newvbtm and varvbtm
Packages for Variants of `verbatim` Environment*

Hiroshi Nakashima
(Toyohashi Univ. of Tech.)

1999/08/11

Abstract

This file provides two style files; newvbtm to define `verbatim`-like environments; varvbtm to provide set of macros for variants of `verbatim`, e.g. in which `^I` acts as a tab.

Contents

*This file has version number v1.0, last revised 1999/08/11.

1 Introduction

LaTeX users often have trouble when they wish to have their own customized `verbatim`-like environment. Probably you once wished to have an indented-footnotesize-`verbatim` instead of always typing;

```
\begin{itemize}\item[]\footnotesize
\begin{verbatim}
...
\end{verbatim}
\end{itemize}
```

and tried the following just to know it does not work.

```
\newenvironment{myverbatim}{\begin{itemize}\item[]\footnotesize
                             \begin{verbatim}}%
                            {\end{verbatim}\end{itemize}}
```

Another trouble you probablly have had is that what you see in `verbatim` text with `<TAB>` is not what you get because `<TAB>` does not acts as an tab but a space.

Of course it is possible to define your own `verbatim`-like environments if you have enough knowledge of the implementation of `verbatim` including dirty tricks with `\catcode`. However, even a TeXpert should be bored with typing a dirty code like;

```
\begingroup \catcode`\|=0 \catcode`\[=1 \catcode`\]=2
\catcode`\{=12 \catcode`\}=12 \catcode`\\=12
|long|def|@myxverbatim##1\end{myverbatim}[##1|end[myverbatim]]
|endgroup
```

`newvbtm` `varvbtm` The style files distributed with this document will solve these problems. You will have two style files, newvbtm.sty and varvbtm.sty, by processing newvbtm.dtx with docstrip, or simply doing the following.

```
% tex newvbtm.ins
```

The former style provides you `\(re)newverbatim` command to (re)define your own `verbatim`-like environment easily. The latter gives you a set of various macros for tab-emulation, page break control, etc.

2 Usage

2.1 Loading Style Files

Both style files are usable to both LaTeX 2_ε and LaTeX-2.09 users with their standard package loading declaration. If you use LaTeX 2_ε and wish to load, for example, newvbtm, simply do the following.

```
\usepackage{newvbtm}
```

If you still love LaTeX-2.09, the following is what you have to do.

```
\documentstyle[..,newvbtm,...]{⟨main-style⟩}
```

Note that loading varvbtm automatically loads newvbtm too. Thus you may not load both though doing so is safe.

2.2 newvbtm: Define verbatim-like Environments

\newverbatim The command;

\newverbatim{⟨*env*⟩}[⟨*n-args*⟩]{⟨*beg-def-outer*⟩}{⟨*beg-def-inner*⟩}%
{⟨*end-def-inner*⟩}{⟨*end-def-outer*⟩}

defines an environment named ⟨*env*⟩ with ⟨*n-args*⟩ arguments (optionally), and acting conceptually as follows:

⟨*beg-def-outer*⟩\begin{verbatim}⟨*beg-def-inner*⟩
⟨*body-of-environment*⟩
⟨*end-def-inner*⟩\end{verbatim}⟨*end-def-outer*⟩

Thus to have indented-footnotesize-verbatim named, say indfnsverbatim, you may simply do the following.

```
\newverbatim{indfnsverbatim}{\begin{itemize}\item[]\footnotesize}{}{}%
                                {\end{itemize}}
```

Since \newverbatim defines not only ⟨*env*⟩ but also its starred counterpart ⟨*env*⟩* that acts like verbatim*, the definition above also defines indfnsverbatim* environment.

If you use LaTeX 2ε, you may make ⟨*env*⟩ have an optional argument whose default value is ⟨*default*⟩ by;

\newverbatim{⟨*env*⟩}[⟨*n-args*⟩][⟨*default*⟩]{⟨*beg-def-outer*⟩}{⟨*beg-def-inner*⟩}%
{⟨*end-def-inner*⟩}{⟨*end-def-outer*⟩}

For example, our indfnsverbatim environment can have an optional argument to specify a font size other than \footnotesize by the following definition.

```
\newverbatim{indfnsverbatim}[1][\footnotesize]%
        {\begin{itemize}\item[]#1}{}{}{\end{itemize}}
```

The argument ⟨*beg-def-inner*⟩ is for TeXperts who wish to do something overriding what LaTeX's \verbatim does. Even if you don't have much confidence in your TeXpertise, however, you can do some useful thing with this argument. For example, the following is obtained by itself.

```
\newverbatim{slverbatim}{}{\slshape}{}{}
```

Also you will find a few commands for this argument in §2.3.

The needs of ⟨*end-def-inner*⟩ is much more limited. One example is to check if \end{verbatim} is at the beginning of a line. This examination is done by;

```
\newverbatim{myverbatim}{...}{...}%
        {\ifvmode ⟨at-bol⟩ \else ⟨not-at-bol⟩ \fi}{...}
```

\renewverbatim You may redefine your own verbatim-like environment, or even verbatim itself, by \renewverbatim whose arguments are same as those of \newenvironment.

2.3 varvbtm: To Make Variants of `verbatim`

2.3.1 Tab Emulation

`\newtabverbatim` `\renewtabverbatim`

The commands `\(re)newtabverbatim` is to (re)define a `verbatim`-like environment in which `<TAB>` acts as a tab. The syntax of the command is same as that of `\(re)newverbatim`, and its operation is equivalent to;

```
\(re)newverbatim{⟨env⟩}[⟨n-args⟩][⟨default⟩]
        {⟨beg-def-outer⟩}%
        {⟨beg-def-inner⟩⟨beg-def-for-tab⟩}%
        {⟨end-def-for-tab⟩⟨end-def-inner⟩}%
        {⟨end-def-outer⟩}
```

For example;

```
\newtabverbatim{tabverbatim}{}{}{}{}
```

defines `tabverbatim` environment just to make `<TAB>` act as a tab. Another example to have tab emulation version of `indfnsverbatim` with optional argument, say `indfnstabverbatim` is;

```
\newtabverbatim{indfnstabverbatim}[1][\footnotesize]%
        {\begin{itemize}\item[]#1}{}{}{\end{itemize}}
```

Note that in the starred version, e.g. `tabverbatim*`, a `<TAB>` is translated into a sequence of ␣.

`VVBtabwidth`

The distance between tab stops is the width of eight characters of the font used in the environment, i.e. typewriter font usually. If you want to change this default value, set the counter `VVBtabwidth` to the number of characters of the distance.

`\VVBbegintab` `\VVBendtab`

The magical stuff for ⟨*beg-def-for-tab*⟩ and ⟨*end-def-for-tab*⟩ is also accessible through commands `\VVBbegintab` and `\VVBendtab` for TEXperts who wish to do something with `\(re)newverbatim` rather than `\(re)newtabverbatim`.

2.3.2 Form Feed Character

`\VVBprintFF` `\VVBprintFFas`

You might have found that `<FF>` (or `^L`) in `verbatim` caused a mysterious error;

```
! Forbidden control sequence found while scanning use of \@xverbatim.
```

This is because `<FF>` is not *verbatimized*. Giving the command `\VVBprintFF` to ⟨*beg-def-outer*⟩ (or ⟨*beg-def-inner*⟩) of `\newverbatim` does it for you and makes `<FF>` printed as `^L` in default. You may change this default print image by;

```
\VVBprintFFas{⟨str⟩}
```

where ⟨*str*⟩ is a sequence of any printable characters other than `{` and `}`. Note that this command is very *fragile* as `\verb` and `\index`, and thus should not be used in an argument of other commands including `\(re)newverbatim`.

`\VVBbreakatFF` `\VVBbreakatFFonly`

The other way to make `<FF>` acceptable is to give it a useful and natural job, i.e. page breaking. This is done by giving `\VVBbreakatFF` to ⟨*beg-def-inner*⟩ (not *outer*). Its more powerful relative, `\VVBbreakatFFonly`, is also available to allow page breaking at `<FF>` only. Unfortunately, these two commands are incompatible with `\(re)newtabverbatim` and thus you have to use `\(re)newverbatim` with `\VVBbegintab` *followed by* them.

2.3.3 Non-Verbatim Stuff in `verbatim`-like Environment

`\VVBnonverb` You might have once wished to insert a few non-verbatim stuff, for example math stuff. The command, to be given to ⟨*beg-def-outer*⟩;

```
\VVBnonverb{\⟨char⟩}
```

makes it possible. For example, the author just did the following to produce the result shown above.

```
\newverbatim{verbatimwithnv}{\VVBnonverb{\!}}{}{}{}
\begin{verbatimwithnv}
\VVBnonverb{\!$\langle\mbox{\textit{char}}\rangle$!}
\end{verbatimwithnv}
```

As shown in the example above, the non-verbatim staff is surrounded by a pair of ⟨*char*⟩, the letter '`!`' in this case. Note that ⟨*char*⟩ has to be preceded by '`\`' when it is given as the argument of `\VVBnonbverb`, and ⟨*char*⟩ should not be '`\`'. Also note that the default font for the non-verbatim part is not that for verbatim part, but the font used outside the environment[1].

`\VVBnonverbmath` As mentioned above, math stuffs will be most desirable to be non-verbatim. Thus the macro;

```
\VVBnonverbmath[\⟨char⟩]
```

gives you a shorthand to typeset the stuff surrounded by a pair of ⟨*char*⟩ in math mode. Since the default of ⟨*char*⟩ is `$` as expected, the example above may be;

```
\newverbatim{verbatimwithnv}{\VVBnonverbmath}{}{}{}
\begin{verbatimwithnv}
\VVBnonverb{\$\langle\mbox{\textit{char}}\rangle$}
\end{verbatimwithnv}
```

2.3.4 Verbatim Input

The last thing varvbtm gives you is;

```
\(re)newverbatiminput{⟨command⟩}[⟨n-args⟩][⟨default⟩]%
                     {⟨beg-def-outer⟩}{⟨beg-def-inner⟩}%
                     {⟨end-def-inner⟩}{⟨end-def-outer⟩}
```

to define a ⟨*command*⟩ to `\input` a file. Since this define a ⟨*command*⟩ instead of an environment, ⟨*command*⟩ should have '`\`' as its prefix. The ⟨*command*⟩ has at least one mandatory argument, ⟨*file*⟩ to be input, which can be referred as first argument if `[`⟨*default*⟩`]` is not supplied, or as second otherwise. Note that, however, if the ⟨*command*⟩ does not have any other arguments, you can omit `[`⟨*n-arg*⟩`]`.

For example;

```
\newverbatiminput{\vinput}{}{}{}{}
```

defines `\vinput{`⟨*file*⟩`}` (and `\vinput*`) that `\input` a ⟨*file*⟩ as if the ⟨*file*⟩ has `\begin`/`\end{verbatim}` at its first and last lines. A little bit more complicated example;

```
\newverbatiminput{\indfnsvinput}[2][\footnotesize]%
        {\begin{itemize}\item[]#1}{}{}{\end{itemize}}
```

defines a indented-footnotesize-by-default version of `\vinput`.

[1] Strictly speaking, the font used when `\VVBnonverb` is invoked. Thus if `\VVBnonverb` is preceded by a font changing command, the fond chosen by the command will be used.

3 Implementation

3.1 Tricks for Compatibility

\ifnvb@LaTeXe

At the very beginning of each style file, we check whether it is loaded by LaTeX 2_ε or LaTeX-2.09, declare the flag `\ifnvb@LaTeXe` and set it to true or false, and then declare `\NeedsTeXFormat` and `\ProvidesPackage` if LaTeX 2_ε. This part is embedded at the beginning of newvbtm.dtx in a tricky manner to let docstrip produce the following code, for example for newvbtm.

```
\newif\ifnvb@LaTeXe
\def\next{LaTeX2e}
\ifx\fmtname\next \nvb@LaTeXetrue
\def\next{
\NeedsTeXFormat{LaTeX2e}[1994/12/01]
\ProvidesPackage{newvbtm}}
\else \nvb@LaTeXefalse
\def\next[#1]{}\fi
\next
[1999/08/11 v1.0 ]
```

In varvbtm, loading newvbtm also takes care of the compatibility. That is, newvbtm is loaded by `\RequirePackage` if LaTeX 2_ε, while simply by `\input` otherwise. Since newvbtm does not modifies any existential macros nor declares anything by `\new` except for `\newif`[2], it is safe even if `\newvbtm` is loaded multiple times in LaTeX-2.09.

```
1
2 \ifnvb@LaTeXe
3 \RequirePackage{newvbtm}
4 \else
5 \input{newvbtm.sty}
6 \fi
```

3.2 newvbtm

\newverbatim
\renewverbatim
\nvb@newenv
\nvb@Xnewverbatim

The macros `\newverbatim` and `\renewverbatim` call a macro `\nvb@newverbatim` after `\nvb@newenv` and `\nvb@Xnewverbatim` are made `\let`-equal to appropriate macros. That is, `\nvb@newenv` is either `\newenvironment` or `\renewenvironment`, and `\nvb@Xnewverbatim` is, in both cases, `\nvb@xnewverbatim` that is the body of our environment definition as explained later. Note that `\nvb@Xnewverbatim` will be set differently in the macros defined in varvbtm.

\nvb@newverbatim
\nvb@inewverbatim

The common macro `\nvb@newverbatim` checks the existence of `[`⟨*n-args*⟩`]` and then calls `\nvb@inewverbatim` to check that of `[`⟨*default*⟩`]`. Note that since LaTeX-2.09's `\(re)newenvironment` does not have `[`⟨*default*⟩`]`, these macros have definitions slightly different from those for LaTeX 2_ε to pass an empty argument to `\nvb@Xnewverbatim` always.

```
7
8 \def\newverbatim{\let\nvb@newenv\newenvironment
9         \let\nvb@Xnewverbatim\nvb@xnewverbatim \nvb@newverbatim}
10 \def\renewverbatim{\let\nvb@newenv\renewenvironment
```

[2] Multiple `\newif` for a flag is safe, and thus we do it for `\nvb@LaTeXe` both in newvbtm and varvbtm.

```
        \let\nvb@Xnewverbatim\nvb@xnewverbatim \nvb@newverbatim}

\ifnvb@LaTeXe
\def\nvb@newverbatim#1{\@ifnextchar[%]
        {\nvb@inewverbatim{#1}}{\nvb@inewverbatim{#1}[0]}}
\def\nvb@inewverbatim#1[#2]{\@ifnextchar[%]
        {\nvb@Xnewverbatim{#1}[#2]}{\nvb@Xnewverbatim{#1}[#2][]}}
\else
\def\nvb@newverbatim#1{\@ifnextchar[%]
        {\nvb@inewverbatim{#1}}{\nvb@Xnewverbatim{#1}[0][]}}
\def\nvb@inewverbatim#1[#2]{\nvb@Xnewverbatim{#1}[#2][]}
\fi

```

\nvb@xnewverbatim
\nvb@currenvir
\nvb@defxverbatim
\nvb@xverbatim
\nvb@beginhook
\nvb@endinhook
\nvb@endouthook

The macro `\nvb@xnewverbatim` performs the essential part of the function to define a verbatim-like environment. First it checks if ⟨*default*⟩ is empty so that this optional argument is not passed to `\(re)newenvironment`, especially of LaTeX-2.09. Then it calls `\(re)newenvironment` to define ⟨*env*⟩ which performs the following in ⟨*beg-def*⟩ part.

b1. `\def`-ine `\nvb@currenvir` to let it have ⟨*env*⟩ as its body, so that the name of environment can be referred by the macros in varvbtm.

b2. Execute ⟨*beg-def-outer*⟩.

b3. Open a group to localize assignments in LaTeX's `\@verbatim`, especially those of flags referred in `\list`-related macros.

b4. Perform what LaTeX's `\verbatim` does, i.e. `\@verbatim` etc., except for `\@xverbatim`. In starred-version, only `\@verbatim` is called in this step.

b5. Call `\nvb@defxverbatim` to define `\nvb@xverbatim` that performs what LaTeX's `\@xverbatim` does but its argument terminator is "`\end{`⟨*env*⟩`}`". The definition of `\nvb@defxverbatim` is done in a group in which '`\`', '`{`' and '`}`' have "other" `\catcode` with the well-known technique using '`|`', '`[`' and '`]`'.

b6. Call `\nvb@beginhook`, which is usually `\relax` but will not be if `\VVBnonverb` is executed in ⟨*beg-def-outer*⟩ as described in §3.3.3.

b7. Execute ⟨*beg-def-inner*⟩.

b8. Call `\nvb@xverbatim` to typeset the body of environment verbatim.

The ⟨*end-def*⟩ part for ⟨*env*⟩ performs the following.

e1. Execute ⟨*end-def-inner*⟩.

e2. Call `\nvb@endinhook`, which is `\relax` now but could do something if necessary.

e3. Call `\endverbatim` to close the `\trivlist` opened by `\@verbatim`.

e4. Close the group opened by ⟨*beg-def*⟩.

e5. Execute ⟨*end-def-outer*⟩.

e6. Call `\nvb@endouthook`, which is also `\relax` now but might not be.

e7. Turn `\if@endpe` true to tell `\end` that ⟨*env*⟩ is list-like.

```
\long\def\nvb@xnewverbatim#1[#2][#3]#4#5#6#7{\def\@tempa{#3}%
        \ifx\@tempa\@empty \def\@tempa{[#2]}%
        \else              \def\@tempa{[#2][#3]}\fi
        \def\@tempb{\nvb@newenv{#1}}%
        \expandafter\@tempb\@tempa
                {\def\nvb@currenvir{#1}%
                        #4\begingroup \@verbatim \frenchspacing \@vobeyspaces
                        \nvb@defxverbatim{#1}\nvb@beginhook #5\nvb@xverbatim}%
                {#6\nvb@endinhook \endverbatim \endgroup
                        #7\nvb@endouthook \@endpetrue}%
        \def\@tempb{\nvb@newenv{#1*}}%
        \expandafter\@tempb\@tempa
                {\def\nvb@currenvir{#1*}%
                        #4\begingroup \@verbatim
                        \nvb@defxverbatim{#1*}\nvb@beginhook #5\nvb@xverbatim}%
                {\@nameuse{end#1}}}
\let\nvb@beginhook\relax
\let\nvb@endinhook\relax
\let\nvb@endouthook\relax

\begingroup \catcode`\|\z@ \catcode`\[\@ne \catcode`\]\tw@
\@makeother\{ \@makeother\} \@makeother\\
|gdef|nvb@defxverbatim#1[|long|def|nvb@xverbatim##1\end{#1}[##1|end[#1]]]
|endgroup
```

3.3 varvbtm

3.3.1 Tab Emulation

For tab emulation, we declare the following registers.

`\c@VVBtabwidth`
- The count register `\c@VVBtabwidth` that has the number of characters between tab stops. The default value 8 is set.

`\vvb@tabwidth`
- The dimen register `\vvb@tabwidth` that has the distance between tab stops.

`\vvb@everypar`
- The toks register `\vvb@everypar` to save the contents of `\everypar`, though it is usually ineffective.

`\vvb@tabbox`
- The box register `\vvb@tabbox` that contains stuffs between the beginning of line or a tab stop and the end of line or another tab stop.

```

%% Tab Emulation

\newcounter{VVBtabwidth}\c@VVBtabwidth8
\newdimen\vvb@tabwidth
\newtoks\vvb@everypar
\newbox\vvb@tabbox

```

`\VVBbegintab` `\vvb@tabfil` `\vvb@tabdef`
The macro `\VVBbegintab`, which should be called from ⟨*beg-def-inner*⟩ part, performs a few preprocessing for tabbing some of which override the definition in `\@verbatim`.

A line in a tabbing verbatim environment is enclosed in a box `\vvb@tabbox` so that we can know the horizontal position of a tab. Thus, we let `\everypar` call `\vvb@tabbol` to

open the box, and \par be \vvb@tabeol to close. Note that \obeylines after the \let for \par is necessary to make '^M' \let-equal to our own \par.

Then we set \vvb@tabwidth to \c@VVBtabwidth×⟨*width-of-A*⟩. We also \def-ine \vvb@tabfil to produce a sequence of '␣' in the case of starred environment, i.e. the \catcode of space is not \active.

Finally, we make ^I active and \let-equal to \vvb@tab by \vvb@tabdef.

\VVBendtab The macro \VVBendtab, which should be called from ⟨*end-def-inner*⟩ part, closes the box \vvb@tabbox if necessary, i.e. \end{⟨*env*⟩} is not at the beginning of a line and thus not in vertical mode.

```
\def\VVBbegintab{\vvb@everypar\everypar
        \everypar{\vvb@tabbol \the\vvb@everypar}%
        \let\par\vvb@tabeol \obeylines
        \settowidth\vvb@tabwidth{A}\multiply\vvb@tabwidth\c@VVBtabwidth
        \ifnum\catcode`\ =\active \let\vvb@tabfil\relax
        \else                     \def\vvb@tabfil{\leaders\hbox{\char`\ }}\fi
        \catcode`\^^I\active \vvb@tabdef}
{\catcode`\^^I\active \gdef\vvb@tabdef{\let^^I\vvb@tab}}
\def\VVBendtab{\ifvmode\else \par \fi}

```

\vvb@tabbol \vvb@tabeol The macro \vvb@tabbol opens the box \vvb@tabbox, while \vvb@tabeol closes it after \leavevmode to ensure the box is opened and puts the contents of the box as the last (and maybe only one) element of a paragraph terminated by \@@par. The flag \if@tempswa, which \@verbatim initiated to false, is examined in order to prevent ^M just following \begin{⟨*env*⟩} from making an empty line. Since the flag is turned true by both \vvb@tabbol and \vvb@tabeol, a ^M is in effect in other cases.

```
\def\vvb@tabbol{\@tempswatrue \setbox\vvb@tabbox\hbox\bgroup}
\def\vvb@tabeol{\if@tempswa
        \leavevmode \egroup \box\vvb@tabbox \@@par \penalty\interlinepenalty
        \fi \@tempswatrue}

```

\vvb@tab The macro \vvb@tab, to which ^I is made \let-equal, is the heart of tabbing. It first closes \vvb@tabbox after \leavevmode to ensure its opening. Then it moves to the next tab stop by putting a box of $(\lfloor w/t \rfloor + 1) \times t$ wide, where w is the width of \vvb@tabbox and t is \vvb@tabwidth. In the box, the contents of \vvb@tabbox is flushed left by \vvb@tabfil \hfil, which makes invisible space in non-starred environment because \vvb@tabfil is relax, while produces a sequence of '␣' by \leaders in starred environment. After the box is put, it opens \vvb@tabbox again by \vvb@tabbol.

```
\def\vvb@tab{\leavevmode \egroup
        \@tempdima\wd\vvb@tabbox \divide\@tempdima\vvb@tabwidth
        \multiply\@tempdima\vvb@tabwidth \advance\@tempdima\vvb@tabwidth
        \hbox to\@tempdima{\unhbox\vvb@tabbox \vvb@tabfil\hfil}\vvb@tabbol}

```

\newtabverbatim \renewtabverbatim \vvb@xnewtabverbatim Finally, we define \newtabverbatim and \renewtabverbatim for tabbing verbatim environment definition. They call the common macro \nvb@newverbatim described in §3.2 to check the existence of optional arguments as \(re)newverbatim does, but \nvb@Xnewverbatim is made \let-equal to \vvb@xnewtabverbatim that simply calls \nvb@xnewverbatim attaching \VVBbegintab and \VVBendtab to ⟨*beg-def-inner*⟩ and ⟨*end-def-inner*⟩ respectively.

```
\def\newtabverbatim{\let\nvb@newenv\newenvironment
        \let\nvb@Xnewverbatim\vvb@xnewtabverbatim \nvb@newverbatim}
\def\renewtabverbatim{\let\nvb@newenv\renewenvironment
        \let\nvb@Xnewverbatim\vvb@xnewtabverbatim \nvb@newverbatim}
\def\vvb@xnewtabverbatim#1[#2][#3]#4#5#6{%
        \nvb@xnewverbatim{#1}[#2][#3]{#4}{#5\VVBbegintab}{\VVBendtab#6}}

%%^L
```

3.3.2 Form Feed Character

\VVBprintFF
\vvb@printFF
\VVBprintFFas
\vvb@printFFas

The macro \VVBprintFF simply makes ^L \let-equal to \vvb@printFF whose body is print image of ^L and is defined by \VVBprintFFas. Since the body of \VVBprintFF has ^L that usually cannot appear in the body of a macro because of its \active-ness and *outerness*, its \def-inition is enclosed in a group in which ^L is made \relax together with that of \VVBbreakatFF.

The macro \VVBprintFFas, cooperating with \VVB@printFFas, defines its argument ⟨*str*⟩ verbatim as the body of \vvb@printFF by a well-known trick with grouping and \@sanitize used in, for example, \index. The default print image "^L" is also defined by \VVBprintFFas.

\VVBbreakatFF
\vvb@FFpar
\vvb@breakFF
\vvb@parafterFF

The macro \VVBbreakatFF makes ^L \let-equal to \vvb@breakFF and saves the definition of \par in \vvb@FFpar because it will be modified by \vvb@breakFF. The macro \vvb@breakFF breaks the current page and then makes \par, and ^M by \obeylines, \let-equal to \vvb@parafterFF. Since \vvb@parafterFF will do \par saved in \vvb@FFpar only when horizontal mode, ^M just following ^L will not produce an empty line at the beginning of the new page. After the first ^M in the page, \par and ^M regain their original definitions.

\VVBbreakatFFonly
\vvb@FF@par
\vvb@FFpenalty
\vvb@parnobreak

The macro \VVBbreakatFFonly does what \VVBbreakatFF by calling it but before that it makes \par \let-equal to \vvb@parnobreak saving its definition in \vvb@FF@par. The macro \vvb@parnobreak temporarily makes \penalty \let-equal to \@tempcnta in order that \penalty⟨*num*⟩ in original \par saved in \vvb@FF@par do nothing. Then it restores \penalty from \vvb@FFpenalty and inserts \nobreak to inhibit page break at ^M. The temporary modification of \penalty is done \global-ly because of the compatibility with the tabbing verbatim.

```

%% Form Feed Character

\begingroup \let^^L\relax
\gdef\VVBprintFF{\let^^L\vvb@printFF}
\gdef\VVBbreakatFF{\let^^L\vvb@breakFF \let\vvb@FFpar\par}
\endgroup

\def\VVBprintFFas{\begingroup \@sanitize \vvb@printFFas}
\def\vvb@printFFas#1{\endgroup \def\vvb@printFF{#1}}
\VVBprintFFas{^L}

\def\vvb@breakFF{\par \vfil \break \let\par\vvb@parafterFF \obeylines}
\def\vvb@parafterFF{\ifhmode \vvb@FFpar \fi \let\par\vvb@FFpar \obeylines}

\gdef\VVBbreakatFFonly{\let\vvb@FF@par\par
        \let\par\vvb@parnobreak \obeylines \VVBbreakatFF}
```

```
\let\vvb@FFpenalty\penalty
\def\vvb@parnobreak{\global\let\penalty\@tempcnta \vvb@FF@par
        \global\let\penalty\vvb@FFpenalty \nobreak}

%%^L
```

3.3.3 Non-Verbatim

\VVBnonverb
\vvb@nvfont
\vvb@currsize
\vvb@beginhook

The macro \VVBnonverb saves the current font in \vvb@nvfont. If LaTeX 2_ε, its body will be a sequence of \fontencoding⟨*curr-encoding*⟩ and its relatives followed by \selectfont. Otherwise, its body will be the current font size command saved in \vvb@currsize followed by the current font produced by \the\font. Then \VVBnonverb defines \nvb@beginhook so as to call \vvb@nonverb with the argument \⟨*char*⟩ just before ⟨*beg-def-inner*⟩ is executed.

\VVBnonverbmath
\vvb@nonverbmath
\vvb@@bnonverb
\vvb@@enonverb

The macro \VVBnonverbmath examines the existence of its optional argument \⟨*char*⟩ and calls \VVBnonverb via \vvb@nonverbmath with it or with \$ if omitted. Prior to the call, it makes both \vvb@@bnonverb and \vvb@@enonverb \let-equal to '$' so that the non-verbatim part is surrounded by them.

```

%% Non-Verbatim

\def\VVBnonverb#1{\ifnvb@LaTeXe
        \edef\vvb@nvfont{\noexpand\fontencoding{\f@encoding}%
                \noexpand\fontfamily{\f@family}%
                \noexpand\fontseries{\f@series}%
                \noexpand\fontshape{\f@shape}%
                \noexpand\fontsize{\f@size}{\noexpand\f@baselineskip}%
                \noexpand\selectfont}%
        \else
        \let\vvb@currsize\@currsize
        \edef\vvb@nvfont{\noexpand\vvb@currsize \the\font}\fi
        \def\nvb@beginhook{\vvb@nonverb#1}}
\def\VVBnonverbmath{\@ifnextchar[%]
        {\vvb@nonverbmath}{\vvb@nonverbmath[\$]}}
\def\vvb@nonverbmath[#1]{\let\vvb@@bnonverb$\let\vvb@@enonverb$\VVBnonverb#1}

```

\vvb@nonverb
\vvb@bnonverb
\vvb@enonverb
\vvb@@bnonverb
\vvb@@enonverb
\do
\vvb@regaincat

The macro \vvb@nonverb defines the \active ⟨*char*⟩ to open a \hbox after \leavevmode and then to call \vvb@bnonverb to do the following. First it selects the font saved in \vvb@nvfont and then restores \catcode of special characters by \vvb@regaincat. Since the body of \vvb@regaincat is the expansion result of \dospecials with the defintion of \do as;

```
\def\do#1{\catcode`\noexpand#1\number\catcode`#1\relax}
```

it should be the sequence of "\catcode`\␣10" and so on. The macro \vvb@bnoverb also set \catcode of characters in \verbatim@nolig@list if exists or '' otherwise to 12 (other). Then the \catcode of ⟨*char*⟩ is made \active because it might not be by the preceding \catcode modification. Finaly it calls \vvb@enonverb to get non-verbatim stuff.

The macro \vvb@enonverb, which is also defined in \vvb@nonverb, gets everything before ⟨*char*⟩, puts it in the \hbox surrounding it by \vvb@@bnonverb and \vvb@@enonverb, which are both '$' in the case of \VVBnonverbmath but \relax otherwise, and then closes the \hbox.

Since the definitions of the ⟨*char*⟩ and `\vvb@enonverb` should have `\active` ⟨*char*⟩, we use the trick with `\lowercase` in which the `\lccode` of '`~`' is the code of ⟨*char*⟩.

After the definitions, the character '`\`' is made `\active` and `\let`-equal to `\vvb@esc` by `\vvb@escdef` so that we can find "`end{`⟨*env*⟩`}`", stored in `\vvb@endenvir` by `\vvb@enddef`, following '`\`'. Note that we cannot use the conventional scheme to get everything in the body of ⟨*env*⟩ by `\nvb@xverbatim` because the `\catcode` of special characters are modified in non-verbatim part. Thus we make '`\`' active and `\vvb@xverbatim` `\relax`.

```
\def\vvb@nonverb#1{\catcode`#1\active \begingroup \lccode`\~`#1\relax
        \lowercase{\endgroup
                \def~{\leavevmode \hbox\bgroup \vvb@bnonverb#1}%
                \def\vvb@enonverb##1~{\vvb@@bnonverb
                        ##1\vvb@@enonverb \egroup}}%
        \catcode`\\\active \vvb@escdef \vvb@enddef \let\nvb@xverbatim\relax}
\def\vvb@bnonverb#1{\vvb@nvfont \vvb@regaincat
        \ifx\verbatim@nolig@list\undefined \@makeother\`\relax
        \else \let\do\@makeother \verbatim@nolig@list \fi
        \catcode`#1\active \vvb@enonverb}
\let\vvb@@bnonverb\relax
\let\vvb@@enonverb\relax
\def\do#1{\catcode`\noexpand#1\number\catcode`#1\relax}
\edef\vvb@regaincat{\dospecials}
```

`\vvb@escdef` `\vvb@enddef` `\vvb@endenvir`

The macro `\vvb@escdef` simply let `\active` character '`\`' act as the macro `\vvb@esc`, while `\vvb@enddef` defines the macro `\vvb@endenvir` as "`end{`⟨*env*⟩`}`" referring the environment name saved in `\vvb@currenvir`. Since these two macros has '`\`', '`{`' and '`}`' of "other" category, the well-known technique replacing them is used.

```
\begingroup \catcode`\|\z@ \catcode`\[\@ne \catcode`\]\tw@
\@makeother\{ \@makeother\} \catcode`\\|active
|gdef|vvb@escdef[|let\|vvb@esc]
|gdef|vvb@enddef[|edef|vvb@endenvir[end{|nvb@currenvir}]]
|endgroup
```

`\vvb@esc` `\vvb@checkend`

The macro `\vvb@esc` for '`\`' in `\active` examines if it is followed by `end{`⟨*env*⟩`}` stored in `\vvb@endenvir`. The examination is done in character-by-character manner by `\vvb@checkend` because we might have a partially matching sequence followed by non-verbatim stuff which cannot be picked before the `\catcode` modification. The comparison for the examination is done by `\ifx` because we might have an `\active` character.

If we find the terminator, we call `\end{`⟨*env*⟩`}` to close the environment. Otherwise, the `\char`-acter '`\`' followed by the partial matched (possibly empty) sequence followed by the unmatched character are inserted back.

```
\def\vvb@esc{\let\@tempa\vvb@endenvir \let\@tempb\@empty \vvb@checkend}
\def\vvb@checkend#1{\edef\@tempc{\expandafter\@car\@tempa\@nil}%
        \def\@tempd{#1}\ifx\@tempc\@tempd
                \edef\@tempa{\expandafter\@cdr\@tempa\@nil}%
                \ifx\@tempa\@empty
                        \edef\next{\noexpand\end{\nvb@currenvir}}%
                \else
                        \edef\@tempb{\@tempb#1}\let\next\vvb@checkend \fi
        \else   \def\next{\char`\\\@tempb#1}\fi
```

```
        \next}

%%^L
```

3.3.4 Verbatim Input

\newverbatiminput
\renewverbatiminput
\vvb@Xnewverbatim

The macros \newverbatiminput and \renewverbatiminput call the common macro \nvb@newverbatim described in §3.2 to check the existence of optional arguments, making \nvb@Xnewverbatim \let-equal to their own version, \nvb@xnewvinput.

```

%% Verbatim Input

\def\newverbatiminput{\let\nvb@newenv\newenvironment
        \let\nvb@Xnewverbatim\vvb@xnewvinput \nvb@newverbatim}
\def\renewverbatiminput{\let\nvb@newenv\renewenvironment
        \let\nvb@Xnewverbatim\vvb@xnewvinput \nvb@newverbatim}

```

\nvb@xnewvinput
\nvb@xnewvinputnodef
\nvb@xnewvinputdefault

The macro \nvb@xnewvinput defines environments of weird names, ⟨*command*⟩ name followed by a space and a '*' for starred-version. The ⟨*command*⟩ itself is defined to call \begin{⟨*env*⟩} or \begin{⟨*env*⟩*}, where ⟨*env*⟩ is what we now define, according to the existence of '*' following the ⟨*command*⟩.

Prior to defining ⟨*env*⟩, we check if the optional ⟨*n-args*⟩ is zero, and makes it one if so for the argument ⟨*file*⟩. We also check the existence of the ⟨*default*⟩ argument, because if omitted ⟨*file*⟩ is the first argument as \vvb@xnewvinputnodef defines, while the second otherwise as \vvb@xnewvinputdefault does. The core of the definition is in the ⟨*beg-def-inner*⟩ part given to \nvb@xnewverbatim. In this part, we redefine \nvb@xverbatim as \end{⟨*env*⟩} and then \input the ⟨*file*⟩ so that the environment is immediately closed after the ⟨*file*⟩ is read verbatim.

```
\def\vvb@xnewvinput#1[#2][#3]{%
        \edef\@tempa{\expandafter\@cdr\string#1\@nil\space}
        \edef#1{\noexpand\@ifstar{\noexpand\begin{\@tempa*}}%
                                  {\noexpand\begin{\@tempa}}}%
        \ifnum#2=\z@
                \edef\@tempa{\noexpand\nvb@xnewverbatim{\@tempa}[1]}
        \else
                \edef\@tempa{\noexpand\nvb@xnewverbatim{\@tempa}[#2]}\fi
        \def\@tempb{#3}\ifx\@tempb\@empty \let\@tempb\vvb@xnewvinputnodef
                        \else              \let\@tempb\vvb@xnewvinputdefault \fi
        \@tempb[#3]}
\def\vvb@xnewvinputnodef[#1]#2#3{%
        \@tempa[#1]{#2}{#3%
                \edef\nvb@xverbatim{\noexpand\end{\nvb@currenvir}}\input{##1}}}
\def\vvb@xnewvinputdefault[#1]#2#3{%
        \@tempa[#1]{#2}{#3%
                \edef\nvb@xverbatim{\noexpand\end{\nvb@currenvir}}\input{##2}}}
```

Acknowledgments

The author thanks to Noboru Matsuda and Carlos Puchol whose posts to news groups triggered writing very first version of macros in newvbtm and varvbtm.

For the implementation of these style files, the author refers the base implementations of the macros for `verbatim` environment. These macros are written by Leslie Lamport as a part of LaTeX-2.09 and LaTeX 2_ε (1997/12/01) to which Johannes Braams and other authors also contributed.