The enverb package

read an environment verbatim

Jonathan P. Spratte[⌜]

2025-01-10 v1.1

Abstract

The **enverb** package provides a simple yet effective way to collect the verbatim contents of an environment into a macro, which the user can then pick up to implement additional functionality, for instance a simple output routine or rescanning the contents into code listings or for typesetting alike. Arbitrary contents can be placed on either end of each line allowing flexible line-oriented processing.

Contents

[⌜]jspratte@yahoo.de

1 Documentation

Originally I was unsatisfied by the degree of freedom and customisation present in the available verbatim-environment providing packages like listings, minted, *etc.*, when one wants to document TEX code and show the results next to it (or I didn't bother to dig through their manuals). Also I didn't like them producing temporary files so long as that's not necessary. As a result I implemented what was basically the first version of **enverb** for one of my other packages.

1.1 Grabbing an Environment Verbatim

`\enverb` `\enverb{⟨key=value⟩}[⟨key=value⟩]`
`\enverb⟨*⟩{⟨key=value⟩}`

This function will store the contents of the enclosing environment verbatim inside of `\enverbBody`. It should be the last thing you invoke inside the `\begin` code of your environment and mustn't be enclosed in any nested environment. Both the mandatory and the optional ⟨*key=value*⟩ argument are parsed by `\enverb` – see subsection 1.3 about the specifics of this. The mandatory argument is meant for a package/code developer to assign the sensible default values for the envisioned functionality and always parsed inside the keyset of `\enverb`, while the optional one is meant to be used by the user and as such should be the first thing inside the enclosing environment (if the user wants to use an optional argument).

`\enverb` sets the special characters `\`, `{`, `}`, `$`, `&`, `#`, `^`, `_`, `%`, `~`, `"`, `|` to the category code 12 (other), and makes both spaces and tabulators active tokens. By default each line break will be a category code 12 carriage return (character code 13), but see the `eol` option in subsection 1.3.

A line that contains only whitespace on input will be output as an empty line.

The `\enverb` syntax of the optional argument has a few peculiarities. It is scanned for the opening bracket either on the same line as the `\begin` statement of the enclosing environment, or on any subsequent line until a non-whitespace character is found. The optional argument is read using the "normal" (so surrounding) category code regime, but it is scanned for it in a way that the first token is still read verbatim if no optional argument was given.

The first non-whitespace token that's not an opening bracket must not be found on the same line as the `\begin` statement, and if there was an optional argument there shouldn't be any non-whitespace token trailing it in that line. Additionally there should be no non-whitespace material following the `\end` statement (anything preceding it in the same line is for the most part ignored; `\begin` and `\end` are not balanced, the first `\end` with the correct environment as argument terminates scanning).

The starred version of `\enverb` doesn't search for an optional argument.

1.2 Using Verbatim Environment Contents

`\enverbBody` `\enverbBody`

This macro holds the contents of the last scanned `\enverb`-body, assignments to it by the package are all local.

\enverbExecute `\enverbExecute`

This function executes the current contents of `\enverbBody` as though they would be part of your input at this place (using `\scantokens`, it is tried as good as possible to tackle `\scantokens`'s shortcomings; also it is assumed that you didn't change the `eol` key, or else the newlines aren't carried over but whatever your `eol` value set is used). The entire contents are `\detokenized` once before `\scantokens` does its magic (shouldn't affect the verbatimly read material, but is deemed a sensible safety measure if something unexpected happens; this also assumes that anything you placed inside `bol` or `eol` is rescanned into the same stuff once `\detokenized`).

The function isn't fully expandable, the contents are executed at the current group level.

\enverbListing `\enverbListing {⟨env⟩} {⟨args⟩}`

This function is meant to reuse the current contents of `\enverbBody` inside another verbatim environment. The used mechanism is tested to work with `verbatim`, and environments defined by fancyvrb, listings and minted (provided you use the default `eol` setting). ⟨*env*⟩ is the environment name in which the contents should be nested, and ⟨*args*⟩ is material to be placed after the `\begin` statement (just as you input them, with the outer braces from this argument removed).

Just like `\enverbExecute` this `\detokenizes` the current content of `\enverbBody` as well as your ⟨*args*⟩ and uses `\scantokens`.

1.3 Available Keys

\enverbsetup `\enverbsetup {⟨key=value⟩}`

This macro can be used to (locally) change the settings of **enverb** outside of the `\enverb` macro.

The available keys are:

ignore `ignore = ⟨int⟩` default: *unused*

Set up the number of tokens to ignore at the start of each line. This is only applied after the matching `\end` was found, hence doesn't affect the `\end`-line. A line containing any non-whitespace tokens but in total fewer tokens (counting whitespace) than ⟨*int*⟩ will lead to undefined behaviour.

`ignore` and `auto-ignore` are mutually exclusive.

auto-ignore `auto-ignore` default: `true`
`auto-ignore = ⟨bool⟩`

If this is true (or no value is given) the number of leading tokens is determined by the number of leading tokens found in the line with the matching `\end`. This allows for nice nesting of indentation levels in your LaTeX sources without affecting functionality. A line containing any non-whitespace tokens but in total fewer tokens (counting whitespace) than the leading tokens in the `\end`-line will lead to undefined behaviour.

`ignore` and `auto-ignore` are mutually exclusive.

more-ignore `more-ignore = ⟨int⟩` default: 2

If `auto-ignore` is true this specifies how many tokens in addition to the number of leading tokens shall be ignored at the start of each line. This allows for even the contents of the `\enverb`-environment to be further indented than the `\begin` and `\end` statement.

bol `bol = ⟨tokens⟩` default: *empty*
`bol+ = ⟨tokens⟩`
`+bol = ⟨tokens⟩`

Defines which ⟨*tokens*⟩ are placed at the **b**egin **o**f **l**ines. Your ⟨*tokens*⟩ can be arbitrary contents and are placed as is (so can contain macros which work as you intend when using `\enverbBody`). Keep in mind that if you want to use this with `\enverbExecute` or `\enverbListing` your tokens should be rescanned to the same code after they got `\detokenized`.

The `bol+` variant adds your ⟨*tokens*⟩ after the current contents of `bol`, while `+bol` adds them in front of the contents.

eol `eol = ⟨tokens⟩` default: $\hat{}\hat{}M_{12}$
`eol+ = ⟨tokens⟩`
`+eol = ⟨tokens⟩`

Defines which ⟨*tokens*⟩ are placed at the **e**nd **o**f **l**ines. Your ⟨*tokens*⟩ can be arbitrary contents and are placed as is (so can contain macros which work as you intend when using `\enverbBody`). Keep in mind that if you want to use this with `\enverbExecute` or `\enverbListing` your tokens should be rescanned to the same code after they got `\detokenized`.

The `eol+` variant adds your ⟨*tokens*⟩ after the current contents of `eol`, while `+eol` adds them in front of the contents.

key-handler `key-handler = ⟨code⟩` default: *throw an error*

If used any undefined keys encountered while parsing the mandatory and optional arguments of `\enverb` is parsed via ⟨*code*⟩. Inside your ⟨*code*⟩ use `#1` to specify where to input the list of unknown keys. This can be used to specify additional keys to provide an interface to control your wrapping code. This includes the handling of any unknown keys encountered while parsing the mandatory ⟨*key=value*⟩ argument of `\enverb`.

key-set `key-set = ⟨set⟩`

enverb uses expkv for its key-parsing. This key is a special form of `key-handler` defining a handler that parses the unknown keys as part of ⟨*set*⟩ in expkv (so basically a shorthand for `key-handler = \ekvset{⟨set⟩}{#1}`).

oarg-not-enverb `oarg-not-enverb` default: false
`oarg-not-enverb = ⟨bool⟩`

If this is true (or no value is given) the ⟨*key=value*⟩ list in the optional argument to `\enverb` is not parsed as keys of **enverb** but only inside your defined `key-handler`. Obviously this key doesn't have any effect if only used as part of the optional argument of `\enverb`.

1.4 Examples

The following preamble should work for all of the examples in this section (except for the last one, which can be placed as is in the preamble since it loads all the packages it needs).

Example

```
\documentclass{article}

\usepackage{enverb}
\usepackage{color}
\usepackage{multicol}
\usepackage{listings}
```

1.4.1 A Boxed Verbatim Display

Defining a boxed verbatim macro is pretty simple, we just need to set up the `bol` and `eol` hooks such that line breaks are respected, the rest of the setup already works with LaTeX's default definition for active spaces (and tabs).

Example

```
\newenvironment{myenv}
  {\enverb{bol=\strut, eol=\strut\par}}
  {%
    \par
    \medskip
    \noindent
    \fbox
      {%
        \parbox
          {\dimexpr\linewidth-2\fboxsep-2\fboxrule\relax}
          {%
            \raggedright\frenchspacing\ttfamily
            \enverbBody
          }%
      }%
    \par
    \medskip
  }
```

The probably most complicated part there is the calculation of the `\parbox`'s width, which is just a line width compensated by the space taken up by `\fbox`.

If we now use this environment we get a nice display of our verbatim contents (you might want to play with this and see what happens if your input lines are longer than a `\linewidth`; also note that the spaces after `\begin` and `\end` are optional just like within the normal category code regime, **enverb** will still find the end of its scope).

Example

```
\begin {myenv}
  \This is a%
  \begin{environment}
  that looks \good
\end {myenv}
```

```
\This is a%
\begin{environment}
that looks \good
```

1.4.2 A Page-Breakable Boxed Verbatim Display

The previous example was boxed, lets put that to the next level. We can also achieve page- (and column-)breakable boxed verbatim environments. The simple minded trick here is to draw only line segments at the start and end of each line using the `\vrule` primitive. As a result this will look horribly wrong for lines longer than a single output line, but that would require too much hassle (we just pay attention on our input formatting).

As an added bonus this should have line counters at the start of each line. For that we first define a little helper.

Example

```
\newcount\myline
\newcommand\linecounting
  {%
    \advance\myline1
    \textcolor[gray]{.7}{\footnotesize\the\myline}%
    \quad
  }
```

Next we can define our environment. The `bol` and `eol` hooks look similar to the previous example, but this time they got some added contents for the additional formatting. Also, we borrow some code from LaTeX's starred `verbatim` to make spaces visible (since they are active in `\enverb` this is also pretty straight forward).

Example

```
\newenvironment{myenv}
  {%
    \enverb
      {
         bol=\noindent\strut\vrule\hskip\fboxsep\linecounting
        ,eol=\hfill\hskip\fboxsep\vrule\strut\par
      }%
  }
  {%
    \medskip
    \hrule
    \ttfamily\frenchspacing
```

```
    \UseName{@setupverbvisiblespace}\UseName{@vobeyspaces}%
    \enverbBody
    \hrule
    \medskip
  }
```

The entire result can then be used with ease.

Example

```
\begin{multicols}{2}
  Text above
  \begin{myenv}
    This \is
    {an} $example$
    ^of verbatim_
    input\, being
    displayed boxed.
  \end{myenv}
  Text below
\end{multicols}
```

Text above

```
1  This␣\is
2  {an}␣$example$
3  ^of␣verbatim_
4  input\,␣being
5  displayed␣boxed.
```

Text below

1.4.3 The Real Deal – Using the Body Twice

All of the above is just pretty playthings that are easily possible with **enverb** (and with relative few lines of code) though most likely better done by more full fledged packages like listings or similar. The original reason for all this was to gain control over both the listing and typeset result for code documentation. A simple example could be the following:

Example

```
\newenvironment{myenv}
  {\enverb{}}
  {%
    \par\noindent
    \begin{minipage}{0.48\linewidth}
      \enverbListing{lstlisting}{}
    \end{minipage}%
    \hfill
    \colorbox{yellow}
      {%
        \begin{minipage}{\dimexpr0.48\linewidth-2\fboxsep\relax}
          \enverbExecute
        \end{minipage}%
      }%
    \par
  }
```

Which if used looks like the following (admittedly not that pretty):

Example

```
\begin{myenv}
  This%
  is%
  great.%
\end{myenv}
```

```
This%
is%
great.%
```

Thisisgreat.

1.4.4 enverb's Style

And if you wondered, you guessed it, all examples in this documentation are typeset using an **enverb** derived environment with a few bells and whistles more (for instance collecting material, typesetting the verbatim listing, but executing it together with a later code block).

The following is a complete listing of everything that sets up the looks of this documentation's example blocks. Note that the macros starting with `\enverb@ex@` are not internals of **enverb** but exclusive to the code formatting the examples.

Example

```
\makeatletter
\RequirePackage{enverb}
\global\let\enverb@ex@stored\@empty
\ekvdefinekeys{enverb/ex}
  {
     boolTF        store         = \enverb@ex@ifstore
    ,invboolTF     no-store      = \enverb@ex@ifstore
    ,boolTF        restore       = \enverb@ex@ifrestore
    ,boolTF        no-lst        = \enverb@ex@ifnolst
    ,also nmeta    no-lst        = below
    ,nmeta         undo-no-lst   = {no-lst=false, undo-below}
    ,boolTF        no-tcb        = \enverb@ex@ifnotcb
    ,also nmeta    no-tcb        = below
    ,nmeta         undo-no-tcb   = {no-tcb=false, undo-below}
    ,boolTF        same-line     = \enverb@ex@ifsameline
    ,unknown-choice same-line    =
      \let\enverb@ex@ifsameline\@firstoftwo\def\enverb@ex@codeshare{#1}
    ,initial       same-line     = 0.6
    ,invboolTF     other-line    = \enverb@ex@ifsameline
    ,nmeta         below         = {other-line}
    ,nmeta         undo-below    = {same-line}
    ,store         inter         = \enverb@ex@inter
    ,code          aboveskip     = \def\enverb@ex@above{\vskip#1\relax}
    ,code          belowskip     = \def\enverb@ex@below{\vskip#1\relax}
    ,default       aboveskip     = \medskipamount
    ,default       belowskip     = \medskipamount
    ,initial       aboveskip
    ,initial       belowskip
    ,noval         no-aboveskip  = \let\enverb@ex@above\@empty
    ,noval         no-belowskip  = \let\enverb@ex@below\@empty
    % listings
    ,store         lst           = \enverb@ex@options@lst
    ,initial       lst           = {aboveskip=0pt, belowskip=0pt}
    ,store         same-line-lst = \enverb@ex@options@lst@same
```

```
    ,meta          slst          = same-line-lst={#1}
    ,store         other-line-lst = \enverb@ex@options@lst@other
    ,meta          olst          = other-line-lst={#1}
    % tcolorbox
    ,store         tcb           = \enverb@ex@options@tcb
    ,initial       tcb           = {nobeforeafter}
    ,store         same-line-tcb = \enverb@ex@options@tcb@same
    ,initial       same-line-tcb = {box align=center}
    ,meta          stcb          = same-line-tcb={#1}
    ,store         other-line-tcb = \enverb@ex@options@tcb@other
    ,meta          otcb          = other-line-tcb={#1}
  }
\newenvironment{ex}
  {\enverb{key-set=enverb/ex}}
  {%
    \par
    \enverb@ex@above
    \noindent
    \expanded{\noexpand\begin{exwrap}{\enverb@ex@ifsameline{}{breakable}}}
      \enverb@ex@ifnolst
        {}%
        {%
          \enverb@ex@ifsameline
            {\noindent\begin{minipage}[c]{\enverb@ex@codeshare\linewidth}}%
            {}%
          \ExpandArgs{ne}\enverbListing{lstlisting}%
            {%
              [{%
                \unexpanded\expandafter{\enverb@ex@options@lst},%
                \enverb@ex@ifsameline
                  {\unexpanded\expandafter{\enverb@ex@options@lst@same}}%
                  {\unexpanded\expandafter{\enverb@ex@options@lst@other}}%
              }]%
            }%
          \enverb@ex@ifsameline
            {\end{minipage}}%
            {}%
        }%
      \enverb@ex@ifstore
        {%
          \xdef\enverb@ex@stored
            {%
              \unexpanded\expandafter{\enverb@ex@stored}%
              \unexpanded\expandafter{\enverbBody}%
            }%
        }%
        {}%
      \enverb@ex@ifrestore
        {%
          \edef\enverbBody
            {%
              \unexpanded\expandafter{\enverb@ex@stored}%
              \unexpanded\expandafter{\enverbBody}%
            }%
          \global\let\enverb@ex@stored\@empty
        }%
        {}%
      \enverb@ex@ifnotcb
        {}%
        {%
          \enverb@ex@ifsameline{}{\medskip}%
```

```
          \expanded{\noexpand\begin{exbox}%
              {%
                \unexpanded\expandafter{\enverb@ex@options@tcb},%
                \enverb@ex@ifsameline
                  {\unexpanded\expandafter{\enverb@ex@options@tcb@same}}%
                  {\unexpanded\expandafter{\enverb@ex@options@tcb@other}}%
              }}%
              {%
                \linewidth
                \enverb@ex@ifsameline{-\enverb@ex@codeshare\linewidth}{}%
              }%
            \enverbExecute
          \end{exbox}%
        }%
    \end{exwrap}%
    \par
    \enverb@ex@below
    \@endpetrue
  }

\RequirePackage{listings}% >>=
\newcommand\extexcsstyle{\exbasestyle\color{green!50!black}\bfseries}
\newcommand\exbasestyle{\ttfamily}
\lstset
  {
     language=[LaTeX]TeX
    ,flexiblecolumns=true
    ,basicstyle=\exbasestyle
    ,texcsstyle=*\extexcsstyle
    ,moretexcs=%
      {%
        UseName,ExpandArgs,textcolor,color,colorbox,expanded,unexpanded,@empty,%
        @firstoftwo,@endpetrue,RequirePackage,lstset,usetikzlibrary,%
        pgfmathsetseed,newtcolorbox,tcbsubskin,tikzset,dimexpr%
      }
    ,moretexcs=[2]{enverb,enverbBody,enverbExecute,enverbListing}
    ,texcsstyle=*[2]\exbasestyle\color{enverbred}\bfseries
    ,literate={\{}{{\extexcsstyle\string{\iffalse}\fi}}{1}
              {\}}{{\extexcsstyle\iffalse{\fi\string}}}{1}
  }% =<<
\RequirePackage[most]{tcolorbox}%>>=
\RequirePackage{tikz}
\usetikzlibrary{decorations.pathmorphing,calc}
\pgfmathsetseed{1} % To have predictable results
% Define paper style
\tikzset{
  % based on https://tex.stackexchange.com/questions/580671
  ragged border/.style={%
    decoration={random steps, segment length=2mm, amplitude=0.5mm},
    decorate,
  }
}
\tcbsubskin{exwrap}{standard}
  {%
     width=\linewidth
    ,colframe=enverbgrey
    ,colback=enverbred!15!white
    ,title=Example
    ,nobeforeafter
    ,arc=5mm
    ,sharp corners=northwest
```

```
    ,toprule at break=-1sp
    ,bottomrule at break=-1sp
  }
\tcbsubskin{exwrap-first}{exwrap}{sharp corners=south}
\tcbsubskin{exwrap-mid}{exwrap}{sharp corners=all}
\tcbsubskin{exwrap-last}{exwrap}{sharp corners=north}
\newtcolorbox{exbox}[2]
  {%
     colback=white%
    ,colframe=white%
    ,geometry nodes=true
    ,interior code=
      {%
        \fill[white]
          decorate[ragged border]{
            ($(frame.south east) - (0, 1mm)$) --
            ($(frame.south west) - (0, 1mm)$)
          }
          -- (frame.south west) -- ++(0, 1pt) -| (frame.south east) -- cycle;
        \fill[white]
          decorate[ragged border]{
            ($(frame.north east) + (0, 1mm)$) --
            ($(frame.north west) + (0, 1mm)$)
          }
          -- (frame.north west) -- ++(0, -1pt) -| (frame.north east) -- cycle;
      }
    ,fontupper=\small
    ,width={\dimexpr#2\relax}%
    ,sharp corners=all
    ,boxsep=0pt
    ,top=\fboxsep
    ,bottom=\fboxsep
    ,left=\fboxsep
    ,right=\fboxsep
    ,#1%
  }
\newtcolorbox{exwrap}[1]
  {%
     skin=exwrap
    ,skin first=exwrap-first
    ,skin middle=exwrap-mid
    ,skin last=exwrap-last
    ,#1%
  }
%=<<
\makeatother
```

And if we use that we can do all sorts of interesting things:

Example

```
We define a helper:
\begin{ex}[store,no-tcb]
  \newcommand\foo{bar}
\end{ex}
And then can use it:
\begin{ex}[restore]
  This is \foo
\end{ex}
A wider example:
\begin{ex}[below]
  This is a wider example.
\end{ex}
An example with wide code
but small output:
\begin{ex}[same-line=.9]
  \textcolor{red}{\bfseries.}
\end{ex}
```

We define a helper:

Example

```
\newcommand\foo{bar}
```

And then can use it:

Example

```
This is \foo
```

This is bar

A wider example:

Example

```
This is a wider example.
```

This is a wider example.

An example with wide code but small output:

Example

```
\textcolor{red}{\bfseries.}
```

.

2 Implementation

2.1 Who we are

\enverb@date
\enverb@version

We store the package version and date inside macros.

```
\newcommand*\enverb@date{2025-01-10}
\newcommand*\enverb@version{1.1}
```

(*End of definition for* `\enverb@date` *and* `\enverb@version`.)

And the package identification:

```
\ProvidesPackage{enverb}
  [\enverb@date\space v\enverb@version\space read an environment verbatim]
```

2.2 Some required LaTeX 2ε style argument grabbers

\@thirdofthree
\@firstofnine

This here are just two argument grabbers that aren't defined in all versions of LaTeX 2ε.

```
\providecommand\@thirdofthree[3]{#3}
\providecommand\@firstofnine[9]{#1}
```

(*End of definition for* `\@thirdofthree` *and* `\@firstofnine`.)

2.3 Borrow some code

\enverb@count
\enverb@ifxTF
\enverb@chargen
\enverb@othercr

We'll make use of the following expl3 functions, but since the remainder of this package is coded in LaTeX 2ε style we stick to that.

```
\ExplSyntaxOn
\cs_new_eq:NN \enverb@count \tl_count:n
\cs_new_eq:NN \enverb@ifxTF \token_if_eq_meaning:NNTF
\cs_new_eq:NN \enverb@chargen \char_generate:nn
\cs_set:Npx \enverb@othercr { \char_generate:nn {13} {12} }
\ExplSyntaxOff
```

(*End of definition for* `\enverb@count` *and others.*)

2.4 Key-value Setup

First we need to load a package to allow us defining our keys, then we define a bunch of keys:

```
\RequirePackage{expkv-def}
\ekvdefinekeys{enverb}
  {
     protect code  ignore      = \let\enverb@ifautoignore\@secondoftwo
    ,also    eint  ignore      = \enverb@ignore
    ,boolTF        auto-ignore = \enverb@ifautoignore
    ,initial       auto-ignore
    ,eint          more-ignore = \enverb@moreignore
    ,initial       more-ignore = 2
    ,long store    bol         = \enverb@bol@content
    ,long code     bol+        = \enverb@add\enverb@bol@content{#1}
    ,long code     +bol        = \enverb@pre\enverb@bol@content{#1}
    ,long store    eol         = \enverb@eol@content
    ,einitial      eol         = \enverb@othercr
```

```
    ,long code     eol+        = \enverb@add\enverb@eol@content{#1}
    ,long code     +eol        = \enverb@pre\enverb@eol@content{#1}
    ,protect code  key-handler = \protected\long\def\enverb@keyhandler##1{#1}
    ,protect code  key-set     = \protected\long\ekvsetdef\enverb@keyhandler{#1}
    ,unknown code              = \enverb@add\enverb@unknown@kv{, {#3} = {#2} }
    ,unknown noval             = \enverb@add\enverb@unknown@kv{, {#2} }
    ,boolTF        oarg-not-enverb = \enverb@ifoargtokeyhandler
  }
```

\enverbsetup We need a user facing macro to set our keys.

```
\protected\long\ekvsetdef\enverbsetup{enverb}
```

(*End of definition for* `\enverbsetup`. *This function is documented on page* 3.)

\enverb@unknown@kv This is a storage for all unknown keys encountered.

```
\let\enverb@unknown@kv\@empty
```

(*End of definition for* `\enverb@unknown@kv`.)

2.5 Miscellaneous Auxiliary Functions

\enverb@stop A very simple minded stop-macro, just gobble up to the eponymous mark.

```
\long\def\enverb@stop#1\enverb@stop{}
```

(*End of definition for* `\enverb@stop`.)

\enverb@error

```
\protected\def\enverb@error{\PackageError{enverb}}
```

(*End of definition for* `\enverb@error`.)

\enverb@misused This is an error message that'd be used inside a `\lowercase` environment if put where it originally should've been. Hence we define it now to get correct formatting.

```
\protected\def\enverb@misused
  {%
    \enverb@error
      {Misused \string\enverb. Input already tokenised}%
      {%
        It seems you used the \string\enverb\space based environment
        {\@currenvir} inside another\MessageBreak
        macro's or a primitive's argument; that doesn't work.%
      }%
  }
```

(*End of definition for* `\enverb@misused`.)

\enverb@keyhandler
\enverb@keyhandler@ This is the initial definition of the handler for unknown keys, namely throw an error containing the list of unknown keys.

```
\protected\long\def\enverb@keyhandler#1%
  {%
    \if\relax\detokenize{#1}\relax
      \expandafter\@gobbletwo
    \fi
    \@firstofone
```

```
      {%
        \enverb@error
          {%
            Unknown keys encountered:
            \expanded
              {%
                \ekvparse
                  {\MessageBreak\@spaces\unexpanded}%
                  {\MessageBreak\@spaces\enverb@keyhandler@}%
                  {#1}%
              }%
            \@gobble
          }%
          {%
            Perhaps you misspelled some key names? Or you forgot to set up
            custom key\MessageBreak
            parsing.%
          }%
      }%
  }
\long\def\enverb@keyhandler@#1{\unexpanded{#1} = \unexpanded}
```

(*End of definition for* `\enverb@keyhandler` *and* `\enverb@keyhandler@`.)

2.6 Collector

Much of the code in this package works under a weird category code regime, which the following sets up. After this ~ will be an active newline character, : will be an active space, and ; will be an active tab.

```
\begingroup
\lccode`\~=`\^^M
\catcode`\:=13
\lccode`\:=`\ % <- space
\catcode`\;=13
\lccode`\;=`\^^I % <- tab
\lowercase{\endgroup
```

`\enverb@body@space` `\enverb@body@tab` `\enverb@body@newline` `\enverb@collect`

To quickly check whether a line contains only spaces or tabs we define them as empty, that way we can remove them simply by one step of `\romannumeral` expansion. The newlines on the other hand grab the entire next line, check whether the matching `\end` is encountered, and if not leave the contents in `\unexpanded`. This grabbing is not done by the newline characters directly, but instead via `\enverb@collect` so that low level errors are easier to understand.

```
\def\enverb@body@space{}
\def\enverb@body@tab{}
\def\enverb@body@newline{\enverb@collect}
\def\enverb@collect#1~%
  {\enverb@ifnotend{#1}{\enverb@bol\unexpanded{#1}\enverb@eol\enverb@collect}}
```

(*End of definition for* `\enverb@body@space` *and others.*)

`\enverbBody` `\enverb@body@setup`

This function sets up the entire category code regime and initial assignments of special tokens.

```
\protected\def\enverb@body@setup
  {%
    \let\enverbBody\@empty
    \let\do\@makeother\dospecials
    \endlinechar=`\^^M
    \catcode`\^^M=13 \let~\enverb@body@newline
    \catcode`\   =13 \let:\enverb@body@space
    \catcode`\^^I=13 \let;\enverb@body@tab
    \let\enverb@bol\relax
    \let\enverb@eol\relax
  }
```

(*End of definition for* `\enverbBody` *and* `\enverb@body@setup`*. These functions are documented on page* 2.)

`\enverb@ifxanyTF` `\enverb@ifxanyTF@` `\enverb@ifxanyTF@ifdone` `\enverb@ifxanyTF@F` `\enverb@ifxanyTF@T`

This function provides a loop to check a list of symbols whether any of them matches the meaning of the first argument.

```
\long\def\enverb@ifxanyTF#1#2%
  {\enverb@ifxanyTF@#1#2\enverb@ifxanyTF@}
\long\def\enverb@ifxanyTF@#1#2%
  {%
    \enverb@ifxanyTF@ifdone#2\enverb@ifxanyTF@F\enverb@ifxanyTF@
    \enverb@ifxTF#1#2\enverb@ifxanyTF@T{\enverb@ifxanyTF@#1}%
  }
\long\def\enverb@ifxanyTF@ifdone#1\enverb@ifxanyTF@{}
\long\def\enverb@ifxanyTF@F
    \enverb@ifxanyTF@\enverb@ifxTF#1\enverb@ifxanyTF@T#2#3#4%
  {#4}
\long\def\enverb@ifxanyTF@T#1\enverb@ifxanyTF@#2#3{#2}
```

(*End of definition for* `\enverb@ifxanyTF` *and others.*)

`\enverb@search@oarg@a` `\enverb@search@oarg@ifx`

This function shall search for the optional argument. The first case is rather easy, namely we check for a space (remember that : is an active space character in the current context). If this isn't the first line (so the line containing the `\begin` statement) that space is added to the body and searching continues.

Next repeat the same logic but for an active tab (;). Afterwards we check for an active newline (~) which if found sets the conditional for the first line to false, still applying the same logic otherwise.

Lastly we check for the open bracket. If it's found we collect our optional argument, otherwise the body starts, which is either an illegal character immediately following the `\begin` line, or we normally collect the body.

```
\def\enverb@search@oarg@ifx#1#2%
  {%
    \enverb@ifxTF#1\@let@token
      {%
        \ifenverb@firsteol#2\else\enverb@body@add{#1}\fi
        \enverb@search@oarg@b
      }%
  }
\protected\def\enverb@search@oarg@a
  {%
    \enverb@search@oarg@ifx:{}%
      {%
```

```
        \enverb@search@oarg@ifx;{}%
          {%
            \enverb@search@oarg@ifx~\enverb@firsteolfalse
              {%
                \enverb@ifxTF{[}\@let@token
                  {\enverb@oarg}%
                  {%
                    \enverb@ifxanyTF\@let@token
                      {%
                        \@sptoken\par\bgroup\egroup$&##^_% $
                      }
                      {%
                        \enverb@misused
                        \endgroup
                      }%
                      {%
                        \ifenverb@firsteol
                          \expandafter\enverb@body@after@begin
                        \else
                          \expandafter\enverb@body
                        \fi
                      }%
                  }%
              }%
          }%
      }%
  }
```

(*End of definition for* `\enverb@search@oarg@a` *and* `\enverb@search@oarg@ifx`.)

`\enverb@body`
`\enverb@body@after@oarg`
`\enverb@body@after@begin`

The body collection works by collecting everything inside a macro in one sweep, and a second pass over the collected material to strip leading spaces if the `ignore` or `auto-ignore` feature is used. This macro starts the first sweep. Since during the collection of the optional argument we might already have collected a few tokens we place those at the start by expanding the current definition of `\enverbBody` once.

```
\protected\def\enverb@body
  {\edef\enverbBody{\iffalse}\fi\expandafter\enverb@collect\enverbBody}
```

The other two macros also start collecting the body, but are only called after either an optional argument was found, or if the first non-ignored character of the optional argument search was encountered before the first newline. In both cases we check whether the remainder of that line can be considered blank, and if so collect the body.

```
\def\enverb@body@after@oarg#1~%
  {\enverb@ensure@blank{#1}{closing bracket}\enverb@body}
\def\enverb@body@after@begin#1~%
  {\enverb@ensure@blank{#1}{\string\begin}\enverb@body}
```

This is also the last place we need the strange category setup in, so we close the delimiting brace of the `\lowercase` above here.

```
}
```

(*End of definition for* `\enverb@body`, `\enverb@body@after@oarg`, *and* `\enverb@body@after@begin`.)

\enverb@ensure@blank

We actually need to define the check for an empty line. This check here works for our specifically set up regime of spaces and tabs (which both expand to nothing, the same trick is basically used twice, though we once fully expand and once use romannumeral-expansion).

```
\newcommand\enverb@ensure@blank[2]
  {%
    \expandafter\enverb@ifempty\expanded{{#1}}{}%
      {%
        \expanded
          {%
            \noexpand\enverb@error
              {%
                Line after #2 not empty.\noexpand\MessageBreak
                Contains: \detokenize\expandafter{\romannumeral`\^^@#1}%
              }%
              {%
                \noexpand\enverb will try to ignore this. You should clean up
                your input.%
              }%
          }%
      }%
  }
```

(*End of definition for* `\enverb@ensure@blank`.)

\enverb@ifempty
\enverb@ifempty@
\enverb@ifempty@true

This is just a quick argument-grabbing based test for an empty argument.

```
\newcommand\enverb@ifempty[1]
  {%
    \enverb@ifempty@\enverb@ifempty@A#1\enverb@ifempty@B.\enverb@ifempty@true
      \enverb@ifempty@A\enverb@ifempty@B
  }
\def\enverb@ifempty@#1\enverb@ifempty@A\enverb@ifempty@B#2#3{#3}
\def\enverb@ifempty@true\enverb@ifempty@A\enverb@ifempty@B#1#2{#1}
```

(*End of definition for* `\enverb@ifempty`, `\enverb@ifempty@`, *and* `\enverb@ifempty@true`.)

\enverb@add
\enverb@pre
\enverb@body@add

These are just two functions to add things to a token list storing macro, one is available for arbitrary contents, the other is used for the body.

```
\protected\long\def\enverb@add#1#2{\edef#1{\unexpanded\expandafter{#1#2}}}
\protected\long\def\enverb@pre#1#2%
  {\edef#1{\unexpanded{#2}\unexpanded\expandafter{#1}}}
\protected\def\enverb@body@add{\enverb@add\enverbBody}
```

(*End of definition for* `\enverb@add`, `\enverb@pre`, *and* `\enverb@body@add`.)

\enverb

This macro is the front facing command that starts the entire grabbing logic. The first thing we need to do is define the auxiliary we use to check whether we're done collecting the body, which is dependent on the current environment's name. Next we store the mandatory argument in `marg` and initialise the `oarg`-storage, category regime, and `firsteol`-boolean. Finally we start searching for the optional argument.

```
\NewDocumentCommand \enverb { s +m }
  {%
    \expandafter\enverb@ifnotend@setup@perhaps\expanded
      {{\string{\@currenvir\string}}}%
```

```
    \let\enverb@collected@oarg\@empty
    \edef\enverb@collected@marg{\unexpanded{#2}}%
    \begingroup
    \enverb@body@setup
    \IfBooleanTF{#1}%
      {\enverb@body@after@begin}%
      {%
        \enverb@firsteoltrue
        \enverb@search@oarg
      }%
  }
```

(*End of definition for* `\enverb`. *This function is documented on page* 2.)

`\ifenverb@firsteol` `\enverb@firsteoltrue` `\enverb@firsteolfalse`

A simple TEX-style boolean to keep track whether we're in front of the first newline.

```
\newif\ifenverb@firsteol
```

(*End of definition for* `\ifenverb@firsteol`, `\enverb@firsteoltrue`, *and* `\enverb@firsteolfalse`.)

`\enverb@search@oarg` `\enverb@search@oarg@b`

The core function to search for an optional argument was already defined above, these are the first and third step of that functionality (starting and resuming to search).

```
\protected\def\enverb@search@oarg{\futurelet\@let@token\enverb@search@oarg@a}
\protected\def\enverb@search@oarg@b{\expandafter\enverb@search@oarg\@gobble}
```

(*End of definition for* `\enverb@search@oarg` *and* `\enverb@search@oarg@b`.)

`\enverb@oarg` `\enverb@oarg@`

This is used once the opening brace of the optional argument is found. First we close the group in which the body's category code regime is active, then we actually collect the optional argument with the normal category codes, restore the body's regime and collect it.

```
\protected\def\enverb@oarg{\endgroup\enverb@oarg@}
\NewDocumentCommand\enverb@oarg@{+O{}}
  {%
    \enverb@add\enverb@collected@oarg{#1}%
    \begingroup
    \enverb@body@setup
    \enverb@body@after@oarg
  }
```

(*End of definition for* `\enverb@oarg` *and* `\enverb@oarg@`.)

`\enverb@ifnotend`

These functions check whether the input contains the string representation `\end`. If it does they invoke `\enverb@ifnotend@maybe` with the tokens in front of `\end` and after it as two arguments. Otherwise the third argument after `\enverb@ifnotend@`'s expansion is directly used, which is the second (curried) argument to `\enverb@ifnotend`.

```
\newcommand\enverb@ifnotend[1]%
  {%
    \def\enverb@ifnotend##1%
      {%
        \enverb@ifnotend@
          ##1\enverb@mark\enverb@ifnotend@maybe
          #1\enverb@mark\@thirdofthree
          \enverb@stop
      }%
```

```
    \def\enverb@ifnotend@##1#1##2\enverb@mark##3##4\enverb@stop{##3{##1}{##2}}%
  }
\expandafter\enverb@ifnotend\expanded{{\expandafter\@gobble\string\\end}}
```

(*End of definition for* `\enverb@ifnotend`.)

`\enverb@ifnotend@maybe` This function will be used if an `\end` was encountered and should check whether that is indeed the end of our environment, it uses a second stage `\enverb@ifnotend@perhaps` for that check. The `\romannumeral` expansion it employs will strip any leading spaces or tabs from #2.

```
\newcommand\enverb@ifnotend@maybe[2]
  {\expandafter\enverb@ifnotend@perhaps\expandafter{\romannumeral`\^^@#2}{#1}}
```

(*End of definition for* `\enverb@ifnotend@maybe`.)

`\enverb@ifnotend@setup@perhaps` `\enverb@ifnotend@perhaps` `\enverb@ifnotend@perhaps@` `\@thirdofthree` We need an expandable and fast check to see whether some input really contains the `\end` of our environment. Checking for `\end` itself is already done above, we need to check for the correct argument to it. The following does so by argument grabbing logic.

```
\protected\def\enverb@ifnotend@setup@perhaps#1%
  {%
    \def\enverb@ifnotend@perhaps##1%
      {%
        \enverb@ifnotend@perhaps@\enverb@mark##1\enverb@mark\enverb@ifnotend@end
          \enverb@mark#1\enverb@mark\@thirdofthree
          \enverb@stop
      }%
    \def\enverb@ifnotend@perhaps@
        ##1\enverb@mark#1##2\enverb@mark##3##4\enverb@stop
      {##3{##2}}%
  }
```

(*End of definition for* `\enverb@ifnotend@setup@perhaps` *and others.*)

`\enverb@ifnotend@end` If the correct `\end` statement was found this function will be called by `\enverb@ifnotend@perhaps`. It's first argument will be anything in that line following the `\end` statement, the second argument will be the material in front of it. And #3 will be whatever the argument to `\enverb@ifnotend` was.

The first thing we need to do is end the body collection by placing the matching closing brace. After that we throw an error on lost content after the `\end` if it's non-blank. And finally we need to smuggle the collected body outside the current group's scope.

```
\def\enverb@ifnotend@end#1#2#3%
  {%
    \iffalse{\fi}%
    \enverb@ensure@blank{#1}{\string\end}%
    \expandafter\enverb@finalise\expandafter{\enverbBody}{#2}%
  }
```

(*End of definition for* `\enverb@ifnotend@end`.)

`\enverb@finalise` The finalising function closes the group containing the body's catcode regime. The already tokenised body is #1, and #2 is the contents found in front of the `\end` statement, which we might use for the auto-ignore feature.

It then does the key=val parsing, sets up the aftertreatment and assigns the final meaning to \enverbBody. Eventually it calls the end code of the enclosing environment.

```
\protected\def\enverb@finalise#1#2%
  {%
    \endgroup
    \expandafter\enverbsetup\expandafter{\enverb@collected@marg}%
    \enverb@ifoargtokeyhandler
      {%
        \expandafter\enverb@keyhandler\expanded
          {{%
            \unexpanded\expandafter{\enverb@unknown@kv},%
            \unexpanded\expandafter{\enverb@collected@oarg}%
          }}%
      }%
      {%
        \expandafter\enverbsetup\expandafter{\enverb@collected@oarg}%
        \expandafter\enverb@keyhandler\expandafter{\enverb@unknown@kv}%
      }%
    \enverb@ifautoignore
      {\enverb@setup@ignore{\enverb@count{#2}+\enverb@moreignore}}%
      {\enverb@setup@ignore\enverb@ignore}%
    \edef\enverb@line##1\enverb@eol
      {%
        \noexpand\unexpanded{##1}%
        %\noexpand\detokenize{##1}% % TODO
        \noexpand\unexpanded{\unexpanded\expandafter{\enverb@eol@content}}%
      }%
    \edef\enverbBody{#1}%
    \expandafter\end\expandafter{\@currenvir}%
  }
```

(*End of definition for* `\enverb@finalise`.)

`\enverb@setup@ignore`
`\enverb@setup@ignore@`

```
\protected\def\enverb@setup@ignore#1%
  {\expandafter\enverb@setup@ignore@\the\numexpr#1\relax;\enverb@line}
\protected\def\enverb@setup@ignore@#1;#2%
  {%
    \ifnum#1>9
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi
    {\expandafter\enverb@setup@ignore@\the\numexpr#1-8;{\@firstofnine{#2}}}%
```

Just to make sure that \renewcommand works correctly, we ensure the macro exists.

```
    {%
      \let\enverb@bol@gobble\@empty
      \expanded
        {%
          \unexpanded{\renewcommand\enverb@bol@gobble}\ifnum#1>\z@[#1]\fi
            {\unexpanded{#2}}%
          \unexpanded{\def\enverb@bol##1\enverb@eol}%
            {%
```

```
                \noexpand\unexpanded
                  {\unexpanded\expandafter{\enverb@bol@content}}%
                \unexpanded{\expandafter\enverb@ifempty\expanded}{{##1}}%
                  {\noexpand\enverb@line}%
                  {\noexpand\enverb@bol@gobble}%
                ##1\noexpand\enverb@eol
              }%
          }%
      }%
  }
```

(*End of definition for* `\enverb@setup@ignore` *and* `\enverb@setup@ignore@`.)

2.7 Output Oriented Macros

`\enverb@rmeol` `\enverb@rmeol@`

Many of our output routines require the `\scantokens` primitive. Since that one places an additional end of line at the end of its argument we need a way to remove it (this usually results in an additional space in the output otherwise). For that we place a comment character at the end of the input (if it's possible), if that's not available we hope that a stringified `\relax` will do. The code here will use expl3 for ease of coding.

```
\def\enverb@rmeol
  {%
    \ifnum\catcode`\%=14
      \enverb@chargen{`\%}{12}%
      \expandafter\enverb@stop
    \fi
    \enverb@rmeol@0;%
    \enverb@stop
  }
\def\enverb@rmeol@#1;%
  {%
    \ifnum#1=128
      \string\relax
      \expandafter\enverb@stop
    \fi
    \@firstofone
      {%
        \ifnum\catcode#1=14
          \enverb@chargen{#1}{12}%
          \expandafter\enverb@stop
        \fi
        \expandafter\enverb@rmeol@\the\numexpr#1+1;%
      }%
  }
```

(*End of definition for* `\enverb@rmeol` *and* `\enverb@rmeol@`.)

`\enverbExecute`

```
\NewDocumentCommand\enverbExecute{}
  {%
    \begingroup
      \newlinechar=`\^^M
      \expandafter
    \endgroup
```

```
    \scantokens\expanded
      {{%
        \detokenize\expandafter{\enverbBody}%
        \enverb@rmeol
      }}%
  }
```

(*End of definition for* `\enverbExecute`. *This function is documented on page 3.*)

`\enverbListing`

```
\NewDocumentCommand \enverbListing { m m }
  {%
    \scantokens\expanded
      {{%
        \string\begin{#1}\detokenize{#2}\enverb@othercr
          \detokenize\expandafter{\enverbBody}%
        \string\end{#1}\enverb@othercr
        \enverb@rmeol
      }}%
  }
```

(*End of definition for* `\enverbListing`. *This function is documented on page 3.*)