Colored Verbatim

A vivid look at TEX

Hans Hagen

april 18 1996

Abstract

This module implements (just another) verbatim environment. Especially when the output of TEX is viewed on an electronic medium, coloring has a positive influence on the readability of TEX sources. About half of the module is therefore dedicated to typesetting TEX specific character sequences in color. In this article I'll present some macro's for typesetting inline, display and file verbatim. The macro's are capable of handling ⟨*tabs*⟩ too.

At PRAGMA we use the integrated environment TEXEDIT for editing and processing TEX documents. This program also supports real time spell checking and TEX based file management. Although definitely not exclusive, the programs cooperate nicely with CONTEXT, an integrated, parameter driven macro package that covers most of the things we want TEX to do. Although TEX can be considered a tool for experts, we've tried to put as less a burden on non-technical users as possible. This is accomplished in the following ways:

- We've added some trivial symmetry checking to TEXEDIT. Sources are checked for the use of brackets, braces, begin-end and start-stop like constructions, with or without arguments.
- Although TEX is very tolerant to unformatted input, we stimulate users to make the ASCII source as clean as possible. Many sources I've seen in distribution sets look so awful, that I sometimes wonder how people get them working. In our opinion, a good-looking source leads to less errors.
- We use parameter driven setups and make the commands as tolerant as possible. We don't accept commands that don't look nice in ASCII.
- Finally —I could have added some more— we use color.

When in spell-checking-mode, the words spelled correctly are shown in *green*, the unknown or wrongly spelled words are in *red* and upto four categories of words, for instance passive verbs and nouns, become *blue* (cyan) or *yellow*. Short and nearly always correct words are in white (on a black screen). This makes checking-on-the-fly very easy and convenient, especially because we place the accents automatically.

In TEX-mode we show TEX-specific stuff in appropriate colors and again we use four colors. We use those colors in a way that supports parameter driven setups, table typesetting and easy visual checking of symmetry. Furthermore the text becomes more readable.

color	**characters that are influenced**
red	`{ } $`
green	`\this \!!that \??these \@@those`
yellow	`‘ ’ ˜ ˆ _ & / + - | %`
blue	`( ) # [ ] " < > =`

Macro-definition and style files often look quite green, because they contain many calls to macros. Pure text files on the other hand are mostly white (on the screen) and color clearly shows their structure.

When I prepared the interactive PDF manuals of CONTEXT, TEXEDIT and PPCHTEX, I decided to include the original source text of the manuals as an appendix. At every chapter or (sub)section the reader can go to the corresponding line in the source, just to see how things were done in TEX. Of course, the reader can jump from the source to corresponding typeset text too.

Confronted with those long (boring) sources, I decided that a colored output, in accordance with TEXEDIT would be nice. It would not only visually add some quality to the manual, but also make the sources more readable.

Apart from a lot of ⟨*catcode*⟩-magic, the task at hand was surprisingly easy. Although the macro's are hooked into the standard CONTEXT verbatim mechanism, they are set up in a way that embedding them in another verbatim environment is possible.

This module includes part of the CONTEXT verbatim environment too, because it shows a few tricks that are often overseen by novice, especially the use of the TEX primitive `\meaning`. First I'll show in what way the users are confronted with verbatim typesetting. Because we want to be able to test for symmetry and because we hate the method of closing down the verbatim mode with some strange active character, we use the following construction for display verbatim:

```
\starttyping
the Dutch word 'typen' stands for 'typing', therefore the Dutch
implementation is in fact \starttypen ... \stoptypen
\stoptyping
```

Files can be typed with `\typefile` and inline verbatim can be accomplished with `\type`. This last command comes in two flavors:

```
We can say \type<<something>> or \type{something}. The first one is a bit
longer but also supports slanted typing, which accomplished by typing
\type<<a <<slanted>> word>>. We can also use commands to enhance the text
\type<<with <</bf boldfaced>> text>>. Just to be complete, we decided
to accept also \LaTeX\ alike verbatim, which means that \type+something+
and \type|something| are valid commands too.
```

These commands can be tuned with accompanying setup commands. We can enable commands, slanted typing, control spaces, ⟨*tab*⟩-handling and (here we are:) coloring. We can also setup surrounding white space and indenting.

I only present the framework macro's here, because the CONTEXT-setup command uses specific interface macros.[1] Embedding is up to the user.

The definition part of this text is typeset in color or grayscales. One has to keep in mind that the purpose of these macros was viewing TEX on an electronic medium. On paper, the results can be disappointing, because the quality depends on the printer. We start with some general macro's, some of which are only defined if they are `\undefined`.

```
\chardef\escapecode=0       \chardef\begingroupcode=1
\chardef\lettercode=11      \chardef\endgroupcode=2
\chardef\activecode=13
```

```
\def\zeropoint{0pt}
```

```
\ifx\scratchcounter\undefined \newcount\scratchcounter \fi
\ifx\everyline\undefined      \newtoks\everyline       \fi
\ifx\tempreadfile\undefined   \newread\tempreadfile    \fi
\ifx\verbatimfont\undefined   \def\verbatimfont{\tt}   \fi
```

```
\newif\ifitsdone
```

The inline verbatim commands presented here are a subset of the CONTEXT ones. Both grouped and character bound alternatives are provided. This command takes one argument: the closing command:

```
\processinlineverbatim{\closingcommand}
```

One can define his own verbatim commands, which can be very simple:

```
\def\Verbatim{\processinlineverbatim\relax}
```

or more complex:

```
\def\GroupedVerbatim%
  {\bgroup
   \dosomeusefullthings
   \processinlineverbatim\egroup}
```

Before entering inline verbatim mode, we take care of the unwanted ⟨*tabs*⟩, ⟨*newlines*⟩ and ⟨*newpages*⟩ (form feeds) and turn them into ⟨*space*⟩. We need the double `\bgroup` construction to keep the closing command local.

```
\def\setupinlineverbatim%
  {\verbatimfont
   \let\obeytabs=\ignoretabs
   \let\obeylines=\ignorelines
```

[1]At the moment CONTEXT has a Dutch interface. One of our targets is to fully document the source and make it public. As can be seen in the PPCHTEX-distribution, the underlying macros permit a multilingual interface, so we'll probably come up with an English version someday.

```
   \let\obeypages=\ignorepages
   \setupcopyverbatim}
```

6
```
\def\doprocessinlineverbatim%
  {\ifx\next\bgroup
     \setupinlineverbatim
     \catcode‘\{=\begingroupcode
     \catcode‘\}=\endgroupcode
     \def\next{\let\next=}%
   \else
     \setupinlineverbatim
     \def\next##1{\catcode‘##1=\endgroupcode}%
   \fi
   \next}
```

7
```
\def\processinlineverbatim#1%
  {\bgroup
   \def\endofverbatimcommand{#1\egroup}%
   \bgroup
   \aftergroup\endofverbatimcommand
   \futurelet\next\doprocessinlineverbatim}
```

The closing command is executed afterwards as an internal command and therefore should not be given explicitly when typesetting inline verbatim.

We can define a display verbatim environment with the command `\processdisplayverbatim` in the following way:

```
\processdisplayverbatim{\closingcommand}
```

For instance, we can define a simple command like:

```
\def\BeginVerbatim {\processdisplayverbatim{EndVerbatim}}
```

But we can also do more advance things like:

```
\def\BeginVerbatim {\bigskip \processdisplayverbatim{\EndVerbatim}}
\def\EndVerbatim   {\bigskip}
```

When we compare these examples, we see that the backslash in the closingcommand is optional. One is free in actually defining a closing command. If one is defined, the command is executed after ending verbatim mode.

8
```
\def\processdisplayverbatim#1%
  {\par
   \bgroup
   \escapechar=-1
   \xdef\verbatimname{\string#1}%
   \egroup
   \def\endofdisplayverbatim{\csname\verbatimname\endcsname}%
   \bgroup
   \parindent\zeropoint
   \ifdim\lastskip<\parskip
     \removelastskip
     \vskip\parskip
   \fi
   \parskip\zeropoint
   \processingverbatimtrue
   \expandafter\let\csname\verbatimname\endcsname=\relax
   \edef\endofverbatimcommand{\csname\verbatimname\endcsname}%
   \edef\endofverbatimcommand{\meaning\endofverbatimcommand}%
   \verbatimfont
   \setupcopyverbatim
   \let\doverbatimline=\relax
   \copyverbatimline}
```

The closing is saved in `\endofverbatimcommand` in such a way that it can be compared on a line by line basis. For the conversion we use `\meaning`, which converts the line to non-expandable tokens. We reset `\parskip`, because we don't want inter-paragraph skips to creep into the verbatim source. Furthermore we `\relax` the line-processing macro while getting the rest of the first line. The initialization command `\setupcopyverbatim` does just what we expect it to do: give all characters ⟨*catcode*⟩ 11. Furthermore we switch to french spacing and call for obeyance.

```
\def\setupcopyverbatim%
  {\uncatcodecharacters
   \frenchspacing
   \obeyspaces
   \obeytabs
   \obeylines
   \obeycharacters}
```

As its name says, `\uncatcodecharacters` resets the ⟨*catcode*⟩ of characters. Because we use an upper bound of 127, characters with higher values are not taken into account. When one wants to do special things with higher characters, this macro should be adapted.

```
\def\uncatcodecharacters%
  {\scratchcounter=0
   \loop
     \catcode\scratchcounter=\lettercode
     \advance\scratchcounter by 1
     \ifnum\scratchcounter<127
   \repeat}
```

We follow Knuth in naming macros that make ⟨*space*⟩, ⟨*newline*⟩ and ⟨*newpage*⟩ active and assigning them `\obeysomething`. Their assigned values are saved in `\obeyedvalue`.

```
\def\obeyedspace  {\hbox{ }}
\def\obeyedtab    {\obeyedspace}
\def\obeyedpage   {\vfill\eject}
\def\obeyedline   {\par}
```

First we define `\obeyspaces`. When we want visible spaces (control spaces) we only have to adapt the definition of `\obeyedspace`:

```
\def\controlspace {\hbox{\char32}}
```

```
\bgroup
\catcode`\ =\activecode
\gdef\obeyspaces{\catcode`\ =\activecode\def {\obeyedspace}}
\gdef\setcontrolspaces{\catcode`\ =\activecode\def {\controlspace}}
\egroup
```

Next we take care of ⟨*newline*⟩ and ⟨*newpage*⟩ and because we want to be able to typeset listings that contain ⟨*tabs*⟩, we have to handle those too. Because we have to redefine the ⟨*newpage*⟩ character locally, we redefine the meaning of this (often already) active character.

```
\catcode`\^^L=\activecode \def^^L{\par}
```

```
\bgroup
\catcode`\^^I=\activecode
\catcode`\^^M=\activecode
\catcode`\^^L=\activecode
```

```
\gdef\obeytabs    {\catcode`\^^I=\activecode\def^^I{\obeyedtab}}
\gdef\obeylines   {\catcode`\^^M=\activecode\def^^M{\obeyedline}}
\gdef\obeypages   {\catcode`\^^L=\activecode\def^^L{\obeyedpage}}
```

```
\gdef\ignoretabs  {\catcode`\^^I=\activecode\def^^I{\obeyedspace}}
\gdef\ignorelines {\catcode`\^^M=\activecode\def^^M{\obeyedspace}}
\gdef\ignorepages {\catcode`\^^L=\activecode\def^^L{\obeyedline}}
```

```
\gdef\obeycharacters{}
```

```
\gdef\settabskips%
  {\let\processverbatimline=\doprocesstabskipline%
   \catcode`\^^I=\activecode\let^^I=\doprocesstabskip}
```

```
\egroup
```

The main copying routine of display verbatim does an ordinary string-compare on the saved closing command and the line at hand. The space after `#1` in the definition of `\next` is essential! As a result of using `\obeylines`, we have to use `%`'s after each line but none after the first `#1`.

```
{\obeylines%
\gdef\copyverbatimline#1
  {\ifx\doverbatimline\relax% gobble rest of the first line
     \let\doverbatimline=\dodoverbatimline%
     \def\next{\copyverbatimline}%
   \else%
     \def\next{#1 }%
     \ifx\next\emptyspace%
       \def\next%
         {\doemptyverbatimline{#1}%
          \copyverbatimline}%
     \else%
       \edef\next{\meaning\next}%
       \ifx\next\endofverbatimcommand%
         \def\next%
           {\egroup\endofdisplayverbatim}%
       \else%
         \def\next%
           {\doverbatimline{#1}%
            \copyverbatimline}%
       \fi%
     \fi%
  \fi%
  \next}}
```

The actual typesetting of a line is done by a separate macro, which enables us to implement ⟨*tab*⟩ handling. The trick with `\do` and `\dodo` enables us to obey the preceding `\parskip`, while skipping the rest of the first line. The `\relax` is used as an signal.

```
22
\def\dodoverbatimline#1%
  {\leavevmode\the\everyline\strut\processverbatimline{#1}%
   \everypar{}%
   \obeyedline\par}
```

Empty lines in verbatim can lead to white space on top of a new page. Because this is not what we want, we turn them into vertical skips. This default behavior can be overruled by:

```
\obeyemptylines
```

Although it would cost us only a few lines of code, we decided not to take care of multiple empty lines. When a (display) verbatim text contains more successive empty lines, this probably suits some purpose.

```
23
\bgroup
\catcode`\^^L=\activecode  \gdef\emptypage  {^^L}
\catcode`\^^M=\activecode  \gdef\emptyline  {^^M}
                           \gdef\emptyspace { }
\egroup
```

```
24
\def\doemptyverbatimline%
  {\vskip\ht\strutbox
   \vskip\dp\strutbox
   {\setbox0=\hbox{\the\everyline}}}
```

```
25
\def\obeyemptylines%
  {\def\doemptyverbatimline{\doverbatimline}}
```

Although every line is a separate paragraph, we execute `\everypar` only once. In CONTEXT we use a bit different approach, because there we use `\everypar` for sidefloats, columnfloats and other features. We offer an alternative `\EveryPar`, which stacks `everypar`'s, while leaving the old one intact. For the same reason we implemented `\EveryLine`, which enables us to do things like line numbering while retaining `\everyline` behavior. Some other useful but distracting options have been removed here too.

We still have to take care of the ⟨*tabs*⟩. A ⟨*tab*⟩ takes eight spaces and a ⟨*space*⟩ normally has a width of 0.5em. because we can be halfway a tabulation, we must keep track of the position. This takes time, especially when we print complete files, therefore we `\relax` this mechanism by default.

```
26
\def\doprocesstabskip%
  {\obeyedspace %  \hskip.5em\relax
   \ifitsdone
```

```
     \advance\scratchcounter 1\relax
     \let\next=\doprocesstabskip
     \itsdonefalse
   \else\ifnum\scratchcounter>7\relax
     \let\next=\relax
   \else
     \advance\scratchcounter 1\relax
     \let\next=\doprocesstabskip
   \fi\fi
   \next}
```

```
27 \def\dodoprocesstabskipline#1#2\endoftabskipping%
  {\ifnum\scratchcounter>7\relax
     \scratchcounter=1\relax
     \itsdonetrue
   \else
     \advance\scratchcounter 1\relax
     \itsdonefalse
   \fi
   \ifx#1\relax
     \let\next=\relax
   \else
     \def\next{#1\dodoprocesstabskipline#2\endoftabskipping}%
   \fi
   \next}
```

```
28 \let\endoftabskipping=\relax
\let\processverbatimline=\relax
```

```
29 \def\doprocesstabskipline#1%
  {\bgroup
   \scratchcounter=1\relax
   \dodoprocesstabskipline#1\relax\endoftabskipping
   \egroup}
```

The verbatim typesetting of files is done on a bit different basis. This time we don't check for a closing command, but look for ⟨*eof*⟩ and make sure it does not turn into an empty line.

```
\processfileverbatim{filename}
```

Typesetting a file in most cases results in more than one page. Because we don't want problems with files that are read in during the construction of the page, we declare `\ifprocessingverbatim`, so the output routine can adapt its behavior.

```
30 \newif\ifprocessingverbatim
```

```
31 \def\processfileverbatim#1%
  {\par
   \bgroup
   \parindent\zeropoint
   \ifdim\lastskip<\parskip
     \removelastskip
     \vskip\parskip
   \fi
   \parskip\zeropoint
   \processingverbatimtrue
   \uncatcodecharacters
   \verbatimfont
   \frenchspacing
   \obeyspaces
   \obeytabs
   \obeylines
   \obeypages
   \obeycharacters
   \openin\tempreadfile=#1%
   \def\doreadline%
     {\read\tempreadfile to \next
      \ifeof\tempreadfile
        % we don't want <eof> to be treated as <crlf>
      \else\ifx\next\emptyline
        \expandafter\doemptyverbatimline\expandafter{\next}%
      \else\ifx\next\emptypage
        \expandafter\doemptyverbatimline\expandafter{\next}%
      \else
        \expandafter\dodoverbatimline\expandafter{\next}%
      \fi\fi\fi
      \readline}%
   \def\readline%
     {\ifeof\tempreadfile
        \let\next=\relax
      \else
        \let\next=\doreadline
      \fi
      \next}%
   \readline
   \closein\tempreadfile
   \egroup
   \ignorespaces}
```

These macro's can be used to construct the commands we mentioned in the beginning of this article. We leave this to the fantasy of the reader and only show some Plain TEX alternatives for display verbatim and listings. We define three commands for typesetting inline text, display text and files verbatim. The inline alternative also accepts LATEX-like verbatim.

```
\type{text}

\starttyping
... verbatim text ...
\stoptyping

\typefile{filename}
```

We can turn on the options by:

```
\controlspacetrue
\verbatimtabstrue
\prettyverbatimtrue
```

Here is the implementation:

32
```
\newif\ifcontrolspace
\newif\ifverbatimtabs
\newif\ifprettyverbatim
```

33
```
\def\presettyping%
  {\ifcontrolspace
     \let\obeyspace=\setcontrolspace
   \fi
   \ifverbatimtabs
     \let\obeytabs=\settabskips
   \fi
   \ifprettyverbatim
     \let\obeycharacters=\setupprettytextype
   \fi}
```

34
```
\def\type%
  {\bgroup
   \presettyping
   \processinlineverbatim{\egroup}}
```

35
```
\def\starttyping%
  {\bgroup
   \presettyping
   \processdisplayverbatim{\stoptyping}}
```

36
```
\def\stoptyping%
  {\egroup}
```

37
```
\def\typefile#1%
  {\bgroup
   \presettyping
   \processfileverbatim{#1}%
   \egroup}
```

One can use the different `\obeysomething` commands to influence the behavior of these macro's. We use for instance `\obeycharacters` for making `/` an active character when we want to include typesetting commands.

We'll spend the remainder of this article on coloring the verbatim text.[2] We can turn on coloring by reassigning `\obeycharacters`:

```
\let\obeycharacters=\setupprettytextype
```

During pretty typesetting we can be in two states: *command* and *parameter*. The first condition becomes true if we encounter a backslash, the second state is entered when we meet a #.

38
```
\newif\ifintexcommand
\newif\ifintexparameter
```

The mechanism described here, is meant to be used with color. It is nevertheless possible to use different fonts instead of distinctive colors. When using color, it's better to end parameter mode after the #. When on the other hand we use a slanted typeface for the hashmark, then a slanted number looks better.

39
```
\newif\ifsplittexparameters   \splittexparameterstrue
\newif\ifsplittexcontrols     \splittexcontrolstrue
```

With `\splittexcontrols` we can influence the way control characters are processed in macronames. By default, the `^^` part is uncolored. When this boolean is set to false, they get the same color as the other characters.

The next boolean is used for internal purposes only and keeps track of the length of the name. Because two-character sequences starting with a backslash are always seen as a command.

40
```
\newif\iffirstintexcommand
```

We use a maximum of four colors because more colors will distract too much. In the following table we show the logical names of the colors, their color and rgb-values.

identifier	**color**	**r**	**g**	**b**	**bw**
texcolorone	red	0.9	0.0	0.0	0.30
texcolortwo	green	0.0	0.8	0.0	0.45
texcolorthree	yellow	0.0	0.0	0.9	0.60
texcolorfour	blue	0.8	0.8	0.6	0.75

This following poor mans implementation of color is based on PostScript. One can of course use grayscales too.

41
```
\def\setcolorverbatim%
  {\splittexparameterstrue
   \def\texcolorone   {.9 .0 .0 }        % red
   \def\texcolortwo   {.0 .8 .0 }        % green
   \def\texcolorthree {.0 .0 .9 }        % blue
   \def\texcolorfour  {.8 .8 .6 }        % yellow
   \def\texbeginofpretty[##1]%
     {\special{ps:: \csname##1\endcsname setrgbcolor}}
   \def\texendofpretty%
     {\special{ps:: 0 0 0 setrgbcolor}}} % black
```

42
```
\def\setgrayverbatim%
  {\splittexparameterstrue
   \def\texcolorone   {.30 }             % gray
   \def\texcolortwo   {.45 }             % gray
   \def\texcolorthree {.60 }             % gray
   \def\texcolorfour  {.75 }             % gray
   \def\texbeginofpretty[##1]%
     {\special{ps:: \csname##1\endcsname setgray}}
   \def\texendofpretty%
     {\special{ps:: 0 setgray}}}         % black
```

[2]The original macro's have some primitive symmetry testing options.

One can redefine these two commands after loading this module. If available, one can use appropriate font-switch macro's. We default to color.

```
43 \setcolorverbatim
```

Here come the commands that are responsible for entering and leaving the two states. As we can see, they've got much in common.

```
44 \def\texbeginofcommand%
  {\texendofparameter
   \ifintexcommand
   \else
     \global\intexcommandtrue
     \global\firstintexcommandtrue
     \texbeginofpretty[texcolortwo]%
   \fi}
```

```
45 \def\texendofcommand%
  {\ifintexcommand
     \texendofpretty
     \global\intexcommandfalse
     \global\firstintexcommandfalse
   \fi}
```

```
46 \def\texbeginofparameter%
  {\texendofcommand
   \ifintexparameter
   \else
     \global\intexparametertrue
     \texbeginofpretty[texcolorthree]%
   \fi}
```

```
47 \def\texendofparameter%
  {\ifintexparameter
     \texendofpretty
     \global\intexparameterfalse
   \fi}
```

We've got nine types of characters. The first type concerns the grouping characters that become red and type seven takes care of the backslash. Type eight is the most recently added one and handles the control characters starting with ^^. In the definition part at the end of this article we can see how characters are organized by type.

```
48 \def\ifnotfirstintexcommand#1%
  {\iffirstintexcommand
     \string#1%
     \texendofcommand
   \else}
```

```
49 \def\textypeone#1%
  {\ifnotfirstintexcommand#1%
     \texendofcommand
     \texendofparameter
     \texbeginofpretty[texcolorone]\string#1\texendofpretty
   \fi}
```

```
50 \def\textypetwo#1%
  {\ifnotfirstintexcommand#1%
     \texendofcommand
     \texendofparameter
     \texbeginofpretty[texcolorthree]\string#1\texendofpretty
   \fi}
```

```
51 \def\textypethree#1%
  {\ifnotfirstintexcommand#1%
     \texendofcommand
     \texendofparameter
     \texbeginofpretty[texcolorfour]\string#1\texendofpretty
   \fi}
```

```
52 \def\textypefour#1%
     {\ifnotfirstintexcommand#1%
        \texendofcommand
        \texendofparameter
        \string#1%
      \fi}
53 \def\textypefive#1%
     {\ifnotfirstintexcommand#1%
        \texbeginofparameter
        \string#1%
      \fi}
54 \def\textypesix#1%
     {\ifnotfirstintexcommand#1%
        \ifintexparameter
          \ifsplittexparameters
            \texendofparameter
            \string#1%
          \else
            \string#1%
            \texendofparameter
          \fi
        \else
          \texendofcommand
          \string#1%
        \fi
      \fi}
55 \def\textypeseven#1%
     {\ifnotfirstintexcommand#1%
        \texbeginofcommand
        \string#1%
      \fi}
56 \def\dodotextypeeight#1%
     {\texendofparameter
      \ifx\next#1%
        \ifsplittexcontrols
          \ifintexcommand
            \texendofcommand
            \string#1\string#1%
            \texbeginofcommand
          \else
            \string#1\string#1%
          \fi
        \else
          \string#1\string#1%
        \fi
        \let\next=\relax
      \else
        \textypethree#1%
      \fi
      \next}
57 \def\textypeeight#1%
     {\def\dotextypeeight{\dodotextypeeight#1}%
      \afterassignment\dotextypeeight\let\next=}
58 \def\textypenine#1%
     {\texendofparameter
      \global\firstintexcommandfalse
      \string#1}
```

We have to take care of the control characters we mentioned before. We obey their old values but only after ending our two states.

```
59 \def\texsetcontrols%
```

```
  {\global\let\oldobeyedspace=\obeyedspace
   \global\let\oldobeyedline=\obeyedline
   \global\let\oldobeyedpage=\obeyedpage
   \def\obeyedspace%
     {\texendofcommand
      \texendofparameter
      \oldobeyedspace}%
   \def\obeyedline%
     {\texendofcommand
      \texendofparameter
      \oldobeyedline}%
   \def\obeyedpage%
     {\texendofcommand
      \texendofparameter
      \oldobeyedpage}}
```

Next comes the tough part. We have to change the ⟨*catcode*⟩ of each character. These macro's are tuned for speed and simplicity. When viewed in color they look quite simple.

```
\def\setupprettytextype%
  {\texsetcontrols
   \texsetspecialpretty
   \texsetalphabetpretty
   \texsetextrapretty}
```

When handling the lowercase characters, we cannot use lowercased macro names. This means that we have to redefine some well known macros, like `\bgroup`.

```
\def\texpresetcatcode%
  {\def\\##1%
     {\expandafter\catcode\expandafter`\csname##1\endcsname\activecode}}
```

```
\def\texsettypenine%
  {\def\\##1%
     {\def##1{\textypenine##1}}}
```

```
\bgroup
  \bgroup
    \gdef\texpresetalphapretty%
      {\texpresetcatcode
       \\A\\B\\C\\D\\E\\F\\G\\H\\I\\J\\K\\L\\M%
       \\N\\O\\P\\Q\\R\\S\\T\\U\\V\\W\\X\\Y\\Z}
    \texpresetalphapretty
    \gdef\texsetalphapretty%
      {\texpresetalphapretty
       \texsettypenine
       \\A\\B\\C\\D\\E\\F\\G\\H\\I\\J\\K\\L\\M%
       \\N\\O\\P\\Q\\R\\S\\T\\U\\V\\W\\X\\Y\\Z}
  \egroup
  \global\let\TEXPRESETCATCODE = \texpresetcatcode
  \global\let\TEXSETTYPENINE   = \texsettypenine
  \global\let\BGROUP           = \bgroup
  \global\let\EGROUP           = \egroup
  \global\let\GDEF             = \gdef
  \BGROUP
    \GDEF\TEXPRESETALPHAPRETTY%
      {\TEXPRESETCATCODE
       \\a\\b\\c\\d\\e\\f\\g\\h\\i\\j\\k\\l\\m%
       \\n\\o\\p\\q\\r\\s\\t\\u\\v\\w\\x\\y\\z}
    \TEXPRESETALPHAPRETTY
    \GDEF\TEXSETALPHAPRETTY%
      {\TEXPRESETALPHAPRETTY
       \TEXSETTYPENINE
       \\a\\b\\c\\d\\e\\f\\g\\h\\i\\j\\k\\l\\m%
       \\n\\o\\p\\q\\r\\s\\t\\u\\v\\w\\x\\y\\z}
  \EGROUP
  \gdef\texsetalphabetpretty%
```

```
    {\texsetalphapretty
     \TEXSETALPHAPRETTY}
\egroup
```

Macronames normally only contain characters. As mentioned before, we also permit the characters `@`, `!` and `?`. Of course they are only colored (green) when they are part of the name.

64
```
\bgroup
  \gdef\texpresetextrapretty%
    {\texpresetcatcode
     \\?\\!\\@}
  \texpresetextrapretty
  \gdef\texsetextrapretty%
    {\texpresetextrapretty
     \texsettypenine
     \\?\\!\\@}
\egroup
```

Here comes the main linking routine. In this macro we also have to change the escape character to `!` and use `X`, `Y` and `Z` for grouping and ignoring, which makes the result a bit less readable. Plain TEX defines `\+` as an outer macro, so we have to redefine this one too.

65
```
\def\+{\tabalign} % Plain TeX: \outer\def\+{\tabalign}
```

66
```
%<TeX_Marker>
\bgroup
  \gdef\texpresetspecialpretty%
    {\def\\##1{\catcode`##1\activecode}%
     \\\[\\\]\\\=\\\<\\\>\\\#\\\(\\\)\\\"%
     \\\$\\\{\\\}%
     \\\-\\\+\\\|\\\%\\\/\\\_\\\^\\\&\\\~\\\'\\\`%
     \\\.\\\,\\\:\\\;%
     \\\*%
     \\\1\\\2\\\3\\\4\\\5\\\6\\\7\\\8\\\9%
     \\\\}
  \catcode`\X=\the\catcode`\{
  \catcode`\Y=\the\catcode`\}
  \catcode`\Z=\the\catcode`\%
  \gdef\texsetsometypes%
    {\def\!##1##2{\def##1{##2{##1}}}}%
  XZ
   \catcode`\!=\escapecode
   !texpresetspecialpretty
   !gdef!texsetspecialpretty
     XZ
      !texpresetspecialpretty
      !texsetsometypes
      !! $ !textypeone   !! { !textypeone   !! } !textypeone
      !! [ !textypetwo   !! ] !textypetwo   !! ( !textypetwo   !! ) !textypetwo
      !! = !textypetwo   !! < !textypetwo   !! > !textypetwo   !! " !textypetwo
      !! - !textypethree !! + !textypethree !! / !textypethree
      !! | !textypethree !! % !textypethree !! ' !textypethree !! ` !textypethree
      !! _ !textypethree !! ^ !textypethree !! & !textypethree !! ~ !textypethree
      !! . !textypefour  !! , !textypefour  !! : !textypefour  !! ; !textypefour
      !! * !textypefour
      !! # !textypefive
      !! 1 !textypesix   !! 2 !textypesix   !! 3 !textypesix
      !! 4 !textypesix   !! 5 !textypesix   !! 6 !textypesix
      !! 7 !textypesix   !! 8 !textypesix   !! 9 !textypesix
      !! \ !textypeseven
      !! ^ !textypeeight
     YZ
  YZ
\egroup
```

This text is included in the file where the macro's are defined. In this article, the verbatim part of this text was set with the following commands for the examples:

```
\def\starttypen% We simplify the \ConTeXt\ macro.
  {\bgroup
   \everypar{} % We disable some \ConTeXt\ / \LaTeX/ mechanisms.
   \advance\leftskip by 1em
   \processdisplayverbatim{\stoptypen}}

\def\stoptypen%
  {\egroup}
```

The implementation itself was typeset with:

```
\def\startdefinition%
  {\bgroup
   \everypar{} % We disable some \ConTeXt\ / \LaTeX/ mechanisms.
   \let\obeycharacters=\setupprettytextype
   \everypar{\showparagraphcounter}%
   \everyline{\showlinecounter}%
   \verbatimcorps
   \processdisplayverbatim{\stopdefinition}}

\def\stopdefinition%
  {\egroup}
```

Because we have both `\everypar` and `\everyline` available, we have implemented a dual numbering mechanism:

```
\newcount\paragraphcounter
\newcount\linecounter

\def\showparagraphcounter%
  {\llap
     {\bgroup
      \counterfont
      \hbox to 4em
        {\global\advance\paragraphcounter by 1
         \hss \the\paragraphcounter \hskip2em}%
      \egroup
      \hskip1em}}

\def\showlinecounter%
  {\llap
     {\bgroup
      \counterfont
      \hbox to 2em
        {\global\advance\linecounter by 1
         \hss \the\linecounter}%
      \egroup
      \hskip1em}}
```

Of course commands like this have to be embedded in a decent setup structure, where options can be set at will.

Let's summarize the most important commands.

```
\processinlineverbatim{\closingcommand}
\processdisplayverbatim{\closingcommand}
\processfileverbatim{filename}
```

We can satisfy our own specific needs with the following interfacing macro's:

```
\obeyspaces  \obeytabs  \obeylines  \obeypages  \obeycharacters
```

Some needs are fulfilled already with:

```
\setcontrolspace  \settabskips  \setupprettytextype
```

lines can be enhanced with ornaments using:

```
\everypar  \everyline
```

and color support is implemented by:

```
\texbeginofpretty[#1] ... \texendofpretty
```

We can influence the verbatim environment with the following macro and booleans:

```
\obeyemptylines  \splittexparameters...  \splittexcontrols...
```

This macro can be redefined by the user. The parameter `#1` can be one of the four ’fixed’ identifiers *texcolorone*, *texcolortwo*, *texcolorthree* and *texcolorfour*. We have implemented a more or less general PostScript color support mechanism, using `specials`. One can toggle between color and grayscale with:

```
\setgrayverbatim  \setcolorverbatim
```

We did not mention one drawback of the mechanism described here. The closing command must start at the first position of the line. The original implementation does not have this drawback, because we test if the end command is a substring of the line at hand. Although the two macros that we use for this only take a few lines of code, we think they are out of place in this article.

One can wonder why such a simple application takes 482 lines of TEX-code. But then, TEX was never meant to be simple.