).

In addition, each hyperlink driver supports a number of destination and link *options*. By setting these options you can customize hyperlink parameters (e.g., the thickness of the border drawn around a hyperlink) or pass information to hyperlinks (for example, file name of a document, for a link pointing to a destination in another document).

See Section 5.4 [Hyperlink drivers], page 47, for the description of hyperlink types and options supported by the drivers. See Section 5.5 [Setting hyperlink types and options], page 57, for the information on how to set hyperlink options.

## 5.2 Explicit hyperlinks

Explicit hyperlinks are created by you, in the source of your document. The simplest command is `\hldest`, which marks the current position in your document as a destination:

```
\hldest{type}{options}{label}
```

Here *type* is one of the destination types supported by the hyperlink driver (see Section 5.4 [Hyperlink drivers], page 47), *options* is a comma-separated list of option assignments, and *label* is the hyperlink label to associate with this destination. This label will identify the destination when creating links pointing to this destination. For example, with the `pdftex` driver, the command

```
\hldest{xyz}{zoom=2000}{index}
```

creates a destination of type '`xyz`' ("the current position"), sets the magnification ratio for this destination to be 200%, and associates the label `index` with the destination.

Another command, `\hlstart`, paired with `\hlend`, turns all intervening material into a link:

```
\hlstart{type}{options}{label} ... \hlend
```

Here *type*, *options* and *label* have the same meaning as for `\hldest`. Continuing the previous example,

```
\hlstart{name}{bstyle=U,bwidth=2}{index} Index\hlend
```

typesets the word 'Index' as a link with underline border of width 2 PostScript points, pointing to the named destination `index` defined in the previous example. (The other options, like highlight mode and border color, are determined by the defaults, see Section 5.5.1 [Setting default types and options], page 57).

The *label* argument of both `\hldest` and `\hlstart` can contain special characters (such as '`#`', '`%`', '`&`', '`~`', etc.) without any escaping. This is especially important for URL links supported by some drivers (see Section 5.4 [Hyperlink drivers], page 47).

Both `\hldest` and `\hlstart` ignore following spaces.

Both `\hldest` and `\hlstart` expand the first token of *options* once, so you can save a list of options in a macro and pass it for the *options*. For example:

```
\def\linkopts{bstyle=U,bwidth=2}
\hlstart{name}{\linkopts}{index}Index\hlend
```

is functionally equivalent to the previous example.

See Section 8.1 [Hyperlinks (xhyper.tex)], page 80, for a demonstration of the usage of explicit hyperlinks.

## 5.3 Implicit hyperlinks

*Implicit hyperlinks* are hyperlinks created implicitly by various Eplain macros, such as the macros for citations, cross-references, indexing, etc.

All such macros are divided into *link groups* and *destination groups* (or *linkgroups* and *destgroups* for short) so that parameters can be set individually for each group. For example, all equation macros which define a destination are assigned to the 'eq' destgroup; equation macros which create a link are assigned to the 'eq' linkgroup. By setting parameters for the 'eq' linkgroup (destgroup), you can uniformly customize all links (destinations) related to equation references, without interfering with settings for the other groups.

See Section 5.5 [Setting hyperlink types and options], page 57, for information on how to set parameters for a group.

Here is the list of the linkgroups:

hrefint, hrefext, url, cite, ref, xref, eq, idx, foot, footback.

And here are the destgroups:

bib, li, definexref, xrdef, eq, idx, foot, footback.

See Section 8.1 [Hyperlinks (xhyper.tex)], page 80, for a demonstration of the usage of implicit hyperlinks.

The following subsections describe each of the linkgroups and destgroups and the hyperlink support provided.

### 5.3.1 General hyperlinks: hrefint, hrefext

`\href{`*`url`*`}{`*`text`*`}` typesets *text* as a link to *url*. It basically does what the explicit hyperlink macros do (see Section 5.2 [Explicit hyperlinks], page 40), but is more convenient (at the expense of flexibility).

If *url* starts with ‘`#`’, the rest of *url* is assumed to be a local hyperlink destination name (destination within the same document). Parameters for these links can be set by customizing the ‘hrefint’ linkgroup. For example:

```
See \href{#intro}{Introduction}
```

will make ‘Introduction’ into an internal link, which might have been created, e.g., with `\xrdef{intro}`.

If *url* does not start with ‘`#`’, it is assumed to be a URL link. Parameters for these links can be set by customizing the ‘hrefext’ linkgroup. The special characters (such as ‘`#`’ and ‘`~`’) in the URL don’t need to be escaped. For example:

```
\href{https://tug.org/eplain/doc/eplain.html#Hyperlinks}{Hyperlinks
  in Eplain}
\href{mailto:tex-eplain@tug.org}{Eplain mailing list}
```

See Section 5.3.2 [URL hyperlinks], page 42, for another way to create URL hyperlinks.

The *text* argument of `\href` can contain verbatim text (see Section 4.7 [Verbatim listing], page 12) or other macros which manipulate character catcodes. For example:

```
\href{#WeirdChars}{The weird chars \verbatim #&%$~|endverbatim}
```

`\href` does not currently handle other link types, such as ‘`file:`’ and ‘`run:`’ links.

### 5.3.2 URL hyperlinks: url

The ‘url’ linkgroup covers the `\url` command from the LATEX package `url` (see Section 4.23.3 [Packages known to work], page 36), as well as any new `\url`-like commands you define. The type for this linkgroup is set to ‘`url`’ by the drivers which support this link type. ‘`url`’ links use the parameter to the `\url` command as the URL to point to.

You may be using the `\url` command to typeset something other than a URL, e.g., a path, for which you do not want a link to be created; in that case, you can disable the ‘url’ linkgroup with the command `\hloff[url]` (see Section 5.6.2 [Turning hyperlinks on/off for a group], page 60).

By default, URL (and other types of) links are boxed, so that they are visually marked even if you do not load the LATEX ‘color’ package (see Section 4.23 [Loading LaTeX packages], page 34) and therefore link text is not colored. You can see the effect by compiling the following code snippet (be sure to get a modern `url.sty`, older versions do not work in plain TEX; see Section 4.23.3 [Packages known to work], page 36):

```
\input eplain
\beginpackages
  \usepackage{url}
```

```
\endpackages

\enablehyperlinks

\url{http://foo/bar}

\url{mailto:foobar@example.org}

\bye
```

If the hyperlink driver you use supports the link option `bwidth` (see Section 5.4 [Hyperlink drivers], page 47), you can produce colored links with no border around them. Try this:

```
\input eplain
\beginpackages
  \usepackage{url}
  \usepackage{color}
\endpackages

\enablehyperlinks
\hlopts{bwidth=0}

\url{http://foo/bar}

\url{mailto:foobar@example.org}

\bye
```

The command `\hlopts{bwidth=0}` sets border width to zero as the default for all links, and loading the `color` package automatically colors links using the default color (see Section 5.4.1 [Options supported by all drivers], page 48). If you want the border width setting to apply to URL links only, say `\hlopts[url]{bwidth=0}` (see Section 5.5 [Setting hyperlink types and options], page 57).

### 5.3.3 Citation hyperlinks: cite, bib

The 'cite' linkgroup includes only the `\cite` command (see Section 4.3 [Citations], page 5). `\cite` turns each of the references in the list into a link pointing to the respective bibliography entry produced by the `\bibliography` command.

The 'bib' destgroup includes the macros related to the `\bibliography` command (see Section 4.3 [Citations], page 5). `\bibliography` inputs a `.bbl` file, which contains a list of bibliography entries. For each of the entries, a destination is defined.

Both commands use the citation label as the hyperlink label.

### 5.3.4 List hyperlinks: li

The 'li' destgroup consists of the `\li` command (see Section 4.6 [Lists], page 10), which defines a destination if you supply the optional argument (cross-reference label). This label is also used as the hyperlink label.

### 5.3.5 Cross-reference hyperlinks: definexref, ref

The 'definexref' destgroup is for the `\definexref` command (see Section 4.9.1 [Defining generic references], page 16). `\definexref` defines a destination using the cross-reference label (the first argument) as the hyperlink label.

The 'ref' linkgroup includes `\refn` and `\xrefn` (they are synonyms), `\ref`, and `\refs` (see Section 4.9.2 [Using generic references], page 16).

`\refn` turns the cross-reference it produces into a link, using the cross-reference label as the hyperlink label. If an optional argument is present, it is tied by `\reftie` to the reference and become part of the link.

`\ref` works similarly to `\refn`. It takes an optional argument, which is treated the same way as the optional argument to `\refn`. In addition, `\ref` can produce a "class word". Both the optional argument and the class word become part of the link, when present. The cross-reference is tied by `\reftie` to the preceding word. The optional argument is separated from the class word by `\refspace`.

Unlike `\ref`, `\refs` does not take an optional argument and does not make the class word part of the link, which is appropriate for its intended use.

### 5.3.6 Page reference hyperlinks: xrdef, xref

The 'xrdef' destgroup is for `\xrdef` (see Section 4.10 [Page references], page 16). `\xrdef` defines a destination using cross-reference label as the hyperlink label.

The 'xref' linkgroup includes the `\xref` command (see Section 4.10 [Page references], page 16). `\xref` turns its optional argument (followed by `\refspace`), `\xrefpageword` and the cross-reference (page number) into a link, using the cross-reference label as the hyperlink label.

### 5.3.7 Equation reference hyperlinks: eq

All commands that define equation labels are part of the 'eq' destgroup. These are `\eqdef`, `\eqdefn`, `\eqsubdef` and `\eqsubdefn` (see Section 4.11 [Equation references], page 17). All these commands use the equation label as the hyperlink label. However, if the equation label is empty, they make up a (hopefully) unique hyperlink label for the destination. This label will be used for the link when you refer to this empty-labeled equation with one of the equation reference macros.

The command `\phantomeqlabel` is called to generate hyperlink labels for the empty-labeled equations. By default, it produces the labels in the format '`PHEQ`*number*', where *number* comes from the count register `\phantomeqnumber`; this count register is incremented at every empty-labeled equation definition.

The commands `\eqref` and `\eqrefn` (see Section 4.11 [Equation references], page 17) form the 'eq' linkgroup. These commands take an optional argument, which, when present, is tied with `\reftie` to the equation reference and becomes part of the link. The equation label is used for the hyperlink label; if the label is empty, the link is for the label generated for the last empty-labeled equation.

### 5.3.8 Index hyperlinks: idx

All indexing commands (`\idx`, `\idxname`, `\idxmarked`, `\idxsubmarked` and their silent equivalents, see Section 4.12.1.1 [Indexing commands], page 21) form the 'idx' destgroup.

The ‘idx’ linkgroup consists of the macros which are used to typeset the index when you say `\readindexfile{`*`index-letter`*`}` (see Section 4.12.2 [Typesetting an index], page 25).

To create the links in index entries, Eplain uses MakeIndex's “encapsulation” feature. When you use an indexing macro to mark an index term, Eplain writes out a line to the `.idx` file of the following general form:

```
\indexentry{entry|pagemarkup}{pageno}
```

where *entry* is the index entry (converted into the internal format that MakeIndex understands), *cs* is the markup command you specified with the `pagemarkup=`*`cs`* optional argument to the indexing commands (see Section 4.12.1.2 [Modifying index entries], page 22), and *pageno* is the page number on which the term appeared. When processing the `.idx` file, MakeIndex makes the page number an argument to the page markup command (“encapsulates” the page number), so the page number in the `.ind` file appears as `\`*`cs`*`{`*`pageno`*`}`. Eplain internally replaces the *cs* command name with its own command, which, in addition to calling the original `\`*`cs`* encapsulator, turns the page number into a link.

Eplain provides two approaches to linking page numbers in the index to locations of index terms in the text.

#### 5.3.8.1 Exact destinations for index terms

In this approach, each command that marks an index term defines a unique destination and passes its label on to the `.idx` file as part of the `\indexentry` command. The `\indexentry` line that Eplain writes to the `.idx` file becomes

```
\indexentry{entry|hlidx{label}{cs}}{pageno}
```

where `\hlidx` is the command that is defined by Eplain to take three arguments: a hyperlink label (*label*), a name of page number encapsulator (*cs*) and a page number (*pageno*). In the `.ind` file that MakeIndex will generate, the page number will now appear as

```
\hlidx{label}{cs}{pageno}
```

The result of this command is `\`*`cs`*`{`*`pageno`*`}`, wrapped up into a link pointing to *label* destination.

The hyperlink labels for the index terms are generated by the `\hlidxlabel` command, by default in the format ‘`IDX`*`number`*’, where *number* is the value of the count register `\hlidxlabelnumber`. This count register is incremented at each index term.

The advantage of this approach, as compared to the second approach described below, is that links in the index point to exact locations of the indexed terms on the page. The disadvantage of this approach is that MakeIndex will regard *all* index entries as distinct, because each one contains a (unique) hyperlink label. This disadvantage can be partially overcome by the script `idxuniq` distributed with Eplain in the `util` directory. This script filters out `\indexentry` lines differing only in the hyperlink label but identical otherwise. You should process the `.idx` with this script before passing it on to MakeIndex. For example:

```
prompt$ ./idxuniq file.idx | makeindex > file.ind
```

Still, this solution is not ideal, as the page-range formation ability of MakeIndex will not work, and there will be problems of apparently identical index entries clashing (e.g., when a range-end entry appears on the same page as another entry with the same definition; `idxuniq` will not filter out the second entry).

#### 5.3.8.2 Page destinations for index terms

In the second approach, Eplain does not write out any destination labels for the index terms. Instead, Eplain writes out a wrapper for page number encapsulator which can parse the page number and generate a link pointing to the *page* on which the term appeared. On top of each page containing an index term, Eplain defines a destination with label produced by `\hlidxpagelabel`. The `\hlidxpagelabel` command takes a single argument (page number *number*) and by default produces the label in the format '`IDXPG`*`number`*'.

With this approach, the `\indexentry` line which Eplain writes to the `.idx` file looks like this:

```
\indexentry{entry|hlidxpage{cs}}{pageno}
```

where `\hlidxpage` is the command that is defined by Eplain to take two arguments: a name of page number encapsulator (*cs*) and a page number (*pageno*). In the `.ind` file that MakeIndex will generate, the page number will appear as

```
\hlidxpage{cs}{pageno}
```

The advantage of this approach is that all features of MakeIndex are intact. The drawback is that links in the index do not point to exact locations of indexed terms on a page, but to the top of a page on which the term appears.

Another disadvantage is that this approach depends on the page range and page list separators which MakeIndex was configured to output. `\hlidxpage` must be able to parse the first page number in a page range like '`1--4`'. In addition, page list parsing is needed because MakeIndex combines two consecutive page numbers in one call to the page number encapsulator, so `\hlidxpage` can be passed, e.g., '`1, 2`' for the *pageno*. In this last case, `\hlidxpage` splits the two page numbers, applies `\`*`cs`* to each of them, and makes each of the page numbers a link to the appropriate page. Note that this will alter typesetting slightly, because now the page list separator (a comma followed by a space, by default) is not typeset using the page number encapsulator (`\`*`cs`*).

Eplain's defaults for the page list and page number delimiters are the same as those in MakeIndex, a comma followed by a space ('`, `') and two dashes ('`--`'), respectively. If you customize MakeIndex to use different delimiters, you must not forget to let Eplain know about them with the commands

```
\setidxpagelistdelimiter{list-delim}
\setidxpagerangedelimiter{page-delim}
```

(see Section 7.12 [Page list and page range parsers], page 78).

#### 5.3.8.3 Choosing destination placement

The approach that Eplain should use for the index terms can be selected in the `\enablehyperlinks` command. The optional argument it accepts is a comma-separated list of options. The `idxexact` option selects the first approach, `idxpage` the second, and `idxnone` disables hyperlink support for the index terms altogether, in case you want to stop Eplain from writing its link wrappers into the `.idx` file. The default is `idxpage`.

For example:

```
\enablehyperlinks[idxexact]
```

selects the first approach ("exact index links").

#### 5.3.8.4 Index page list and page range parsers

The macros that Eplain uses to parse page lists and page ranges, `\idxparselist` and `\idxparserange`, can sometimes be useful when defining page number encapsulators. See Section 7.12 [Page list and page range parsers], page 78, for the description of these commands and an example of their usage.

#### 5.3.8.5 Hyperlinks in see and see also entries

There is no automatic support for hyperlinks with "see" and "see also" index entries, as there is not enough information to trace the parameters of `\indexsee` and `\indexseealso` to corresponding index entries. But if desired, this can be implemented with `\hldest` and `\hlstart` (see Section 5.2 [Explicit hyperlinks], page 40); for example:

```
\sidx{semantic theory of truth@%
      \leavevmode\hldest{}{}{idx:theo truth}semantic theory of truth}
...
\sidx[seealso]{truth}[definition of]%
      {\hlstart{}{}{idx:theo truth}semantic theory of truth\hlend}
```

### 5.3.9 Footnote hyperlinks: foot, footback

The 'foot' link and destination groups include the `\numberedfootnote` and `\footnote` macros (see Section 4.17 [Footnotes], page 31). The 'footback' groups include the same macros, but control parameters for links and destinations created inside the footnote to point back to the footnote mark within the text body.

The macros use hyperlink labels generated by `\hlfootlabel` and `\hlfootbacklabel`. The default formats for the labels are '`FOOT`*`number`*' and '`FOOTB`*`number`*', respectively, where *number* is the value of the count register `\hlfootlabelnumber`. This register is incremented at every footnote.

Generally, footnote hyperlinks are not of much use, because the footnotes are usually placed on the same page as the footnote mark. Therefore, footnote hyperlinks are disabled by default. Here is how you can turn them on selectively, without affecting the other kinds of hyperlinks (see Section 5.6.2 [Turning hyperlinks on/off for a group], page 60):

```
\hldeston[foot,footback]
\hlon[foot,footback]
```

### 5.3.10 Contents hyperlinks

There is currently no special support for hyperlinks in the table of contents (see Section 4.8 [Contents], page 13), but implementing them with the `\hldest` and `\hlstart ... \hlend` commands (see Section 5.2 [Explicit hyperlinks], page 40) should be possible.

## 5.4 Hyperlink drivers

This section describes the hyperlink drivers: the types of hyperlinks they support, and the options they accept. During the first reading, you may only want to skim through this section.

Some of the descriptions below come from *Portable Document Format Reference Manual Version 1.3*, March 11, 1999.

### 5.4.1 Options supported by all drivers

This subsection describes the destination and link options which are supported by all hyperlink drivers.

### Destination options supported by all drivers

`raise` Specifies how much to raise destinations above the baseline. When set to zero or empty, destinations are placed at the baseline.

It is usually convenient to set this option to some variable parameter, so that the height to which destinations are raised is automatically adjusted according to the current context. For example, setting it to `\normalbaselineskip` (or some fraction of it, like `1.7\normalbaselineskip`) makes the setting appropriate for different point sizes, in case your document uses more than one.

The default setting is `\normalbaselineskip`. Initially, the destgroups do not define this option, so they fall back on the default, except for the 'eq' destgroup, for which this option is set to `1.7\normalbaselineskip`, to accommodate the usual cases of large operators in displayed math.

Example: `\hldestopts[eq]{raise=2.5\normalbaselineskip}`

### Link options supported by all drivers

`colormodel`
`color` These two options define the color to be used for rendering the link text. The colors are used only when a `\color` command is defined, e.g., by loading the LATEX 'color' package (see Section 4.23.3 [Packages known to work], page 36). The `\color` command is called as `\color[`*`colormodel`*`]{`*`color`*`}`, where *colormodel* and *color* are the definitions of the `colormodel` and `color` options, respectively. However, if *colormodel* is empty, the optional argument to `\color` is omitted; and if *color* is empty, the `\color` command is omitted altogether. The default setting is *`colormodel`*`=cmyk` and *`color`*`=0.28,1,1,0.35`.

When specifying colors with several components delimited by commas (e.g., RGB and CMYK colors in the LATEX 'color' package), it is not possible to specify the components directly in the option list of `\hlopts`, because comma is the option list delimiter. With the 'color' package, it is possible to specify such colors by defining a custom color with `\definecolor` and using the new color name with an empty *colormodel* (see examples below).

Examples:

```
\hlopts{colormodel=,color=blue}% predefined color
\definecolor{mycolor}{rgb}{.3,.8,.95}
\hlopts{colormodel=,color=mycolor}% custom color
\hlopts{colormodel=gray,color=.4}
```

### 5.4.2 Hyperlink driver `hypertex`

HyperTEX is a standard for inclusion of hyperlink information in TEX (and LATEX) documents (see `http://arxiv.org/hypertex`). This standard defines a set of hyperlink tags implemented as `\special` commands written into the DVI file. The major advantage of such standard is that a single `.dvi` file containing HyperTEX commands can be viewed by any HyperTEX-enabled viewer (e.g., any more or less modern version of `xdvi`) or converted to other file formats (e.g., PDF) by any HyperTEX-enabled DVI converter (e.g., `dvipdfmx` or `dvips` with Ghostscript's `ps2pdf` script).

The downside to the standard is that it is by design "the common factor" of other formats supporting hyperlinks, so many features of a particular file format cannot be supported by HyperTEX. Therefore, if you need to use special features of a particular format, HyperTEX is not a good choice. For the PDF file format, Eplain provides the `pdftex` and `dvipdfm` drivers which provide fine control over the PDF options (see Section 5.4.3 [Hyperlink drivers pdftex and dvipdfm], page 51).

For more information on programs which support the HyperTEX standard, please see

`http://arxiv.org/hypertex`
`http://mirror.ctan.org/support/hypertex/hypertex`

For convenience, we list a few HyperTEX-enabled converters:

`dvips` Note that you need to pass the `-z` option to `dvips` to tell it to preserve the information about the hyperlinks. To generate a `.pdf` file with hyperlinks, you can use the `ps2pdf` script of Ghostscript. For example, if `foo.tex` is a TEX file using HyperTEX commands, then

```
prompt$ tex foo.tex
prompt$ dvips -z foo.dvi -o
prompt$ ps2pdf foo.ps
```

will produce `foo.pdf` with hyperlinks.

`dvipdfm`
`dvipdfmx` No special command line arguments are required, these programs automatically detect the HyperTEX commands.

One more note is in place: support for the HyperTEX commands varies from one previewer/converter to another, so you might need to experiment to see if what you need is supported by the program(s) you intend to use. For example, dvips(k) as of version 5.92b converts all internal hyperlinks into page links pointing to a page containing the destination, so that "exact" linking to destinations is not possible (this was confirmed to be a bug and most probably has already been fixed in later versions of dvips(k)); dvipdfm as of version 0.13.2c and dvipdfmx as of version 20040411 do not correctly parse links to external local files, and produce a URL link instead of a file link.

#### 5.4.2.1 Destination types for `hypertex`

`xyz` "Current position". This is the default type.

Example: `\hldest{xyz}{}{dest123}`

`raw` The destination specification (in the form of a HyperTEX `\special`) is taken from the command sequence `\`*`cs`*, where *cs* is the value of the `cmd` option. In the definition of `\`*`cs`*, use `\@hllabel` to refer to the hyperlink label. This option is intended to be used with destgroups (see Section 5.5 [Setting hyperlink types and options], page 57), as it does not make sense in a direct call to `\hldest`—you can just call the raw command.

Example:

```
\makeatletter
\def\mydest{\special{html:<a name="\@hllabel">}%
            \special{html:</a>}}
\resetatcatcode
\indent\hldest{raw}{cmd=mydest}{SpecialDest}
  Special destination.
```

#### 5.4.2.2 Destination options for `hypertex`

| | |
|---|---|
| `cmd` | Name of the macro (without the leading '\') containing a HyperTEX `\special` for the `raw` destination.<br>See Section 5.4.2.1 [Destination types for hypertex], page 49, the description of the `raw` destination, for an example. |

#### 5.4.2.3 Link types for `hypertex`

`name` Go to a "named destination". The label is the destination name. All destinations in HyperTEX are named destinations. This is the default type.

Example: `\hlstart{name}{}{dest123}Link to dest123\hlend`

`url` Go to a URL. The label is the URL.

Example:

```
\hlstart{url}{}{https://tug.org/eplain/}Eplain home\hlend
```

`filename` Go to a named destination in another file. The label is the destination name. The file name is specified by the `file` option. The file name extension can be specified separately by the `ext` option. The idea is to set the `ext` option globally at the beginning of the document to avoid hard-coding the extension together with the file name within each link—HyperTEX is not restricted to any single output format, it can be DVI, PDF, possibly other formats.

Example:

```
\hlopts{ext=.pdf}
\hlstart{filename}{file=book}{dest123}
  Link to dest123 in file 'book.pdf'\hlend
```

`raw` The link specification (in the form of a HyperTEX `\special`) is taken from the command sequence `\`*cs*, where *cs* is the value of the `cmd` option. In the definition of `\`*cs*, use `\@hllabel` to refer to the hyperlink label. Use the predefined command `\hlhash` to insert the `#` characters. This option is intended to be used with linkgroups (see Section 5.5 [Setting hyperlink types and options], page 57), as it does not make sense in a direct call to `\hlstart`—you can just call the raw command.

Example:

```
\makeatletter
\def\mylink{\special{html:<a href="\hlhash\@hllabel">}}
\resetatcatcode
\hlstart{raw}{cmd=mylink}{SpecialDest}
  Link to the special destination.\hlend
```

### 5.4.2.4 Link options for `hypertex`

`cmd` Name of the macro (without the leading '\') containing a HyperTEX `\special` for the `raw` link.

See Section 5.4.2.3 [Link types for hypertex], page 50, the description of the `raw` link, for an example.

`file` File name for the `filename` link type. See also the `ext` option.

See Section 5.4.2.3 [Link types for hypertex], page 50, the description of the `filename` link, for an example.

`ext` File name extension for the `filename` link type. The idea is to set the `ext` option globally at the beginning of the document to avoid hard-coding the extension together with the file name within each link—HyperTEX is not restricted to any single output format, it can be DVI, PDF, possibly other formats.

See Section 5.4.2.3 [Link types for hypertex], page 50, the description of the `filename` link, for an example.

## 5.4.3 Hyperlink drivers `pdftex` and `dvipdfm`

This subsection describes link and destination types and options supported by the `pdftex` and `dvipdfm` drivers. Many of the hyperlink types and options are common to both drivers, so we describe them together.

### 5.4.3.1 Destination types for `pdftex` and `dvipdfm`

`xyz` "Current position". The option `zoom` specifies magnification to use (zero or empty means leave magnification unchanged, which is the default). This is the default type.

For `dvipdfm`: the options `left` and `top` specify position coordinates to use (empty options mean current position coordinate, which is the default).

Example: `\hldest{xyz}{zoom=2000}{dest123}`

`fit` Fit the page to the window.

Example: `\hldest{fit}{}{dest123}`

`fith` Fit the width of the page to the window.

For `dvipdfm`: the `top` option specifies vertical position (default is empty, meaning current position).

Example: `\hldest{fith}{}{dest123}`

`fitv` Fit the height of the page to the window.

For `dvipdfm`: The option `left` specifies horizontal position (default is empty, meaning current position).

Example: `\hldest{fitv}{}{dest123}`

`fitb` Fit the page's bounding box to the window.

Example: `\hldest{fitb}{}{dest123}`

`fitbh` Fit the width of the page's bounding box to the window.

For `dvipdfm`: the option `top` specifies vertical position (default is empty, meaning current position).

Example: `\hldest{fitbh}{}{dest123}`

`fitbv` Fit the height of the page's bounding box to the window.

For `dvipdfm`: the option `left` specifies horizontal position (default is empty, meaning current position).

Example: `\hldest{fitbv}{}{dest123}`

`fitr` For `pdftex`: fit the rectangle specified by the options `width`, `height` and `depth` (as a TEX rule specification) in the window. For dimensions set to empty, the corresponding value of the parent box is used (these are the defaults).

For `dvipdfm`: fit the rectangle specified by the options `left`, `bottom`, `right` and `top` (in PostScript points, 72 points per inch) in the window. For dimensions set to empty, current position coordinate is substituted (these are the defaults).

Example for `pdftex`:

```
\hldest{fitr}{width=\hsize,
              height=.5\vsize,depth=0pt}{dest123}
```

Example for `dvipdfm`:

```
\hldest{fitr}{left=72,bottom=72,
              right=720,top=360}{dest123}
```

`raw` The destination specification (in the form of a pdfTEX command or a dvipdfm `\special`) is taken from the command sequence `\`*cs*, where *cs* is the value of the `cmd` option. In the definition of `\`*cs*, use `\@hllabel` to refer to the hyperlink label. This option is intended to be used with destgroups (see Section 5.5 [Setting hyperlink types and options], page 57), as it does not make sense in a direct call to `\hldest`—you can just call the raw command.

Example for `pdftex`:

```
\makeatletter
\def\mydest{\pdfdest name{\@hllabel} xyz}
\resetatcatcode
\hldesttype{raw}
\hldestopts{cmd=mydest}
```

Example for `dvipdfm`:

```
\makeatletter
\def\mydest{\special{pdf: dest (\@hllabel)
                          [@thispage /XYZ @xpos @ypos 0]}}
\resetatcatcode
\hldesttype{raw}
\hldestopts{cmd=mydest}
```

#### 5.4.3.2 Destination options for `pdftex` and `dvipdfm`

With respect to the destination options, the `pdftex` and `dvipdfm` differ in the way the fit rectangle is specified (relative coordinates for `pdftex`, absolute coordinates for `dvipdfm`).

### Common destination options

| | |
|---|---|
| `cmd` | Name of the macro (without the leading '\') containing a pdfTeX command or a dvipdfm `\special` for the `raw` destination.<br>See Section 5.4.3.1 [Destination types for pdftex and dvipdfm], page 51, the description of the `raw` destination, for an example. |
| `zoom` | Magnification ratio times 1000 (like TeX's scale factor). Zero or empty means leave magnification unchanged, which is the default.<br>Example: `\hldest{xyz}{zoom=2000}{dest123}` |

### `pdftex`-specific destination options

The dimension options below must be specified as a TeX rule specification. When set to empty, the corresponding value of the parent box is used (this is the default for all dimension options).

| | |
|---|---|
| `depth` | Depth of the fit rectangle for the `fitr` destination. |
| `height` | Height of the fit rectangle for the `fitr` destination. |
| `width` | Width of the fit rectangle for the `fitr` destination.<br>Example: |

```
\hldest{fitr}{width=\hsize,
              height=.5\vsize,depth=0pt}{dest123}
```

### `dvipdfm`-specific destination options

The dimension options below must be specified in PostScript points (72 points per inch), as a number without the '`bp`' unit name. When set to empty, the current position coordinate is used (this is the default for all dimension options).

| | |
|---|---|
| `bottom` | Bottom position coordinate of a box specification for the various destination types. |
| `left` | Left position coordinate of a box specification for the various destination types. |
| `right` | Right position coordinate of a box specification for the various destination types. |
| `top` | Top position coordinate of a box specification for the various destination types.<br>Example: |

```
\hldest{fitr}{left=72,bottom=72,
              right=720,top=360}{dest123}
```

### 5.4.3.3 Link types for `pdftex` and `dvipdfm`

Link types are the same for the `pdftex` and `dvipdfm` drivers, except that the `pdftex` driver provides one additional link type `num` (link to a numbered destination). dvipdfm does not support numbered destinations, therefore it does not have this link type. Note that all destinations created by Eplain hyperlink macros are named destinations; to define a numbered destination, you have to use low-level pdfTeX commands.

### Common link types

`name`

Go to a “named destination”. The label is the destination name. All destinations created with `\hldest` are named destinations (see Section 5.2 [Explicit hyperlinks], page 40). This is the default type.

Example: `\hlstart{name}{}{dest123}Link to dest123\hlend`

`url`

Go to a URL. The label is the URL.

Example: `\hlstart{url}{}{https://tug.org/eplain/}Eplain home\hlend`

`page`

Go to a page. The label is the page number (counting from 1). Page fitting is specified by the `pagefit` option.

Example:

```
\hlstart{page}{pagefit=/FitH 600}{123}
  Link to page~123\hlend
```

`filename`

Go to a named destination in another file. The label is the destination name. The file name is specified by the `file` option. Page fitting is specified by the `pagefit` option. The `newwin` option specifies whether the destination document is opened in the same window or in a new window.

Example:

```
\hlstart{filename}{file=book.pdf,newwin=1}{dest123}
  Link to dest123 in file ‘book.pdf’\hlend
```

`filepage`

Go to a page in another file. The label is the page number (counting from 1). The file name is specified by the `file` option. Page fitting is specified by the `pagefit` option. The `newwin` option specifies whether the destination document is opened in the same window or in a new window.

Example:

```
\hlstart{filepage}{file=book.pdf,newwin=,
                   pagefit=/FitR 50 100 300 500}{1}
  Link to page~1 in file ‘book.pdf’\hlend
```

`raw`

The link specification (in the form of a pdfTeX command or a dvipdfm `\special` primitive) is taken from the command sequence `\`*cs*, where *cs* is the value of the `cmd` option. In the definition of `\`*cs*, use `\@hllabel` to refer to the hyperlink label. This option is intended to be used with linkgroups (see Section 5.5 [Setting hyperlink types and options], page 57), as it does not make sense in a direct call to `\hlstart`—you can just call the raw command.

Example for `pdftex`:

```
% Redirect all \url links to the first page
\def\mycmd{\pdfstartlink goto page 1 {/Fit}}
\hltype[url]{raw}
\hlopts[url]{cmd=mycmd}
```

Example for `dvipdfm`:

```
% Redirect all \url links to the first page
\def\mycmd{\special{pdf: beginann <</Type/Annot /Subtype/Link
```

```
                                        /Dest[0 /Fit]>>}}
          \hltype[url]{raw}
          \hlopts[url]{cmd=mycmd}
```

### `pdftex`-specific link types

`num` Go to a “numbered destination”. The label is the destination number.

Example: `\hlstart{num}{}{123}Link to 123\hlend`

### 5.4.3.4 Link options for `pdftex` and `dvipdfm`

Link options are mostly the same for the `pdftex` and `dvipdfm` drivers. The `pdftex` driver has additional options to specify link dimensions.

### Common link options

`bcolor` Border color. An array of three numbers in the range 0 to 1, representing a color in DeviceRGB.

Example: `\hlstart{name}{bcolor=.1 .5 1}{dest123}Link\hlend`

`bdash` Array of numbers representing on and off stroke lengths for drawing dashes.

Example: `\hlstart{name}{bstyle=D,bdash=2 4}{dest123}Link\hlend`

`bstyle` Link border style:

- `S` The border is drawn as a solid line.
- `D` The border is drawn with a dashed line (the dash pattern is specified by the `bdash` option).
- `B` The border is drawn in a beveled style.
- `I` The border is drawn in an inset style.
- `U` The border is drawn as a line on the bottom of the link rectangle.

The default is ‘`S`’.

Example: `\hlstart{name}{bstyle=D,bdash=2 4}{dest123}Link\hlend`

`bwidth` Border width in PostScript points (72 points per inch). The default is 1.

Example: `\hlstart{name}{bwidth=2}{dest123}Link\hlend`

`cmd` Name of the macro (without the leading ‘\’) containing a pdfTEX command or a dvipdfm `\special` for the `raw` link.

See Section 5.4.3.3 [Link types for pdftex and dvipdfm], page 53, the description of the `raw` link, for an example.

`file` File name for the `filename` and `filepage` link types.

See Section 5.4.3.3 [Link types for pdftex and dvipdfm], page 53, the descriptions of the `filename` and `filepage` links, for an example.

`hlight` Link border highlight modes:

- `I` The rectangle specified by the bounding box of the link is inverted.

| | |
|---|---|
| `N` | No highlighting is done. |
| `O` | The border of the link is inverted. |
| `P` | The region underneath the bounding box of the link is drawn inset into the page. |

The default is ‘`I`’.

Example: `\hlstart{name}{bstyle=S,hlight=O}{dest123}Link\hlend`

`newwin` For the `filename` and `filepage` links, specifies whether the destination document is opened in the same window or in a new window. The settings are:

| | |
|---|---|
| `0` | Open in the same window. |
| non-`0` | Open in a new window. |
| empty | Behavior according to the viewer settings. |

The default is empty.

See Section 5.4.3.3 [Link types for pdftex and dvipdfm], page 53, the descriptions of the `filename` and `filepage` links, for an example.

`pagefit` For the `page` and `filepage links`, specifies how the page must be fitted to the window. `pagefit` specification is written to the PDF file as is, so it must conform to the PDF standard.

See Section 5.4.3.3 [Link types for pdftex and dvipdfm], page 53, the descriptions of the `page` and `filepage` links, for an example.

### `pdftex`-specific link options

The dimension options below must be specified as a TEX rule specification. When set to empty, the corresponding value of the parent box is used (this is the default for all dimension options).

| | |
|---|---|
| `depth` | Depth of the link. |
| `height` | Height of the link. |
| `width` | Width of the link. |

Example:

```
\hlstart{name}{width=5in,height=20pc,depth=0pt}{dest123}
  Link\hlend
```

### 5.4.4 Hyperlink driver `nolinks`

Select this driver to suppress all hyperlinks in your document.

Selecting this driver is quite different from not selecting any driver at all, or from selecting some driver and then turning hyperlinks off for the entire document with `\hloff` and `\hldestoff` (see Section 5.6 [Turning hyperlinks on/off], page 59).

The purpose of `\hldestoff` and `\hloff` is to mark (parts) of your document where hyperlinks should never appear. (Imagine you want to prevent a cross-referencing macro from generating a link at a certain spot in your document.)

If instead you have prepared a document with hyperlinks and just want to compile a version without them, it is better to select the driver `nolinks`. This ensures that spacing and page-breaking are the same as what you were getting with hyperlinks enabled.

The reason for this is that hyperlinks are produced by the `\special` primitives or low-level hyperlink commands. Each such command is placed inside a *whatsit* (an internal TeX object), which may introduce legitimate breakpoints at places where none would exist without the whatsits. The macros `\hldestoff` and `\hloff` disable the hyperlink macros completely, so that no whatsits are produced.

In contrast, the `nolinks` driver does not completely disable hyperlink macros. Instead, it defines them to merely write to the log file (what gets written is unimportant). This also produces whatsits, thus imitating the whatsits from the hyperlink commands. (This trick was borrowed from the LaTeX 'color' package.)

Another reason for using `nolinks` is that in horizontal mode `\hldest` places destinations inside boxes of zero width, height, and depth. When you say `\hldestoff`, `\hldest` will omit both destination definitions and these boxes. The missing boxes can again cause the typesetting to be inconsistent with that with destinations enabled. Here again, the `nolinks` driver helps by defining `\hldest` to produce the empty boxes.

So, if you are planning to produce versions of your PDF document both with and without hyperlinks, here is the recommended way to enable the hyperlinks under pdfTeX:

```
\ifpdf
  \enablehyperlinks
\else
  \enablehyperlinks[nolinks]%
\fi
```

## 5.5 Setting hyperlink types and options

You can define default types for links and destinations, which will be used when you do not specify a type in `\hlstart` or `\hldest`. Similarly, you can define default values for the options; the default value for an option is used when you do not set the option in the argument to `\hlstart` or `\hldest`.

The parameters for implicit links and destinations can be customized by setting the "group" types and options. When not set, the defaults are used.

All these settings are local to the current (TeX) group, so if you want to set an option temporarily, you can do so inside a `\begingroup...\endgroup` block; when the group ends, the previous settings are restored.

### 5.5.1 Setting default types and options

The default types for links and destinations can be set with the following commands:

```
\hltype{type}
\hldesttype{type}
```

where *type* is one of the link/destination types supported by the selected hyperlink driver (see Section 5.4 [Hyperlink drivers], page 47).

Default values for the options can be set with the following commands:

```
\hlopts{options}
```

```
\hldestopts{options}
```

where *options* is a comma-separated list of option assignments in the format '*option=value*'. Again, what options are allowed depends on the selected hyperlink driver.

Many people regard the default boxed links as not aesthetic and intruding on page layout. The reason why boxed links are the default is that the links are not colored until you load the LATEX 'color' package (see Section 4.23 [Loading LaTeX packages], page 34) or use other means to define the `\color` command; therefore, not producing any kind of link border may result in the links not being marked in any way. However, when the links are actually colored, there is no need to produce the link boxes anymore; to avoid the boxes, you can set the default border width to zero (if the driver you use supports the link option `bwidth`; see Section 5.4 [Hyperlink drivers], page 47):

```
\hlopts{bwidth=0}
```

### 5.5.2 Setting group types

When called with an optional argument, as in

```
\hltype[groups]{type}
\hldesttype[groups]{type}
```

where *groups* is a comma-separated list of groups, `\hltype` and `\hldesttype` set the type for each group from *groups* to *type*. The default type is used for all groups with an empty type (this is the initial setting for all groups, except that the type for the 'url' linkgroup is set to '`url`' by the drivers which support this link type).

There are two special "groups" which can be used inside the *groups* list. An empty group sets the default type. This allows to set the default type and group types in one command, for example:

```
\hltype[eq,]{type}
```

sets the link type for the 'eq' linkgroup and the default link type to *type*.

Another special group is a star ('`*`') group, which signifies all defined groups. For example, the command

```
\hldesttype[*,]{type}
```

sets the destination type to *type* for each group, as well as the default destination type.

### 5.5.3 Setting group options

Option values for each group are stored as a list of option assignments. This list does not have to contain every supported option. Values for options missing from this list are taken from the default option values.

To manipulate the list of option values for the groups, you use the `\hlopts` and `\hldestopts` commands with an optional argument:

```
\hlopts[groups]{options}
\hldestopts[groups]{options}
\hlopts![groups]{options}
\hldestopts![groups]{options}
```

where *groups* is a comma-separated list of groups and *options* is a comma-separated list of option assignments. The two special "groups", the empty group and the star ('`*`') group, have the same meaning as for `\hltype` and `\hldesttype`. When used without the exclamation mark, `\hlopts` and `\hldestopts` preserve the current list of options for the groups, and only update the options listed in *options*. If you add the exclamation mark, the current list of options for each group in *groups* is discarded and the new list is set to *options*.

The "overriding" nature of the '`!`' is appropriate when you give a complete specification of the options for a group, e.g., at the beginning of your document. On the other hand, when you want to adjust some option(s) and leave others intact, you should use the macros without the '`!`'.

For example, with displayed mathematical formulas, you often need to adjust the '`raise`' option for the 'eq' destgroup, because the formulas often contain large parentheses and brackets. But when doing so, you want to leave the other settings unchanged. To achieve this, call `\hldestopts` without the '`!`', for example:

```
$$\hldestopts[eq]{raise=2.5\normalbaselineskip}
...
$$
```

The display commands ('`$$`') implicitly put the entire formula inside a (TEX) group (`\begingroup...\endgroup`), so you do not need to isolate the setting of the '`raise`' option—it will be restored after the closing '`$$`'.

Initially, Eplain sets the option lists for almost all groups to empty, so that the groups fall back on the default values for all options. The one exception to this rule is the 'eq' destgroup, whose initial option list contains one setting:

```
raise=1.7\normalbaselineskip
```

This setting usually accommodates the large operators, which often appear in displayed math.

## 5.6 Turning hyperlinks on/off

Links and/or destinations can be turned on or off globally by disabling the low-level commands, or for each group individually.

All these settings are local to the current (TEX) group, so if you want to enable or disable links/destinations temporarily, you can do so inside a `\begingroup...\endgroup` block; when the group ends, the previous settings are restored.

### 5.6.1 Turning low-level commands on/off

The low-level commands `\hlstart`, `\hlend` and `\hldest` can be turned on/off with the following commands:

```
\hldeston
\hldestoff
\hlon
\hloff
```

See Section 5.4.4 [Hyperlink driver nolinks], page 56, for the implications of using these commands to disable hyperlinks for the entire document.

### 5.6.2 Turning hyperlinks on/off for a group

If you want to disable links/destinations produced by certain groups, you can give a list of the groups as an optional argument to these commands:

```
\hldeston[groups]
\hldestoff[groups]
\hlon[groups]
\hloff[groups]
```

where *groups* is the list of linkgroups/destgroups. This list can contain two special groups. The empty group switches the low-level commands (see Section 5.6.1 [Turning low-level commands on/off], page 59), and the star (‘`*`’) group operates on all defined groups.

Note that turning off the low-level commands disables all hyperlinks globally, including groups which have them enabled. Turning off certain linkgroups/destgroups records the fact that the macros in the group should not produce links/destinations. To illustrate the distinction, assume that all links are on; after the following sequence of commands:

```
\hloff
\hloff[eq]
\hlon
```

all links are on except for the ‘eq’ linkgroup.

## 5.7 Making PDF outlines

PDF outlines (a.k.a. bookmarks) are more or less a table of contents that PDF viewers can display alongside the main document. Eplain’s hyperlink features can be used to create them; there isn’t any special support for them. A continuing example interspersed with commentary follows.

First we must enable hyperlinks.

```
\input eplain
\enablehyperlinks %[dvipdfm] doesn't work
```

We will separate the code to support `pdftex` from `dvips` with the `\ifpdf` conditional (provided by Eplain).

For `pdftex`, we can use the `\pdfoutline` primitive. The keyword `"count"` is followed by the number of subentries in this entry. If negative, the bookmark is closed (that is, subentries are hidden).

```
\ifpdf
  \pdfoutline goto name {sec1} count -1 {Mysec-pdf}%
  \pdfoutline goto name {sec1.1} {Mysubsec-pdf}%
```

For `dvips`, we use TEX’s `\special` command to emit a `ps:` special using the PDF `pdfmark` operator. The `ps:` prefix tells `dvips` that the following is literal PostScript.

`[ ... pdfmark` (there is no closing ‘`]`’) is a extension to the PostScript language for specifying various PDF-related things. It is recognized by Ghostscript, Distiller, et al. Adobe publishes a reference manual for it: `https://adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdfmark_reference.pdf`.

The `/DOCVIEW` pdfmark used here says the outline panel should be used.

```
\else % not pdf output
```

```
  \special{ps:[/PageMode /UseOutlines /DOCVIEW pdfmark}
  %
  % The individual outline entries, using a different syntax
  % than pdftex, but the same information.
  \special{ps:[/Count -1 /Dest (sec1) cvn /Title (Mysec-dvi)
              /OUT pdfmark}
  \special{ps:[/Count -0 /Dest (sec1.1) cvn /Title (Mysubsec-dvi)
              /OUT pdfmark}
\fi
```

The '`-pdf`' and '`-dvi`' suffixes in the strings above in the outline entries are just to make it clear which branch is being executed, for purposes of this example. Ordinarily the entries would be the same in both branches.

Also, the strings above are literal PostScript constants, again for this example. Usually they would come from control sequences, e.g., as the table of contents is read.

It is necessary to "pdf-escape" such arbitrary strings, else backslashes, parentheses, etc., would not come out right. pdfTEX's `\pdfescapestring` primitive is an easy way to do this, e.g., `\xdef#1{\pdfescapestring{#1}}`.

Here is the document text, constructing three pages with the section and subsection given above in the outlines.

```
First page.\vfil\eject

\hldest{}{}{sec1}%
1. Mysec on second page.\vfil\eject

\hldest{}{}{sec1.1}%
1.1. Mysubsec on third page.

\end
```

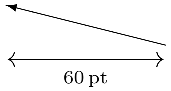
# 6 Arrow theoretic diagrams

This chapter describes definitions for producing commutative diagrams.

Steven Smith wrote this documentation (and the macros).

## 6.1 Slanted lines and vectors

The macros `\drawline` and `\drawvector` provide the capability found in LaTeX's picture mode to draw slanted lines and vectors of certain directions. Both of these macros take three arguments: two integer arguments to specify the direction of the line or vector, and one argument to specify its length. For example, ‘`\drawvector(-4,1){60pt}`’ produces the vector



which lies in the 2d quadrant, has a slope of minus 1/4, and a width of 60 pt.

Note that if an `\hbox` is placed around `\drawline` or `\drawvector`, then the width of the `\hbox` will be the positive dimension specified in the third argument, except when a vertical line or vector is specified, e.g., `\drawline(0,1){1in}`, which has zero width. If the specified direction lies in the 1st or 2d quadrant (e.g., `(1,1)` or `(-2,3)`), then the `\hbox` will have positive height and zero depth. Conversely, if the specified direction lies in the 3d or 4th quadrant (e.g., `(-1,-1)` or `(2,-3)`), then the `\hbox` will have positive depth and zero height.

There are a finite number of directions that can be specified. For `\drawline`, the absolute value of each integer defining the direction must be less than or equal to six, i.e., `(7,-1)` is incorrect, but `(6,-1)` is acceptable. For `\drawvector`, the absolute value of each integer must be less than or equal to four. Furthermore, the two integers cannot have common divisors; therefore, if a line with slope 2 is desired, say `(2,1)` instead of `(4,2)`. Also, specify `(1,0)` instead of, say, `(3,0)` for horizontal lines and likewise for vertical lines.

Finally, these macros depend upon the LaTeX font `line10`. If your site doesn't have this font, ask your system administrator to get it. Future enhancements will include macros to draw dotted lines and dotted vectors of various directions.

## 6.2 Commutative diagrams

The primitive commands `\drawline` and `\drawvector` can be used to typeset arrow theoretic diagrams. This section describes (1) macros to facilitate typesetting arrows and morphisms, and (2) macros to facilitate the construction of commutative diagrams. All macros described in this section must be used in math mode.

### 6.2.1 Arrows and morphisms

The macros `\mapright` and `\mapleft` produce right and left pointing arrows, respectively. Use superscript (`^`) to place a morphism above the arrow, e.g., ‘`\mapright^\alpha`’; use subscript (`_`) to place a morphism below the arrow, e.g., ‘`\mapright_{\tilde l}`’. Superscripts and subscripts may be used simultaneously, e.g., ‘`\mapright^\pi_{\rm epimor.}`’.

Similarly, the macros `\mapup` and `\mapdown` produce up and down pointing arrows, respectively. Use `\rt` to place a morphism to the right of the arrow, e.g., ‘`\mapup\rt{\rm` `id}`’; use `\lft` to place a morphism to the left of the arrow, e.g., ‘`\mapup\lft\omega`’. `\lft` and `\rt` may be used simultaneously, e.g., ‘`\mapdown\lft\pi\rt{\rm monomor.}`’.

Slanted arrows are produced by the macro `\arrow`, which takes a direction argument (e.g., ‘`\arrow(3,-4)`’). Use `\rt` and `\lft` to place morphisms to the right and left, respectively, of the arrow. A slanted line (no arrowhead) is produced with the macro `\sline`, whose syntax is identical to that of `\arrow`.

The length of these macros is predefined by the default TeX dimensions `\harrowlength`, for horizontal arrows (or lines), `\varrowlength`, for vertical arrows (or lines), and `\sarrowlength`, for slanted arrows (or lines). To change any of these dimensions, say, e.g., ‘`\harrowlength=40pt`’. As with all other TeX dimensions, the change may be as global or as local as you like. Furthermore, the placement of morphisms on the arrows is controlled by the dimensions `\hmorphposn`, `\vmorphposn`, and `\morphdist`. The first two dimensions control the horizontal and vertical position of the morphism from its default position; the latter dimension controls the distance of the morphism from the arrow. If you have more than one morphism per arrow (i.e., a `^/_` or `\lft/\rt` construction), use the parameters `\hmorphposnup`, `\hmorphposndn`, `\vmorphposnup`, `\vmorphposndn`, `\hmorphposnrt`, `\hmorphposnlft`, `\vmorphposnrt`, and `\vmorphposnlft`. The default values of all these dimensions are provided in the section on parameters that follows below.

There is a family of macros to produce horizontal lines, arrows, and adjoint arrows. The following macros produce horizontal maps and have the same syntax as `\mapright`:

| | |
|---|---|
| `\mapright` | `$X\mapright Y$` $\equiv X \longrightarrow Y$. |
| `\mapleft` | `$X\mapleft Y$` $\equiv X \longleftarrow Y$. |
| `\hline` | `$X\hline Y$` $\equiv X \;\rule[0.5ex]{2em}{0.4pt}\; Y$. |
| `\bimapright` | `$X\bimapright Y$` $\equiv X \rightrightarrows Y$. |
| `\bimapleft` | `$X\bimapleft Y$` $\equiv X \leftleftarrows Y$. |
| `\adjmapright` | `$X\adjmapright Y$` $\equiv X \leftrightarrows Y$. |
| `\adjmapleft` | `$X\adjmapleft Y$` $\equiv X \rightleftarrows Y$. |
| `\bihline` | `$X\bihline Y$` $\equiv X = Y$. |

There is also a family of macros to produce vertical lines, arrows, and adjoint arrows. The following macros produce vertical maps and have the same syntax as `\mapdown`:

| | |
|---|---|
| `\mapdown` | (a down arrow) |
| `\mapup` | (an up arrow) |
| `\vline` | (vertical line) |
| `\bimapdown` | (two down arrows) |

| | |
|---|---|
| `\bimapup` | (two up arrows) |
| `\adjmapdown` | (two adjoint arrows; down then up) |
| `\adjmapup` | (two adjoint arrows; up then down) |
| `\bivline` | (two vertical lines) |

Finally, there is a family of macros to produce slanted lines, arrows, and adjoint arrows. The following macros produce slanted maps and have the same syntax as `\arrow`:

| | |
|---|---|
| `\arrow` | (a slanted arrow) |
| `\sline` | (a slanted line) |
| `\biarrow` | (two straight arrows) |
| `\adjarrow` | (two adjoint arrows) |
| `\bisline` | (two straight lines) |

The width between double arrows is controlled by the parameter `\channelwidth`. The parameters `\hchannel` and `\vchannel`, if nonzero, override `\channelwidth` by controlling the horizontal and vertical shifting from the first arrow to the second.

There are no adornments on these arrows to distinguish inclusions from epimorphisms from monomorphisms. Many texts, such as Lang's book *Algebra*, use as a tasteful alternative the symbol 'inc' (in roman) next to an arrow to denote inclusion.

Future enhancements will include a mechanism to draw curved arrows found in, e.g., the Snake Lemma, by employing a version of the `\path` macros of Appendix D of *The TEXbook*.

### 6.2.2 Construction of commutative diagrams

There are two approaches to the construction of commutative diagrams described here. The first approach, and the simplest, treats commutative diagrams like fancy matrices, as Knuth does in Exercise 18.46 of *The TEXbook*. This case is covered by the macro `\commdiag`, which is an altered version of the Plain TEX macro `\matrix`. An example suffices to demonstrate this macro. The following commutative diagram (illustrating the covering homotopy property; Bott and Tu, *Differential Forms in Algebraic Topology*)

$$\begin{tikzcd}
Y \arrow[r, "f"] \arrow[d] & E \arrow[d] \\
Y \times I \arrow[ur, "f_t"] \arrow[r, "\bar f_t"] & X
\end{tikzcd}$$

is produced with the code

```
$$\commdiag{Y&\mapright^f&E\cr \mapdown&\arrow(3,2)\lft{f_t}&\mapdown\cr
Y\times I&\mapright^{\bar f_t}&X}$$
```

Of course, the parameters may be changed to produce a different effect. The following commutative diagram (illustrating the universal mapping property; Warner, *Foundations of Differentiable Manifolds and Lie Groups*)

$$\begin{tikzcd}
V\otimes W \arrow[dr, "\tilde{l}"] & \\
V\times W \arrow[u, "\phi"] \arrow[r, "l"] & U
\end{tikzcd}$$

is produced with the code

```
$$\varrowlength=20pt
\commdiag{V\otimes W\cr \mapup\lft\phi&\arrow(3,-1)\rt{\tilde l}\cr
V\times W&\mapright^l&U\cr}$$
```

A diagram containing isosceles triangles is achieved by placing the apex of the triangle in the center column, as shown in the example (illustrating all constant minimal realizations of a linear system; Brockett, *Finite Dimensional Linear Systems*)

$$\begin{tikzcd}
 & R^m \arrow[dl, "\mathbf{B}"'] \arrow[dr, "\mathbf{G}"] & \\
R^n \arrow[rr, "\mathbf{P}"] \arrow[dd, "e^{\mathbf{A}t}"'] & & R^n \arrow[dd, "e^{\mathbf{F}t}"] \\
 & & \\
R^n \arrow[rr, "\mathbf{P}"] \arrow[dr, "\mathbf{C}"'] & & R^n \arrow[dl, "\mathbf{H}"] \\
 & R^q &
\end{tikzcd}$$

which is produced with the code

```
$$\sarrowlength=.42\harrowlength
\commdiag{&R^m\cr &\arrow(-1,-1)\lft{\bf B}\quad \arrow(1,-1)\rt{\bf G}\cr
R^n&\mapright^{\bf P}&R^n\cr
\mapdown\lft{e^{{\bf A}t}}&&\mapdown\rt{e^{{\bf F}t}}\cr
R^n&\mapright^{\bf P}&R^n\cr
&\arrow(1,-1)\lft{\bf C}\quad \arrow(-1,-1)\rt{\bf H}\cr
&R^q\cr}$$
```

Other commutative diagram examples appear in the file `commdiags.tex`, which is distributed with this package.

In these examples the arrow lengths and line slopes were carefully chosen to blend with each other. In the first example, the default settings for the arrow lengths are used, but a direction for the arrow must be chosen. The ratio of the default horizontal and vertical arrow lengths is approximately the golden mean $\gamma = 1.618\ldots$; the arrow direction closest to this mean is `(3,2)`. In the second example, a slope of $-1/3$ is desired and the default horizontal arrow length is 60 pt; therefore, choose a vertical arrow length of 20 pt. You may affect the interline glue settings of `\commdiag` by redefining the macro `\commdiagbaselines`. (cf. Exercise 18.46 of *The TeXbook* and the section on parameters below.)

The width, height, and depth of all morphisms are hidden so that the morphisms' size do not affect arrow positions. This can cause a large morphism at the top or bottom of a diagram to impinge upon the text surrounding the diagram. To overcome this problem, use TeX's `\noalign` primitive to insert a `\vskip` immediately above or below the offending line, e.g., '`$$\commdiag{\noalign{\vskip6pt}X&\mapright^\int&Y\cr ...}`'.

The macro `\commdiag` is too simple to be used for more complicated diagrams, which may have intersecting or overlapping arrows. A second approach, borrowed from Francis Borceux's *Diagram* macros for LaTeX, treats the commutative diagram like a grid of identically shaped boxes. To compose the commutative diagram, first draw an equally spaced grid, e.g.,

. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .

on a piece of scratch paper. Then draw each element (vertices and arrows) of the commutative diagram on this grid, centered at each grid point. Finally, use the macro `\gridcommdiag` to implement your design as a TeX alignment. For example, the cubic diagram

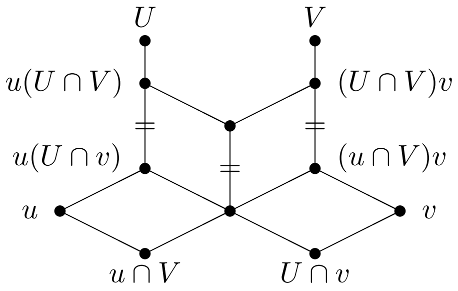
$$\begin{tikzcd}
 & B \arrow[rr, "b"] \arrow[dd, "f"'] & & D \arrow[dd, "h"] \\
A \arrow[ur, "a"] \arrow[rr, "c"] \arrow[dd, "e"'] & & C \arrow[ur, "d"] \arrow[dd, "g"] & \\
 & F \arrow[rr, "j"'] & & H \\
E \arrow[ur, "i"] \arrow[rr, "k"'] & & G \arrow[ur, "l"'] &
\end{tikzcd}$$

that appears in Francis Borceux's documentation can be implemented on a 7 by 7 grid, and is achieved with the code

```
$$\harrowlength=48pt \varrowlength=48pt \sarrowlength=20pt
\def\cross#1#2{\setbox0=\hbox{$#1$}%
  \hbox to\wd0{\hss\hbox{$#2$}\hss}\llap{\unhbox0}}
\gridcommdiag{&&B&&\mapright^b&&D\cr
&\arrow(1,1)\lft a&&&&\arrow(1,1)\lft d\cr
A&&\cross{\hmorphposn=12pt\mapright^c}{\vmorphposn=-12pt\mapdown\lft f}
&&C&&\mapdown\rt h\cr\cr
\mapdown\lft e&&F&&\cross{\hmorphposn=-12pt\mapright_j}
{\vmorphposn=12pt\mapdown\rt g}&&H\cr
&\arrow(1,1)\lft i&&&&\arrow(1,1)\rt l\cr
E&&\mapright_k&&G\cr}$$
```

The dimensions `\hgrid` and `\vgrid` control the horizontal and vertical spacing of the grid used by `\gridcommdiag`. The default setting for both of these dimensions is 15 pt. Note that in the example of the cube the arrow lengths must be adjusted so that the arrows overlap into neighboring boxes by the desired amount. Hence, the `\gridcommdiag` method, albeit more powerful, is less automatic than the simpler `\commdiag` method. Furthermore, the ad hoc macro `\cross` is introduced to allow the effect of overlapping arrows. Finally, note that the positions of four of the morphisms are adjusted by setting `\hmorphposn` and `\vmorphposn`.

One is not restricted to a square grid. For example, the proof of Zassenhaus's Butterfly Lemma can be illustrated by the diagram (appearing in Lang's book *Algebra*)



This diagram may be implemented on a 9 by 12 grid with an aspect ratio of 1/2, and is set with the code

```
$$\hgrid=16pt \vgrid=8pt \sarrowlength=32pt
\def\cross#1#2{\setbox0=\hbox{$#1$}%
  \hbox to\wd0{\hss\hbox{$#2$}\hss}\llap{\unhbox0}}
\def\l#1{\llap{$#1$\hskip.5em}}
\def\r#1{\rlap{\hskip.5em$#1$}}
\gridcommdiag{&&U&&&&V\cr &&\bullet&&&&\bullet\cr
&&\sarrowlength=16pt\sline(0,1)&&&&\sarrowlength=16pt\sline(0,1)\cr
&&\l{u(U\cap V)}\bullet&&&&\bullet\r{(U\cap V)v}\cr
&&&\sline(2,-1)&&\sline(2,1)\cr
&&\cross{=}{\sline(0,1)}&&\bullet&&\cross{=}{\sline(0,1)}\cr\cr
&&\l{^{\textstyle u(U\cap v)}}\bullet&&\cross{=}{\sline(0,1)}&&
 \bullet\r{^{\textstyle(u\cap V)v}}\cr
&\sline(2,1)&&\sline(2,-1)&&\sline(2,1)&&\sline(2,-1)\cr
\l{u}\bullet&&&&\bullet&&&&\bullet\r{v}\cr
&\sline(2,-1)&&\sline(2,1)&&\sline(2,-1)&&\sline(2,1)\cr
&&\bullet&&&&\bullet\cr &&u\cap V&&&&U\cap v\cr}$$
```

Again, the construction of this diagram requires careful choices for the arrow lengths and is facilitated by the introduction of the ad hoc macros `\cross`, `\r`, and `\l`. Note also that superscripts were used to adjust the position of the vertices $u(U\cap v)$ and $(u\cap V)v$. Many diagrams may be typeset with the predefined macros that appear here; however, ingenuity is often required to handle special cases.

### 6.2.3 Commutative diagram parameters

The following is a list describing the parameters used in the commutative diagram macros. These dimensions may be changed globally or locally.

`\harrowlength`
: (Default: 60 pt) The length of right or left arrows.

`\varrowlength`
: (Default: 0.618`\harrowlength`) The length of up or down arrows.

`\sarrowlength`
: (Default: 60 pt) The horizontal length of slanted arrows.

`\hmorphposn`
: (Default: 0 pt) The horizontal position of the morphism with respect to its default position. There are also the dimensions `\hmorphposnup`, `\hmorphposndn`, `\hmorphposnrt`, and `\hmorphposnlft` for `^`/`_` or `\lft`/`\rt` constructions.

`\vmorphposn`
: (Default: 0 pt) The vertical position of the morphism with respect to its default position. There are also the dimensions `\vmorphposnup`, `\vmorphposndn`, `\vmorphposnrt`, and `\vmorphposnlft` for `^`/`_` or `\lft`/`\rt` constructions.

`\morphdist`
: (Default: 4 pt) The distance of morphisms from slanted lines or arrows.

`\channelwidth`
: (Default: 3 pt) The distance between double lines or arrows.

`\hchannel, \vchannel`
: (Defaults: 0 pt) Overrides `\channelwidth`. The horizontal and vertical shifts between double lines or arrows.

`\commdiagbaselines`
: (Default: `\baselineskip=15pt \lineskip=3pt \lineskiplimit=3pt` ) The parameters used by `\commdiag` for setting interline glue.

`\hgrid` (Default: 15 pt) The horizontal spacing of the grid used by `\gridcommdiag`.

`\vgrid` (Default: 15 pt) The vertical spacing of the grid used by `\gridcommdiag`.

# 7 Programming definitions

The definitions in this section are only likely to be useful when you are writing nontrivial macros, not when writing a document.

## 7.1 Category codes

Plain TEX defines `\active` (as the number 13) for use in changing category codes. Although the author of *The TEXbook* has "intentionally kept the category codes numeric", two other categories are commonly used: letters (category code 11) and others (12). Therefore, Eplain defines `\letter` and `\other`.

Sometimes it is cleaner to make a character active without actually writing a `\catcode` command. The `\makeactive` command takes a character as an argument to make active (and ignores following spaces). For example, here are two commands which both make `\` active:

```
\makeactive\\   \makeactive92
```

Sometimes you might want to temporarily change the category code of the '@' character to `\letter`, so that you can use or define macros which are normally inaccessible to the user. For such situations, Eplain provides the `\makeatletter` command. It sets the category code of '@' to `\letter` (11) and defines `\resetatcatcode` to restore the category code to whatever it was before the call to `\makeatletter`. For example:

```
\makeatletter
\def\@hidden@macro{This macro cannot normally be
                   called / redefined by the user}
\resetatcatcode
```

There is also `\makeatother` which works similarly but sets the category code of '@' to `\other` (12).

Usually, when you give a definition to an active character, you have to do so inside a group where you temporarily make the character active, and then give it a global definition (cf. the definition of `\obeyspaces` in *The TEXbook*). This is inconvenient if you are writing a long macro, or if the character already has a global definition you do not wish to transcend. Eplain provides `\letreturn`, which defines the usual end-of-line character to be the argument. For example:

```
\def\mymacro{... \letreturn\myreturn ... }
\mymacro hello
there
```

The end-of-line between '`hello`' and '`there`' causes `\myreturn` to be expanded.

*The TEXbook* describes `\uncatcodespecials`, which makes all characters which are normally "special" into "other" characters, but the definition never made it into plain TEX. Eplain therefore defines it. For notes on the usage, see Section 4.7 [Verbatim listing], page 12.

Finally, `\percentchar` expands into a literal '%' character. This is useful when you `\write` TEX output to a file, and want to avoid spurious spaces. For example, Eplain writes a `\percentchar` after the definition of cross-references. The macros `\lbracechar` and `\rbracechar` expand similarly.

## 7.2 Allocation macros

Plain TEX provides macros that allocate registers of each primitive type in TEX, to prevent different sets of macros from using the same register for two different things. The macros are all named starting with '`new`', e.g., `\newcount` allocates a new "count" (integer) register. Such allocations are usually needed only at the top level of some macro definition file; therefore, plain TEX makes the allocation registers `\outer`, to help find errors. (The error this helps to find is a missing right brace in some macro definition.)

Sometimes, however, it is useful to allocate a register as part of some macro. An outer control sequence cannot be used as part of a macro definition (or in a few other contexts: the parameter text of a definition, an argument to a definition, the preamble of an alignment, or in conditional text that is being skipped). Therefore, Eplain defines "inner" versions of all the allocation macros, named with the prefix '`inner`': `\innernewbox`, `\innernewcount`, `\innernewdimen`, `\innernewfam`, `\innernewhelp`, `\innernewif`, `\innernewinsert`, `\innernewlanguage`, `\innernewread`, `\innernewskip`, `\innernewtoks`, `\innernewwrite`.

You can also define non-outer versions of other macros in the same way that Eplain defines the above. The basic macro is called `\innerdef`:

```
\innerdef \innername {outername}
```

The first argument (\*innername*) to `\innerdef` is the control sequence that you want to define. Any previous definition of \*innername* is replaced. The second argument (*outername*) is the *characters* in the name of the outer control sequence. (You can't use the actual control sequence name, since it's outer!)

If the outer control sequence is named \*cs*, and you want to define `innercs` as the inner one, you can use `\innerinnerdef`, which is just an abbreviation for a call to `\innerdef`. For example, these two calls are equivalent:

```
\innerdef\innerproclaim{proclaim}
\innerinnerdef{proclaim}
```

## 7.3 Iteration

You can iterate through a comma-separated list of items with `\for`. Here is an example:

```
\for\name:=karl,kathy\do{%
   \message{\name}%
}%
```

This writes '`karl`' and '`kathy`' to the terminal. Spaces before or after the commas in the list, or after the `:=`, are *not* ignored. To strip leading spaces off the items, use `\For`:

```
\For\name:=
   karl,
   kathy\do{%
      \message{\name}%
   }%
```

Note that trailing spaces are still *not* ignored.

Both `\for` and `\For` expand the first token of the item list fully, so this is equivalent to the above:

```
\def\namelist{karl,kathy}%
```

```
\for\name:=\namelist\do ...
```

However, this won't work, either with `\for` or with `\For`:

```
\def\namelist{karl,kathy}%
\For\name:= \namelist\do ...
```

because `\for` and `\For` expand the first token after `:=` which is space, not `\namelist`.

Eplain provides another kind of loops, which is an extension of plain TEX's `\loop`. If you say:

```
\loop
  loop-text
\if condition
  if-text
\repeat
```

then *loop-text* will be repeated as long as *condition* is satisfied (`\if` can be any of the TEX's conditional commands, without the matching `\fi`). Eplain extends this with the optional else clause:

```
\loop
  loop-text
\if condition
  if-text
\else
  else-text
\repeat
```

Here, *loop-text* will be repeated as long as *condition* is *not* satisfied. This extension is from Victor Eijkhout's *TEX by Topic* (page 104).

## 7.4 Macro arguments

It is occasionally useful to redefine a macro that takes arguments to do nothing. Eplain defines `\gobble`, `\gobbletwo`, and `\gobblethree` to swallow one, two, and three arguments, respectively.

For example, if you want to produce a "short" table of contents—one that includes only chapters, say—the easiest thing to do is read the entire `.toc` file (see Section 4.8 [Contents], page 13), and just ignore the commands that produce section or subsection entries. To be specific:

```
\let\tocchapterentry = \shorttocchapter
\let\tocsectionentry = \gobbletwo
\let\tocsubsectionentry = \gobbletwo
\readtocfile
```

(Of course, this assumes you only have chapters, sections, and subsections in your document.)

In addition, Eplain defines `\eattoken` to swallow the single following token, using `\let`. Thus, `\gobble` followed by '`{...}`' ignores the entire brace-enclosed text. `\eattoken` followed by the same ignores only the opening left brace.

Eplain defines a macro `\identity` which takes one argument and expands to that argument. This may be useful if you want to provide a function for the user to redefine, but don't need to do anything by default. (For example, the default definition of `\eqconstruct` (see Section 4.11.1 [Formatting equation references], page 18) is `\identity`.)

You may also want to read an optional argument. The established convention is that optional arguments are put in square brackets, so that is the syntax Eplain recognizes. Eplain ignores space tokens before and after optional arguments, via `\futurenonspacelet`.

You test for an optional argument by using `\@getoptionalarg`. It takes one argument, a control sequence to expand after reading the argument, if present. If an optional argument is present, the control sequence `\@optionalarg` expands to it; otherwise, `\@optionalarg` is `\empty`. You must therefore have the category code of `@` set to 11 (letter). Here is an example:

```
\catcode`@=\letter
\def\cmd{\@getoptionalarg\finishcmd}
\def\finishcmd{%
  \ifx\@optionalarg\empty
    % No optional argument present.
  \else
    % One was present.
  \fi
}
```

It's possible to define macros that appear to accept optional arguments intermixed with mandatory arguments in any imaginable way. For example:

```
\makeatletter
% \mo{m}[o]
\def\mo#1{\def\mo@arg{#1}\@getoptionalarg\fin@mo}
\def\fin@mo{\vskip1pc
  ARG:    \mo@arg       \par
  OPTARG: \@optionalarg \par
}
% \mom{m}[o]{m}
\def\mom#1{\def\mom@arg{#1}\@getoptionalarg\fin@mom}
\def\fin@mom#1{\vskip1pc
  ARG1:   \mom@arg      \par
  OPTARG: \@optionalarg \par
  ARG2:   #1\par
}
% \omo[o]{m}[o]
\def\omo{\@getoptionalarg\fin@omo}
\def\fin@omo#1{\let\omo@optarg\@optionalarg
               \def\omo@arg{#1}\@getoptionalarg\@fin@omo}
\def\@fin@omo{\vskip1pc
  OPTARG1: \omo@optarg   \par
  ARG:     \omo@arg      \par
  OPTARG2: \@optionalarg \par
}
\makeatother
```

If an optional argument contains another optional argument, the inner one will need to be enclosed in braces, so TEX does not mistake the end of the first for the end of the second.

## 7.5 Converting to characters

Eplain defines `\xrlabel` to produce control sequence names for cross-reference labels, et al. This macro expands to its argument with an '`_`' appended. (It does this because the usual use of `\xrlabel` is to generate a control sequence name, and we naturally want to avoid conflicts between control sequence names.)

Because `\xrlabel` is fully expandable, to make a control sequence name out of the result you need only do

```
\csname \xrlabel{label}\endcsname
```

The `\csname` primitive makes a control sequence name out of any sequence of character tokens, regardless of category code. Labels can therefore include any characters except for '`\`', '`{`', '`}`', and '`#`', all of which are used in macro definitions themselves.

`\sanitize` takes a control sequence as an argument and converts the expansion of the control sequence into a list of character tokens. This is the behavior you want when writing information like chapter titles to an output file. For example, here is part of the definition of `\writenumberedtocentry`; `#2` is the title that the user has given.

```
...
\def\temp{#2}%
...
  \write\tocfile{%
    ...
    \sanitize\temp
    ...
  }%
```

## 7.6 Expansion

This section describes some miscellanous macros for expansion, etc.

### 7.6.1 `\csn` and `\ece`

`\csn`{*name*} simply abbreviates `\csname` *name* `\endcsname`, thus saving some typing. The extra level of expansion does take some time, though, so I don't recommend it for an inner loop.

`\ece`{*token*}{*name*} abbreviates

```
\expandafter token \csname name \endcsname
```

For example,

```
\def\fontabbrevdef#1#2{\ece\def{@#1font}{#2}}
\fontabbrevdef{normal}{ptmr}
```

defines a control sequence `\@normalfont` to expand to `ptmr`.

### 7.6.2 `\edefappend`

`\edefappend` is a way of adding on to an existing definition. It takes two arguments: the first is the control sequence name, the second the new tokens to append to the definition. The second argument is fully expanded (in the `\edef` that redefines the control sequence).

For example:

```
\def\foo{abc}
\def\bar{xyz}
\edefappend\foo{\bar karl}
```

results in `\foo` being defined as '`abcxyzkarl`'.

### 7.6.3 Hooks

A *hook* is simply a name for a group of actions which is executed in certain places—presumably when it is most useful to allow customization or modification. TEX already provides many builtin hooks; for example, the `\every ...` token lists are all examples of hooks.

Eplain provides several macros for adding actions to hooks. They all take two arguments: the name of the hook and the new actions.

`hookaction` *`name actions`*
`hookappend` *`name actions`*
`hookprepend` *`name actions`*

Each of these adds *actions* to the hook *name*. (Any previously-defined actions are retained.) *name* is not a control sequence, but rather the characters of the name.

`hookactiononce` *`name`* `\`*`cs`*

`\hookactiononce` adds *cs* to *name*, like the macros above, but first it adds

```
\global\let \cs \relax
```

to the definition of \*cs*. (This implies \*cs* must be a true expandable macro, not a control sequence `\let` to a primitive or some other such thing.) Thus, \*cs* is expanded the next time the hook *name* is run, but it will disappear after that.

The `\global` is useful because `\hookactiononce` is most useful when the grouping structure of the TEX code could be anything. Neither this nor the other hook macros do global assignments to the hook variable itself, so TEX's usual grouping rules apply.

The companion macro to defining hook actions is `\hookrun`, for running them. This takes a single argument, the name of the hook. If no actions for the hook are defined, no error ensues.

Here is a skeleton of general `\begin` and `\end` macros that run hooks, and a couple of calls to define actions. The use of `\hookprepend` for the begin action and `\hookappend` for the end action ensures that the actions are executed in proper sequence with other actions (as long as the other actions use `\hookprepend` and `\hookappend` also).

```
\def\begin#1{ ... \hookrun{begin} ... }
\def\end#1{ ... \hookrun{end} ... }
```

```
\hookprepend{begin}\start_underline
\hookappend{end}\finish_underline
```

### 7.6.4 Properties

A *property* is a name/value pair associated with another symbol, traditionally called an *atom*. Both atom and property names are control sequence names.

Eplain provides two macros for dealing with property lists: `\setproperty` and `\getproperty`.

`\setproperty` *atom propname value*

`\setproperty` defines the property *property* on the atom *atom* to be *value*. *atom* and *propname* can be anything acceptable to `\csname`. *value* can be anything.

`\getproperty` *atom propname*

`\getproperty` expands to the value stored for *propname* on *atom*. If *propname* is undefined, it expands to nothing (i.e., `\empty`).

The idea of properties originated in Lisp (I believe). There, the implementation truly does associate properties with atoms. In TEX, where we have no builtin support for properties, the association is only conceptual.

The following example typesets '`xyz`'.

```
\setproperty{a}{pr}{xyz}
\getproperty{a}{pr}
```

### 7.6.5 `\expandonce`

`\expandonce` is defined as `\expandafter\noexpand`. Thus, `\expandonce` *token* expands *token* once, instead of to TEX primitives. This is most useful in an `\edef`.

For example, the following defines `\temp` to be `\foo`, not '`abc`'.

```
\def\foo{abc}
\def\bar{\foo}
\edef\temp{\expandonce\bar}
```

### 7.6.6 `\ifundefined`

`\ifundefined{`*cs*`}` *t* `\else` *f* `\fi` expands the *t* text if the control sequence `\`*cs* is undefined or has been `\let` to `\relax`, and the *f* text otherwise.

Since `\ifundefined` is not a primitive conditional, it cannot be used in places where TEX might skip tokens "at high speed", e.g., within another conditional—TEX can't match up the `\if`'s and `\fi`'s.

This macro was taken directly from *The TEXbook*, page 308.

### 7.6.7 `\ifempty`

`\ifempty{`*arg*`}` *t* `\else` *f* `\fi` expands the *t* text if *arg* is the empty string, and the *f* text otherwise. This macro is useful when you need to test for empty arguments to your macros, for example:

```
\def\foo#1{\ifempty{#1} t \else f \fi}
```

Since `\ifempty` is not a primitive conditional, it cannot be used in places where TEX might skip tokens "at high speed", e.g., within another conditional—TEX can't match up the `\if`'s and `\fi`'s.

Note that the following code

```
\def\empty{}
\ifempty\empty\message{empty}\else\message{not empty}\fi
```

will produce the message '`not empty`'.

### 7.6.8 `\ifinteger` and `\isinteger`

`\ifinteger{`*arg*`}` *t* `\else` *f* `\fi` expands the *t* text if *arg* is an integer, and the *f* text otherwise. This macro can detect positive and negative integers.

Since `\ifinteger` is not a primitive conditional, it cannot be used in places where TEX might skip tokens "at high speed", e.g., within another conditional—TEX can't match up the `\if`'s and `\fi`'s. For such situations Eplain provides `\isinteger`, which can be used as follows:

```
\if\isinteger{arg} t \else f \fi
```

Although `\ifinteger` and `\isinteger` work well with regular input, they are not bulletproof. For example, the following code

```
\ifinteger{12_ab}integer\else not integer\fi
```

will expand to '`ab_integer`' (and thus would not even compile outside math mode).

These macros come from the TEX Frequently Asked Questions (`http://www.tex.ac.uk/cgi-bin/texfaq2html`).

### 7.6.9 `\futurenonspacelet`

The `\futurelet` primitive allows you to look at the next token from the input. Sometimes, though, you want to look ahead while ignoring any spaces. This is what `\futurenonspacelet` does. It is otherwise the same as `\futurelet`: you give it two control sequences as arguments, and it assigns the next nonspace token to the first, and then expands the second. For example:

```
\futurenonspacelet\temp\finishup
\def\finishup{\ifx\temp ...}
```

## 7.7 Obeying spaces

`\obeywhitespace` makes both end-of-lines and space characters in the input be respected in the output. Unlike plain TEX's `\obeyspaces`, even spaces at the beginnings of lines turn into blank space.

By default, the size of the space that is produced by a space character is the natural space of the current font, i.e., what `\ ` produces.

Ordinarily, a blank line in the input produces as much blank vertical space as a line of text would occupy. You can adjust this by assigning to the parameter `\blanklineskipamount`: if you set this negative, the space produced by a blank line will be smaller; if positive, larger.

Tabs are not affected by this routine. In particular, if tabs occur at the beginning of a line, they will disappear. (If you are trying to make TEX do the "right thing" with tabs, don't. Use a utility program like *expand* instead.)

## 7.8 Writing out numbers

`\numbername` produces the written-out form of its argument, i.e., ‘zero’ through ‘ten’ for the numbers 0–10, and numerals for all others.

## 7.9 Mode-specific penalties

TeX’s built-in `\penalty` command simply appends to the current list, no matter what kind of list it is. You might intend a particular penalty to always be a “vertical” penalty, however, i.e., appended to a vertical list. Therefore, Eplain provides `\vpenalty` and `\hpenalty` which first leave the other mode and then do `\penalty`.

More precisely, `\vpenalty` inserts `\par` if the current mode is horizontal, and `\hpenalty` inserts `\leavevmode` if the current mode is vertical. (Thus, `\vpenalty` cannot be used in math mode.)

## 7.10 Auxiliary files

It is common to write some information out to a file to be used on a subsequent run. But when it is time to read the file again, you only want to do so if the file actually exists. `\testfileexistence` is given an argument which is appended to `\jobname`, and sets the conditional `\iffileexists` appropriately. For example:

```
\testfileexistence{toc}%
\iffileexists
   \input \jobname.toc
\fi
```

`\testfileexistence` takes an optional parameter; when given, it will override `\jobname` for the root part of the file name. For example, if you want to test for the file `answers.aux`, you can do this with the following:

```
\testfileexistence[answers]{aux}%
\iffileexists
   \input answers.aux
\fi
```

## 7.11 User-defined environments

Plain TeX does not provide “named” block structures, only the anonymous `\begingroup` and `\endgroup` pair. The disadvantage of this is that when there are several such groups and one is mismatched, it can be difficult to find the error. Eplain provides a named block structure so that if you forget an `\environment` or an `\endenvironment`, you will (probably) get an error message about it.

For example:

```
\def\itpar{
  \environment{@italicpar}
  \it\par
}
\def\enditpar{
  \par
  \endenvironment{@italicpar}%
}
```

which could then be used to set italicized paragraphs:

```
\itpar
If I reprehend anything in this world, it is the use of my oracular
tongue, and a nice derangement of epitaphs!
\enditpar
```

The above sort of environment allows nesting. But environments shouldn't always be allowed to nest. Put the control sequence `\checkenv` at the beginning of a macro that is going to define an environment that should not be nested.

## 7.12 Page list and page range parsers

The macros which Eplain uses to parse the page lists and ranges in the index, `\idxparselist` and `\idxparserange` (see Section 5.3.8.2 [Page destinations for index terms], page 46), are sometimes useful when defining page number encapsulators. They take one argument, text to parse. When a page list (range) is not present, they set `\idxpagei` to be `\empty`; when a list (range) is detected, they set `\idxpagei` and `\idxpageii` to the first and the second page numbers, respectively.

Eplain's defaults for the page list and page range delimiters are the same as those in MakeIndex, a comma followed by a space ('`, `') and two dashes ('`--`'), respectively. If you customize MakeIndex to use different delimiters, you must not forget to let Eplain know about them with the commands

```
\setidxpagelistdelimiter{list-delim}
\setidxpagerangedelimiter{page-delim}
```

These commands save the *list-delim* and *page-delim* delimiters in `\idxpagelistdelimiter` and `\idxpagerangedelimiter`, respectively.

For example, you may want to define a page number markup command which italicizes and properly underlines page ranges by underlining only the page numbers and not the delimiter:

```
\def\ituline#1{%
  {\it
  \idxparserange{#1}%
  \ifx\idxpagei\empty
    % The argument is a single page number.
    \underbar{#1}%
  \else
    % The argument is a page range.
    \underbar{\idxpagei}\idxpagerangedelimiter\underbar{\idxpageii}%
  \fi}%
}
```

Note that the `\ituline` macro is not aware of page lists. This is not needed if you use hyperlinks in the index, because `\hlidx` and `\hlidxpage` will break up the page lists before calling the user's page encapsulator (see Section 5.3.8.2 [Page destinations for index terms], page 46), so `\ituline` will never see the lists. If, however, you need to design a macro which also takes care of the lists, you can extend `\ituline` with an additional call to `\idxparselist`.

# 8 Demo files

This chapter contains listings of source files, along with the output they produce (where appropriate), which illustrate various aspects of Eplain. The files can be found in the `demo` subdirectory of Eplain distribution. These demos, both the sources and the compiled PDF and PS files, are also available from `https://tug.org/eplain/demo`.

## 8.1 Hyperlinks (`xhyper.tex`)

Figure 1. Lion in the archives

Show me the lion in fig. 1.
Show me the CTAN lion.
Take me to `http://tug.org/eplain`.
Take me to Eplain homepage.

```
% (This file is public domain.)
%
% This file demonstrates the following features of Eplain:
%
%   - explicit and implicit hyperlinks;
%   - symbolic cross-references;
%   - inserting external graphics using |\includegraphics| from
%     the \LaTeX\ package |graphicx.sty|.
%   - rotating text using |\rotatebox| from |graphicx.sty|.
%
% The compiled demo can be downloaded from
%
%   http://tug.org/eplain/demo
%
% In order to compile this file yourself, you will need the CTAN lion
% drawing by Duane Bibby from
%
%   ftp://tug.ctan.org/ftpmaint/CTAN_lion/ctan_lion_350x350.png
%
```

```
% (thanks, |www.ctan.org|).  Place the image file in the same
% directory with this file, and change to that directory.  Now, to
% produce a PDF, run twice the command
%
%   pdftex xhyper.tex
%
% During the first run, Eplain will write the information about the
% cross-references into |xhyper.aux|, and during the second run this
% information will be read by Eplain to typeset the references.
%
% Demo case:
%
%   Suppose you are using pdf\TeX, have a figure you want to insert
%   scaled to $200\,pt$ horizontally, and you want this figure to
%   completely fill the PDF viewer's window whenever the reader
%   selects a link pointing to this figure.  Additionally, you want to
%   typeset some text as live hyperlinks, clicking on which will start
%   a Web browser and open a URL.

\input eplain

% Load \LaTeX\ packages.
\beginpackages
  % |url.sty| provides the |\url| command which we will use to typeset
  % a URL.  We request that |url.sty| be the version from June~27,
  % 2005, or later, because earlier versions had problems interacting
  % with plain \TeX.
  \usepackage{url}[2005/06/27]
  % |color.sty| provides support for colored text; all hyperlinks are
  % automatically colored by Eplain when this package is loaded.  We give
  % the |dvipsnames| option because we want to use named colors from the
  % |dvips| graphics driver.
  \usepackage[dvipsnames]{color}
  % Finally, we load |graphicx.sty|, for the macros |\includegraphics|
  % and |\rotatebox|.
  \usepackage{graphicx}
\endpackages

% Remember that hyperlinks are off by default.  Therefore, we need to
% enable them.
\enablehyperlinks

% Customize some hyperlink options.  First, we set border width to~$0$
% for all links to get rid of the default boxes around links (we
% prefer colored links over the boxed links). Next, we say that all
% links created by the |url| hyperlink group (which means the |\url|
% command from |url.sty|) must be colored using the named color
% |BlueViolet|.
\hlopts{bwidth=0}
\hlopts[url]{colormodel=named,color=BlueViolet}

% Inhibit page numbering (we have only one page).
\nopagenumbers

% Define a class word for the cross-reference class |figure|.  This
% word, when defined, will be automatically prepended by Eplain to the
% reference created by |\ref| (read on to see its use).
\def\figureword{fig.}

% Allocate a count register for the figure's number, and a box
% register which we'll use to measure dimensions of the image.
\newcount\fignumber
\newbox\imgbox

% Now comes the fun part--constructing the figure for the image of the
% \CTAN\ lion.  We define a macro
%
%   \fig{LABEL}{FILENAME}{CAPTION}{WIDTH}
%
% which creates a figure using LABEL for the cross-reference and
% hyperlink label, reading the image from file FILENAME, using CAPTION
% to name the figure, and WIDTH to scale the image horizontally.
\def\fig#1#2#3#4{%
  % Leave some space above the figure.  This will also ensure that we
  % are in the vertical mode before we produce a |\vbox|.
  \medskip
  % Advance the figure number.  |\global| ensures that the change to
  % the count register is global, even when |\fig| is buried inside a
  % group.
  \global\advance\fignumber by 1
  % We use |\includegraphics| (from |graphicx.sty|) to load the image,
  % scaled to the specified width, into our ``measuring'' box
  % |\imgbox|.
  \setbox\imgbox = \hbox{\includegraphics[width=#4]{#2}}%
  % To make the demo even more exciting, let's put the figure's
  % caption to the left of the image into the |\indent| space of the
  % new paragraph, and rotate the caption~$90^\circ$.
  \textindent{%
    \rotatebox{90}{F{\sc IGURE}~\the\fignumber.  #3}%
  }%
  % Continue the paragraph by constructing a |\vbox| with the image of
  % the lion.  We use |\definexref| to define the cross-reference
  % label.
  \vbox{%
```

```
    % In addition to the cross-reference label, |\definexref| will
    % create a hyperlink destination with the same label.  Therefore,
    % we customize this destination to do what we need.  We say that
    % destination type for the hyperlink group |definexref| (to which
    % |\definexref| belongs) should be |fitr|.  This destination type
    % will magnify the rectangle specified by the options |width|,
    % |height| and |depth| to the PDF viewer's window.  Therefore, we
    % set those options accordingly with |\hldestopts| (notice the use
    % of |depth| instead of |height|---we will want the rectangle to
    % extend {\it downward}, to cover the image which will come
    % next).  Notice that these settings will be isolated within the
    % current group (i.e., the |\vbox| we're constructing).
    \hldesttype[definexref]{fitr}%
    \hldestopts[definexref]{width=\wd\imgbox,height=0pt,depth=\ht\imgbox}%
    % We define a symbolic label so that we can later refer
    % to the figure with |\ref|.  The command |\definexref| does
    % exactly that.  The last argument to |\definexref| specifies
    % class of the label, which determines the word used by |\ref| in
    % front of the reference text (remember that we've defined
    % |\figureword| above).
    \definexref{#1}{\the\fignumber}{figure}%
    % Finally, produce the image which we've been stashing in the box
    % register |\imgbox|.
    \box\imgbox
  }%
  \medskip
}

% Create the figure.
\fig{CTANlion}{ctan_lion_350x350}{Lion in the archives}{200pt}

% Finished with the fun part, we can relax and typeset some
% hyperlinks.  The easiest way to do that is to use the |\ref|
% cross-reference command.  We can even pass an optional argument
% (|the lion in|), which will be placed before the class word (|fig.|)
% and become part of the link (to make sure the reader does not have
% to aim too hard).
Show me \ref[the lion in]{CTANlion}.

% If you are the restless kind, here's another way to create a
% hyperlink to the image:  we create a link explicitly by using
% |\hlstart ... \hlend|.  We don't specify the link type, therefore
% the default type |name| will be used (these are ``named links'',
% i.e., links pointing to destinations identified by labels).  In the
% options argument, we specify that the border of the link should be
% inverted when the user clicks the link (|hlight=O|), and also set
% special color for this link, overriding the default dark-red.  The
% label for the destination is the same as the cross-reference label,
% |CTANlion|.
Show me
\hlstart{}{hlight=O,colormodel=named,color=OliveGreen}{CTANlion}
  the CTAN lion\hlend.

% Let's now point somewhere outside our document.  Eplain homepage is
% a good target.  In the similar spirit, let's consider two
% approaches.  The easy one is to use the |\url| command from
% |url.sty|.  Remember that we have customized the color of |url|
% hyperlinks, so this one will show up with a different color than the
% default dark-red.
Take me to \url{http://tug.org/eplain}.

% The second approach is to create an explicit URL link.  We specify
% yet another border highlighting mode, |P|, which means that the
% region underneath the bounding box of the link will be drawn inset
% into the page.  Also, let's set the color of the hyperlink to an RGB
% color |0.4,0.1,0.4|.  Since we cannot use commas to separate the
% color elements inside the options parameter to |\hlstart| (commas
% there separate options), we have to first create a user-defined
% color with |\definecolor| from |color.sty|, and use that in
% |\hlstart|.
\definecolor{mycolor}{rgb}{0.4,0.1,0.4}

Take me to
\hlstart{url}{hlight=P,colormodel=,color=mycolor}{http://tug.org/eplain}
  Eplain homepage\hlend.

\bye
```

## 8.2 Highlighting TEX comments in listings (`lscommnt.tex`)

```
% (This file is public domain.)
% Demonstrate how Eplain can be used to include a TeX source file
% verbatim, typesetting comments in colored italic typewriter type.

% Load Eplain and LaTeX's color.sty package.
\input eplain
\beginpackages \usepackage{color} \endpackages
\nopagenumbers % Disable page numbers.
\font\commentfont = cmitt10 % Font for the comments (italic \tt).
% We'll define some `protected' macros with `@' in their names.
\makeatletter
% Define an equivalent of Eplain's \letreturn, to be able to assign
% various actions to the (active) percent character.
\begingroup \makeactive\%
  \gdef\letpercent{\let%}
\endgroup
% The listing hook to be called in \setuplistinghook, see below.  It
% makes `%' active and assigns it a `start comment' action.
\def\hlightcommentslisting{\makeactive\% \letpercent\start@comment}%
% This is what `%' is aliased to before a comment is started.
\def\start@comment{%
  \leavevmode % Needed in the very first line of the input to process
              % the new par (possibly inserting line number) before we
              % kick in with the color and stuff.
  \begingroup % Isolate color and font changes and the redefinitions.
    \commentfont
    \color[cmyk]{0.28,1,1,0.35}%
    \percentchar            % Produce the actual `%' and
    \letpercent\percentchar % make all following `%'s do the same.
    \letreturn\end@comment  % Call \end@comment at end-of-line.
}
% \end@comment (alias for ^^M inside a comment) will end the comment
% started by \start@comment.  We make ^^M active temporarily so that
% the active version of ^^M gets "frozen" into \end@comment.
\begingroup \makeactive\^^M % Avoid ^^M's from here on.
  \gdef\end@comment{\endgroup ^^M}%
\endgroup
\resetatcatcode % Make `@' again inaccessible for use in macro names.

% Define \setuplistinghook to setup comments highlighting, line
% numbering and omitting the last (empty) line of input.
\def\setuplistinghook{\hlightcommentslisting \linenumberedlisting
                      \nolastlinelisting}
% Isn't this fun?  This file typesets itself, with the extra bonus of
% the pretty-printed comments and numbered source lines!
\listing{lscommnt}
\bye
```

```
% (This file is public domain.)
% Demonstrate how Eplain can be used to include a TeX source file
% verbatim, typesetting comments in colored italic typewriter type.

% Load Eplain and LaTeX’s color.sty package.
\input eplain
\beginpackages \usepackage{color} \endpackages
\nopagenumbers % Disable page numbers.
\font\commentfont = cmitt10 % Font for the comments (italic \tt).
% We’ll define some ‘protected’ macros with ‘@’ in their names.
\makeatletter
% Define an equivalent of Eplain’s \letreturn, to be able to assign
% various actions to the (active) percent character.
\begingroup \makeactive\%
  \gdef\letpercent{\let%}
\endgroup
% The listing hook to be called in \setuplistinghook, see below.  It
% makes ‘%’ active and assigns it a ‘start comment’ action.
\def\hlightcommentslisting{\makeactive\% \letpercent\start@comment}%
% This is what ‘%’ is aliased to before a comment is started.
\def\start@comment{%
  \leavevmode % Needed in the very first line of the input to process
              % the new par (possibly inserting line number) before we
              % kick in with the color and stuff.
  \begingroup % Isolate color and font changes and the redefinitions.
    \commentfont
    \color[cmyk]{0.28,1,1,0.35}%
    \percentchar            % Produce the actual ‘%’ and
    \letpercent\percentchar % make all following ‘%’s do the same.
    \letreturn\end@comment  % Call \end@comment at end-of-line.
}
% \end@comment (alias for ^^M inside a comment) will end the comment
% started by \start@comment.  We make ^^M active temporarily so that
% the active version of ^^M gets "frozen" into \end@comment.
\begingroup \makeactive\^^M % Avoid ^^M’s from here on.
  \gdef\end@comment{\endgroup ^^M}%
\endgroup
\resetatcatcode % Make ‘@’ again inaccessible for use in macro names.

% Define \setuplistinghook to setup comments highlighting, line
% numbering and omitting the last (empty) line of input.
\def\setuplistinghook{\hlightcommentslisting \linenumberedlisting
                      \nolastlinelisting}
% Isn’t this fun?  This file typesets itself, with the extra bonus of
% the pretty-printed comments and numbered source lines!
\listing{lscommnt}
\bye
```

# Macro index

# Concept index