

# Verbatim phrases and listings in L<sup>A</sup>T<sub>E</sub>X

Péter Szabó (pts@fazekas.hu) \*

Budapest University of Technology and Economics,  
 Department of Analysis,  
 Műgyetem rakpart 3–9.,  
 Budapest, Hungary H-1111

2004-11-11

## Abstract

The `example` package written by the author recently provides sophisticated features for typesetting verbatim source code listings, including the display of the source code and its compiled L<sup>A</sup>T<sub>E</sub>X or METAPOST output side-by-side, with automatic width detection and enabled page breaks (in the source), without the need for specifying the source twice. Special care is taken so section, page and footnote numbers do not interfere with the main document. For typesetting short verbatim phrases, a replacement for the `\verb` command is also provided in the package, which can be used inside tables and moving arguments such as footnotes and section titles. The listings package is used for syntax highlighting.

The article reviews the design decisions made during the package development and also presents some interesting implementation internals. `example` is compared to standard L<sup>A</sup>T<sub>E</sub>X packages such as `listings`, `ltxdoc`, `sverb` and `moreverb`. The new `codep` package and its accompanying Perl script, which provide a convenient interface to the `example` package for authors of manuals, is also presented. With `codep` it is possible to generate the source code, the L<sup>A</sup>T<sub>E</sub>X or METAPOST output and the compilable example file onto the CD from a single source embedded into the appropriate place of the `.tex` document file.

## 1 Terminology

**verbatim text** A visually distinguishable textual part of the document (usually typeset with a monospaced, or typewriter font) that is allowed to contain the full ASCII character set. Verbatim text is often used to typeset parts of program source files, including T<sub>E</sub>X source. Verbatim text must be marked up (i.e. surrounded) in the source of the L<sup>A</sup>T<sub>E</sub>X document, so backslash and other control characters are typeset verbatim instead of being interpreted as commands or special L<sup>A</sup>T<sub>E</sub>X characters.

**inline verbatim** A verbatim text passage inside a paragraph or table cell.

**display verbatim** is a vertical verbatim text block between paragraphs.

**side-by-side display** When typesetting program source in a display verbatim, it is often desirable to show the output of the program as well. This is especially useful when teaching scripting languages, so the reader can see the command and its effect side-by-side in a quick glance. For T<sub>E</sub>X or METAPOST sources and EPS and PDF source files it is also useful to see the source and the typeset result side-by-side.

**Source and Sample** Side-by-side displays can be divided to *Source* and *Sample*, the latter being program output or typeset material.

**CD-files** Files accompanying a book, usually on a CD or DVD shipped with the book, or available for download on the home page of the book. These files usually contain some of the display verbatim material in the book, so readers do not have to retype them.

---

\*Thanks to Ferenc Wetzl for the fruitful long discussion about the line syntax of the `code` environment, and also for reviewing the article.

## 2 Special characters in the L<sup>A</sup>T<sub>E</sub>X source

The special meaning of the input characters in the source file must be disabled in verbatim mode – except for the character(s) that delimit the end of the verbatim text. The following characters have to be dealt with:

**ASCII symbols** The ligatures have to be disabled, especially ? ‘ → ¿ etc. The safest way is to make both characters of such a ligature active, and defining `\def?{\relax\string?\relax}` and `\def‘{\relax\string‘\relax}`. `examplep` issues similar definitions covering all such ligatures in the OT1- and T1-encoded CM fonts.

**ASCII letters** Most verbatim fonts don’t contain the “fi” or similar ligatures, so `examplep` takes no care to disable them.

**special T<sub>E</sub>X source symbols** To disable the special meaning of the symbols `\ { } $ % ^ & _`, `examplep` redefines their `\catcode` for display verbatim. However, `catcode` changes are not always appropriate for inline verbatim, so `examplep` provides `\÷` (see below) which doesn’t change `catcodes` at all.

**other ASCII punctuation** Some of these characters (`[ ] ; ’ , . / ~ ! @ * ( ) + | : " < > ? ‘ - =`) may be active, for example, `~` is `÷ ÷`, `‘` is a Babel shorthand (e.g. in the Hungarian language module, `magyar.ldr`); `"` is a Babel shorthand (e.g. in the German language module); `?`, `!`, `:` and `;` are activated (e.g. by the French language module), and other characters may be activated, too. So `examplep` sets the `catcode` of all characters in the range 33...126 to other (12). This includes all ASCII punctuation, letter and digit characters.

**double carets** For example, the letter “J” can be input as its ASCII code in hex, prefixed by double carets: `^^4a`. However, in verbatim mode we want the 4 characters, not the letter J. There is no problem in inline verbatim mode, because `^` loses its special meaning once its `catcode` is changed. However, when the verbatim material is written to a file or to the terminal, T<sub>E</sub>X may change “unprintable” characters to escapes prefixed by `^^`. These changes must be reverted when reading the file back. See Subsection 7.2 for more.

**high characters** Input characters in the range 128...255 are usually activated by the `inputenc` package, and they know how to typeset themselves – so `examplep` leaves the `catcode` of such characters intact. However, in some modes, `examplep` has already changed all `catcodes` to 12 and 10 (with `\meaning`), so it has to change the `catcodes` of such high codes back to 13 (active).

## 3 Features of `examplep`

These are the most important, unique features:

- layout of side-by-side display may depend on maximum Source width
- automatic hyphenation of inline verbatim. The text is divided into words and punctuation symbols (based on `catcodes`). For words, the normal T<sub>E</sub>X hyphenation patterns apply, and it is allowed to break the line on both sides of a punctuation symbol.
- customizable isolation of page, section etc. numbers in the Sample and the host document with the `PexaMiniPage` environment
- besides the outer level, inline verbatim (when properly escaped inside `\Q` or `\÷`) works safely inside macro arguments, section titles, footnotes, table cells and index entries
- generated CD-files with automatic page and chapter number
- writing verbatim data to CD-files with a Perl script; exact, binary reproduction of verbatim text is guaranteed
- ability to write different material to Source, Sample and CD-files

Table 1: Contexts and features of inline verbatim commands

	outer	argument	tablular	elsewhere	escaped
<code>\verb</code>	+	- <sup>1</sup>	+	- <sup>1</sup>	no
<code>\PVerbOpt</code>	+	+ <sup>2</sup>	+	-	no
<code>÷</code>	+	+ <sup>2</sup>	-	+ <sup>2</sup>	no
<code>\÷</code>	+	+	+	+	yes
<code>\Q</code>	+	+	+	+	yes

<sup>1</sup>sometimes displays the proper error message

<sup>2</sup>inner mode only (spaces compressed or lost, % is comment etc.)

- the accents `\H` and `\.` work as expected with monospaced fonts in the OT1 encoding. By default, `\texttt{\H o}` produces `ø` in OT1 encoding, because such typewriter fonts (such as *cm<sub>tt</sub>10*) have those accents replaced by ASCII symbols. `examplep` solves the problem by getting the accent from the *cmr* font family.

Some other features:

- side-by-side display of the Source and the Sample
- between-word hyphenation of inline verbatim
- customizable left and right indentation of display verbatim
- specifying inline verbatim with nested braces (`\PVerb`, `\Q`) or terminating character (`\PVerb`, `÷`, `\÷`)
- automatic line breaks with hyphenation in display verbatim
- the discretionary hyphen (`\hyphenchar`) of verbatim text is different from the one in normal text
- line numbering in display verbatim
- writing to temporary files only if needed
- reading back contents of any file in display verbatim
- inline verbatim with a single character (`÷`) and its escaped version (`\÷` and `\Q`)
- automatic `\indent/\noindent`, based on empty line above `\begin{...}`
- ISO Latin accented input character support in all modes (also present in `\verb`)
- support for syntax highlighting with the `listings` package [1]
- emits a tab as eight spaces in normal mode, but tabs are supported properly with `ttlistings=yes` and `ttlistings=showtabs`
- simple side-by-side display emulation without temporary files, using `srcstyle=leftboth` or `srcstyle=leftleft`

### 3.1 Escaped mode of inline verbatim locations

For compatibility reasons, `examplep` doesn't change the original `\verb` and `\verb*` commands in any way, but defines its own commands: `\PVerb`, `\PVerbH`, `\PVerbInner`, `\PVerbOpt`, `\Q`, `÷` and `\÷`. Most of the `\PVerb...` commands are historical. For new documents, only the use of `\PVerbOpt`, `\Q`, `÷` and `\÷` is recommended. Some of these commands have to be activated with package load options:

`\usepackage[Q=yes,div=yes,bsdiv=yes]{examplep}`. The reason why `÷` was introduced is that it is a high Latin-1 (and Latin-2) character available on the Hungarian keyboard, which is usually not used in L<sup>A</sup>T<sub>E</sub>X documents (in fact, `$\div$` is used instead). Inline verbatim sources with such a character are compact, and they can contain all ASCII symbols.

`examplep` supports inline verbatim text at the outer level, inside macro arguments, in table cells and elsewhere (in section titles, in footnotes and in index entries), see also in Table 1. The reason why some of these cases are treated differently is that catcode changes must be timed correctly so that the proper catcodes are active by the time T<sub>E</sub>X reads the verbatim text from the input file for the first time. (Please note that section titles and index entries are also written to and read back from auxiliary files.)

This is quite hard to accomplish in several cases (because T<sub>E</sub>X’s mouth gathers macro arguments at high speed, a way before T<sub>E</sub>X’s stomach could change the catcodes), so `examplep` provides the commands `÷` and `\Q`, which do not change catcodes at all, so they work everywhere. Each special (say, not alphanumeric) character of the source text of these commands must be prefixed by a backslash, so T<sub>E</sub>X’s eyes will see it as a controls sequence token. The backslashes are retained when the construct is written to auxiliary files, but they get removed upon typesetting. Letters, when prefixed by a backslash, get special meaning, for example `\V` denotes a visible space. For example, The construct `\÷\\\}\div` is seen by T<sub>E</sub>X’s eyes as `\÷13\\13\}\div13`, and its gets typeset as `\}` (by running the command `\div`). Please note that the construct is properly nested, because all braces are inside control sequence names. The same result (`\}`) can be achieved with `\Q{\\}\}`. Both constructs are safe, because they can be freely moved to anywhere in the source file. However, for compatibility reasons, `\Q` is recommended, because its execution doesn’t rely on the current catcode of the terminating `÷` of `\div`. A more complicated example: ‘`\Q{\\\V X\ }`’ gets typeset as ‘`\_X`’. In escaped mode, `\V` denotes a visible, unbreakable space, `\S` and `\_` denote default space (affected by the `pverb-space=` option), `\B` allows a line break there with a discretionary hyphen (affected by the `pverb-linebreak=` option), and `\n` flushes left and starts a new line.

The `\Pverb` macros can detect whether they have been invoked from within a macro argument. If so, they do not insist on catcode changes, but they emit all the tokens that has been seen by T<sub>E</sub>X’s eyes. (Spaces are already compressed now, and everything after `%` is ignored etc., so this is not purely verbatim anymore.) However, this works only if the macro argument is properly nested with respect to braces, and it is delimited by braces (not a terminator character).

Please note that there might be problems with verbatim material in index entries processed by `makeindex` if characters `"`, `@`, `!` and `|` are not quoted properly with `"`. This is a generic `makeindex` issue. The quoting must be applied even inside verbatim material.

## 3.2 Horizontal alignment of the Source lines

The `verbatim` environment of standard L<sup>A</sup>T<sub>E</sub>X reads the whole verbatim text into a macro argument, thus limiting the length of the verbatim material to the available main memory. This is enough for about 3400 80-character lines. The `verbatim`, `moreverb`, `listings` and `examplep` packages parse the input line-by-line, so there is no such limit. However, with `examplep`, additional memory is required for aligned mode, which limits the maximum number of lines to about 2200 (32 pages) when the average line width is 80 characters. Please note that the maximums mentioned here may be lower if more packages are loaded; in another situation the maximum number of lines was 375. As a reference, a plain `\halign` with all lines having `(9999)\hfil\cr` only could accomodate about 5000 lines when no packages were loaded. The maximum memory can be increased by increasing the `extra_mem_bot` variable in `texmf.cnf` or in the environment.

There are two display modes used for display verbatim: paragraph and aligned (see the `\pexa@show@pars` and `\pexa@show@halign` macros, respectively, in the source). Aligned mode is used when multiple columns (such as line numbers and text) have to be aligned horizontally. Aligned mode uses the T<sub>E</sub>X `\halign` primitive to do the alignment, and this primitive reads the whole construct into memory before typesetting it (in order to be able to calculate the column widths). The other one, paragraph mode, is used when horizontal alignment is not needed and side-by-side display is not used. In paragraph mode, each line is typeset as a separate paragraph, so the length of the verbatim text is only limited by the available disk space to hold the resulting DVI file. `examplep` chooses the mode automatically: for side-by-side display it always chooses aligned mode (so it can measure the width of the Source before typesetting it), otherwise, if the `srcstyle=` makes it possible, it chooses paragraph mode, otherwise it chooses aligned mode. See Figure 1 for details about Source styles.

Please note that the Source styles `leftboth` and `leftbothnumcol` display each line twice, as Source and as Sample, too. This is different from regular side-by-side display, because lines of the Source and Sample here are forcibly aligned, and this solution doesn’t use a temporary file. The source style `leftleft` is similar, but it lets the author specify different Source and Sample for the same line (they should be separated by `&` in the source).

```

srcstyle=left PAF
9srcstyle=leftnumhang PAF
9srcstyle=leftnum PAF
 9srcstyle=leftnumcol AF when the last page number has 2 digits
                                srcstyle=center PAF

srcstyle=paralign           PF           with           srcstyle=right PAF
srcstyle=leftboth | srcstyle=leftboth A           source-par-align=justjust
 9srcstyle=leftbothnumcol | srcstyle=leftbothnumcol A
srcstyle=leftleft | anything A

```

*P*: works in paragraph mode  
*A*: works in aligned mode  
*F*: works when Source is read back from file

Figure 1: The effect of the `srcstyle=` option

<pre> 1\chapter{My chapter}\label{c} 2Welcome\footnote{to our isolated 3minipage environment}! 4\newpage 5\section{My section}\label{s} 6The chapter begins on page 7\pageref{c}.\par 8\begin{equation}a^2+b^2=c^2 9\end{equation} 10\par Indented. </pre>	<pre> <b>I My chapter</b> Welcome<sup>1</sup>! <hr/> <sup>1</sup>to our isolated minipage environment <hr/> 1 <hr/> <b>1 My section</b> The chapter begins on page 1.  <math display="block">a^2 + b^2 = c^2 \quad (1)</math>  Indented.  2 </pre>
--	--

Figure 2: Display verbatim isolation with the `PexaMinipage` environment

### 3.3 Display verbatim isolation

The `PexaMinipage` environment is provided, which is similar to the built-in `minipage` environment, but provides better isolation of the Sample from the container document, because it saves and restores section, page, equation (etc.) numbers and also marks (section titles in page headers). Labels (for `\label`, `\ref` etc.) are not isolated, because many packages use them in a non-standard way. See Figure 2 for an example.

The environment also cancels vertical skips (including `\belowdisplayskip`) at the bottom of its contents. For space conservation, `\abovedisplayskip` above the very first displayed equation is also canceled. To void this, put `\everydisplay{}` before the formula. The environment starts with `\noindent`, but subsequent paragraphs are indented.

### 3.4 Feature comparison

Although there are several L<sup>A</sup>T<sub>E</sub>X packages providing display and/or inline verbatim environments for L<sup>A</sup>T<sub>E</sub>X, `examplep` has some important unique features not found in other packages (see the beginning of this section for details). The author has tried the following packages before deciding to write `examplep`:

`verbatim` Although the `verbatim` environment is built-in into L<sup>A</sup>T<sub>E</sub>X, its most important limitation is that it eats up T<sub>E</sub>X memory when typesetting very long verbatim material (of several hundred or thousand lines). The `verbatim` package fixes this, and provides the `\verbatiminput` command (similar to the `\PexaShowSource` command of `examplep`) and the `comment` environment (similar to `PIgnore` in `examplep`).

**moreverb** This package extends the `verbatim` package with additional features: proper handling of tabulators (also accessible from `examplep` with the `listings` interface), line numbering (also available in `examplep` with much more customization), `verbatim` surrounded by a frame (this is not available in `examplep`, but it works with `listings`, with page breaks allowed), the `verbatimwrite` environment writes its contents to a file (similar to the `WFile` environment in `examplep`).

**sverb** It provides display `verbatim` with tabulators and long environments, and it can read and write text from files. It also has a side-by-side environment (`demo`) with fancy frames. The `verbfwr` package (part of the `examplep` distribution) was derived from parts of this package.

**syntax** This package is written by the author of `sverb`. It provides generic and customizable inline `verbatim` support and it also has powerful features to typeset BNF-like grammars and syntax diagrams. It is documented that no attempt is made to make the constructs work inside macro arguments or section titles.

**alltt** This standard L<sup>A</sup>T<sub>E</sub>X package defines the `alltt` environment in which the characters `\ { }` retain their original meaning, so it is possible to do some manual formatting in the `verbatim` text.

**fancyvrb** [5] This is extremely configurable `verbatim` package provides inline `verbatim` even in footnotes, display `verbatim` even with side-by-side, line numbers on any side, all kinds of fancy frames even with page breaks, text formatting and writing and reading from files; and very long display `verbatim` text. Options can be specified any time within the argument of the `\fvset` command. The original `verbatim` environment is not modified, but a new one, `Verbatim` is defined. Setting the background color is not possible.

This package is not actively developed. Version 2.7 (dated 2000/03/21) is part of `teTEX`. Oddly enough, the newest version on CTAN is 2.6, which a file timestamp in 2004, but it dated 1998/07/17).

**fvr-ex** This package is part of the `fancyvrb` distribution and uses the `fancyvrb` package. It provides a side-by-side display `verbatim` environment (`SideBySideExample`). A page break is not allowed in the Source after the Sample. The `xrightmargin` option has to be specified manually (e.g. `xrightmargin=3cm`). The first two characters of each line in the environment are ignored.

**ltxdoc** The most important features of `examplep` inspired by the L<sup>A</sup>T<sub>E</sub>X documentation package are display `verbatim` line numbering with `srcstyle=leftnum` and inline `verbatim` started with `÷`. The `ltxdoc` package typesets everything between two `|` characters as inline `verbatim`. This is not supported by `examplep` to avoid making an ASCII character active.

**listings** [1] The most important layout elements of this sophisticated, highly customizable, actively developed package missing from `examplep` are: background color, frames (with page breaks allowed), syntax highlighting and proper tabulator support. Except for the background color and frames, these features can be used from `examplep` with its interface to `listings`, see in Subsection 4.1. Use `\lstset{columns=fullflexible,language=C,backgroundcolor=\color{red},frame=trBL}` to try these features. `listings` also provides inline `verbatim` mode (with syntax highlighting), in the character-delimited argument of the command `\lstinline`; unfortunately, spaces at line breaks (with option `breaklines`) are rendered in an inconsistent way, and line breaks do not work well with background color.

`listings` has an important weak point: it cannot typeset ISO Latin accented characters with the `inputenc` package; the way described in the manual doesn't work as expected: it puts the accented characters to the wrong place in the line. This problem, however, is solved when `listings` is invoked from `examplep`. See more about `listings` in Subsection 4.1.

## 4 Customization

The operation of `examplep` can be customized with options as `<key>=<value>` pairs. Global options, which affect all subsequent commands within the current block, can be specified as package load options (`\usepackage[...]{examplep}`) or as argument of the `\PexaDefaults` command. L<sup>A</sup>T<sub>E</sub>X doesn't allow complicated option values (such as values containing some expandable macros, for example after `linenumberformat=`) to be specified in the `\usepackage` line – use `\PexaDefaults` in such cases. Many commands and environment accept local options, which affect only that construct.

All options have default values, which are indicated below right after the option name. If the option has a fixed set of possible values, all of them are mentioned, and they are prefixed by =. The defaults have been chosen so the `PSource` environment matches the builtin `verbatim` environment as closely as possible. Note that the original environment is not overridden before the `verbatimenv=yes` is specified.

`Q=unchanged` To enable `\Q`, use `=yes` instead of the default.

`abreak=unchanged` To enable `\abreak`, use `=yes` instead of the default.

`addvspace-bottom={\vskip\z@skip\addvspace}` Specifies the command to add vertical space below display `verbatim`. The default works fine, but e.g. packages maintaining the baseline grid might want to change it.

`addvspace-top=\addvspace` Command to add vertical space above display `verbatim`. The default works fine, but for example, packages maintaining the baseline grid might want to change it.

`allowbreak=yes` Use `=no` to disable page breaks in the Source of display `verbatim`.

`allowshrink=yes` The default will shrink the Sample horizontally if the Source is too wide. Use `=no` to disable this. Use `=force` to enable shrinking of Source, and with `srcstyle=leftleft` or `srcstyle=leftboth`, also enable shrinking of the Source if it is narrow.

`baseline-grid=no` Use `=yes` to adjust the height of the Sample to be an integer multiply of `\baselineskip` (with `yalign=u` and `yalign=v`).

`boxstyle=p` Controls how the Sample is boxed. Capital letters are not allowed for side-by-side display. By default (`=p`), a `PexaMinipage` environment is put inside a `\vtop`. Use `=h` to put a `\hbox` only, `=v` to put a `\vtop` only, `=V` to put a `\vbox` only, `=m` to put a `minipage` environment inside a `\vtop`, `=M` to put a `minipage` environment inside a `\vbox`, `=P` to put a `PexaMinipage` environment inside a `\vbox`, `=G` to add `\begingroup` and `\endgroup` only. The default is recommended for most cases, because the `\vtop` provides proper alignment with the Source, and the `PexaMinipage` environment provides isolation (of page and section numbers etc.) from the main document.

`bsdiv=unchanged` To enable `\÷`, use `=yes` instead of the default.

`div=unchanged` To enable `÷`, use `=yes` instead of the default.

`firstlinenum=1` Specifies the number of the first line of a numbered Source listing.  
Useful with `srcstyle=leftnumcol` and `srcstyle=leftnumhang`.

`linenumberformat={{...}}` Commands to display a line number and the separator in a numbered Source listing. See the default value in `examplep.sty`.

`linenumbersep={}` Commands to display the separator in a numbered Source listing.

`listings=no` Use `=yes` to display each Source line with the `listings` package. Specify options (to be executed in `\lstset`), separated by commas in the argument. Read more about the options in the documentation of the `listings` package. For example, `listings=yes` and `listings={}` uses `listings` with default options (no syntax highlighting), and `listings={language=C,showtabs}` enables syntax highlighting for C language and enables visible tabulators. See Subsection 4.1 for more information.

`listings-verbatimfont=pexavf` Set the font to be used in the Source when `listings=` is active.  
See `source-verbatimfont=` for the possible values.

`mp-equation-reset=yes` Use `=no` to make the main document and the Samples inside `PexaMinipage` share the same equation counter.

`mp-varioref-reset=no` Use `=yes` to make the internal counter `vrcont` of the `varioref` package to be reset for each Sample inside `PexaMinipage`. This option doesn't affect the final output, and `varioref` expects this counter not to be reset, so it is not recommended to change the default.

`noligs=some` By default, only those ligatures are disabled whose second character is one of ‘ ’ , - < > . Use `=kernel` to get the same effect, but using the L<sup>A</sup>T<sub>E</sub>X built-in `\@noligs`. Use `=most` to get all ligatures with either the first or the second character having code between 32 and 127 and catcode 12. Please note that ligatures in inline verbatim mode are disabled anyway, because `\allowbreak` is inserted between characters of catcode 12, depending on the value of `pverb-linebreak=`.

`pexaminiPAGE-setuphook={}` Extra commands to run when starting the PexaMiniPage environment, just after the environment has finished its own initialization.

`pverb-hash=full` Use `=half` to make `\PVerb` convert `##` to `#`. The command `\PVerbH` is the same as `\PVerb` but forces `=half`. This is required when `\PVerb` or `÷` is used inside a macro argument. For example, `\textit{÷#÷}` yields the error message *Illegal parameter number in definition of \reserved@a*, but `\textit{\PVerbH{##}}` works fine. The error message is a general L<sup>A</sup>T<sub>E</sub>X kernel limitation, for example `\textit{\@gobble{#}}` doesn't work either.

`pverb-hyphenchar=hyphen` By default, the minus character (ASCII code 45) is used to for automatic word hyphenation in inline verbatim. Use `=char'30` to have character with code 24; see also Subsection 7.6. Use `=none` to disable word hyphenation (by setting `\hyphenchar` to `-1`). Use `=unchanged` to get the hyphenchar from the font (the *cmtt* and *ectt* fonts have word hyphenation disabled). Please note that hyphenation around symbols is affected the `pverb-linebreak=` option, not this one.

`pverb-leftbreakmin=2` Specifies the minimum number of characters in inline verbatim after which it is allowed to break the line (with `pverb-linebreak=`). Values allowed are 0, 1 and 2, but 0 usually doesn't make sense.

`pverb-linebreak=char` By default, `\PexaAllowBreak` is inserted around symbols in inline verbatim, so a line break (with a discretionary hyphen affected by `pverb-linebreakchar=`) is allowed there. Use `=yes` to insert `\allowbreak` instead, which allows a line break without discretionaries. Use `=no` to disable line breaks around symbols in inline verbatim. The option `pverb-hyphenchar=` affects intra-word hyphenation in inline verbatim, not this one.

`pverb-linebreakchar={\!\not$}` Specifies the discretionary hyphen to be used in `\PexaAllowBreak`. See also `pverb-linebreak=`.

`pverb-space=invbreak` By default, spaces in inline verbatim are invisible, variable width (as allowed by the font, see also `pverb-stretchshrink=`) and breakable (i.e. a space can be replaced by a line break if necessary). Use `=invdisc` to get an invisible, variable width space which becomes visible (␣) when it is broken at the end of the line. Use `=invfixbreak` to get an invisible, fixed width and breakable space. Use `=invnobreak` to get an visible, variable width and unbreakable space. Use `=visnobreak` to get a visible, fixed width and unbreakable space. Use `=visbreak` to get a visible, fixed width space with line breaks allowed on both sides. Use `=invbreakleft` to get a visible, variable width space with infinite stretchability if the line is broken there (this may have strange effect on other line breaks in the paragraph, so please try to avoid it). The built-in `\verb*` command uses `source-space=visnobreak`.

`pverb-stretchshrink=yes` By default, spaces in inline verbatim are forced to be stretchable and shrinkable (by `\quad/9`). Use `=no` to disable stretchability and shrinkability. Use `=unchanged` to keep the settings in the font. Note that `\fontdimen3` and `\fontdimen4` are changed by this option, and the changes are local to inline verbatim mode.

`pverb-verbatimfont=pexavf` Set the font to be used in inline verbatim mode. See `source-verbatimfont=` for the possible values.

`samplewidth=.5\PexaWidth` Specifies the maximum width of the Sample in side-by-side display as a T<sub>E</sub>X dimension. The actual Sample can become actually narrower (see `allowshrink=`. The dimensions `\hsize`, `\linewidth` and `\PexaWidth` can be used. (Our L<sup>A</sup>T<sub>E</sub>X book used `samplewidth=.45\PexaWidth`.) `\leftskip` and `\rightskip` do not affect this option. `\hsize` can be used, which is the total width available (including the extra margins added by surrounding list environments), `\linewidth` is `\hsize` without the extra margins produced by lists, and `\PexaWidth` is the total width of the Source, the separator (see `vrule=`) and the Sample.



- `source-par-align=left` Specifies the alignment of Source lines when `srcstyle=paralign` is active. Use `=left` (default), `=right` or `=center` to specified flush-left, flush-right or centered alignment, respectively. Use `=justify` to have the last line flush-left and the previous line justified (please note that each Source line is mapped to a single paragraph, so the paragraph will have more than 1 line only if the source line is too long). Use `=justjust` to have all lines justified. Use `=unchanged` to keep the alignment of the enclosing block.
- `source-sepwidth=\tabcolsep` Specifies the horizontal distance between the Source and the Sample. See also `vrule=`.
- `source-space=invfixbreak` Specifies how to typeset spaces of the Source. See `pverb-space=` for the possible values. The built-in `verbatim*` environment uses `source-space=visnobreak`.
- `source-verbatimfont=pexavf` Sets the font to be used for the Source when `listings=` is not active (see also `listings-verbatimfont=` for `listings=`). Give `=ttfamily` to use `\ttfamily`, `=pexavf` to use `\pexa@@verbatimfont` (which defaults to `\verbatim@font`), `=latexvf` to use `\verbatim@font` (which defaults to `\normalfont\ttfamily`), `=unchanged` to keep the current font, or `=normalfont` to use `\normalfont`.
- `srcstyle=left` Specifies the horizontal alignment of the Source lines. See more in Subsection 3.2 and Figure 1.
- `ttlistings=` (no default) Shorthand of `listings-verbatimfont=ttfamily,listings=`.
- `url=unchanged` To enable `\url`, use `=yes` instead of the default. The `\url` will be defined as `\def\url{\PVerbOpt{}}`. This has the disadvantage that inside `\textit` etc. it cannot typeset URLs having a single `#` (see `pverb-hash=` for more), but the `url` package has the same limitation. A quick fix: use `\itshape` instead of `\textit` etc.
- `usewidth=skipwidth` Specifies which horizontal part of the main text should be used in a display `verbatim`. By default, left and right margins introduced by list environments (such as `itemize`) and `\leftskip` and `\rightskip` are respected. Use `=linewidth` to ignore `\leftskip` and `\rightskip` but respect list environments. Use `=hsize` to use the whole width of main text. Note that this option affects the calculation of `\PexaWidth`.
- `vxtrabotdepth=\z@` Dimension to add to the depth of the display `verbatim` with `yalign=v`. The default works fine, but for example, packages maintaining the baseline grid might want to change it for each instance.
- `vxtravskip=\z@` Amount of vertical space to be added above display `verbantim` with `yalign=v`. The default works fine, but for example, packages maintaining the baseline grid might want to change it for each instance.
- `vsmallht=1pt` Specifies Sample height threshold for `yalign=v`. If the Sample is lower than this (or sample a higher than the 1st line of the Source plus `\vxtravskip` – typical for `\includegraphics`), its top will be aligned to the top of the Source, otherwise its top baseline (with `\vtop`) will be aligned to the top baseline of the Source.
- `xalign=l` Specifies horizontal alignment (`=l` for left, `=r` for right) of the Sample box (and the separator) within its allocated width for side-by-side display. Please note that `=r` works only with `boxstyle=h`, because all other box sizes use their full allocated width.
- `xindent=deeppe` Specifies additional horizontal indentation in display `verbatim` mode. Use `=none` to get no extra indentation. Use `=narrower` to get `\narrower` (both `\leftskip` and `\rightskip` are decreased by `\parindent`). Use `=deeper` to move one level deeper in the list environment hierarchy and get that indentation. Use `=deeppe` (default) to move one level deeper, but don't change indentation (this is useful with `yindent=deeper` – otherwise it is equivalent to `=none`). Use `=deepright` to set both left and right indentation from the left indentation of `=deeper`.
- `yalign=u` Specifies vertical alignment in side-by-side display. By default, the top of the bounding boxes of the Source and the Sample is aligned, which looks nice if the Sample is an image, but doesn't align properly if the Sample is text with a font of similar size to the Source. Use `=b` to align the topmost baselines of

the Sample and the Source. This looks nice if the Sample is text, but it is ugly if the Sample is an image higher than `\baselineskip`. The use of `=v` is recommended, which decides between `=b` and `=u` based on the height of the Sample (see `vsmallht=` for the details).

`yindent=deeper` Specifies the vertical space separating display verbatim from the surrounding text. Use `=none` to have no extra vertical space, the display verbatim appears to be a new paragraph as far as `\baselineskip` and `\vskips` are concerned. Use `=deeper` (default) to move one level deeper in the list environment hierarchy (and use the `\parsep` and `\partopsep` etc. specified there). It is recommended to have `yindent=deeper` and `xindent=deeper` together, so there is no extra horizontal indentation.

`verbatimenv=unchanged` Use `=yes` to change the implementation of the `verbatim` and `verbatim*` environments to use the `PSource` environment.

`vrule=rule` By default, the Source and the Sample are separated with a vertical rule of width `\arrayrulewidth` in the middle of a horizontal space specified by `source-sepwidth=`. Use `=skip` to omit the rule but keep the space. Use `=none` to have no separator at all.

The other packages (`codep` and `verbfwr`) shipped with `examplep` do not have load options.

## 4.1 Interface to the listings package

The `listings` package [1] provides advanced typographic for display verbatim, including proper typesetting of tabulators and syntax highlighting for more than a hundred languages. `examplep` doesn't try to reimplement these features, but it supports calling the `listings` package to typeset the Source lines in display verbatim. The surroundings (line numbers, vertical separation, horizontal margins and the Sample) are not effected, only the Source line contents are passed to `listings`. This implies that the border and the background color support provided by the `listings` package doesn't work with `examplep`. To use the interface, the `listings` package must be loaded, and either the `listings=` or the `ttl listings=` options of `examplep` has to be active when the Source is typeset. Additional options can be specified to `listings` in the argument of `\lstset` at any time. The interface has been tested with the `listings` package dated 2000/08/23 and 2004/09/07.

`examplep` treats tabulators (ASCII code 9) as 8 spaces. This is acceptable at the beginning of the line, but it may be incorrect elsewhere. To get tabs right, specify the `ttl listings=yes`, or, to be more precise, the `ttl listings={tabsize=8}` option to `examplep`. It is also possible to have visible tabulators: specify, for example, `ttl listings={tabsize=8,showtabs}`. In our tests `listings` failed to detect the width of a character of a fixed width font, so `examplep` enforces character width using the natural width of the space each time it calls `listings`. This workaround made the `showtabs` `listings` option work properly. See the documentation of the `listings` package for options that affect the typesetting of Source line contents. See an example of using `listings` from `examplep` on page 16.

`listings` supports fixed width characters with a variable with fonts. However, this support seems to be broken when used with `examplep`, so the `columns=fullflexible` `listings` option is enforced so proportional fonts will look proportional. Although the `listings` package claims that it has accented letter support, this didn't work well with the single-character accented letters input using the `inputenc` package (those characters were positioned to a wrong place inside the line, possibly because `listings` has failed to recognise that `\lst@UseLostSpace\lst@PrintToken` has to be inserted in front of the accented character into its internal token list). `examplep` contains a work-around to this problem, with the following limitations: multibyte input encodings such as UTF-8 are not supported (will print strange error message); accented characters may not be part of keyword names in syntax highlighting; accented characters are shown as `^^` hex escapes in aligned mode (see in Subsection 3.2), so they don't work with `\PexaShowBoth`.

`listings`, when called from `examplep`, failed to break ligatures such as `'?` and `<<`. This has the side effect that guillemots would be typeset instead of bitwise right shift in C language sources. `examplep` modifies the `\lst@FillOutputBox@` macro so it will add a `\relax` between each character displayed – so all ligatures are broken. (This approach is quite different from the way L<sup>A</sup>T<sub>E</sub>X disables a few ligatures with `\@noligs`; `\pexa@noligs` is similar to `\@noligs` in this respect.)

It is possible to customize the `listings` package so it typesets some strings differently. For example, with the `literate={<=}{{\leq$}}1` `listings` option, all occurrences of `<=` (even those inside strings of the target programming language) are typeset as `≤`. There are no problems when using this feature from within `examplep`. It is also possible for strings and comments in the syntax-highlighted Source to span multiple lines – `listings` takes care to remember its internal state between lines.

## 5 Commands and environments

The arguments between brackets ( [ and ] ) are optional: either the the argument and the brackets are all missing all all present. The arguments named “options” is a comma-separated list of local customization options, defined in Section 4. The {+ notation in front of an argument means that the argument can be delimited by braces (thus it must be properly nested), or with any symbol in `\dospecials` ( `\ $ & # ^ _ % ~` ) or in `\pexa@cverb@donormals` ( `' ! @ * - + = | : ; ' " , . / ? < > ( ) [ ]` ).

`\PVerb[⟨options⟩]{+⟨verbatimtext⟩}` Typesets its argument in inline verbatim mode. Similar to the L<sup>A</sup>T<sub>E</sub>X `\verb` macro, but respects the options. The use of [ ] is recommended instead of omitting the options altogether, because [ ] will ensure that the proper catcode changes are in effect even for the first verbatim character. This command is robust.

`\PVerbH{+⟨verbatimtext⟩}` Shorthand for `\PVerb[pverb-hash=half]` (extra options cannot be specified). This command is robust.

`\PVerbInner\PVerb...` Forces the `\PVerb...` command immediately following it to work in inner mode, thus compressing spaces, respecting comment characters etc. Because of how T<sub>E</sub>X works, it is impossible to go the other way round, and force outer mode, because it is too late change catcodes – the argument has already been tokenized in inner mode. This command is robust.

`\PVerbOpt{⟨options⟩}{+⟨verbatimtext⟩}` Equivalent to `\PVerb`, but uses a different syntax. For example, `\item[\PVerb[pverb-space=visbreak]{xy}]` doesn't work because of the nested [ . Use this instead: `\item[\PVerbOpt{pverb-space=visbreak}{xy}]`, or `\item[{\PVerb[pverb-space=visbreak]{xy}}]`. This command is robust.

`\Q{⟨verbatimtext⟩}` Similar to `\PVerb`, but its argument must be escaped (see in Subsection 3.1), and it can be used in section titles etc. Must be enabled with `Q=yes`. This command is robust.

`÷⟨verbatimtext⟩÷` Similar to `\PVerb`, but it can be used in section titles etc. (but not int `tabular`) (see in Subsection 3.1). Must be enabled with `div=yes`. This command is robust.

`\÷⟨verbatimtext⟩÷` Equivalent to `\Q`, but the argument delimiter is different. Similar to `\PVerb`, but its argument must be escaped (see in Subsection 3.1), and it can be used in section titles etc. Must be enabled with `bsdiv=yes`. This command is robust.

`\url{⟨url⟩}` Must be enabled with `url=yes`. This command is robust.

`\begin{WFile}{⟨filename⟩}` (defined in the `verbfwr` package) Writes its contents verbatim to the specified file. T<sub>E</sub>X `.tcx` and line ending transformations apply, so it is possible that accented letters will be converted to `^^hex` according to the input encoding.

`\begin{WAux}` (defined in the `verbfwr` package) Writes its contents verbatim into the current `.aux` file. T<sub>E</sub>X `.tcx` and line ending transformations apply, so it is possible that accented letters will be converted to `^^hex` according to the input encoding.

`\begin{PWSource}[⟨options⟩]` Comination of `\begin{WSource}` and `\PexaShowSource`. It is recommended to have a [ ] even if there are no options, so the very first token of the contents will be read with proper catcodes.

`\begin{WBoth}` Writes its contents to the Source and the Sample temporary file. It is a combination of `WSample` and `WSample`.

`\begin{WSample}` Writes its contents to the Sample temporary file (`pexa-sam.tex`), to be typeset by a subsequent `\PexaShowSample` or `\PexaShowBoth`. The line must end at `\end{WSample}` because of technical reasons.

`\begin{WSource}` Writes its contents to the Source temporary file (`pexa-src.tex`), to be typeset by a subsequent `\PexaShowSource` or `\PexaShowBoth`. It is similar to `\begin{verbwrite}` in the `sverb` package and the `\begin{filecontents}` L<sup>A</sup>T<sub>E</sub>X built-in environment. The line must end at `\end{WSource}` because of technical reasons.

- `\begin{PIgnore}` Ignores everything up to `\end{PIgnore}`. The environment closer must be at the end of its line. Similar to the `comment` environment in some other packages.
- `\begin{PSource}[options]` Typesets its contents in display verbatim. Similar to the L<sup>A</sup>T<sub>E</sub>X `\begin{verbatim}` environment, but respects the customization options. It is recommended to have a `[]` even if there are no options, so the very first token of the contents will be read with proper catcodes. This environment is similar to `PWSource`, but it doesn't create a temporary file, so it is faster, `srcstyle=leftboth` (etc.) can be used, and there is no ambiguity between `~e1` and `â` (etc., see more in Subsection 7.2). Page breaks are allowed between each Source line. (The implementation of this environment is fairly complex compared to `PWSource`.)
- `\begin{verbatim}` Equivalent to `\begin{PSource}[]`. Must be enabled with `verbatimenv=yes`.
- `\begin{verbatim*}` Equivalent to `\begin{PSource}[source-space=visbreak]`. Must be enabled with `verbatimenv=yes`.
- `\begin{PexaMinipage}[vbox-type]{width}` Similar to the L<sup>A</sup>T<sub>E</sub>X `minipage` environment (and accepts the same arguments), but isolates (concerning section numbers etc.) of its contents from the main document more thoroughly. See Subsection 3.3 for details of isolation.
- `\PexaShowBoth{options}` Typesets the Source and the Sample side-by-side in display verbatim mode. The Source comes from the temporary file written by the last `WSource` or `WBoth` environment, and the Sample comes from the temporary file written by the last `WSample` or `WBoth` environment. By default, a vertical separator line is drawn between the Source and the Sample, and page breaks are allowed in the Source after the end of Sample. It can be called multiple times with different options for the same file.
- `\PexaShowSample{options}` Typesets the Sample (written by the last `WSample` or `WBoth` environment) in display mode. It can be called multiple times with different options for the same file.
- `\PexaShowSource{options}` Equivalent to `\PexaInputSource` with the file written by the last `WSource` or `WBoth` environment. It can be called multiple times with different options for the same file.
- `\PexaInputSource{filename}{options}` Typesets the contents of the specified file as Source in display verbatim mode.
- `\begin{code}` (defined in the `codep` package) Typesets its contents side-by-side and also marks its contents to be dumped to the CD. By default, each line is emitted to all three streams, but lines with special prefixes will go into the Source, Sample or CD-file stream only. See Section 6 for details.
- `\PexaAllowBreak` Allows a line break here with a discretionary specified in the option `pverb-linebreakchar=` inserted.
- `\abreak` A robust command which inserts `\PexaAllowBreak` when the font `{\ttdefault}{m}{n}` is active; inserts `\allowbreak` otherwise. Must be enabled with `abreak=yes`.

## 6 Writing examples with the `codep` package

Textbooks and manuals tend to have many display verbatim examples. The examples are usually code snippets which can be further processed by a compiler or another program. Sometimes minor modifications, such as adding the proper header or trailer, are necessary before the code snippet can be processed. It is customary to put all code snippets in the book onto the CD accompanying the book. The `code` environment of the `codep` package (part of the `examplep` distribution) generates CD-files automatically.

Three streams are generated from the contents of each `code` environment: the Source, the Sample and the CD-file streams. Most parts of these streams are identical. The Sample usually differs from the Source because the code snippet has to be typeset specially in the book (for example, `\includegraphics` has to be used to typeset an EPS file whose Source is displayed). The CD-file differs from Source because additional header and footer may be required (such as `\begin{document}` etc.), which are omitted from the book to conserve space.

The `code` environment reads the code snippet line-by-line. The type of the line is specified in first two characters. Lines having the default type are written to all 3 streams, and special line types exist to write to a specific stream only. The `code` environment writes the Source and Sample streams to temporary files, and

upon the end of the environment, it calls `\PexaShowBoth` (or `\PexaShowSource`, if the Sample stream is empty) to typeset the example. The CD-file stream is not written to a file by T<sub>E</sub>X, but the file name and starting line number of the `code` environment is reported in the `.aux` file. A Perl script (`wrfiles.pl`, part of the `examplep` distribution) has to be called later to the actual generation of CD-files. It will examine the `.aux` files, extract the CD-file stream from the `.tex` files, and dump these streams to individual files in the `CDfiles` directory. The file names can be specified in the `code` environment, and the environment can generate file names based on chapter and page numbers (so the reader will know from the file name where to read more about the example). The same file name is never generated again.

The `code` package was used in our recent L<sup>A</sup>T<sub>E</sub>X textbook [4] to typeset its examples. Most of the examples were written in L<sup>A</sup>T<sub>E</sub>X, but many of them were METAPOST sources, and some of them were others (e.g. configuration files, shell scripts or EPS files). Because of the huge amount of L<sup>A</sup>T<sub>E</sub>X examples, special features were added to make them easy and convenient to input for the author. For example,

```
\begin{code}
t \usepackage{url}
  URL:
  \\url{http://foo.org/~user/}
\end{code}
```

is displayed as (depending on the `examplep` options)

```
1%~\usepackage{url}      | URL:
2URL:                    | http://foo.org/~user/
3\\url{http://foo.org/~user/}
```

As seen above, examples are quite convenient to input, and `examplep` takes care of typesetting side-by-side, determining width of the Source, allowing page breaks, putting margins and `\vskips` right, adding the rule to separate the Source and the Sample, adding line numbers, generating file name for CD-file and writing the CD-file with header and footer.

With `codep` it is easy to fulfill the following quality criterias: the Sample must be consistent with the Source (i.e. if the Source is changed during editing to book, the Sample should change automatically); the CD-file must be consistent with the Source; the CD-file must be directly compilable with L<sup>A</sup>T<sub>E</sub>X (so a header and a footer have to be added). When the deadline of finishing the book approaches, there might not be enough time left to ensure these manually, so a package such as `codep` is very useful in this situation.

## 6.1 Example files on the CD

The following CD-file is generated from the code snippet above:

```
\documentclass{article}

\usepackage[latin2]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[magyar]{babel}
\usepackage{url}

\begin{document}

URL:
\\url{http://foo.org/~user/}

\end{document}
```

The `CodeDefaultD`, `CodeDefaultL`, `CodeDefaultB` and `CodeDefaultE` environments can be used in the preamble to customize the default header and footer generated into the CD-file. For example:

```
\begin{CodeDefaultD}
\documentclass[10pt]{article}
\end{CodeDefaultD}
\begin{CodeDefaultL}
```

```
\usepackage[latin2]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[english]{babel}
\end{CodeDefaultL}
```

Although T<sub>E</sub>X is able to write to external files with `\textsfrite`, there were several reasons for using an external program (a Perl script) to extract the source snippets from the document sources:

- with `\write` the file always ends at end-of-line
- `\write` forces `.tex` if no extension is specified
- `\write` removes whitespace from end-of-line
- `\write` translates accented letters to hat-escapes (e.g. `á` to `^^e1`) unless compiled with `latex -translate-file cp8bit.tcx` (`-translate-file il2-t1.tcx` makes `ö` in DVI incorrect). There is the same problem when emitting UTF-8 text.
- it is impossible to distinguish missing files from empty files, so accidental file overwrites are hard to prevent
- it is too late to verbatize if the verbatim text is inside braced macro arguments

The only limitations of this solution are: is not possible to `\input` or `\include` a subfile, and then use the `code` environment in the referrer file; the subfile has to be included with `\include{...}` or `\input{...}` (with braces); and the subfile must have extension `.tex`. The first one is usually not a problem, since referrer files themselves do not typeset text, they only include subfiles. See Subsection 7.7 for implementation details.

## 6.2 `\begin{code}` invocation

The input syntax of the `code` environment has been designed so that typing the most common examples (short L<sup>A</sup>T<sub>E</sub>X code snippets) is simple and straightforward, but the author can have full control over all three streams if he wants to. The contents of the environment is divided into lines. The first two characters of each line specify the line type and the rest is the line data. The first character of the line type is usually a lowercase ASCII letter or a punctuation symbol. Line types belong to classes, which are denoted by capital ASCII letters. The order of the classes in the environment is significant, but the order of the individual types or lines within the class is irrelevant. Some classes have default lines, which are used only if the class is omitted from the environment. The default lines make it possible to have default CD-file header and trailer. The classes, in proper order with allowed types in parentheses, are:

**F** (**f**, **f!**, **v**, **v!**) specify the file name.

**D** (**d**) the `\documentclass` line, default uses `article`

**L** (**l**) the preamble specific to the natural language, defaults for Hungarian `babel`, Latin-2 `inputenc`, T1 `fontenc`. Use the `CodeDefaultL` environment to override.

**P** (**p**≡**0**, **t**) the preamble with the `\usepackage` lines

**B** (**b**) `\begin{document}`

**C** (<≡**c**, >≡**o**, \_≡**2**, **w**, **s**, **x**, **%**) the document contents

**E** (**e**) `\end{document}`

The meaning of the complicated types are:

**f** Accepts a file name with extension. The use of `_` in the name is not recommended. The extension (e.g. `.tex`) is mandatory. The chapter and page numbers will be prepended to the file name (only the page number for document classes without chapters), for example `f foo.mp` may become `2_63_foo.mp` in chapter 2, on page 63.

**v** Like **f**, but removes the default lines from classes **D**, **L**, **P**, **B** and **E**. This is ideal for emitting non-L<sup>A</sup>T<sub>E</sub>X examples.

**f!** Like **f**, but don't prepend numbers to the file name.

**v!** Like **v**, but don't prepend numbers to the file name.

**p≡0** Writes only to the preamble of the CD-file.

**t** Writes to CD-file, appends line prefixed by  $\%^$  to Source. Useful to indicate in the book that a package is needed. Example: `t\usepackage{url}`.

**<≡c** Writes to Source and CD-file.

**>** Writes only to Sample.

**x** Writes to Sample and CD-file.

**⌋≡2** Writes to Source, CD-file and Sample.

**w** Writes only to CD-file.

**s** Writes only to Source.

**%** Comment, ignored.

The `code` environment omits the Sample part from the book if the Sample is empty, and it omits the whole display verbatim environment (but still writes to CD-files) if both the Sample and Source are empty.

### 6.3 An example with METAPOST code

If the `eampost` package is also loaded, the following code can be used to typeset a simple, syntax-highlighted METAPOST source and its output:

```
{\PexaDefaults{listings={language=metapost}}\begin{code}
v house.mp
> \begin{EempDef}{house.1}{\}{}
w beginfig(1)
  u:=18bp; picture V; V:=image(
    draw unitsquare scaled u xscaled 2;
    fill (0,u)--(2u,u)--(u,1.5u)--cycle
      withcolor red);
  draw V rotated 10;
  draw V shifted (3u,0);
w endfig; end
> \end{EempDef}
> \leavevmode\EempUseFig{house.1}{0}{0}
% ^^^ Dat: \leavevmode to get the Overfull \hbox warning
\end{code}
} % Dat: nothing allowed after \end{code} in its line
```

If `eampost` is not loaded, the following code should be used instead:

```
{\PexaDefaults{listings={language=metapost}}\begin{code}
v house.mp
> \begin{WFile}{house.mp}
x beginfig(2)
  u:=18bp; picture V; V:=image(
    draw unitsquare scaled u xscaled 2;
    fill (0,u)--(2u,u)--(u,1.5u)--cycle
      withcolor red);
  draw V rotated 10;
  draw V shifted (3u,0);
x endfig; end
> \end{WFile}
```

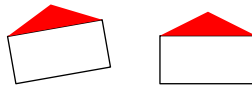
```
> \leavevmode\includemps{house.2}
\end{code}
}
```

The `\includemps` command should be defined in the preamble as:

```
\usepackage{graphicx}
\DeclareGraphicsRule{*}{mps}{*}{}
\makeatletter
\@ifundefined{Ginclude@eps}{\def\Ginclude@mps{\Ginclude@eps}}
\def\includemps{\@ifnextchar[\includempsb{\includempsb[]}}
\def\includempsb[#1]#2{\includempsc{#1}#2\@nil}
\def\includempsc#1#2.#3\@nil{%
  \IfFileExists{#2.#3}{\includegraphics[#1]{#2.#3}}{
  \GenericWarning{}{Please run: mpost #2~^J@gobble}}
\makeatother
```

This should work with both `dvips` and `pdflatex`. The typeset output looks like this:

```
1u:=18bp; picture V; V:=image(
2 draw unitsquare scaled u xscaled 2;
3 fill (0,u)--(2u,u)--(u,1.5u)--cycle
4 withcolor red);
5draw V rotated 10;
6draw V shifted (3u,0);
```



## 7 Some implementation details

### 7.1 Starting from poor man’s inline verbatim

The following macro, derived from a macro in the `.dtx` documentation of David Kastrup’s `binhex` package [2], typesets its argument in inline verbatim mode:

```
{\catcode\string'>12 \gdef\stripprefix#1>{}}
\def\verbatimize#1{{\ttfamily
  \toks0{#1}\edef\next{\the\toks0}% Dat: make # OK
  \fontdimen2\font=0pt % Dat: hide spaces
  \expandafter\stripprefix\meaning\next
  \unskip % Dat: strip final space, possibly after command
  \fontdimen2\font=\dimen0}}% Dat: reset global change
```

This demonstration show how useful the T<sub>E</sub>X primitives `\string` and `\meaning` are. Both of them convert tokens to characters with catcode 12 (other) or 10 (space). Token lists with spaces are hard to post-process by T<sub>E</sub>X macros, because T<sub>E</sub>X macro expansion ignores spaces before undelimited macro arguments. But it is possible to write a macro which converts spaces to anything with catcode 12, for example the `\sca` macro below does this:

```
\begingroup\catcode\string'‘12 \lccode'‘‘%\lowercase{\endgroup
  \def\scc#1 {\ifx\hfuzz#1\else#1'\expandafter\scc\fi}}
\def\scb#1#2{\scc#2\hfuzz#1} \def\sca{\scb{ }}
% try with: \message{\sca{foo bar }}
```

It is possible to change `%` in the definition above to anything, including a space: the replacement character will have catcode 12. After such a conversion, the text to be emitted can be easily processed to add T<sub>E</sub>X macros, change catcodes back to 13 for ISO Latin high accented characters, replace spaces with appropriate constructs, insert `\allowbreak` to the right places to enable line breaks etc. The `\PVerb` macro, when invoked in inner mode (i.e. read inside a macro argument) works this way, and respects the options specified by the author.



## 7.2 Hex escapes with output translation

The T<sub>E</sub>X primitives `\write`, `\message` and `\errmessage` may escape some characters when printing them. By default, T<sub>E</sub>X changes the code ranges 0–31 and 127–255 (the codes outside the printable ASCII range), escaping such codes with a `^^`: for example, the tabulator (code 9) becomes `^^I`, and characters having a high code in the font (not the input) encoding are dumped in hexadecimal, for example `š` (having code 174 in T1 encoding) becomes `Ž`. (This behaviour depends on the default `.tcx` file the T<sub>E</sub>X distribution uses. No translation occurs with `cp8bit.tcx`. To spot the difference, run `tex -translate-file cp8bit "\message{^^I^^1fá}\end`, and then change `cp8bit` to `.missing.`, and run again.) The transformation is lossy: both `\message{š}` and `\message{\string^^ae}` yield the same result: `^^ae`. Escaping the caret as `^^5e` doesn't help either, because the T<sub>E</sub>X unescapes carets recursively when reading back the written file. Since ISO Latin accented characters are more often needed in verbatim environments than double carets, `examplep` does the necessary unescaping when it reads the file back. The back-transformation doesn't work with UTF-8, because the 2nd byte is not decoded by the time the first one is being executed. The unescaping would be done by T<sub>E</sub>X itself if the caret had its original catcode 7, but that would imply that the non-escaping, verbatim carets wouldn't work.

The unescaping is implemented in a straightforward, but ugly way in the `\pexa@dohex@low...` macros. The caret escapes are parsed in a huge `\if\else\if` construct nested in 40 levels, and once the hexadecimal code is available and converted to upper case, the `\lccode'+="/code)\lowercase{+}` construct is used to insert the appropriate character with catcode 12 (~ is used instead of + to get an active character, catcode 13). The construct is not expandable, but it works because it is used for typesetting. The caret is made active and defined to execute `\pexa@dohex`, so each caret in the file will get unescaped.

## 7.3 Disabling ligatures

The only way to disable a ligature in T<sub>E</sub>X is to insert a nonexpandable tokens into the input stream between the characters forming the ligature. For example, `f{i}` or `f\relax i` can be used to get “fi” instead of “fi”. The most important ligatures (in addition to ligature letters) to be disabled in verbatim mode are: `<< >> ? ' ! ' , , ' ' ' --` and `---`. This can be accomplished by inserting a `\relax` token in front of each `' ' , - <` and `>`. The `\pexa@noligs@some` command of `examplep` does exactly this, for example, it defines `{\lccode'~'<13 \gdef~{\relax\string~}}`. The definition slightly different from the one of the `\@noligs` command in the L<sup>A</sup>T<sub>E</sub>X kernel: `\def<{\leavevmode\kern\z@\char'\<}`; but the effect is the same. The `\pexa@noligs@most` command, on the other hand, makes all characters with category code 12 in the range 32...127 active, and adds `\relax` to both sides. This change doesn't affect ASCII or accented letters, but usually there are no ligatures with letters in typewriter fonts. See also the `noligs=` load option.

## 7.4 Detecting inner/outer brace in inline verbatim mode

The `\PVerb` commands work differently based on whether they are inside a macro argument or not. More precisely, they detect whether they are able to change the catcode of the following token. If so, they are in outer mode (i.e. outside a macro argument), so they change all the other catcodes as well, so consecutive spaces and comment characters will be included in verbatim, too. Otherwise, they are in inner mode, their argument is already read and tokenized by T<sub>E</sub>X's eyes, so changing catcodes is pointless.

The auto-detection works this way: the catcode of all the special characters (as enumerated in `\dospecials`; including braces) is changed to 3 (math-shift). Then the next token is read into `\reserved@a` with `\afterassignment\pexa@cverb@gottoken\let\reserved@a=.` No tokens are ignored this way, not even spaces. The `\pexa@cverb@gottoken` macro then examines the catcode of the character in `\reserved@a`, and if it is 3, it continues in outer mode, otherwise it continues in inner mode. In inner mode, the next token is forced to be an open-brace, because verbatim material with braces not nested cannot be read into inner mode anyway (T<sub>E</sub>X would print an error message when it is trying to find the end of the macro argument containing the `\PVerb` construct).

Another common trick is used when parsing the argument in outer mode when it is delimited by braces. Normally a T<sub>E</sub>X macro expansion (using the definition `\def\pexa@cverb@outerc#1{...}`) can read an argument that is in braces, but in our case the very first opening brace has been already read (by `\let` above), so we have to insert it back: `\catcode'\{1 \catcode'\}2 \expandafter\pexa@cverb@outerc\expandafter{\iffalse}\fi`. The `\iffalse}\fi` here is needed for making the definition properly nested.

## 7.5 Inline verbatim in section titles

The T<sub>E</sub>X command `\write`, `\message` and `\edef` fully expand their arguments, and similar expansion is enforced by the `\markboth` built-in L<sup>A</sup>T<sub>E</sub>X macro for section titles and page headings. Therefore macros in section titles have to be protected so their expansion is delayed until the section title is typeset. L<sup>A</sup>T<sub>E</sub>X offers `\protect` for this: if the macro control sequence is preceded by `\protect`, its expansion is properly delayed; the expansion of the argument has to be delayed manually in a similar way. Some macros have `\protection` included; they are called “robust”. If the definition a macro starts with `\DeclareRobustCommand` instead of `\newcommand`, the macro is defined to be robust (and its body can be retrieved by looking at the control sequence with a space added, e.g. `\expandafter\show\curname_sqrt_\endcurname`).

`\protect` can have three definitions depending on what time it is processed: it is `\string` in a `\typeout` or a L<sup>A</sup>T<sub>E</sub>X error or warning message (try `\typeout{\meaning\protect}`); it is `\noexpand\protect\noexpand` when `\write`ing to a file (most commonly the `.aux` file); otherwise it is just `\relax` ( $\equiv$  `\@typeset@protect`; try `\pagestyle{headings}\section{\meaning\protect}` and spot the difference between the main text, the section title and the `.aux` file).

The `\div` and `\Q` inline verbatim commands are made robust, so they can be used in macro arguments. In fact, they are extra-robust, since they take care of protecting their arguments when being written to a file by L<sup>A</sup>T<sub>E</sub>X. Protecting here means adding `\noexpand` in front of each token in the argument. The token parsing is easy since the argument – by the nature of these commands – may not contain braces or spaces. The implementation looks like this.

```
\long\def\div#1\div{\Q{#1}}
\long\def\Q{\ifx\protect\@typeset@protect\expandafter\@gobble\fi
  \@thirdofthree\@firstoftwo\displayit\protectit}
\def\displayit#1{...}
\def\protectit#1{\noexpand\div\protectnext#1\div}
\long\def\protectnext#1{\noexpand#1%
  \ifx#1\div\else\expandafter\protectnext\fi}
```

The first trick is in the body of `\Q`: the argument is passed to either `\displayit` or `\protectit`, depending on the current value of `\protect`. If the condition is true, `\@gobble` is called, which removes `\@thirdofthree`, so `\@firstoftwo` will choose `\displayit` (otherwise, `\@thirdofthree` chooses `\protectit`). The second, more classical trick is the rôle of `\expandafter` in the definition of `\protectnext`: it makes the `\fi` token disappear, so the tail-recursive call to `\protectnext` will grab the next token into `#1` instead of `\fi` itself.

## 7.6 Special hyphenchar in inline verbatim

When inline verbatim is hyphenated, care has to be taken to make the discretionary hyphen different from a regular, verbatim hyphen. (There is a similar problem with spaces disappearing when the line is broken; to avoid this, try setting the option `pverb-space=visbreak` or `pverb-space=invdisc`.) T<sub>E</sub>X auto-hyphenation takes the discretionary hyphen from the `\hyphenchar` of the font. So the solution is adding a new glyph to the verbatim font, changing the font encoding vector to include the glyph, and then setting `\hyphenchar`.

We have chosen character position 24 (per-thousand sign) of the T1 encoding to be replaced by a soft hyphen (̣), which is deliberately narrower than all the other characters, so the reader immediately sees its function. For example:

“foo-bar”. We have drawn the glyph in Fontforge, saved the data to PFB, converted it to human-readable format with the command `type1fix.pl shorthyp.pfb gsx: shorthyp.gsx`, extracted the human readable glyph definition (`/shorthyp { ... }`) from the output. We have changed the `/FontName` and injected the glyph to original font with the following command:

```
perl -x -S type1fix.pl --set-leniv=0 --dump-spaces=no --pack \
  --dump-bars --dump-stde --dump-ends=no --debug-warnings \
  --chk-insize=no --set-uniqueid=random --set-fontname=t1xtts \
  --set-glyph="/shorthyp { 50 354 hsbw 315 vmoveto -17 vlineto 0
-8 0 -8 6 -4 rrcurveto 4 -6 8 0 5 0 rrcurveto 195 hlineto -124
vlineto -11 0 -21 15 vhcuroveto 2 0 3 1 2 1 rrcurveto 10 2 1 12
0 12 rrcurveto 0 8 -1 8 0 5 rrcurveto 130 vlineto 0 5 1 7 0 7
rrcurveto 0 12 -2 11 -11 3 rrcurveto -5 1 -6 0 -5 0 rrcurveto
-12 0 -12 -1 -10 0 rrcurveto -98 hlineto -16 0 -19 2 -17 0
```

```
rrcurveto -32 -6 -3 -24 hvcurveto closepath endchar} def" \
t1xtt.pfb pfb: t1xtt-shorthyp.pfb
```

We have changed six lines in `tex256.enc` to match the glyph names in the font (e.g. `/endash` → `/rangedash`), and we have changed position 24 to `/shorthyp`. We have also changed the name of the encoding in the beginning of the file. We have inserted the following line to the PostScript font map files (e.g. `psfonts.map`), without the line break:

```
t1xtts t1xtts "TeX256-shorthypEncoding ReEncodeFont"
<tex256-shorthyp.enc <t1xtt-shorthyp.pfb
```

We have also added a new TFM file based on the old one. We have dumped the old one with `tftopl -charcode-format=octal t1xtt.tfm`, modified the width (`CHARWD`) of character 24 (`CHARACTER 0 30`), and saved the modifications with `pltotf modified.pl t1xtts.tfm`. We've added the L<sup>A</sup>T<sub>E</sub>X font map file `t1xtts.fd` with the following content:

```
\DeclareFontFamily{T1}{xtts}{\hyphenchar\font\m@ne}
\DeclareFontShape {T1}{xtts}{m}{n}{<->t1xtts}
```

The `\hyphenchar` settings above disables automatic word hyphenation, so words inside `\texttt` etc. won't be accidentally hyphenated. We have copied all the files above to the appropriate directories and we have run `mktexlsr` to update the file list. We have included some options in `\PexaDefaults` line in the document preamble: `pverb-hyphenchar=char'30` (for automatic word hyphenation) `pverb-linebreak=char, pverb-linebreakchar={\string\char'30}` (inserted around symbols).

We have also defined `\def\pexa@verbatimfont{\normalfont\fontfamily{xtts}\selectfont}`, and we have made sure that the T1 encoding is in use (`\usepackage{t1enc}`).

The overall effect of these modifications was that `examplep` now used our glyph for automatic word hyphenation and as discretionary hyphen around symbols in inline verbatim mode. The demonstrations above shows that it is quite complicated to change a single glyph in a L<sup>A</sup>T<sub>E</sub>X font. It is hoped that the situation will improve with T<sub>E</sub>X's successors.

## 7.7 Passing information about the CD-files to `wfiles.pl`

`wfiles.pl` is used to extract the CD-files from the L<sup>A</sup>T<sub>E</sub>X sources of a book. The reasons why an external program is used instead of T<sub>E</sub>X's built-in `\write` command are described in Subsection 6.1.

The file names and environment start line numbers are passed to `wfiles.pl` in the `.aux` file(s). For example, the line `\@gobble{code:foo.tex:156:2_pic3.mp}` is a declaration that there is a `code` environment starting at line 156 in the file `foo.tex`. `wfiles.pl` understands such declarations, and it also understands lines like `\@input{foo1.aux}`, so dumping works even if the document is separated to several `\included` source files. The declaration above is ignored by L<sup>A</sup>T<sub>E</sub>X when it reads back the `.aux` file (because `\@gobble` gobbles its argument).

Although the `\inputlineno` primitive is mentioned twice in the T<sub>E</sub>Xbook [3], its – rather straightforward – purpose is not documented there. But the real problem is that T<sub>E</sub>X doesn't remember the name of the file being read. `\jobname` contains the name of the top-level `.tex` file, so it doesn't work when that file `\includes` or `\inputs` subfiles containing `code`. The `codep` package thus modifies the `\InputIfFileExists` command to save the file name to the macro `\codep@code@inputfile` if the extension is `.tex`. (The other most common extension after the preamble is `.fd`: such a file is loaded each time a L<sup>A</sup>T<sub>E</sub>X font that has not been used yet is selected.) The implicit limitation here that `code` won't work unless the extension of the file included is `.tex`. Hooking `\InputIfFileExists` affects `\include{...}` and `\input{...}`, but not `\input_...`, `\documentclass` or `\usepackage`. This is not a problem if the author remember that he has to use braces around the file name.

Since there is no hook for `\endinput` (and some packages rely on that `\endinput` is an expandable primitive), it is not possible to set up a stack of names of files being read. Thus, if file *A* has included file *B*, an after that `code` environment placed in *A* will not work, because the declaration line read by `wfiles.pl` will contain the name of *B* instead of *A*. This is not a serious limitation, because files including other files usually don't typeset text by themselves after the inclusion.

The primary reason why `wfiles.pl` needs the `.aux` file is that it has to embed the page and chapter numbers into the file names. Although `wfiles.pl` could find the source file with the `code` environments by trying to match line numbers with all source files in the current directory, we have decided to make it fail when the file name is not emitted properly into the declaration, so it is sure that the examples in the book and on the CD are consistent.

## 8 Future work

The most important features to be added and other improvement possibilities:

- a better approach towards automatic hyphenation of inline verbatim, after studies in typography
- allow wider sample if source is small enough
- why doesn't `\selectlanguage` work inside `\begin{PSource}[srcstyle=leftboth]`
- `\PVerb{foo}` mustn't insert “–” if foo is at end-of-line
- `\PVerb{...}` inner unnested braces (`\futurelet?`)
- differentiated `\penalty` values in `\PVerb`
- paragraph mode should work with side-by-side displays (of course, measuring the width of the Source has still to be done in aligned mode)
- ASCII tabulator (9) characters aren't supported properly, they are just converted to spaces. The width of the tab character should depend on its horizontal position in the line. (With `listings=`, the results are already correct.)
- framing and background color support to display verbatim
- accented characters should work with `listings` and `\PexaShowBoth`. The original catcode of `^` should be kept so T<sub>E</sub>X itself would parse the hex escapes.
- an interface to `\lstinline` in `listings`, with line breaks allowed

## 9 Conclusion

`examplep`, as it is now, is a highly customizable L<sup>A</sup>T<sub>E</sub>X package that provides both inline and display verbatim mode with several advanced features, many of which are not available in any other packages. The `code` environment is also provided which can typeset both the Source and the Sample column of a side-by-side display verbatim from the same L<sup>A</sup>T<sub>E</sub>X source stream, furthermore it can emit the stand-alone working version of the Source into a CD-file. These features make the `code` environment especially useful for software textbook and manual authoring. The whole `examplep` distribution is under the GNU GPL, and it is freely available from CTAN. An earlier version of the packages was used to typeset all the examples in a 770-page introductory book about L<sup>A</sup>T<sub>E</sub>X.

`examplep` is not complete. Some important features are not implemented yet and the package has not been tested thoroughly. Some parts of the code are really ugly, partially because it has not been polished up after writing, and partially because the architecture of T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X doesn't provide an elegant way to address the problem. For example, active characters are overloaded: they are used by `inputenc`, `babel` (shorthands) and `listings` (syntax highlighting) for different purposes – these packages have to make extra effort to cooperate with each other. We hope that T<sub>E</sub>X's successors will improve these conditions, and the core system will provide a generic way to tokenize verbatim text instead of changing catcodes.

## References

- [1] Carsten Heinz. *The Listings Package*, 7 September 2004.  
CTAN:macros/latex/contrib/listings/listings-1.3.dtx.
- [2] David Kastrup. *The binhex.tex package for expansible conversion into binary-based number systems*, 2001.  
CTAN:macros/generic/kastrup/binhex.dtx.
- [3] Donald E. Knuth. *The T<sub>E</sub>Xbook*. Addison–Wesley, 1984.
- [4] Ferenc Wettl, Gyula Mayer, and Péter Szabó. *L<sup>A</sup>T<sub>E</sub>X kézikönyv*. Panem, Budapest, 2004.
- [5] Timothy Van Zandt, Denis Girou, and Sebastian Rahtz. *The ‘fancyvrb’ package. Fancy Verbatims in L<sup>A</sup>T<sub>E</sub>X*, 1998.  
CTAN:macros/latex/contrib/fancyvrb/fancyvrb.dtx.