The sverb*, svcolour†, and svsplit‡ packages

Mark Wooding

4 September 2003

Contents

1 User guide **1**
1.1 The listing environment . 2
1.1.1 Configuring the listing environment 2
1.1.2 Choosing a different end-text 3
1.2 Writing text to a file . . . 4
1.3 The `\verbinput` command 4
1.4 The demo environment . . 5

2 Programmer interface **5**
2.1 Environment hooks 6
2.2 Reading the verbatim text 6

3 Colour support **6**

4 The svsplit package **7**

5 Implementation **7**
5.1 Options processing 8
5.2 Simple things 8
5.3 Tab character handling . . 9
5.4 Reading verbatim text . . 10
5.5 Listing environments . . . 14
5.6 The verbwrite environment 16
5.7 The demo environment . . 16
5.8 Loading the colour package 19
5.9 The svcolour package . . . 19
5.10 The svsplit package 20

A The GNU General Public License **23**

1 User guide

The sverb package provides some useful commands and environments for doing things with verbatim text. I prefer this code to the standard verbatim package (by Rainer Schöpf et al.) although I'm biased.

The package was written to fulfil a particular purpose: I wanted to be able to typeset ARM assembler code, 77 columns wide, on A5 paper, with the fields separated by *tab* characters. It's grown up fairly organically from that, and I've tidied it when I've seen the code get too ugly.

The current features are:

- A 'listing' environment which typesets verbatim text nicely.
- A command to read verbatim text from an external file.

*The sverb package is currently at version 1.4, dated 4 September 2003.
†The svcolour package is currently at version 1.4, dated 4 September 2003.
‡The svsplit package is currently at version 1.4, dated 4 September 2003.

- Support for arbitrary-sized chunks of text without overflowing TEX's memory.
- Support for *tab* characters in the verbatim text.
- An environment for typesetting demonstrations of LATEX markup.
- It all works correctly with the doc system for documenting LATEX packages.
- A fairly hairy but quite powerful programmer interface to the yukky bits of the package.

The interface is described in its own section, so that more timid readers can avoid it. That said, some of the stuff in this section gets rather technical.

Note that this package doesn't even try to do anything with short bits of verbatim text (as handled by the `\verb:...:` command). I have a separate package (syntax) which does all sorts of horrible things along those lines.

1.1 The listing environment

`listing` The main method for typesetting verbatim text is the listing environment. This works pretty much the same as the standard verbatim environment, with some exceptions, which are described below.

So that you know exactly what you're getting, here are the rules by which sverb decides what the verbatim text actually is:

- If there's any text, other than spaces, on the same line as the '`\begin{listing}`', then the contents of the environment begins immediately after the closing brace (with all leading spaces preserved). Otherwise, the text begins on the following line.
- If there is any text, other than spaces, before the '`\end{listing}`', but on the same line, this is considered to be the last line of the text; otherwise the text is presumed to have ended at the end of the previous line.
- Any text following the `\end{listing}` on the same line is thrown away. There are good reasons for this, but they're technical. Essentially there's nothing I can do about it.

Tab characters are supported within the environment: tab stops are set every eighth column, although this can be modified.

1.1.1 Configuring the listing environment

`\listingsize` The text size used in the listing environment is set by the `\listingsize` command. By default, this is set to `\footnotesize`, although you can redefine it in the document preamble, or it can be set in the document class. You can put other declarations (e.g., colours) here if you like.

`\listingindent` The amount by which the listing text is indented is controlled by the `\listingindent` length parameter. This is a fixed length, whose default value is 1 em.

`\listinghook` `\svafter` `\svline` `\svdoline` `listinglist` The `\listinghook` command is called by the listing environment (and `\verbinput` and demo) to set up the formatting of the listing. It can do any

The listing environment

. . . in the following code:

```
init            MOV     R0,#200         ;Version 2.00 please
                LDR     R1,=&4B534154   ;Magic number ('TASK')
                ADR     R2,appName      ;Find application name
                SWI     Wimp_Initialise ;Register as a WIMP task
```

The next step is to . . .

```
\dots in the following code:

\begin{listing}
init            MOV     R0,#200         ;Version 2.00 please
                LDR     R1,=&4B534154   ;Magic number ('TASK')
                ADR     R2,appName      ;Find application name
                SWI     Wimp_Initialise ;Register as a WIMP task
\end{listing}

The next step is to \dots
```

setting up it likes, and may configure `\svline` and `\svafter` as necessary. The macro `\svline` is run once for each line of verbatim text, with the line gathered into a box register, the number of which is given as an argument. The macro `\svafter` is called when processing has finished.

The default setting for `\listinghook` is (similar to)

```
\newcommand{\listinghook}{%
  \par%
  \begin{listinglist}%
  \listingsize%
  \renewcommand{\svline}{\listingline}%
  \renewcommand{\svafter}{\end{listinglist}}%
}
```

(see the source for the true definition). The default `\listingline` macro just writes out the line using `\svdoline`, which is a simple no-nonsense macro which just writes the text. As an example, you could say

```
\renewcommand{\listingline}{\leavevmode\llap{\strut\vrule\space}\svdoline}
```

to put a rule down the left-hand side of your listings.

The listinglist environment is a relatively straightforward list-based environment which sets pu the indentation of a listing. Feel free to redefine it.

1.1.2 Choosing a different end-text

`listing*` The listing environment is terminated by the exact character sequence '`\end{listing}`'. This isn't too much of a problem, unless you want to include this string in the text. This is achieved by the listing∗ environment, which allows you to specify the end-text to find as an argument.

For example:

The listing∗ environment

Type a listing as follows:

```
\begin{listing}
This is a listing.  Yes.
\end{listing}
```

```
Type a listing as follows:

\begin{listing*}{<end-listing*>}
\begin{listing}
This is a listing.  Yes.
\end{listing}
<end-listing*>
```

Don't include 'special' characters in your chosen end-text unless you know what you're doing.

1.2 Writing text to a file

`verbwrite` You can write verbatim text to a file using the verbwrite environment. The syntax is fairly straightforward:

`\begin{verbwrite}{`⟨*file-name*⟩`}` ... `\end{verbwrite}`

The text of the environment is written to the named file. The rules about where the text actually starts and ends are the same as for the listing environment.

There is also a ∗-variant, like listing∗, which allows you to choose the end-text. The end-text is the first argument, the filename comes second.

There is a restriction on the characters you can write to the file: they must all be considered 'printable' by TEX; otherwise they will be read back in as '`^^`⟨*chars*⟩' which isn't too good. Unfortunately, this includes tab characters, so you can't write them.[1]

For example:

```
\begin{verbwrite}{wrdemo.tmp}
This is some text written to
a file near the beginning of
the file.
\end{verbwrite}
```

1.3 The `\verbinput` command

`\verbinput` You can input a pre-prepared text file exactly as it is in the input using the `\verbinput` command. The filename is given as an argument. For example:

The `\verbinput` command

```
This is some text written to
a file near the beginning of
the file.
```

```
\verbinput{wrdemo.tmp}
```

[1] Well, not without doing serious surgery on TEX itself, anyway.

1.4 The demo environment

Package authors need to document their packages, and it's common to want to display examples showing the original text and the output side-by-side (or, when space doesn't permit this, one above the other). Both the LaTeX book and *The LaTeX Companion* contain such examples.

The demo environment allows such displays to be created easily. The syntax of the environment is as follows:

```
\begin{demo}[⟨shape⟩]{⟨title⟩} ... \end{demo}
```

The optional ⟨*shape*⟩ argument can be either '`w`' (wide), or '`n`' (narrow). A 'wide' shape places the input and output one above the other, while the 'narrow' shape puts them side-by-side. The default shape is 'narrow'. An attractive border is drawn around the display to finish it off nicely.

An example:

The demo environment

From the *TeXbook*

$$\sum_{p\ \text{prime}} f(p) = \int_{t>1} f(t)\,\mathrm{d}\pi(t)$$

```
\[ \sum_{p\;\rm prime}
   f(p) = \int_{t>1}
      f(t)\,{\rm d}\pi(t) \]
```

```
\begin{demo}{From the \textit{\TeX book}}
\[ \sum_{p\;\rm prime}
   f(p) = \int_{t>1}
      f(t)\,{\rm d}\pi(t) \]
\end{demo}
```

`demo*` As with the other environments created by this package, there's a ∗-variant which takes the end-text as an argument.

`\demohook` The `\demohook` does the same job for demo environments as `\listinghook` does for listings. The default version just says

```
\newcommand{\demohook}{\setlength{\listingindent}{0pt}\listinghook}
```

(near enough), which turns off the indentation for the listing (which would otherwise look rather odd).

2 Programmer interface

This section describes the publicly available routines provided by the sverb package. Routines not described here are libable to be changed or even removed without warning, so don't use them.

2.1 Environment hooks

Each of the environments created here works in the same way. For each environment foo, there's a main command responsible for doing the work, called `\sv@foo`. This is given all the arguments of the normal environment, and two more:

- The 'end-text' to search for, which marks the end of the environment.
- Some actions to perform after the text has been read and processed. This allows the calling macro to do some extra actions, like closing boxes, etc.

All the environments do is call the main command with appropriate arguments.

2.2 Reading the verbatim text

`\sv@read` The main scanning routine is `\sv@read`. It is called with three arguments:

- The end-text marking the end of the environment.
- The name of a macro (which must be a single token) which is called with a line of text as its single argument. This is given each line of text which is read from the environment in turn.
- A macro, or other sort of action, which is to be done when the text has been read and processed.

The macro `\sv@read` assumes that the caller has already made some provision for removing the category codes of the following text, by either calling `\@verbatim` or using the construction

```
\let\do=\@makeother
\dospecials
```

`\sv@safespc` Note that any space characters you read using `\sv@read` will be catcoded as `\active`. Normally this is OK because `\obeyspaces` (or `\@vobeyspaces`) will be in effect. If you're doing something more exotic, like writing text to a file or building a command string, you can call `\sv@safespc` which defines the active-space character to be a normal whitespace-space when expanded.

3 Colour support

There's now a little colour support in sverb. To use it, give the colour (or color) package option, or load the svcolour package.

`\svcolourline` Say `\svcolourline[`⟨*model*⟩`]{`⟨*colour*⟩`}{`⟨*box*⟩`}` to typeset ⟨*box*⟩ against a background of the given colour. This is a good thing to put in your `\listingline` command.

Coloured listings

Consider, for example, this more complicated program.

```
#include <stdio.h>

int main(void)
{
  puts("Hello, world!");
  return (0);
}
```

```
\renewcommand{\listingline}
  {\svcolourline[rgb]{1, 0.8, 0.9}}
Consider, for example, this more
complicated program.
\begin{listing}
#include <stdio.h>

int main(void)
{
  puts("Hello, world!");
  return (0);
}
\end{listing}
```

For coloured text rather than background, put a `\color` command in `\listinghook` itself.

4 The svsplit package

A new toy!

splitverb
splitverb*
\svsplitchars

The splitverb environment typesets verbatim material very slowly. On the plus side, however, it does know how to do simple line-breaking. It will break lines at spaces or tabs, or after any character listed in `\svsplitchars`. Continuation lines have the same initial intentation as the original. If a line has no 'good' breaking point, it's broken as late as possible, and a little hyphen is inserted.

The splitverb environment

```
The \package{url} package is
rather fine at splitting up
long URLs such as
  \url{http://www.excessus.
  demon.co.uk/tex}
though it can't do its thing in
the midst of verbatim text.  It
also doesn't cope when
  allthespacesinalongphrasehave-
  mysteriouslydisappeared!
```

```
\begin{multicols}{2}
\begin{splitverb}
The \package{url} package is rather fine at splitting up long URLs such as
  \url{http://www.excessus.demon.co.uk/tex}
though it can't do its thing in the midst of verbatim text.  It
also doesn't cope when
  allthespacesinalongphrasehavemysteriouslydisappeared!
\end{splitverb}
\end{multicols}
```

5 Implementation

This section defines several macros and environments which allow verbatim typing, with a high degree of configurability. OK, so this sort of thing's been done so often

before that it isn't true, but I don't really care.

```
1 ⟨*package⟩
```

5.1 Options processing

Notice options, load package.

```
2 \newif\ifsv@colour\sv@colourfalse
3 \DeclareOption{colour}{\sv@colourtrue}
4 \DeclareOption{color}{\sv@colourtrue}
5 \ProcessOptions
```

5.2 Simple things

To help us build funny macros which involve strange and different category codes, I'll write some simple macros which I can use while building my complicated and clever ones.

`\@cspecials` This macro is used to assist the definition of some of the environments. It makes '`\`', '`{`' and '`}`' into 'other' characters, and replaces them with '`|`', '`<`' and '`>`' respectively. Note that '`[`' and '`]`' aren't used, because they make defining commands which take optional arguments awkward. Note that we open a group here. This should be closed using `|endgroup` at the end of the special section.

```
6 \def\@cspecials{%
7   \begingroup%
8   \catcode`|0%
9   \catcode`<1%
10  \catcode`>2%
11  \catcode`\{12%
12  \catcode`\}12%
13  \catcode`\\12%
14 }
```

`\sv@addtobox` Add stuff to a horizontal box.

```
15 \def\sv@addtobox#1#2{\setbox#1\hbox{\unhbox#1\box#2}}
```

`\sv@emptybox` Clear out a horizontal box.

```
16 \def\sv@emptybox#1{\setbox#1\hbox{}}
```

`\sv@startlisting` This macro sets everything up nicely for a listing-type verbatim environment.

```
17 \def\sv@startlisting{%
18   \def\par{\@@par\penalty\interlinepenalty}%
19   \@@par%
20   \leftskip\@totalleftmargin%
21   \obeylines%
22   \@noligs%
23   \let\do\@makeother\dospecials%
24   \verbatim@font%
25   \frenchspacing%
26   \@vobeyspaces%
27   \settabwidth%
28   \catcode9\active%
```

```
  \lccode`\~9\lowercase{\let~\sv@vtab}%
  \lccode`\~13\lowercase{\let~\vinput@cr}%
  \interlinepenalty500%
}
```

5.3 Tab character handling

One of the things we want to do here is handle tab characters properly. (Here, 'properly' means 'moving to the next column which is a multiple of eight', the way these things were always meant to.)

`\settabwidth` The tabs used by our tabbed verbatim environments are set up by this routine. It sets the tab width parameter `\svtab` to 8 times the width of a `\tt` space. If you really want, you can redefine this macro.

```
\newdimen\svtab
\def\settabwidth{\setbox\z@\hbox{\texttt{\space}}\svtab8\wd\z@}
```

`\sv@vtab` Here we handle tabs inside verbatim environments. We expect to be inside `\box` 0. This is padded to the correct width and contributed to `\box` 2; `\box` 0 is then cleared and re-entered.

The idea is that you make tab active, and set it to this macro. We stop the current box, stretch it to the right width, and start another one straight after, so nobody knows the difference. The code here is straight from Appendix D of *The TEXbook*.

```
\def\sv@vtab{%
  \hfill\egroup%
  \@tempdima\wd\z@%
  \divide\@tempdima\svtab%
  \multiply\@tempdima\svtab%
  \advance\@tempdima\svtab%
  \wd\z@\@tempdima%
  \sv@addtobox\tw@\z@%
  \setbox\z@\hbox\bgroup%
}
```

`\verbinput` We allow input from a file, by the `\verbinput` command. We display the text pretty much the same as the listing environment below.

We set tab and return active, and get them to do appropriate things. This isn't actually all that hard.

```
\def\verbinput{\listinghook\@ifstar{\verbinput@\@input}{\verbinput@\input}}
\def\verbinput@#1#2{%
  \sv@startlisting%
  \setbox\z@\hbox\bgroup%
  #1{#2}%
  \sv@stripspc%
  \egroup%
  \sv@addtobox\tw@\z@%
  \ifdim\wd\tw@=\z@\listingline\tw@\fi%
  \svafter%
}
```

\vinput@cr This macro handles return characters while inputting text in \verbinput. We just output our current box, and start another.

```
\def\vinput@cr{%
  \egroup%
  \sv@addtobox\tw@\z@%
  \listingline\tw@%
  \sv@emptybox\tw@%
  \setbox\z@\hbox\bgroup%
}
```

5.4 Reading verbatim text

The traditional way of reading verbatim text is to use a delimited argument, as described in the *TEXbook*. This works well-ish if the text isn't very long. A better solution would be to pick out the text line-by-line and process it like that. So this is what we do.

\matcher For long verbatim environments, we need to be able to find the end text. This is rather tricky. The solution here is rather horrible. The environment picks out each line of the text at a time, as an argument, and tests to see if it contains the text we're after. We do the test in a particularly yukky way: we add the actual target text to the end of the line, and inspect the text following the match to see if the match is at the end.

The \matcher macro creates a 'matcher' which will test strings to see if they contain something interesting.

To create a matcher, say \matcher{⟨*cmd-name*⟩}{⟨*target*⟩}{⟨*process-cmd*⟩}. The command ⟨*cmd-name*⟩ accepts a line of text as an argument and calls the ⟨*process-cmd*⟩ with the text of the line before the match, or the whole lot. It also sets \@ifmatched appropriately.

(Having spent ages coming up with this cruft myself, I found some very similar, but slightly better, code in Appendix D. So I've changed mine to match Donald's. Anyway, credit where it's due: cheers Don.)

```
\newif\if@matched
\def\matcher#1#2#3{%
  \expandafter\def\csname\string#1$match\endcsname##1#2##2##3\end{%
    \ifx##2\relax%
      \@matchedfalse%
    \else%
      \@matchedtrue%
    \fi%
    #3{##1}%
  }%
  \expandafter\def\expandafter#1\expandafter##\expandafter1\expandafter{%
    \csname\string#1$match\endcsname##1#2\relax\end%
  }%
}
```

\sv@stripspc This macro strips any trailing glue in the current horizontal list. This is fairly simple, actually: we just loop while glue is the last item. It's slightly complicated by penalties which TEX puts into the list between the glue items, but we just remove them too.

```
\def\sv@stripspc{%
  \unpenalty%
  \ifdim\lastskip=\z@\else%
    \unskip\expandafter\sv@stripspc%
  \fi%
}
```

`\sv@percent` This macro strips a single leading percent character if there is one, and if the doc package is loaded. We store the possibly stripped text in `\@tempa`.

```
\begingroup
\catcode`\%=12
\gdef\sv@percent#1#2\relax
    {\ifx\check@percent\@@undefined
                        \ifx#1\relax\def\@tempa{}\else
                            \def\@tempa{#1#2}\fi\else
                        \ifx#1\relax\def\@tempa{}\else
                            \ifx#1%\def\@tempa{#2}\else
                                \def\@tempa{#1#2}\fi\fi\fi}
\endgroup
```

`\@isspaces` We want to avoid writing the first and last lines of the environment to the file if there's nothing in them. To do this, we need to know whether a piece of text contains only space characters. This macro does this, in a rather nasty way. See the other macros below for details of how this works.

We define `\sv@safespc` at the same time: this makes space active and expand to a space character which is not active. Neat, huh?

```
\begingroup
\lccode`\~32
\lccode`\!32
\lowercase{%
  \endgroup
  \def\@isspaces#1{%
    \ifx#1\relax%
      \def\@tempb{\@tempswafalse}%
    \else\ifx#1~%
      \let\@tempb\@isspaces%
    \else%
      \def\@tempb##1\relax{}%
    \fi\fi%
    \@tempb%
  }
  \def\sv@safespc{%
    \catcode32\active%
    \def~{ }%
  }
}
```

`\sv@read` This macro does the main job of reading a chunk of verbatim text. You call it like this:

`\sv@read{`⟨*end-text*⟩`}{`⟨*process-line-proc*⟩`}{`⟨*end-proc*⟩`}`

The ⟨*end-text*⟩ is the text to find at the end of the 'environment': we stop when we find it.

The ⟨*process-line-proc*⟩ is a macro which is passed as an argument each line which we read from the text.

The ⟨*end-proc*⟩ is a macro to call once we've finished reading all of the text. This can tidy up an environment or close a file or whatever.

We read the text by picking out newlines using a delimited macro. We have to be a little clever, because newlines are active in verbatim text.

We will also strip '%' signs off the beginning if the doc package is here (doc tries to play with LaTeX's verbatim stuff, and doesn't understand the way we do things).

```
\def\sv@read#1#2#3{%
```

This code does all sorts of evil things, so I'll start by opening a group.

```
  \begingroup%
```

So that I can spot the end-text, I'll create a matcher macro.

```
  \matcher\@match{#1}\sv@read@ii%
```

So that I can identify line ends, I'll make them active. I'll also make spaces active so that they can expand to whatever they ought to expand to (spaces in files, or funny ␣ characters or whatever.

```
  \catcode13\active%
  \catcode32\active%
```

I'll use the `\if@tempswa` flag to tell me whether I ought to output the current line. This is a little messy, so I'll describe it later. I'll initialise it to false because this is the correct thing to do.

```
  \@tempswafalse%
```

Most of the job is done by two submacros. I'll define them in terms of my current arguments (to save lots of token munging). The first just extracts the next line (which ends at the next newline character) and tries to match it.

```
  \lccode`\~13\lowercase{%
    \def\sv@read@i##1~{\@match{##1}}%
  }%
```

The results of the match get passed here, along with the text of the line up to the matched text.

```
  \def\sv@read@ii##1{%
```

The first job to do is to maybe strip off percent signs from the beginning, to keep doc happy.

```
    \sv@percent##1\relax\relax%
```

Now I need to decide whether I ought to output this line. The method goes like this: if this is the first line (`\if@tempswa` is false) or the last (`\if@matched` is true), *and* the text consists only of spaces, then I'll ignore it.

The first thing to do is to notice the last line – if `\if@matched` is true, then I'll make `\if@tempswa` false to make the first-line and last-line cases work the same way.

```
    \if@matched\@tempswafalse\fi%
```

Now if this is the first or last line, I'll examine it for spaces. This is done in a separate macro. It will set `\if@tempswa` false if the text contains only spaces.

```
    \if@tempswa\else\@tempswatrue\expandafter\@isspaces\@tempa\relax\fi%
```

Now, if `\if@tempswa` is still true, perform the ⟨*process-line-proc*⟩ on the line of text. I'll provide a group, so that it doesn't upset me too much.

```
    \if@tempswa%
      \begingroup%
      \expandafter#2\expandafter{\@tempa}%
      \endgroup%
    \fi%
```

The next line won't be the first one, so I'll set the flag true in readiness.

```
    \@tempswatrue%
```

Now, if that wasn't the last line, go round again; otherwise end the group I started ages ago, and do the user's ⟨*end-proc*⟩.

```
    \if@matched\def\@tempa{\endgroup#3}\else\let\@tempa\sv@read@i\fi%
    \@tempa%
  }%
```

Now to start the thing up. I'll read the first line.

```
  \sv@read@i%
}
```

`\sv@readenv` This macro works out an appropriate end-text for the current environment. If you say `\sv@readenv{`⟨*macro-name*⟩`}`, it will expand do

⟨*macro-name*⟩`{\`$_{12}$`end{`$_{12}$⟨*current-env-name*⟩`}`$_{12}$`}{\end{`⟨*current-env-name*⟩`}}`

Easy, no?

This is all done with mirrors. No, err… it's done with `\expandafter`.

```
\begingroup
\lccode`\<=`\{
\lccode`\>=`\}
\lccode`\|=`\\
\lowercase{\endgroup
\def\sv@readenv#1{\expandafter\sv@readenv@i\expandafter{\@currenvir}{#1}}
\def\sv@readenv@i#1#2{#2{|end<#1>}{\end{#1}}}
}
```

`\sv@verbline` This macro typesets a line in a verbatim way, so you can construct a real verbatim environment from it. It's a bit tricky in the way that it catches the last line. Don't worry about this: it's easy really. Note the `\relax` after the `\par` – this is because doc tries to do clever things with `\par` to strip '`%`' signs out.

```
\def\sv@verbline#1{%
  \sv@emptybox\tw@%
  \setbox\z@\hbox{#1\sv@stripspc}%
  \sv@addtobox\tw@\z@%
  \if1\ifdim\wd\tw@=\z@\if@matched0\else1\fi\else1\fi%
    \svline\tw@\relax%
  \fi%
}
```

5.5 Listing environments

The listing environment is our equivalent of the standard verbatim environment. We do some slightly cleverer things, though, to make sure (for example) that even text which contains `\end{listing}` can be typeset.

`\listinghook` Set everything up as required. This is here for customization. The underlying machinery doesn't mess with this directly, but assumes that `\svline` and `\svafter` are set up appropriately.

```
\def\listinghook{%
  \par%
  \begingroup
  \listinglist%
  \listingsize%
  \let\svline\listingline%
  \def\svafter{\endlistinglist\endgroup}%
}
```

`\listinglist` `listinglist` This defines the layout for the listing environment. It starts a list with the appropriate shape. It's also made into an environment, so that the end-paragraph-environment bits work correctly.

The `\listingindent` length parameter sets up the indentation of the listings. If there's a `\parindent` setting, I'll line listings up with that; otherwise I'll just choose something which looks right.

```
\newdimen\listingindent
\AtBeginDocument{%
  \ifdim\parindent=\z@\listingindent1em\else\listingindent\parindent\fi%
}
```

Now to define a size hook for the environment. This is fairly simple stuff.

```
\ifx\listingsize\@@undefined
  \let\listingsize\footnotesize
\fi
```

Now to define the environment itself. Suppress the indentation if we're first thing on a new list item, so that the listing lines up with everything else.

```
\def\listinglist{%
  \list{}{%
    \if@inlabel%
      \leftmargin\z@%
    \else%
      \leftmargin\listingindent%
    \fi%
    \rightmargin\z@%
    \labelwidth\z@%
    \labelsep\z@%
    \itemindent\z@%
    \listparindent\z@%
    \let\makelabel\relax%
    \parsep\z@skip%
  }%
  \parfillskip\@flushglue%
  \item\relax%
```

```
}
\let\endlistinglist\endlist
```

`\svline` `\svdoline` `\listingline` The simple spit-out-a-line macro.

```
\def\svdoline#1{\leavevmode\box#1\par}
\let\svline\svdoline
\let\listingline\svline
```

`\svafter` This is called when the machinery finishes. A default is set for safety's sake.

```
\let\svafter\relax
```

`listing` The listing environment is the only real verbatim-like environment we create will all this kit, although it does the job very nicely.

The environment indents its contents slightly, unlike verbatim, and uses a smaller typeface in an attempt to fit 77-column text on an A5 page. There is also a $*$-variant, which allows you to specify the terminating text. This enables you to include absolutely any text in the environment, including `\end{listing}`.

First, we must define the `\listing` command.

```
\def\listing{\listinghook\sv@readenv\sv@listing}
```

Now we define the `\@listing` command, which does most of the work. We base the listing environment on a list.

```
\def\sv@listing#1#2{\sv@startlisting\sv@read{#1}\sv@verbline{\svafter#2}}
```

Now we define the starred version. The command name needs to include the '`*`' character, so we must use `\csname`. There's some hacking here to allow us to read the name using the appropriate catcodes for otherwise normal characters: LaTeX activates some characters and makes them typeset themselves to suppress some ligaturing.

```
\expandafter\def\csname listing*\endcsname{%
  \listinghook\begingroup\@noligs\listing@star%
}
\def\listing@star#1{\endgroup\sv@listing{#1}{\end{listing*}}}
```

`ignore` The ignore environment entirely ignores its contents. Anything at all may be put into the environment: it is discarded utterly.

We define some macros for defining ignoring environments, because this can be useful for version control, possibly.

```
\def\sv@ignore#1#2{%
  \@bsphack%
  \let\do\@makeother\dospecials%
  \sv@read{#1}\@gobble{\@esphack#2}%
}
\def\ignore{\sv@readenv\sv@ignore}
\def\ignoreenv#1{%
  \expandafter\let\csname #1\endcsname\ignore%
}
\def\unignoreenv#1{%
  \expandafter\def\csname #1\endcsname{\endgroup}%
  \expandafter\def\csname end#1\endcsname%
      {\begingroup\def\@currenvir{#1}}%
}
```

5.6 The verbwrite environment

The verbwrite environment allows text to be written to a file in a verbatim way. Note that tab characters don't work, because TEX refuses to be nice.

`\sv@write` As seems to be traditional now, we first define a general hookable macro which allows a caller to specify the end-text and what to do afterwards.

```
\newwrite\sv@writefile
\def\sv@write#1#2{%
  \begingroup%
  \@bsphack%
  \let\do\@makeother\dospecials%
  \sv@safespc%
  \sv@read{#1}\sv@writeline{\sv@endwrite#2}%
}
\def\sv@writeline#1{%
  \immediate\write\sv@writefile{#1}%
}
\def\sv@endwrite{%
  \@esphack%
  \endgroup%
}
```

`verbwrite` Now we can define the actual environment. We define a $*$-variant which allows the user to specify the end-text, just to make sure.

```
\def\verbwrite#1{%
  \immediate\openout\sv@writefile#1\relax%
  \sv@readenv\sv@write%
}
\def\endverbwrite{\immediate\closeout\sv@writefile}
\expandafter\def\csname verbwrite*\endcsname#1#2{%
  \immediate\openout\sv@writefile#2\relax%
  \sv@write{#1}{\immediate\closeout\sv@writefile\end{verbwrite*}}%
}
```

5.7 The demo environment

By way of tying all of this together, I present an environment for displaying demonstrations of LATEX markup. We read the contents of the environment, write it to a temporary file, and read it back twice, typesetting it the first time and displaying it verbatim the second time.

`\sv@demoname` This macro expands to the filename to use for the temporary data. To allow the package documentation to demonstrate the demo environment itself, we need to keep a nesting count. This avoids too much hackery, which unfortunately appears to plague all of my TEX code.

```
\newcount\sv@nestcount
\def\sv@demoname{\jobname-demo\number\sv@nestcount.tmp}
```

`\sv@demo` As for listing, we do all the business through a private macro. This is good because it means we can leave the main macro readable. The argument is the end-text to spot.

```
\def\sv@demo#1#2{%
  \@ifnextchar[{\sv@demo@i{#1}{#2}}{\sv@demo@i{#1}{#2}[n]}%
}
\def\sv@demo@i#1#2[#3]#4{%
  \advance\sv@nestcount by\@ne%
  \immediate\openout\sv@writefile\sv@demoname\relax%
  \sv@write{#1}{%
    \immediate\closeout\sv@writefile%
    \sv@dodemo{#2}{#3}{#4}%
  }%
}
```

`demo` This is the real environment. We provide demo$*$ too, to allow the user to choose the end-text.

```
\def\demo{\let\@demohook\demohook\sv@readenv\sv@demo}
\expandafter\def\csname demo*\endcsname#1%
  {\let\@demohook\demohook\sv@demo{#1}{\end{demo*}}}
```

`\demohook` Like `\listinghook`. So much so that we just call it, but first ensure that the indent is zero (otherwise it looks really odd!).

```
\def\demohook{\listingindent\z@\listinghook}
```

`\sv@dodemo` First, let's define some common bits of code in the stuff below. The minipages used to typeset the material has some clever stuff to avoid strange spacing in the output.

```
\def\sv@demosmp{%
  \begin{minipage}[t]{\@tempdima}%
  \vskip8\p@%
  \hrule\@height\z@%
  \raggedright%
  \vbox\bgroup%
}
\def\sv@demoemp{%
    \par\unpenalty\unskip%
  \egroup%
  \vskip8\p@%
  \hrule\@height\z@%
  \end{minipage}%
}
```

This is the macro which actually typesets the demonstration.

```
\def\sv@dodemo#1#2#3{%
```

Now work out some values. We set `\hsize` to the line width leaving 2 em of space on either side. The size of the minipages is calculated depending on the shape of the demonstration. This is all fairly simple.

```
  \begingroup%
  \@tempdima\linewidth%
  \advance\@tempdima-2em%
  \hsize\@tempdima%
  \if#2w%
    \advance\@tempdima-2em%
  \else%
```

```
    \advance\@tempdima-3em%
    \divide\@tempdima2%
  \fi%
```

Now we open a big vertical box, and put in a header to mark off the demonstration.

```
  \par%
  \setbox\z@\hbox{\strut\enspace#3\enspace\strut}%
  \@tempdimb.5\dp\z@%
  \advance\@tempdimb-.5\ht\z@%
  \ht\z@\@tempdimb\dp\z@\@tempdimb%
  \noindent\hskip1em\vtop{%
    \hb@xt@\hsize{%
      \hrulefill%
      \raise\@tempdimb\box\z@%
      \hrulefill%
    }%
    \nointerlineskip%
    \hb@xt@\hsize{\vrule\@height5\p@\hfil\vrule\@height5\p@}%
    \nointerlineskip%
```

Now we insert the output text in the first minipage. I'll force '%' to be a comment character, in case something like doc has had its wicked way.

```
    \vskip-\parskip%
    \noindent\hbox{}\hskip1em%
    \sv@demosmp%
    \catcode`\%14\relax%
    \@input{\sv@demoname}%
    \sv@demoemp%
```

Insert some kind of separation between the two. In 'wide' format, we start a new line, and put a ruleoff between the two. In 'narrow' format, we just leave some space.

```
    \if#2w%
      \vskip8\p@\hrule\vskip8\p@%
      \noindent\hbox{}%
    \fi%
    \hskip1em%
```

Now we put the verbatim copy of the text in the other minipage.

```
    \sv@demosmp%
    \@demohook%
    \verbinput@\@input\sv@demoname%
    \sv@demoemp%
    \par%
    \nointerlineskip%
    \hb@xt@\hsize{\vrule\@height5\p@\hfil\vrule\@height5\p@}%
    \hrule%
  }%
  \endgroup%
  \par%
  \vskip\baselineskip%
  #1%
}
```

5.8 Loading the colour package

If requested, we load the svcolour package here. This ensures that it can patch this code if it needs to.

```
\ifsv@colour
  \RequirePackage{svcolour}
\fi
```

That's all there is. Have fun.

⟨/package⟩

5.9 The svcolour package

This is in a separate package to avoid dragging in the color package if it's unwanted.

I prefer English spellings. Here's a trivial redirection for Americans.

```
⟨*color⟩
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{svcolour}}
\ProcessOptions
\RequirePackage{svcolour}
⟨/color⟩
```

And now we can start the thing properly.

```
⟨*colour⟩
\RequirePackage{color}
```

`\@snarfcolour` Reading a colour specification is something we'll need to do a few times, so an abstraction is useful. Its single argument is a continuation to which we pass a colour-spec acceptable to the `\color` command. (This is the same code as found in the mdwtab package. Remember to keep them in step.)

```
\def\@snarfcolour#1{%
  \@ifnextchar[{\@snarfcolour@i{#1}}{\@snarfcolour@ii{#1}{}}%
}
\def\@snarfcolour@i#1[#2]{\@snarfcolour@ii{#1}{[#2]}}
\def\@snarfcolour@ii#1#2#3{#1{#2{#3}}}
```

`\svcolourline` `\svcolorline` Snarf the option, and plot the coloured bar. Note the penalties which are meant to stick the glue and leaders onto the colour specials.

```
\def\svcolourline{\@snarfcolour\svcl@i}
\def\svcl@i#1#2{%
  \skip@\wd#2%
  \advance\skip@\parfillskip%
  \advance\skip@.2em%
  \strut%
  \kern.2em%
  \begingroup\color#1\nobreak\leaders\vrule\hskip\skip@\endgroup%
  \nobreak\hskip-\skip@%
  \kern.2em%
  \box#2%
  \nobreak\hskip-\rightskip\vadjust{}%
  \par%
}
\let\svcolorline\svcolourline
```

Done!

⟨/colour⟩

5.10 The svsplit package

⟨*split⟩

```
\RequirePackage{sverb}
```

splitverb
splitverb*

The basic environments are simple enough.

```
\def\splitverb{\listinghook\sv@readenv\splitverb@}
\expandafter\def\csname splitverb*\endcsname%
  {\listinghook\begingroup\@noligs\svsplit@star}
\def\svsplit@star#1{\endgroup\splitverb@{#1}{\end{splitverb*}}}
```

\splitverb@

Even this isn't so bad, really.

```
\def\splitverb@#1#2{\sv@startlisting\sv@read{#1}\svsplit@line{\svafter#2}}
```

\svsplit@line

For the sake of readability (and maybe saving a few tokens), we define some synonyms for TEX's scratch registers. `\svsplit@remain` will be a `\global` register containing the remaining horizontal space on the line; `\svsplit@indent` is a local register containing the amount of initial whitespace on the line.

```
\dimendef\svsplit@remain=1
\dimendef\svsplit@indent=2
```

The switch `\svsplit@` is set if we've found a good place to split the current line.

```
\newif\ifsvsplit@
```

And finally a delimiter. This is the same one I use everywhere else.

```
\def\q@delim{\q@delim}
```

```
\begingroup
\catcode`\~=\active \lccode`\~=32
\catcode`\!=\active \lccode`\!=9
\lowercase{\endgroup
```

So far, so good. The `\svsplit@line` macro is given a line of text. We initialize `\svtab` to be a *single* space, `\svsplit@remain` to be the text width, and `\svsplit@indent` to zero. Then we embark on the first loop, which attempts to find the width of the leading whitespace.

```
\def\svsplit@line#1{%
  \divide\svtab8%
  \global\svsplit@remain\linewidth%
  \svsplit@indent\z@%
  \sv@emptybox\tw@%
  \let\next@\svsplit@findindent%
  \next@#1\q@delim%
}
```

A straightforward tail-recursive loop finds out how much whitespace there is at the start of the current line. Note that `\next@` is already set up for the optimized case of continuing the loop. Also, if we reach the end then this is a blank line, so

only emit something if we didn't see the end-marker. This is the only place we need to check for this.

```
\def\svsplit@findindent#1{%
  \ifx~#1%
    \advance\svsplit@indent\svtab%
  \else\ifx!#1%
    \dimen@8\svtab%
    \divide\svsplit@indent\dimen@%
    \multiply\svsplit@indent\dimen@%
    \advance\svsplit@indent\dimen@%
  \else\ifx\q@delim#1%
    \if@matched\else\svline\tw@\fi%
    \let\next@\relax%
  \else%
    \def\next@{\svsplit@scanline{#1}}%
  \fi\fi\fi%
  \next@%
}
```

Now we have to actually scan the line to find breakpoints. We build the current unbreakable chunk in `\box` 0. When we find a breakpoint, we close the box, maybe stretch it to take into account trailing space, and attach it to `\box` 2, which is gathering the current line. If `\svsplit@remain` hits zero then we flush `\box` 2 to the output and continue on the next line with a (more-or-less) clean slate.

If there's no breakpoint then we're hosed. In that case, we just insert a (`\normalfont`) hyphen and eject what we've got.

Note that this assumes that the indentation will fit. If not, then we're deeply stuffed.

```
\def\svsplit@scanline{%
  \svsplit@false%
  \let\next@\svsplit@char%
  \setbox\z@\hbox\bgroup%
    \kern\svsplit@indent%
    \global\advance\svsplit@remain-\svsplit@indent%
    \next@%
}
```

Scanning a character isn't so bad, if we take it a step at a time.

```
\def\svsplit@char#1{%
```

If the character is a space or a tab, then we call `\svsplit@space` which knows about adding breakable whitespace. For tabs, this involves computing the correct tab size.

```
    \ifx~#1%
      \svsplit@space\svtab%
    \else\ifx!#1%
      \@tempdima\linewidth%
      \advance\@tempdima-\svsplit@remain%
      \@tempdimb\@tempdima%
      \dimen@8\svtab%
      \divide\@tempdimb\dimen@%
      \multiply\@tempdimb\dimen@%
      \advance\@tempdimb\dimen@%
```

```
      \advance\@tempdimb-\@tempdima%
      \svsplit@space\@tempdimb%
```

We might have reached the end of the line. If so, then we finish off.

```
    \else\ifx\q@delim#1%
      \let\next@\svsplit@done%
```

Otherwise it's a normal character. If there's not enough space then force a break.

```
    \else%
      \ifdim\svsplit@remain<2\svtab%
        \ifsvsplit@\else\normalfont-\svsplit@break\fi%
        \svsplit@eject%
      \fi%
```

Insert the character and decrement the distance-left register.

```
      #1%
      \global\advance\svsplit@remain-\svtab%
```

Now we see if it's a breakable-after character and if so mark it as being breakable.

```
      \def\temp@##1#1##2\q@delim%
        {\ifx\q@delim##2\q@delim\else\svsplit@break\fi}%
      \expandafter\temp@\svsplitchars#1\q@delim%
```

And with that, we're done.

```
    \fi\fi\fi%
    \next@%
}
```

Our next macro is the break-insertion subroutine, which is quite easy.

```
\def\svsplit@break{%
  \egroup%
  \sv@addtobox\tw@\z@%
  \svsplit@true%
  \setbox\z@\hbox\bgroup%
}
```

Now we add space to the current box. The argument is a dimen register.

```
\def\svsplit@space#1{%
    \ifdim\svsplit@remain>#1\kern#1\global\advance\svsplit@remain-#1\fi%
    \svsplit@break%
    \ifdim\svsplit@remain>#1\else\svsplit@eject\fi%
}
```

We now come to a slightly involved piece of code, which is how to flush out a line, and then fix up the registers for the next line correctly.

```
\def\svsplit@eject{%
  \egroup%
  \svline\tw@%
  \sv@emptybox\tw@%
  \svsplit@false%
  \setbox\z@\hbox\bgroup%
    \kern\svsplit@indent%
```

```
441     \global\svsplit@remain\linewidth%
442     \global\advance\svsplit@remain-\svsplit@indent%
443     \global\advance\svsplit@remain-\wd\z@%
444     \unhbox\z@%
445 }
```

Finally, how to finish the line and go home.

```
446 \def\svsplit@done{%
447   \egroup%
448   \sv@addtobox\tw@\z@%
449   \svline\tw@%
450 }
```

End the `\lowercase` hack.

```
451 }
```

Finally, set the breakable characters to something plausible.

```
452 \def\svsplitchars{:/.}
```

And with that, we're done!

453 ⟨/split⟩

Mark Wooding, 4 September 2003

Appendix

A The GNU General Public License

The following is the text of the GNU General Public License, under the terms of which this software is distributed.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered

by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

End of Terms and Conditions

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> one line to give the program's name and a brief idea of what it does.
> Copyright (C) yyyy name of author

> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) yyyy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> signature of Ty Coon, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Symbols

\! 95, 361

\% 84, 294
\< 138
\> 139
\@@par 18, 19
\@@undefined 86, 165
\@bsphack 198, 214
\@cspecials 6
\@currenvir 142, 209
\@demohook 248, 250, 303
\@esphack 200, 223
\@flushglue 183
\@gobble 200
\@height 255, 263, 289, 308
\@ifnextchar 238, 328
\@ifstar 45
\@input 45, 295, 304
\@isspaces 93, 125
\@makeother 23, 199, 215
\@match 115, 120
\@matchedfalse 67
\@matchedtrue 69
\@noligs 22, 194, 352
\@snarfcolour 327, 332
\@snarfcolour@i 328, 330
\@snarfcolour@ii 328, 330, 331
\@tempa 87–91, 125, 128, 132, 133
\@tempb 100, 102, 104, 106
\@tempdima 37–41, 253, 268–270, 272, 274, 275, 399–401, 406
\@tempdimb 279–281, 285, 401, 403–407
\@tempswafalse 100, 118, 124
\@tempswatrue 125, 131
\@totalleftmargin 20
\@vobeyspaces 26
\\ 13, 140
\{ 11, 138
\} 12, 139
\| 140
\~ 29, 30, 94, 119, 360

A

\active 28, 109, 116, 117, 360, 361
\AtBeginDocument 162

B

\baselineskip 313
\begin 253

C

\check@percent 86
\color 339
\CurrentOption 321

D

\DeclareOption 3, 4, 321
\demo 248
demo (environment) 248
demo* (environment) *5*
\demohook *5*, 248, 250, 251
\dimen@ 375–378, 402–405
\dimendef 355, 356
\do 23, 199, 215
\dospecials 23, 199, 215

E

\end . 65, 74, 143, 196, 233, 250, 264, 353
\endlist 186
\endlistinglist 159, 186
\endverbwrite 230
\enspace 278
environments:
demo 248
demo* *5*
ignore 197
listing *2*, 191
listing* *3*
listinglist *2*, 161
splitverb *7*, 350
splitverb* *7*, 350
verbwrite *4*, 226

F

\footnotesize 166
\frenchspacing 25

H

\hb@xt@ 283, 289, 308
\hrule 255, 263, 298, 309
\hrulefill 284, 286

I

\if@inlabel 170
\if@matched 63, 124, 132, 149, 380
\if@tempswa 125, 126
\ifsv@colour 2, 316
\ifsvsplit@ 357, 412
\ignore 202, 204
ignore (environment) 197
\ignoreenv 203
\input 45
\interlinepenalty 18, 31
\item 184
\itemindent 178

L

\labelsep 177
\labelwidth 176
\lastskip 79
\lccode 29, 30, 94, 95, 119, 138–140, 360, 361

\leaders 339
\leavevmode 187
\leftmargin 171, 173
\leftskip 20
\linewidth 268, 365, 399, 441
\list 169
\listing 191
listing (environment) *2*, 191
listing* (environment) *3*
\listing@star 194, 196
\listinghook *2*, 45, 153, 191, 194, 251, 350, 352
\listingindent . . *2*, 161, 163, 173, 251
\listingline 53, 59, 158, 187
\listinglist 156, 161
listinglist (environment) *2*, 161
\listingsize *2*, 157, 165, 166
\listparindent 179
\lowercase . . 29, 30, 96, 119, 141, 362

M

\makelabel 180
\matcher 63, 115

N

\newcount 235
\newdimen 33, 161
\newif 2, 63, 357
\newwrite 211
\next@ 368, 369, 381, 383, 385, 389, 393, 409, 421
\nobreak 339, 340, 343
\noindent 282, 292, 299
\nointerlineskip 288, 290, 307
\normalfont 412
\number 236

O

\obeylines 21

P

\par 18, 154, 187, 260, 277, 306, 312, 344
\parfillskip 183, 335
\parindent 163
\parsep 181
\parskip 291
\PassOptionsToPackage 321
\penalty 18
\ProcessOptions 5, 322

Q

\q@delim . . 358, 369, 379, 408, 417–419

R

\raggedright 256
\raise 285
\RequirePackage . . . 317, 323, 326, 349
\rightmargin 175
\rightskip 343

S

\settabwidth 27, 33
\skip@ 334–336, 339, 340
\splitverb 350
splitverb (environment) *7*, 350
splitverb* (environment) *7*, 350
\splitverb@ 350, 353, 354
\strut 278, 337
\sv@addtobox 15, 42, 52, 58, 148, 425, 448
\sv@colourfalse 2
\sv@colourtrue 3, 4
\sv@demo 237, 248, 250
\sv@demo@i 238, 240
\sv@demoemp 259, 296, 305
\sv@demoname 235, 242, 295, 304
\sv@demosmp 252, 293, 302
\sv@dodemo 245, 252
\sv@emptybox 16, 60, 146, 367, 437
\sv@endwrite 217, 222
\sv@ignore 197, 202
\sv@listing 191, 192, 196
\sv@nestcount 235, 236, 241
\sv@percent 83, 123
\sv@read *6*, 113, 192, 200, 217, 354
\sv@read@i 120, 132, 135
\sv@read@ii 115, 122
\sv@readenv 137, 191, 202, 228, 248, 350
\sv@readenv@i 142, 143
\sv@safespc *6*, 108, 216
\sv@startlisting 17, 47, 192, 354
\sv@stripspc 50, 77, 147
\sv@verbline 145, 192
\sv@vtab 29, 35
\sv@write 211, 228, 233, 243
\sv@writefile 211, 220, 227, 230, 232, 233, 242, 244
\sv@writeline 217, 219
\svafter *2*, 54, 159, 190, 192, 354
\svcl@i 332, 333
\svcolorline 332
\svcolourline *6*, 332
\svdoline *2*, 187
\svline . *2*, 150, 158, 187, 380, 436, 449
\svsplit@break 412, 418, 423, 431
\svsplit@char 389, 395
\svsplit@done 409, 446
\svsplit@eject 413, 432, 434
\svsplit@false 388, 438
\svsplit@findindent 368, 371

\svsplit@indent 356, 366, 373, 376–378, 391, 392, 440, 442
\svsplit@line 354, 355
\svsplit@remain . . . 355, 365, 392, 400, 411, 416, 430, 432, 441–443
\svsplit@scanline 383, 387
\svsplit@space 397, 407, 429
\svsplit@star 352, 353
\svsplit@true 426
\svsplitchars 7, 419, 452
\svtab 33, 34, 38–40, 364, 373, 375, 397, 402, 411, 416

T

\temp@ 417, 419
\texttt 34

U

\unignoreenv 206
\unpenalty 78, 260

V

\vadjust 343
\verbatim@font 24
\verbinput 4, 45
\verbinput@ 45, 46, 304
\verbwrite 226
verbwrite (environment) 4, 226
\vinput@cr 30, 56
\vrule 289, 308, 339