

The **fvextra** package

Geoffrey M. Poore

gpoore@gmail.com

github.com/gpoore/fvextra

v1.2.1 from 2016/09/02

Abstract

fvextra provides several extensions to **fancyvrb**, including automatic line breaking and improved math mode. It also patches some **fancyvrb** internals.

Contents

1	Introduction	4
2	Usage	4
3	General options	5
4	General commands	10
4.1	Line and text formatting	10
5	Line breaking	10
5.1	Line breaking options	11
5.2	Line breaking and tab expansion	15
5.3	Advanced line breaking	16
5.3.1	A few notes on algorithms	16
5.3.2	Breaks within macro arguments	17
5.3.3	Customizing break behavior	18
6	Pygments support	19
6.1	Options for users	19
6.2	For package authors	19
7	Patches	20
7.1	Visible spaces	20
7.2	<code>obeytabs</code> with visible tabs and with tabs inside macro arguments .	20
7.3	Math mode	21
7.3.1	Spaces	21
7.3.2	Symbols and fonts	21
7.4	Orphaned labels	22
7.5	<code>rulecolor</code> and <code>fillcolor</code>	22
8	Additional modifications to <code>fancyvrb</code>	22
8.1	Backtick and single quotation mark	22
8.2	Line numbering	23
9	Undocumented features of <code>fancyvrb</code>	23
9.1	Undocumented options	23
9.2	Undocumented macros	23
	Version History	24
10	Implementation	25
10.1	Required packages	25
10.2	Utility macros	25
10.3	Hooks	25
10.4	Escaped characters	26

10.5	Patches	26
10.5.1	Visible spaces	26
10.5.2	<code>obeytabs</code> with visible tabs and with tabs inside macro arguments	27
10.5.3	Spacing in math mode	30
10.5.4	Fonts and symbols in math mode	31
10.5.5	Ophaned label	32
10.5.6	<code>rulecolor</code> and <code>fillcolor</code>	32
10.6	Extensions	33
10.6.1	New options requiring minimal implementation	33
10.6.2	Formatting with <code>\FancyVerbFormatLine</code> , <code>\FancyVerbFormatText</code> , and <code>\FancyVerbHighlightLine</code>	34
10.6.3	Line numbering	35
10.6.4	Line highlighting or emphasis	39
10.7	Line breaking	42
10.7.1	Options and associated macros	42
10.7.2	Line breaking implementation	49
10.8	Pygments compatibility	64

1 Introduction

The `fancyvrb` package had its first public release in January 1998. In July of the same year, a few additional features were added. Since then, the package has remained almost unchanged except for a few bug fixes. `fancyvrb` has become one of the primary \LaTeX packages for working with verbatim text.

Additional verbatim features would be nice, but since `fancyvrb` has remained almost unchanged for so long, a major upgrade could be problematic. There are likely many existing documents that tweak or patch `fancyvrb` internals in a way that relies on the existing implementation. At the same time, creating a completely new verbatim package would require a major time investment and duplicate much of `fancyvrb` that remains perfectly functional. Perhaps someday there will be an amazing new verbatim package. Until then, we have `fvextra`.

`fvextra` is an add-on package that gives `fancyvrb` several additional features, including automatic line breaking. Because `fvextra` patches and overwrites some of the `fancyvrb` internals, it may not be suitable for documents that rely on the details of the original `fancyvrb` implementation. `fvextra` tries to maintain the default `fancyvrb` behavior in most cases. All patches (section 7) and modifications to `fancyvrb` defaults (section 8) are documented.

Some features of `fvextra` were originally created as part of the `pythontex` and `minted` packages. `fancyvrb`-related patches and extensions that currently exist in those packages will gradually be migrated into `fvextra`, and both packages will require `fvextra` in the future.

2 Usage

`fvextra` may be used as a drop-in replacement for `fancyvrb`. It will load `fancyvrb` if it has not yet been loaded, and then proceeds to patch `fancyvrb` and define additional features.

The `upquote` package is loaded to give correct backticks (```) and typewriter single quotation marks (`'`). When this is not desirable within a given environment, use the option `curlyquotes`. `fvextra` modifies the behavior of these and other symbols in typeset math within verbatim, so that they will behave as expected (section 7.3). `fvextra` uses the `lineno` package for working with automatic line breaks. `lineno` gives a warning when the `csquotes` package is loaded before it, so `fvextra` should be loaded before `csquotes`. The `ifthen` and `etoolbox` packages are required. `color` or `xcolor` should be loaded manually to use color-dependent features.

While `fvextra` attempts to minimize changes to the `fancyvrb` internals, in some cases it completely overwrites `fancyvrb` macros with new definitions. New definitions typically follow the original definitions as much as possible, but code that depends on the details of the original `fancyvrb` implementation may be incompatible with `fvextra`.

3 General options

`fvextra` adds several general options to `fancyvrb`. All options related to automatic line breaking are described separately in section 5. All options related to syntax highlighting using Pygments are described in section 6.

curlyquotes (boolean) (default: `false`)
 Unlike `fancyvrb`, `fvextra` requires the `upquote` package, so the backtick (```) and typewriter single quotation mark (`'`) always appear literally by default, instead of becoming the left and right curly single quotation marks (`‘``’`). This option allows these characters to be replaced by the curly quotation marks when that is desirable.

<pre>\begin{Verbatim} `quoted text' \end{Verbatim}</pre>	<pre>`quoted text'</pre>
<pre>\begin{Verbatim}[curlyquotes] `quoted text' \end{Verbatim}</pre>	<pre>‘quoted text’</pre>

highlightcolor (string) (default: `LightCyan`)
 Set the color used for `highlightlines`, using a predefined color name from `color` or `xcolor`, or a color defined via `\definecolor`.

highlightlines (string) (default: `<none>`)
 This highlights a single line or a range of lines based on line numbers. The line numbers refer to the line numbers that `fancyvrb` would show if `numbers=left`, etc. They do not refer to original or actual line numbers before adjustment by `firstnumber`.
 The highlighting color can be customized with `highlightcolor`.

```

\begin{Verbatim}[numbers=left, highlightlines={1, 3-4}]
First line
Second line
Third line
Fourth line
Fifth line
\end{Verbatim}

```

```

1 First line
2 Second line
3 Third line
4 Fourth line
5 Fifth line

```

The actual highlighting is performed by a set of commands. These may be customized for additional fine-tuning of highlighting. See the default definition of `\FancyVerbHighlightLineFirst` as a starting point.

- `\FancyVerbHighlightLineFirst`: First line in a range.
- `\FancyVerbHighlightLineMiddle`: Inner lines in a range.
- `\FancyVerbHighlightLineLast`: Last line in a range.
- `\FancyVerbHighlightLineSingle`: Single highlighted lines.
- `\FancyVerbHighlightLineNormal`: Normal lines without highlighting.

If these are customized in such a way that indentation or inter-line spacing is changed, then `\FancyVerbHighlightLineNormal` may be modified as well to make all lines uniform. When working with the `First`, `Last`, and `Single` commands, keep in mind that `fvextra` merges all numbers ranges, so that `{1, 2-3, 3-5}` is treated the same as `{1-5}`.

Highlighting is applied after `\FancyVerbFormatText`, so any text formatting defined via that command will work with highlighting. Highlighting is applied before `\FancyVerbFormatLine`, so if `\FancyVerbFormatLine` puts a line in a box, the box will be behind whatever is created by highlighting. This prevents highlighting from vanishing due to user-defined customization.

linenos (boolean) (default: `false`)
`fancyvrb` allows line numbers via the options `numbers=<position>`. This is essentially an alias for `numbers=left`. It primarily exists for better compatibility with the `minted` package.

mathescape (boolean) (default: `false`)
This causes everything between dollar signs `$...$` to be typeset as math. The caret `^` and underscore `_` have their normal math meanings.

This is equivalent to `codes={\catcode`$=3\catcode`^=7\catcode`_=8}`. `mathescape` is always applied *before* `codes`, so that `codes` can be used to override

some of these definitions.

Note that `fvextra` provides several patches that make math mode within verbatim as close to normal math mode as possible (section 7.3).

numberfirstline (boolean) (default: `false`)
 When line numbering is used with `stepnumber` $\neq 1$, the first line may not always be numbered, depending on the line number of the first line. This causes the first line always to be numbered.

<pre> \begin{Verbatim}[numbers=left, stepnumber=2, numberfirstline] First line Second line Third line Fourth line \end{Verbatim} </pre>	
<pre> 1 First line 2 Second line Third line 4 Fourth line </pre>	

numbers (none | left | right | both) (default: `none`)
`fvextra` adds the `both` option for line numbering.

<pre> \begin{Verbatim}[numbers=both] First line Second line Third line Fourth line \end{Verbatim} </pre>	<pre> 1 First line 1 2 Second line 2 3 Third line 3 4 Fourth line 4 </pre>
--	---

space (macro) (default: `\textvisiblespace`, `␣`)
 Redefine the visible space character. Note that this is only used if `showspaces=true`. The color of the character may be set with `spacecolor`.

spacecolor (string) (default: `none`)
 Set the color of visible spaces. By default (`none`), they take the color of their surroundings.

```
\color{gray}
\begin{Verbatim}[showspaces, spacecolor=red]
One two three
\end{Verbatim}
```

Oneutwouthree

stepnumberfromfirst (boolean) (default: **false**)

By default, when line numbering is used with **stepnumber** $\neq 1$, only line numbers that are a multiple of **stepnumber** are included. This offsets the line numbering from the first line, so that the first line, and all lines separated from it by a multiple of **stepnumber**, are numbered.

```
\begin{Verbatim}[numbers=left, stepnumber=2,
                  stepnumberfromfirst]

First line
Second line
Third line
Fourth line
\end{Verbatim}
```

```
1 First line
  Second line
3 Third line
  Fourth line
```

stepnumberoffsetvalues (boolean) (default: **false**)

By default, when line numbering is used with **stepnumber** $\neq 1$, only line numbers that are a multiple of **stepnumber** are included. Using **firstnumber** to offset the numbering will change which lines are numbered and which line gets which number, but will not change which *numbers* appear. This option causes **firstnumber** to be ignored in determining which line numbers are a multiple of **stepnumber**. **firstnumber** is still used in calculating the actual numbers that appear. As a result, the line numbers that appear will be a multiple of **stepnumber**, plus **firstnumber** minus 1.

This option gives the original behavior of **fancyvrb** when **firstnumber** is used with **stepnumber** $\neq 1$ (section 8.2).


```

\begin{Verbatim}[numbers=left, stepnumber=2,
                  firstnumber=4, stepnumberoffsetvalues]

First line
Second line
Third line
Fourth line
\end{Verbatim}

```

```

First line
5 Second line
Third line
7 Fourth line

```

tab (macro) (default: fancyvrb's `\FancyVerbTab`, ↗)
 Redefine the visible tab character. Note that this is only used if `showtabs=true`.
 The color of the character may be set with `tabcolor`.

When redefining the tab, you should include the font family, font shape, and text color in the definition. Otherwise these may be inherited from the surrounding text. This is particularly important when using the tab with syntax highlighting, such as with the `minted` or `pythontex` packages.

`fvetra` patches `fancyvrb` tab expansion so that variable-width symbols such as `\rightarrowfill` may be used as tabs. For example,

```

\begin{Verbatim}[obeytabs, showtabs, breaklines,
                  tab=\rightarrowfill, tabcolor=orange]
↗First ↗Second ↗Third ↗And more text that goes on for a
↪ while until wrapping is needed
↗First ↗Second ↗Third ↗Forth
\end{Verbatim}

```

```

→First→Second→Third→And more text that goes on for a
↪ while until wrapping is needed
→First→Second→Third→Forth

```

tabcolor (string) (default: `none`)
 Set the color of visible tabs. By default (`none`), they take the color of their surroundings.

4 General commands

4.1 Line and text formatting

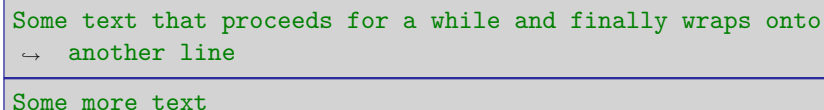
`\FancyVerbFormatLine`
`\FancyVerbFormatText`

`fancyvrb` defines `\FancyVerbFormatLine`, which can be used to apply custom formatting to each individual line of text. By default, it takes a line as an argument and inserts it with no modification. This is equivalent to `\newcommand{\FancyVerbFormatLine}[1]{#1}`.¹

`fvextra` introduces line breaking, which complicates line formatting. We might want to apply formatting to the entire line, including line breaks, line continuation symbols, and all indentation, including any extra indentation provided by line breaking. Or we might want to apply formatting only to the actual text of the line. `fvextra` leaves `\FancyVerbFormatLine` as applying to the entire line, and introduces a new command `\FancyVerbFormatText` that only applies to the text part of the line.² By default, `\FancyVerbFormatText` inserts the text unmodified. When it is customized, it should not use boxes that do not allow line breaks to avoid conflicts with line breaking code.

```
\renewcommand{\FancyVerbFormatLine}[1]{%
  \fcolorbox{DarkBlue}{LightGray}{#1}}
\renewcommand{\FancyVerbFormatText}[1]{\textcolor{Green}{#1}}

\begin{Verbatim}[breaklines]
Some text that proceeds for a while and finally wraps onto another line
Some more text
\end{Verbatim}
```



5 Line breaking

Automatic line breaking may be turned on with `breaklines=true`. By default, breaks only occur at spaces. Breaks may be allowed anywhere with `breakanywhere`,

¹The actual definition in `fancyvrb` is `\def\FancyVerbFormatLine#1{\FV@ObeyTabs{#1}}`. This is problematic because redefining the macro could easily eliminate `\FV@ObeyTabs`, which governs tab expansion. `fvextra` redefines the macro to `\def\FancyVerbFormatLine#1{#1}` and patches all parts of `fancyvrb` that use `\FancyVerbFormatLine` so that `\FV@ObeyTabs` is explicitly inserted at the appropriate points.

²When `breaklines=true`, each line is wrapped in a `\parbox`. `\FancyVerbFormatLine` is outside the `\parbox`, and `\FancyVerbFormatText` is inside.

or only before or after specified characters with **breakbefore** and **breakafter**. Many options are provided for customizing breaks. A good place to start is the description of **breaklines**.

5.1 Line breaking options

Options are provided for customizing typical line breaking features. See section 5.3 for details about low-level customization of break behavior.

breakafter (string) (default: *none*)
 Break lines after specified characters, not just at spaces, when **breaklines=true**. For example, **breakafter=-/** would allow breaks after any hyphens or slashes. Special characters given to **breakafter** should be backslash-escaped (usually #, {, }, %, [,]; the backslash `\` may be obtained via `\\` and the space via `\space`).³

For an alternative, see **breakbefore**. When **breakbefore** and **breakafter** are used for the same character, **breakbeforegroup** and **breakaftergroup** must both have the same setting.

Note that when **commandchars** or **codes** are used to include macros within verbatim content, breaks will not occur within mandatory macro arguments by default. Depending on settings, macros that take optional arguments may not work unless the entire macro including arguments is wrapped in a group (curly braces `{}`, or other characters specified with **commandchars**). See section 5.3 for details.

```
\begin{Verbatim}[breaklines, breakafter=d]
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNeverFitOnOneLine'
\end{Verbatim}

some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCould
↪ NeverFitOnOneLine'
```

breakaftergroup (boolean) (default: **true**)
 When **breakafter** is used, group all adjacent identical characters together, and only allow a break after the last character. When **breakbefore** and **breakafter** are used for the same character, **breakbeforegroup** and **breakaftergroup** must both have the same setting.

breakaftersymbolpre (string) (default: `\,\footnotesize\ensuremath{_ \rfloor}`, `_`)
 The symbol inserted pre-break for breaks inserted by **breakafter**.

breakaftersymbolpost (string) (default: *none*)

³**breakafter** expands each token it is given once, so when it is given a macro like `\%`, the macro should expand to a literal character that will appear in the text to be typeset. **fveextra** defines special character escapes that are activated for **breakafter** so that this will work with common escapes. The only exception to token expansion is non-ASCII characters under pdfTeX; these should appear literally. **breakafter** is not catcode-sensitive.

The symbol inserted post-break for breaks inserted by `breakafter`.

`breakanywhere` (boolean) (default: `false`)
Break lines anywhere, not just at spaces, when `breaklines=true`.

Note that when `commandchars` or `codes` are used to include macros within verbatim content, breaks will not occur within mandatory macro arguments by default. Depending on settings, macros that take optional arguments may not work unless the entire macro including arguments is wrapped in a group (curly braces `{}`), or other characters specified with `commandchars`). See section 5.3 for details.

```
\begin{Verbatim}[breaklines, breakanywhere]
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNeverFitOnOneLine'
\end{Verbatim}

some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNeve
→ rFitOnOneLine'
```

`breakanywheresymbolpre` (string) (default: `\,\footnotesize\ensuremath{_}\rfloor`)
The symbol inserted pre-break for breaks inserted by `breakanywhere`.

`breakanywheresymbolpost` (string) (default: `none`)
The symbol inserted post-break for breaks inserted by `breakanywhere`.

`breakautoindent` (boolean) (default: `true`)
When a line is broken, automatically indent the continuation lines to the indentation level of the first line. When `breakautoindent` and `breakindent` are used together, the indentations add. This indentation is combined with `breaksymbolindentleft` to give the total actual left indentation.

`breakbefore` (string) (default: `none`)
Break lines before specified characters, not just at spaces, when `breaklines=true`. For example, `breakbefore=A` would allow breaks before capital A's. Special characters given to `breakbefore` should be backslash-escaped (usually `#`, `{`, `}`, `%`, `[`, `]`; the backslash `\` may be obtained via `\\` and the space via `\space`).⁴

For an alternative, see `breakafter`. When `breakbefore` and `breakafter` are used for the same character, `breakbeforegroup` and `breakaftergroup` must both have the same setting.

Note that when `commandchars` or `codes` are used to include macros within verbatim content, breaks will not occur within mandatory macro arguments by default. Depending on settings, macros that take optional arguments may not work

⁴`breakbefore` expands each token it is given once, so when it is given a macro like `\%`, the macro should expand to a literal character that will appear in the text to be typeset. `fvetra` defines special character escapes that are activated for `breakbefore` so that this will work with common escapes. The only exception to token expansion is non-ASCII characters under pdfTeX; these should appear literally. `breakbefore` is not catcode-sensitive.

unless the entire macro including arguments is wrapped in a group (curly braces `{}`), or other characters specified with `commandchars`). See section 5.3 for details.

```
\begin{Verbatim}[breaklines, breakbefore=A]
some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNeverFitOnOneLine'
\end{Verbatim}
```

```
some_string = 'SomeTextThatGoesOn_
↪ AndOnForSoLongThatItCouldNeverFitOnOneLine'
```

breakbeforegroup (boolean) (default: `true`)
When **breakbefore** is used, group all adjacent identical characters together, and only allow a break before the first character. When **breakbefore** and **breakafter** are used for the same character, **breakbeforegroup** and **breakaftergroup** must both have the same setting.

breakbeforesymbolpre (string) (default: `\,\footnotesize\ensuremath{_\rfloor}`)
The symbol inserted pre-break for breaks inserted by **breakbefore**.

breakbeforesymbolpost (string) (default: `none`)
The symbol inserted post-break for breaks inserted by **breakbefore**.

breakindent (dimension) (default: `0pt`)
When a line is broken, indent the continuation lines by this amount. When **breakautoindent** and **breakindent** are used together, the indentations add. This indentation is combined with **breaksymbolindentleft** to give the total actual left indentation.

breaklines (boolean) (default: `false`)
Automatically break long lines.
By default, automatic breaks occur at spaces. Use **breakanywhere** to enable breaking anywhere; use **breakbefore** and **breakafter** for more fine-tuned breaking.

```
...text.
\begin{Verbatim}[breaklines]
def f(x):
    return 'Some text ' + str(x)
\end{Verbatim}
```

```
...text.
def f(x):
    return 'Some text ' +
↪ str(x)
```

To customize the indentation of broken lines, see **breakindent** and **breakautoindent**. To customize the line continuation symbols, use **breaksymbolleft** and **breaksymbolright**. To customize the separation between the continuation symbols and the text, use **breaksymbolsepleft** and **breaksymbolsepright**. To customize the extra indentation that is supplied to make room for the break symbols, use

`breaksymbolindentleft` and `breaksymbolindentright`. Since only the left-hand symbol is used by default, it may also be modified using the alias options `breaksymbol`, `breaksymbolsep`, and `breaksymbolindent`.

An example using these options to customize the `Verbatim` environment is shown below. This uses the `\carriagereturn` symbol from the `dingbat` package.

```
\begin{Verbatim}[breaklines,
                    breakautoindent=false,
                    breaksymbolleft=\raisebox{0.8ex}{\small\reflectbox{\carriagereturn}},
                    breaksymbolindentleft=0pt,
                    breaksymbolsepleft=0pt,
                    breaksymbolright=\small\carriagereturn,
                    breaksymbolindentright=0pt,
                    breaksymbolsepright=0pt]

def f(x):
    return 'Some text ' + str(x) + ' some more text ' +
        ↪ str(x) + ' even more text that goes on for a while'
\end{Verbatim}
```

```
def f(x):
    return 'Some text ' + str(x) + ' some more text ' +
    ↵ str(x) + ' even more text that goes on for a while'
```

Automatic line breaks will not work with `showspaces=true` unless you use `breakanywhere`, or use `breakbefore` or `breakafter` with `\space`. For example,

```
\begin{Verbatim}[breaklines, showspaces, breakafter=\space]
some_string = 'Some Text That Goes On And On For So Long That It Could Never Fit'
\end{Verbatim}
```

```
some_string_ = 'Some_Text_That_Goes_On_And_On_For_So_Long_That_
    ↪ It_Could_Never_Fit'
```

breaksymbol (string) (default: `breaksymbolleft`)
 Alias for `breaksymbolleft`.

breaksymbolleft (string) (default: `\tiny\ensuremath{\hookrightarrow}`)
 The symbol used at the beginning (left) of continuation lines when `breaklines=true`. To have no symbol, simply set `breaksymbolleft` to an empty string (“=,” or “={}”). The symbol is wrapped within curly braces {} when used, so there is no danger of formatting commands such as `\tiny` “escaping.”

The `\hookrightarrow` and `\hookleftarrow` may be further customized by the use of the `\rotatebox` command provided by `graphicx`. Additional arrow-type symbols that may be useful are available in the `dingbat` (`\carriagereturn`) and `mnsymbol` (hook and curve arrows) packages, among others.

<code>breaksymbolright</code>	(string) (default: <i>none</i>) The symbol used at breaks (right) when <code>breaklines=true</code> . Does not appear at the end of the very last segment of a broken line.
<code>breaksymbolindent</code>	(dimension) (default: <code>breaksymbolindentleft</code>) Alias for <code>breaksymbolindentleft</code> .
<code>breaksymbolindentleft</code>	(dimension) (default: <i>width of 4 characters in teletype font at default point size</i>) The extra left indentation that is provided to make room for <code>breaksymbolleft</code> . This indentation is only applied when there is a <code>breaksymbolleft</code> . This may be set to the width of a specific number of (fixed-width) characters by using an approach such as <code>\newdimen\temporarydimen</code> <code>\settowidth{\temporarydimen}{\ttfamily aaaa}</code> and then using <code>breaksymbolindentleft=\temporarydimen</code> .
<code>breaksymbolindentrigh</code>	(dimension) (default: <i>width of 4 characters in teletype font at default point size</i>) The extra right indentation that is provided to make room for <code>breaksymbolright</code> . This indentation is only applied when there is a <code>breaksymbolright</code> .
<code>breaksymbolsep</code>	(dimension) (default: <code>breaksymbolsepleft</code>) Alias for <code>breaksymbolsepleft</code> .
<code>breaksymbolsepleft</code>	(dimension) (default: 1em) The separation between the <code>breaksymbolleft</code> and the adjacent text.
<code>breaksymbolsepright</code>	(dimension) (default: 1em) The separation between the <code>breaksymbolright</code> and the adjacent text.

5.2 Line breaking and tab expansion

`fancyvrb` provides an `obeytabs` option that expands tabs based on tab stops rather than replacing them with a fixed number of spaces (see `fancyvrb`'s `tabsize`). The `fancyvrb` implementation of tab expansion is not directly compatible with `fvextra`'s line-breaking algorithm, but `fvextra` builds on the `fancyvrb` approach to obtain identical results.

Tab expansion in the context of line breaking does bring some additional considerations that should be kept in mind. In each line, all tabs are expanded exactly as they would have been had the line not been broken. This means that after a line break, any tabs will not align with tab stops unless the total left

indentation of continuation lines is a multiple of the tab stop width. The total indentation of continuation lines is the sum of `breakindent`, `breakautoindent`, and `breaksymbolindentleft` (alias `breaksymbolindent`).

A sample `Verbatim` environment that uses `obeytabs` with `breaklines` is shown below, with numbers beneath the environment indicating tab stops (`tabsize=8` by default). The tab stops in the wrapped and unwrapped lines are identical. However, the continuation line does not match up with the tab stops because by default the width of `breaksymbolindentleft` is equal to four monospace characters. (By default, `breakautoindent=true`, so the continuation line gets a tab plus `breaksymbolindentleft`.)

```
\begin{Verbatim}[obeytabs, showtabs, breaklines]
  ↵First  ↵Second ↵Third ↵And more text that goes on for a
    ↵ while until wrapping is needed
  ↵First  ↵Second ↵Third ↵Forth
\end{Verbatim}
```

1234567812345678123456781234567812345678123456781234567812345678

We can set the symbol indentation to eight characters by creating a `dimen`,

```
\newdimen\temporarydimen
```

setting its width to eight characters,

```
\settothewidth{\temporarydimen}{\ttfamily AaAaAaAa}
```

and finally adding the option `breaksymbolindentleft=\temporarydimen` to the `Verbatim` environment to obtain the following:

```
↵First  ↵Second ↵Third ↵And more text that goes on for a
    ↵ while until wrapping is needed
↵First  ↵Second ↵Third ↵Forth
```

1234567812345678123456781234567812345678123456781234567812345678

5.3 Advanced line breaking

5.3.1 A few notes on algorithms

`breakanywhere`, `breakbefore`, and `breakafter` work by scanning through the tokens in each line and inserting line breaking commands wherever a break should be allowed. By default, they skip over all groups (`{...}`) and all math (`$...$`). Note that this refers to curly braces and dollar signs with their normal \LaTeX meaning (catcodes), not verbatim curly braces and dollar signs; such non-verbatim content may be enabled with `commandchars` or `codes`. This means that math and macros

that only take mandatory arguments (`{...}`) will function normally within otherwise verbatim text. However, macros that take optional arguments may not work because `[...]` is not treated specially, and thus break commands may be inserted within `[...]` depending on settings. Wrapping an entire macro, including its arguments, in a group will protect the optional argument: `{\<macro>[\<oarg>]{\<marg>}}`.

`breakbefore` and `breakafter` insert line breaking commands around specified characters. This process is catcode-independent; tokens are `\detokenized` before they are checked against characters specified via `breakbefore` and `breakafter`.

5.3.2 Breaks within macro arguments

```
\FancyVerbBreakStart
\FancyVerbBreakStop
```

When `commandchars` or `codes` are used to include macros within verbatim content, the options `breakanywhere`, `breakbefore`, and `breakafter` will not generate breaks within mandatory macro arguments. Macros with optional arguments may not work, depending on settings, unless they are wrapped in a group (curly braces `{}`, or other characters specified via `commandchars`).

If you want to allow breaks within macro arguments (optional or mandatory), then you should (re)define your macros so that the relevant arguments are wrapped in the commands

```
\FancyVerbBreakStart ... \FancyVerbBreakStop
```

For example, suppose you have the macro

```
\newcommand{\mycmd}[1]{\_before:#1:after\_}
```

Then you would discover that line breaking does not occur:

```
\begin{Verbatim}[commandchars=\\\{\}, breaklines, breakafter=a]
\mycmd{1}\mycmd{2}\mycmd{3}\mycmd{4}\mycmd{5}
\end{Verbatim}
```

```
_before:1:after__before:2:after__before:3:after__before:4:after__before:5:after_
```

Now redefine the macro:

```
\renewcommand{\mycmd}[1]{\FancyVerbBreakStart\_before:#1:after\_ \FancyVerbBreakStop}
```

This is the result:

```
\begin{Verbatim}[commandchars=\\\{\}, breaklines, breakafter=a]
\mycmd{1}\mycmd{2}\mycmd{3}\mycmd{4}\mycmd{5}
\end{Verbatim}
```

```
_before:1:after__before:2:after__before:3:after__before:4:a
→ fter__before:5:after_
```

Instead of completely redefining macros, it may be more convenient to use `\let`. For example,

```
\let\originalmycmd\mycmd
\renewcommand{\mycmd}[1]{%
  \expandafter\FancyVerbBreakStart\originalmycmd{#1}\FancyVerbBreakStop}
```

Notice that in this case `\expandafter` is required, because `\FancyVerbBreakStart` does not perform any expansion and thus will skip over `\originalmycmd{#1}` unless it is already expanded. The `etoolbox` package provides commands that may be useful for patching macros to insert line breaks.

When working with `\FancyVerbBreakStart ... \FancyVerbBreakStop`, keep in mind that any groups `{...}` or math `...$` between the two commands will be skipped as far as line breaks are concerned, and breaks may be inserted within any optional arguments `[...]` depending on settings. Inserting breaks within groups requires another level of `\FancyVerbBreakStart` and `\FancyVerbBreakStop`, and protecting optional arguments requires wrapping the entire macro in a group `{...}`. Also, keep in mind that `\FancyVerbBreakStart` cannot introduce line breaks in a context in which they are never allowed, such as in an `\hbox`.

5.3.3 Customizing break behavior

`\FancyVerbBreakAnywhereBreak`

These macros govern the behavior of breaks introduced by `breakanywhere`, `breakbefore`, and `breakafter`. Breaks introduced by the default `breaklines` when `showspaces=false` are standard breaks following spaces. No special commands are provided for working with them; the normal L^AT_EX commands for breaking should suffice.

By default, these macros use `\discretionary`. `\discretionary` takes three arguments: commands to insert before the break, commands to insert after the break, and commands to insert if there is no break. For example, the default definition of `\FancyVerbBreakAnywhereBreak`:

```
\newcommand{\FancyVerbBreakAnywhereBreak}{%
  \discretionary{\FancyVerbBreakAnywhereSymbolPre}%
    {\FancyVerbBreakAnywhereSymbolPost}{}}
```

The other macros are equivalent, except that “Anywhere” is swapped for “Before” or “After”.

`\discretionary` will generally only insert breaks when breaking at spaces simply cannot make lines short enough (this may be tweaked to some extent with hyphenation settings). This can produce a somewhat ragged appearance in some cases. If you want breaks exactly at the margin (or as close as possible) regardless of whether a break at a space is an option, you may want to use `\allowbreak` instead. Another option is `\linebreak[⟨n⟩]`, where `⟨n⟩` is between 0 to 4, with 0 allowing a break and 4 forcing a break.

6 Pygments support

6.1 Options for users

`fvextra` defines additional options for working code that has been highlighted with `Pygments`. These options work with the `minted` and `pythontex` packages, and may be enabled for other packages that work with `Pygments` output (section 6.2).

`breakbytoken` (boolean) (default: `false`)
When `breaklines=true`, do not allow breaks within `Pygments` tokens. This would prevent, for example, line breaking within strings.

`breakbytokenanywhere` (boolean) (default: `false`)
When `breaklines=true`, do not allow breaks within `Pygments` tokens, but always allow breaks between tokens even when they are immediately adjacent (not separated by spaces). **This option should be used with care.** Due to the details of how each `Pygments` lexer works, and due to the tokens defined in each lexer, this may result in breaks in locations that might not be anticipated. Also keep in mind that this will not allow breaks between tokens if those tokens are actually “subtokens” within another token.

`\FancyVerbBreakByTokenAnywhereBreak`

This defines the break inserted when `breakbytokenanywhere=true`. By default, it is `\allowbreak`.

6.2 For package authors

By default, line breaking will only partially work with `Pygments` output; `breakbefore` and `breakafter` will not work with any characters that do not appear literally in `Pygments` output but rather are replaced with a character macro. Also, `breakbytoken` and `breakbytokenanywhere` will not function at all.

`\VerbatimPygments{⟨literal_macro⟩}{⟨actual_macro⟩}`

To enable full `Pygments` support, use this macro before `\begin{Verbatim}`, etc. This macro must be used within `\begingroup... \endgroup` to prevent settings from escaping into the rest of the document. It may be used safely at the beginning of a `\newenvironment` definition. When used with `\newcommand`, though, the `\begingroup... \endgroup` will need to be inserted explicitly.

`⟨literal_macro⟩` is the `Pygments` macro that literally appears in `Pygments` output; it corresponds to the `Pygments` `commandprefix`. For `minted` and `pythontex`,

this is `\PYG`. $\langle actual_macro \rangle$ is the Pygments macro that should actually be used. For `minted` and `pythontex`, this is `\PYG<style>`. In the `minted` and `pythontex` approach, code is only highlighted once (`\PYG`), and then the style is changed by redefining the macro that literally appears (`\PYG`) to use the appropriate style macro (`\PYG<style>`).

`\VerbatimPygments` takes the two Pygments macros and redefines $\langle literal_macro \rangle$ so that it will invoke $\langle actual_macro \rangle$ while fully supporting line breaks, `breakbytoken`, and `breakbytokenanywhere`. No further modification of either $\langle literal_macro \rangle$ or $\langle actual_macro \rangle$ is possible after `\VerbatimPygments` is used.

In packages that do not make a distinction between $\langle literal_macro \rangle$ and $\langle actual_macro \rangle$, simply use `\VerbatimPygments` with two identical arguments; `\VerbatimPygments` is defined to handle this case.

7 Patches

`fvextra` modifies some `fancyvrb` behavior that is the result of bugs or omissions.

7.1 Visible spaces

The command `\FancyVerbSpace` defines the visible space when `showspaces=true`. The default `fancyvrb` definition allows a font command to escape under some circumstances, so that all following text is forced to be teletype font. The command is redefined to use `\textvisiblespace`.

7.2 obeytabs with visible tabs and with tabs inside macro arguments

The original `fancyvrb` treatment of visible tabs when `showtabs=true` and `obeytabs=true` did not allow variable-width tab symbols such as `\rightarrowfill` to function correctly. This is fixed through a redefinition of `\FV@TrueTab`.

Various macros associated with `obeytabs=true` are also redefined so that tabs may be expanded regardless of whether they are within a group (within `{...}` with the normal \LaTeX meaning due to `commandchars`, etc.). In the `fancyvrb` implementation, using `obeytabs=true` when a tab is inside a group typically causes the entire line to vanish. `fvextra` patches this so that the tab is expanded and will be visible if `showtabs=true`. Note, though, that the tab expansion in these cases is only guaranteed to be correct for leading whitespace that is inside a group. The start of each run of whitespace that is inside a group is treated as a tab stop, whether or not it actually is, due to limitations of the tab expansion algorithm. A more detailed discussion is provided in the implementation.

The example below shows correct tab expansion of leading whitespace within a macro argument. With `fancyvrb`, the line of text would simply vanish in this case.

```
\begin{Verbatim}[obeytabs, showtabs, showspaces, tabsize=4,
  commandchars=\\{\}, tab=\textcolor{orange}{\rightarrowfill}]
```

```
\textcolor{blue}{          ↗          ↗Text after 1 space + 2 tabs}
\end{Verbatim}
```

↗→↗→↗Text↗after↗1↗space↗+↗2↗tabs

The next example shows that tab expansion inside macros in the midst of text typically does not match up with the correct tab stops, since in such circumstances the beginning of the run of whitespace must be treated as a tab stop.

```
\begin{Verbatim}[obeytabs, showtabs, commandchars=\\\{\},
                  tab=\textcolor{orange}{\rightarrowfill}]
\textcolor{blue}{          ↗          ↗2 leading tabs}
\textcolor{blue}{Text    ↗          ↗then 2 tabs}
\end{Verbatim}
```

↗→↗→↗→↗2 leading tabs
Text↗→↗→↗→↗then 2 tabs

7.3 Math mode

7.3.1 Spaces

When typeset math is included within verbatim material, `fancyvrb` makes spaces within the math appear literally.

```
\begin{Verbatim}[commandchars=\\\{\}, mathescape]
Verbatim $\displaystyle\frac{1}{x^2 + y^2}$ verbatim
\end{Verbatim}
```

Verbatim $\frac{1}{x^2 + y^2}$ verbatim

`fvextra` patches this by redefining `fancyvrb`'s space character within math mode so that it behaves as expected:

Verbatim $\frac{1}{x^2 + y^2}$ verbatim

7.3.2 Symbols and fonts

With `fancyvrb`, using a single quotation mark (') in typeset math within verbatim material results in an error rather than a prime symbol (').⁵ `fvextra` redefines the

⁵The single quotation mark is made active within verbatim material to prevent ligatures, via `\@noligs`. The default definition is incompatible with math mode.

behavior of the single quotation mark within math mode to fix this, so that it will become a proper prime.

The `amsmath` package provides a `\text` command for including normal text within math. With `fancyvrb`, `\text` does not behave normally when used in typeset math within verbatim material. `fvextra` redefines the backtick (```) and the single quotation mark so that they function normally within `\text`, becoming left and right quotation marks. It redefines the greater-than sign, less-than sign, comma, and hyphen so that they function normally as well. `fvextra` also switches back to the default document font within `\text`, rather than using the verbatim font, which is typically a monospace or typewriter font.

The result of these modifications is a math mode that very closely mimics the behavior of normal math mode outside of verbatim material.

```
\begin{Verbatim}[commandchars=\\\{\}, mathescape]
Verbatim $\displaystyle f'''(x) = \text{``Some quoted text---''}$
\end{Verbatim}
```

Verbatim $f'''(x) = \text{“Some quoted text—”}$

7.4 Orphaned labels

When `frame=lines` is used with a label, `fancyvrb` does not prevent the label from being orphaned under some circumstances. `\FV@BeginListFrame@Lines` is patched to prevent this.

7.5 rulecolor and fillcolor

The `rulecolor` and `fillcolor` options are redefined so that they accept color names directly, rather than requiring `\color{<color_name>}`. The definitions still allow the old usage.

8 Additional modifications to fancyvrb

`fvextra` modifies some `fancyvrb` behavior with the intention of improving logical consistency or providing better defaults.

8.1 Backtick and single quotation mark

With `fancyvrb`, the backtick ``` and typewriter single quotation mark `'` are typeset as the left and right curly single quotation marks `‘` and `’`. `fvextra` loads the `upquote` package so that these characters will appear literally by default. The original `fancyvrb` behavior can be restored with the `fvextra` option `curlyquotes` (section 3).

8.2 Line numbering

With `fancyvrb`, using `firstnumber` to offset line numbering in conjunction with `stepnumber` changes which line numbers appear. Lines are numbered if their original line numbers, without the `firstnumber` offset, are a multiple of `stepnumber`. But the actual numbers that appear are the offset values that include `firstnumber`. Thus, using `firstnumber=2` with `stepnumber=5` would cause the original lines 5, 10, 15, ... to be numbered, but with the values 6, 11, 16,

`fvextra` changes line numbering so that when `stepnumber` is used, the actual line numbers that appear are always multiples of `stepnumber` by default, regardless of any `firstnumber` offset. The original `fancyvrb` behavior may be turned on by setting `stepnumberoffsetvalues=true` (section 3).

9 Undocumented features of `fancyvrb`

`fancyvrb` defines some potentially useful but undocumented features.

9.1 Undocumented options

<code>codes*</code>	(macro) (default: <i>empty</i>) <code>fancyvrb</code> 's <code>codes</code> is used to specify catcode changes. It overwrites any existing <code>codes</code> . <code>codes*</code> appends changes to existing settings.
<code>defineactive*</code>	(macro) (default: <i>empty</i>) <code>fancyvrb</code> 's <code>defineactive</code> is used to define the effect of active characters. It overwrites any existing <code>defineactive</code> . <code>defineactive*</code> appends changes to existing settings.
<code>formatcom*</code>	(macro) (default: <i>empty</i>) <code>fancyvrb</code> 's <code>formatcom</code> is used to execute commands before verbatim text. It overwrites any existing <code>formatcom</code> . <code>formatcom*</code> appends changes to existing settings.

9.2 Undocumented macros

`\FancyVerbTab`

This defines the visible tab character (\rightarrow) that is used when `showtabs=true`. The default definition is

```
\def\FancyVerbTab{%
  \valign{%
    \vfil##\vfil\cr
    \hbox{$\scriptscriptstyle-$}\cr
    \hbox to 0pt{\hss$\scriptscriptstyle\rangle\mskip -.8mu$}\cr
    \hbox{$\scriptstyle\mskip -3mu\mid\mskip -1.4mu$}\cr}}}
```

While this may be redefined directly, `fvextra` also defines a new option `tab`

`\FancyVerbSpace`

This defines the visible space character (`_`) that is used when `showspaces=true`. The default definition (as patched by `fvextra`, section 7.1) is `\textvisiblespace`. While this may be redefined directly, `fvextra` also defines a new option `space`.

Version History

v1.2.1 (2016/09/02)

- The package is now compatible with classes and packages that redefine `\raggedright`.
- Fixed a bug that introduced extra space in inline contexts such as `\mintinline` when `breaklines=true` (#3).

v1.2 (2016/07/20)

- Added support for line breaking when working with Pygments for syntax highlighting.
- The default `highlightcolor` is now defined with `rgb` for compatibility with the `color` package. Fixed a bug in the conditional color definition when `color` and `xcolor` are not loaded before `fvextra`.

v1.1 (2016/07/14)

- The options `rulecolor` and `fillcolor` now accept color names directly; using `\color{<color_name>}` is no longer necessary, though it still works.
- Added `tabcolor` and `spacecolor` options for use with `showtabs` and `showspaces`.
- Added `highlightlines` option that takes a line number or range of line numbers and highlights the corresponding lines. Added `highlightcolor` option that controls highlighting color.
- `obeytabs` no longer causes lines to vanish when tabs are inside macro arguments. Tabs and spaces inside a macro argument but otherwise at the beginning of a line are expanded correctly. Tabs inside a macro argument that are preceded by non-whitespace characters (not spaces or tabs) are expanded based on the starting position of the run of whitespace in which they occur.
- The line breaking options `breakanywhere`, `breakbefore`, and `breakafter` now work with multi-byte UTF-8 code points under pdfTeX with `inputenc`. They were already fully functional under XeTeX and LuaTeX.
- Added `curlyquotes` option, which essentially disables the `uquote` package.

v1.0 (2016/06/28)

- Initial release.

10 Implementation

10.1 Required packages

The `upquote` package performs some font checks when it is loaded to determine whether `textcomp` is needed, but errors can result if the font is changed later in the preamble, so duplicate the package's font check at the end of the preamble. Also check for a package order issue with `lineno` and `csquotes`.

```
1 \RequirePackage{ifthen}
2 \RequirePackage{etoolbox}
3 \RequirePackage{fancyvrb}
4 \RequirePackage{upquote}
5 \AtEndPreamble{%
6   \ifx\encodingdefault\upquote@OTone
7     \ifx\ttdefault\upquote@cmtt\else\RequirePackage{textcomp}\fi
8   \else
9     \RequirePackage{textcomp}
10  \fi}
11 \RequirePackage{lineno}
12 \ifpackageloaded{csquotes}%
13   {\PackageWarning{fvextra}{csquotes should be loaded after fvextra, %
14     to avoid a warning from the lineno package}}{}
```

10.2 Utility macros

`\FV@Space@ifx` Macro for testing if a `\let` token is `\FV@Space` with `\ifx`. The space will be active and defined as `\FV@Space`.

```
15 \def\FV@Space@ifx{\FV@Space}
```

`\FV@Tab@ifx` Macro for testing if a `\let` token is `\FV@Tab` with `\ifx`. The tab will be active and defined as `\FV@Tab`.

```
16 \def\FV@Tab@ifx{\FV@Tab}
```

10.3 Hooks

`\FV@FormattingPrepHook` This is a hook for extending `\FV@FormattingPrep`. `\FV@FormattingPrep` is inside a group, before the beginning of processing, so it is a good place to add extension code. This hook is used for such things as tweaking math mode behavior and preparing for `breakbefore` and `breakafter`.

```
17 \let\FV@FormattingPrepHook\@empty
18 \expandafter\def\expandafter\FV@FormattingPrep\expandafter{%
19   \expandafter\FV@FormattingPrepHook\FV@FormattingPrep}
```

`\FV@PygmentsHook` This is a hook for turning on Pygments-related features for packages like `minted` and `pythontex` (section 10.8). It needs to be the first thing in `\FV@FormattingPrepHook`, since it will potentially affect some of the later things in the hook. It is activated by `\VerbatimPygments`.

```

20 \let\FV@PygmentsHook\relax
21 \g@addto@macro\FV@FormattingPrepHook{\FV@PygmentsHook}

```

10.4 Escaped characters

`\FV@EscChars` Define versions of common escaped characters that reduce to raw characters. This is useful, for example, when working with text that is almost verbatim, but was captured in such a way that some escapes were unavoidable.

```

22 \edef\FV@hashchar{\string#}
23 \edef\FV@dollarchar{\string$}
24 \edef\FV@ampchar{\string&}
25 \edef\FV@underscorechar{\string_}
26 \edef\FV@tildechar{\string~}
27 \edef\FV@leftsquarebracket{\string[]}
28 \edef\FV@rightsquarebracket{\string[]}]
29 \newcommand{\FV@EscChars}{%
30   \let\#\FV@hashchar
31   \let%\FV@percentchar
32   \let{\FV@charlb
33   \let}\FV@charrb
34   \let$\FV@dollarchar
35   \let&\FV@ampchar
36   \let_\FV@underscorechar
37   \let\\FV@backslashchar
38   \let~\FV@tildechar
39   \let~\FV@tildechar
40   \let\[FV@leftsquarebracket
41   \let\]FV@rightsquarebracket
42 } %$ <- highlighting

```

10.5 Patches

10.5.1 Visible spaces

`\FancyVerbSpace` The default definition of visible spaces (`showspaces=true`) could allow font commands to escape under some circumstances, depending on how it is used:

```
{\catcode'\ =12 \gdef\FancyVerbSpace{\tt }}
```

The command is redefined in more robust and standard L^AT_EX form.

```
43 \def\FancyVerbSpace{\textvisiblespace}
```

10.5.2 obeytabs with visible tabs and with tabs inside macro arguments

`\FV@TrueTab` governs tab appearance when `obeytabs=true` and `showtabs=true`. It is redefined so that symbols with flexible width, such as `\rightarrowfill`, will work as expected. In the original `fancyvrb` definition, `\kern\@tempdima\hbox to\z@{...}`. The `\kern` is removed and instead the `\hbox` is given the width `\@tempdima`.

`\FV@TrueTab` and related macros are also modified so that they function for tabs inside macro arguments when `obeytabs=true` (inside curly braces `{}` with their normal meaning, when using `commandchars`, etc.). The `fancyvrb` implementation of tab expansion assumes that tabs are never inside a group; when a group that contains a tab is present, the entire line typically vanishes. The new implementation keeps the `fancyvrb` behavior exactly for tabs outside groups; they are perfectly expanded to tab stops. Tabs inside groups cannot be perfectly expanded to tab stops, at least not using the `fancyvrb` approach. Instead, when `fvextra` encounters a run of whitespace characters (tabs and possibly spaces), it makes the assumption that the nearest tab stop was at the beginning of the run. This gives the correct behavior if the whitespace characters are leading indentation that happens to be within a macro. Otherwise, it will typically not give correct tab expansion—but at least the entire line will not be discarded, and the run of whitespace will be represented, even if imperfectly.

A general solution to tab expansion may be possible, but will almost certainly require multiple compiles, perhaps even one compile (or more) per tab. The `zref` package provides a `\zsavaposx` macro that stores the current x position on the page for subsequent compiles. This macro, or a similar macro from another package, could be used to establish a reference point at the beginning of each line. Then each run of whitespace that contains a tab could have a reference point established at its start, and tabs could be expanded based on the distance between the start of the run and the start of the line. Such an approach would allow the first run of whitespace to measure its distance from the start of the line on the 2nd compile (once both reference points were established), so it would be able to expand the first run of whitespace correctly on the 3rd compile. That would allow a second run of whitespace to definitely establish its starting point on the 3rd compile, which would allow it to expand correctly on the 4th compile. And so on. Thus, while it should be possible to perform completely correct tab expansion with such an approach, it will in general require at least 4 compiles to do better than the current approach. Furthermore, the sketch of the algorithm provided so far does not include any complications introduced by line breaking. In the current approach, it is necessary to determine how each tab would be expanded in the absence of line breaking, save all tab widths, and then expand using saved widths during the actual typesetting with line breaking.

`FV@TrueTabGroupLevel` Counter for keeping track of the group level (`\currentgrouplevel`) at the very beginning of a line, inside `\FancyVerbFormatLine` but outside `\FancyVerbFormatText`, which is where the tab expansion macro is invoked. This allows us to determine whether we are in a group, and expand tabs accordingly.

```
44 \newcounter{FV@TrueTabGroupLevel}
```

`\FV@@ObeyTabs` The `fancyvrb` macro responsible for tab expansion is modified so that it can handle tabs inside groups, even if imperfectly. We need to use a special version of the space, `\FV@Space@ObeyTabs`, that within a group will capture all following spaces or tabs and then insert them with tab expansion based on the beginning of the run of whitespace. We need to record the current group level, but then increment it by 1 because all comparisons will be performed within the `\hbox{...}`.

```

45 \def\FV@@ObeyTabs#1{%
46   \let\FV@Space@Orig\FV@Space
47   \let\FV@Space\FV@Space@ObeyTabs
48   \setcounter{FV@TrueTabGroupLevel}{\the\currentgrouplevel}%
49   \addtocounter{FV@TrueTabGroupLevel}{1}%
50   \setbox\FV@TabBox=\hbox{#1}\box\FV@TabBox
51   \let\FV@Space\FV@Space@Orig}

```

`\FV@TrueTab` Version that follows `fancyvrb` if not in a group and takes another approach otherwise.

```

52 \def\FV@TrueTab{%
53   \ifnum\value{FV@TrueTabGroupLevel}=\the\currentgrouplevel\relax
54     \expandafter\FV@TrueTab@NoGroup
55   \else
56     \expandafter\FV@TrueTab@Group
57   \fi}

```

`\FV@TrueTabSaveWidth` When linebreaking is in use, the `fancyvrb` tab expansion algorithm cannot be used directly, since it involves `\hbox`, which doesn't allow for line breaks. In those cases, tab widths will be calculated for the case without breaks and saved, and then saved widths will be used in the actual typesetting. This macro is `\let` to width-saving code in those cases.

```

58 \let\FV@TrueTabSaveWidth\relax

```

`FV@TrueTabCounter` Counter for tracking saved tabs.

```

59 \newcounter{FV@TrueTabCounter}

```

`\FV@TrueTabSaveWidth@Save` Save the current tab width, then increment the tab counter. `\@tempdima` will hold the current tab width.

```

60 \def\FV@TrueTabSaveWidth@Save{%
61   \expandafter\xdef\csname FV@TrueTab:Width\arabic{FV@TrueTabCounter}\endcsname{%
62     \number\@tempdima}%
63   \stepcounter{FV@TrueTabCounter}}

```

`\FV@TrueTab@NoGroup` This follows the `fancyvrb` approach exactly, except for the `\hbox` to `\@tempdima` adjustment and the addition of `\FV@TrueTabSaveWidth`.

```

64 \def\FV@TrueTab@NoGroup{%
65   \egroup
66   \@tempdima=\FV@ObeyTabSize sp\relax
67   \@tempcnta=\wd\FV@TabBox
68   \advance\@tempcnta\FV@ObeyTabSize\relax
69   \divide\@tempcnta\@tempdima
70   \multiply\@tempdima\@tempcnta

```

```

71 \advance\@tempdima-\wd\FV@TabBox
72 \FV@TrueTabSaveWidth
73 \setbox\FV@TabBox=\hbox\bgroup
74 \unhbox\FV@TabBox\hbox to\@tempdima{\hss\FV@TabChar}}

FV@ObeyTabs@Whitespace@Tab In a group where runs of whitespace characters are collected, we need to keep track
of whether a tab has been found, so we can avoid expansion and the associated
\hbox for spaces without tabs.
75 \newboolean{FV@ObeyTabs@Whitespace@Tab}

\FV@TrueTab@Group If in a group, a tab should start collecting whitespace characters for later tab
expansion, beginning with itself. The collected whitespace will use \FV@Tab@ifx
and \FV@Space@ifx so that any \ifx comparisons performed later will behave as
expected. This shouldn't be strictly necessary, because \FancyVerbBreakStart
operates with saved tab widths rather than using the tab expansion code directly.
But it is safer in case any other unanticipated scanning is going on.
76 \def\FV@TrueTab@Group{%
77 \booltrue{FV@ObeyTabs@Whitespace@Tab}%
78 \gdef\FV@TmpWhitespace{\FV@Tab@ifx}%
79 \FV@ObeyTabs@ScanWhitespace}

\FV@Space@ObeyTabs Space treatment, like tab treatment, now depends on whether we are in a group,
because in a group we want to collect all runs of whitespace and then expand any
tabs.
80 \def\FV@Space@ObeyTabs{%
81 \ifnum\value{FV@TrueTabGroupLevel}=\the\currentgrouplevel\relax
82 \expandafter\FV@Space@ObeyTabs@NoGroup
83 \else
84 \expandafter\FV@Space@ObeyTabs@Group
85 \fi}

\FV@Space@ObeyTabs@NoGroup Fall back to normal space.
86 \def\FV@Space@ObeyTabs@NoGroup{\FV@Space@Orig}

\FV@Space@ObeyTabs@Group Make a note that no tabs have yet been encountered, store the current space, then
scan for following whitespace.
87 \def\FV@Space@ObeyTabs@Group{%
88 \boolfalse{FV@ObeyTabs@Whitespace@Tab}%
89 \gdef\FV@TmpWhitespace{\FV@Space@ifx}%
90 \FV@ObeyTabs@ScanWhitespace}

\FV@ObeyTabs@ScanWhitespace Collect whitespace until the end of the run, then process it. Proper lookahead
comparison requires \FV@Space@ifx and \FV@Tab@ifx.
91 \def\FV@ObeyTabs@ScanWhitespace{%
92 \@ifnextchar\FV@Space@ifx%
93 {\FV@TrueTab@CaptureWhitespace@Space}%
94 {\ifx\@let@token\FV@Tab@ifx
95 \expandafter\FV@TrueTab@CaptureWhitespace@Tab

```

```

96     \else
97         \expandafter\FV@ObeyTabs@ResolveWhitespace
98     \fi}}
99 \def\FV@TrueTab@CaptureWhitespace@Space#1{%
100 \g@addto@macro\FV@TmpWhitespace{\FV@Space@ifx}%
101 \FV@ObeyTabs@ScanWhitespace}
102 \def\FV@TrueTab@CaptureWhitespace@Tab#1{%
103 \booltrue\FV@ObeyTabs@Whitespace@Tab}%
104 \g@addto@macro\FV@TmpWhitespace{\FV@Tab@ifx}%
105 \FV@ObeyTabs@ScanWhitespace}

```

\FV@TrueTab@Group@Expand Yet another tab definition, this one for use in the actual expansion of tabs in whitespace. This uses the fancyvrb algorithm, but only over a restricted region known to contain no groups.

```

106 \newbox\FV@TabBox@Group
107 \def\FV@TrueTab@Group@Expand{%
108     \egroup
109     \@tempdima=\FV@ObeyTabSize sp\relax
110     \@tempcnta=\wd\FV@TabBox@Group
111     \advance\@tempcnta\FV@ObeyTabSize\relax
112     \divide\@tempcnta\@tempdima
113     \multiply\@tempdima\@tempcnta
114     \advance\@tempdima-\wd\FV@TabBox@Group
115     \FV@TrueTabSaveWidth
116     \setbox\FV@TabBox@Group=\hbox\bgroup
117         \unhbox\FV@TabBox@Group\hbox to\@tempdima{\hss\FV@TabChar}}

```

\FV@ObeyTabs@ResolveWhitespace Need to make sure the right definitions of the space and tab are in play here. Only do tab expansion, with the associated \hbox, if a tab is indeed present.

```

118 \def\FV@ObeyTabs@ResolveWhitespace{%
119     \let\FV@Space\FV@Space@Orig
120     \let\FV@Tab\FV@TrueTab@Group@Expand
121     \expandafter\FV@ObeyTabs@ResolveWhitespace@i\expandafter{\FV@TmpWhitespace}%
122     \let\FV@Space\FV@Space@ObeyTabs
123     \let\FV@Tab\FV@TrueTab}
124 \def\FV@ObeyTabs@ResolveWhitespace@i#1{%
125     \ifbool\FV@ObeyTabs@Whitespace@Tab}%
126     {\setbox\FV@TabBox@Group=\hbox{#1}\box\FV@TabBox@Group}%
127     {#1}}

```

10.5.3 Spacing in math mode

\FancyVerbMathSpace \FV@Space is defined as either a non-breaking space or a visible representation of a space, depending on the option `showspaces`. Neither option is desirable when typeset math is included within verbatim content, because spaces will not be discarded as in normal math mode. Define a space for math mode.

```

128 \def\FancyVerbMathSpace{ }

```

\FV@SetupMathSpace Define a macro that will activate math spaces, then add it to an fvextra hook.

```

129 \def\FV@SetupMathSpace{%
130   \everymath\expandafter{\the\everymath\let\FV@Space\FancyVerbMathSpace}}
131 \g@addto@macro\FV@FormattingPrepHook{\FV@SetupMathSpace}

```

10.5.4 Fonts and symbols in math mode

The single quote (') does not become `\prime` when typeset math is included within verbatim content, due to the definition of the character in `\@noligs`. This patch adds a new definition of the character in math mode, inspired by <http://tex.stackexchange.com/q/223876/10742>. It also redefines other characters in `\@noligs` to behave normally within math mode and switches the default font within math mode, so that `amsmath`'s `\text` will work as expected.

```

\FV@pr@m@s Define a version of \pr@m@s from latex.ltx that works with active '. In verbatim
              contexts, ' is made active by \@noligs.

132 \begingroup
133 \catcode'\=' \active
134 \catcode'\^=7
135 \gdef\FV@pr@m@s{%
136   \ifx'\@let@token
137     \expandafter\pr@@@s
138   \else
139     \ifx^\@let@token
140       \expandafter\expandafter\expandafter\pr@@@t
141     \else
142       \egroup
143     \fi
144   \fi}
145 \endgroup

\FV@SetupMathFont Set the font back to default from the verbatim font.

146 \def\FV@SetupMathFont{%
147   \everymath\expandafter{\the\everymath\fontfamily{\familydefault}\selectfont}}
148 \g@addto@macro\FV@FormattingPrepHook{\FV@SetupMathFont}

\FV@SetupMathLigs Make all characters in \@noligs behave normally, and switch to \FV@pr@m@s. The
                  relevant definition from latex.ltx:

\def\verbatim@nolig@list{\do\'\do\<\do\>\do\,\do\''\do\~}

149 \def\FV@SetupMathLigs{%
150   \everymath\expandafter{%
151     \the\everymath
152     \let\pr@m@s\FV@pr@m@s
153     \begingroup\lccode'\~=''\lowercase{\endgroup\def~}{'}%
154     \ifmmode\expandafter\active@math@prime\else'\fi}%
155     \begingroup\lccode'\~=''\lowercase{\endgroup\def~}{'}%
156     \begingroup\lccode'\~='<\lowercase{\endgroup\def~}{<}%
157     \begingroup\lccode'\~='>\lowercase{\endgroup\def~}{>}%
158     \begingroup\lccode'\~='\' \lowercase{\endgroup\def~}{'}%

```

```

159     \begingroup\lccode'\~='\-\lowercase{\endgroup\def~}{-}%
160   }%
161 }
162 \g@addto@macro\FV@FormattingPrepHook{\FV@SetupMathLigs}

```

10.5.5 Orphaned label

`\FV@BeginListFrame@Lines` When `frame=lines` is used with a label, the label can be orphaned. This overwrites the default definition to add `\penalty\@M`. The fix is attributed to <http://tex.stackexchange.com/a/168021/10742>.

```

163 \def\FV@BeginListFrame@Lines{%
164   \begingroup
165   \lineskip\z@skip
166   \FV@SingleFrameLine{\z@}%
167   \kern-0.5\baselineskip\relax
168   \baselineskip\z@skip
169   \kern\FV@FrameSep\relax
170   \penalty\@M
171   \endgroup}

```

10.5.6 rulecolor and fillcolor

The `rulecolor` and `fillcolor` options are redefined so that they accept color names directly, rather than requiring `\color{<color_name>}`. The definitions still allow the old usage.

`rulecolor`

```

172 \define@key{FV}{rulecolor}{%
173   \ifstrepty{#1}%
174     {\let\FancyVerbRuleColor\relax}%
175     {\ifstrequal{#1}{none}%
176       {\let\FancyVerbRuleColor\relax}%
177       {\def\@tempa{#1}%
178         \FV@KVProcess@RuleColor#1\FV@Undefined}}}
179 \def\FV@KVProcess@RuleColor#1#2\FV@Undefined{%
180   \ifx#1\color
181   \else
182     \expandafter\def\expandafter\@tempa\expandafter{%
183       \expandafter\color\expandafter{\@tempa}}%
184   \fi
185   \let\FancyVerbRuleColor\@tempa}
186 \fvset{rulecolor=none}

```

`fillcolor`

```

187 \define@key{FV}{fillcolor}{%
188   \ifstrepty{#1}%
189     {\let\FancyVerbFillColor\relax}%
190     {\ifstrequal{#1}{none}%
191       {\let\FancyVerbFillColor\relax}%

```



```

192     {\def\@tempa{#1}%
193      \FV@KVProcess@FillColor#1\FV@Undefined}}
194 \def\FV@KVProcess@FillColor#1#2\FV@Undefined{%
195   \ifx#1\color
196   \else
197     \expandafter\def\expandafter\@tempa\expandafter{%
198       \expandafter\color\expandafter{\@tempa}}%
199   \fi
200   \let\FancyVerbFillColor\@tempa}
201 \fvset{fillcolor=none}

```

10.6 Extensions

10.6.1 New options requiring minimal implementation

linenos fancyvrb allows line numbers via the options `numbers=left` and `numbers=right`. This creates a `linenos` key that is essentially an alias for `numbers=left`.

```

202 \define@booleankey{FV}{linenos}%
203   {\@nameuse{FV@Numbers@left}}{\@nameuse{FV@Numbers@none}}

```

tab Redefine `\FancyVerbTab`.

```

204 \define@key{FV}{tab}{\def\FancyVerbTab{#1}}

```

tabcolor Set tab color, or allow it to adjust to surroundings (the default fancyvrb behavior). This involves re-creating the `showtabs` option to add `\FV@TabColor`.

```

205 \define@key{FV}{tabcolor}%
206   {\ifstrepty{#1}%
207     {\let\FV@TabColor\relax}%
208     {\ifstrequal{#1}{none}%
209       {\let\FV@TabColor\relax}%
210       {\def\FV@TabColor{\textcolor{#1}}}}}
211 \define@booleankey{FV}{showtabs}%
212   {\def\FV@TabChar{\FV@TabColor{\FancyVerbTab}}}%
213   {\let\FV@TabChar\relax}
214 \fvset{tabcolor=none, showtabs=false}

```

space Redefine `\FancyVerbSpace`.

```

215 \define@key{FV}{space}{\def\FancyVerbSpace{#1}}

```

spacecolor Set space color, or allow it to adjust to surroundings (the default fancyvrb behavior). This involves re-creating the `showspaces` option to add `\FV@SpaceColor`.

```

216 \define@key{FV}{spacecolor}%
217   {\ifstrepty{#1}%
218     {\let\FV@SpaceColor\relax}%
219     {\ifstrequal{#1}{none}%
220       {\let\FV@SpaceColor\relax}%
221       {\def\FV@SpaceColor{\textcolor{#1}}}}}
222 \define@booleankey{FV}{showspaces}%
223   {\def\FV@Space{\FV@SpaceColor{\FancyVerbSpace}}}%

```

```

224 {\def\FV@Space{\ }}
225 \fvset{spacecolor=none, showspaces=false}

mathescape Give $, ^, and _ their normal catcodes to allow normal typeset math.
226 \define@booleankey{FV}{mathescape}%
227 {\let\FancyVerbMathEscape\FV@MathEscape}%
228 {\let\FancyVerbMathEscape\relax}
229 \def\FV@MathEscape{\catcode'\$=3\catcode'\^=7\catcode'\_ =8\relax}
230 \FV@AddToHook\FV@CatCodesHook\FancyVerbMathEscape
231 \fvset{mathescape=false}

curlyquotes Let ` and ' produce curly quotation marks ‘ and ’ rather than the backtick and
typewriter single quotation mark produced by default via upquote.
232 \newbool{FV@CurlyQuotes}
233 \define@booleankey{FV}{curlyquotes}%
234 {\booltrue{FV@CurlyQuotes}}%
235 {\boolfalse{FV@CurlyQuotes}}
236 \def\FancyVerbCurlyQuotes{%
237   \ifbool{FV@CurlyQuotes}%
238     {\expandafter\def\expandafter\@noligs\expandafter{\@noligs
239       \begingroup\lccode'\~=''\lowercase{\endgroup\def~}{'}%
240       \begingroup\lccode'\~=''\lowercase{\endgroup\def~}{'}}}%
241     {}}
242 \g@addto@macro\FV@FormattingPrepHook{\FancyVerbCurlyQuotes}
243 \fvset{curlyquotes=false}

```

10.6.2 Formatting with `\FancyVerbFormatLine`, `\FancyVerbFormatText`, and `\FancyVerbHighlightLine`

`fancyvrb` defines `\FancyVerbFormatLine`, which defines the formatting for each line. The introduction of line breaks introduces an issue for `\FancyVerbFormatLine`. Does it format the entire line, including any whitespace in the margins or behind line break symbols (that is, is it outside the `\parbox` in which the entire line is wrapped when breaking is active)? Or does it only format the text part of the line, only affecting the actual characters (inside the `\parbox`)? Since both might be desirable, `\FancyVerbFormatLine` is assigned to the entire line, and a new macro `\FancyVerbFormatText` is assigned to the text, within the `\parbox`.

An additional complication is that the `fancyvrb` documentation says that the default value is `\def\FancyVerbFormatLine#1{#1}`. But the actual default is `\def\FancyVerbFormatLine#1{\FV@ObeyTabs{#1}}`. That is, `\FV@ObeyTabs` needs to operate directly on the line to handle tabs. As a result, *all* `fancyvrb` commands that involve `\FancyVerbFormatLine` are patched, so that `\def\FancyVerbFormatLine#1{#1}`.

An additional macro `\FancyVerbHighlightLine` is added between `\FancyVerbFormatLine` and `\FancyVerbFormatText`. This is used to highlight selected lines (section 10.6.4). It is inside `\FancyVerbHighlightLine` so that if `\FancyVerbHighlightLine` is used to provide a background color, `\FancyVerbHighlightLine` can override it.

`\FancyVerbFormatLine` Format the entire line, following the definition given in the `fancyvrb` documentation. Because this is formatting the entire line, using boxes works with line breaking.

```
244 \def\FancyVerbFormatLine#1{#1}
```

`\FancyVerbFormatText` Format only the text part of the line. Because this is inside all of the line breaking commands, using boxes here can conflict with line breaking.

```
245 \def\FancyVerbFormatText#1{#1}
```

`\FV@ListProcessLine@NoBreak` Redefined `\FV@ListProcessLine` in which `\FancyVerbFormatText` is added and tab handling is explicit. The `@NoBreak` suffix is added because `\FV@ListProcessLine` will be `\let` to either this macro or to `\FV@ListProcessLine@Break` depending on whether line breaking is enabled.

```
246 \def\FV@ListProcessLine@NoBreak#1{%
247   \hbox to \hsize{%
248     \kern\leftmargin
249     \hbox to \linewidth{%
250       \FV@LeftListNumber
251       \FV@LeftListFrame
252       \FancyVerbFormatLine{%
253         \FancyVerbHighlightLine{%
254           \FV@ObeyTabs{\FancyVerbFormatText{#1}}}\hss
255         \FV@RightListFrame
256         \FV@RightListNumber}%
257       \hss}}
```

`\FV@BProcessLine` Redefined `\FV@BProcessLine` in which `\FancyVerbFormatText` is added and tab handling is explicit.

```
258 \def\FV@BProcessLine#1{%
259   \hbox{\FancyVerbFormatLine{%
260     \FancyVerbHighlightLine{%
261       \FV@ObeyTabs{\FancyVerbFormatText{#1}}}\hss}}
```

10.6.3 Line numbering

Add several new line numbering options. `numberfirstline` always numbers the first line, regardless of `stepnumber`. `stepnumberfromfirst` numbers the first line, and then every line that differs from its number by a multiple of `stepnumber`. `stepnumberoffsetvalues` determines whether line numbers are always an exact multiple of `stepnumber` (the new default behavior) or whether there is an offset when `firstnumber` \neq 1 (the old default behavior). A new option `numbers=both` is created to allow line numbers on both left and right simultaneously.

```
FV@NumberFirstLine
262 \newbool{FV@NumberFirstLine}

numberfirstline
263 \define@booleankey{FV}{numberfirstline}%
264 {\booltrue{FV@NumberFirstLine}}%
```

```

265 {\boolfalse{FV@NumberFirstLine}}
266 \fvset{numberfirstline=false}

FV@StepNumberFromFirst
267 \newbool{FV@StepNumberFromFirst}

stepnumberfromfirst
268 \define@booleankey{FV}{stepnumberfromfirst}%
269 {\booltrue{FV@StepNumberFromFirst}}%
270 {\boolfalse{FV@StepNumberFromFirst}}
271 \fvset{stepnumberfromfirst=false}

FV@StepNumberOffsetValues
272 \newbool{FV@StepNumberOffsetValues}

stepnumberoffsetvalues
273 \define@booleankey{FV}{stepnumberoffsetvalues}%
274 {\booltrue{FV@StepNumberOffsetValues}}%
275 {\boolfalse{FV@StepNumberOffsetValues}}
276 \fvset{stepnumberoffsetvalues=false}

\FV@Numbers@left Redefine fancyvrb macro to account for numberfirstline, stepnumberfromfirst,
and stepnumberoffsetvalues. The \let\FancyVerbStartNum\@one is needed to
account for the case where firstline is never set, and defaults to zero (\z@).
277 \def\FV@Numbers@left{%
278   \let\FV@RightListNumber\relax
279   \def\FV@LeftListNumber{%
280     \ifx\FancyVerbStartNum\z@
281       \let\FancyVerbStartNum\@one
282     \fi
283     \ifbool{FV@StepNumberFromFirst}%
284       {\@tempcnta=\FV@CodeLineNo
285        \@tempcntb=\FancyVerbStartNum
286        \advance\@tempcntb\FV@StepNumber
287        \divide\@tempcntb\FV@StepNumber
288        \multiply\@tempcntb\FV@StepNumber
289        \advance\@tempcnta\@tempcntb
290        \advance\@tempcnta-\FancyVerbStartNum
291        \@tempcntb=\@tempcnta}%
292     {\ifbool{FV@StepNumberOffsetValues}%
293       {\@tempcnta=\FV@CodeLineNo
294        \@tempcntb=\FV@CodeLineNo}%
295       {\@tempcnta=\c@FancyVerbLine
296        \@tempcntb=\c@FancyVerbLine}}%
297     \divide\@tempcntb\FV@StepNumber
298     \multiply\@tempcntb\FV@StepNumber
299     \ifnum\@tempcnta=\@tempcntb
300       \if@FV@NumberBlankLines
301         \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%

```

```

302     \else
303         \ifx\FV@Line\empty
304         \else
305             \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
306         \fi
307     \fi
308 \else
309     \ifbool{FV@NumberFirstLine}{%
310         \ifnum\FV@CodeLineNo=\FancyVerbStartNum
311             \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
312         \fi}{}%
313 \fi}%
314 }

```

`\FV@Numbers@right` Redefine fancyvrb macro to account for numberfirstline, stepnumberfromfirst, and stepnumberoffsetvalues.

```

315 \def\FV@Numbers@right{%
316     \let\FV@LeftListNumber\relax
317     \def\FV@RightListNumber{%
318         \ifx\FancyVerbStartNum\z@
319             \let\FancyVerbStartNum\@ne
320         \fi
321         \ifbool{FV@StepNumberFromFirst}{%
322             {\@tempcnta=\FV@CodeLineNo
323              \@tempcntb=\FancyVerbStartNum
324              \advance\@tempcntb\FV@StepNumber
325              \divide\@tempcntb\FV@StepNumber
326              \multiply\@tempcntb\FV@StepNumber
327              \advance\@tempcnta\@tempcntb
328              \advance\@tempcnta-\FancyVerbStartNum
329              \@tempcntb=\@tempcnta}%
330             {\ifbool{FV@StepNumberOffsetValues}{%
331                 {\@tempcnta=\FV@CodeLineNo
332                  \@tempcntb=\FV@CodeLineNo}%
333                 {\@tempcnta=\c@FancyVerbLine
334                  \@tempcntb=\c@FancyVerbLine}}}%
335             \divide\@tempcntb\FV@StepNumber
336             \multiply\@tempcntb\FV@StepNumber
337             \ifnum\@tempcnta=\@tempcntb
338                 \ifFV@NumberBlankLines
339                     \hbox to\z@{\kern\FV@NumberSep\theFancyVerbLine\hss}%
340                 \else
341                     \ifx\FV@Line\empty
342                     \else
343                         \hbox to\z@{\kern\FV@NumberSep\theFancyVerbLine\hss}%
344                     \fi
345                 \fi
346             \else
347                 \ifbool{FV@NumberFirstLine}{%
348                     \ifnum\FV@CodeLineNo=\FancyVerbStartNum

```

```

349         \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
350         \fi}{}%
351     \fi}%
352 }

\FV@Numbers@both Define a new macro to allow numbers=both. This copies the definitions of
\FV@LeftListNumber and \FV@RightListNumber from \FV@Numbers@left and
\FV@Numbers@right, without the \relax's.
353 \def\FV@Numbers@both{%
354     \def\FV@LeftListNumber{%
355         \ifx\FancyVerbStartNum\z@
356             \let\FancyVerbStartNum\@ne
357         \fi
358         \ifbool{FV@StepNumberFromFirst}%
359             {\@tempcnta=\FV@CodeLineNo
360              \@tempcntb=\FancyVerbStartNum
361              \advance\@tempcntb\FV@StepNumber
362              \divide\@tempcntb\FV@StepNumber
363              \multiply\@tempcntb\FV@StepNumber
364              \advance\@tempcnta\@tempcntb
365              \advance\@tempcnta-\FancyVerbStartNum
366              \@tempcntb=\@tempcnta}%
367             {\ifbool{FV@StepNumberOffsetValues}%
368              {\@tempcnta=\FV@CodeLineNo
369               \@tempcntb=\FV@CodeLineNo}%
370              {\@tempcnta=\c@FancyVerbLine
371               \@tempcntb=\c@FancyVerbLine}}}%
372         \divide\@tempcntb\FV@StepNumber
373         \multiply\@tempcntb\FV@StepNumber
374         \ifnum\@tempcnta=\@tempcntb
375             \if@FV@NumberBlankLines
376                 \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
377             \else
378                 \ifx\FV@Line\empty
379                     \else
380                         \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
381                     \fi
382                 \fi
383             \else
384                 \ifbool{FV@NumberFirstLine}{%
385                     \ifnum\FV@CodeLineNo=\FancyVerbStartNum
386                         \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
387                     \fi}{}%
388                 \fi}%
389     \def\FV@RightListNumber{%
390         \ifx\FancyVerbStartNum\z@
391             \let\FancyVerbStartNum\@ne
392         \fi
393         \ifbool{FV@StepNumberFromFirst}%
394             {\@tempcnta=\FV@CodeLineNo

```

```

395 \@tempcntb=\FancyVerbStartNum
396 \advance\@tempcntb\FV@StepNumber
397 \divide\@tempcntb\FV@StepNumber
398 \multiply\@tempcntb\FV@StepNumber
399 \advance\@tempcnta\@tempcntb
400 \advance\@tempcnta-\FancyVerbStartNum
401 \@tempcntb=\@tempcnta}%
402 {\ifbool{FV@StepNumberOffsetValues}}%
403   {\@tempcnta=\FV@CodeLineNo
404     \@tempcntb=\FV@CodeLineNo}%
405   {\@tempcnta=\c@FancyVerbLine
406     \@tempcntb=\c@FancyVerbLine}}%
407 \divide\@tempcntb\FV@StepNumber
408 \multiply\@tempcntb\FV@StepNumber
409 \ifnum\@tempcnta=\@tempcntb
410   \ifFV@NumberBlankLines
411     \hbox to\z@{\kern\FV@NumberSep\theFancyVerbLine\hss}%
412   \else
413     \ifx\FV@Line\empty
414     \else
415       \hbox to\z@{\kern\FV@NumberSep\theFancyVerbLine\hss}%
416     \fi
417   \fi
418 \else
419   \ifbool{FV@NumberFirstLine}{%
420     \ifnum\FV@CodeLineNo=\FancyVerbStartNum
421       \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
422     \fi}{}%
423 \fi}%
424 }

```

10.6.4 Line highlighting or emphasis

This adds an option `highlightlines` that allows specific lines, or lines within a range, to be highlighted or otherwise emphasized.

```

highlightlines
\FV@HighlightLinesList 425 \define@key{FV}{highlightlines}{\def\FV@HighlightLinesList{#1}}%
426 \fvset{highlightlines=}

highlightcolor Define color for highlighting. The default is LightCyan. A good alternative for a
\FV@HighlightColor brighter color would be LemonChiffon.
427 \define@key{FV}{highlightcolor}{\def\FancyVerbHighlightColor{#1}}%
428 \let\FancyVerbHighlightColor\@empty
429 \ifcsname definecolor\endcsname
430 \ifx\definecolor\relax
431 \else
432 \definecolor{FancyVerbHighlightColor}{rgb}{0.878, 1, 1}
433 \fvset{highlightcolor=FancyVerbHighlightColor}

```

```

434 \fi\fi
435 \AtBeginDocument{%
436   \ifx\FancyVerbHighlightColor\@empty
437     \ifcsname definecolor\endcsname
438     \ifx\definecolor\relax
439     \else
440     \definecolor{FancyVerbHighlightColor}{rgb}{0.878, 1, 1}
441     \fvset{highlightcolor=FancyVerbHighlightColor}
442     \fi\fi
443 \fi}

```

`\FancyVerbHighlightLine` This is the entry macro into line highlighting. By default it should do nothing. It is always invoked between `\FancyVerbFormatLine` and `\FancyVerbFormatText`, so that it can provide a background color (won't interfere with line breaking) and can override any formatting provided by `\FancyVerbFormatLine`. It is `\let` to `\FV@HighlightLine` when highlighting is active.

```

444 \def\FancyVerbHighlightLine#1{#1}

```

`\FV@HighlightLine` This determines whether highlighting should be performed, and if so, which macro should be invoked.

```

445 \def\FV@HighlightLine#1{%
446   \@tempcnta=\c@FancyVerbLine
447   \@tempcntb=\c@FancyVerbLine
448   \ifcsname FV@HighlightLine:\number\@tempcnta\endcsname
449     \advance\@tempcntb\m@ne
450     \ifcsname FV@HighlightLine:\number\@tempcntb\endcsname
451     \advance\@tempcntb\tw@
452     \ifcsname FV@HighlightLine:\number\@tempcntb\endcsname
453     \let\FV@HighlightLine@Next\FancyVerbHighlightLineMiddle
454     \else
455     \let\FV@HighlightLine@Next\FancyVerbHighlightLineLast
456     \fi
457   \else
458     \advance\@tempcntb\tw@
459     \ifcsname FV@HighlightLine:\number\@tempcntb\endcsname
460     \let\FV@HighlightLine@Next\FancyVerbHighlightLineFirst
461     \else
462     \let\FV@HighlightLine@Next\FancyVerbHighlightLineSingle
463     \fi
464   \fi
465 \else
466   \let\FV@HighlightLine@Next\FancyVerbHighlightLineNormal
467 \fi
468 \FV@HighlightLine@Next{#1}%
469 }

```

`\FancyVerbHighlightLineNormal` A normal line that is not highlighted or otherwise emphasized. This could be redefined to de-emphasize the line.

```

470 \def\FancyVerbHighlightLineNormal#1{#1}

```


`\FV@TmpLength`

```
471 \newlength{\FV@TmpLength}
```

`\FancyVerbHighlightLineFirst` The first line in a multi-line range.

`\fboxsep` is set to zero so as to avoid indenting the line or changing inter-line spacing. It is restored to its original value inside to prevent any undesired effects. The `\strut` is needed to get the highlighting to be the appropriate height. The `\rlap` and `\hspace` make the `\colorbox` expand to the full `\linewidth`. Note that if `\fboxsep ≠ 0`, then we would want to use `\dimexpr\linewidth-2\fboxsep` or add `\hspace{-2\fboxsep}` at the end.

If this macro is customized so that the text cannot take up the full `\linewidth`, then adjustments may need to be made here or in the line breaking code to make sure that line breaking takes place at the appropriate location.

```
472 \def\FancyVerbHighlightLineFirst#1{%
473   \setlength{\FV@TmpLength}{\fboxsep}%
474   \setlength{\fboxsep}{0pt}%
475   \colorbox{\FancyVerbHighlightColor}{%
476     \setlength{\fboxsep}{\FV@TmpLength}%
477     \rlap{\strut#1}%
478     \hspace{\linewidth}}}
```

`\FancyVerbHighlightLineMiddle` A middle line in a multi-line range.

```
479 \let\FancyVerbHighlightLineMiddle\FancyVerbHighlightLineFirst
```

`\FancyVerbHighlightLineLast` The last line in a multi-line range.

```
480 \let\FancyVerbHighlightLineLast\FancyVerbHighlightLineFirst
```

`\FancyVerbHighlightLineSingle` A single line not in a multi-line range.

```
481 \let\FancyVerbHighlightLineSingle\FancyVerbHighlightLineFirst
```

`\FV@HighlightLinesPrep` Process the list of lines to highlight (if any). A macro is created for each line to be highlighted. During highlighting, a line is highlighted if the corresponding macro exists. All of the macro creating is ultimately within the current environment group so it stays local. `\FancyVerbHighlightLine` is `\let` to a version that will invoke the necessary logic.

```
482 \def\FV@HighlightLinesPrep{%
483   \ifx\FV@HighlightLinesList\@empty
484   \else
485     \let\FancyVerbHighlightLine\FV@HighlightLine
486     \expandafter\FV@HighlightLinesPrep@i
487   \fi}
488 \def\FV@HighlightLinesPrep@i{%
489   \renewcommand{\do}[1]{%
490     \ifstrempy{##1}{\FV@HighlightLinesParse##1-\FV@Undefined}}%
491   \expandafter\docsvlist\expandafter{\FV@HighlightLinesList}}
492 \def\FV@HighlightLinesParse#1-#2\FV@Undefined{%
493   \ifstrempy{#2}%
494   {\FV@HighlightLinesParse@Single{#1}}%
```

```

495   {\FV@HighlightLinesParse@Range{#1}#2\relax}}
496 \def\FV@HighlightLinesParse@Single#1{%
497   \expandafter\let\csname FV@HighlightLine:\detokenize{#1}\endcsname\relax}
498 \newcounter{FV@HighlightLinesStart}
499 \newcounter{FV@HighlightLinesStop}
500 \def\FV@HighlightLinesParse@Range#1#2-{%
501   \setcounter{FV@HighlightLinesStart}{#1}%
502   \setcounter{FV@HighlightLinesStop}{#2}%
503   \stepcounter{FV@HighlightLinesStop}%
504   \FV@HighlightLinesParse@Range@Loop}
505 \def\FV@HighlightLinesParse@Range@Loop{%
506   \ifnum\value{FV@HighlightLinesStart}<\value{FV@HighlightLinesStop}\relax
507     \expandafter\let\csname FV@HighlightLine:\arabic{FV@HighlightLinesStart}\endcsname\relax
508     \stepcounter{FV@HighlightLinesStart}%
509     \expandafter\FV@HighlightLinesParse@Range@Loop
510   \fi}
511 \g@addto@macro\FV@FormattingPrepHook{\FV@HighlightLinesPrep}

```

10.7 Line breaking

The following code adds automatic line breaking functionality to `fancyvrb`'s `Verbatim` environment. Automatic breaks may be inserted after spaces, or before or after specified characters. Breaking before or after specified characters involves scanning each line token by token to insert `\discretionary` at all potential break locations.

10.7.1 Options and associated macros

Begin by defining keys, with associated macros, bools, and dimens.

FV@BreakLines Turn line breaking on or off. The `\FV@ListProcessLine` from `fancyvrb` is `\let` to a (patched) version of the original or a version that supports line breaks.

```

512 \newboolean{FV@BreakLines}
513 \define@booleankey{FV}{breaklines}%
514   {\FV@BreakLinestrue
515     \let\FV@ListProcessLine\FV@ListProcessLine@Break}%
516   {\FV@BreakLinesfalse
517     \let\FV@ListProcessLine\FV@ListProcessLine@NoBreak}
518 \AtEndOfPackage{\fvset{breaklines=false}}

```

\FV@BreakIndent Indentation of continuation lines.

```

519 \newdimen\FV@BreakIndent
520 \define@key{FV}{breakindent}{\FV@BreakIndent=#1\relax}
521 \fvset{breakindent=0pt}

```

FV@BreakAutoIndent Auto indentation of continuation lines to indentation of original line. Adds to `\FV@BreakIndent`.

```

522 \newboolean{FV@BreakAutoIndent}
523 \define@booleankey{FV}{breakautoindent}%

```

```

524 {\FV@BreakAutoIndenttrue}{\FV@BreakAutoIndentfalse}
525 \fvset{breakautoindent=true}

```

\FancyVerbBreakSymbolLeft The left-hand symbol indicating a break. Since breaking is done in such a way that a left-hand symbol will often be desired while a right-hand symbol may not be, a shorthand option `breaksymbol` is supplied. This shorthand convention is continued with other options applying to the left-hand symbol.

```

526 \define@key{FV}{breaksymbolleft}{\def\FancyVerbBreakSymbolLeft{#1}}
527 \define@key{FV}{breaksymbol}{\fvset{breaksymbolleft=#1}}
528 \fvset{breaksymbolleft=\tiny\ensuremath{\hookrightarrow}}

```

\FancyVerbBreakSymbolRight The right-hand symbol indicating a break.

```

529 \define@key{FV}{breaksymbolright}{\def\FancyVerbBreakSymbolRight{#1}}
530 \fvset{breaksymbolright={}}

```

\FV@BreakSymbolSepLeft Separation of left break symbol from the text.

```

531 \newdimen\FV@BreakSymbolSepLeft
532 \define@key{FV}{breaksymbolsepleft}{\FV@BreakSymbolSepLeft=#1\relax}
533 \define@key{FV}{breaksymbolsep}{\fvset{breaksymbolsepleft=#1}}
534 \fvset{breaksymbolsepleft=1em}

```

\FV@BreakSymbolSepRight Separation of right break symbol from the text.

```

535 \newdimen\FV@BreakSymbolSepRight
536 \define@key{FV}{breaksymbolsepright}{\FV@BreakSymbolSepRight=#1\relax}
537 \fvset{breaksymbolsepright=1em}

```

\FV@BreakSymbolIndentLeft Additional left indentation to make room for the left break symbol.

```

538 \newdimen\FV@BreakSymbolIndentLeft
539 \settowidth{\FV@BreakSymbolIndentLeft}{\ttfamily xxxx}
540 \define@key{FV}{breaksymbolindentleft}{\FV@BreakSymbolIndentLeft=#1\relax}
541 \define@key{FV}{breaksymbolindent}{\fvset{breaksymbolindentleft=#1}}

```

\FV@BreakSymbolIndentRight Additional right indentation to make room for the right break symbol.

```

542 \newdimen\FV@BreakSymbolIndentRight
543 \settowidth{\FV@BreakSymbolIndentRight}{\ttfamily xxxx}
544 \define@key{FV}{breaksymbolindentright}{\FV@BreakSymbolIndentRight=#1\relax}

```

We need macros that contain the logic for typesetting the break symbols. By default, the symbol macros contain everything regarding the symbol and its typesetting, while these macros contain pure logic. The symbols should be wrapped in braces so that formatting commands (for example, `\tiny`) don't escape.

\FancyVerbBreakSymbolLeftLogic The left break symbol should only appear with continuation lines. Note that `linenumber` here refers to local line numbering for the broken line, *not* line numbering for all lines in the environment being typeset.

```

545 \newcommand{\FancyVerbBreakSymbolLeftLogic}[1]{%
546   \ifnum\value{linenumber}=1\relax\else{#1}\fi}

```

FancyVerbLineBreakLast We need a counter for keeping track of the local line number for the last segment of a broken line, so that we can avoid putting a right continuation symbol there. A line that is broken will ultimately be processed twice when there is a right continuation symbol, once to determine the local line numbering, and then again for actual insertion into the document.

```
547 \newcounter{FancyVerbLineBreakLast}
```

\FV@SetLineBreakLast Store the local line number for the last continuation line.

```
548 \newcommand{\FV@SetLineBreakLast}{%
549   \setcounter{FancyVerbLineBreakLast}{\value{linenumber}}}
```

\FancyVerbBreakSymbolRightLogic Only insert a right break symbol if not on the last continuation line.

```
550 \newcommand{\FancyVerbBreakSymbolRightLogic}[1]{%
551   \ifnum\value{linenumber}=\value{FancyVerbLineBreakLast}\relax\else{#1}\fi}
```

\FancyVerbBreakStart Macro that starts fine-tuned breaking (`breakanywhere`, `breakbefore`, `breakafter`) by examining a line token-by-token. Initially `\let` to `\relax`; later `\let` to `\FV@Break` as appropriate.

```
552 \let\FancyVerbBreakStart\relax
```

\FancyVerbBreakStop Macro that stops the fine-tuned breaking region started by `\FancyVerbBreakStart`. Initially `\let` to `\relax`; later `\let` to `\FV@EndBreak` as appropriate.

```
553 \let\FancyVerbBreakStop\relax
```

\FV@Break@Token Macro that controls token handling between `\FancyVerbBreakStart` and `\FancyVerbBreakStop`. Initially `\let` to `\relax`; later `\let` to `\FV@Break@AnyToken` or `\FV@Break@BeforeAfterToken` as appropriate. There is no need to `\let\FV@Break@Token\relax` when `breakanywhere`, `breakbefore`, and `breakafter` are not in use. In that case, `\FancyVerbBreakStart` and `\FancyVerbBreakStop` are `\let` to `\relax`, and `\FV@Break@Token` is never invoked.

```
554 \let\FV@Break@Token\relax
```

FV@BreakAnywhere Allow line breaking (almost) anywhere. Set `\FV@Break` and `\FV@EndBreak` to be used, and `\let \FV@Break@Token` to the appropriate macro.

```
555 \newboolean{FV@BreakAnywhere}
556 \define@booleankey{FV}{breakanywhere}%
557   {\FV@BreakAnywheretrue
558     \let\FancyVerbBreakStart\FV@Break
559     \let\FancyVerbBreakStop\FV@EndBreak
560     \let\FV@Break@Token\FV@Break@AnyToken}%
561   {\FV@BreakAnywheretruefalse
562     \let\FancyVerbBreakStart\relax
563     \let\FancyVerbBreakStop\relax}
564 \fvset{breakanywhere=false}
```

\FV@BreakBefore Allow line breaking (almost) anywhere, but only before specified characters.

```
565 \define@key{FV}{breakbefore}{%
```

```

566 \ifstrempy{#1}%
567 {\let\FV@BreakBefore\@empty
568 \let\FancyVerbBreakStart\relax
569 \let\FancyVerbBreakStop\relax}%
570 {\def\FV@BreakBefore{#1}%
571 \let\FancyVerbBreakStart\FV@Break
572 \let\FancyVerbBreakStop\FV@EndBreak
573 \let\FV@Break@Token\FV@Break@BeforeAfterToken}%
574 }
575 \fvset{breakbefore={}}

```

FV@BreakBeforeGroup Determine whether breaking before specified characters is always allowed before each individual character, or is only allowed before the first in a group of identical characters.

```

576 \newboolean{FV@BreakBeforeGroup}
577 \define@booleankey{FV}{breakbeforegroup}%
578 {\FV@BreakBeforeGrouptrue}%
579 {\FV@BreakBeforeGroupfalse}%
580 \fvset{breakbeforegroup=true}

```

\FV@BreakBeforePrep We need a way to break before characters if and only if they have been specified as breaking characters. It would be possible to do that via a nested conditional, but that would be messy. It is much simpler to create an empty macro whose name contains the character, and test for the existence of this macro. This needs to be done inside a `\begingroup...\endgroup` so that the macros do not have to be cleaned up manually. A good place to do this is in `\FV@FormattingPrep`, which is inside a group and before processing starts. The macro is added to `\FV@FormattingPrepHook`, which contains `fvextra` extensions to `\FV@FormattingPrep`, after `\FV@BreakAfterPrep` is defined below.

The procedure here is a bit roundabout. We need to use `\FV@EscChars` to handle character escapes, but the character redefinitions need to be kept local, requiring that we work within a `\begingroup...\endgroup`. So we loop through the breaking tokens and assemble a macro that will itself define character macros. Only this defining macro is declared global, and it contains *expanded* characters so that there is no longer any dependence on `\FV@EscChars`.

`\FV@BreakBeforePrep@PygmentsHook` allows additional break preparation for Pygments-based packages such as `minted` and `pythontex`. When Pygments highlights code, it converts some characters into macros; they do not appear literally. As a result, for breaking to occur correctly, breaking macros need to be created for these character macros and not only for the literal characters themselves.

A pdfTeX-compatible version for working with UTF-8 is defined later, and `\FV@BreakBeforePrep` is `\let` to it under pdfTeX as necessary.

```

581 \def\FV@BreakBeforePrep{%
582 \ifx\FV@BreakBefore\@empty\relax
583 \else
584 \gdef\FV@BreakBefore@Def{}%
585 \begingroup

```

```

586 \def\FV@BreakBefore@Process##1##2\FV@Undefined{%
587 \expandafter\FV@BreakBefore@Process@i\expandafter{##1}%
588 \expandafter\ifx\expandafter\relax\detokenize{##2}\relax
589 \else
590 \FV@BreakBefore@Process##2\FV@Undefined
591 \fi
592 }%
593 \def\FV@BreakBefore@Process@i##1{%
594 \g@addto@macro\FV@BreakBefore@Def{%
595 \@namedef{FV@BreakBefore@Token\detokenize{##1}}{}}%
596 }%
597 \FV@EscChars
598 \expandafter\FV@BreakBefore@Process\FV@BreakBefore\FV@Undefined
599 \endgroup
600 \FV@BreakBefore@Def
601 \FV@BreakBeforePrep@PygmentsHook
602 \fi
603 }
604 \let\FV@BreakBeforePrep@PygmentsHook\relax

```

\FV@BreakAfter Allow line breaking (almost) anywhere, but only after specified characters.

```

605 \define@key{FV}{breakafter}{%
606 \ifstrempy{#1}%
607 {\let\FV@BreakAfter@empty
608 \let\FancyVerbBreakStart\relax
609 \let\FancyVerbBreakStop\relax}%
610 {\def\FV@BreakAfter{#1}%
611 \let\FancyVerbBreakStart\FV@Break
612 \let\FancyVerbBreakStop\FV@EndBreak
613 \let\FV@Break@Token\FV@Break@BeforeAfterToken}%
614 }
615 \fvset{breakafter={}}

```

FV@BreakAfterGroup Determine whether breaking after specified characters is always allowed after each individual character, or is only allowed after groups of identical characters.

```

616 \newboolean{FV@BreakAfterGroup}
617 \define@booleankey{FV}{breakaftergroup}%
618 {\FV@BreakAfterGrouptrue}%
619 {\FV@BreakAfterGroupfalse}%
620 \fvset{breakaftergroup=true}

```

\FV@BreakAfterPrep This is the `breakafter` equivalent of `\FV@BreakBeforePrep`. It is also used within `\FV@FormattingPrep`. The order of `\FV@BreakBeforePrep` and `\FV@BreakAfterPrep` is important; `\FV@BreakAfterPrep` must always be second, because it checks for conflicts with `breakbefore`.

A pdfTeX-compatible version for working with UTF-8 is defined later, and `\FV@BreakAfterPrep` is `\let` to it under pdfTeX as necessary.

```

621 \def\FV@BreakAfterPrep{%
622 \ifx\FV@BreakAfter@empty\relax

```

```

623 \else
624   \gdef\FV@BreakAfter@Def{%
625     \begingroup
626     \def\FV@BreakAfter@Process##1##2\FV@Undefined{%
627       \expandafter\FV@BreakAfter@Process@i\expandafter{##1}%
628       \expandafter\ifx\expandafter\relax\detokenize{##2}\relax
629     \else
630       \FV@BreakAfter@Process##2\FV@Undefined
631     \fi
632   }%
633   \def\FV@BreakAfter@Process@i##1{%
634     \ifcsname FV@BreakBefore@Token\detokenize{##1}\endcsname
635     \ifthenelse{\boolean{FV@BreakBeforeGroup}}{%
636       {\ifthenelse{\boolean{FV@BreakAfterGroup}}{%
637         {}%
638         {\PackageError{fvextra}%
639           {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1}"%
640             {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1}}}%
641         {\ifthenelse{\boolean{FV@BreakAfterGroup}}{%
642           {\PackageError{fvextra}%
643             {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1}"%
644               {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1}}}%
645           {}}%
646         \fi
647       \g@addto@macro\FV@BreakAfter@Def{%
648         \@namedef{FV@BreakAfter@Token\detokenize{##1}}{}}}%
649     }%
650     \FV@EscChars
651     \expandafter\FV@BreakAfter@Process\FV@BreakAfter\FV@Undefined
652     \endgroup
653     \FV@BreakAfter@Def
654     \FV@BreakAfterPrep@PygmentsHook
655   \fi
656 }
657 \let\FV@BreakAfterPrep@PygmentsHook\relax

```

Now that `\FV@BreakBeforePrep` and `\FV@BreakAfterPrep` are defined, add them to `\FV@FormattingPrepHook`, which is the `fvextra` extension to `\FV@FormattingPrep`. The ordering here is important, since `\FV@BreakAfterPrep` contains compatibility checks with `\FV@BreakBeforePrep`, and thus must be used after it. Also, we have to check for the pdfTeX engine with inputenc using UTF-8, and use the UTF macros instead when that is the case.

```

658 \g@addto@macro\FV@FormattingPrepHook{%
659   \ifcsname pdfmatch\endcsname
660   \ifx\pdfmatch\relax
661   \else
662     \ifcsname inputencodingname\endcsname
663     \ifx\inputencodingname\relax
664     \else

```

```

665     \ifdefstring{\inputencodingname}{utf8}%
666     {\let\FV@BreakBeforePrep\FV@BreakBeforePrep@UTF
667      \let\FV@BreakAfterPrep\FV@BreakAfterPrep@UTF}%
668     }%
669     \fi\fi
670 \fi\fi
671 \FV@BreakBeforePrep\FV@BreakAfterPrep}

```

`\FancyVerbBreakAnywhereSymbolPre` The pre-break symbol for breaks introduced by `breakanywhere`. That is, the symbol before breaks that occur between characters, rather than at spaces.

```

672 \define@key{FV}{breakanywheresymbolpre}{%
673   \ifstrempy{#1}%
674   {\def\FancyVerbBreakAnywhereSymbolPre{}}%
675   {\def\FancyVerbBreakAnywhereSymbolPre{\hbox{#1}}}%
676   \fvset{breakanywheresymbolpre={\,\footnotesize\ensuremath{\_ \rfloor}}}%

```

`\FancyVerbBreakAnywhereSymbolPost` The post-break symbol for breaks introduced by `breakanywhere`.

```

677 \define@key{FV}{breakanywheresymbolpost}{%
678   \ifstrempy{#1}%
679   {\def\FancyVerbBreakAnywhereSymbolPost{}}%
680   {\def\FancyVerbBreakAnywhereSymbolPost{\hbox{#1}}}%
681   \fvset{breakanywheresymbolpost={}}%

```

`\FancyVerbBreakBeforeSymbolPre` The pre-break symbol for breaks introduced by `breakbefore`.

```

682 \define@key{FV}{breakbeforesymbolpre}{%
683   \ifstrempy{#1}%
684   {\def\FancyVerbBreakBeforeSymbolPre{}}%
685   {\def\FancyVerbBreakBeforeSymbolPre{\hbox{#1}}}%
686   \fvset{breakbeforesymbolpre={\,\footnotesize\ensuremath{\_ \rfloor}}}%

```

`\FancyVerbBreakBeforeSymbolPost` The post-break symbol for breaks introduced by `breakbefore`.

```

687 \define@key{FV}{breakbeforesymbolpost}{%
688   \ifstrempy{#1}%
689   {\def\FancyVerbBreakBeforeSymbolPost{}}%
690   {\def\FancyVerbBreakBeforeSymbolPost{\hbox{#1}}}%
691   \fvset{breakbeforesymbolpost={}}%

```

`\FancyVerbBreakAfterSymbolPre` The pre-break symbol for breaks introduced by `breakafter`.

```

692 \define@key{FV}{breakaftersymbolpre}{%
693   \ifstrempy{#1}%
694   {\def\FancyVerbBreakAfterSymbolPre{}}%
695   {\def\FancyVerbBreakAfterSymbolPre{\hbox{#1}}}%
696   \fvset{breakaftersymbolpre={\,\footnotesize\ensuremath{\_ \rfloor}}}%

```

`\FancyVerbBreakAfterSymbolPost` The post-break symbol for breaks introduced by `breakafter`.

```

697 \define@key{FV}{breakaftersymbolpost}{%
698   \ifstrempy{#1}%
699   {\def\FancyVerbBreakAfterSymbolPost{}}%
700   {\def\FancyVerbBreakAfterSymbolPost{\hbox{#1}}}%
701   \fvset{breakaftersymbolpost={}}%

```


`\FancyVerbBreakAnywhereBreak` The macro governing breaking for `breakanywhere=true`.

```

702 \newcommand{\FancyVerbBreakAnywhereBreak}{%
703   \discretionary{\FancyVerbBreakAnywhereSymbolPre}%
704   {\FancyVerbBreakAnywhereSymbolPost}{}}

```

`\FancyVerbBreakBeforeBreak` The macro governing breaking for `breakbefore=true`.

```

705 \newcommand{\FancyVerbBreakBeforeBreak}{%
706   \discretionary{\FancyVerbBreakBeforeSymbolPre}%
707   {\FancyVerbBreakBeforeSymbolPost}{}}

```

`\FancyVerbBreakAfterBreak` The macro governing breaking for `breakafter=true`.

```

708 \newcommand{\FancyVerbBreakAfterBreak}{%
709   \discretionary{\FancyVerbBreakAfterSymbolPre}%
710   {\FancyVerbBreakAfterSymbolPost}{}}

```

10.7.2 Line breaking implementation

Helper macros

`\FV@LineBox` A box for saving a line of text, so that its dimensions may be determined and thus we may figure out if it needs line breaking.

```
711 \newsavebox{\FV@LineBox}
```

`\FV@LineIndentBox` A box for saving the indentation of code, so that its dimensions may be determined for use in auto-indentation of continuation lines.

```
712 \newsavebox{\FV@LineIndentBox}
```

`\FV@LineIndentChars` A macro for storing the indentation characters, if any, of a given line. For use in auto-indentation of continuation lines

```
713 \let\FV@LineIndentChars\@empty
```

`\FV@GetLineIndent` A macro that takes a line and determines the indentation, storing the indentation chars in `\FV@LineIndentChars`.

```

714 \def\FV@CleanRemainingChars#1\FV@Undefined{}
715 \def\FV@GetLineIndent{\afterassignment\FV@CheckIndentChar\let\FV@NextChar=}
716 \def\FV@CheckIndentChar{%
717   \ifx\FV@NextChar\FV@Undefined\relax
718     \let\FV@Next=\relax
719   \else
720     \ifx\FV@NextChar\FV@Space@ifx\relax
721       \g@addto@macro{\FV@LineIndentChars}{\FV@Space@ifx}%
722       \let\FV@Next=\FV@GetLineIndent
723     \else
724       \ifx\FV@NextChar\FV@Tab@ifx\relax
725         \g@addto@macro{\FV@LineIndentChars}{\FV@Tab@ifx}%
726         \let\FV@Next=\FV@GetLineIndent
727       \else
728         \let\FV@Next=\FV@CleanRemainingChars

```

```

729         \fi
730     \fi
731 \fi
732 \FV@Next
733 }

```

Tab expansion

The `fancyvrb` option `obeytabs` uses a clever algorithm involving boxing and unboxing to expand tabs based on tab stops rather than a fixed number of equivalent space characters. (See the definitions of `\FV@ObeyTabs` and `\FV@TrueTab` in section 10.5.2.) Unfortunately, since this involves `\hbox`, it interferes with the line breaking algorithm, and an alternative is required.

There are probably many ways tab expansion could be performed while still allowing line breaks. The current approach has been chosen because it is relatively straightforward and yields identical results to the case without line breaks. Line breaking involves saving a line in a box, and determining whether the box is too wide. During this process, if `obeytabs=true`, `\FV@TrueTabSaveWidth`, which is inside `\FV@TrueTab`, is `\let` to a version that saves the width of every tab in a macro. When a line is broken, all tabs within it will then use a variant of `\FV@TrueTab` that sequentially retrieves the saved widths. This maintains the exact behavior of the case without line breaks.

Note that the special version of `\FV@TrueTab` is based on the `fvextra` patched version of `\FV@TrueTab`, not on the original `\FV@TrueTab` defined in `fancyvrb`.

```

\FV@TrueTab@UseWidth Version of \FV@TrueTab that uses pre-computed tab widths.
734 \def\FV@TrueTab@UseWidth{%
735   \@tempdima=\csname FV@TrueTab:Width\arabic{FV@TrueTabCounter}\endcsname sp\relax
736   \stepcounter{FV@TrueTabCounter}%
737   \hbox to\@tempdima{\hss\FV@TabChar}}

```

Line scanning and break insertion macros

The strategy here is to scan a line token-by-token, and insert breaks at appropriate points. An alternative would be to make characters active, and have them expand to literal versions of themselves plus appropriate breaks. Both approaches have advantages and drawbacks. A catcode-based approach could work, but in general would require redefining some existing active characters to insert both appropriate breaks and their original definitions. The current approach works regardless of catcodes. It is also convenient for working with macros that expand to single characters, such as those created in highlighting code with `Pygments` (which is used by `minted` and `pythontex`). In that case, working with active characters would not be enough, and scanning for macros (or redefining them) is necessary. With the current approach, working with more complex macros is also straightforward. Adding support for line breaks within a macro simply requires wrapping macro contents with `\FancyVerbBreakStart...\FancyVerbBreakStop`. A catcode-based

approach could require `\scantokens` or a similar retokenization in some cases, but would have the advantage that in other cases no macro redefinition would be needed.

\FV@Break The entry macro for breaking lines, either anywhere or before/after specified characters. The current line (or argument) will be scanned token by token/group by group, and accumulated (with added potential breaks) in `\FV@TmpLine`. After scanning is complete, `\FV@TmpLine` will be inserted. It would be possible to insert each token/group into the document immediately after it is scanned, instead of accumulating them in a “buffer.” But that would interfere with macros. Even in the current approach, macros that take optional arguments are problematic.⁶ The last token is tracked with `\FV@LastToken`, to allow lookbehind when breaking by groups of identical characters. `\FV@LastToken` is `\let` to `\FV@Undefined` any time the last token was something that shouldn’t be compared against (for example, a non-empty group), and it is not reset whenever the last token may be ignored (for example, `{}`). When setting `\FV@LastToken`, it is vital always to use `\let\FV@LastToken=...` so that `\let\FV@LastToken==` will work (so that the equals sign = won’t break things).

The current definition of `\FV@Break@Token` is swapped for a UTF-8 compatible one under pdfTeX when necessary. The standard macros are defined next, since they make the algorithms simpler to understand. The more complex UTF variants are defined later. When swapping for the UTF macros, it is important to make sure that pdfTeX is indeed in use, that `inputenc` is indeed in use, and that the current encoding is UTF-8. The checks take into account the possibility of an errant `\ifx` test creating a previously non-existent macro and then `\letting` it to `\relax`.

```

738 \def\FV@Break{%
739   \def\FV@TmpLine{}%
740   \let\FV@LastToken=\FV@Undefined
741   \ifcsname pdfmatch\endcsname
742   \ifx\pdfmatch\relax
743   \else
744     \ifcsname inputencodingname\endcsname
745     \ifx\inputencodingname\relax
746     \else
747       \ifdefstring{\inputencodingname}{utf8}%
748       {\ifx\FV@Break@Token\FV@Break@AnyToken
749        \let\FV@Break@Token\FV@Break@AnyToken@UTF
750        \else
751          \ifx\FV@Break@Token\FV@Break@BeforeAfterToken
752          \let\FV@Break@Token\FV@Break@BeforeAfterToken@UTF
753          \fi
754          \fi}%
755       {}%
756     \fi\fi

```

⁶Through a suitable definition that tracks the current state and looks for square brackets, this might be circumvented. Then again, in verbatim contexts, macro use should be minimal, so the restriction to macros without optional arguments should generally not be an issue.

```

757 \fi\fi
758 \FV@Break@Scan
759 }

\FV@EndBreak
760 \def\FV@EndBreak{\FV@TmpLine}

\FV@Break@Scan Look ahead via \@ifnextchar. Don't do anything if we're at the end of the region
                to be scanned. Otherwise, invoke a macro to deal with what's next based on
                whether it is math, or a group, or something else.
                This and some following macros are defined inside of groups, to ensure proper
                catcodes.
761 \begingroup
762 \catcode'\$=3%
763 \gdef\FV@Break@Scan{%
764   \@ifnextchar\FV@EndBreak%
765   }%
766   {\ifx\@let@token$\relax
767     \let\FV@Break@Next\FV@Break@Math
768   \else
769     \ifx\@let@token\bgroup\relax
770       \let\FV@Break@Next\FV@Break@Group
771     \else
772       \let\FV@Break@Next\FV@Break@Token
773     \fi
774   \fi
775   \FV@Break@Next}%
776 }
777 \endgroup

\FV@Break@Math Grab an entire math span, and insert it into \FV@TmpLine. Due to grouping, this
                works even when math contains things like \text{$x$}. After dealing with the
                math span, continue scanning.
778 \begingroup
779 \catcode'\$=3%
780 \gdef\FV@Break@Math$#1${%
781   \g@addto@macro{\FV@TmpLine}{\$#1$}%
782   \let\FV@LastToken=\FV@Undefined
783   \FV@Break@Scan}
784 \endgroup

\FV@Break@Group Grab the group, and insert it into \FV@TmpLine (as a group) before continuing
                scanning.
785 \def\FV@Break@Group#1{%
786   \g@addto@macro{\FV@TmpLine}{\{#1\}}%
787   \ifstrempy{#1}{\}{\let\FV@LastToken=\FV@Undefined}%
788   \FV@Break@Scan}

```

`\FV@Break@AnyToken` Deal with breaking around any token. This doesn't break macros with *mandatory* arguments, because `\FancyVerbBreakAnywhereBreak` is inserted *before* the token. Groups themselves are added without any special handling. So a macro would end up right next to its original arguments, without anything being inserted. Optional arguments will cause this approach to fail; there is currently no attempt to identify them, since that is a much harder problem.

If it is ever necessary, it would be possible to create a more sophisticated version involving catcode checks via `\ifcat`. Something like this:

```

\begingroup
\catcode'\a=11%
\catcode'\+=12%
\gdef\FV@Break...
  \ifcat\noexpand#1a%
    \g@addto@macro{\FV@TmpLine}...
  \else
...
\endgroup

```

```

789 \def\FV@Break@AnyToken#1{%
790   \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakAnywhereBreak#1}%
791   \FV@Break@Scan}

```

`\FV@Break@BeforeAfterToken` Deal with breaking around only specified tokens. This is a bit trickier. We only break if a macro corresponding to the token exists. We also need to check whether the specified token should be grouped, that is, whether breaks are allowed between identical characters. All of this has to be written carefully so that nothing is accidentally inserted into the stream for future scanning.

Dealing with tokens followed by empty groups (for example, `\x{}`) is particularly challenging when we want to avoid breaks between identical characters. When a token is followed by a group, we need to save the current token for later reference (`\x` in the example), then capture and save the following group, and then—only if the group was empty—see if the following token is identical to the old saved token.

```

792 \def\FV@Break@BeforeAfterToken#1{%
793   \ifcsname FV@BreakBefore@Token\detokenize{#1}\endcsname
794     \let\FV@Break@Next\FV@Break@BeforeTokenBreak
795   \else
796     \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
797       \let\FV@Break@Next\FV@Break@AfterTokenBreak
798     \else
799       \let\FV@Break@Next\FV@Break@BeforeAfterTokenNoBreak
800     \fi
801   \fi
802   \FV@Break@Next{#1}%
803 }
804 \def\FV@Break@BeforeAfterTokenNoBreak#1{%
805   \g@addto@macro{\FV@TmpLine}{#1}%

```

```

806 \let\FV@LastToken=#1%
807 \FV@Break@Scan}
808 \def\FV@Break@BeforeTokenBreak#1{%
809 \ifthenelse{\boolean{FV@BreakBeforeGroup}}{%
810 {\ifx#1\FV@LastToken\relax
811 \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
812 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan
813 \def\FV@RescanToken{#1}%
814 \else
815 \g@addto@macro{\FV@TmpLine}{#1}%
816 \let\FV@Break@Next\FV@Break@Scan
817 \let\FV@LastToken=#1%
818 \fi
819 \else
820 \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
821 \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakBeforeBreak}%
822 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan
823 \def\FV@RescanToken{#1}%
824 \else
825 \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakBeforeBreak#1}%
826 \let\FV@Break@Next\FV@Break@Scan
827 \let\FV@LastToken=#1%
828 \fi
829 \fi}%
830 {\ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
831 \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakBeforeBreak}%
832 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan
833 \def\FV@RescanToken{#1}%
834 \else
835 \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakBeforeBreak#1}%
836 \let\FV@Break@Next\FV@Break@Scan
837 \let\FV@LastToken=#1%
838 \fi}%
839 \FV@Break@Next}
840 \def\FV@Break@BeforeTokenBreak@AfterRescan{%
841 \expandafter\FV@Break@AfterTokenBreak\FV@RescanToken}
842 \def\FV@Break@AfterTokenBreak#1{%
843 \let\FV@LastToken=#1%
844 \@ifnextchar\FV@Space@ifx%
845 {\g@addto@macro{\FV@TmpLine}{#1}\FV@Break@Scan}%
846 {\ifthenelse{\boolean{FV@BreakAfterGroup}}{%
847 {\ifx\@let@token#1\relax
848 \g@addto@macro{\FV@TmpLine}{#1}%
849 \let\FV@Break@Next\FV@Break@Scan
850 \else
851 \ifx\@let@token\bgroup\relax
852 \g@addto@macro{\FV@TmpLine}{#1}%
853 \let\FV@Break@Next\FV@Break@AfterTokenBreak@Group
854 \else
855 \g@addto@macro{\FV@TmpLine}{#1\FancyVerbBreakAfterBreak}%

```

```

856         \let\FV@Break@Next\FV@Break@Scan
857         \fi
858     \fi}%
859     {\g@addto@macro{\FV@TmpLine}{#1\FancyVerbBreakAfterBreak}%
860     \let\FV@Break@Next\FV@Break@Scan}%
861     \FV@Break@Next}%
862 }
863 \def\FV@Break@AfterTokenBreak@Group#1{%
864     \g@addto@macro{\FV@TmpLine}{#{1}}%
865     \ifstrempy{#1}%
866         {\let\FV@Break@Next\FV@Break@AfterTokenBreak@Group#i}%
867         {\let\FV@Break@Next\FV@Break@Scan\let\FV@LastToken=\FV@Undefined}%
868         \FV@Break@Next}
869 \def\FV@Break@AfterTokenBreak@Group#i{%
870     \ifnextchar\FV@LastToken%
871         {\FV@Break@Scan}%
872         {\g@addto@macro{\FV@TmpLine}{\FancyVerbBreakAfterBreak}%
873         \FV@Break@Scan}}

```

Line scanning and break insertion macros for pdfTeX with UTF-8

The macros above work with the XeTeX and LuaTeX engines and are also fine for pdfTeX with 8-bit character encodings. Unfortunately, pdfTeX works with multi-byte UTF-8 code points at the byte level, making things significantly trickier. The code below re-implements the macros in a manner compatible with the `inputenc` package with option `utf8`. Note that there is no attempt for compatibility with `utf8x`; `utf8` has been significantly improved in recent years and should be sufficient in the vast majority of cases. And implementing variants for `utf8` was already sufficiently painful.

All of the UTF macros are only needed with pdfTeX, so they are created conditionally, inspired by the approach of the `iftex` package. The pdfTeX test deals with the possibility that a previous test using `\ifx` rather than the cleaner `\ifcsname` has already been performed.

```

874 \ifcsname pdfmatch\endcsname
875 \ifx\pdfmatch\relax
876 \else

```

```

\FV@UTF@two@octets
\FV@UTF@three@octets
\FV@UTF@four@octets

```

These are variants of the `utf8.def` macros that capture all bytes of a multi-byte code point and then pass them on as a single argument for further processing. The current `\FV@Break` (or other invoking macro) will have `\let \FV@Break@NextNext` to an appropriate macro that performs further processing. All code points are checked for validity here so as to raise errors as early as possible. Otherwise an invalid terminal byte sequence might gobble `\FV@EndBreak`, `\FV@Undefined`, or another delimiting macro, potentially making debugging much more difficult. It would be possible to use `\UTFviii@defined{<bytes>}` to trigger an error directly, but the current approach is to attempt to typeset invalid code points, which should trigger errors without relying on the details of the `utf8.def` implementation.

```

877 \def\FV@UTF@two@octets#1#2{%

```

```

878 \ifcsname u8:\detokenize{#1#2}\endcsname
879 \else
880   #1#2%
881 \fi
882 \FV@Break@NextNext{#1#2}}
883 \def\FV@UTF@three@octets#1#2#3{%
884 \ifcsname u8:\detokenize{#1#2#3}\endcsname
885 \else
886   #1#2#3%
887 \fi
888 \FV@Break@NextNext{#1#2#3}}
889 \def\FV@UTF@four@octets#1#2#3#4{%
890 \ifcsname u8:\detokenize{#1#2#3#4}\endcsname
891 \else
892   #1#2#3#4%
893 \fi
894 \FV@Break@NextNext{#1#2#3#4}}

```

`\FV@U8:<byte>` Define macros for each active byte. These are used for determining whether the current token is the first byte in a multi-byte sequence, and if so, invoking the necessary macro to capture the remaining bytes. The code is adapted from the beginning of `utf8.def`. Completely capitalized macro names are used to avoid having to worry about `\uppercase`.

```

895 \begingroup
896 \catcode'\~ =13
897 \catcode'\ " =12
898 \def\FV@UTFviii@loop{%
899 \uccode'\~\count@
900 \uppercase\expandafter{\FV@UTFviii@Tmp}%
901 \advance\count@\@ne
902 \ifnum\count@<\@tempcnta
903 \expandafter\FV@UTFviii@loop
904 \fi}

```

Setting up 2-byte UTF-8:

```

905 \count@"C2
906 \@tempcnta"E0
907 \def\FV@UTFviii@Tmp{\expandafter\gdef\csname FV@U8:\string~\endcsname{%
908 \FV@UTF@two@octets}}
909 \FV@UTFviii@loop

```

Setting up 3-byte UTF-8:

```

910 \count@"E0
911 \@tempcnta"F0
912 \def\FV@UTFviii@Tmp{\expandafter\gdef\csname FV@U8:\string~\endcsname{%
913 \FV@UTF@three@octets}}
914 \FV@UTFviii@loop

```

Setting up 4-byte UTF-8:

```

915 \count@"F0

```



```

916 \@tempcnta"F4
917 \def\FV@UTFviii@Tmp{\expandafter\gdef\csname FV@U8:\string~\endcsname{%
918   \FV@UTF@four@octets}}
919 \FV@UTFviii@loop
920 \endgroup

```

\FV@BreakBeforePrep@UTF We need UTF variants of the `breakbefore` and `breakafter` prep macros. These are only ever used with `inputenc` with UTF-8. There is no need for encoding checks here; checks are performed in `\FV@FormattingPrepHook` (checks are inserted into it after the non-UTF macro definitions).

```

921 \def\FV@BreakBeforePrep@UTF{%
922   \ifx\FV@BreakBefore\@empty\relax
923   \else
924     \gdef\FV@BreakBefore@Def{%
925     \begingroup
926     \def\FV@BreakBefore@Process##1{%
927       \ifcsname FV@U8:\detokenize{##1}\endcsname
928       \expandafter\let\expandafter\FV@Break@Next\csname FV@U8:\detokenize{##1}\endcsname
929       \let\FV@Break@NextNext\FV@BreakBefore@Process@ii
930       \else
931       \ifx##1\FV@Undefined
932       \let\FV@Break@Next\@gobble
933       \else
934       \let\FV@Break@Next\FV@BreakBefore@Process@i
935       \fi
936       \fi
937       \FV@Break@Next##1%
938     }%
939     \def\FV@BreakBefore@Process@i##1{%
940       \expandafter\FV@BreakBefore@Process@ii\expandafter{##1}}%
941     \def\FV@BreakBefore@Process@ii##1{%
942       \g@addto@macro\FV@BreakBefore@Def{%
943       \@namedef{FV@BreakBefore@Token\detokenize{##1}}{}}%
944       \FV@BreakBefore@Process
945     }%
946     \FV@EscChars
947     \expandafter\FV@BreakBefore@Process\FV@BreakBefore\FV@Undefined
948     \endgroup
949     \FV@BreakBefore@Def
950     \FV@BreakBeforePrep@PygmentsHook
951   \fi
952 }

```

\FV@BreakAfterPrep@UTF

```

953 \def\FV@BreakAfterPrep@UTF{%
954   \ifx\FV@BreakAfter\@empty\relax
955   \else
956     \gdef\FV@BreakAfter@Def{%
957     \begingroup

```

```

958 \def\FV@BreakAfter@Process##1{%
959 \ifcsname FV@U8:\detokenize{##1}\endcsname
960 \expandafter\let\expandafter\FV@Break@Next\csname FV@U8:\detokenize{##1}\endcsname
961 \let\FV@Break@NextNext\FV@BreakAfter@Process@ii
962 \else
963 \ifx##1\FV@Undefined
964 \let\FV@Break@Next\@gobble
965 \else
966 \let\FV@Break@Next\FV@BreakAfter@Process@i
967 \fi
968 \fi
969 \FV@Break@Next##1%
970 }%
971 \def\FV@BreakAfter@Process@i##1{%
972 \expandafter\FV@BreakAfter@Process@ii\expandafter{##1}}%
973 \def\FV@BreakAfter@Process@ii##1{%
974 \ifcsname FV@BreakBefore@Token\detokenize{##1}\endcsname
975 \ifthenelse{\boolean{FV@BreakBeforeGroup}}%
976 {\ifthenelse{\boolean{FV@BreakAfterGroup}}%
977 {}%
978 {\PackageError{fvextra}%
979 {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1}"}%
980 {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1}"}}}%
981 {\ifthenelse{\boolean{FV@BreakAfterGroup}}%
982 {\PackageError{fvextra}%
983 {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1}"}%
984 {Conflicting breakbeforegroup and breakaftergroup for "\detokenize{##1}"}}}%
985 {}%
986 \fi
987 \g@addto@macro\FV@BreakAfter@Def{%
988 \@namedef{FV@BreakAfter@Token\detokenize{##1}}{}}%
989 \FV@BreakAfter@Process
990 }%
991 \FV@EscChars
992 \expandafter\FV@BreakAfter@Process\FV@BreakAfter\FV@Undefined
993 \endgroup
994 \FV@BreakAfter@Def
995 \FV@BreakAfterPrep@PygmentsHook
996 \fi
997 }

```

\FV@Break@AnyToken@UTF Instead of just adding each token to \FV@TmpLine with a preceding break, also check for multi-byte code points and capture the remaining bytes when they are encountered.

```

998 \def\FV@Break@AnyToken@UTF#1{%
999 \ifcsname FV@U8:\detokenize{#1}\endcsname
1000 \expandafter\let\expandafter\FV@Break@Next\csname FV@U8:\detokenize{#1}\endcsname
1001 \let\FV@Break@NextNext\FV@Break@AnyToken@UTF@i
1002 \else
1003 \let\FV@Break@Next\FV@Break@AnyToken@UTF@i

```

```

1004 \fi
1005 \FV@Break@Next{#1}%
1006 }
1007 \def\FV@Break@AnyToken@UTF#1{%
1008 \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakAnywhereBreak#1}%
1009 \FV@Break@Scan}

```

\FV@Break@BeforeAfterToken@UTF Due to the way that the flow works, #1 will sometimes be a single byte and sometimes be a multi-byte UTF-8 code point. As a result, it is vital use use `\detokenize` in the UTF-8 leading byte checks; `\string` would only deal with the first byte. It is also important to keep track of the distinction between `\FV@Break@Next#1` and `\FV@Break@Next{#1}`. In some cases, a multi-byte sequence is being passed on as a single argument, so it must be enclosed in curly braces; in other cases, it is being re-inserted into the scanning stream and curly braces must be avoided lest they be interpreted as part of the original text.

```

1010 \def\FV@Break@BeforeAfterToken@UTF#1{%
1011 \ifcsname FV@U8:\detokenize{#1}\endcsname
1012 \expandafter\let\expandafter\FV@Break@Next\csname FV@U8:\detokenize{#1}\endcsname
1013 \let\FV@Break@NextNext\FV@Break@BeforeAfterToken@UTF@i
1014 \else
1015 \let\FV@Break@Next\FV@Break@BeforeAfterToken@UTF@i
1016 \fi
1017 \FV@Break@Next{#1}%
1018 }
1019 \def\FV@Break@BeforeAfterToken@UTF@i#1{%
1020 \ifcsname FV@BreakBeforeToken\detokenize{#1}\endcsname
1021 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@UTF
1022 \else
1023 \ifcsname FV@BreakAfterToken\detokenize{#1}\endcsname
1024 \let\FV@Break@Next\FV@Break@AfterTokenBreak@UTF
1025 \else
1026 \let\FV@Break@Next\FV@Break@BeforeAfterTokenNoBreak@UTF
1027 \fi
1028 \fi
1029 \FV@Break@Next{#1}%
1030 }
1031 \def\FV@Break@BeforeAfterTokenNoBreak@UTF#1{%
1032 \g@addto@macro{\FV@TmpLine}{#1}%
1033 \def\FV@LastToken{#1}%
1034 \FV@Break@Scan}
1035 \def\FV@Break@BeforeTokenBreak@UTF#1{%
1036 \def\FV@CurrentToken{#1}%
1037 \ifthenelse{\boolean{FV@BreakBeforeGroup}}{
1038 {\ifx\FV@CurrentToken\FV@LastToken\relax
1039 \ifcsname FV@BreakAfterToken\detokenize{#1}\endcsname
1040 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan@UTF
1041 \def\FV@RescanToken{#1}%
1042 \else
1043 \g@addto@macro{\FV@TmpLine}{#1}%

```

```

1044     \let\FV@Break@Next\FV@Break@Scan
1045     \def\FV@LastToken{#1}%
1046     \fi
1047   \else
1048     \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
1049       \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakBeforeBreak}%
1050       \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan@UTF
1051       \def\FV@RescanToken{#1}%
1052     \else
1053       \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakBeforeBreak#1}%
1054       \let\FV@Break@Next\FV@Break@Scan
1055       \def\FV@LastToken{#1}%
1056     \fi
1057   \fi}%
1058   {\ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
1059     \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakBeforeBreak}%
1060     \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan@UTF
1061     \def\FV@RescanToken{#1}%
1062   \else
1063     \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakBeforeBreak#1}%
1064     \let\FV@Break@Next\FV@Break@Scan
1065     \def\FV@LastToken{#1}%
1066   \fi}%
1067   \FV@Break@Next}
1068   \def\FV@Break@BeforeTokenBreak@AfterRescan@UTF{%
1069     \expandafter\FV@Break@AfterTokenBreak@UTF\expandafter{\FV@RescanToken}}
1070   \def\FV@Break@AfterTokenBreak@UTF#1{%
1071     \def\FV@LastToken{#1}%
1072     \@ifnextchar\FV@Space@ifx%
1073     {\g@addto@macro{\FV@TmpLine}{#1}\FV@Break@Scan}%
1074     {\ifthenelse{\boolean{FV@BreakAfterGroup}}{%
1075       {\g@addto@macro{\FV@TmpLine}{#1}%
1076         \ifx\@let@token\bgroup\relax
1077           \let\FV@Break@Next\FV@Break@AfterTokenBreak@Group@UTF
1078         \else
1079           \let\FV@Break@Next\FV@Break@AfterTokenBreak@UTF@i
1080         \fi}%
1081       {\g@addto@macro{\FV@TmpLine}{#1\FancyVerbBreakAfterBreak}%
1082         \let\FV@Break@Next\FV@Break@Scan}%
1083       \FV@Break@Next}%
1084   }
1085   \def\FV@Break@AfterTokenBreak@UTF@i#1{%
1086     \ifcsname FV@U8:\detokenize{#1}\endcsname
1087       \expandafter\let\expandafter\FV@Break@Next\csname FV@U8:\detokenize{#1}\endcsname
1088       \let\FV@Break@NextNext\FV@Break@AfterTokenBreak@UTF@i
1089     \else
1090       \def\FV@NextToken{#1}%
1091       \ifx\FV@LastToken\FV@NextToken
1092       \else
1093         \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakAfterBreak}%

```

```

1094     \fi
1095     \let\FV@Break@Next\FV@Break@Scan
1096 \fi
1097 \FV@Break@Next#1}
1098 \def\FV@Break@AfterTokenBreak@Group@UTF#1{%
1099   \g@addto@macro{\FV@TmpLine}{\FV@Break@AfterTokenBreak@Group@UTF#1}%
1100   \ifstrempy{#1}%
1101     {\let\FV@Break@Next\FV@Break@AfterTokenBreak@Group@UTF#1}%
1102     {\let\FV@Break@Next\FV@Break@Scan\let\FV@LastToken=\FV@Undefined}%
1103   \FV@Break@Next}
1104 \def\FV@Break@AfterTokenBreak@Group@UTF#i{%
1105   \ifnextchar\bgroup%
1106     {\FV@Break@Scan}%
1107     {\FV@Break@AfterTokenBreak@Group@UTF#ii}}
1108 \def\FV@Break@AfterTokenBreak@Group@UTF#ii#1{%
1109   \ifcsname FV@U8:\detokenize{#1}\endcsname
1110     \expandafter\let\expandafter\FV@Break@Next\csname FV@U8:\detokenize{#1}\endcsname
1111     \let\FV@Break@NextNext\FV@Break@AfterTokenBreak@Group@UTF#ii
1112   \else
1113     \def\FV@NextToken{#1}%
1114     \ifx\FV@LastToken\FV@NextToken
1115     \else
1116       \g@addto@macro{\FV@TmpLine}{\FancyVerbBreakAfterBreak}%
1117     \fi
1118     \let\FV@Break@Next\FV@Break@Scan
1119 \fi
1120 \FV@Break@Next#1}

```

End the conditional creation of the pdfTeX UTF macros:

```
1121 \fi\fi
```

Line processing before scanning

\FV@makeLineNumber The `lineno` package is used for formatting wrapped lines and inserting break symbols. We need a version of `lineno`'s `\makeLineNumber` that is adapted for our purposes. This is adapted directly from the example `\makeLineNumber` that is given in the `lineno` documentation under the discussion of internal line numbers. The `\FV@SetLineBreakLast` is needed to determine the internal line number of the last segment of the broken line, so that we can disable the right-hand break symbol on this segment. When a right-hand break symbol is in use, a line of code will be processed twice: once to determine the last internal line number, and once to use this information only to insert right-hand break symbols on the appropriate lines. During the second run, `\FV@SetLineBreakLast` is disabled by `\letting` it to `\relax`.

```

1122 \def\FV@makeLineNumber{%
1123   \hss
1124   \FancyVerbBreakSymbolLeftLogic{\FancyVerbBreakSymbolLeft}%
1125   \hbox to \FV@BreakSymbolSepLeft{\hfill}%

```

```

1126 \rlap{\hskip\linewidth
1127 \hbox to \FV@BreakSymbolSepRight{\hfill}}%
1128 \FancyVerbBreakSymbolRightLogic{\FancyVerbBreakSymbolRight}}%
1129 \FV@SetLineBreakLast
1130 }%
1131 }

```

\FV@RaggedRight We need a copy of the default `\raggedright` to ensure that everything works with classes or packages that use a special definition.

```

1132 \def\FV@RaggedRight{%
1133 \let\\@centercr
1134 \@rightskip\@flushglue\rightskip\@rightskip\leftskip\z@skip\parindent\z@}

```

\FV@SaveLineBox This is the macro that does most of the work. It was inspired by Marco Daniel's code at <http://tex.stackexchange.com/a/112573/10742>.

This macro is invoked when a line is too long. We modify the `\linewidth` to take into account `breakindent` and `breakautoindent`, and insert `\hboxes` to fill the empty space. We also account for `breaksymbolindentleft` and `breaksymbolindentright`, but *only* when there are actually break symbols. The code is placed in a `\parbox`. Break symbols are inserted via `lineno's internallinenumbers*`, which does internal line numbers without continuity between environments (the `linenumber` counter is automatically reset). The beginning of the line has negative `\hspace` inserted to pull it out to the correct starting position. `\struts` are used to maintain correct line heights. The `\parbox` is followed by an empty `\hbox` that takes up the space needed for a right-hand break symbol (if any). `\FV@BreakByTokenAnywhereHook` is a hook for using `breakbytokenanywhere` when working with Pygments. Since it is within `internallinenumbers*`, its effects do not escape.

```

1135 \def\FV@SaveLineBox#1{%
1136 \savebox{\FV@LineBox}{%
1137 \advance\linewidth by -\FV@BreakIndent
1138 \hbox to \FV@BreakIndent{\hfill}}%
1139 \ifthenelse{\boolean{FV@BreakAutoIndent}}{%
1140 {\let\FV@LineIndentChars\empty
1141 \FV@GetLineIndent#1\FV@Undefined
1142 \savebox{\FV@LineIndentBox}{\FV@LineIndentChars}}%
1143 \hbox to \wd\FV@LineIndentBox{\hfill}}%
1144 \advance\linewidth by -\wd\FV@LineIndentBox
1145 \setcounter{FV@TrueTabCounter}{0}}%
1146 {}%
1147 \ifdefempty{\FancyVerbBreakSymbolLeft}{}%
1148 {\hbox to \FV@BreakSymbolIndentLeft{\hfill}}%
1149 \advance\linewidth by -\FV@BreakSymbolIndentLeft}%
1150 \ifdefempty{\FancyVerbBreakSymbolRight}{}%
1151 {\advance\linewidth by -\FV@BreakSymbolIndentRight}}%
1152 \parbox[t]{\linewidth}{%
1153 \FV@RaggedRight
1154 \leftlinenumbers*

```

```

1155 \begin{internallinenumbers*}%
1156 \let\makeLineNumber\FV@makeLineNumber
1157 \noindent\hspace*{-\FV@BreakIndent}%
1158 \ifdefempty{\FancyVerbBreakSymbolLeft}{}%
1159 \hspace*{-\FV@BreakSymbolIndentLeft}}%
1160 \ifthenelse{\boolean{FV@BreakAutoIndent}}%
1161 {\hspace*{-\wd\FV@LineIndentBox}}%
1162 {}%
1163 \FV@BreakByTokenAnywhereHook
1164 \strut\FancyVerbFormatText{%
1165 \FancyVerbBreakStart #1\FancyVerbBreakStop}\nobreak\strut
1166 \end{internallinenumbers*}
1167 }%
1168 \ifdefempty{\FancyVerbBreakSymbolRight}{}%
1169 {\hbox to \FV@BreakSymbolIndentRight{\hfill}}%
1170 }%
1171 }
1172 \let\FV@BreakByTokenAnywhereHook\relax

```

\FV@ListProcessLine@Break This macro is based on the original `\FV@ListProcessLine` and follows it as closely as possible. The `\linewidth` is reduced by `\FV@FrameSep` and `\FV@FrameRule` so that text will not overrun frames. This is done conditionally based on which frames are in use. We save the current line in a box, and only do special things if the box is too wide. For uniformity, all text is placed in a `\parbox`, even if it doesn't need to be wrapped.

If a line is too wide, then it is passed to `\FV@SaveLineBox`. If there is no right-hand break symbol, then the saved result in `\FV@LineBox` may be used immediately. If there is a right-hand break symbol, then the line must be processed a second time, so that the right-hand break symbol may be removed from the final segment of the broken line (since it does not continue). During the first use of `\FV@SaveLineBox`, the counter `FancyVerbLineBreakLast` is set to the internal line number of the last segment of the broken line. During the second use of `\FV@SaveLineBox`, we disable this (`\let\FV@SetLineBreakLast\relax`) so that the value of `FancyVerbLineBreakLast` remains fixed and thus may be used to determine when a right-hand break symbol should be inserted.

```

1173 \def\FV@ListProcessLine@Break#1{%
1174 \hbox to \hsize{%
1175 \kern\leftmargin
1176 \hbox to \linewidth{%
1177 \ifx\FV@RightListFrame\relax\else
1178 \advance\linewidth by -\FV@FrameSep
1179 \advance\linewidth by -\FV@FrameRule
1180 \fi
1181 \ifx\FV@LeftListFrame\relax\else
1182 \advance\linewidth by -\FV@FrameSep
1183 \advance\linewidth by -\FV@FrameRule
1184 \fi
1185 \ifx\FV@Tab\FV@TrueTab

```

```

1186 \let\FV@TrueTabSaveWidth\FV@TrueTabSaveWidth@Save
1187 \setcounter{FV@TrueTabCounter}{0}%
1188 \fi
1189 \sbox{\FV@LineBox}{%
1190 \FancyVerbFormatLine{%
1191 \FancyVerbHighlightLine %<-- Default definition using \rlap breaks breaking
1192 {\FV@ObeyTabs{\FancyVerbFormatText{#1}}}}}%
1193 \ifx\FV@Tab\FV@TrueTab
1194 \let\FV@TrueTabSaveWidth\relax
1195 \fi
1196 \ifdim\wd\FV@LineBox>\linewidth
1197 \setcounter{FancyVerbLineBreakLast}{0}%
1198 \ifx\FV@Tab\FV@TrueTab
1199 \let\FV@Tab\FV@TrueTab@UseWidth
1200 \setcounter{FV@TrueTabCounter}{0}%
1201 \fi
1202 \FV@SaveLineBox{#1}%
1203 \ifdefempty{\FancyVerbBreakSymbolRight}{\FancyVerbBreakSymbolRight}{%
1204 \let\FV@SetLineBreakLast\relax
1205 \setcounter{FV@TrueTabCounter}{0}%
1206 \FV@SaveLineBox{#1}}}%
1207 \FV@LeftListNumber
1208 \FV@LeftListFrame
1209 \FancyVerbFormatLine{%
1210 \FancyVerbHighlightLine{\usebox{\FV@LineBox}}}%
1211 \FV@RightListFrame
1212 \FV@RightListNumber
1213 \ifx\FV@Tab\FV@TrueTab@UseWidth
1214 \let\FV@Tab\FV@TrueTab
1215 \fi
1216 \else
1217 \FV@LeftListNumber
1218 \FV@LeftListFrame
1219 \FancyVerbFormatLine{%
1220 \FancyVerbHighlightLine{%
1221 \parbox[t]{\linewidth}{%
1222 \noindent\strut\FV@ObeyTabs{\FancyVerbFormatText{#1}}\strut}}}%
1223 \FV@RightListFrame
1224 \FV@RightListNumber
1225 \fi}%
1226 \hss}\baselineskip\z@\lineskip\z@}

```

10.8 Pygments compatibility

This section makes line breaking compatible with [Pygments](#), which is used by several packages including `minted` and `pythontex` for syntax highlighting. A few additional line breaking options are also defined for working with Pygments.

`\FV@BreakBeforePrep@Pygments` Pygments converts some characters into macros to ensure that they appear literally. As a result, `breakbefore` and `breakafter` would fail for these characters. This

macro checks for the existence of breaking macros for these characters, and creates breaking macros for the corresponding Pygments character macros as necessary.

The argument that the macro receives is the detokenized name of the main Pygments macro, with the trailing space that detokenization produces stripped. All macro names must end with a space, because the breaking algorithm uses detokenization on each token when checking for breaking macros, and this will produce a trailing space.

```

1227 \def\FV@BreakBeforePrep@Pygments#1{%
1228   \ifcsname FV@BreakBefore@Token\@backslashchar\endcsname
1229     \@namedef{FV@BreakBefore@Token#1Zbs }{}%
1230   \fi
1231   \ifcsname FV@BreakBefore@Token\FV@underscorechar\endcsname
1232     \@namedef{FV@BreakBefore@Token#1Zus }{}%
1233   \fi
1234   \ifcsname FV@BreakBefore@Token\@charlb\endcsname
1235     \@namedef{FV@BreakBefore@Token#1Zob }{}%
1236   \fi
1237   \ifcsname FV@BreakBefore@Token\@charrb\endcsname
1238     \@namedef{FV@BreakBefore@Token#1Zcb }{}%
1239   \fi
1240   \ifcsname FV@BreakBefore@Token\detokenize{~}\endcsname
1241     \@namedef{FV@BreakBefore@Token#1Zca }{}%
1242   \fi
1243   \ifcsname FV@BreakBefore@Token\FV@ampchar\endcsname
1244     \@namedef{FV@BreakBefore@Token#1Zam }{}%
1245   \fi
1246   \ifcsname FV@BreakBefore@Token\detokenize{<}\endcsname
1247     \@namedef{FV@BreakBefore@Token#1Zlt }{}%
1248   \fi
1249   \ifcsname FV@BreakBefore@Token\detokenize{>}\endcsname
1250     \@namedef{FV@BreakBefore@Token#1Zgt }{}%
1251   \fi
1252   \ifcsname FV@BreakBefore@Token\FV@hashchar\endcsname
1253     \@namedef{FV@BreakBefore@Token#1Zsh }{}%
1254   \fi
1255   \ifcsname FV@BreakBefore@Token\@percentchar\endcsname
1256     \@namedef{FV@BreakBefore@Token#1Zpc }{}%
1257   \fi
1258   \ifcsname FV@BreakBefore@Token\FV@dollarchar\endcsname
1259     \@namedef{FV@BreakBefore@Token#1Zdl }{}%
1260   \fi
1261   \ifcsname FV@BreakBefore@Token\detokenize{-}\endcsname
1262     \@namedef{FV@BreakBefore@Token#1Zhy }{}%
1263   \fi
1264   \ifcsname FV@BreakBefore@Token\detokenize{'}\endcsname
1265     \@namedef{FV@BreakBefore@Token#1Zsq }{}%
1266   \fi
1267   \ifcsname FV@BreakBefore@Token\detokenize{"}\endcsname
1268     \@namedef{FV@BreakBefore@Token#1Zdq }{}%

```

```

1269 \fi
1270 \ifcsname FV@BreakBefore@Token\FV@tildechar\endcsname
1271 \namedef{FV@BreakBefore@Token#1Zti }{}%
1272 \fi
1273 \ifcsname FV@BreakBefore@Token\detokenize{@}\endcsname
1274 \namedef{FV@BreakBefore@Token#1Zat }{}%
1275 \fi
1276 \ifcsname FV@BreakBefore@Token\detokenize{[]\endcsname
1277 \namedef{FV@BreakBefore@Token#1Zlb }{}%
1278 \fi
1279 \ifcsname FV@BreakBefore@Token\detokenize{[]}\endcsname
1280 \namedef{FV@BreakBefore@Token#1Zrb }{}%
1281 \fi
1282 }

\FV@BreakAfterPrep@Pygments

1283 \def\FV@BreakAfterPrep@Pygments#1{%
1284 \ifcsname FV@BreakAfter@Token\@backslashchar\endcsname
1285 \namedef{FV@BreakAfter@Token#1Zbs }{}%
1286 \fi
1287 \ifcsname FV@BreakAfter@Token\FV@underscorechar\endcsname
1288 \namedef{FV@BreakAfter@Token#1Zus }{}%
1289 \fi
1290 \ifcsname FV@BreakAfter@Token\@charlb\endcsname
1291 \namedef{FV@BreakAfter@Token#1Zob }{}%
1292 \fi
1293 \ifcsname FV@BreakAfter@Token\@charrb\endcsname
1294 \namedef{FV@BreakAfter@Token#1Zcb }{}%
1295 \fi
1296 \ifcsname FV@BreakAfter@Token\detokenize{^}\endcsname
1297 \namedef{FV@BreakAfter@Token#1Zca }{}%
1298 \fi
1299 \ifcsname FV@BreakAfter@Token\FV@ampchar\endcsname
1300 \namedef{FV@BreakAfter@Token#1Zam }{}%
1301 \fi
1302 \ifcsname FV@BreakAfter@Token\detokenize{<}\endcsname
1303 \namedef{FV@BreakAfter@Token#1Zlt }{}%
1304 \fi
1305 \ifcsname FV@BreakAfter@Token\detokenize{>}\endcsname
1306 \namedef{FV@BreakAfter@Token#1Zgt }{}%
1307 \fi
1308 \ifcsname FV@BreakAfter@Token\FV@hashchar\endcsname
1309 \namedef{FV@BreakAfter@Token#1Zsh }{}%
1310 \fi
1311 \ifcsname FV@BreakAfter@Token\@percentchar\endcsname
1312 \namedef{FV@BreakAfter@Token#1Zpc }{}%
1313 \fi
1314 \ifcsname FV@BreakAfter@Token\FV@dollarchar\endcsname
1315 \namedef{FV@BreakAfter@Token#1Zdl }{}%
1316 \fi

```

```

1317 \ifcsname FV@BreakAfter@Token\detokenize{-}\endcsname
1318 \namedef{FV@BreakAfter@Token#1Zhy }{ }%
1319 \fi
1320 \ifcsname FV@BreakAfter@Token\detokenize{'}\endcsname
1321 \namedef{FV@BreakAfter@Token#1Zsq }{ }%
1322 \fi
1323 \ifcsname FV@BreakAfter@Token\detokenize{"}\endcsname
1324 \namedef{FV@BreakAfter@Token#1Zdq }{ }%
1325 \fi
1326 \ifcsname FV@BreakAfter@Token\FV@tildechar\endcsname
1327 \namedef{FV@BreakAfter@Token#1Zti }{ }%
1328 \fi
1329 \ifcsname FV@BreakAfter@Token\detokenize{@}\endcsname
1330 \namedef{FV@BreakAfter@Token#1Zat }{ }%
1331 \fi
1332 \ifcsname FV@BreakAfter@Token\detokenize{[]\endcsname
1333 \namedef{FV@BreakAfter@Token#1Zlb }{ }%
1334 \fi
1335 \ifcsname FV@BreakAfter@Token\detokenize{[]}\endcsname
1336 \namedef{FV@BreakAfter@Token#1Zrb }{ }%
1337 \fi
1338 }

```

breakbytoken When Pygments is used, do not allow breaks within **Pygments tokens**. So, for example, breaks would not be allowed within a string, but could occur before or after it. This has no affect when Pygments is not in use, and is only intended for minted, pythontex, and similar packages.

```

1339 \newbool{FV@breakbytoken}
1340 \define@booleankey{FV}{breakbytoken}%
1341 {\booltrue{FV@breakbytoken}}%
1342 {\boolfalse{FV@breakbytoken}\boolfalse{FV@breakbytokenanywhere}}

```

breakbytokenanywhere **breakbytoken** prevents breaks *within* tokens. Breaks outside of tokens may still occur at spaces. This option also enables breaks between immediately adjacent tokens that are not separated by spaces. Its definition is tied in with **breakbytoken** so that **breakbytoken** may be used as a check for whether either option is in use; essentially, **breakbytokenanywhere** is treated as a special case of **breakbytoken**.

```

1343 \newbool{FV@breakbytokenanywhere}
1344 \define@booleankey{FV}{breakbytokenanywhere}%
1345 {\booltrue{FV@breakbytokenanywhere}\booltrue{FV@breakbytoken}}%
1346 {\boolfalse{FV@breakbytokenanywhere}\boolfalse{FV@breakbytoken}}

```

FancyVerbBreakByTokenAnywhereBreak This is the break introduced when **breakbytokenanywhere=true**. Alternatives would be `\discretionary{}{}{} or \linebreak[0]`.

```

1347 \def\FancyVerbBreakByTokenAnywhereBreak{\allowbreak{}}

```

\VerbatimPygments This is the command that activates Pygments features. It must be invoked before `\begin{Verbatim}`, etc., but inside a `\begingroup... \endgroup` so that its effects do not escape into the rest of the document (for example, within the beginning of an

environment. It takes two arguments: The Pygments macro that literally appears (`\PYG` for `minted` and `pythontex`), and the Pygments macro that should actually be used (`\PYG<style_name>` for `minted` and `pythontex`). The two are distinguished because it can be convenient to highlight everything using the same literal macro name, and then `\let` it to appropriate values to change styles, rather than redoing all highlighting to change styles. It modifies `\FV@PygmentsHook`, which is at the beginning of `\FV@FormattingPrepHook`, to make the actual changes at the appropriate time.

```
1348 \def\VerbatimPygments#1#2{%
1349   \def\FV@PygmentsHook{\FV@VerbatimPygments{#1}{#2}}}
```

`\FV@VerbatimPygments` This does all the actual work. Again, `#1` is the Pygments macro that literally appears, and `#2` is the macro that is actually to be used.

The `breakbefore` and `breakafter` hooks are redefined. This requires some trickery to get the detokenized name of the main Pygments macro without the trailing space that detokenization of a macro name produces.

In the non-`breakbytoken` case, `#1` is redefined to use `#2` internally, bringing in `\FancyVerbBreakStart` and `\FancyVerbBreakStop` to allow line breaks.

In the `breakbytoken` cases, an `\hbox` is used to prevent breaks within the macro (breaks could occur at spaces even without `\FancyVerbBreakStart`). The `breakbytokenanywhere` case is similar but a little tricky. `\FV@BreakByTokenAnywhereHook`, which is inside `\FV@SaveLineBox` where line breaking occurs, is used to define `\FV@BreakByTokenAnywhereBreak` so that it will “do nothing” the first time it is used and on subsequent invocations become `\FancyVerbBreakByTokenAnywhereBreak`. Because the hook is within the `internallinenumbers*` environment, the redefinition doesn’t escape, and the default global definition of `\FV@BreakByTokenAnywhereBreak` as `\relax` is not affected. We don’t want the actual break to appear before the first Pygments macro in case it might cause a spurious break after leading whitespace. But we must have breaks *before* Pygments macros because otherwise lookahead would be necessary.

An intermediate variable `\FV@PYG` is defined to avoid problems in case `#1=#2`. There is also a check for a non-existent `#2` (`\PYG<style_name>` may not be created until a later compile in the `pythontex` case); if `#2` does not exist, fall back to `#1`. For the existence check, `\ifx... \relax` must be used instead of `\ifcurname`, because `#2` will be a macro, and will typically be created with `\curname... \endcurname` which will `\let` the macro to `\relax` if it doesn’t already exist.

```
1350 \def\FV@VerbatimPygments#1#2{%
1351   \edef\FV@PYG@Literal{\expandafter\FV@DetokMacro@StripSpace\detokenize{#1}}%
1352   \def\FV@BreakBeforePrep@PygmentsHook{%
1353     \expandafter\FV@BreakBeforePrep@Pygments\expandafter{\FV@PYG@Literal}}%
1354   \def\FV@BreakAfterPrep@PygmentsHook{%
1355     \expandafter\FV@BreakAfterPrep@Pygments\expandafter{\FV@PYG@Literal}}%
1356   \ifx#2\relax
1357     \let\FV@PYG#1%
1358   \else
1359     \let\FV@PYG#2%
1360   \fi
```

```

1361 \ifbool{FV@breakbytoken}%
1362   {\ifbool{FV@breakbytokenanywhere}%
1363     {\def\FV@BreakByTokenAnywhereHook{%
1364       \def\FV@BreakByTokenAnywhereBreak{%
1365         \let\FV@BreakByTokenAnywhereBreak\FancyVerbBreakByTokenAnywhereBreak}}%
1366       \def#1##1##2{%
1367         \FV@BreakByTokenAnywhereBreak
1368         \leavevmode\hbox{\FV@PYG{##1}{##2}}}%
1369       {\def#1##1##2{%
1370         \leavevmode\hbox{\FV@PYG{##1}{##2}}}%
1371       {\def#1##1##2{%
1372         \FV@PYG{##1}{\FancyVerbBreakStart##2\FancyVerbBreakStop}}}%
1373   }
1374 \let\FV@BreakByTokenAnywhereBreak\relax
1375 \def\FV@DetokMacro@StripSpace#1 {#1}

```