

# The `fvextra` package

Geoffrey M. Poore  
[gpoore@gmail.com](mailto:gpoore@gmail.com)  
[github.com/gpoore/fvextra](https://github.com/gpoore/fvextra)

v1.13.0 from 2025/05/14

## **Abstract**

`fvextra` provides several extensions to `fancyvrb`, including automatic line breaking and improved math mode. `\Verb` is reimplemented so that it works (with a few limitations) inside other commands, even in movable arguments and PDF bookmarks. The new command `\EscVerb` is similar to `\Verb` except that it works everywhere without limitations by allowing the backslash to serve as an escape character. `fvextra` also patches some `fancyvrb` internals.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Usage</b>	<b>5</b>
2.1	Pandoc compatibility . . . . .	5
<b>3</b>	<b>General options</b>	<b>6</b>
<b>4</b>	<b>General commands</b>	<b>13</b>
4.1	Inline-only settings with <code>\fvinline</code> . . . . .	13
4.2	Custom formatting for inline commands like <code>\Verb</code> with <code>\FancyVerbFormatInline</code> . . . . .	13
4.3	Custom formatting for environments like <code>Verbatim</code> with <code>\FancyVerbFormatLine</code> and <code>\FancyVerbFormatText</code> . . . . .	13
<b>5</b>	<b>Reimplemented commands</b>	<b>14</b>
5.1	<code>\Verb</code> . . . . .	14
5.2	<code>\SaveVerb</code> . . . . .	15
5.3	<code>\UseVerb</code> . . . . .	16
<b>6</b>	<b>New commands and environments</b>	<b>16</b>
6.1	<code>\EscVerb</code> . . . . .	16
6.2	<code>VerbEnv</code> . . . . .	17
6.3	<code>VerbatimWrite</code> . . . . .	17
6.4	Buffers . . . . .	18
6.4.1	<code>VerbatimBuffer</code> . . . . .	18
6.4.2	<code>\VerbatimInsertBuffer</code> . . . . .	21
6.4.3	<code>\VerbatimClearBuffer</code> . . . . .	22
6.4.4	<code>\InsertBuffer</code> . . . . .	22
6.4.5	<code>\ClearBuffer</code> . . . . .	23
6.4.6	<code>\BufferMdfivesum</code> . . . . .	23
6.4.7	<code>\IterateBuffer</code> . . . . .	23
6.4.8	<code>\WriteBuffer</code> . . . . .	23
<b>7</b>	<b>Line breaking</b>	<b>23</b>
7.1	Line breaking options . . . . .	24
7.2	Line breaking and tab expansion . . . . .	30
7.3	Advanced line breaking . . . . .	31
7.3.1	A few notes on algorithms . . . . .	31
7.3.2	Breaks within macro arguments . . . . .	31
7.3.3	Customizing break behavior . . . . .	33
<b>8</b>	<b>Pygments support</b>	<b>33</b>
8.1	Options for users . . . . .	33
8.2	For package authors . . . . .	34

<b>9 Patches</b>	<b>34</b>
9.1 Visible spaces . . . . .	34
9.2 <code>obeytabs</code> with visible tabs and with tabs inside macro arguments . . . . .	34
9.3 Math mode . . . . .	35
9.3.1 Spaces . . . . .	35
9.3.2 Symbols and fonts . . . . .	36
9.4 Orphaned labels . . . . .	36
9.5 <code>rulecolor</code> and <code>fillcolor</code> . . . . .	36
9.6 Command lookahead tokenization . . . . .	36
<b>10 Additional modifications to <code>fancyvrb</code></b>	<b>37</b>
10.1 Backtick and single quotation mark . . . . .	37
10.2 Line numbering . . . . .	37
<b>11 Undocumented features of <code>fancyvrb</code></b>	<b>37</b>
11.1 Undocumented option . . . . .	37
11.2 Undocumented macros . . . . .	37
<b>12 Implementation</b>	<b>38</b>
12.1 Required packages . . . . .	38
12.2 Utility macros . . . . .	38
12.2.1 <code>fancyvrb</code> space and tab tokens . . . . .	38
12.2.2 ASCII processing . . . . .	39
12.2.3 Sentinels . . . . .	46
12.2.4 Active character definitions . . . . .	46
12.3 pdfTeX with <code>inputenc</code> using UTF-8 . . . . .	47
12.4 Reading and processing command arguments . . . . .	50
12.4.1 Tokenization and lookahead . . . . .	50
12.4.2 Reading arguments . . . . .	51
12.4.3 Reading and protecting arguments in expansion-only contexts . . . . .	55
12.4.4 Converting detokenized tokens into PDF strings . . . . .	59
12.4.5 Detokenizing verbatim arguments . . . . .	60
12.4.6 Retokenizing detokenized arguments . . . . .	77
12.5 Hooks . . . . .	78
12.6 Escaped characters . . . . .	79
12.7 Inline-only options . . . . .	79
12.8 Reimplementations . . . . .	80
12.8.1 <code>extra</code> option . . . . .	80
12.8.2 <code>\FancyVerbFormatInline</code> . . . . .	80
12.8.3 <code>\Verb</code> . . . . .	80
12.8.4 <code>\SaveVerb</code> . . . . .	82
12.8.5 <code>\UseVerb</code> . . . . .	83
12.9 New commands and environments . . . . .	84
12.9.1 <code>\EscVerb</code> . . . . .	84
12.9.2 <code>VerbEnv</code> . . . . .	85
12.9.3 <code>VerbatimWrite</code> . . . . .	87
12.9.4 <code>VerbatimBuffer</code> . . . . .	88
12.9.5 <code>\VerbatimInsertBuffer</code> . . . . .	91
12.9.6 <code>\VerbatimClearBuffer</code> . . . . .	93
12.9.7 <code>\InsertBuffer</code> . . . . .	93

12.9.8	<code>\ClearBuffer</code>	94
12.9.9	<code>\BufferMdfivesum</code>	94
12.9.10	<code>\IterateBuffer</code> , <code>\IterateBufferBreak</code>	95
12.10	Patches	96
12.10.1	Delimiting characters for verbatim commands	96
12.10.2	<code>\CustomVerbatimCommand</code> compatibility with <code>\FVExtraRobustCommand</code>	97
12.10.3	Visible spaces	98
12.10.4	<code>obeytabs</code> with visible tabs and with tabs inside macro arguments	98
12.10.5	Fonts and symbols in math mode	102
12.10.6	Ophaned label	104
12.10.7	<code>rulecolor</code> and <code>fillcolor</code>	104
12.11	Extensions	105
12.11.1	New options requiring minimal implementation	105
12.11.2	Formatting with <code>\FancyVerbFormatLine</code> , <code>\FancyVerbFormatText</code> , and <code>\FancyVerbHighlightLine</code>	108
12.11.3	Line numbering	109
12.11.4	Background color	112
12.11.5	Line highlighting or emphasis	117
12.12	Line breaking	120
12.12.1	Options and associated macros	120
12.12.2	Line breaking implementation	130
12.13	Pygments compatibility	147

# 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  $\text{\LaTeX}$  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 reimplementations (section 5), patches (section 9), and modifications to `fancyvrb` defaults (section 10) are documented. In most cases, there are options to switch back to original implementations or original default behavior.

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`.

## 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 9.3). `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 `etoolbox` package is 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`.

### 2.1 Pandoc compatibility

`fvextra` supports line breaking in `Pandoc`  $\text{\LaTeX}$  output that includes highlighted source code. Enabling basic line breaking at spaces is as simple as adding

`\usepackage{fvextra}` and `\fvset{breaklines}` to the Pandoc Markdown `header-includes`.

By default, more advanced line breaking features such as `breakanywhere`, `breakbefore`, and `breakafter` will not work with Pandoc highlighted output, due to the presence of the syntax highlighting macros. This can be fixed by using `breaknonspaceingroup`, which enables all line breaking features within macros. For example, the following YAML metadata in a Markdown document would redefine the Pandoc `Highlighting` environment to enable line breaking anywhere.

---

```
---
header-includes:
- |
  ```{=latex}
  \usepackage{fvextra}
  \DefineVerbatimEnvironment{Highlighting}{Verbatim}{
    commandchars=\\\{\},
    breaklines, breaknonspaceingroup, breakanywhere}
  ```
---
```

---

### 3 General options

`fvextra` adds several general options to `fancyvrb`. All options related to automatic line breaking are described separately in section 7. All options related to syntax highlighting using Pygments are described in section 8.

`backgroundcolor` (string) (default: none)  
Background color behind commands and environments. This is only a basic, lightweight implementation of background colors using `\colorbox`. For more control of background colors, consider `tcolorbox` or a similar package, or a custom background color implementation.

Background colors are implemented with `\colorbox`, which prevents line breaking for `\Verb`, `\UseVerb`, and `\EscVerb`. Background colors are only compatible with `fvextra`'s reimplemented `\Verb` and `\UseVerb` (`extra=true`).

When there is a background color in environments such as `Verbatim` that typeset blocks of text, some PDF readers and browsers can introduce rendering artifacts. These appear in between lines of text as extremely thin horizontal rules that are the color of the page behind the background. They tend to be more noticeable for dark background colors on a light page. `fvextra` attempts to prevent rendering artifacts by slightly oversizing the `\colorbox` behind each line and then slightly overlapping these boxes. This overlap can be fine-tuned if necessary with `backgroundcolorboxoverlap`. If rendering artifacts are an issue with your software in spite of a small overlap, consider `tcolorbox` or a similar package, or a custom background color implementation.

See `backgroundcolorvphantom` to customize the height of the background for each line of text, especially for `\Verb`, `\EscVerb`, and `BVerbatim`.

When `backgroundcolor` is used with `Verbatim` with a `frame`, it may be necessary to adjust `framesep` and `fillcolor` depending on the desired result. `backgroundcolorpadding` provides a shortcut for this.

`backgroundcolorboxoverlap` (length) (default: 0.25pt)

The amount by which `\colorboxes` that are used to provide `backgroundcolor` are oversized and then overlapped, to prevent rendering artifacts.

When there is a background color in environments such as `Verbatim` that typeset blocks of text, some PDF readers and browsers can introduce rendering artifacts. These appear in between lines of text as extremely thin horizontal rules that are the color of the page behind the background. They tend to be more noticeable for dark background colors on a light page. `fvextra` attempts to prevent rendering artifacts by slightly oversizing the `\colorbox` behind each line by `backgroundcolorboxoverlap`, and then slightly overlapping these boxes by the same amount.

Chrome and Adobe Acrobat Reader generally work correctly even with no overlap (`backgroundcolorboxoverlap=0pt`). Many PDF previews in  $\TeX$  editing software and PDF readers need an overlap in the 0.1pt to 0.15pt range. Whether Firefox needs overlap, and if so the amount needed, seems to depend on resolution and possibly operating system. Safari and macOS Preview may show artifacts, particularly when zoomed out, even with overlaps of 0.25pt or greater.

`backgroundcolorpadding` (length) (default: none)

Padding when `backgroundcolor` is set. For `\Verb`, `\UseVerb`, `\EscVerb`, and `BVerbatim`, this sets `\fboxsep` for the `\colorbox` that is used to create the background color. For `Verbatim`, `fancyvrb`'s frame options are used instead, particularly `framesep` and `fillcolor`.

For `Verbatim`, this amounts to a shortcut for a combination of frame-related settings that provide padding when there is a background color. This is only intended for cases with `frame=none` or `frame=single`. In other cases, it may be better to modify the `fancyvrb` frame settings directly, and it may be necessary to modify the `fancyvrb` internals to obtain padding on all sides.

For `Verbatim`, if `frame=single`, then this is equivalent to the following settings:

- `framesep=\backgroundcolorpadding`
- `fillcolor=\FancyVerbBackgroundColor`

For `Verbatim`, if `frame` has any value other than `single`, then this is equivalent to the following settings:

- `frame=single`
- `framerule=0pt`
- `rulecolor=\FancyVerbBackgroundColor`
- `framesep=\backgroundcolorpadding`
- `fillcolor=\FancyVerbBackgroundColor`

`backgroundcolorvphantom` (macro) (default: `\vphantom{"Apgjy}`)

`\vphantom` or similar macro such as `\strut` that is inserted at the beginning of each line of text using `backgroundcolor`. This allows the height of the background for each line of text to be customized, especially for `\Verb`, `\EscVerb`, and `BVerbatim`. `backgroundcolorvphantom` will typically have no effect on `Verbatim`-based environments unless it is set to a size larger than `\strut`.

`bgcolor` (string) (default: none)

Alias for `backgroundcolor`.

`bgcolorboxoverlap` (length) (default: 0.25pt)  
Alias for `backgroundcolorboxoverlap`.

`bgcolorpadding` (length) (default: none)  
Alias for `backgroundcolorpadding`.

`bgcolorvphantom` (macro) (default: `\vphantom{"Apgjy}`)  
Alias for `backgroundcolorvphantom`.

`beameroverlays` (boolean) (default: false)  
Give the `<` and `>` characters their normal text meanings, so that `beameroverlays` of the form `\only<1>{...}` will work. Note that something like `commandchars=\\{\}` is required separately to enable macros. This is not incorporated in the `beameroverlays` option because essentially arbitrary command characters could be used; only the `<` and `>` characters are hard-coded for overlays.

With some font encodings and language settings, `beameroverlays` prevents literal (non-overlay) `<` and `>` characters from appearing correctly, so they must be inserted using commands.

`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> |
|---|--------------------------|

`extra` (boolean) (default: true)  
Use `fvextra` reimplementations of `fancyvrb` commands and environments when available. For example, use `fvextra`'s reimplemented `\Verb` that works (with a few limitations) inside other commands, rather than the original `fancyvrb` implementation that essentially functions as `\texttt` inside other commands.

`fontencoding` (string) (default: `<document font encoding>`)  
Set the font encoding inside `fancyvrb` commands and environments. Setting `fontencoding=none` resets to the default document font encoding.

`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 ampersand `&`, caret `^`, and underscore `_` have their normal math meanings.

This is equivalent to

codes={\catcode`\\$=3\catcode`\&=4\catcode`\^=7\catcode`\\_ =8}

`mathscape` 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 9.3).

`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 style="margin: 0;">\begin{Verbatim}[numbers=left, stepnumber=2,                     numberfirstline]  First line Second line Third line Fourth line \end{Verbatim}</pre> |
| <pre style="margin: 0;"> 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 style="margin: 0;">\begin{Verbatim}[numbers=both] First line Second line Third line Fourth line \end{Verbatim}</pre> | <pre style="margin: 0;"> 1 First line      1 2 Second line    2 3 Third line     3 4 Fourth line    4</pre> |
|---|---|

`retokenize` (boolean) (default: `false`)

By default, `\UseVerb` inserts saved `verbatim` material with the catcodes (`commandchars`, `codes`, etc.) under which it was originally saved with `\SaveVerb`. When `retokenize` is used, the saved `verbatim` material is retokenized under the settings in place at `\UseVerb`.

This only applies to the reimplemented `\UseVerb`, when paired with the reimplemented `\SaveVerb`. It may be extended to environments (`\UseVerbatim`, etc.) in the future, if the relevant commands and environments are reimplemented.

`space` (macro) (default: `_`)

Redefine the visible space character. Note that this is only used if `showspaces=true`. The color of the character may be set with `spacecolor`.

`spacebreak` (macro) (default: `\discretionary{}{}{}`)

This determines the break that is inserted around spaces when `breaklines=true` and one or more of the following conditions applies: `breakcollapsespaces=false`, `showspaces=true`, or the space is affected by `breakbefore` or `breakafter`. If it is redefined, it should typically be similar to `\FancyVerbBreakAnywhereBreak`, `\FancyVerbBreakBeforeBreak`, and `\FancyVerbBreakAfterBreak` to obtain consistent breaks.

`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}
```

---

One two three

`stepnumberfromfirst` (boolean) (default: false)  
By default, when line numbering is used with `stepnumber ≠ 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 ≠ 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 10.2).

```

\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.

`fvextra` 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.

**vargsingleline** (boolean) (default: false)  
 This determines whether `fvextra`'s `\Verb` and `\SaveVerb` take multi-line (but not multi-paragraph) verbatim arguments, or if they instead require arguments to be on a single line like the original `fancyvrb` commands.

## 4 General commands

### 4.1 Inline-only settings with `\fvlineset`

`\fvlineset{options}`

This is like `\fvset`, except that options only apply to commands that typeset inline verbatim, like `\Verb` and `\EscVerb`. Settings from `\fvlineset` override those from `\fvset`.

Note that `\fvlineset` only works with commands that are reimplemented, patched, or defined by `fvextra`; it is not compatible with the original `fancyvrb` definitions.

### 4.2 Custom formatting for inline commands like `\Verb` with `\FancyVerbFormatInline`

`\FancyVerbFormatInline`

This can be used to apply custom formatting to inline verbatim text created with commands like `\Verb`. It only works with commands that are reimplemented, patched, or defined by `fvextra`; it is not compatible with the original `fancyvrb` definitions. The default definition does nothing; it is equivalent to `\newcommand{\FancyVerbFormatInline}[1]{#1}`.

This is the inline equivalent of `\FancyVerbFormatLine` and `\FancyVerbFormatText`. In the inline context, there is no need to distinguish between entire line formatting and only text formatting, so only `\FancyVerbFormatInline` exists.

### 4.3 Custom formatting for environments like `Verbatim` with `\FancyVerbFormatLine` and `\FancyVerbFormatText`

`\FancyVerbFormatLine`

`\FancyVerbFormatText`

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

`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.<sup>2</sup> 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.

<sup>1</sup>The 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.

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

```

\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}

```

---

```

Some text that proceeds for a while and finally wraps onto
↪ another line

Some more text

```

## 5 Reimplemented commands

`fvextra` reimplements parts of `fancyvrb`. These new implementations stay close to the original definitions while allowing for new features that otherwise would not be possible. Reimplemented versions are used by default. The original implementations may be used via `\fvset{extra=false}` or by using `extra=false` in the optional arguments to a command or environment.

Reimplemented commands restrict the scope of catcode-related options compared to the original `fancyvrb` versions. This prevents catcode-related options from interfering with new features such as `\FancyVerbFormatInline`. With `fvextra`, the `codes` option should only be used for catcode modifications. Including non-catcode commands in `codes` will typically have no effect, unlike with `fancyvrb`. If you want to customize verbatim content using general commands, consider `formatcom`.

### 5.1 `\Verb`

```
\Verb*[(options)](delim char or {)(text)(delim char or }
```

The new `\Verb` works as expected (with a few limitations) inside other commands. It even works in movable arguments (for example, in `\section`), and is compatible with `hyperref` for generating PDF strings (for example, PDF bookmarks). The `fancyvrb` definition did work inside some other commands, but essentially functioned as `\texttt` in that context.

By default, `\Verb` takes a multi-line (but not multi-paragraph) verbatim argument. To restore the `fancyvrb` behavior of requiring a single-line argument, set `vargsingleline=true`.

`\Verb` is compatible with `breaklines` and the relevant line-breaking options.

Like the original `fancyvrb` implementation, the new `\Verb` can be starred (`\Verb*`) and accepts optional arguments. While `fancyvrb`'s starred command `\Verb*` is a shortcut for `showspaces`, `fvextra`'s `\Verb*` is a shortcut for both `showspaces` and `showtabs`. This is more similar to the current behavior of `LATEX`'s `\verb*`, except that `\verb*` converts tabs into visible spaces instead of displaying them as visible tabs.

**Delimiters** A repeated character like normal `\verb`, or a pair of curly braces `{...}`. If curly braces are used, then `<text>` cannot contain unpaired curly braces. Note that curly braces should be preferred when using `\Verb` inside other commands, and curly braces are *required* when `\Verb` is in a movable argument, such as in a `\section`. Non-ASCII characters now work as delimiters under pdfTeX with inputenc using UTF-8.<sup>3</sup> For example, `\Verb$verbs` now works as expected.

**Limitations inside other commands** While the new `\Verb` does work inside arbitrary other commands, there are a few limitations.

- `#` and `%` cannot be used. If you need them, consider `\EscVerb` or perhaps `\SaveVerb` plus `\UseVerb`.
- Curly braces are only allowed in pairs.
- Multiple adjacent spaces will be collapsed into a single space.
- Be careful with backslashes. A backslash that is followed by one or more ASCII letters will cause a following space to be lost, if the space is not immediately followed by an ASCII letter. For example, `\Verb{\r \n}` becomes `\r\n`, but `\Verb{\r n}` becomes `\r n`. Basically, anything that looks like a L<sup>A</sup>T<sub>E</sub>X command (control word) will gobble following spaces, unless the next character after the spaces is an ASCII letter.
- A single `^` is fine, but avoid `^^` because it will serve as an escape sequence for an ASCII command character.

**Using in movable arguments** `\Verb` works automatically in movable arguments, such as in a `\section`. `\protect` or similar measures are not needed for `\Verb` itself, or for any of its arguments, and should not be used. `\Verb` performs operations that amount to applying `\protect` to all of these automatically.

**hyperref PDF strings** `\Verb` is compatible with `hyperref` for generating PDF strings such as PDF bookmarks. Note that the PDF strings are *always* a literal rendering of the verbatim text, with all `fancyvrb` options ignored. For example, things like `showspaces` and `commandchars` have no effect. If you need options to be applied to obtain desired PDF strings, consider a custom approach, perhaps using `\texorpdfstring`.

**Line breaking** `breaklines` allows breaks at spaces. `breakbefore`, `breakafter`, and `breakanywhere` function as expected, as do things like `breakaftersymbolpre` and `breakaftersymbolpost`. Break options that are only applicable to block text like a `Verbatim` environment do not have any effect. For example, `breakindent` and `breaksymbol` do nothing.

## 5.2 `\SaveVerb`

```
\SaveVerb[options]{name}{delim char or { }<text>{delim char or } }
```

`\SaveVerb` is reimplemented so that it is equivalent to the reimplemented `\Verb`. Like the new `\Verb`, it accepts `<text>` delimited by a pair of curly braces

<sup>3</sup>Under pdfTeX, non-ASCII code points are processed at the byte rather than code point level, so `\Verb` must treat a sequence of multiple bytes as the delimiter.

{...}. It supports `\fvlineset`. It also adds support for the new `retokenize` option for `\UseVerb`.

By default, `\SaveVerb` takes a multi-line (but not multi-paragraph) verbatim argument. To restore the `fancyvrb` behavior of requiring a single-line argument, set `vargsingleline=true`.

### 5.3 `\UseVerb`

`\UseVerb*[{options}]{<name>}`

`\UseVerb` is reimplemented so that it is equivalent to the reimplemented `\Verb`. It supports `\fvlineset` and `breaklines`.

Like `\Verb`, `\UseVerb` is compatible with `hyperref` for generating PDF strings such as PDF bookmarks. Note that the PDF strings are *always* a literal rendering of the verbatim text, with all `fancyvrb` options ignored. For example, things like `showspaces` and `commandchars` have no effect. The new option `retokenize` also has no effect. If you need options to be applied to obtain desired PDF strings, consider a custom approach, perhaps using `\texorpdfstring`

There is a new option `retokenize` for `\UseVerb`. By default, `\UseVerb` inserts saved verbatim material with the catcodes (`commandchars`, `codes`, etc.) under which it was originally saved with `\SaveVerb`. When `retokenize` is used, the saved verbatim material is retokenized under the settings in place at `\UseVerb`.

For example, consider `\SaveVerb{save}{\textcolor{red}{\#%}}`:

- `\UseVerb{save} ⇒ \textcolor{red}{\#%}`
- `\UseVerb[commandchars=\\{\}]{save} ⇒ \textcolor{red}{\#%}`
- `\UseVerb[retokenize, commandchars=\\{\}]{save} ⇒ \#%`

## 6 New commands and environments

### 6.1 `\EscVerb`

`\EscVerb*[{options}]{<backslash-escaped text>}`

This is like `\Verb` but with backslash escapes to allow for characters such as `#` and `%`. For example, `\EscVerb{\Verb{\#%}}` gives `\Verb{\#%}`. It behaves exactly the same regardless of whether it is used inside another command. Like the reimplemented `\Verb`, it works in movable arguments (for example, in `\section`), and is compatible with `hyperref` for generating PDF strings (for example, PDF bookmarks).

**Delimiters** Text must *always* be delimited with a pair of curly braces `{...}`.

This ensures that `\EscVerb` is always used in the same manner regardless of whether it is inside another command.

#### Escaping rules

- Only printable, non-alphanumeric ASCII characters (symbols, punctuation) can be escaped with backslashes.<sup>4</sup>
- Always escape these characters: `\`, `%`, `#`.

<sup>4</sup>Allowing backslash escapes of letters would lead to ambiguity regarding spaces; see `\Verb`.



- Escape spaces when there are more than one in a row.
- Escape `~` if there are more than one in a row.
- Escape unpaired curly braces.
- Additional symbols or punctuation characters may require escaping if they are made `\active`, depending on their definitions.

**Using in movable arguments** `\EscVerb` works automatically in movable arguments, such as in a `\section`. `\protect` or similar measures are not needed for `\EscVerb` itself, or for any of its arguments, and should not be used. `\EscVerb` performs operations that amount to applying `\protect` to all of these automatically.

**hyperref PDF strings** `\EscVerb` is compatible with `hyperref` for generating PDF strings such as PDF bookmarks. Note that the PDF strings are *always* a literal rendering of the verbatim text after backslash escapes have been applied, with all `fancyvrb` options ignored. For example, things like `showspaces` and `commandchars` have no effect. If you need options to be applied to obtain desired PDF strings, consider a custom approach, perhaps using `\texorpdfstring`.

## 6.2 VerbatimWrite

```
\begin{VerbatimEnv}[\langle options \rangle]
  \langle single line \rangle
\end{VerbatimEnv}
```

This is an environment variant of `\Verb`. The environment must contain only a single line of text, and the closing `\end{VerbatimEnv}` must be on a line by itself. The `\langle options \rangle` and `\langle single line \rangle` are read and then passed on to `\Verb` internally for actual typesetting.

While `VerbatimEnv` can be used by document authors, it is primarily intended for package creators. For example, it is used in `minted` to implement `\mintinline`. In that case, highlighted code is always generated within a `Verbatim` environment. It is possible to process this as inline rather than block verbatim by `\letting \Verbatim` to `\VerbatimEnv`.

|   |                                |
|---|--------------------------------|
| <pre>BEFORE\begin{VerbatimEnv} _inline_ \end{VerbatimEnv} AFTER</pre> | <pre>BEFORE_inline_AFTER</pre> |
|---|--------------------------------|

`VerbatimEnv` is not implemented using the typical `fancyvrb` environment implementation style, so it is not compatible with `\RecustomVerbatimEnvironment`.

## 6.3 VerbatimWrite

```
\begin{VerbatimWrite}[\langle opt \rangle]
  \langle lines \rangle
\end{VerbatimWrite}
```

This writes environment contents verbatim to an external file. It is similar to `fancyvrb`'s `VerbatimOut`, except that (1) it allows writing to a file multiple times (multiple environments can write to the same file) and (2) by default it uses `\detokenize` to guarantee truly verbatim output.

By default, all `fancyvrb` options except for `VerbatimWrite`-specific options are ignored. This can be customized on a per-environment basis via environment optional arguments.

Options defined specifically for `VerbatimWrite`:

`writefilehandle` (file handle) (default: *none*)  
File handle for writing. For example,

```
\newwrite\myfile
\immediate\openout\myfile=myfile.txt\relax

\begin{VerbatimWrite}[writefilehandle=\myfile]
...
\end{VerbatimWrite}

\immediate\closeout\myfile
```

`writer` (macro) (default: `\FancyVerbDefaultWriter`)  
This is the macro that processes each line of text in the environment and then writes it to file. This is the default implementation:

```
\def\FancyVerbDefaultWriter#1{%
  \immediate\write\FancyVerbWriteFileHandle{\detokenize{#1}}
```

## 6.4 Buffers

### 6.4.1 VerbatimBuffer

```
\begin{VerbatimBuffer}[(opt)]
  <lines>
\end{VerbatimBuffer}
```

This environment stores its contents verbatim in a “buffer,” a sequence of numbered macros each of which contains one line of the environment. The “buffered” lines can then be iterated over for further processing or later use. This is similar to `fancyvrb`’s `SaveVerbatim`, which saves an environment for later use. `VerbatimBuffer` offers additional flexibility by capturing truly verbatim environment contents using `\detokenize` and saving environment contents in a format designed for further processing.

By default, all `fancyvrb` options except for `VerbatimBuffer`-specific options are ignored. This can be customized on a per-environment basis via environment optional arguments.

Below is an extended example that demonstrates what is possible with `VerbatimBuffer` combined with various buffer commands. This uses `\ifstrequal` from the `etoolbox` package.

- `\setformatter` defines an empty `\formatter` macro. Then it defines a `\lineprocessor` macro that takes a buffer line as an argument. If this line contains only the text “red”, then `\lineprocessor` redefines `\formatter` to `\color{red}` and invokes `\IterateBufferBreak` to stop iteration over the buffer. Finally, `\IterateBuffer` is used to apply `\lineprocessor` to each line in the buffer, until the final line is reached or `\IterateBufferBreak` is invoked.

- `afterbuffer` involves two steps: (1) `\setformatter` loops through the buffer and defines `\formatter` based on the buffer contents, and (2) `\VerbatimInsertBuffer` typesets the buffer, using `formatcom=\formatter` to format the text based on whether any line contains only the text “red”.

```

\def\setformatter{%
  \def\formatter{}%
  \def\lineprocessor##1{%
    \ifstrequal{##1}{red}{\def\formatter{\color{red}}\IterateBufferBreak}{}}%
  \IterateBuffer{\lineprocessor}}

\begin{VerbatimBuffer}[
  afterbuffer={\setformatter\VerbatimInsertBuffer[formatcom=\formatter]}
]
first
second
red
\end{VerbatimBuffer}

```

---

```

first
second
red

```

Here is the same example, but rewritten to use a global buffer with custom buffer name instead.

```

\begin{VerbatimBuffer}[globalbuffer, buffername=exbuff]
first
second
red
\end{VerbatimBuffer}

\def\formatter{}
\def\lineprocessor#1{%
  \ifstrequal{#1}{red}{\def\formatter{\color{red}}\IterateBufferBreak}{}}
\IterateBuffer[buffername=exbuff]{\lineprocessor}

\VerbatimInsertBuffer[buffername=exbuff, formatcom=\formatter]

```

---

```

first
second
red

```

Options defined specifically for `VerbatimBuffer`:

`afterbuffer` (macro) (default: *none*)  
 Macro or macros invoked at the end of the environment, after all lines of the environment have been buffered. This is outside the `\begingroup... \endgroup` that wraps verbatim processing, so `fancyvrb` settings are no longer active. However, the buffer line macros and the buffer length macro are still accessible even when `globalbuffer=false`.

When `afterbuffer` is used to typeset the buffer, the typeset buffer may contain `VerbatimBuffer` or environments based on it. Typically, nested buffering should be avoided for a given buffer; a different buffer should be used at each level of nesting. Otherwise, for nested `VerbatimBuffer` environments, the buffer will contain the contents of the outermost `VerbatimBuffer` environment concatenated with the contents of nested environments.

The current buffer depth is available in `\FancyVerbBufferDepth`. This has a value of 0 outside of any `VerbatimBuffer` environments, a value of 1 within the outermost `VerbatimBuffer` environment, and so forth. This can be used to automatically generate a unique `buffername` at a given nesting depth.

`bufferer` (macro) (default: `\FancyVerbDefaultBufferer`)  
 This is the macro that adds lines to the buffer. The default is designed to create a truly verbatim buffer via `\detokenize`. This can be customized if you wish to use `fancyvrb` options related to catcodes to create a buffer that is only partially verbatim (that contains macros).

```
\def\FancyVerbDefaultBufferer#1{%
  \expandafter\xdef\csname\FancyVerbBufferLineName\FancyVerbBufferIndex\endcsname{%
    \detokenize{#1}}
```

A custom `bufferer` must take a single argument `#1` (a line of the environment text) and ultimately store the processed line in a macro called

```
\csname\FancyVerbBufferLineName\FancyVerbBufferIndex\endcsname
```

This macro must be defined globally, so `\xdef` or `\gdef` is necessary (this does not interfere with scoping from `globalbuffer`). Otherwise, there are no restrictions. The `\xdef` and `\detokenize` in the default definition guarantee that the buffer consists only of the literal text from the environment, but this is not required for a custom `bufferer`.

`bufferlengthname` (string) (default: `FancyVerbBufferLength`)  
 Name of the macro storing the length of the buffer. This is the number of lines stored.

Macros that operate within a `VerbatimBuffer` environment can access the current value of `bufferlengthname` via `\FancyVerbBufferLengthName`.

`bufferlinename` (string) (default: `FancyVerbBufferLine`)  
 The base name of the buffer line macros. The default is `FancyVerbBufferLine`, which will result in buffer macros `\FancyVerbBufferLine<n>` with integer `n` greater than or equal to one and less than or equal to the number of lines (one-based indexing). Since buffer macro names contain a number, they must be accessed differently than typical macros:

```
\csname FancyVerbBufferLine<n>\endcsname
\@nameuse{FancyVerbBufferLine<n>}
```

If the buffer macros are looped over with a counter that is incremented, then `<n>` should be the counter value `\arabic{<counter>}`.

Macros that operate within a `VerbatimBuffer` environment can access the current value of `bufferlinename` via `\FancyVerbBufferLineName`.

`buffername` (string) (default: `<none>`)  
Shortcut for setting `bufferlengthname` and `bufferlinename` simultaneously, using the same root name. This sets `bufferlengthname` to `<buffername>length` and `bufferlinename` to `<buffername>line`.

`globalbuffer` (bool) (default: `false`)  
This determines whether buffer line macros are defined globally, that is, whether they are accessible after the end of the `VerbatimBuffer` environment. If the line macros are defined globally, then the buffer length macro is also increased appropriately outside the environment. `globalbuffer` does not affect any `afterbuffer` macro, since that is invoked inside the environment.

When buffered lines are used immediately, consider using `afterbuffer` instead of `globalbuffer`. When buffered lines must be used later in a document, consider using `globalbuffer` with custom (and perhaps unique) `bufferlinename` and `bufferlengthname`.

When `globalbuffer=true`, `VerbatimBuffer` environments with the same buffer name will append to a single buffer, so that it ultimately contains the concatenated contents of all environments. A `VerbatimBuffer` environment with `globalbuffer=false` will append to the buffer created by any previous `VerbatimBuffer` that had `globalbuffer=true` and shared the same buffer name. Any `afterbuffer` macro will have access to a buffer containing the concatenated data. At the very end of the environment with `globalbuffer=false`, after any `afterbuffer`, this appended content will be removed. All buffer line macros (from `bufferlinename`) that were created by that environment are “deleted” (`\let` to an undefined macro), and the buffer length macro (from `bufferlengthname`) is reduced proportionally.

#### 6.4.2 `\VerbatimInsertBuffer`

`\VerbatimInsertBuffer[<options>]`

This inserts an existing buffer created by `VerbatimBuffer` as a verbatim environment. The `Verbatim` environment is used by default, but this can be customized by setting `insertenvname`. `\VerbatimInsertBuffer` modifies `Verbatim` and `BVerbatim` internals to function with a buffer in a command context. See the `VerbatimBuffer` documentation for an example of usage.

Options related to catcodes cause the buffer to be retokenized during typesetting. That is, the `fancyvrb` options used for `\VerbatimInsertBuffer` are not restricted by those that were in effect when `VerbatimBuffer` originally created the buffer, so long as the buffer contains a complete representation of the original `VerbatimBuffer` environment contents.

`\VerbatimInsertBuffer` is not implemented using the typical `fancyvrb` command and environment implementation styles, so it is not compatible with `\RecustomVerbatimCommand` or `\RecustomVerbatimEnvironment`.

Options defined specifically for `\VerbatimInsertBuffer`:

`insertenvname` (string) (default: `Verbatim`)  
 This is the name of the verbatim environment used for inserting the buffer. `insertenvname` can be any `Verbatim`- or `BVerbatim`-based environment. Environments defined with `\CustomVerbatimEnvironment` and `\RecustomVerbatimEnvironment` are supported. User-implemented environments that serve as wrappers around `Verbatim` or `BVerbatim` should typically be compatible so long as they accept `fancyvrb`/`fvextra` optional arguments in the same way as `Verbatim` and `BVerbatim`.

### 6.4.3 `\VerbatimClearBuffer`

`\VerbatimClearBuffer` [*options*]

Clear an existing buffer created with `VerbatimBuffer`. `\global\let` all buffer line macros to an undefined macro and set the buffer length macro to zero.

### 6.4.4 `\InsertBuffer`

`\InsertBuffer` [*options*]

This inserts an existing buffer created with `VerbatimBuffer` so that it is interpreted as  $\LaTeX$ . The result is essentially the same as if the buffered text had been included literally at the insertion point. The buffer is processed with `\scantokens`.

For typesetting verbatim text, `\InsertBuffer` with the `wrapperenv` option set to a verbatim environment can be used as an alternative to `\VerbatimInsertBuffer`. Both typically have similar performance.

`wrapperenvname` (string) (default: *none*)  
 Name of environment used to wrap the buffer. This is inserted within the `\scantokens` that is used to process the buffer, so if `wrapperenvname` is set to a verbatim environment, then it will cause the buffer to be typeset verbatim. This is useful for non-`fancyvrb` verbatim environments that are not supported by `\VerbatimInsertBuffer`.

`wrapperenvopt` (string) (default: *none*)  
 Optional argument [*opt*] passed to wrapper environment `wrapperenvname`.

`wrapperenvarg` (string) (default: *none*)  
 Mandatory argument `{arg}` passed to wrapper environment `wrapperenvname`.

```

\begin{VerbatimBuffer}[buffername=ins, globalbuffer]
Text. Math  $f(x)$ . Verb \Verb|~#\{|.
\begin{Verbatim}
Verbatim. ~#\{|.
\end{Verbatim}
More text.
\end{VerbatimBuffer}

\textbf{BEFORE}\InsertBuffer[buffername=ins]\textbf{AFTER}
-----
BEFOREText. Math  $f(x)$ . Verb ~#\{|.

Verbatim. ~#\{|.

More text.AFTER

```

#### 6.4.5 `\ClearBuffer`

`\ClearBuffer[options]` Alias for `\VerbatimClearBuffer`.

#### 6.4.6 `\BufferMdfivesum`

`\BufferMdfivesum` Calculate the MD5 sum of the current buffer, using `\pdf@mdfivesum` from `pdftexcmds`.

`\BufferMdfivesum` operates on the current buffer. Because it is fully expandable, it cannot take an optional argument to switch buffers. To switch buffers, use something like `\fvset{buffername=...}` before `\BufferMdfivesum`.

```
\begin{VerbatimBuffer}[afterbuffer=\xdef\bufferhash{\BufferMdfivesum}]
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
 eiusmod tempor incididunt ut labore et dolore magna aliqua.
\end{VerbatimBuffer}

\texttt{\bufferhash}
-----
8B5FBE4F759EB41199344C22AE311B70
```

#### 6.4.7 `\IterateBuffer`

`\IterateBuffer[options]{macro}`

Iterate over buffer from beginning to end, invoking *macro* on each line. *macro* must be a macro or sequence of macros that consumes a single brace-delimited argument, which is the current line in the buffer. `\IterateBufferBreak` is available to break iteration immediately after the current *macro* invocation completes. During iteration, `\FancyVerbBufferLengthName`, `\FancyVerbBufferLineName`, and `\FancyVerbBufferIndex` can be used to access the name of the current buffer and index within it.

#### 6.4.8 `\WriteBuffer`

`\WriteBuffer[options]` This writes the current buffer to an external file, using the `writefilehandle` and `writer` options from `VerbatimWrite`. It is the buffer equivalent of `VerbatimWrite`.

## 7 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`, 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`.

When a line is broken, the result must fit on a single page. There is no support for breaking a line across multiple pages.

## 7.1 Line breaking options

Options are provided for customizing typical line breaking features. See section 7.3 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 #, {, }, %, [, ], and the comma ,; the backslash \ may be obtained via \\ and the space via \space).<sup>5</sup>

For an alternative, see **breakbefore**. When **breakbefore** and **breakafter** are used for the same character, **breakbeforeinrun** and **breakafterinrun** 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 7.3 for details, and consider **breaknonspaceingroup** as a solution in simple cases.

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

_____

some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCould
↪ NeverFitOnOneLine'
```

**breakafterinrun** (boolean) (default: *false*)  
When **breakafter** is used, insert breaks within runs of identical characters. If *false*, treat sequences of identical characters as a unit that cannot contain breaks. When **breakbefore** and **breakafter** are used for the same character, **breakbeforeinrun** and **breakafterinrun** must both have the same setting.

**breakaftersymbolpre** (string) (default: \, \footnotesize\ensurmath{\\_}\rfloor, \\_ )  
The symbol inserted pre-break for breaks inserted by **breakafter**. This does not apply to breaks inserted next to spaces; see **spacebreak**.

**breakaftersymbolpost** (string) (default: *none*)  
The symbol inserted post-break for breaks inserted by **breakafter**. This does not apply to breaks inserted next to spaces; see **spacebreak**.

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

<sup>5</sup>**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. **fextra** 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.



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 7.3 for details, and consider `breaknonspaceingroup` as a solution in simple cases.

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

-----

some_string = 'SomeTextThatGoesOnAndOnForSoLongThatItCouldNeve
→ rFitOnOneLine'
```

`breakanywheresymbolpre` (string) (default: `\,\footnotesize\ensuremath{\_}\rfloor`, `_j`)  
The symbol inserted pre-break for breaks inserted by `breakanywhere`. This does not apply to breaks inserted next to spaces; see `spacebreak`.

`breakanywhereinlinestretch` (length) (default: `<none>`)  
Stretch glue to insert at potential `breakanywhere` break locations in inline contexts, to give better line widths and avoid overfull `\hbox`. This allows the spacing between adjacent non-space characters to stretch, so it should not be used when column alignment is important. For typical line lengths, values between `0.01em` and `0.02em` should be sufficient to provide a cumulative stretch per line that is equal to or greater than the width of one character.

This is typically not needed in cases where an overfull `\hbox` only overflows by tiny amount, perhaps a fraction of a pt. In those cases, the overfull `\hbox` could be ignored, `\hfuzz` could be set to `1pt` or `2pt` to suppress tiny overfull `\hbox` warnings, or `breakanywheresymbolpre` might be redefined to adjust spacing.

Implementation: Before each `breakanywhere` break location, this inserts the following sequence of macros:

```
\nobreak\hspace{0pt plus \FV@breaknonspaceinlinestretch}
```

`breakanywheresymbolpost` (string) (default: `<none>`)  
The symbol inserted post-break for breaks inserted by `breakanywhere`. This does not apply to breaks inserted next to spaces; see `spacebreak`.

`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 `#`, `{`, `}`, `%`,

[, ], and the comma ;; the backslash \ may be obtained via \\ and the space via \space).<sup>6</sup>

For an alternative, see `breakafter`. When `breakbefore` and `breakafter` are used for the same character, `breakbeforeinrun` and `breakafterinrun` 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 7.3 for details, and consider `breaknonspaceingroup` as a solution in simple cases.

```

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

-----

some_string = 'SomeTextThatGoesOn
↪ AndOnForSoLongThatItCouldNeverFitOnOneLine'
```

- `breakbeforeinrun` (boolean) (default: `false`)  
When `breakbefore` is used, insert breaks within runs of identical characters. If `false`, treat sequences of identical characters as a unit that cannot contain breaks. When `breakbefore` and `breakafter` are used for the same character, `breakbeforeinrun` and `breakafterinrun` must both have the same setting.
- `breakbeforesymbolpre` (string) (default: `\,\footnotesize\ensuremath{\_}\rfloor`, `_j`)  
The symbol inserted pre-break for breaks inserted by `breakbefore`. This does not apply to breaks inserted next to spaces; see `spacebreak`.
- `breakbeforesymbolpost` (string) (default: `<none>`)  
The symbol inserted post-break for breaks inserted by `breakbefore`. This does not apply to breaks inserted next to spaces; see `spacebreak`.
- `breakcollapsespaces` (bool) (default: `true`)  
When `true` (default), a line break within a run of regular spaces (`showspaces=false`) replaces all spaces with a single break, and the wrapped line after the break starts with a non-space character. When `false`, a line break within a run of regular spaces preserves all spaces, and the wrapped line after the break may start with one or more spaces. This causes regular spaces to behave exactly like the visible spaces produced with `showspaces`; both give identical line breaks, with the only difference being the appearance of spaces.
- `breakindent` (dimension) (default: `<breakindentnchars>`)  
When a line is broken, indent the continuation lines by this amount. When

<sup>6</sup>`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. `fvextra` 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.

`breakautoindent` and `breakindent` are used together, the indentations add. This indentation is combined with `breaksymbolindentleft` to give the total actual left indentation.

`breakindentnchars` (integer) (default: 0)  
 This allows `breakindent` to be specified as an integer number of characters rather than as a dimension (assumes a fixed-width font).

`breaklines` (boolean) (default: false)  
 Automatically break long lines.

**Limitations for verbatim environments/block text**, such as `Verbatim`: When a line is broken, the result must fit on a single page. There is no support for breaking a line across multiple pages.<sup>7</sup>

**Limitations for verbatim commands/inline text**, such as `\Verb`: When a line break is inserted, the text may still overflow into the margin or cause an overfull `\hbox`, depending on hyphenation settings and various penalties related to line breaks. It may be possible to avoid this by allowing additional break locations with `breakbefore`, `breakafter`, or `breakanywhere`. Small overfull `\hbox` warnings can be suppressed by setting `\hfuzz` to a larger value, for example setting it to 2pt instead of the default 0.1pt. It is also possible to combine `breakanywhere` with `breakanywhereinlinestretch` to allow flexible spacing between adjacent non-space characters. In cases where it is better to break before the margin rather than overflowing into the margin, consider setting `\emergencystretch`.

By default, automatic breaks occur at spaces (even when `showspaces=true`). Use `breakanywhere` to enable breaking anywhere; use `breakbefore` and `breakafter` for more fine-tuned breaking.

|  |  |
|--|--|
| <pre>...text. \begin{Verbatim}[breaklines] def f(x):     return 'Some text ' + str(x) \end{Verbatim}</pre> | <pre>...text. def f(x):     return 'Some text ' +     ↪ str(x)</pre> |
|--|--|

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.

<sup>7</sup>Following the implementation in `fancyvrb`, each line is typeset within an `\hbox`, so page breaks are not possible.

```

\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'

```

Beginning in version 1.6, automatic line breaks work with `showspaces=true` by default. Defining `breakbefore` or `breakafter` for `\space` is no longer necessary. For example,

```

\begin{Verbatim}[breaklines, showspaces]
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'

```

`breaknonspaceingroup` (boolean) (default: `false`)

By using `commandchars`, it is possible to include  $\LaTeX$  commands within otherwise verbatim text. In these cases, there can be groups (typically `{...}` but depends on `commandchars`) within verbatim. Spaces within groups are treated as potential line break locations when `breaklines=true`, but by default no other break locations are inserted (`breakbefore`, `breakafter`, `breakanywhere`). This is because inserting non-space break locations can interfere with command functionality. For example, in `\textcolor{red}{text}`, breaks shouldn't be inserted within `red`.

`breaknonspaceingroup` allows non-space breaks to be inserted within groups. This option should only be used when `commandchars` is including  $\LaTeX$  commands that do not take optional arguments and only take mandatory arguments that are typeset. Something like `\textit{text}` is fine, but `\textcolor{red}{text}` is not because one of the mandatory arguments is not typeset but rather provides a setting. For more complex commands, it is typically better to redefine them to insert breaks in appropriate locations using `\FancyVerbBreakStart... \FancyVerbBreakStop`.

`breakpreferspaces` (boolean) (default: `true`)  
 This determines whether line breaks are preferentially inserted at normal spaces (`breakcollapsespaces=true`, `showspaces=false`) rather than at other locations allowed by `breakbefore`, `breakafter`, or `breakanywhere`.

Using `breakpreferspaces=false` with `breakanywhere=true` will typically result in all broken segments of a line going all the way to the right margin.

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

`breaksymbolleft` (string) (default: `\tiny\ensuremath{\hookrightarrow}`, `\rightarrow`)  
 The symbol used at the beginning (left) of continuation lines when `breaklines=true`. To have no symbol, simply set `breaksymbolleft` to an empty string ("`=`", "`={}`"). 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.

`breaksymbolright` (string) (default: `<none>`)  
 The symbol used at breaks (right) when `breaklines=true`. Does not appear at the end of the very last segment of a broken line.

`breaksymbolindent` (dimension) (default: `<breaksymbolindentleftnchars>`)  
 Alias for `breaksymbolindentleft`.

`breaksymbolindentnchars` (integer) (default: `<breaksymbolindentleftnchars>`)  
 Alias for `breaksymbolindentleftnchars`.

`breaksymbolindentleft` (dimension) (default: `<breaksymbolindentleftnchars>`)  
 The extra left indentation that is provided to make room for `breaksymbolleft`. This indentation is only applied when there is a `breaksymbolleft`.

`breaksymbolindentleftnchars` (integer) (default: 4)  
 This allows `breaksymbolindentleft` to be specified as an integer number of characters rather than as a dimension (assumes a fixed-width font).

`breaksymbolindentright` (dimension) (default: `<breaksymbolindentrightnchars>`)  
 The extra right indentation that is provided to make room for `breaksymbolright`. This indentation is only applied when there is a `breaksymbolright`.

`breaksymbolindentrightnchars` (integer) (default: 4)  
 This allows `breaksymbolindentright` to be specified as an integer number of characters rather than as a dimension (assumes a fixed-width font).

`breaksymbolsep` (dimension) (default: `<breaksymbolsepleftnchars>`)  
 Alias for `breaksymbolsepleft`.

`breaksymbolsepnchars` (integer) (default: `<breaksymbolsepleftnchars>`)  
 Alias for `breaksymbolsepleftnchars`.

`breaksymbolsepleft` (dimension) (default: `<breaksymbolsepleftnchars>`)  
 The separation between the `breaksymbolleft` and the adjacent text.

- breaksymbolsepleftnchars** (integer) (default: 2)  
 Allows **breaksymbolsepleft** to be specified as an integer number of characters rather than as a dimension (assumes a fixed-width font).
- breaksymbolsepright** (dimension) (default:  $\langle\text{breaksymbolseprightnchars}\rangle$ )  
 The *minimum* separation between the **breaksymbolright** and the adjacent text. This is the separation between **breaksymbolright** and the furthest extent to which adjacent text could reach. In practice, **\linewidth** will typically not be an exact integer multiple of the character width (assuming a fixed-width font), so the actual separation between the **breaksymbolright** and adjacent text will generally be larger than **breaksymbolsepright**. This ensures that break symbols have the same spacing from the margins on both left and right. If the same spacing from text is desired instead, **breaksymbolsepright** may be adjusted. (See the definition of **\FV@makeLineNumber** for implementation details.)
- breaksymbolseprightnchars** (integer) (default: 2)  
 Allows **breaksymbolsepright** to be specified as an integer number of characters rather than as a dimension (assumes a fixed-width font).
- spacebreak** (macro) (default:  $\langle\text{\discretionary}{ }{ }{ }\rangle$ )  
 This determines the break that is inserted around spaces when **breaklines=true** and one or more of the following conditions applies: **breakcollapsespaces=false**, **showspaces=true**, or the space is affected by **breakbefore** or **breakafter**. If it is redefined, it should typically be similar to **\FancyVerbBreakAnywhereBreak**, **\FancyVerbBreakBeforeBreak**, and **\FancyVerbBreakAfterBreak** to obtain consistent breaks.

## 7.2 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,

```
\settowidth{\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

```

---

## 7.3 Advanced line breaking

### 7.3.1 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  $\text{\LaTeX}$  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 other-wise 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`.

### 7.3.2 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`.

### 7.3.3 Customizing break behavior

`\FancyVerbBreakAnywhereBreak`

These macros govern the behavior of breaks introduced by `breakanywhere`, `breakbefore`, and `breakafter`. These do not apply to breaks inserted next to spaces; see `spacebreak`.

`\FancyVerbBreakBeforeBreak`

`\FancyVerbBreakAfterBreak`

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.

## 8 Pygments support

### 8.1 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 8.2).

`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`.

## 8.2 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 `\begin{group}... \end{group}` 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 `\begin{group}... \end{group}` 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`. `<actual_macro>` is the Pygments macro that should actually be used. For `minted` this is `\PYG`; for `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 to use the appropriate style macro.

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

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

`\VerbatimPygments` also improves other features of Pygments token macros, including improving support for Pygments options `escapeinside` and `texcomments`. See the implementation for details.

## 9 Patches

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

### 9.1 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 following <https://tex.stackexchange.com/a/120231/10742>.

### 9.2 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
```

## 9.3 Math mode

### 9.3.1 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
```

### 9.3.2 Symbols and fonts

With `fancyvrb`, ASCII symbols may give errors within math or may not appear as expected. `fvextra` redefines all ASCII symbols so that they will behave as literal characters or (if `\active` outside the verbatim context) as macros within math. This behavior is overridden for specific characters by the `codes`, `commandchars`, and `mathescape` options.

With `fancyvrb`, using a single quotation mark (') in typeset math within verbatim material results in an error rather than a prime symbol (').<sup>8</sup> `fvextra` redefines the 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  $\end{Verbatim}$ 
```

---

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

### 9.4 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.

### 9.5 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.

### 9.6 Command lookahead tokenization

`\FV@Command` is used internally by commands like `\Verb` to read stars (\*) and optional arguments ([...]) before invoking the core of the command. This is redefined so that lookahead tokenizes under a verbatim catcode regime. The original definition could prevent commands like `\Verb` from using characters like %

---

<sup>8</sup>The single quotation mark is made active within verbatim material to prevent ligatures, via `\@noligs`. The default definition is incompatible with math mode.

as delimiters, because the lookahead for a star and optional argument could read the % and give it its normal meaning of comment character. The new definition fixes this, so that commands like `\Verb` behave as closely to `\verb` as possible.

## 10 Additional modifications to `fancyvrb`

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

### 10.1 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 `‘`. `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).

### 10.2 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).

## 11 Undocumented features of `fancyvrb`

`fancyvrb` defines some potentially useful but undocumented features.

### 11.1 Undocumented option

`listparameters` (macro) (default: `\empty`)  
Set list-related lengths to modify spacing around lines of code. For example, `listparameters=\setlength{\topsep}{0pt}` will remove space before and after a `Verbatim` environment.

### 11.2 Undocumented macros

`\FancyVerbTab`

This defines the visible tab character (`\t`) 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 9.1) follows <https://tex.stackexchange.com/a/120231/10742>. While this may be redefined directly, `fvextra` also defines a new option `space`.

## 12 Implementation

### 12.1 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{etoolbox}
2 \RequirePackage{fancyvrb}
3 \RequirePackage{pdftexcmds}
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}}{}}

```

### 12.2 Utility macros

#### 12.2.1 `fancyvrb` space and tab tokens

##### `\FV@ActiveSpaceToken`

Active space for `\ifx` token comparisons.

```

15 \begingroup
16 \catcode\ =\active%
17 \gdef\FV@ActiveSpaceToken{ }%
18 \endgroup%

```

##### `\FV@SpaceCatTen`

Space with catcode 10. Used instead of `\_` and `\space` in some contexts to avoid issues in the event that these are redefined.

```

19 \edef\FV@SpaceCatTen{\detokenize{ }}

```

##### `\FV@FVSpaceToken`

Macro with the same definition as `fancyvrb`'s active space. Useful for `\ifx` comparisons, such as `\@ifnextchar` lookaheads.

```
20 \def\FV@FVSpaceToken{\FV@Space}
```

`\FV@FVTabToken`

Macro with the same definition as `fancyvrb`'s active tab. Useful for `\ifx` comparisons, such as `\@ifnextchar` lookaheads.

```
21 \def\FV@FVTabToken{\FV@Tab}
```

### 12.2.2 ASCII processing

`\FVExtraDoSpecials`

Apply `\do` to all printable, non-alphanumeric ASCII characters (codepoints 0x20 through 0x7E except for alphanumeric characters).

These punctuation marks and symbols are the most likely characters to be made `\active`, so it is convenient to be able to change the catcodes for all of them, not just for those in the `\dospecials` defined in `latex.ltx`:

```
\def\dospecials{\do\ \do\\\do{\do}\do\$\do\&%
\do#\do\^{\do\_ \do%\do\~}
```

If a command takes an argument delimited by a given symbol, but that symbol has been made `\active` and defined as `\outer` (perhaps it is being used as a short `\verb`), then changing the symbol's catcode is the only way to use it as a delimiter.

```
22 \def\FVExtraDoSpecials{%
23 \do\ \do!\do!"\do#\do\$\do%\do\&\do\'\do\(\do\)\do*\do\+\do\,\do\-%
24 \do\.\do\/\do\:\do\;\do\<\do>=\do>\do? \do\@\do\[\do\\\do\]\do\^{\do\_%
25 \do\` \do\{\do\|\do\}\do\~}
```

`\FV@Char@Special:<char>`

Create macros for all printable, non-alphanumeric ASCII characters. This is used in creating backslash escapes that can only be applied to ASCII symbols and punctuation; these macros serve as `\ifcsname` lookups for valid escapes.

```
26 \begingroup
27 \def\do#1{%
28 \expandafter\global\expandafter
29 \let\csname FV@Char@Special:\number`#1\endcsname\relax}
30 \FVExtraDoSpecials
31 \endgroup
```

`\FV@Char@CatTwelve:<char>`

`\FV@Char@CatDetok:<char>`

`\FV@Char@CatActive:<char>`

Macros for accessing printable, non-alphanumeric ASCII characters with various catcodes.

```
32 \begingroup
33 \catcode`\!=0
34 \catcode`\<=1
35 \catcode`\>=2
36 \catcode`\&=14!relax&
37 !edef!FVCharNum<!number`!\>&
38 !catcode`\!=12!relax&
39 !expandafter!gdef!csname!detokenize<FV@Char@CatTwelve:>!FVCharNum!endcsname<\>&
40 !expandafter!xdef!csname!detokenize<FV@Char@CatDetok:>!FVCharNum!endcsname<!detokenize<\>>&
41 !catcode`\!=!active&
42 !expandafter!gdef!csname!detokenize<FV@Char@CatActive:>!FVCharNum!endcsname<\>&
```

```

43 !edef\FVCharNum<!number`!>&
44 !catcode`!={12!relax&
45 !expandafter!gdef!csname!detokenize<FV@Char@CatTwelve:>!FVCharNum!endcsname<{>&
46 !expandafter!xdef!csname!detokenize<FV@Char@CatDetok:>!FVCharNum!endcsname<!detokenize<{}>&
47 !catcode`!={!active&
48 !expandafter!gdef!csname!detokenize<FV@Char@CatActive:>!FVCharNum!endcsname<{>&
49 !edef\FVCharNum<!number`!>&
50 !catcode`!}=12!relax&
51 !expandafter!gdef!csname!detokenize<FV@Char@CatTwelve:>!FVCharNum!endcsname<}>&
52 !expandafter!xdef!csname!detokenize<FV@Char@CatDetok:>!FVCharNum!endcsname<!detokenize<{}>&
53 !catcode`!}=!active&
54 !expandafter!gdef!csname!detokenize<FV@Char@CatActive:>!FVCharNum!endcsname<}>&
55 !edef\FVCharNum<!number`!>&
56 !catcode`!}=12!relax&
57 !expandafter!gdef!csname!detokenize<FV@Char@CatTwelve:>!FVCharNum!endcsname< >&
58 !expandafter!xdef!csname!detokenize<FV@Char@CatDetok:>!FVCharNum!endcsname<!detokenize< >>&
59 !catcode`!}=!active&
60 !expandafter!gdef!csname!detokenize<FV@Char@CatActive:>!FVCharNum!endcsname< >&
61 !edef\FVCharNum<!number`!>&
62 !catcode`!}=12!relax&
63 !expandafter!gdef!csname!detokenize<FV@Char@CatTwelve:>!FVCharNum!endcsname<%>&
64 !expandafter!xdef!csname!detokenize<FV@Char@CatDetok:>!FVCharNum!endcsname<!detokenize<%>>&
65 !catcode`!}=!active&
66 !expandafter!gdef!csname!detokenize<FV@Char@CatActive:>!FVCharNum!endcsname<%>&
67 !endgroup
68 \begingroup
69 \edef\FVCharNum{\number`!}
70 \catcode`\! =12
71 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{!}
72 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{!}}
73 \catcode`\! =\active
74 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{!}
75 \endgroup
76 \begingroup
77 \edef\FVCharNum{\number`"}
78 \catcode`" =12
79 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{"}
80 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{"}}
81 \catcode`" =\active
82 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{"}
83 \endgroup
84 \begingroup
85 \edef\FVCharNum{\number`#}
86 \catcode`# =12
87 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{#}
88 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{#}}
89 \catcode`# =\active
90 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{#}
91 \endgroup
92 \begingroup
93 \edef\FVCharNum{\number`$}
94 \catcode`$ =12
95 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{$}
96 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{$}}

```



```

97 \catcode`\$=\active
98 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{&}
99 \endgroup
100 \begingroup
101 \edef\FVCharNum{\number`&}
102 \catcode`\&=12
103 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{&}
104 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{&}}
105 \catcode`\&=\active
106 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{&}
107 \endgroup
108 \begingroup
109 \edef\FVCharNum{\number`'}
110 \catcode`'=12
111 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{' }
112 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{' }}
113 \catcode`'=\active
114 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{' }
115 \endgroup
116 \begingroup
117 \edef\FVCharNum{\number`()}
118 \catcode`\(=12
119 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{()}
120 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{()}}
121 \catcode`\(=\active
122 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{()}
123 \endgroup
124 \begingroup
125 \edef\FVCharNum{\number`)}
126 \catcode`\)=12
127 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{)}
128 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{)}}
129 \catcode`\)=\active
130 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{)}
131 \endgroup
132 \begingroup
133 \edef\FVCharNum{\number`*}
134 \catcode`\*=12
135 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{*}
136 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{*}}
137 \catcode`\*=\active
138 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{*}
139 \endgroup
140 \begingroup
141 \edef\FVCharNum{\number`+}
142 \catcode`\+=12
143 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{+}
144 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{+}}
145 \catcode`\+=\active
146 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{+}
147 \endgroup
148 \begingroup
149 \edef\FVCharNum{\number`\,}
150 \catcode`\,=12

```

```

151 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{,}
152 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{,}}
153 \catcode`\,=\active
154 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{,}
155 \endgroup
156 \begingroup
157 \edef\FVCharNum{\number`\-}
158 \catcode`\-=12
159 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{-}
160 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{-}}
161 \catcode`\-=\active
162 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{-}
163 \endgroup
164 \begingroup
165 \edef\FVCharNum{\number`\.}
166 \catcode`\.=12
167 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{.}
168 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{.}}
169 \catcode`\.=\active
170 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{.}
171 \endgroup
172 \begingroup
173 \edef\FVCharNum{\number`\/}
174 \catcode`\/=12
175 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{/}
176 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{/}}
177 \catcode`\/= \active
178 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{/}
179 \endgroup
180 \begingroup
181 \edef\FVCharNum{\number`\:}
182 \catcode`\:=12
183 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{:}
184 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{:}}
185 \catcode`\:=\active
186 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{:}
187 \endgroup
188 \begingroup
189 \edef\FVCharNum{\number`\:;}
190 \catcode`\:=12
191 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{;}
192 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{;}}
193 \catcode`\:=\active
194 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{;}
195 \endgroup
196 \begingroup
197 \edef\FVCharNum{\number`\<}
198 \catcode`\<=12
199 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{<}
200 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{<}}
201 \catcode`\<=\active
202 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{<}
203 \endgroup
204 \begingroup

```

```

205 \edef\FVCharNum{\number` \=}
206 \catcode` \=12
207 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{=}
208 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{=}}
209 \catcode` \=\active
210 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{=}
211 \endgroup
212 \begingroup
213 \edef\FVCharNum{\number` \>}
214 \catcode` \>=12
215 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{>}
216 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{>}}
217 \catcode` \>=\active
218 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{>}
219 \endgroup
220 \begingroup
221 \edef\FVCharNum{\number` \?}
222 \catcode` \?=12
223 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{?}
224 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{?}}
225 \catcode` \?=\active
226 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{?}
227 \endgroup
228 \begingroup
229 \edef\FVCharNum{\number` \@}
230 \catcode` \@=12
231 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{@}
232 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{@}}
233 \catcode` \@=\active
234 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{@}
235 \endgroup
236 \begingroup
237 \edef\FVCharNum{\number` \[}
238 \catcode` \[=12
239 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{[}
240 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{[}}
241 \catcode` \[=\active
242 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{[}
243 \endgroup
244 \begingroup
245 \edef\FVCharNum{\number` \]}
246 \catcode` \]=12
247 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{]}
248 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{]}}
249 \catcode` \]=\active
250 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{]}
251 \endgroup
252 \begingroup
253 \edef\FVCharNum{\number` \^}
254 \catcode` \^=12
255 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{^}
256 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{^}}
257 \catcode` \^=\active
258 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{^}

```

```

259 \endgroup
260 \begingroup
261 \edef\FVCharNum{\number`\_}
262 \catcode`\_ =12
263 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{\_}
264 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{\_}}
265 \catcode`\_ =\active
266 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{\_}
267 \endgroup
268 \begingroup
269 \edef\FVCharNum{\number`\`}
270 \catcode`\` =12
271 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{`\`}
272 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{`\`}}
273 \catcode`\` =\active
274 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{`\`}
275 \endgroup
276 \begingroup
277 \edef\FVCharNum{\number`\|}
278 \catcode`\| =12
279 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{|\|}
280 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{|\|}}
281 \catcode`\| =\active
282 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{|\|}
283 \endgroup
284 \begingroup
285 \edef\FVCharNum{\number`\~}
286 \catcode`\~ =12
287 \expandafter\gdef\csname\detokenize{FV@Char@CatTwelve:}\FVCharNum\endcsname{~}
288 \expandafter\xdef\csname\detokenize{FV@Char@CatDetok:}\FVCharNum\endcsname{\detokenize{~}}
289 \catcode`\~ =\active
290 \expandafter\gdef\csname\detokenize{FV@Char@CatActive:}\FVCharNum\endcsname{~}
291 \endgroup

```

`\FVExtraSaveCodes`

`\FVExtraUseCodes`

`\FancyVerbRestoreCodes`

Save catcodes of all characters in `\FVExtraDoSpecials`, and then restore them. The `\FVExtra*` commands take arguments of the form `[(pkg)]{<name>}`.

`\FancyVerbRestoreCodes` is a shortcut for `\FVExtraUseCodes [fancyvrb]{BeforeVerbatim}`.

```

292 \newcommand{\FVExtraSaveCodes}[2][{}]{%
293   \ifcsname do\endcsname
294     \let\FV@dotmp\do
295   \fi
296   \if\relax\detokenize{#1}\relax
297     \def\do##1{%
298       \expandafter
299       \chardef\csname FV@SVCode:\number`##1@#2\endcsname=%
300       \catcode`##1\relax}%
301   \expandafter\let\csname FV@SVCodes@#2\endcsname\relax
302 \else
303   \def\do##1{%
304     \expandafter
305     \chardef\csname FV@PkgSVCode:\number`##1@#1:#2\endcsname=%

```

```

306         \catcode`##1\relax}%
307     \expandafter\let\csname FV@PkgSVCodes@#1:#2\endcsname\relax
308 \fi
309 \FVExtraDoSpecials
310 \ifcsname FV@dotmp\endcsname
311     \let\do\FV@dotmp
312     \let\FV@dotmp\FV@Undefined
313 \else
314     \let\do\FV@Undefined
315 \fi}
316 \newcommand{\FVExtraUseCodes}[2][ ]{%
317 \if\relax\detokenize{#1}\relax
318     \expandafter\FVExtraUseCodes@NoPkg
319 \else
320     \expandafter\FVExtraUseCodes@Pkg
321 \fi
322 {#1}{#2}}
323 \def\FVExtraUseCodes@NoPkg#1#2{%
324 \ifcsname FV@SVCodes@#2\endcsname
325     \expandafter\FVExtraUseCodes@NoPkg@i
326 \else
327     \PackageError{fvextra}%
328     {Unknown name "#2" for saved catcodes}%
329     {Check name of saved catcodes}%
330     \expandafter\@gobble
331 \fi
332 {#2}}
333 \def\FVExtraUseCodes@NoPkg@i#1{%
334 \FVExtraUseCodes@end{FV@SVCodes}{#1}}
335 \def\FVExtraUseCodes@Pkg#1#2{%
336 \ifcsname FV@PkgSVCodes@#1:#2\endcsname
337     \expandafter\FVExtraUseCodes@Pkg@i
338 \else
339     \PackageError{fvextra}%
340     {Unknown name "#2" for saved catcodes for package "#1"}%
341     {Check name of saved catcodes}%
342     \expandafter\@gobble
343 \fi
344 {#1:#2}}
345 \def\FVExtraUseCodes@Pkg@i#1{%
346 \FVExtraUseCodes@end{FV@PkgSVCodes}{#1}}
347 \def\FVExtraUseCodes@end#1#2{%
348 \ifcsname do\endcsname
349     \let\FV@dotmp\do
350 \fi
351 \def\do##1{%
352     \catcode`##1=%
353     \csname #1:\number`##1@#2\endcsname\relax}%
354 \FVExtraDoSpecials
355 \ifcsname FV@dotmp\endcsname
356     \let\do\FV@dotmp
357     \let\FV@dotmp\FV@Undefined
358 \else
359     \let\do\FV@Undefined

```

```

360 \fi}
361 \AtEndOfPackage{%
362 \FV@AddToHook{\FV@UseKeyValues@Hook}{\FVExtraSaveCodes[fancyvrb]{BeforeVerbatim}}
363 \def\FancyVerbRestoreCodes{\FVExtraUseCodes[fancyvrb]{BeforeVerbatim}}

```

### 12.2.3 Sentinels

Sentinel macros are needed for scanning tokens.

There are two contexts in which sentinels may be needed. In delimited macro arguments, such as `\def\macro#1\sentinel{...}`, a sentinel is needed as the delimiter. Because the delimiting macro need not be defined, special delimiting macros need not be created for this case. The important thing is to ensure that the macro name is sufficiently unique to avoid collisions. Typically, using `\makeatletter` to allow something like `\@sentinel` will be sufficient. For added security, additional characters can be given catcode 11, to allow things like `\@sent!nel`.

The other context for sentinels is in scanning through a sequence of tokens that is delimited by a sentinel, and using `\ifx` comparisons to identify the sentinel and stop scanning. In this case, using an undefined macro is risky. Under normal conditions, the sequence of tokens could contain an undefined macro due to mistyping. In some `fvextra` applications, the tokens will have been incorrectly tokenized under a normal catcode regime, and need to be retokenized as verbatim, in which case undefined macros must be expected. Thus, a sentinel macro whose expansion is resistant to collisions is needed.

`\FV@<Sentinel>`

This is the standard default `fvextra` delimited-macro sentinel. It is used with `\makeatletter` by changing `<` and `>` to catcode 11. The `<` and `>` add an extra level of collision resistance. Because it is undefined, it is *only* appropriate for use in delimited macro arguments.

`\FV@Sentinel`

This is the standard `fvextra` `\ifx` comparison sentinel. It expands to the control word `\FV@<Sentinel>`, which is very unlikely to be in any other macro since it requires that `@`, `<`, and `>` all have catcode 11 and appear in the correct sequence. Because its definition is itself undefined, this sentinel will result in an error if it escapes.

```

364 \begingroup
365 \catcode`\<=11
366 \catcode`\>=11
367 \gdef\FV@Sentinel{\FV@<Sentinel>}
368 \endgroup

```

### 12.2.4 Active character definitions

`\FV@OuterDefEOLEmpty`

Macro for defining the active end-of-line character  $\text{\r}$ , which `fancyvrb` uses to prevent runaway command arguments. `fancyvrb` uses macro definitions of the form

```

\begingroup
\catcode`\^M=\active%
\gdef\macro{%
...

```

```

\outer\def^^M{%
...
}%
\endgroup

```

While this works, it is nice to avoid the `\begingroup... \endgroup` and especially the requirement that all lines now end with `%` to discard the `^^M` that would otherwise be inserted.

```

369 \begingroup
370 \catcode^^M=\active%
371 \gdef\FV@OuterDefEOLEmpty{\outer\def^^M{}}%
372 \endgroup

```

#### `\FV@DefEOLEmpty`

The same thing, without the `\outer`. This is used to ensure that `^^M` is not `\outer` when it should be read.

```

373 \begingroup
374 \catcode^^M=\active%
375 \gdef\FV@DefEOLEmpty{\def^^M{}}%
376 \endgroup

```

#### `\FV@OuterDefSTXEmpty`

Define start-of-text (STX) `^^B` so that it cannot be used inside other macros. This makes it possible to guarantee that `^^B` is not part of a verbatim argument, so that it can be used later as a sentinel in retokenizing the argument.

```

377 \begingroup
378 \catcode^^B=\active
379 \gdef\FV@OuterDefSTXEmpty{\outer\def^^B{}}
380 \endgroup

```

#### `\FV@OuterDefETXEmpty`

Define end-of-text (ETX) `^^C` so that it cannot be used inside other macros. This makes it possible to guarantee that `^^C` is not part of a verbatim argument, so that it can be used later as a sentinel in retokenizing the argument.

```

381 \begingroup
382 \catcode^^C=\active
383 \gdef\FV@OuterDefETXEmpty{\outer\def^^C{}}
384 \endgroup

```

## 12.3 pdfTeX with inputenc using UTF-8

Working with verbatim text often involves handling individual code points. While these are treated as single entities under LuaTeX and XeTeX, under pdfTeX code points must be handled at the byte level instead. This means that reading a single code point encoded in UTF-8 may involve a macro that reads up to four arguments.

Macros are defined for working with non-ASCII code points under pdfTeX. These are only for use with the `inputenc` package set to `utf8` encoding.

#### `\ifFV@pdfTeXinputenc`

All of the UTF macros are only needed with pdfTeX when `inputenc` is loaded, so they are created conditionally, inspired by the approach of the `iftex` package. The tests deal with the possibility that a previous test using `\ifx` rather than the cleaner `\ifcsname` has already been performed. These assume that `inputenc` will be loaded before `fvextra`. The `\inputencodingname` tests should be redundant after

the `\@ifpackageloaded` test, but do provide some additional safety if another package is faking `inputenc` being loaded but not providing an equivalent encoding interface.

Note that an encoding test of the form

```
\ifdefstring{\inputencodingname}{utf8}{<true>}{<false>}
```

is still required before switching to the UTF variants in any given situation. A document using `inputenc` can switch encodings (for example, around an `\input`), so simply checking encoding when `fvextra` is loaded is *not* sufficient.

```
385 \newif\ifFV@pdfTeXinputenc
386 \FV@pdfTeXinputencfalse
387 \ifcsname pdfmatch\endcsname
388 \ifx\pdfmatch\relax
389 \else
390 \@ifpackageloaded{inputenc}%
391 {\ifcsname inputencodingname\endcsname
392 \ifx\inputencodingname\relax
393 \else
394 \FV@pdfTeXinputenctrue
395 \fi\fi}
396 {}%
397 \fi\fi
```

Define UTF macros conditionally:

```
398 \ifFV@pdfTeXinputenc
```

`\FV@U8:<byte>`

Define macros of the form `\FV@U8:<byte>` for each active byte. These are used for determining whether a 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`.

```
399 \begingroup
400 \catcode`\~ =13
401 \catcode`\ " =12
402 \def\FV@UTFviii@loop{%
403 \uccode`\~\count@
404 \uppercase\expandafter{\FV@UTFviii@Tmp}%
405 \advance\count@\@ne
406 \ifnum\count@<\@tempcnta
407 \expandafter\FV@UTFviii@loop
408 \fi}
```

Setting up 2-byte UTF-8:

```
409 \count@"C2
410 \@tempcnta"E0
411 \def\FV@UTFviii@Tmp{\expandafter\gdef\csname FV@U8:\string~\endcsname{%
412 \FV@UTF@two@octets}}
413 \FV@UTFviii@loop
```

Setting up 3-byte UTF-8:

```
414 \count@"E0
415 \@tempcnta"F0
416 \def\FV@UTFviii@Tmp{\expandafter\gdef\csname FV@U8:\string~\endcsname{%
```



```

417 \FV@UTF@three@octets}}
418 \FV@UTFviii@loop
Setting up 4-byte UTF-8:
419 \count@"F0
420 \@tempcnta"F4
421 \def\FV@UTFviii@Tmp{\expandafter\gdef\csname FV@U8:\string~\endcsname{%
422 \FV@UTF@four@octets}}
423 \FV@UTFviii@loop
424 \endgroup

```

```

\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 to `\FV@UTF@octets@after` as a single argument for further processing. The invoking macro should `\let` or `\def`'ed `\FV@UTF@octets@after` to an appropriate macro that performs further processing.

Typical use will involve the following steps:

1. Read a token, say `#1`.
2. Use `\ifcsname FV@U8:\detokenize{#1}\endcsname` to determine that the token is the first byte of a multi-byte code point.
3. Ensure that `\FV@UTF@octets@after` has an appropriate value, if this has not already been done.
4. Use `\csname FV@U8:\detokenize{#1}\endcsname#1` at the end of the original reading macro to read the full multi-byte code point and then pass it on as a single argument to `\FV@UTF@octets@after`.

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 a sentinel macro in a scanning context, 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.

```

425 \def\FV@UTF@two@octets#1#2{%
426 \ifcsname u8:\detokenize{#1#2}\endcsname
427 \else
428 #1#2%
429 \fi
430 \FV@UTF@octets@after{#1#2}}
431 \def\FV@UTF@three@octets#1#2#3{%
432 \ifcsname u8:\detokenize{#1#2#3}\endcsname
433 \else
434 #1#2#3%
435 \fi
436 \FV@UTF@octets@after{#1#2#3}}
437 \def\FV@UTF@four@octets#1#2#3#4{%
438 \ifcsname u8:\detokenize{#1#2#3#4}\endcsname
439 \else
440 #1#2#3#4%
441 \fi

```

```

442 \FV@UTF@octets@after{#1#2#3#4}}
      End conditional creation of UTF macros:
443 \fi

```

## 12.4 Reading and processing command arguments

`fvextra` provides macros for reading and processing verbatim arguments. These are primarily intended for creating commands that take verbatim arguments but can still be used within other commands (with some limitations). These macros are used in reimplementing `fancyvrb` commands like `\Verb`. They may also be used in other packages; `minted` and `pythontex` use them for handling inline code.

All macros meant for internal use have names of the form `\FV@<Name>`, while all macros meant for use in other packages have names of the form `\FVExtra<Name>`. Only the latter are intended to have a stable interface.

### 12.4.1 Tokenization and lookahead

#### `\FVExtra@ifnextcharAny`

A version of `\@ifnextchar` that can detect any character, including catcode 10 spaces. This is an exact copy of the definition from `latex.ltx`, modified with the “`\let\reserved@d= #1%`” (note space!) trick from `amsgen`.

```

444 \long\def\FVExtra@ifnextcharAny#1#2#3{%
445   \let\reserved@d= #1%
446   \def\reserved@a{#2}%
447   \def\reserved@b{#3}%
448   \futurelet\@let@token\FVExtra@ifnchAny}
449 \def\FVExtra@ifnchAny{%
450   \ifx\@let@token\reserved@d
451     \expandafter\reserved@a
452   \else
453     \expandafter\reserved@b
454   \fi}

```

#### `\FVExtra@ifnextcharVArg`

This is a wrapper for `\@ifnextchar` from `latex.ltx` (`ltxdefns.dtx`) that tokenizes lookaheads under a mostly verbatim catcode regime rather than the current catcode regime. This is important when looking ahead for stars `*` and optional argument delimiters `[`, because if these are not present when looking ahead for a verbatim argument, then the first thing tokenized will be the verbatim argument’s delimiting character. Ideally, the delimiter should be tokenized under a verbatim catcode regime. This is necessary for instance if the delimiter is `\active` and `\outer`.

The catcode of the space is preserved (in the unlikely event it is `\active`) and curly braces are given their normal catcodes for the lookahead. This simplifies space handling in an untokenized context, and allows paired curly braces to be used as verbatim delimiters.

```

455 \long\def\FVExtra@ifnextcharVArg#1#2#3{%
456   \begingroup
457   \edef\FV@TmpSpaceCat{\the\catcode` }%
458   \let\do\@makeother\FVExtraDoSpecials
459   \catcode`\ =\FV@TmpSpaceCat\relax

```

```

460 \catcode`\{=1
461 \catcode`\}=2
462 \@ifnextchar#1{\endgroup#2}{\endgroup#3}}

```

### `\FVExtra@ifstarVArg`

A starred command behaves differently depending on whether it is followed by an optional star or asterisk `*`. `\@ifstar` from `latex.ltx` is typically used to check for the `*`. In the process, it discards following spaces (catcode 10) and tokenizes the next non-space character under the current catcode regime. While this is fine for normal commands, it is undesirable if the next character turns out to be not a `*` but rather a verbatim argument's delimiter. This reimplementaion prevents such issues for all printable ASCII symbols via `\FVExtra@ifnextcharVArg`.

```

463 \begingroup
464 \catcode`\*=12
465 \gdef\FVExtra@ifstarVArg#1{\FVExtra@ifnextcharVArg*{\@firstoftwo{#1}}}
466 \endgroup

```

## 12.4.2 Reading arguments

### `\FV@Read0ArgContinue`

Read a macro followed by an optional argument, then pass the optional argument to the macro for processing and to continue.

```

467 \def\FV@Read0ArgContinue#1[#2]{#1{#2}}

```

### `\FVExtraRead0ArgBeforeVArg`

Read an optional argument that comes before a verbatim argument. The lookahead for the optional argument tokenizes with a verbatim catcode regime in case it encounters the delimiter for the verbatim argument rather than `[`. If the lookahead doesn't find `[`, the optional argument for `\FVExtraRead0ArgBeforeVArg` can be used to supply a default optional argument other than *empty*.

```

468 \newcommand{\FVExtraRead0ArgBeforeVArg}[2][ ]{%
469   \FVExtra@ifnextcharVArg[%
470     {\FV@Read0ArgContinue{#2}}}%
471   {\FV@Read0ArgContinue{#2}{#1}}

```

### `\FVExtraRead0ArgBeforeVEnv`

Read an optional argument at the start of a verbatim environment, after the `\begin{environment}` but before the start of the next line where the verbatim content begins. Check for extraneous content after the optional argument and discard the following newline. Note that this is not needed when an environment takes a mandatory argument that follows the optional argument.

The case with only an optional argument is tricky because the default behavior of `\@ifnextchar` is to read into the next line looking for the optional argument. Setting `^^M` as `\active` prevents this. That does mean, though, that the end-of-line token will have to be read and removed later as an `\active ^^M`.

`\@ifnextchar` is used instead of `\FVExtra@ifnextcharVArg` because the latter is not needed since there is an explicit, required delimiter (`^^M`) before the actual start of verbatim content. Lookahead can never tokenize verbatim content under an incorrect catcode regime.

```

472 \newcommand{\FVExtraRead0ArgBeforeVEnv}[2][ ]{%
473   \begingroup
474   \catcode`^^M=\active
475   \@ifnextchar[%

```

```

476   {\endgroup\FVExtraRead0ArgBeforeVEnv@i{#2}}%
477   {\endgroup\FVExtraRead0ArgBeforeVEnv@i{#2}[#1]}}
478 \def\FVExtraRead0ArgBeforeVEnv@i#1[#2]{%
479   \begingroup
480   \catcode`\^^M=\active
481   \FVExtraRead0ArgBeforeVEnv@ii{#1}{#2}}
482 \begingroup
483 \catcode`\^^M=\active%
484 \gdef\FVExtraRead0ArgBeforeVEnv@ii#1#2#3^^M{%
485   \endgroup%
486   \FVExtraRead0ArgBeforeVEnv@iii{#1}{#2}{#3}}%
487 \endgroup%
488 \def\FVExtraRead0ArgBeforeVEnv@iii#1#2#3{%
489   \if\relax\detokenize{#3}\relax
490   \else
491     \PackageError{fvextra}%
492     {Discarded invalid text while checking for optional argument of verbatim environment}%
493     {Discarded invalid text while checking for optional argument of verbatim environment}%
494   \fi
495   #1{#2}}

```

`\FVExtraReadVArg`

`\FVExtraReadVArgSingleLine`

Read a verbatim argument that is bounded by two identical characters or by paired curly braces. There are two variants: one reads a multi-line (but not multi-paragraph) argument, while the other restricts the argument to a single line via the `\outer ^^M` trick from `fancyvrb`. An `\outer ^^C` is used to prevent `^^C` from being part of arguments, so that it can be used later as a sentinel if retokenization is needed. `^^B` is handled in the same manner for symmetry with later usage, though technically it is not used as a sentinel so this is not strictly necessary. Alternate UTF macros, defined later, are invoked when under pdfTeX with `inputenc` using UTF-8.

The lookahead for the type of delimiting character is done under a verbatim catcode regime, except that the space catcode is preserved and curly braces are given their normal catcodes. This provides consistency with any `\FVExtra@ifnextcharVArg` or `\FVExtra@ifstarVArg` that may have been used previously, allows characters like `#` and `%` to be used as delimiters when the verbatim argument is read outside any other commands (untokenized), and allows paired curly braces to serve as delimiters. Any additional command-specific catcode modifications should only be applied to the argument after it has been read, since they do not apply to the delimiters.

Once the delimiter lookahead is complete, catcodes revert to full verbatim, and are then modified appropriately given the type of delimiter. The space and tab must be `\active` to be preserved correctly when the verbatim argument is not inside any other commands (otherwise, they collapse into single spaces).

Note that `\FVExtraReadVArg` will interpret a line break as a catcode 10 space, not as an `\active` space. Depending on usage, the argument may need to be processed with `\FVExtraDetokenizeVArg` and `\FVExtraRetokenizeVArg` to fix this.

```

496 \def\FVExtraReadVArg#1{%
497   \begingroup
498   \ifFV@pdfTeXinputenc

```

```

499     \ifdefstring{\inputencodingname}{utf8}%
500     {\let\FV@ReadVArg@Char\FV@ReadVArg@Char@UTF}%
501     {}}%
502 \fi
503 \edef\FV@TmpSpaceCat{\the\catcode`}%
504 \let\do\@makeother\FVExtraDoSpecials
505 \catcode`\^B=\active
506 \FV@OuterDefSTXEmpty
507 \catcode`\^C=\active
508 \FV@OuterDefETXEmpty
509 \begingroup
510 \catcode`\ =\FV@TmpSpaceCat\relax
511 \catcode`\{=1
512 \catcode`\}=2
513 \@ifnextchar\bgroup
514 {\endgroup
515   \catcode`\{=1
516   \catcode`\}=2
517   \catcode`\ =\active
518   \catcode`\^I=\active
519   \FV@ReadVArg@Group{#1}}%
520 {\endgroup
521   \catcode`\ =\active
522   \catcode`\^I=\active
523   \FV@ReadVArg@Char{#1}}
524 \def\FVExtraReadVArgSingleLine#1{%
525   \begingroup
526   \ifFV@pdfTeXinputenc
527     \ifdefstring{\inputencodingname}{utf8}%
528     {\let\FV@ReadVArg@Char\FV@ReadVArg@Char@UTF}%
529     {}}%
530 \fi
531 \edef\FV@TmpSpaceCat{\the\catcode`}%
532 \let\do\@makeother\FVExtraDoSpecials
533 \catcode`\^B=\active
534 \FV@OuterDefSTXEmpty
535 \catcode`\^C=\active
536 \FV@OuterDefETXEmpty
537 \catcode`\^M=\active
538 \FV@OuterDefEOLEmpty
539 \begingroup
540 \catcode`\ =\FV@TmpSpaceCat\relax
541 \catcode`\{=1
542 \catcode`\}=2
543 \@ifnextchar\bgroup
544 {\endgroup
545   \catcode`\{=1
546   \catcode`\}=2
547   \catcode`\ =\active
548   \catcode`\^I=\active
549   \FV@ReadVArg@Group{#1}}%
550 {\endgroup
551   \catcode`\ =\active
552   \catcode`\^I=\active

```

```
553 \FV@ReadVArg@Char{#1}}}
```

### `\FV@ReadVArg@Group`

The argument is read under the verbatim catcode regime already in place from `\FVExtraReadVArg`. The `\endgroup` returns to prior catcodes. Any command-specific catcodes can be applied later via `\scantokens`. Using them here in reading the argument would have no effect as far as later processing with `\scantokens` is concerned, unless the argument were read outside any other commands and additional characters were given catcodes 1 or 2 (like the curly braces). That scenario is not allowed because it makes reading the argument overly dependent on the argument content. (Technically, reading the argument is already dependent on the argument content in the sense that the argument cannot contain unescaped unpaired curly braces, given that it is delimited by curly braces.)

```
554 \def\FV@ReadVArg@Group#1#2{%
555 \endgroup
556 #1{#2}}
```

### `\FV@ReadVArg@Char`

The delimiting character is read under the verbatim catcode regime in place from `\FVExtraReadVArg`. If the command is not inside a normal command, then this means the delimiting character will typically have catcode 12 and that characters like `#` and `%` can be used as delimiters; otherwise, the delimiter may have any catcode that is possible for a single character captured by a macro. If the argument is read inside another command (already tokenized), then it is possible for the delimiter to be a control sequence rather than a single character. An error is raised in this case. The `\endgroup` in `\FV@ReadVArg@Char@i` returns to prior catcodes after the argument is captured.

It would be possible to read the argument using any command-specific catcode settings, but that would result in different behavior depending on whether the argument is already tokenized, and would make reading the argument overly dependent on the argument content.

```
557 \def\FV@ReadVArg@Char#1#2{%
558 \expandafter\expandafter\expandafter
559 \if\expandafter\expandafter\expandafter\relax\expandafter\@gobble\detokenize{#2}\relax
560 \expandafter\@gobble
561 \else
562 \expandafter\@firstofone
563 \fi
564 {\PackageError{fvextra}%
565 {Verbatim delimiters must be single characters, not commands}%
566 {Try a different delimiter}}%
567 \def\FV@ReadVArg@Char@i##1##2#2{%
568 \endgroup
569 ##1{##2}}%
570 \FV@ReadVArg@Char@i{#1}}%
```

## Alternate implementation for pdfTeX with inputenc using UTF-8

Start conditional creation of macros:

```
571 \ifFV@pdfTeXinputenc
\FV@ReadVArg@Char@UTF
```

This is a variant of `\FV@ReadVArg@Char` that allows non-ASCII codepoints as delimiters under the pdfTeX engine with `inputenc` using UTF-8. Under pdfTeX, non-ASCII codepoints must be handled as a sequence of bytes rather than as a single entity. `\FV@ReadVArg@Char` is automatically `\let` to this version when appropriate. This uses the `\FV@U8:<byte>` macros for working with `inputenc`'s UTF-8.

```

572 \def\FV@ReadVArg@Char@UTF#1#2{%
573   \expandafter\expandafter\expandafter
574   \if\expandafter\expandafter\expandafter\relax\expandafter\@gobble\detokenize{#2}\relax
575     \expandafter\@gobble
576   \else
577     \expandafter\@firstofone
578   \fi
579   {\PackageError{fvextra}%
580    {Verbatim delimiters must be single characters, not commands}%
581    {Try a different delimiter}}%
582   \ifcsname FV@U8:\detokenize{#2}\endcsname
583     \expandafter\@firstoftwo
584   \else
585     \expandafter\@secondoftwo
586   \fi
587   {\def\FV@UTF@octets@after##1{\FV@ReadVArg@Char@UTF@i{#1}{##1}}%
588    \csname FV@U8:\detokenize{#2}\endcsname#2}%
589   {\FV@ReadVArg@Char@UTF@i{#1}{#2}}}

```

`\FV@ReadVArg@Char@UTF@i`

```

590 \def\FV@ReadVArg@Char@UTF@i#1#2{%
591   \def\FV@ReadVArg@Char@i##1##2#2{%
592     \endgroup
593     ##1{##2}}%
594   \FV@ReadVArg@Char@i{#1}}%

```

End conditional creation of UTF macros:

```
595 \fi
```

`vargsingleline`

This determines whether `\Verb` and `\SaveVerb` use `\FVExtraReadVArg` or `\FVExtraReadVArgSingleLine` to read their arguments. It has no effect on `\EscVerb`, since that does not use special tokenization.

```

596 \newbool{FV@vargsingleline}
597 \define@booleankey{FV}{vargsingleline}%
598 {\booltrue{FV@vargsingleline}}
599 {\boolfalse{FV@vargsingleline}}
600 \fvset{vargsingleline=false}

```

### 12.4.3 Reading and protecting arguments in expansion-only contexts

The objective here is to make possible commands that can function correctly after being in expansion-only contexts like `\edef`. The general strategy is to allow commands to be defined like this:

```
\def\cmd{\FVExtraRobustCommand\robustcmd\reader}
```

`\robustcmd` is the actual command, including argument reading and processing, and is `\protected`. `\reader` is an expandable macro that reads all of `\robustcmd`'s arguments, then wraps them in `\FVExtraAlwaysUnexpanded`. When `\FVExtraAlwaysUnexpanded{<args>}` is expanded, the result is always `\FVExtraAlwaysUnexpanded{<args>}`. `\FVExtraRobustCommand` is `\protected` and manages everything in a context-sensitive manner.

- In a normal context, `\FVExtraRobustCommand` reads two arguments, which will be `\robustcmd` and `\reader`. It detects that `\reader` has not expanded to `\FVExtraAlwaysUnexpanded{<args>}`, so it discards `\reader` and reinserts `\robustcmd` so that it can operate normally.
- In an expansion-only context, neither `\FVExtraRobustCommand` nor `\robustcmd` will expand, because both are `\protected`. `\reader` will read `\robustcmd`'s arguments and protect them with `\FVExtraAlwaysUnexpanded`. When this is used later in a normal context, `\FVExtraRobustCommand` reads two arguments, which will be `\robustcmd` and `\FVExtraAlwaysUnexpanded`. It detects that `\reader` did expand, so it discards `\FVExtraAlwaysUnexpanded` and reads its argument to discard the wrapping braces. Then it reinserts `\robustcmd<args>` so that everything can proceed as if expansion had not occurred.

#### `\FVExtrapdfstringdef`

#### `\FVExtrapdfstringdefDisableCommands`

Conditionally allow alternate definitions for PDF bookmarks when `hyperref` is in use. This is helpful for working with `\protected` or otherwise unexpandable commands.

```

601 \def\FVExtrapdfstringdef#1#2{%
602   \AfterPreamble{%
603     \ifcsname pdfstringdef\endcsname
604     \ifx\pdfstringdef\relax
605     \else
606     \pdfstringdef#1{#2}%
607     \fi\fi}}
608 \def\FVExtrapdfstringdefDisableCommands#1{%
609   \AfterPreamble{%
610     \ifcsname pdfstringdefDisableCommands\endcsname
611     \ifx\pdfstringdefDisableCommands\relax
612     \else
613     \pdfstringdefDisableCommands{#1}%
614     \fi\fi}}

```

#### `\FVExtraAlwaysUnexpanded`

Always expands to itself, thanks to `\unexpanded`.

```

615 \long\def\FVExtraAlwaysUnexpanded#1{%
616   \unexpanded{\FVExtraAlwaysUnexpanded{#1}}}
617 \FVExtrapdfstringdefDisableCommands{%
618   \long\def\FVExtraAlwaysUnexpanded#1{#1}}

```

#### `FVExtraRobustCommandExpanded`

Boolean to track whether expansion occurred. Set in `\FVExtraRobustCommand`. Useful in creating commands that behave differently depending on whether expansion occurred.

```

619 \newbool{FVExtraRobustCommandExpanded}

```



### `\FVExtraRobustCommand`

```
620 \protected\def\FVExtraRobustCommand#1#2{%
621   \ifx#2\FVExtraAlwaysUnexpanded
622     \expandafter\@firstoftwo
623   \else
624     \expandafter\@secondoftwo
625   \fi
626   {\booltrue{FVExtraRobustCommandExpanded}\FV@RobustCommand@i{#1}}%
627   {\boolfalse{FVExtraRobustCommandExpanded}#1}}
628 \FVextrapdfstringdefDisableCommands{%
629   \def\FVExtraRobustCommand{}}
```

### `\FV@RobustCommand@i`

#2 will be the argument of `\FVExtraAlwaysUnexpanded`. Reading this strips the braces. At the beginning of #2 will be the reader macro, which must be `\@gobble'd`.

```
630 \def\FV@RobustCommand@i#1#2{\expandafter#1\@gobble#2}
```

### `\FVExtraUnexpandedReadStar0ArgMArg`

Read the arguments for a command that may be starred, may have an optional argument, and has a single brace-delimited mandatory argument. Then protect them with `\FVExtraAlwaysUnexpanded`. The reader macro is itself maintained in the protected result, so that it can be redefined to provide a simple default value for `hyperref`.

Note the argument signature `#1#{`. This reads everything up to, but not including, the next brace group.

```
631 \def\FVExtraUnexpandedReadStar0ArgMArg#1#{%
632   \FV@UnexpandedReadStar0ArgMArg@i{#1}}
```

### `\FV@UnexpandedReadStar0ArgMArg@i`

```
633 \def\FV@UnexpandedReadStar0ArgMArg@i#1#2{%
634   \FVExtraAlwaysUnexpanded{\FVExtraUnexpandedReadStar0ArgMArg#1{#2}}
635 \FVextrapdfstringdefDisableCommands{%
636   \makeatletter
637   \def\FV@UnexpandedReadStar0ArgMArg@i#1#2{#2}%
638   \makeatother}
```

### `\FVExtraUseVerbUnexpandedReadStar0ArgMArg`

This is a variant of `\FVExtraUnexpandedReadStar0ArgMArg` customized for `\UseVerb`. It would be tempting to use `\pdfstringdef` to define a PDF string based on the final tokenization in `\UseVerb`, rather than applying `\FVExtraPDFStringVerbatimDetokenize` to the original raw (read) tokenization. Unfortunately, `\pdfstringdef` apparently can't handle catcode 12 `\` and `%`. Since the final tokenization could contain arbitrary catcodes, that approach might fail even if the `\` and `%` issue were resolved. It may be worth considering more sophisticated approaches in the future.

```
639 \def\FVExtraUseVerbUnexpandedReadStar0ArgMArg#1#{%
640   \FV@UseVerbUnexpandedReadStar0ArgMArg@i{#1}}
```

### `\FV@UseVerbUnexpandedReadStar0ArgMArg@i`

```
641 \def\FV@UseVerbUnexpandedReadStar0ArgMArg@i#1#2{%
642   \FVExtraAlwaysUnexpanded{\FVExtraUseVerbUnexpandedReadStar0ArgMArg#1{#2}}
643 \FVextrapdfstringdefDisableCommands{%
```

```

644 \makeatletter
645 \def\FV@UseVerbUnexpandedReadStar0ArgMArg@i#1#2{%
646   \ifcsname FV@SVRaw@#2\endcsname
647     \expandafter\expandafter\expandafter\FVExtraPDFStringVerbatimDetokenize
648     \expandafter\expandafter\expandafter{\csname FV@SVRaw@#2\endcsname}%
649   \fi}%
650 \makeatother}

```

#### \FVExtraUnexpandedReadStar0ArgBVarG

Same as \FVExtraUnexpandedReadStar0ArgMArg, except BVarG, brace-delimited verbatim argument.

```

651 \def\FVExtraUnexpandedReadStar0ArgBVarG#1#2{%
652   \FV@UnexpandedReadStar0ArgBVarG@i{#1}}

```

#### \FV@UnexpandedReadStar0ArgBVarG@i

```

653 \def\FV@UnexpandedReadStar0ArgBVarG@i#1#2{%
654   \FVExtraAlwaysUnexpanded{\FVExtraUnexpandedReadStar0ArgBVarG#1{#2}}
655 \FVExtrapdfstringdefDisableCommands{%
656   \makeatletter
657   \def\FV@UnexpandedReadStar0ArgBVarG@i#1#2{%
658     \FVExtraPDFStringVerbatimDetokenize{#2}}%
659   \makeatother}

```

#### \FVExtraUnexpandedReadStar0ArgBEscVarG

Same as \FVExtraUnexpandedReadStar0ArgMArg, except BEscVarG, brace-delimited escaped verbatim argument.

```

660 \def\FVExtraUnexpandedReadStar0ArgBEscVarG#1#2{%
661   \FV@UnexpandedReadStar0ArgBEscVarG@i{#1}}

```

#### \FV@UnexpandedReadStar0ArgBEscVarG@i

```

662 \def\FV@UnexpandedReadStar0ArgBEscVarG@i#1#2{%
663   \FVExtraAlwaysUnexpanded{\FVExtraUnexpandedReadStar0ArgBEscVarG#1{#2}}
664 \FVExtrapdfstringdefDisableCommands{%
665   \makeatletter
666   \def\FV@UnexpandedReadStar0ArgBEscVarG@i#1#2{%
667     \FVExtraPDFStringEscapedVerbatimDetokenize{#2}}%
668   \makeatother}

```

#### \FVExtraUnexpandedReadStar0ArgMArgBVarG

Read arguments for a command that has a mandatory argument before a verbatim argument, such as minted's \mintinline.

```

669 \def\FVExtraUnexpandedReadStar0ArgMArgBVarG#1#2{%
670   \FV@UnexpandedReadStar0ArgMArgBVarG@i{#1}}
671 \def\FV@UnexpandedReadStar0ArgMArgBVarG@i#1#2{%
672   \FV@UnexpandedReadStar0ArgMArgBVarG@ii{#1}{#2}}
673 \def\FV@UnexpandedReadStar0ArgMArgBVarG@ii#1#2#3{%
674   \FV@UnexpandedReadStar0ArgMArgBVarG@iii{#1}{#2}{#3}}
675 \def\FV@UnexpandedReadStar0ArgMArgBVarG@iii#1#2#3#4{%
676   \FVExtraAlwaysUnexpanded{\FVExtraUnexpandedReadStar0ArgMArgBVarG#1{#2}#3{#4}}
677 \FVExtrapdfstringdefDisableCommands{%
678   \makeatletter
679   \def\FV@UnexpandedReadStar0ArgMArgBVarG@iii#1#2#3#4{%
680     \FVExtraPDFStringVerbatimDetokenize{#4}}%
681   \makeatother}

```

#### 12.4.4 Converting detokenized tokens into PDF strings

At times it will be convenient to convert detokenized tokens into PDF strings, such as bookmarks. Define macros to escape such detokenized content so that it is in a suitable form.

##### `\FVExtraPDFStringEscapeChar`

Note that this does not apply any special treatment to spaces. If there are multiple adjacent spaces, then the octal escape `\040` is needed to prevent them from being merged. In the detokenization macros where `\FVExtraPDFStringEscapeChar` is currently used, spaces are processed separately without `\FVExtraPDFStringEscapeChar`, and literal spaces or `\040` are inserted in a context-dependent manner.

```
682 \def\FVExtraPDFStringEscapeChar#1{%
683   \ifcsname FV@PDFStringEscapeChar@#1\endcsname
684     \csname FV@PDFStringEscapeChar@#1\endcsname
685   \else
686     #1%
687   \fi}
688 \begingroup
689 \catcode`\&=14
690 \catcode`\%=12&
691 \catcode`\(=12&
692 \catcode`\)=12&
693 \catcode`\^^J=12&
694 \catcode`\^^M=12&
695 \catcode`\^^I=12&
696 \catcode`\^^H=12&
697 \catcode`\^^L=12&
698 \catcode`\!=0\relax&
699 !catcode`\!=12!relax&
700 !expandafter!gdef!csname FV@PDFStringEscapeChar@!\!endcsname{\}&
701 !expandafter!gdef!csname FV@PDFStringEscapeChar@%!\endcsname{\}%&
702 !expandafter!gdef!csname FV@PDFStringEscapeChar@(!endcsname{\}&
703 !expandafter!gdef!csname FV@PDFStringEscapeChar@)!\endcsname{\})&
704 !expandafter!gdef!csname FV@PDFStringEscapeChar@^^J!\endcsname{\n}&
705 !expandafter!gdef!csname FV@PDFStringEscapeChar@^^M!\endcsname{\r}&
706 !expandafter!gdef!csname FV@PDFStringEscapeChar@^^I!\endcsname{\t}&
707 !expandafter!gdef!csname FV@PDFStringEscapeChar@^^H!\endcsname{\b}&
708 !expandafter!gdef!csname FV@PDFStringEscapeChar@^^L!\endcsname{\f}&
709 !catcode`\!=0!relax&
710 \endgroup
```

##### `\FVExtraPDFStringEscapeChars`

```
711 \def\FVExtraPDFStringEscapeChars#1{%
712   \FV@PDFStringEscapeChars#1\FV@Sentinel}
```

##### `\FV@PDFStringEscapeChars`

```
713 \def\FV@PDFStringEscapeChars#1{%
714   \ifx#1\FV@Sentinel
715   \else
716     \FVExtraPDFStringEscapeChar{#1}%
717     \expandafter\FV@PDFStringEscapeChars
718   \fi}%
```

### 12.4.5 Detokenizing verbatim arguments

Ensure correct catcodes for this subsection (note < and > for \FV@<Sentinel>):

```
719 \begingroup
720 \catcode`\ =10
721 \catcode`\a=11
722 \catcode`\<=11
723 \catcode`\>=11
724 \catcode`\^C=\active
```

#### Detokenize as if the original source were tokenized verbatim

##### \FVExtraVerbatimDetokenize

Detokenize tokens as if their original source was tokenized verbatim, rather than under any other catcode regime that may actually have been in place. This recovers the original source when tokenization was verbatim. Otherwise, it recovers the closest approximation of the source that is possible given information loss during tokenization (for example, adjacent space characters may be merged into a single space token). This is useful in constructing nearly verbatim commands that can be used inside other commands. It functions in an expansion-only context (“fully expandable,” works in \edef).

This yields spaces with catcode 12, *not* spaces with catcode 10 like \detokenize. Spaces with catcode 10 require special handling when being read by macros, so detokenizing them to catcode 10 makes further processing difficult. Spaces with catcode 12 may be used just like any other catcode 12 token.

This requires that the \active end-of-text (ETX) ^C (U+0003) not be defined as \outer, since ^C is used as a sentinel. Usually, it should not be defined at all, or defined to an error sequence. When in doubt, it may be worth explicitly defining ^C before using \FVExtraVerbatimDetokenize:

```
\begingroup
\catcode`\^C=\active
\def^C{}
...
\FVExtraVerbatimDetokenize{...}
...
\endgroup
```

\detokenize inserts a space after each control word (control sequence with a name composed of catcode 11 tokens, ASCII letters [a-zA-Z]). For example,

```
\detokenize{\macroA\macroB{}}\csname name\endcsname 123}
```

yields

```
\macroA \macroB {} \csname name\endcsname 123
```

That is the correct behavior when detokenizing text that will later be retokenized for normal use. The space prevents the control word from accidentally merging with any letters that follow it immediately, and will be gobbled by the macro when retokenized. However, the inserted spaces are unwanted in the current context, because

```
\FVExtraVerbatimDetokenize{\macroA\macroB{ }\csname name\endcsname123}
```

should yield

```
\macroA\macroB{ }\csname_name\endcsname123
```

Note that the space is visible since it is catcode 12.

Thus, `\FVExtraVerbatimDetokenize` is essentially a context-sensitive wrapper around `\detokenize` that removes extraneous space introduced by `\detokenize`. It iterates through the tokens, detokenizing them individually and then removing any trailing space inserted by `\detokenize`.

```
725 \gdef\FVExtraVerbatimDetokenize#1{%
```

```
726 \FV@VDetok@Scan{ }#1^^C \FV@<Sentinel>}
```

`\FV@VDetok@Scan`

This scans through a token sequence while performing two tasks:

1. Replace all catcode 10 spaces with catcode 12 spaces.
2. Insert macros that will process groups, after which they will insert yet other macros to process individual tokens.

Usage must *always* have the form

```
\FV@VDetok@Scan{ }<tokens>^^C_\FV@<Sentinel>
```

where `^^C` is `\active`, the catcode 10 space after `^^C` is mandatory, and `\FV@<Sentinel>` is a *single*, undefined control word (this is accomplished via catcodes).

- `\FV@VDetok@Scan` searches for spaces to replace. After any spaces in `<tokens>` have been handled, the space in `^^C_\FV@<Sentinel>` triggers space processing. When `\FV@VDetok@Scan` detects the sentinel macro `\FV@<Sentinel>`, scanning stops.
- The `{ }` protects the beginning of `<tokens>`, so that if `<tokens>` is a group, its braces won't be gobbled. Later, the inserted `{ }` must be stripped so that it does not become part the processed `<tokens>`.
- `^^C` is a convenient separator between `<tokens>` and the rest of the sentinel sequence.
  - Since `\FV@VDetok@Scan` has delimited arguments, a leading catcode 10 space in `<tokens>` will be preserved automatically. Preserving a trailing catcode 10 space is much easier if it is immediately adjacent to a non-space character in the sentinel sequence; two adjacent catcode 10 spaces would be difficult to handle with macro pattern matching. However, the sentinel sequence must contain a catcode 10 space, so the sentinel sequence must contain at least 3 tokens.
  - Since `^^C` is not a control word, it does not gobble following spaces. That makes it much easier to assemble macro arguments that contain a catcode 10 space. This is useful because the sentinel sequence `^^C_\FV@<Sentinel>` may have to be inserted into processing multiple times (for example, in recursive handling of groups).

- `\FVExtraReadVArg` defines `^^C` as `\outer`, so any verbatim argument read by it is guaranteed not to contain `^^C`. This is in contrast to `\active` ASCII symbols and to two-character sequences `<backslash><symbol>` that should be expected in arbitrary verbatim content. It is a safe sentinel from that perspective.
- A search of a complete TeX Live 2018 installation revealed no other uses of `^^C` that would clash (thanks, `ripgrep!`). As a control character, it should not be in common use except as a sentinel or for similar special purposes.

If `<tokens>` is empty or contains no spaces, then `#1` will contain `{<tokens>}^^C` and `#2` will be empty. Otherwise, `#1` will contain `{<tokens_to_space>}` and `#2` will contain `<tokens_after_space>^^C`.

This uses the `\if\relax\detokenize{<argument>}\relax` approach to check for an empty argument. If `#2` is empty, then the space that was just removed by `\FV@VDetok@Scan` reading its arguments was the space in the sentinel sequence, in which case scanning should end. `#1` is passed on raw so that `\FV@VDetok@ScanEnd` can strip the `^^C` from the end, which is the only remaining token from the sentinel sequence `^^C_\FV@<Sentinel>`. Otherwise, if `#2` is not empty, continue. In that case, the braces in `{#1}{#2}` ensure arguments remain intact.

Note that `\FV@<Sentinel>` is removed during each space search, and thus must be reinserted in `\FV@VDetok@ScanCont`. It would be possible to use the macro signature `#1 #2` instead of `#1 #2\FV@<Sentinel>`, and then do an `\ifx` test on `#2` for `\FV@<Sentinel>`. However, that is problematic, because `#2` may contain an arbitrary sequence of arbitrary tokens, so it cannot be used safely without `\detokenize`.

```

727 \gdef\FV@VDetok@Scan#1 #2\FV@<Sentinel>{%
728   \if\relax\detokenize{#2}\relax
729     \expandafter\@firstoftwo
730   \else
731     \expandafter\@secondoftwo
732   \fi
733   {\FV@VDetok@ScanEnd#1}%
734   {\FV@VDetok@ScanCont{#1}{#2}}}
```

#### `\FV@VDetok@ScanEnd`

This removes the `^^C` from the sentinel sequence `^^C_\FV@<Sentinel>`, so the sentinel sequence is now completely gone. If `#1` is empty, there is nothing to do (`#1` being empty means that `#1` consumed the `{}` that was inserted to protect anything following, because there was nothing after it). Otherwise, `\@gobble` the inserted `{}` before starting a different scan to deal with groups. The group scanner `\FV@VDetok@ScanGroup` has its own sentinel sequence `{\FV@<Sentinel>}`.

```

735 \gdef\FV@VDetok@ScanEnd#1^^C{%
736   \if\relax\detokenize{#1}\relax
737     \expandafter\@gobble
738   \else
739     \expandafter\@firstofone
740   \fi
741   {\expandafter\FV@VDetok@ScanGroup\@gobble#1{\FV@<Sentinel>}}}
```

#### `\FV@VDetok@ScanCont`

Continue scanning after removing a space in `\FV@VDetok@Scan`.

#1 is everything before the space. If #1 is empty, there is nothing to do related to it; #1 simply consumed an inserted {} that preceded nothing (that would be a leading space). Otherwise, start a different scan on #1 to deal with groups. A non-empty #1 will start with the {} that was inserted to protect groups, hence the \@gobble before group scanning.

Then insert a literal catcode 12 space to account for the space removed in \FV@VDetok@Scan. Note the catcode, and thus the lack of indentation and the % to avoid unwanted catcode 12 spaces.

#2 is everything after the space, ending with ^^C\_ from the sentinel sequence ^^C\_\FV@<Sentinel>. This needs continued scanning to deal with spaces, with {} inserted in front to protect a leading group and \FV@<Sentinel> after to complete the sentinel sequence.

```

742 \begingroup
743 \catcode\ =12%
744 \gdef\FV@VDetok@ScanCont#1#2{%
745 \if\relax\detokenize{#1}\relax%
746 \expandafter\@gobble%
747 \else%
748 \expandafter\@firstofone%
749 \fi%
750 {\expandafter\FV@VDetok@ScanGroup\@gobble#1{\FV@<Sentinel>}}%
751 %<-catcode 12 space
752 \FV@VDetok@Scan{}#2\FV@<Sentinel>}%
753 \endgroup

```

#### \FV@VDetok@ScanGroup

The macro argument #1# reads up to the next group. When this macro is invoked, the sentinel sequence {\FV@<Sentinel>} is inserted, so there is guaranteed to be at least one group.

Everything in #1 contains no spaces and no groups, and thus is ready for token scanning, with the sentinel \FV@Sentinel. Note that \FV@Sentinel, which is defined as \def\FV@Sentinel{\FV@<Sentinel>}, is used here, *not* \FV@<Sentinel>. \FV@<Sentinel> is not defined and is thus unsuitable for \ifx comparisons with tokens that may have been tokenized under an incorrect catcode regime and thus are undefined. \FV@Sentinel is defined, and its definition is resistant against accidental collisions.

```

754 \gdef\FV@VDetok@ScanGroup#1#{%
755 \FV@VDetok@ScanToken#1\FV@Sentinel
756 \FV@VDetok@ScanGroup@i}

```

#### \FV@VDetok@ScanGroup@i

The braces from the group are stripped during reading #1. Proceed based on whether the group is empty. If the group is not empty, {} must be inserted to protect #1 in case it is a group, and the new sentinel sequence \FV@<Sentinel>^^C is added for the group contents. \FV@<Sentinel> cannot be used as a sentinel for the group contents, because if this is the sentinel group {\FV@<Sentinel>}, then #1 is \FV@<Sentinel>.

```

757 \gdef\FV@VDetok@ScanGroup@i#1{%
758 \if\relax\detokenize{#1}\relax
759 \expandafter\@firstoftwo
760 \else
761 \expandafter\@secondoftwo

```

```

762 \fi
763 {\FV@VDetok@ScanEmptyGroup}%
764 {\FV@VDetok@ScanGroup@ii{ }#1\FV@<Sentinel>^^C}}

```

#### \FV@VDetok@ScanEmptyGroup

Insert {} to handle the empty group, then continue group scanning.

```

765 \begingroup
766 \catcode`\(=1
767 \catcode`\)=2
768 \catcode`\{=12
769 \catcode`\}=12
770 \gdef\FV@VDetok@ScanEmptyGroup({}\FV@VDetok@ScanGroup)
771 \endgroup

```

#### \FV@VDetok@ScanGroup@ii

The group is not empty, so determine whether it contains \FV@<Sentinel> and thus is the sentinel group. The group contents are followed by the sentinel sequence \FV@<Sentinel>^^C inserted in \FV@VDetok@ScanGroup@i. This means that if #2 is empty, the group did not contain \FV@<Sentinel> and thus is not the sentinel group. Otherwise, #2 will be \FV@<Sentinel>.

If this is not the sentinel group, then the group contents must be scanned, with surrounding literal braces inserted. #1 already contains an inserted leading {} to protect groups; see \FV@VDetok@ScanGroup@i. A sentinel sequence ^^C\FV@<Sentinel> is needed, though. Then group scanning must continue.

```

772 \begingroup
773 \catcode`\(=1
774 \catcode`\)=2
775 \catcode`\{=12
776 \catcode`\}=12
777 \gdef\FV@VDetok@ScanGroup@ii#1\FV@<Sentinel>#2^^C(%)
778 \if\relax\detokenize(#2)\relax
779 \expandafter\@firstofone
780 \else
781 \expandafter\@gobble
782 \fi
783 ({\FV@VDetok@Scan#1^^C \FV@<Sentinel>}\FV@VDetok@ScanGroup))
784 \endgroup

```

#### \FV@VDetok@ScanToken

Scan individual tokens. At this point, all spaces and groups have been handled, so this will only ever encounter individual tokens that can be iterated with a #1 argument. The sentinel for token scanning is \FV@Sentinel. This is the appropriate sentinel because \ifx comparisons are now safe (individual tokens) and \FV@Sentinel is defined. Processing individual detokenized tokens requires the same sentinel sequence as handling spaces, since it can produce them.

```

785 \gdef\FV@VDetok@ScanToken#1{%(
786 \ifx\FV@Sentinel#1%
787 \expandafter\@gobble
788 \else
789 \expandafter\@firstofone
790 \fi
791 {\expandafter\FV@VDetok@ScanToken@i\detokenize{#1}^^C \FV@<Sentinel>}}

```

#### \FV@VDetok@ScanToken@i



If #2 is empty, then there are no spaces in the detokenized token, so it is either an \active character other than the space, or a two-character sequence of the form <backslash><symbol> where the second character is not a space. Thus, #1 contains <detokenized>^^C. Otherwise, #1 contains <detokenized\_without\_space>, and #2 may be discarded since it contains ^^C\_\FV@<Sentinel>. (If the detokenized token contains a space, it is always at the end.)

```

792 \gdef\FV@VDetok@ScanToken@i#1 #2\FV@<Sentinel>{%
793   \if\relax\detokenize{#2}\relax
794     \expandafter\@firstoftwo
795   \else
796     \expandafter\@secondoftwo
797   \fi
798   {\FV@VDetok@ScanTokenNoSpace#1}%
799   {\FV@VDetok@ScanTokenWithSpace{#1}}}
```

#### \FV@VDetok@ScanTokenNoSpace

Strip ^^C sentinel in reading, then insert character(s) and continue scanning.

```
800 \gdef\FV@VDetok@ScanTokenNoSpace#1^^C{#1\FV@VDetok@ScanToken}
```

#### \FV@VDetok@ScanTokenWithSpace

Handle a token that when detokenized produces a space. If there is nothing left once the space is removed, this is the \active space. Otherwise, process further.

```

801 \gdef\FV@VDetok@ScanTokenWithSpace#1{%
802   \if\relax\detokenize{#1}\relax
803     \expandafter\@firstoftwo
804   \else
805     \expandafter\@secondoftwo
806   \fi
807   {\FV@VDetok@ScanTokenActiveSpace}%
808   {\FV@VDetok@ScanTokenWithSpace@i#1\FV@<Sentinel>}}
```

#### \FV@VDetok@ScanTokenActiveSpace

```

809 \begingroup
810 \catcode`\ =12%
811 \gdef\FV@VDetok@ScanTokenActiveSpace{ \FV@VDetok@ScanToken}%
812 \endgroup
```

#### \FV@VDetok@ScanTokenWithSpace@i

If there is only one character left once the space is removed, this is the escaped space \\_. Otherwise, this is a command word that needs further processing.

```

813 \gdef\FV@VDetok@ScanTokenWithSpace@i#1#2\FV@<Sentinel>{%
814   \if\relax\detokenize{#2}\relax
815     \expandafter\@firstoftwo
816   \else
817     \expandafter\@secondoftwo
818   \fi
819   {\FV@VDetok@ScanTokenEscSpace{#1}}%
820   {\FV@VDetok@ScanTokenCW{#1#2}}}
```

#### \FV@VDetok@ScanTokenEscSpace

```

821 \begingroup
822 \catcode`\ =12%
823 \gdef\FV@VDetok@ScanTokenEscSpace#1{#1 \FV@VDetok@ScanToken}%
824 \endgroup
```

### `\FV@VDetok@ScanTokenCW`

Process control words in a context-sensitive manner by looking ahead to the next token (#2). The lookahead must be reinserted into processing, hence the `\FV@VDetok@ScanToken#2`.

A control word will detokenize to a sequence of characters followed by a space. If the following token has catcode 11, then this space represents one or more space characters that must have been present in the original source, because otherwise the catcode 11 token would have become part of the control word's name. If the following token has another catcode, then it is impossible to determine whether a space was present, so assume that one was not.

```
825 \begingroup
826 \catcode\ =12%
827 \gdef\FV@VDetok@ScanTokenCW#1#2{%
828 \ifcat\noexpand#2a%
829 \expandafter\@firstoftwo%
830 \else%
831 \expandafter\@secondoftwo%
832 \fi%
833 {#1\FV@VDetok@ScanToken#2}%
834 {#1\FV@VDetok@ScanToken#2}}%
835 \endgroup
```

**Detokenize as if the original source were tokenized verbatim, then convert to PDF string**

### `\FVExtraPDFStringVerbatimDetokenize`

This is identical to `\FVExtraVerbatimDetokenize`, except that the output is converted to a valid PDF string. Some spaces are represented with the octal escape `\040` to prevent adjacent spaces from being merged.

```
836 \gdef\FVExtraPDFStringVerbatimDetokenize#1{%
837 \FV@PDFStrVDetok@Scan{#1}^C\FV@<Sentinel>}
```

### `\FV@PDFStrVDetok@Scan`

```
838 \gdef\FV@PDFStrVDetok@Scan#1 #2\FV@<Sentinel>{%
839 \if\relax\detokenize{#2}\relax
840 \expandafter\@firstoftwo
841 \else
842 \expandafter\@secondoftwo
843 \fi
844 {\FV@PDFStrVDetok@ScanEnd#1}%
845 {\FV@PDFStrVDetok@ScanCont{#1}{#2}}}
```

### `\FV@PDFStrVDetok@ScanEnd`

```
846 \gdef\FV@PDFStrVDetok@ScanEnd#1^C{%
847 \if\relax\detokenize{#1}\relax
848 \expandafter\@gobble
849 \else
850 \expandafter\@firstofone
851 \fi
852 {\expandafter\FV@PDFStrVDetok@ScanGroup\@gobble#1{\FV@<Sentinel>}}}
```

### `\FV@PDFStrVDetok@ScanCont`

```

853 \begingroup
854 \catcode`\ =12%
855 \gdef\FV@PDFStrVDetok@ScanCont#1#2{%
856 \if\relax\detokenize{#1}\relax%
857 \expandafter\@gobble%
858 \else%
859 \expandafter\@firstofone%
860 \fi%
861 {\expandafter\FV@PDFStrVDetok@ScanGroup\@gobble#1{\FV@<Sentinel>}}}%
862 %<-catcode 12 space
863 \FV@PDFStrVDetok@Scan{ }#2\FV@<Sentinel>}%
864 \endgroup

\FV@PDFStrVDetok@ScanGroup
865 \gdef\FV@PDFStrVDetok@ScanGroup#1#1{%
866 \FV@PDFStrVDetok@ScanToken#1\FV@Sentinel
867 \FV@PDFStrVDetok@ScanGroup@i}

\FV@PDFStrVDetok@ScanGroup@i
868 \gdef\FV@PDFStrVDetok@ScanGroup@i#1#1{%
869 \if\relax\detokenize{#1}\relax
870 \expandafter\@firstoftwo
871 \else
872 \expandafter\@secondoftwo
873 \fi
874 {\FV@PDFStrVDetok@ScanEmptyGroup}%
875 {\FV@PDFStrVDetok@ScanGroup@ii{ }#1\FV@<Sentinel>^^C}}

\FV@PDFStrVDetok@ScanEmptyGroup
876 \begingroup
877 \catcode`\ (=1
878 \catcode`\ )=2
879 \catcode`\ {=12
880 \catcode`\ }=12
881 \gdef\FV@PDFStrVDetok@ScanEmptyGroup({}\FV@PDFStrVDetok@ScanGroup)
882 \endgroup

\FV@PDFStrVDetok@ScanGroup@ii
883 \begingroup
884 \catcode`\ (=1
885 \catcode`\ )=2
886 \catcode`\ {=12
887 \catcode`\ }=12
888 \gdef\FV@PDFStrVDetok@ScanGroup@ii#1\FV@<Sentinel>#2^^C(%
889 \if\relax\detokenize{#2}\relax
890 \expandafter\@firstofone
891 \else
892 \expandafter\@gobble
893 \fi
894 ({\FV@PDFStrVDetok@Scan#1^^C \FV@<Sentinel>}\FV@PDFStrVDetok@ScanGroup))
895 \endgroup

\FV@PDFStrVDetok@ScanToken
896 \gdef\FV@PDFStrVDetok@ScanToken#1#1{%

```

```

897 \ifx\FV@Sentinel#1%
898   \expandafter\@gobble
899 \else
900   \expandafter\@firstofone
901 \fi
902 {\expandafter\FV@PDFStrVDetok@ScanToken@i\detokenize{#1}^^C \FV@<Sentinel>}}

\FV@PDFStrVDetok@ScanToken@i
903 \gdef\FV@PDFStrVDetok@ScanToken@i#1 #2\FV@<Sentinel>{%
904   \if\relax\detokenize{#2}\relax
905     \expandafter\@firstoftwo
906   \else
907     \expandafter\@secondoftwo
908   \fi
909   {\FV@PDFStrVDetok@ScanTokenNoSpace#1}%
910   {\FV@PDFStrVDetok@ScanTokenWithSpace{#1}}}}

\FV@PDFStrVDetok@ScanTokenNoSpace
  This is modified to use \FVExtraPDFStringEscapeChars.
911 \gdef\FV@PDFStrVDetok@ScanTokenNoSpace#1^^C{%
912   \FVExtraPDFStringEscapeChars{#1}\FV@PDFStrVDetok@ScanToken}

\FV@PDFStrVDetok@ScanTokenWithSpace
913 \gdef\FV@PDFStrVDetok@ScanTokenWithSpace#1{%
914   \if\relax\detokenize{#1}\relax
915     \expandafter\@firstoftwo
916   \else
917     \expandafter\@secondoftwo
918   \fi
919   {\FV@PDFStrVDetok@ScanTokenActiveSpace}%
920   {\FV@PDFStrVDetok@ScanTokenWithSpace@i#1\FV@<Sentinel>}}

\FV@PDFStrVDetok@ScanTokenActiveSpace
  This is modified to use \040 rather than a catcode 12 space.
921 \begingroup
922 \catcode`\!=0\relax
923 \catcode`\=12\relax
924 !gdef\FV@PDFStrVDetok@ScanTokenActiveSpace{\040\FV@PDFStrVDetok@ScanToken}%
925 !catcode`\!=0!relax
926 \endgroup

\FV@PDFStrVDetok@ScanTokenWithSpace@i
  If there is only one character left once the space is removed, this is the escaped
  space \.. Otherwise, this is a command word that needs further processing.
927 \gdef\FV@PDFStrVDetok@ScanTokenWithSpace@i#1#2\FV@<Sentinel>{%
928   \if\relax\detokenize{#2}\relax
929     \expandafter\@firstoftwo
930   \else
931     \expandafter\@secondoftwo
932   \fi
933   {\FV@PDFStrVDetok@ScanTokenEscSpace{#1}}%
934   {\FV@PDFStrVDetok@ScanTokenCW{#1#2}}}}

\FV@PDFStrVDetok@ScanTokenEscSpace

```

This is modified to add `\FVExtraPDFStringEscapeChar` and use `\040` for the space, since a space could follow.

```

935 \begingroup
936 \catcode`\!=0\relax
937 \catcode`\=12\relax
938 !gdef\FV@PDFStrVDetok@ScanTokenEscSpace#1{%
939   !FVExtraPDFStringEscapeChar{#1}\040!FV@PDFStrVDetok@ScanToken}%
940 !catcode`\=0\relax
941 \endgroup

```

`\FV@PDFStrVDetok@ScanTokenCW`

This is modified to add `\FVExtraPDFStringEscapeChars`.

```

942 \begingroup
943 \catcode`\ =12%
944 \gdef\FV@PDFStrVDetok@ScanTokenCW#1#2{%
945   \ifcat\noexpand#2a%
946     \expandafter\@firstoftwo%
947   \else%
948     \expandafter\@secondoftwo%
949   \fi%
950   {\FVExtraPDFStringEscapeChars{#1} \FV@PDFStrVDetok@ScanToken#2}%
951   {\FVExtraPDFStringEscapeChars{#1}\FV@PDFStrVDetok@ScanToken#2}}
952 \endgroup

```

**Detokenize as if the original source were tokenized verbatim, except for backslash escapes of non-catcode 11 characters**

`\FVExtraEscapedVerbatimDetokenize`

This is a variant of `\FVExtraVerbatimDetokenize` that treats character sequences of the form `\<char>` as escapes for `<char>`. It is primarily intended for making `\<symbol>` escapes for `<symbol>`, but allowing arbitrary escapes simplifies the default behavior and implementation. This is useful in constructing nearly verbatim commands that can be used inside other commands, because the backslash escapes allow for characters like `#` and `%`, as well as making possible multiple adjacent spaces via `\.` It should be applied to arguments that are read verbatim insofar as is possible, except that the backslash `\` should have its normal meaning (catcode 0). Most of the implementation is identical to that for `\FVExtraVerbatimDetokenize`. Only the token processing requires modification to handle backslash escapes.

It is possible to restrict escapes to ASCII symbols and punctuation. See `\FVExtraDetokenizeREscVArg`. The disadvantage of restricting escapes is that it prevents functioning in an expansion-only context (unless you want to use undefined macros as a means of raising errors). The advantage is that it eliminates ambiguity introduced by allowing arbitrary escapes. Backslash escapes of characters with catcode 11 (ASCII letters, [A-Za-z]) are typically not necessary, and introduce ambiguity because something like `\x` will gobble following spaces since it will be tokenized originally as a control word.

```

953 \gdef\FVExtraEscapedVerbatimDetokenize#1{%
954   \FV@EscVDetok@Scan{#1}^C \FV@<Sentinel>}

```

`\FV@EscVDetok@Scan`

```

955 \gdef\FV@EscVDetok@Scan#1 #2\FV@<Sentinel>{%
956   \if\relax\detokenize{#2}\relax

```

```

957     \expandafter\@firstoftwo
958   \else
959     \expandafter\@secondoftwo
960   \fi
961   {\FV@EscVDetok@ScanEnd#1}%
962   {\FV@EscVDetok@ScanCont{#1}{#2}}}}

\FV@EscVDetok@ScanEnd
963 \gdef\FV@EscVDetok@ScanEnd#1^^C{%
964   \if\relax\detokenize{#1}\relax
965     \expandafter\@gobble
966   \else
967     \expandafter\@firstofone
968   \fi
969   {\expandafter\FV@EscVDetok@ScanGroup\@gobble#1{\FV@<Sentinel>}}}}

\FV@EscVDetok@ScanCont
970 \begingroup
971 \catcode`\ =12%
972 \gdef\FV@EscVDetok@ScanCont#1#2{%
973   \if\relax\detokenize{#1}\relax%
974   \expandafter\@gobble%
975   \else%
976   \expandafter\@firstofone%
977   \fi%
978   {\expandafter\FV@EscVDetok@ScanGroup\@gobble#1{\FV@<Sentinel>}}%
979   %<-catcode 12 space
980   \FV@EscVDetok@Scan{#2}\FV@<Sentinel>}}%
981 \endgroup

\FV@EscVDetok@ScanGroup
982 \gdef\FV@EscVDetok@ScanGroup#1#1{%
983   \FV@EscVDetok@ScanToken#1\FV@Sentinel
984   \FV@EscVDetok@ScanGroup@i}

\FV@EscVDetok@ScanGroup@i
985 \gdef\FV@EscVDetok@ScanGroup@i#1{%
986   \if\relax\detokenize{#1}\relax
987     \expandafter\@firstoftwo
988   \else
989     \expandafter\@secondoftwo
990   \fi
991   {\FV@EscVDetok@ScanEmptyGroup}%
992   {\FV@EscVDetok@ScanGroup@ii{#1}\FV@<Sentinel>^^C}}

\FV@EscVDetok@ScanEmptyGroup
993 \begingroup
994 \catcode`\ (=1
995 \catcode`\ )=2
996 \catcode`\ {=12
997 \catcode`\ }=12
998 \gdef\FV@EscVDetok@ScanEmptyGroup({}\FV@EscVDetok@ScanGroup)
999 \endgroup

```

```

\FV@EscVDetok@ScanGroup@ii
1000 \begingroup
1001 \catcode`\(=1
1002 \catcode`\)=2
1003 \catcode`\{=12
1004 \catcode`\}=12
1005 \gdef\FV@EscVDetok@ScanGroup@ii#1\FV@<Sentinel>#2^C{%
1006   \if\relax\detokenize{#2}\relax
1007     \expandafter\@firstofone
1008   \else
1009     \expandafter\@gobble
1010   \fi
1011   ({\FV@EscVDetok@Scan#1^C \FV@<Sentinel>}\FV@EscVDetok@ScanGroup)}
1012 \endgroup

\FV@EscVDetok@ScanToken
1013 \gdef\FV@EscVDetok@ScanToken#1{%
1014   \ifx\FV@Sentinel#1%
1015     \expandafter\@gobble
1016   \else
1017     \expandafter\@firstofone
1018   \fi
1019   {\expandafter\FV@EscVDetok@ScanToken@i\detokenize{#1}^C \FV@<Sentinel>}}

\FV@EscVDetok@ScanToken@i
1020 \gdef\FV@EscVDetok@ScanToken@i#1 #2\FV@<Sentinel>{%
1021   \if\relax\detokenize{#2}\relax
1022     \expandafter\@firstoftwo
1023   \else
1024     \expandafter\@secondoftwo
1025   \fi
1026   {\FV@EscVDetok@ScanTokenNoSpace#1}%
1027   {\FV@EscVDetok@ScanTokenWithSpace#1}}

```

**Parallel implementations, with a restricted option** Starting here, there are alternate macros for restricting escapes to ASCII punctuation and symbols. These alternates have names of the form `\FV@REscVDetok@<name>`. They are used in `\FVExtraDetokenizeREscVArg`. The alternate `\FV@REscVDetok@<name>` macros replace invalid escape sequences with the undefined `\FV@<InvalidEscape>`, which is later scanned for with a delimited macro.

#### `\FV@EscVDetok@ScanTokenNoSpace`

This was modified from `\FV@VDetok@ScanTokenNoSpace` to discard the first character of multi-character sequences (that would be the backslash `\`).

```

1028 \gdef\FV@EscVDetok@ScanTokenNoSpace#1#2^C{%
1029   \if\relax\detokenize{#2}\relax
1030     \expandafter\@firstoftwo
1031   \else
1032     \expandafter\@secondoftwo
1033   \fi
1034   {#1\FV@EscVDetok@ScanToken}%
1035   {#2\FV@EscVDetok@ScanToken}}

```

### `\FV@EscVDetok@ScanTokenNoSpace`

```
1036 \gdef\FV@EscVDetok@ScanTokenNoSpace#1#2^^C{%
1037   \if\relax\detokenize{#2}\relax
1038     \expandafter\@firstoftwo
1039   \else
1040     \expandafter\@secondoftwo
1041   \fi
1042   {#1\FV@EscVDetok@ScanToken}%
1043   {\ifcsname FV@Char@Special:\number`#2\endcsname#2\else\noexpand\FV@<InvalidEscape>\fi
1044   \FV@EscVDetok@ScanToken}}
```

### `\FV@EscVDetok@ScanTokenWithSpace`

```
1045 \gdef\FV@EscVDetok@ScanTokenWithSpace#1{%
1046   \if\relax\detokenize{#1}\relax
1047     \expandafter\@firstoftwo
1048   \else
1049     \expandafter\@secondoftwo
1050   \fi
1051   {\FV@EscVDetok@ScanTokenActiveSpace}%
1052   {\FV@EscVDetok@ScanTokenWithSpace@i#1\FV@<Sentinel>}}
```

### `\FV@EscVDetok@ScanTokenActiveSpace`

```
1053 \begingroup
1054 \catcode`\ =12%
1055 \gdef\FV@EscVDetok@ScanTokenActiveSpace{ \FV@EscVDetok@ScanToken}%
1056 \endgroup
```

### `\FV@EscVDetok@ScanTokenWithSpace@i`

If there is only one character left once the space is removed, this is the escaped space `\.`. Otherwise, this is a command word. A command word is passed on so as to keep the backslash and letters separate.

```
1057 \gdef\FV@EscVDetok@ScanTokenWithSpace@i#1#2\FV@<Sentinel>{%
1058   \if\relax\detokenize{#2}\relax
1059     \expandafter\@firstoftwo
1060   \else
1061     \expandafter\@secondoftwo
1062   \fi
1063   {\FV@EscVDetok@ScanTokenEscSpace{#1}}%
1064   {\FV@EscVDetok@ScanTokenCW{#1}{#2}}}
```

### `\FV@EscVDetok@ScanTokenWithSpace@i`

```
1065 \gdef\FV@EscVDetok@ScanTokenWithSpace@i#1#2\FV@<Sentinel>{%
1066   \if\relax\detokenize{#2}\relax
1067     \expandafter\@firstoftwo
1068   \else
1069     \expandafter\@secondoftwo
1070   \fi
1071   {\FV@EscVDetok@ScanTokenEscSpace{#1}}%
1072   {\noexpand\FV@<InvalidEscape>\FV@EscVDetok@ScanToken}}
```

### `\FV@EscVDetok@ScanTokenEscSpace`

This is modified to drop #1, which will be the backslash.

```
1073 \begingroup
1074 \catcode`\ =12%
```



```

1075 \gdef\FV@EscVDetok@ScanTokenEscSpace#1{ \FV@EscVDetok@ScanToken}%
1076 \endgroup

```

#### \FV@EscVDetok@ScanTokenCW

This is modified to accept an additional argument, since the control word is now split into backslash plus letters.

```

1077 \begingroup
1078 \catcode` \ =12%
1079 \gdef\FV@EscVDetok@ScanTokenCW#1#2#3{%
1080 \ifcat\noexpand#2a%
1081 \expandafter\@firstoftwo%
1082 \else%
1083 \expandafter\@secondoftwo%
1084 \fi%
1085 {#2 \FV@EscVDetok@ScanToken#3}%
1086 {#2\FV@EscVDetok@ScanToken#3}}
1087 \endgroup

```

**Detokenize as if the original source were tokenized verbatim, except for backslash escapes of non-catcode 11 characters, then convert to PDF string**

#### \FVExtraPDFStringEscapedVerbatimDetokenize

This is identical to \FVExtraEscapedVerbatimDetokenize, except that the output is converted to a valid PDF string. All spaces are represented with the octal escape \040 to prevent adjacent spaces from being merged. There is no alternate implementation for restricting escapes to ASCII symbols and punctuation. Typically, this would be used in an expansion-only context to create something like bookmarks, while \FVExtraEscapedVerbatimDetokenize (potentially with escape restrictions) would be used in parallel to generate whatever is actually typeset. Escape errors can be handled in generating what is typeset.

```

1088 \gdef\FVExtraPDFStringEscapedVerbatimDetokenize#1{%
1089 \FV@PDFStrEscVDetok@Scan{ }#1^~C \FV@<Sentinel>}

```

#### \FV@PDFStrEscVDetok@Scan

```

1090 \gdef\FV@PDFStrEscVDetok@Scan#1 #2\FV@<Sentinel>{%
1091 \if\relax\detokenize{#2}\relax
1092 \expandafter\@firstoftwo
1093 \else
1094 \expandafter\@secondoftwo
1095 \fi
1096 {\FV@PDFStrEscVDetok@ScanEnd#1}%
1097 {\FV@PDFStrEscVDetok@ScanCont{#1}{#2}}}

```

#### \FV@PDFStrEscVDetok@ScanEnd

```

1098 \gdef\FV@PDFStrEscVDetok@ScanEnd#1^~C{%
1099 \if\relax\detokenize{#1}\relax
1100 \expandafter\@gobble
1101 \else
1102 \expandafter\@firstofone
1103 \fi
1104 {\expandafter\FV@PDFStrEscVDetok@ScanGroup\@gobble#1{\FV@<Sentinel>}}}

```

### `\FV@PDFStrEscVDetok@ScanCont`

This is modified to use `\040` for the space. In the unescaped case, using a normal space here is fine, but in the escaped case, the preceding or following token could be an escaped space.

```
1105 \begingroup
1106 \catcode`\!=0\relax
1107 \catcode`\=12\relax
1108 !gdef\FV@PDFStrEscVDetok@ScanCont#1#2{%
1109   !if\relax\detokenize{#1}\relax
1110     !expandafter!\@gobble
1111   !else
1112     !expandafter!\@firstofone
1113   !fi
1114   {!expandafter\FV@PDFStrEscVDetok@ScanGroup!\@gobble#1{!FV@<Sentinel>}}}%
1115   \040%<-space
1116   !FV@PDFStrEscVDetok@Scan{ }#2!FV@<Sentinel>}}}%
1117 !catcode`\!=0\relax
1118 \endgroup
```

### `\FV@PDFStrEscVDetok@ScanGroup`

```
1119 \gdef\FV@PDFStrEscVDetok@ScanGroup#1#{%
1120   \FV@PDFStrEscVDetok@ScanToken#1\FV@Sentinel
1121   \FV@PDFStrEscVDetok@ScanGroup@i}
```

### `\FV@PDFStrEscVDetok@ScanGroup@i`

```
1122 \gdef\FV@PDFStrEscVDetok@ScanGroup@i#1{%
1123   \if\relax\detokenize{#1}\relax
1124     \expandafter!\@firstoftwo
1125   \else
1126     \expandafter!\@secondoftwo
1127   \fi
1128   {\FV@PDFStrEscVDetok@ScanEmptyGroup}%
1129   {\FV@PDFStrEscVDetok@ScanGroup@ii{ }#1\FV@<Sentinel>^^C}}
```

### `\FV@PDFStrEscVDetok@ScanEmptyGroup`

```
1130 \begingroup
1131 \catcode`\(=1
1132 \catcode`\)=2
1133 \catcode`\{=12
1134 \catcode`\}=12
1135 \gdef\FV@PDFStrEscVDetok@ScanEmptyGroup({}\FV@PDFStrEscVDetok@ScanGroup)
1136 \endgroup
```

### `\FV@PDFStrEscVDetok@ScanGroup@ii`

```
1137 \begingroup
1138 \catcode`\(=1
1139 \catcode`\)=2
1140 \catcode`\{=12
1141 \catcode`\}=12
1142 \gdef\FV@PDFStrEscVDetok@ScanGroup@ii#1\FV@<Sentinel>#2^^C(%
1143   \if\relax\detokenize{#2}\relax
1144     \expandafter!\@firstofone
1145   \else
1146     \expandafter!\@gobble
```

```

1147 \fi
1148 ({\FV@PDFStrEscVDetok@Scan#1^^C \FV@<Sentinel>}\FV@PDFStrEscVDetok@ScanGroup))
1149 \endgroup

\FV@PDFStrEscVDetok@ScanToken
1150 \gdef\FV@PDFStrEscVDetok@ScanToken#1{%
1151 \ifx\FV@Sentinel#1%
1152 \expandafter@gobble
1153 \else
1154 \expandafter@firstofone
1155 \fi
1156 {\expandafter\FV@PDFStrEscVDetok@ScanToken@i\detokenize{#1}^^C \FV@<Sentinel>}}

\FV@PDFStrEscVDetok@ScanToken@i
1157 \gdef\FV@PDFStrEscVDetok@ScanToken@i#1 #2\FV@<Sentinel>{%
1158 \if\relax\detokenize{#2}\relax
1159 \expandafter@firstoftwo
1160 \else
1161 \expandafter@secondoftwo
1162 \fi
1163 {\FV@PDFStrEscVDetok@ScanTokenNoSpace#1}%
1164 {\FV@PDFStrEscVDetok@ScanTokenWithSpace{#1}}}

\FV@PDFStrEscVDetok@ScanTokenNoSpace
This was modified to add \FVExtraPDFStringEscapeChar
1165 \gdef\FV@PDFStrEscVDetok@ScanTokenNoSpace#1#2^^C{%
1166 \if\relax\detokenize{#2}\relax
1167 \expandafter@firstoftwo
1168 \else
1169 \expandafter@secondoftwo
1170 \fi
1171 {\FVExtraPDFStringEscapeChar{#1}\FV@PDFStrEscVDetok@ScanToken}%
1172 {\FVExtraPDFStringEscapeChar{#2}\FV@PDFStrEscVDetok@ScanToken}}

\FV@PDFStrEscVDetok@ScanTokenWithSpace
1173 \gdef\FV@PDFStrEscVDetok@ScanTokenWithSpace#1{%
1174 \if\relax\detokenize{#1}\relax
1175 \expandafter@firstoftwo
1176 \else
1177 \expandafter@secondoftwo
1178 \fi
1179 {\FV@PDFStrEscVDetok@ScanTokenActiveSpace}%
1180 {\FV@PDFStrEscVDetok@ScanTokenWithSpace@i#1\FV@<Sentinel>}}

\FV@PDFStrEscVDetok@ScanTokenActiveSpace
This is modified to use \040 for the space.
1181 \begingroup
1182 \catcode`\!=0\relax
1183 \catcode`\|=12\relax
1184 !gdef!FV@PDFStrEscVDetok@ScanTokenActiveSpace{\040!FV@PDFStrEscVDetok@ScanToken}%
1185 !catcode`\!=0!relax
1186 \endgroup

\FV@PDFStrEscVDetok@ScanTokenWithSpace@i
1187 \gdef\FV@PDFStrEscVDetok@ScanTokenWithSpace@i#1#2\FV@<Sentinel>{%

```

```

1188 \if\relax\detokenize{#2}\relax
1189 \expandafter\@firstoftwo
1190 \else
1191 \expandafter\@secondoftwo
1192 \fi
1193 {\FV@PDFStrEscVDetok@ScanTokenEscSpace{#1}}%
1194 {\FV@PDFStrEscVDetok@ScanTokenCW{#1}{#2}}

```

#### `\FV@PDFStrEscVDetok@ScanTokenEscSpace`

This is modified to drop #1, which will be the backslash, and use \040 for the space.

```

1195 \begingroup
1196 \catcode`\!=0\relax
1197 \catcode`\=12\relax
1198 !gdef\FV@PDFStrEscVDetok@ScanTokenEscSpace#1{\040\FV@PDFStrEscVDetok@ScanToken}
1199 !catcode`\!=0\relax
1200 \endgroup

```

#### `\FV@PDFStrEscVDetok@ScanTokenCW`

This is modified to use \FVExtraPDFStringEscapeChars.

```

1201 \begingroup
1202 \catcode`\ =12%
1203 \gdef\FV@PDFStrEscVDetok@ScanTokenCW#1#2#3{%
1204 \ifcat\noexpand#2a%
1205 \expandafter\@firstoftwo%
1206 \else%
1207 \expandafter\@secondoftwo%
1208 \fi%
1209 {\FVExtraPDFStringEscapeChars{#2} \FV@PDFStrEscVDetok@ScanToken#3}%
1210 {\FVExtraPDFStringEscapeChars{#2}\FV@PDFStrEscVDetok@ScanToken#3}}
1211 \endgroup

```

### Detokenization wrappers

#### `\FVExtraDetokenizeVArg`

Detokenize a verbatim argument read by `\FVExtraReadVArg`. This is a wrapper around `\FVExtraVerbatimDetokenize` that adds some additional safety by ensuring `^^C` is `\active` with an appropriate definition, at the cost of not working in an expansion-only context. This tradeoff isn't an issue when working with `\FVExtraReadVArg`, because it has the same expansion limitations.

```

1212 \gdef\FVExtraDetokenizeVArg#1#2{%
1213 \begingroup
1214 \catcode`^^C=\active
1215 \let^^C\FV@Sentinel
1216 \edef\FV@Tmp{\FVExtraVerbatimDetokenize{#2}}%
1217 \expandafter\FV@DetokenizeVArg@i\expandafter{\FV@Tmp}{#1}}
1218 \gdef\FV@DetokenizeVArg@i#1#2{%
1219 \endgroup
1220 #2{#1}}

```

#### `\FVExtraDetokenizeEscVArg`

This is the same as `\FVExtraDetokenizeVArg`, except it is intended to work with `\FVExtraReadEscVArg` by using `\FVExtraEscapedVerbatimDetokenize`.

```

1221 \gdef\FVExtraDetokenizeEscVArg#1#2{%
1222   \begingroup
1223   \catcode\^^C=\active
1224   \let^^C\FV@Sentinel
1225   \edef\FV@Tmp{\FVExtraEscapedVerbatimDetokenize{#2}}%
1226   \expandafter\FV@DetokenizeVArg@i\expandafter{\FV@Tmp}{#1}}

```

#### \FVExtraDetokenizeREscVArg

```

1227 \gdef\FVExtraDetokenizeREscVArg#1#2{%
1228   \begingroup
1229   \catcode\^^C=\active
1230   \let^^C\FV@Sentinel
1231   \let\FV@EscVDetok@ScanTokenNoSpace\FV@REscVDetok@ScanTokenNoSpace
1232   \let\FV@EscVDetok@ScanTokenWithSpace@i\FV@REscVDetok@ScanTokenWithSpace@i
1233   \edef\FV@Tmp{\FVExtraEscapedVerbatimDetokenize{#2}}%
1234   \expandafter\FV@DetokenizeREscVArg@InvalidEscapeCheck\FV@Tmp\FV@<InvalidEscape>\FV@<Sentinel>{%
1235     \expandafter\FV@DetokenizeVArg@i\expandafter{\FV@Tmp}{#1}}
1236   \gdef\FV@DetokenizeREscVArg@InvalidEscapeCheck#1\FV@<InvalidEscape>#2\FV@<Sentinel>{%
1237     \if\relax\detokenize{#2}\relax
1238       \expandafter\@gobble
1239     \else
1240       \expandafter\@firstofone
1241     \fi
1242     {\PackageError{fvextra}%
1243      [Invalid backslash escape; only escape ASCII symbols and punctuation]}%
1244     {Only use \@backslashchar <char> for ASCII symbols and punctuation}}

```

End catcodes for this subsection:

```
1245 \endgroup
```

### 12.4.6 Retokenizing detokenized arguments

#### \FV@RetokVArg@Read

Read all tokens up to `\active ^^C^^M`, then save them in a macro for further use. This is used to read tokens inside `\scantokens` during retokenization. The `\endgroup` disables catcode modifications that will have been put in place for the reading process, including making `^^C` and `^^M` `\active`.

```

1246 \begingroup
1247 \catcode\^^C=\active%
1248 \catcode\^^M=\active%
1249 \gdef\FV@RetokVArg@Read#1^^C^^M{%
1250   \endgroup%
1251   \def\FV@TmpRetoked{#1}}%
1252 \endgroup

```

#### \FVExtraRetokenizeVArg

This retokenizes the detokenized output of something like `\FVExtraVerbatimDetokenize` or `\FVExtraDetokenizeVArg`. #1 is a macro that receives the output, #2 sets catcodes but includes no `\begingroup` or `\endgroup`, and #3 is the detokenized characters. `\FV@RetokVArg@Read` contains an `\endgroup` that returns catcodes to their prior state.

This is a somewhat atypical use of `\scantokens`. There is no `\everyeof{\noexpand}` to handle the end-of-file marker, and no `\endlinechar=-1` to ignore the end-of-line token so that it does not become a space. Rather, the end-of-line `^^M` is made

`\active` and used as a delimiter by `\FV@RetokVArg@Read`, which reads characters under the new catcode regime, then stores them unexpanded in `\FV@TmpRetoked`.

Inside `\scantokens` is `^^B#3^^C`. This becomes `^^B#3^^C^^M` once `\scantokens` inserts the end-of-line token. `^^B` is `\let` to `\FV@RetokVArg@Read`, rather than using `\FV@RetokVArg@Read` directly, because `\scantokens` acts as a `\write` followed by `\input`. That means that a command word like `\FV@RetokVArg@Read` will have a space inserted after it, while an `\active` character like `^^B` will not. Using `^^B` is a way to avoid needing to remove this space; it is simpler not to handle the scenario where `\FV@RetokVArg@Read` introduces a space and the detokenized characters also start with a space. The `^^C` is needed because trailing spaces on a line are automatically stripped, so a non-space character must be part of the delimiting token sequence.

```

1253 \begingroup
1254 \catcode`^^B=\active
1255 \catcode`^^C=\active
1256 \gdef\FVExtraRetokenizeVArg#1#2#3{%
1257   \begingroup
1258   #2%
1259   \catcode`^^B=\active
1260   \catcode`^^C=\active
1261   \catcode`^^M=\active
1262   \let^^B\FV@RetokVArg@Read
1263   \let^^C\@empty
1264   \FV@DefEOLEmpty
1265   \scantokens{^^B#3^^C}%
1266   \expandafter\FV@RetokenizeVArg@i\expandafter{\FV@TmpRetoked}{#1}}%
1267 \gdef\FV@RetokenizeVArg@i#1#2{%
1268   #2{#1}}
1269 \endgroup

```

## 12.5 Hooks

### `\FV@UseKeyValues@Hook`

This hook is useful for tasks that must be accomplished immediately after the beginning of a command or environment, just after `\begingroup`, before any catcode or similar changes are made. Hooking into `\FV@CatCodes` is not a straightforward alternative, because it may be invoked multiple times in the original `fancyvrb` code, plus additional times in `fvextra` code during retokenization. Hooking into `\FV@FormattingPrep` is not a reliable alternative, because it may be invoked before or after `\FV@CatCodes`.

```

1270 \let\FV@UseKeyValues@Hook\@empty
1271 \expandafter\def\expandafter\FV@UseKeyValues\expandafter{%
1272   \FV@UseKeyValues\FV@UseKeyValues@Hook}

```

### `\FV@FormattingPrep@PreHook`

### `\FV@FormattingPrep@PostHook`

These are hooks 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. These hooks are used for such things as tweaking math mode behavior and preparing for `breakbefore` and `breakafter`. The `PreHook` should typically be used, unless `fancyvrb`'s font settings, whitespace setup, and active character definitions are needed for extension code.

```

1273 \let\FV@FormattingPrep@PreHook\@empty
1274 \let\FV@FormattingPrep@PostHook\@empty
1275 \expandafter\def\expandafter\FV@FormattingPrep\expandafter{%
1276   \expandafter\FV@FormattingPrep@PreHook\FV@FormattingPrep\FV@FormattingPrep@PostHook}

```

### \FV@PygmentsHook

This is a hook for turning on Pygments-related features for packages like `minted` and `pythontex` (section 12.13). It needs to be the first thing in `\FV@FormattingPrep@PreHook`, since it will potentially affect some of the later things in the hook. It is activated by `\VerbatimPygments`.

```

1277 \def\FV@PygmentsHook{}
1278 \g@addto@macro\FV@FormattingPrep@PreHook{\FV@PygmentsHook}

```

## 12.6 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.

```

1279 \edef\FV@hashchar{\string#}
1280 \edef\FV@dollarchar{\string$}
1281 \edef\FV@ampchar{\string&}
1282 \edef\FV@underscorechar{\string_}
1283 \edef\FV@caretchar{\string^}
1284 \edef\FV@tildechar{\string~}
1285 \edef\FV@leftsquarebracket{\string[]}
1286 \edef\FV@rightsquarebracket{\string]}
1287 \edef\FV@commachar{\string,}
1288 \newcommand{\FV@EscChars}{%
1289   \let#\FV@hashchar
1290   \let%\FV@percentchar
1291   \let{\FV@charlb
1292   \let}\FV@charrb
1293   \let$\FV@dollarchar
1294   \let&\FV@ampchar
1295   \let_\FV@underscorechar
1296   \let^\FV@caretchar
1297   \let\\\FV@backslashchar
1298   \let~\FV@tildechar
1299   \let~\FV@tildechar
1300   \let[\FV@leftsquarebracket
1301   \let]\FV@rightsquarebracket
1302   \let,\FV@commachar
1303 } %$ <- highlighting

```

## 12.7 Inline-only options

Create `\fvinline` for inline-only options. Note that this only applies to new or reimplemented inline commands that use `\FV@UseInlineKeyValues`.

### \FV@InlineKeyValues

```

1304 \def\FV@InlineKeyValues{}

```

```

\fvinlineset
1305 \def\fvinlineset#1{%
1306   \expandafter\def\expandafter\FV@InlineKeyValues\expandafter{%
1307     \FV@InlineKeyValues#1,}}
\FV@UseInlineKeyValues
1308 \def\FV@UseInlineKeyValues{%
1309   \expandafter\fvset\expandafter{\FV@InlineKeyValues}%
1310   \FV@ApplyBreakAnywhereInlineStretch}

```

## 12.8 Reimplementations

`fvextra` reimplements some `fancyvrb` internals. The patches in section 12.10 fix bugs, handle edge cases, and extend existing functionality in logical ways, while leaving default `fancyvrb` behavior largely unchanged. In contrast, reimplementations add features by changing existing behavior in significant ways. As a result, there is a boolean option `extra` that allows them to be disabled.

### 12.8.1 `extra` option

Boolean option that governs whether reimplemented commands and environments should be used, rather than the original definitions.

```

FV@extra
1311 \newbool{FV@extra}
extra
1312 \define@booleankey{FV}{extra}%
1313 {\booltrue{FV@extra}}%
1314 {\boolfalse{FV@extra}}
1315 \fvset{extra=true}

```

### 12.8.2 `\FancyVerbFormatInline`

This allows customization of inline verbatim material. It is the inline equivalent of `\FancyVerbFormatLine` and `\FancyVerbFormatText`.

```

\FancyVerbFormatInline
1316 \def\FancyVerbFormatInline#1{#1}

```

### 12.8.3 `\Verb`

`\Verb` is reimplemented so that it functions as well as possible when used within other commands.

`\verb` cannot be used inside other commands. The original `fancyvrb` implementation of `\Verb` does work inside other commands, but being inside other commands reduces its functionality since there is no attempt at retokenization. When used inside other commands, it essentially reduces to `\texttt`. `\Verb` also fails when the delimiting characters are active, since it assumes that the closing delimiting character will have catcode 12.

`fvextra`'s re-implemented `\Verb` uses `\scantokens` and careful consideration of catcodes to (mostly) remedy this. It also adds support for paired curly braces `{...}` as the delimiters for the verbatim argument, since this is often convenient



when `\Verb` is used within another command. The original `\Verb` implementation is completely incompatible with curly braces being used as delimiters, so this doesn't affect backward compatibility.

The re-implemented `\Verb` is constructed with `\FVExtraRobustCommand` so that it will function correctly after being in an expansion-only context, so long as the argument is delimited with curly braces.

`\Verb`

```
1317 \def\Verb{%
1318   \FVExtraRobustCommand\RobustVerb\FVExtraUnexpandedReadStar0ArgBVar}
```

`\RobustVerb`

```
1319 \protected\def\RobustVerb{\FV@Command}{\Verb}}
1320 \FVExtraPDFstringdefDisableCommands{%
1321   \def\RobustVerb{}}
```

`\FVC@Verb@FV`

Save the original fancyvrb definition of `\FVC@Verb`, so that the extra option can switch back to it.

```
1322 \let\FVC@Verb@FV\FVC@Verb
```

`\FVC@Verb`

Redefine `\FVC@Verb` so that it will adjust based on `extra`.

```
1323 \def\FVC@Verb{%
1324   \begingroup
1325   \FV@UseInlineKeyValues\FV@UseKeyValues
1326   \ifFV@extra
1327     \expandafter\endgroup\expandafter\FVC@Verb@Extra
1328   \else
1329     \expandafter\endgroup\expandafter\FVC@Verb@FV
1330   \fi}
```

`\FVC@Verb@Extra`

`fvextra` reimplementation of `\FVC@Verb`.

When used after expansion, there is a check for valid delimiters, curly braces. If incorrect delimiters are used, and there are no following curly braces, then the reader macro `\FVExtraUnexpandedReadStar0ArgBVar` will give an error about unmatched braces. However, if incorrect delimiters are used, and there *are* following braces in a subsequent command, then this error will be triggered, preventing interference with the following command by the reader macro.

```
1331 \def\FVC@Verb@Extra{%
1332   \ifbool{FVExtraRobustCommandExpanded}%
1333   {\@ifnextchar\bgroup
1334     {\FVC@Verb@Extra@i}%
1335     {\PackageError{fvextra}%
1336       {\string\Verb\space delimiters must be paired curly braces in this context}%
1337       {Use curly braces as delimiters}}}%
1338   {\FVC@Verb@Extra@i}}
```

`\FVC@Verb@Extra@i`

```
1339 \def\FVC@Verb@Extra@i{%
1340   \begingroup
1341   \ifbool{FV@varsingleline}%
1342   {\let\FV@Reader\FVExtraReadVArgSingleLine}%
```

```

1343   {\let\FV@Reader\FVExtraReadVArg}%
1344   \FV@Reader{%
1345     \FV@UseInlineKeyValues\FV@UseKeyValues\FV@FormattingPrep
1346     \FVExtraDetokenizeVArg{%
1347       \FVExtraRetokenizeVArg{\FVC@Verb@Extra@ii}{\FV@CatCodes}}}}

```

#### `\FVC@Verb@Extra@ii`

`breaklines` is only applied when there is no background color, since `\colorbox` prevents line breaks.

```

1348 \def\FVC@Verb@Extra@ii#1{%
1349   \ifx\FancyVerbBackgroundColor\relax
1350     \expandafter\@firstoftwo
1351   \else
1352     \expandafter\@secondoftwo
1353   \fi
1354   {\ifbool{FV@breaklines}%
1355     {\FV@InsertBreaks{\FancyVerbFormatInline}{#1}}%
1356     {\mbox{\FancyVerbFormatInline{#1}}}%
1357   \setlength{\FV@TmpLength}{\fboxsep}%
1358   \ifx\FancyVerbBackgroundColorPadding\relax
1359     \setlength{\fboxsep}{0pt}%
1360   \else
1361     \setlength{\fboxsep}{\FancyVerbBackgroundColorPadding}%
1362   \fi
1363   \colorbox{\FancyVerbBackgroundColor}{%
1364     \setlength{\fboxsep}{\FV@TmpLength}%
1365     \FancyVerbBackgroundColorVPhantom\FancyVerbFormatInline{#1}}%
1366   \endgroup}

```

#### 12.8.4 `\SaveVerb`

This is reimplemented, following `\Verb` as a template, so that both `\Verb` and `\SaveVerb` are using the same reading and tokenization macros. This also adds support for `\fvinlineset`. Since the definition in `fancyvrb` is

```
\def\SaveVerb{\FV@Command}{\SaveVerb}}
```

only the internal macros need to be reimplemented.

#### `\FVC@SaveVerb@FV`

```
1367 \let\FVC@SaveVerb@FV\FVC@SaveVerb
```

#### `\FVC@SaveVerb`

```

1368 \def\FVC@SaveVerb{%
1369   \begingroup
1370   \FV@UseInlineKeyValues\FV@UseKeyValues
1371   \ifFV@extra
1372     \expandafter\endgroup\expandafter\FVC@SaveVerb@Extra
1373   \else
1374     \expandafter\endgroup\expandafter\FVC@SaveVerb@FV
1375   \fi}

```

#### `\FVC@SaveVerb@Extra`

In addition to following the `\Verb` implementation, this saves a raw version of the text to allow `retokenize` with `\UseVerb`. The raw version is also used for conversion to a PDF string if that is needed.

```

1376 \def\FVC@SaveVerb@Extra#1{%
1377   \@namedef{FV@SV@#1}{}%
1378   \@namedef{FV@SVRaw@#1}{}%
1379   \begingroup
1380   \ifbool{FV@vargsingleline}%
1381     {\let\FV@Reader\FVExtraReadVArgSingleLine}%
1382     {\let\FV@Reader\FVExtraReadVArg}%
1383   \FV@Reader{%
1384     \FVC@SaveVerb@Extra@i{#1}}
\FVC@SaveVerb@Extra@i
1385 \def\FVC@SaveVerb@Extra@i#1#2{%
1386   \FV@UseInlineKeyValues\FV@UseKeyValues\FV@FormattingPrep
1387   \FVExtraDetokenizeVArg{%
1388     \FVExtraRetokenizeVArg{\FVC@SaveVerb@Extra@ii{#1}{#2}}{\FV@CatCodes}}{#2}}
\FVC@SaveVerb@Extra@ii
1389 \def\FVC@SaveVerb@Extra@ii#1#2#3{%
1390   \global\let\FV@AfterSave\FancyVerbAfterSave
1391   \endgroup
1392   \@namedef{FV@SV@#1}{#3}%
1393   \@namedef{FV@SVRaw@#1}{#2}%
1394   \FV@AfterSave}%

```

### 12.8.5 \UseVerb

This adds support for `\fvinline` and line breaking. It also adds movable argument and PDF string support. A new option `retokenize` is defined that determines whether the typeset output is based on the `commandchars` and `codes` in place when `\SaveVerb` was used (default), or is retokenized under current `commandchars` and `codes`.

**FV@retokenize**  
**retokenize**

Whether `\UseVerb` uses saved verbatim with its original tokenization, or retokenizes under current `commandchars` and `codes`.

```

1395 \newbool{FV@retokenize}
1396 \define@booleankey{FV}{retokenize}%
1397 {\booltrue{FV@retokenize}}{\boolfalse{FV@retokenize}}

```

**\UseVerb**

```

1398 \def\UseVerb{%
1399   \FVExtraRobustCommand\RobustUseVerb\FVExtraUseVerbUnexpandedReadStarOArgMArg}

```

**\RobustUseVerb**

```

1400 \protected\def\RobustUseVerb{\FV@Command}{\UseVerb}}
1401 \FVExtrapdfstringdefDisableCommands{%
1402   \def\RobustUseVerb{}}

```

**\FVC@UseVerb@FV**

```

1403 \let\FVC@UseVerb@FV\FVC@UseVerb

```

**\FVC@UseVerb**

```

1404 \def\FVC@UseVerb{%
1405   \begingroup

```

```

1406 \FV@UseInlineKeyValues\FV@UseKeyValues
1407 \ifFV@extra
1408   \expandafter\endgroup\expandafter\FVC@UseVerb@Extra
1409 \else
1410   \expandafter\endgroup\expandafter\FVC@UseVerb@FV
1411 \fi}

\FVC@UseVerb@Extra
1412 \def\FVC@UseVerb@Extra#1{%
1413 \@ifundefined{FV@SV@#1}%
1414   {\FV@Error{Short verbatim text never saved to name `#1'}\FV@eha}%
1415   {\begingroup
1416     \FV@UseInlineKeyValues\FV@UseKeyValues\FV@FormattingPrep
1417     \ifbool{FV@retokenize}%
1418       {\expandafter\let\expandafter\FV@Tmp\csname FV@SVRaw@#1\endcsname
1419         \expandafter\FV@UseVerb@Extra@Retok\expandafter{\FV@Tmp}}%
1420       {\expandafter\let\expandafter\FV@Tmp\csname FV@SV@#1\endcsname
1421         \expandafter\FV@UseVerb@Extra\expandafter{\FV@Tmp}}}}

\FV@UseVerb@Extra@Retok
1422 \def\FV@UseVerb@Extra@Retok#1{%
1423   \FVExtraDetokenizeVArg{%
1424     \FVExtraRetokenizeVArg{\FV@UseVerb@Extra}{\FV@CatCodes}}{#1}}

\FV@UseVerb@Extra
1425 \let\FV@UseVerb@Extra\FVC@Verb@Extra@ii

```

## 12.9 New commands and environments

### 12.9.1 \EscVerb

This is a variant of `\Verb` in which backslash escapes of the form `\<char>` are used for `<char>`. Backslash escapes are *only* permitted for printable, non-alphanumeric ASCII characters. The argument is read under a normal catcode regime, so any characters that cannot be read under normal catcodes must always be escaped, and the argument must always be delimited by curly braces. This ensures that `\EscVerb` behaves identically whether or not it is used inside another command.

`\EscVerb` is constructed with `\FVExtraRobustCommand` so that it will function correctly after being in an expansion-only context.

#### \EscVerb

Note that while the typeset mandatory argument will be read under normal catcodes, the reader macro for expansion is `\FVExtraUnexpandedReadStar0ArgBEscVArg`. This reflects how the argument will be typeset.

```

1426 \def\EscVerb{%
1427   \FVExtraRobustCommand\RobustEscVerb\FVExtraUnexpandedReadStar0ArgBEscVArg}

```

#### \RobustEscVerb

```

1428 \protected\def\RobustEscVerb{\FV@Command}{\EscVerb}}
1429 \FVExtrapdfstringdefDisableCommands{%
1430   \def\RobustEscVerb{}}

```

#### \FVC@EscVerb

Delimiting with curly braces is required, so that the command will always behave the same whether or not it has been through expansion.

```

1431 \def\FVC@EscVerb{%
1432   \@ifnextchar\bgroup
1433   {\FVC@EscVerb@i}%
1434   {\PackageError{fvextra}%
1435    {Invalid argument; argument must be delimited by paired curly braces}%
1436    {Delimit argument with curly braces}}}

```

`\FVC@EscVerb@i`

```

1437 \def\FVC@EscVerb@i#1{%
1438   \begingroup
1439   \FV@UseInlineKeyValues\FV@UseKeyValues\FV@FormattingPrep
1440   \FVExtraDetokenizeREscVArg{%
1441     \FVExtraRetokenizeVArg{\FVC@EscVerb@ii}{\FV@CatCodes}}{#1}}

```

`\FVC@EscVerb@ii`

```

1442 \let\FVC@EscVerb@ii\FVC@Verb@Extra@ii

```

### 12.9.2 VerbEnv

Environment variant of `\Verb`. Depending on how this is used in the future, it may be worth improving error message and error recovery functionality, using techniques from `fancyvrb`.

`\VerbEnv`

```

1443 \def\VerbEnv{%
1444   \ifcsname @currenvir\endcsname
1445   \ifx@currenvir\empty
1446     \PackageError{fvextra}{VerbEnv is an environment}{VerbEnv is an environment}%
1447   \else
1448     \ifx@currenvir\relax
1449       \PackageError{fvextra}{VerbEnv is an environment}{VerbEnv is an environment}%
1450     \fi
1451   \fi
1452 \else
1453   \PackageError{fvextra}{VerbEnv is an environment}{VerbEnv is an environment}%
1454 \fi
1455 \VerbatimEnvironment
1456 \FVExtraReadOArgBeforeVEnv{\expandafter\VerbEnv@i\expandafter{\FV@EnvironName}}}
1457 \def\VerbEnv@i#1#2{%
1458   \begingroup
1459   \let\do\@makeother\FVExtraDoSpecials
1460   \catcode`\ =\active
1461   \catcode`\^I=\active
1462   \catcode`\^M=\active
1463   \VerbEnv@ii{#1}{#2}}
1464 \begingroup
1465 \catcode`\!=0
1466 \catcode`\<=1
1467 \catcode`\>=2
1468 !catcode`\=12
1469 !catcode`\!=12
1470 !catcode`\!}=12
1471 !catcode`\^M=!active%
1472 !gdef!VerbEnv@ii#1#2#3^^M<%
1473   !endgroup%

```

```

1474 !def!VerbEnv@CheckLine##1\end{#1}##2!FV@Sentinel<%
1475 !if!relax!detokenize<##2>!relax%
1476 !else%
1477 !PackageError<fvextra><Missing environment contents><Missing environment contents>%
1478 !let!VerbEnv@iii!VerbEnv@iii@Error%
1479 !fi>%
1480 !VerbEnv@CheckLine#3\end{#1}!FV@Sentinel%
1481 !VerbEnv@iii<#1><#2><#3>>%
1482 !endgroup%
1483 \def\VerbEnv@iii@Error#1#2#3{
1484 \def\VerbEnv@iii#1#2#3{%
1485 \begingroup
1486 \let\do\@makeother\FVExtraDoSpecials
1487 \catcode`\ =10\relax
1488 \catcode`\^M=\active
1489 \VerbEnv@iv{#1}{#2}{#3}}
1490 \begingroup
1491 \catcode`\ !=0
1492 \catcode`\ <=1
1493 \catcode`\ >=2
1494 !catcode`\ !=12
1495 !catcode`\ {=12
1496 !catcode`\ }=12
1497 !catcode`\ ^M=\active%
1498 !gdef!VerbEnv@iv#1#2#3#4^M<%
1499 !endgroup%
1500 !def!VerbEnv@CheckEndDelim##1\end{#1}##2!FV@Sentinel<%
1501 !if!relax!detokenize<##2>!relax%
1502 !PackageError<fvextra><Missing end for environment !FV@EnvironName><Add environment en
1503 !let!VerbEnv@v!VerbEnv@v@Error%
1504 !else%
1505 !VerbEnv@CheckEndLeading##1!FV@Sentinel%
1506 !VerbEnv@CheckEndTrailing##2!FV@Sentinel%
1507 !fi>%
1508 !def!VerbEnv@CheckEndTrailing##1\end{#1}!FV@Sentinel<%
1509 !if!relax!detokenize<##1>!relax%
1510 !else%
1511 !PackageError<fvextra>%
1512 <Discarded text after end of environment !FV@EnvironName>%
1513 <Discarded text after end of environment !FV@EnvironName>%
1514 !let!VerbEnv@v!VerbEnv@v@Error%
1515 !fi>%
1516 !VerbEnv@CheckEndDelim#4\end{#1}!FV@Sentinel%
1517 !VerbEnv@v<#2><#3>>%
1518 !endgroup
1519 \def\VerbEnv@CheckEndLeading{%
1520 \FVExtra@ifnextcharAny\@sptoken%
1521 {\VerbEnv@CheckEndLeading@Continue}%
1522 {\ifx\@let\@token\FV@Sentinel
1523 \expandafter\VerbEnv@CheckEndLeading@End
1524 \else
1525 \expandafter\VerbEnv@CheckEndLeading@EndError
1526 \fi}}
1527 \def\VerbEnv@CheckEndLeading@Continue#1{%

```

```

1528 \VerbEnv@CheckEndLeading}
1529 \def\VerbEnv@CheckEndLeading@End#1\FV@Sentinel{}
1530 \def\VerbEnv@CheckEndLeading@EndError{%
1531 \PackageError{fvextra}%
1532 {Discarded text before end of environment \FV@EnvironName}%
1533 {Discarded text before end of environment \FV@EnvironName}%
1534 \let\VerbEnv@v\VerbEnv@v@Error}
1535 \def\VerbEnv@v@Error#1#2{}
1536 \def\VerbEnv@v#1#2{%
1537 \Verb[#1]{#2}%
1538 \expandafter\end\expandafter{\FV@EnvironName}}
\endVerbEnv
1539 \def\endVerbEnv{\global\let\FV@EnvironName\relax}

```

### 12.9.3 VerbatimWrite

This environment writes its contents to a file verbatim. Differences from fancyvrb's VerbatimOut:

- Multiple VerbatimWrite environments can write to the same file. The file is set via the writefilehandle option. This does mean that the user is responsible for creating a new file handle via \newwrite and then ideally invoking \closeout at the appropriate time.
- By default, text is really written verbatim. This is accomplished by a combination of setting catcodes to 12 (other) and \detokenize. This can be customized using the new writer option, which defines a macro that performs any processing on each line before writing it to file. By default, all fancyvrb options except for VerbatimWrite-specific options are ignored. This can be customized on a per-environment basis via environment optional arguments.

```

writefilehandle
\FancyVerbWriteFileHandle
    Set file handle for VerbatimWrite.
1540 \define@key{FV}{writefilehandle}{%
1541 \FV@SetWrite#1\FV@Sentinel}
1542 \def\FV@SetWrite#1#2\FV@Sentinel{%
1543 \let\FancyVerbWriteFileHandle\relax
1544 \if\relax\detokenize{#2}\relax
1545 \let\FancyVerbWriteFileHandle#1\relax
1546 \fi
1547 \ifx\FancyVerbWriteFileHandle\relax
1548 \PackageError{fvextra}%
1549 {Missing or invalid file handle for write}%
1550 {Need file handle from \string\newwrite}%
1551 \fi}
1552 \let\FancyVerbWriteFileHandle\relax

writer
\FV@Writer
    Define writer macro that processes each line before writing.
1553 \define@key{FV}{writer}{%

```

```

1554 \let\FV@Writer#1\relax}
1555 \def\FancyVerbDefaultWriter#1{%
1556 \immediate\write\FancyVerbWriteFileHandle{\detokenize{#1}}
1557 \fvset{writer=\FancyVerbDefaultWriter}

```

### VerbatimWrite

The environment implementation follows standard `fancyvrb` environment style.

A special write counter is used to track line numbers while avoiding incrementing the regular counter that is used for typeset code. Some macros do nothing with the default `writer`, but are needed to enable `fancyvrb` options when a custom `writer` is used in conjunction with optional environment arguments. These include `\FancyVerbDefineActive`, `\FancyVerbFormatCom`, and `\FV@DefineTabOut`.

```

1558 \newcounter{FancyVerbWriteLine}
1559 \def\VerbatimWrite{%
1560 \FV@Environment
1561 {codes=,commandchars=none,commentchar=none,defineactive,%
1562 gobble=0,formatcom=,firstline,lastline}%
1563 {VerbatimWrite}}
1564 \def\FVB@VerbatimWrite{%
1565 \@bsphack
1566 \begingroup
1567 \setcounter{FancyVerbWriteLine}{0}%
1568 \let\c@FancyVerbLine\c@FancyVerbWriteLine
1569 \FV@UseKeyValues
1570 \FV@DefineWhiteSpace
1571 \def\FV@Space{\space}%
1572 \FV@DefineTabOut
1573 \let\FV@ProcessLine\FV@Writer
1574 \let\FV@FontScanPrep\relax
1575 \let\@noligs\relax
1576 \FancyVerbDefineActive
1577 \FancyVerbFormatCom
1578 \FV@Scan}
1579 \def\FVE@VerbatimWrite{%
1580 \endgroup
1581 \@esphack}
1582 \def\endVerbatimWrite{\FVE@VerbatimWrite}

```

### 12.9.4 VerbatimBuffer

This environment stores its contents verbatim in a “buffer,” a sequence of numbered macros each of which contains one line of the environment. The “buffered” lines can then be looped over for further processing or later use.

By default, all `fancyvrb` options except for `VerbatimBuffer`-specific options are ignored. This can be customized on a per-environment basis via environment optional arguments.

```

afterbuffer
\FV@afterbuffer

```

Macro that is inserted after the last line of the environment is buffered, immediately before the environment ends.

```

1583 \define@key{FV}{afterbuffer}{%
1584 \def\FV@afterbuffer{#1}}
1585 \fvset{afterbuffer=}

```



### `\FancyVerbBufferIndex`

Current index in buffer during buffering. This is given a `FancyVerb*` macro name since it may be accessed by the user in defining custom `bufferer`.

```
1586 \def\FancyVerbBufferIndex{0}
```

### `bufferer`

#### `\FV@Bufferer`

#### `\FancyVerbDefaultBufferer`

This is the macro that adds lines to the buffer. The default is designed to create a truly verbatim buffer via `\detokenize`.

```
1587 \define@key{FV}{bufferer}{%
1588   \let\FV@Bufferer=#1\relax}
1589 \def\FancyVerbDefaultBufferer#1{%
1590   \expandafter\xdef\csname\FancyVerbBufferLineName\FancyVerbBufferIndex\endcsname{%
1591     \detokenize{#1}}
1592   \fvset{bufferer=\FancyVerbDefaultBufferer}
```

### `bufferlengthname`

#### `\FancyVerbBufferLengthName`

Name of macro storing the length of the buffer.

```
1593 \define@key{FV}{bufferlengthname}{%
1594   \ifcsname#1\endcsname
1595   \else
1596     \expandafter\xdef\csname#1\endcsname{0}%
1597   \fi
1598   \def\FancyVerbBufferLengthName{#1}%
1599   \expandafter\def\expandafter\FV@bufferlengthmacro\expandafter{%
1600     \csname#1\endcsname}}
1601   \fvset{bufferlengthname=FancyVerbBufferLength}
```

### `bufferlinename`

#### `\FancyVerbBufferLineName`

Base name of buffer line macros. This is given a `\FancyVerb*` macro name since it may be accessed by the user in defining custom `bufferer`.

```
1602 \define@key{FV}{bufferlinename}{%
1603   \def\FancyVerbBufferLineName{#1}}
1604 \fvset{bufferlinename=FancyVerbBufferLine}
```

### `buffername`

Shortcut for setting `bufferlengthname` and `bufferlinename`.

```
1605 \define@key{FV}{buffername}{%
1606   \fvset{bufferlengthname=#1length,bufferlinename=#1line}}
```

### `globalbuffer`

#### `FV@globalbuffer`

Whether buffer line macros and the buffer length macro are defined globally.

```
1607 \newbool{FV@globalbuffer}
1608 \define@booleankey{FV}{globalbuffer}%
1609   {\booltrue{FV@globalbuffer}}%
1610   {\boolfalse{FV@globalbuffer}}
1611 \fvset{globalbuffer=false}
```

### `VerbatimBuffer`

The environment implementation follows standard `fancyvrb` environment style.

A special buffer counter is used to track line numbers while avoiding incrementing the regular counter that is used for typeset code. Some macros do nothing with the default `bufferer`, but are needed to enable `fancyvrb` options when a custom `bufferer` is used in conjunction with optional environment arguments. These include `\FancyVerbDefineActive` and `\FancyVerbFormatCom`. Since counters are global, the exact location of the `\setcounter` commands at the end of the environment relative to `\begingroup... \endgroup` is not important.

```

1612 \newcounter{FancyVerbBufferLine}
1613 \def\FancyVerbBufferDepth{0}
1614 \def\VerbatimBuffer{%
1615   \FV@Environment
1616   {codes=,commandchars=none,commentchar=none,defineactive,%,
1617     gobble=0,formatcom=,firstline,lastline}%
1618   {VerbatimBuffer}}
1619 \def\FVB@VerbatimBuffer{%
1620   \@bsphack
1621   \xdef\FancyVerbBufferDepth{\the\numexpr\FancyVerbBufferDepth+1\relax}%
1622   \begingroup
1623   \FV@UseKeyValues
1624   \setcounter{FancyVerbBufferLine}{\FV@bufferlengthmacro}%
1625   \let\c@FancyVerbLine\c@FancyVerbBufferLine
1626   \xdef\FancyVerbBufferIndex{\FV@bufferlengthmacro}%
1627   \ifbool{FV@globalbuffer}%
1628     {}%
1629     {\expandafter\xdef\csname FV@setbufferlocalscopevars\FancyVerbBufferDepth\endcsname{%
1630       \unexpanded{\def\FV@oldbufferlength}%
1631         {\FV@bufferlengthmacro}}%
1632       \unexpanded{\def\FV@bufferlengthmacro}%
1633         {\unexpanded\expandafter{\FV@bufferlengthmacro}}}%
1634       \unexpanded{\def\FancyVerbBufferLineName}%
1635         {\unexpanded\expandafter{\FancyVerbBufferLineName}}}}%
1636   \ifx\FV@afterbuffer\@empty
1637   \else
1638     \ifx\FV@afterbuffer\relax
1639     \else
1640       \expandafter\global\expandafter
1641       \let\csname FV@afterbuffer\FancyVerbBufferDepth\endcsname\FV@afterbuffer
1642       \fi
1643     \fi
1644     \FV@DefineWhiteSpace
1645     \def\FV@ProcessLine{%
1646       \xdef\FancyVerbBufferIndex{\the\numexpr\FancyVerbBufferIndex+1\relax}%
1647       \FV@Bufferer}%
1648     \let\FV@FontScanPrep\relax
1649     \let\@noligs\relax
1650     \FancyVerbDefineActive
1651     \FancyVerbFormatCom
1652     \FV@Scan}
1653 \def\FVE@VerbatimBuffer{%
1654   \expandafter\xdef\FV@bufferlengthmacro{\FancyVerbBufferIndex}%
1655   \gdef\FancyVerbBufferIndex{0}%
1656   \endgroup
1657   \@esphack

```

```

1658 \ifcsname FV@afterbuffer\FancyVerbBufferDepth\endcsname
1659   \begingroup
1660   \csname FV@afterbuffer\FancyVerbBufferDepth\endcsname
1661   \endgroup
1662   \expandafter\global\expandafter
1663   \let\csname FV@afterbuffer\FancyVerbBufferDepth\endcsname\FV@Undefined
1664 \fi
1665 \ifcsname FV@setbufferlocalscopevars\FancyVerbBufferDepth\endcsname
1666   \begingroup
1667   \csname FV@setbufferlocalscopevars\FancyVerbBufferDepth\endcsname
1668   \loop\unless\ifnum\FV@bufferlengthmacro=\FV@oldbufferlength\relax
1669     \expandafter\global\expandafter
1670     \let\csname\FancyVerbBufferLineName\FV@bufferlengthmacro\endcsname\FV@Undefined
1671     \expandafter\xdef\FV@bufferlengthmacro{%
1672       \the\numexpr\FV@bufferlengthmacro-1\relax}%
1673   \repeat
1674   \endgroup
1675   \expandafter\global\expandafter
1676   \let\csname FV@setbufferlocalscopevars\FancyVerbBufferDepth\endcsname\FV@Undefined
1677 \fi
1678 \xdef\FancyVerbBufferDepth{\the\numexpr\FancyVerbBufferDepth-1\relax}}
1679 \def\endVerbatimBuffer{\FVE@VerbatimBuffer}

```

### 12.9.5 \VerbatimInsertBuffer

`\VerbatimInsertBuffer`  
`insertenvname`

This inserts an existing buffer created with `VerbatimBuffer` as a verbatim environment. By default, the inserted environment is `Verbatim`; this can be modified with the option `insertenvname` to any `Verbatim`- or `BVerbatim`-based environment, or any environment with a compatible implementation. The `Verbatim` and `BVerbatim` internals are customized to function with a buffer in a command context.

Notes on the implementation of `\VerbatimInsertBuffer@i`:

- The active `^^M` allows a verbatim environment to read optional arguments in the usual way, without requiring modifications to argument-reading macros.
- The `\begingroup\fvset{#1}\global\let\FV@CurrentVerbatimInsertEnvName...` is used to extract any `insertenvname` setting from optional arguments. Most optional arguments apply to the verbatim environment or to the customized verbatim internals that are invoked within it. However, `insertenvname` is needed earlier to determine which verbatim environment is in use. It is not possible simply to use `\fvset{#1}` before the verbatim environment to apply all settings, because that would conflict with the precedence of option processing in `\FV@Environment`.

```

1680 \define@key{FV}{insertenvname}{%
1681   \def\FV@VerbatimInsertEnvName{#1}}
1682 \fvset{insertenvname=Verbatim}
1683 \def\FV@Environment@InsertBuffer#1#2{%
1684   \def\FV@KeyValues{#1}%
1685   \FV@GetKeyValues{\@nameuse{FVB@#2}}

```

```

1686 \def\FV@Scan@InsertBuffer{%
1687   \FV@CatCodes
1688   \xdef\FV@EnvName{\FV@VerbatimInsertEnvName}%
1689   \ifnum\FV@bufferlengthmacro=\z@\relax
1690     \PackageError{fvextra}%
1691     {Buffer length macro \expandafter\string\FV@bufferlengthmacro\space
1692     is invalid or zero}%
1693     {}%
1694     \let\FV@GetLine\relax
1695   \fi
1696   \FV@BeginScanning}%
1697 \def\VerbatimInsertBuffer@def\FV@Line#1{%
1698   \FVExtraRetokenizeVArg{\def\FV@Line}{-}{#1}}
1699 \def\FancyVerbGetLine@VerbatimInsertBuffer{%
1700   \ifnum\FancyVerbBufferIndex>\FV@bufferlengthmacro\relax
1701     \global\let\FV@EnvName\relax
1702     \let\next\relax
1703   \else
1704     \ifcsname\FancyVerbBufferLineName\FancyVerbBufferIndex\endcsname
1705       \expandafter\let\expandafter\FV@Line@Buffer
1706       \csname\FancyVerbBufferLineName\FancyVerbBufferIndex\endcsname
1707       \expandafter\VerbatimInsertBuffer@def\FV@Line\expandafter{\FV@Line@Buffer}%
1708       \def\next{\FV@PreProcessLine\FV@GetLine}%
1709       \xdef\FancyVerbBufferIndex{\the\numexpr\FancyVerbBufferIndex+1\relax}%
1710     \else
1711       \def\next{%
1712         \PackageError{fvextra}%
1713         {Buffer with line macro named
1714         "\FancyVerbBufferLineName\FancyVerbBufferIndex" does not exist}%
1715         {Check bufferlinename, bufferlengthname, and globalbuffer settings}%
1716       }%
1717     \fi
1718   \fi
1719   \next}
1720 \newcommand{\VerbatimInsertBuffer}[1] [] {%
1721   \begingroup
1722   \let\FV@Scan\FV@Scan@InsertBuffer
1723   \let\FV@CheckScan\relax
1724   \let\FV@Environment\FV@Environment@InsertBuffer
1725   \let\FancyVerbGetLine\FancyVerbGetLine@VerbatimInsertBuffer
1726   \gdef\FancyVerbBufferIndex{1}%
1727   \VerbatimInsertBuffer@i{#1}%
1728   \gdef\FancyVerbBufferIndex{0}%
1729   \endgroup
1730   \@doendpe}
1731 \begingroup
1732 \catcode\^^M=\active
1733 \gdef\VerbatimInsertBuffer@i#1{%
1734   \begingroup%
1735   \fvset{#1}%
1736   \global\let\FV@CurrentVerbatimInsertEnvName\FV@VerbatimInsertEnvName%
1737   \endgroup%
1738   \csname\FV@CurrentVerbatimInsertEnvName\endcsname[#1]^^M%
1739   \csname end\FV@CurrentVerbatimInsertEnvName\endcsname%

```

```

1740 \global\let\FV@CurrentVerbatimInsertEnvName\FV@Undefined}%
1741 \endgroup

```

### 12.9.6 \VerbatimClearBuffer

#### \VerbatimClearBuffer

Clear an existing buffer.

```

1742 \newcommand{\VerbatimClearBuffer}[1] [] {%
1743 \begingroup
1744 \def\FV@KeyValues{#1}%
1745 \FV@UseKeyValues
1746 \xdef\FancyVerbBufferIndex{\FV@bufferlengthmacro}%
1747 \expandafter\xdef\FV@bufferlengthmacro{0}%
1748 \loop\unless\ifnum\FancyVerbBufferIndex<1\relax
1749 \expandafter\global\expandafter\let
1750 \csname\FancyVerbBufferLineName\FancyVerbBufferIndex\endcsname
1751 \FV@Undefined
1752 \xdef\FancyVerbBufferIndex{\the\numexpr\FancyVerbBufferIndex-1\relax}%
1753 \repeat
1754 \gdef\FancyVerbBufferIndex{0}%
1755 \endgroup}

```

### 12.9.7 \InsertBuffer

#### \InsertBuffer

wrapperenvname

wrapperenvopt

wrapperenvarg

This inserts an existing buffer created with `VerbatimBuffer` so that it is interpreted as  $\LaTeX$ . The result is essentially the same as if the buffered text had been included literally at the insertion point.

```

1756 \define@key{FV}{wrapperenvname}{%
1757 \def\FV@WrapperEnvName{#1}%
1758 \ifx\FV@WrapperEnvName\@empty
1759 \let\FV@WrapperEnvName\relax
1760 \fi}
1761 \fvset{wrapperenvname=}
1762 \define@key{FV}{wrapperenvopt}{%
1763 \def\FV@WrapperEnvOpt{#1}%
1764 \ifx\FV@WrapperEnvOpt\@empty
1765 \let\FV@WrapperEnvOpt\relax
1766 \fi}
1767 \fvset{wrapperenvopt=}
1768 \define@key{FV}{wrapperenvarg}{%
1769 \def\FV@WrapperEnvArg{#1}%
1770 \ifx\FV@WrapperEnvArg\@empty
1771 \let\FV@WrapperEnvArg\relax
1772 \fi}
1773 \fvset{wrapperenvarg=}
1774 \newcommand{\InsertBuffer}[1] [] {%
1775 \begingroup
1776 \def\FV@KeyValues{#1}%
1777 \FV@UseKeyValues

```

```

1778 \ifnum\FV@bufferlengthmacro<1
1779   \expandafter\endgroup\expandafter@gobble
1780 \else
1781   \expandafter@firstofone
1782 \fi
1783 \InsertBuffer@i}
1784 \def\InsertBuffer@i{%
1785   \InsertBuffer@expandbuffer
1786   \expandafter\endgroup\expandafter\scantokens\expandafter{%
1787     \FV@expandedbuffer\noexpand}\relax}
1788 \def\InsertBuffer@expandbuffer{%
1789   \edef\FV@expandedbuffer{%
1790     \ifx\FV@WrapperEnvName\relax
1791       \else
1792         \unexpanded\expandafter\expandafter\expandafter{%
1793           \expandafter\string\expandafter\begin\expandafter{\FV@WrapperEnvName}}%
1794         \ifx\FV@WrapperEnvOpt\relax
1795           \else
1796             \unexpanded\expandafter{\expandafter[\FV@WrapperEnvOpt]}%
1797           \fi
1798         \ifx\FV@WrapperEnvArg\relax
1799           \else
1800             \unexpanded\expandafter{\expandafter{\FV@WrapperEnvArg}}%
1801           \fi
1802         \unexpanded{^^J}%
1803       \fi
1804       \InsertBuffer@expandbufferlines{1}%
1805       \ifx\FV@WrapperEnvName\relax
1806         \unskip
1807       \else
1808         \unexpanded\expandafter\expandafter\expandafter{%
1809           \expandafter\string\expandafter\end\expandafter{\FV@WrapperEnvName}^^J}%
1810       \fi}}
1811 \def\InsertBuffer@expandbufferlines#1{%
1812   \unexpanded\expandafter\expandafter\expandafter{%
1813     \csname\FancyVerbBufferLineName#1\endcsname^^J}%
1814   \ifnum\FV@bufferlengthmacro=#1
1815     \expandafter@gobble
1816   \else
1817     \expandafter@firstofone
1818   \fi
1819   {\expandafter\InsertBuffer@expandbufferlines\expandafter{\the\numexpr#1+1\relax}}}

```

### 12.9.8 \ClearBuffer

#### \ClearBuffer

Clear an existing buffer. Alias for \VerbatimClearBuffer.

```
1820 \let\ClearBuffer\VerbatimClearBuffer
```

### 12.9.9 \BufferMdfivesum

#### \BufferMdfivesum

Calculate the MD5 sum of the current buffer.

```

1821 \def\BufferMdfivesum{%
1822   \pdf@mdfivesum{%
1823     \ifnum\FV@bufferlengthmacro<1
1824       \expandafter\@gobble
1825     \else
1826       \expandafter\@firstofone
1827     \fi
1828     {\BufferMdfivesum@i{1}}}}
1829 \def\BufferMdfivesum@i#1{%
1830   \csname\FancyVerbBufferLineName#1\endcsname^^J%
1831   \ifnum\FV@bufferlengthmacro=#1
1832     \expandafter\@gobble
1833   \else
1834     \expandafter\@firstofone
1835   \fi
1836   {\expandafter\BufferMdfivesum@i\expandafter{\the\numexpr#1+1\relax}}}

```

### 12.9.10 \IterateBuffer, \IterateBufferBreak

`\IterateBuffer`

`\IterateBufferBreak`

Loop over buffer, applying a macro to each line.

```

1837 \newcommand{\IterateBuffer}[2][\relax]{%
1838   \if\relax\detokenize{#1}\relax
1839   \else
1840     \let\FancyVerbBufferLengthName@beforeiter\FancyVerbBufferLengthName
1841     \let\FV@bufferlengthmacro@beforeiter\FV@bufferlengthmacro
1842     \let\FancyVerbBufferLineName@beforeiter\FancyVerbBufferLineName
1843     \begingroup
1844     \def\FV@KeyValues{#1}%
1845     \FV@UseKeyValues
1846     \global\let\FancyVerbBufferLengthName@iter\FancyVerbBufferLengthName
1847     \global\let\FV@bufferlengthmacro@iter\FV@bufferlengthmacro
1848     \global\let\FancyVerbBufferLineName@iter\FancyVerbBufferLineName
1849     \endgroup
1850     \let\FancyVerbBufferLengthName\FancyVerbBufferLengthName@iter
1851     \let\FV@bufferlengthmacro\FV@bufferlengthmacro@iter
1852     \let\FancyVerbBufferLineName\FancyVerbBufferLineName@iter
1853     \global\let\FancyVerbBufferLengthName@iter\FV@Undefined
1854     \global\let\FV@bufferlengthmacro@iter\FV@Undefined
1855     \global\let\FancyVerbBufferLineName@iter\FV@Undefined
1856   \fi
1857   \gdef\FancyVerbBufferIndex{1}%
1858   \def\FV@IterateBuffer@cmd{#2}%
1859   \def\IterateBufferBreak{\xdef\FancyVerbBufferIndex{\FV@bufferlengthmacro}}%
1860   \loop\unless\ifnum\FancyVerbBufferIndex>\FV@bufferlengthmacro\relax
1861     \expandafter\let\expandafter\FV@IterateBuffer@line
1862     \csname\FancyVerbBufferLineName\FancyVerbBufferIndex\endcsname
1863     \expandafter\FV@IterateBuffer@cmd\expandafter{\FV@IterateBuffer@line}%
1864     \xdef\FancyVerbBufferIndex{\the\numexpr\FancyVerbBufferIndex+1\relax}%
1865   \repeat
1866   \gdef\FancyVerbBufferIndex{0}%
1867   \let\FV@IterateBuffer@cmd\FV@Undefined
1868   \let\IterateBufferBreak\FV@Undefined

```

```

1869 \let\FV@IterateBuffer@line\FV@Undefined
1870 \if\relax\detokenize{#1}\relax
1871 \else
1872   \let\FancyVerbBufferLengthName\FancyVerbBufferLengthName@beforeiter
1873   \let\FV@bufferlengthmacro\FV@bufferlengthmacro@beforeiter
1874   \let\FancyVerbBufferLineName\FancyVerbBufferLineName@beforeiter
1875   \let\FancyVerbBufferLengthName@beforeiter\FV@Undefined
1876   \let\FV@bufferlengthmacro@beforeiter\FV@Undefined
1877   \let\FancyVerbBufferLineName@beforeiter\FV@Undefined
1878 \fi}

```

## \WriteBuffer

Buffer equivalent of VerbatimWrite.

```

1879 \def\WriteBuffer{%
1880   \FV@Command}{\WriteBuffer}}
1881 \def\FVC@WriteBuffer{%
1882   \@bsphack
1883   \begingroup
1884   \setcounter{FancyVerbWriteLine}{0}%
1885   \let\c@FancyVerbLine\c@FancyVerbWriteLine
1886   \FV@UseKeyValues
1887   \FV@DefineWhiteSpace
1888   \def\FV@Space{\space}%
1889   \FV@DefineTabOut
1890   \gdef\FancyVerbBufferIndex{1}%
1891   \loop\unless\ifnum\FancyVerbBufferIndex>\FV@bufferlengthmacro\relax
1892     \stepcounter{FancyVerbWriteLine}%
1893     \expandafter\let\expandafter\FV@WriteBuffer@line
1894     \csname\FancyVerbBufferLineName\FancyVerbBufferIndex\endcsname
1895     \expandafter\FV@Writer\expandafter{\FV@WriteBuffer@line}%
1896     \xdef\FancyVerbBufferIndex{\the\numexpr\FancyVerbBufferIndex+1\relax}%
1897   \repeat
1898   \gdef\FancyVerbBufferIndex{0}%
1899   \endgroup
1900   \@esphack}

```

## 12.10 Patches

### 12.10.1 Delimiting characters for verbatim commands

Unlike `\verb`, `fancyvrb`'s commands like `\Verb` cannot take arguments delimited by characters like `#` and `%` due to the way that starred commands and optional arguments are implemented. The relevant macros are redefined to make this possible.

`fancyvrb`'s `\Verb` is actually implemented in `\FVC@Verb`. This is invoked by a helper macro `\FV@Command` which allows versions of commands with customized options:

$$\FV@Command{\langle customized\_options \rangle}{\langle base\_command\_name \rangle}$$

`\Verb` is then defined as `\def\Verb{\FV@Command}{\Verb}}`. The definition of `\FV@Command` (and `\FVC@Command` which it uses internally) involves looking ahead for a star `*` (`\@ifstar`) and for a left square bracket `[` that delimits an optional argument (`\@ifnextchar`). As a result, the next character is tokenized under



the current, normal catcode regime. This prevents `\Verb` from being able to use delimiting characters like `#` and `%` that work with `\verb`.

`\FV@Command` and `\FV@@Command` are redefined so that this lookahead tokenizes under a typical verbatim catcode regime (with one exception that is explained below). This enables `\verb`-style delimiters. This does not account for any custom catcode changes introduced by `\fvset`, customized commands, or optional arguments. However, delimiting characters should never need custom catcodes, and both the `fancyvrb` definition of `\Verb` (when not used inside another macro) as well as the `fvextra` reimplementation (in all cases) handle the possibility of delimiters with valid but non-typical catcodes. Other, non-verbatim commands that use `\FV@Command`, such as `\UseVerb`, are not affected by the patch.

The catcode regime for lookahead has one exception to a typical verbatim catcode regime: The curly braces `{}` retain their normal codes. This allows the `fvextra` reimplementation of `\Verb` to use a pair of curly braces as delimiters, which can be convenient when `\Verb` is used within another command. Since the original `fancyvrb` implementation of `\Verb` with unpatched `\FV@Command` is incompatible with curly braces being used as delimiters in any form, this does not affect any pre-existing `fancyvrb` functionality.

#### `\FV@Command`

```
1901 \def\FV@Command#1#2{%
1902   \FVExtra@ifstarVArg
1903   {\def\FV@KeyValues{#1,showspaces,showtabs}\FV@@Command{#2}}%
1904   {\def\FV@KeyValues{#1}\FV@@Command{#2}}}
```

#### `\FV@@Command`

```
1905 \def\FV@@Command#1{%
1906   \FVExtra@ifnextcharVArg[%
1907     {\FV@GetKeyValues{\@nameuse{FVC@#1}}}%
1908     {\@nameuse{FVC@#1}}}
```

### 12.10.2 `\CustomVerbatimCommand` compatibility with `\FVExtraRobustCommand`

#### `\@CustomVerbatimCommand`

`#1` is `\newcommand` or `\renewcommand`, `#2` is the (re)new command, `#3` is the base `fancyvrb` command, `#4` is options.

```
1909 \def\@CustomVerbatimCommand#1#2#3#4{%
1910   \begingroup\fvset{#4}\endgroup
1911   \@ifundefined{FVC@#3}%
1912     {\FV@Error{Command `string#3' is not a FancyVerb command.}\@eha}%
1913     {\ifcsname Robust#3\endcsname
1914       \expandafter\@firstoftwo
1915       \else
1916         \expandafter\@secondoftwo
1917       \fi
1918     {\expandafter\let\expandafter\@tempa\csname #3\endcsname
1919       \def\@tempb##1##2##3{%
1920         \expandafter\def\expandafter\@tempc\expandafter{%
1921           \csname Robust\expandafter@gobble\string#2\endcsname}%
1922         \def\@tempd####1{%
1923           #1{#2}{##1####1##3}}%
1924         \expandafter\@tempd\@tempc
1925         \expandafter\protected\expandafter\def\@tempc{\FV@Command{#4}{#3}}%}
```

```

1926 \expandafter\@tempb\@tempa}%
1927  {#1{#2}{\FV@Command{#4}{#3}}}}

```

### 12.10.3 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 }}
```

`\textvisiblespace` is not an alternative because it does not have the correct width. The redefinition follows <https://tex.stackexchange.com/a/120231/10742>.

```

1928 \def\FancyVerbSpace{%
1929 \makebox[0.5em]{%
1930 \kern.07em
1931 \vrule height.3ex
1932 \hrulefill
1933 \vrule height.3ex
1934 \kern.07em}}

```

### 12.10.4 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 `\zsaveposx` 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 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.

```
1935 \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{...}`. The `\FV@TmpCurrentGroupLevel` is needed for compatibility with the `calc` package, which redefines `\setcounter`.

```
1936 \def\FV@@ObeyTabs#1{%
1937   \let\FV@Space@Orig\FV@Space
1938   \let\FV@Space\FV@Space@ObeyTabs
1939   \edef\FV@TmpCurrentGroupLevel{\the\currentgrouplevel}%
1940   \setcounter{FV@TrueTabGroupLevel}{\FV@TmpCurrentGroupLevel}%
1941   \addtocounter{FV@TrueTabGroupLevel}{1}%
1942   \setbox\FV@TabBox=\hbox{#1}\box\FV@TabBox
1943   \let\FV@Space\FV@Space@Orig}
```

#### `\FV@TrueTab`

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

```
1944 \def\FV@TrueTab{%
1945   \ifnum\value{FV@TrueTabGroupLevel}=\the\currentgrouplevel\relax
1946     \expandafter\FV@TrueTab@NoGroup
1947   \else
1948     \expandafter\FV@TrueTab@Group
1949   \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.

```
1950 \let\FV@TrueTabSaveWidth\relax
```

### FV@TrueTabCounter

Counter for tracking saved tabs.

```
1951 \newcounter{FV@TrueTabCounter}
```

### \FV@TrueTabSaveWidth@Save

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

```
1952 \def\FV@TrueTabSaveWidth@Save{%
1953   \expandafter\xdef\csname FV@TrueTab:Width\arabic{FV@TrueTabCounter}\endcsname{%
1954     \number\@tempdima}%
1955   \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.

```
1956 \def\FV@TrueTab@NoGroup{%
1957   \egroup
1958   \@tempdima=\FV@ObeyTabSize sp\relax
1959   \@tempcnta=\wd\FV@TabBox
1960   \advance\@tempcnta\FV@@ObeyTabSize\relax
1961   \divide\@tempcnta\@tempdima
1962   \multiply\@tempdima\@tempcnta
1963   \advance\@tempdima-\wd\FV@TabBox
1964   \FV@TrueTabSaveWidth
1965   \setbox\FV@TabBox=\hbox\bgroup
1966   \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.

```
1967 \newbool{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@FVTabToken and \FV@FVSpaceToken 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.

```
1968 \def\FV@TrueTab@Group{%
1969   \booltrue{FV@ObeyTabs@Whitespace@Tab}%
1970   \gdef\FV@TmpWhitespace{\FV@FVTabToken}%
1971   \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.

```
1972 \def\FV@Space@ObeyTabs{%
1973   \ifnum\value{FV@TrueTabGroupLevel}=\the\currentgrouplevel\relax
1974     \expandafter\FV@Space@ObeyTabs@NoGroup
1975   \else
1976     \expandafter\FV@Space@ObeyTabs@Group
1977   \fi}
```

```

\FV@Space@ObeyTabs@NoGroup
    Fall back to normal space.
1978 \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.
1979 \def\FV@Space@ObeyTabs@Group{%
1980   \boolfalse\FV@ObeyTabs@Whitespace@Tab}%
1981   \gdef\FV@TmpWhitespace{\FV@FVSpaceToken}%
1982   \FV@ObeyTabs@ScanWhitespace}

\FV@ObeyTabs@ScanWhitespace
    Collect whitespace until the end of the run, then process it. Proper lookahead
    comparison requires \FV@FVSpaceToken and \FV@FVTabToken.
1983 \def\FV@ObeyTabs@ScanWhitespace{%
1984   \@ifnextchar\FV@FVSpaceToken%
1985   {\FV@TrueTab@CaptureWhitespace@Space}%
1986   {\ifx\@let@token\FV@FVTabToken
1987     \expandafter\FV@TrueTab@CaptureWhitespace@Tab
1988     \else
1989     \expandafter\FV@ObeyTabs@ResolveWhitespace
1990     \fi}}
1991 \def\FV@TrueTab@CaptureWhitespace@Space#1{%
1992   \g@addto@macro\FV@TmpWhitespace{\FV@FVSpaceToken}%
1993   \FV@ObeyTabs@ScanWhitespace}
1994 \def\FV@TrueTab@CaptureWhitespace@Tab#1{%
1995   \booltrue\FV@ObeyTabs@Whitespace@Tab}%
1996   \g@addto@macro\FV@TmpWhitespace{\FV@FVTabToken}%
1997   \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.
1998 \newbox\FV@TabBox@Group
1999 \def\FV@TrueTab@Group@Expand{%
2000   \egroup
2001   \@tempdima=\FV@ObeyTabSize sp\relax
2002   \@tempcnta=\wd\FV@TabBox@Group
2003   \advance\@tempcnta\FV@ObeyTabSize\relax
2004   \divide\@tempcnta\@tempdima
2005   \multiply\@tempdima\@tempcnta
2006   \advance\@tempdima-\wd\FV@TabBox@Group
2007   \FV@TrueTabSaveWidth
2008   \setbox\FV@TabBox@Group=\hbox\bgroup
2009   \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.
2010 \def\FV@ObeyTabs@ResolveWhitespace{%
2011   \let\FV@Space\FV@Space@Orig
2012   \let\FV@Tab\FV@TrueTab@Group@Expand

```

```

2013 \expandafter\FV@ObeyTabs@ResolveWhitespace@i\expandafter{\FV@TmpWhitespace}%
2014 \let\FV@Space\FV@Space@ObeyTabs
2015 \let\FV@Tab\FV@TrueTab}
2016 \def\FV@ObeyTabs@ResolveWhitespace@i#1{%
2017 \ifbool{FV@ObeyTabs@Whitespace@Tab}%
2018   {\setbox\FV@TabBox@Group=\hbox{#1}\box\FV@TabBox@Group}%
2019   {#1}}

```

### 12.10.5 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`.

```

2020 \begingroup
2021 \catcode'\=' \active
2022 \catcode'\^=7
2023 \gdef\FV@pr@m@s{%
2024 \ifx'\@let@token
2025   \expandafter\pr@@@s
2026 \else
2027   \ifx^\@let@token
2028     \expandafter\expandafter\expandafter\pr@@@t
2029   \else
2030     \egroup
2031   \fi
2032 \fi}
2033 \endgroup

```

#### `\FV@SetupMathFont`

Set the font back to default from the verbatim font.

```

2034 \def\FV@SetupMathFont{%
2035 \everymath\expandafter{\the\everymath\fontfamily{\familydefault}\selectfont}}
2036 \g@addto@macro\FV@FormattingPrep@PreHook{\FV@SetupMathFont}

```

#### `\FV@SetupMathActive`

Make all characters in `\FVExtraDoSpecials` behave more normally within math. This is for `mathescape` or when `$` has its normal catcode via `codes`. This does not retokenize, so it can only provide a better approximation of regular math.

At minimum, the characters in `\@noligs` will typically be made `\active` within verbatim contexts to produce literal versions of themselves. The relevant definition from `latex.ltx`:

```
\def\verbatimnolig@list{\do\` \do\<\do\>\do\, \do\' \do\ -}
```

It is also possible that additional characters will be made `\active` to produce literal versions of themselves or for other purposes. These `\active` characters can interfere with math.

The strategy here has three components:

- For a character in `\FVExtraDoSpecials` that is active before the verbatim context begins, save its definition. Then restore this within math via `\everymath`.
- For a character in `\FVExtraDoSpecials` that is not active, but does have a `\mathcode` of "8000, save the active definition. Then via `\everymath` define the active version of this character to give the saved active definition within math mode and a detokenized character otherwise.
- For a character in `\FVExtraDoSpecials` that is not active and does not have a `\mathcode` of "8000, use `\everymath` to define the active version of the character to give a detokenized character.

Also switch to `\FV@pr@m@s` so that math prime ' behaves as expected.

```

2037 \def\FV@SetupMathActive{%
2038   \def\do##1{%
2039     \ifnum\catcode`##1=\active
2040       \expandafter\expandafter\expandafter\let\expandafter\expandafter\expandafter\FV@Tmp
2041         \csname FV@Char@CatActive:\number`##1\endcsname
2042       \expandafter\let
2043         \csname FV@Char@Math:\number`##1\endcsname\FV@Tmp
2044     \else
2045       \ifnum\mathcode`##1="8000
2046         \expandafter\expandafter\expandafter\let\expandafter\expandafter\expandafter\FV@Tmp
2047           \csname FV@Char@CatActive:\number`##1\endcsname
2048         \expandafter\let
2049           \csname FV@Char@MathActive:\number`##1\endcsname\FV@Tmp
2050         \expandafter
2051           \def\csname FV@Char@Math:\number`##1\endcsname{%
2052             \expandafter\ifmode\expandafter\expandafter
2053               \csname FV@Char@MathActive:\number`##1\endcsname
2054             \else
2055               \expandafter\expandafter
2056                 \csname FV@Char@CatDetok:\number`##1\endcsname
2057             \fi}%
2058         \else
2059           \expandafter\let\expandafter\FV@Tmp
2060             \csname FV@Char@CatDetok:\number`##1\endcsname
2061           \expandafter\let
2062             \csname FV@Char@Math:\number`##1\endcsname\FV@Tmp
2063         \fi
2064       \fi}%
2065 \FVExtraDoSpecials
2066 \everymath\expandafter{%
2067   \the\everymath
2068   \FV@ApplyMathActive}}
2069 \def\FV@ApplyMathActive{%
2070   \let\pr@m@s\FV@pr@m@s
2071   \def\do##1{%
2072     \expandafter\let\expandafter\FV@Tmp
2073       \csname FV@Char@CatActive:\number`##1\endcsname
2074     \expandafter\expandafter\expandafter\let\expandafter\FV@Tmp
2075       \csname FV@Char@Math:\number`##1\endcsname}%
2076 \FVExtraDoSpecials}

```

```
2077 \FV@AddToHook\FV@UseKeyValues@Hook{\FV@SetupMathActive}
```

### 12.10.6 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>.

```
2078 \def\FV@BeginListFrame@Lines{%
2079   \begingroup
2080   \lineskip\z@skip
2081   \FV@SingleFrameLine{\z@}%
2082   \kern-0.5\baselineskip\relax
2083   \baselineskip\z@skip
2084   \kern\FV@FrameSep\relax
2085   \penalty\@M
2086   \endgroup}
```

### 12.10.7 `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`

```
2087 \define@key{FV}{rulecolor}{%
2088   \ifstrepty{#1}%
2089   {\let\FancyVerbRuleColor\relax}%
2090   {\ifstrequal{#1}{none}%
2091     {\let\FancyVerbRuleColor\relax}%
2092     {\def\@tempa{#1}%
2093       \FV@KVProcess@RuleColor#1\FV@Undefined}}}
2094 \def\FV@KVProcess@RuleColor#1#2\FV@Undefined{%
2095   \ifx#1\color
2096   \else
2097     \expandafter\def\expandafter\@tempa\expandafter{%
2098       \expandafter\color\expandafter{\@tempa}}%
2099   \fi
2100   \let\FancyVerbRuleColor\@tempa}
2101 \fvset{rulecolor=none}
```

#### `fillcolor`

```
2102 \define@key{FV}{fillcolor}{%
2103   \ifstrepty{#1}%
2104   {\let\FancyVerbFillColor\relax}%
2105   {\ifstrequal{#1}{none}%
2106     {\let\FancyVerbFillColor\relax}%
2107     {\def\@tempa{#1}%
2108       \FV@KVProcess@FillColor#1\FV@Undefined}}}
2109 \def\FV@KVProcess@FillColor#1#2\FV@Undefined{%
2110   \ifx#1\color
2111   \else
2112     \expandafter\def\expandafter\@tempa\expandafter{%
2113       \expandafter\color\expandafter{\@tempa}}%
2114   \fi}
```



```

2114 \fi
2115 \let\FancyVerbFillColor\@tempa}
2116 \fvset{fillcolor=none}

```

## 12.11 Extensions

### 12.11.1 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`.

```

2117 \define@booleankey{FV}{linenos}%
2118   {\@nameuse{FV@Numbers@left}}{\@nameuse{FV@Numbers@none}}

```

#### tab

Redefine `\FancyVerbTab`.

```

2119 \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`.

```

2120 \define@key{FV}{tabcolor}%
2121   {\ifstrepty{#1}%
2122     {\let\FV@TabColor\relax}%
2123     {\ifstrequal{#1}{none}%
2124       {\let\FV@TabColor\relax}%
2125       {\def\FV@TabColor{\textcolor{#1}}}}}
2126 \define@booleankey{FV}{showtabs}%
2127   {\def\FV@TabChar{\FV@TabColor{\FancyVerbTab}}}%
2128   {\let\FV@TabChar\relax}
2129 \fvset{tabcolor=none, showtabs=false}

```

#### showspaces

##### FV@showspaces

Reimplement `showspaces` with a `bool` to work with new space options.

```

2130 \newbool{FV@showspaces}
2131 \define@booleankey{FV}{showspaces}%
2132   {\booltrue{FV@showspaces}}%
2133   {\boolfalse{FV@showspaces}}
2134 \fvset{showspaces=false}

```

#### space

Redefine `\FancyVerbSpace`, which is the visible space.

```

2135 \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`.

```

2136 \define@key{FV}{spacecolor}%
2137   {\ifstrepty{#1}%
2138     {\let\FV@SpaceColor\relax}%
2139     {\ifstrequal{#1}{none}%
2140       {\let\FV@SpaceColor\relax}%
2141       {\def\FV@SpaceColor{\textcolor{#1}}}}}
2142 \fvset{spacecolor=none}

```

spacebreak

`\FancyVerbSpaceBreak`

Line break for spaces that is inserted when spaces are visible (`showspaces=true`) or when breaks around spaces are handled specially (`breakcollapsespaces=false`). Not used for regular spaces under default conditions.

```
2143 \define@key{FV}{spacebreak}{%
2144   \def\FancyVerbSpaceBreak{#1}}
2145 \fvset{spacebreak=\discretionary{}{}{}}
```

breakcollapsespaces

`FV@breakcollapsespaces`

When a line break occurs within a sequence of regular space characters (`showspaces=false`), collapse the spaces into a single space and then replace it with the break. When this is `true`, a sequence of spaces will cause at most a single line break, and the first character on the wrapped line after the break will be a non-space character. When this is `false`, a sequence of spaces may result in multiple line breaks. Each wrapped line besides the last will contain only spaces. The final wrapped line may contain leading spaces before any non-space character(s).

```
2146 \newbool{FV@breakcollapsespaces}
2147 \define@booleankey{FV}{breakcollapsespaces}%
2148   {\booltrue{FV@breakcollapsespaces}}%
2149   {\boolfalse{FV@breakcollapsespaces}}%
2150 \fvset{breakcollapsespaces=true}
```

`\FV@DefFVSpace`

Redefine `\FV@Space` based on `fvextra` options that affect spaces.

This must be added to `\FV@FormattingPrep@PreHook`, but only after `breakbefore` and `breakafter` macros are defined. Hence the `\AtEndOfPackage`.

```
2151 \def\FV@DefFVSpace{%
2152   \ifbool{FV@showspaces}%
2153     {\ifbool{FV@breaklines}%
2154       {\ifcsname FV@BreakBefore@Token\FV@SpaceCatTen\endcsname
2155         \def\FV@Space{\FV@SpaceColor{\FancyVerbSpace}}%
2156       \else\ifcsname FV@BreakAfter@Token\FV@SpaceCatTen\endcsname
2157         \def\FV@Space{\FV@SpaceColor{\FancyVerbSpace}}%
2158       \else
2159         \def\FV@Space{\FV@SpaceColor{\FancyVerbSpace}\FancyVerbSpaceBreak}%
2160       \fi\fi}%
2161     {\def\FV@Space{\FV@SpaceColor{\FancyVerbSpace}}}%
2162   {\ifbool{FV@breaklines}%
2163     {\ifcsname FV@BreakBefore@Token\FV@SpaceCatTen\endcsname
2164       \def\FV@Space{\mbox{\FV@SpaceCatTen}}%
2165     \else\ifcsname FV@BreakAfter@Token\FV@SpaceCatTen\endcsname
2166       \def\FV@Space{\mbox{\FV@SpaceCatTen}}%
2167     \else
2168       \ifbool{FV@breakcollapsespaces}%
2169         {\def\FV@Space{\FV@SpaceCatTen}}%
2170         {\def\FV@Space{\mbox{\FV@SpaceCatTen}\FancyVerbSpaceBreak}}%
2171       \fi\fi}%
2172     {\def\FV@Space{\FV@SpaceCatTen}}}%
2173 \AtEndOfPackage{%
2174   \g@addto@macro\FV@FormattingPrep@PreHook{\FV@DefFVSpace}}
```

### mathescape

Give \$, &, ^, and \_ their normal catcodes to allow normal typeset math.

```
2175 \define@booleankey{FV}{mathescape}%
2176 {\let\FancyVerbMathEscape\FV@MathEscape}%
2177 {\let\FancyVerbMathEscape\relax}
2178 \def\FV@MathEscape{\catcode`\$=3\catcode`\&=4\catcode`\^=7\catcode`\_ =8\relax}
2179 \FV@AddToHook\FV@CatCodesHook\FancyVerbMathEscape
2180 \fvset{mathescape=false}
```

### beameroverlays

Give < and > their normal catcodes (not \active), so that beamer overlays will work. This modifies \@noligs because that is the only way to prevent the settings from being overwritten later. This could have used \FV@CatCodesHook, but then it would have had to compare \@noligs to \relax to avoid issues when \let\@noligs\relax in VerbatimOut.

```
2181 \define@booleankey{FV}{beameroverlays}%
2182 {\let\FancyVerbBeamerOverlays\FV@BeamerOverlays}%
2183 {\let\FancyVerbBeamerOverlays\relax}
2184 \def\FV@BeamerOverlays{%
2185   \expandafter\def\expandafter\@noligs\expandafter{\@noligs
2186     \catcode`\<=12\catcode`\>=12\relax}}
2187 \FV@AddToHook\FV@FormattingPrep@PreHook\FancyVerbBeamerOverlays
2188 \fvset{beameroverlays=false}
```

### curlyquotes

Let ` and ' produce curly quotation marks ‘ and ’ rather than the backtick and typewriter single quotation mark produced by default via upquote.

```
2189 \newbool{FV@CurlyQuotes}
2190 \define@booleankey{FV}{curlyquotes}%
2191 {\booltrue{FV@CurlyQuotes}}%
2192 {\boolfalse{FV@CurlyQuotes}}
2193 \def\FancyVerbCurlyQuotes{%
2194   \ifbool{FV@CurlyQuotes}%
2195     {\expandafter\def\expandafter\@noligs\expandafter{\@noligs
2196       \begingroup\lccode`\~=`\lowercase{\endgroup\def~}{`}%
2197       \begingroup\lccode`\~=`'\lowercase{\endgroup\def~}{'}}}%
2198   {}}
2199 \g@addto@macro\FV@FormattingPrep@PreHook{\FancyVerbCurlyQuotes}
2200 \fvset{curlyquotes=false}
```

### fontencoding

Add option for font encoding.

```
2201 \define@key{FV}{fontencoding}%
2202 {\ifstrempy{#1}%
2203   {\let\FV@FontEncoding\relax}%
2204   {\ifstrequal{#1}{none}%
2205     {\let\FV@FontEncoding\relax}%
2206     {\def\FV@FontEncoding{\fontencoding{#1}}}}
2207 \expandafter\def\expandafter\FV@SetupFont\expandafter{%
2208   \expandafter\FV@FontEncoding\FV@SetupFont}
2209 \fvset{fontencoding=none}
```

### 12.11.2 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 12.11.5). 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.

```
2210 \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.

```
2211 \def\FancyVerbFormatText#1{#1}
```

#### `\FV@ListProcessLine@NoBreak`

Redefined `\FV@ListProcessLine` in which `bgcolor` support is added, `\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.

```
2212 \def\FV@ListProcessLine@NoBreak#1{%
2213   \hbox to \hsize{%
2214     \kern\leftmargin
2215     \hbox to \linewidth{%
2216       \FV@LeftListNumber
2217       \FV@LeftListFrame
2218       \FV@BGColor@List{%
2219         \FancyVerbFormatLine{%
2220           \FancyVerbHighlightLine{%
2221             \FV@ObeyTabs{\FancyVerbFormatText{#1}}}\hss
2222         \FV@RightListFrame
2223         \FV@RightListNumber}%
2224       \hss}\FV@bgcoloroverlap}
```

#### `\FV@BProcessLine`

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

```

2225 \def\FV@BProcessLine#1{%
2226   \hbox{\FancyVerbFormatLine{%
2227     \ifx\FancyVerbBackgroundColor\relax
2228     \else
2229       \expandafter\FancyVerbBackgroundColorVPhantom
2230     \fi
2231     \FancyVerbHighlightLine{%
2232       \FV@ObeyTabs{\FancyVerbFormatText{#1}}}}}}

```

### 12.11.3 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`

```
2233 \newbool{FV@NumberFirstLine}
```

#### `numberfirstline`

```

2234 \define@booleankey{FV}{numberfirstline}%
2235 {\booltrue{FV@NumberFirstLine}}%
2236 {\boolfalse{FV@NumberFirstLine}}
2237 \fvset{numberfirstline=false}

```

#### `FV@StepNumberFromFirst`

```
2238 \newbool{FV@StepNumberFromFirst}
```

#### `stepnumberfromfirst`

```

2239 \define@booleankey{FV}{stepnumberfromfirst}%
2240 {\booltrue{FV@StepNumberFromFirst}}%
2241 {\boolfalse{FV@StepNumberFromFirst}}
2242 \fvset{stepnumberfromfirst=false}

```

#### `FV@StepNumberOffsetValues`

```
2243 \newbool{FV@StepNumberOffsetValues}
```

#### `stepnumberoffsetvalues`

```

2244 \define@booleankey{FV}{stepnumberoffsetvalues}%
2245 {\booltrue{FV@StepNumberOffsetValues}}%
2246 {\boolfalse{FV@StepNumberOffsetValues}}
2247 \fvset{stepnumberoffsetvalues=false}

```

#### `\FV@Numbers@left`

Redefine `fancyvrb` macro to account for `numberfirstline`, `stepnumberfromfirst`, and `stepnumberoffsetvalues`. The `\let\FancyVerbStartNum\@ne` is needed to account for the case where `firstline` is never set, and defaults to zero (`\z@`).

```

2248 \def\FV@Numbers@left{%
2249   \let\FV@RightListNumber\relax

```

```

2250 \def\FV@LeftListNumber{%
2251   \ifx\FancyVerbStartNum\z@
2252     \let\FancyVerbStartNum\@ne
2253   \fi
2254   \ifbool{FV@StepNumberFromFirst}%
2255     {\@tempcnta=\FV@CodeLineNo
2256      \@tempcntb=\FancyVerbStartNum
2257      \advance\@tempcntb\FV@StepNumber
2258      \divide\@tempcntb\FV@StepNumber
2259      \multiply\@tempcntb\FV@StepNumber
2260      \advance\@tempcnta\@tempcntb
2261      \advance\@tempcnta-\FancyVerbStartNum
2262      \@tempcntb=\@tempcnta}%
2263     {\ifbool{FV@StepNumberOffsetValues}%
2264      {\@tempcnta=\FV@CodeLineNo
2265       \@tempcntb=\FV@CodeLineNo}%
2266      {\@tempcnta=\c@FancyVerbLine
2267       \@tempcntb=\c@FancyVerbLine}}}%
2268   \divide\@tempcntb\FV@StepNumber
2269   \multiply\@tempcntb\FV@StepNumber
2270   \ifnum\@tempcnta=\@tempcntb
2271     \ifFV@NumberBlankLines
2272       \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
2273     \else
2274       \ifx\FV@Line\empty
2275         \else
2276           \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
2277         \fi
2278       \fi
2279     \else
2280       \ifbool{FV@NumberFirstLine}{%
2281         \ifnum\FV@CodeLineNo=\FancyVerbStartNum
2282           \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
2283         \fi}{}%
2284       \fi}%
2285 }

```

### `\FV@Numbers@right`

Redefine fancyvrb macro to account for numberfirstline, stepnumberfromfirst, and stepnumberoffsetvalues.

```

2286 \def\FV@Numbers@right{%
2287   \let\FV@LeftListNumber\relax
2288   \def\FV@RightListNumber{%
2289     \ifx\FancyVerbStartNum\z@
2290       \let\FancyVerbStartNum\@ne
2291     \fi
2292     \ifbool{FV@StepNumberFromFirst}%
2293       {\@tempcnta=\FV@CodeLineNo
2294        \@tempcntb=\FancyVerbStartNum
2295        \advance\@tempcntb\FV@StepNumber
2296        \divide\@tempcntb\FV@StepNumber
2297        \multiply\@tempcntb\FV@StepNumber
2298        \advance\@tempcnta\@tempcntb
2299        \advance\@tempcnta-\FancyVerbStartNum

```

```

2300     \@tempcntb=\@tempcnta}%
2301     {\ifbool{FV@StepNumberOffsetValues}%
2302     {\@tempcnta=FV@CodeLineNo
2303     \@tempcntb=FV@CodeLineNo}%
2304     {\@tempcnta=c@FancyVerbLine
2305     \@tempcntb=c@FancyVerbLine}}%
2306     \divide\@tempcntb\FV@StepNumber
2307     \multiply\@tempcntb\FV@StepNumber
2308     \ifnum\@tempcnta=\@tempcntb
2309     \if@FV@NumberBlankLines
2310     \hbox to\z@{\kern\FV@NumberSep\theFancyVerbLine\hss}%
2311     \else
2312     \ifx\FV@Line\empty
2313     \else
2314     \hbox to\z@{\kern\FV@NumberSep\theFancyVerbLine\hss}%
2315     \fi
2316     \fi
2317     \else
2318     \ifbool{FV@NumberFirstLine}{%
2319     \ifnum\FV@CodeLineNo=FancyVerbStartNum
2320     \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
2321     \fi}{}%
2322     \fi}%
2323 }

```

#### \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.

```

2324 \def\FV@Numbers@both{%
2325   \def\FV@LeftListNumber{%
2326     \ifx\FancyVerbStartNum\z@
2327     \let\FancyVerbStartNum@ne
2328     \fi
2329     \ifbool{FV@StepNumberFromFirst}%
2330     {\@tempcnta=FV@CodeLineNo
2331     \@tempcntb=FancyVerbStartNum
2332     \advance\@tempcntb\FV@StepNumber
2333     \divide\@tempcntb\FV@StepNumber
2334     \multiply\@tempcntb\FV@StepNumber
2335     \advance\@tempcnta\@tempcntb
2336     \advance\@tempcnta-FancyVerbStartNum
2337     \@tempcntb=\@tempcnta}%
2338     {\ifbool{FV@StepNumberOffsetValues}%
2339     {\@tempcnta=FV@CodeLineNo
2340     \@tempcntb=FV@CodeLineNo}%
2341     {\@tempcnta=c@FancyVerbLine
2342     \@tempcntb=c@FancyVerbLine}}%
2343     \divide\@tempcntb\FV@StepNumber
2344     \multiply\@tempcntb\FV@StepNumber
2345     \ifnum\@tempcnta=\@tempcntb
2346     \if@FV@NumberBlankLines
2347     \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
2348     \else

```

```

2349     \ifx\FV@Line\empty
2350     \else
2351         \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
2352     \fi
2353 \fi
2354 \else
2355     \ifbool{FV@NumberFirstLine}{%
2356         \ifnum\FV@CodeLineNo=\FancyVerbStartNum
2357             \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
2358         \fi}{}%
2359 \fi}%
2360 \def\FV@RightListNumber{%
2361     \ifx\FancyVerbStartNum\z@
2362         \let\FancyVerbStartNum\@ne
2363     \fi
2364     \ifbool{FV@StepNumberFromFirst}{%
2365         {\@tempcnta=\FV@CodeLineNo
2366         \@tempcntb=\FancyVerbStartNum
2367         \advance\@tempcntb\FV@StepNumber
2368         \divide\@tempcntb\FV@StepNumber
2369         \multiply\@tempcntb\FV@StepNumber
2370         \advance\@tempcnta\@tempcntb
2371         \advance\@tempcnta-\FancyVerbStartNum
2372         \@tempcntb=\@tempcnta}%
2373     {\ifbool{FV@StepNumberOffsetValues}{%
2374         {\@tempcnta=\FV@CodeLineNo
2375         \@tempcntb=\FV@CodeLineNo}%
2376         {\@tempcnta=\c@FancyVerbLine
2377         \@tempcntb=\c@FancyVerbLine}}}%
2378     \divide\@tempcntb\FV@StepNumber
2379     \multiply\@tempcntb\FV@StepNumber
2380     \ifnum\@tempcnta=\@tempcntb
2381         \if@FV@NumberBlankLines
2382             \hbox to\z@{\kern\FV@NumberSep\theFancyVerbLine\hss}%
2383         \else
2384             \ifx\FV@Line\empty
2385                 \else
2386                     \hbox to\z@{\kern\FV@NumberSep\theFancyVerbLine\hss}%
2387             \fi
2388         \fi
2389     \else
2390         \ifbool{FV@NumberFirstLine}{%
2391             \ifnum\FV@CodeLineNo=\FancyVerbStartNum
2392                 \hbox to\z@{\hss\theFancyVerbLine\kern\FV@NumberSep}%
2393             \fi}{}%
2394     \fi}%
2395 }

```

#### 12.11.4 Background color

Define an option `backgroundcolor` that provides a basic implementation of a background color behind commands and environments. `tcolorbox` or a similar package should be used for more sophisticated background colors.

The patch to `\FV@List` prevents narrow horizontal gaps in the background



color between lines of text under some circumstances. The `\ExplSyntax*` is for compatibility with patches from the L<sup>A</sup>T<sub>E</sub>X Tagged PDF project.

`backgroundcolor`

`bgcolor`

`\FancyVerbBackgroundColor`

```

2396 \define@key{FV}{backgroundcolor}{%
2397   \def\FancyVerbBackgroundColor{#1}%
2398   \ifx\FancyVerbBackgroundColor\FV@None
2399     \let\FancyVerbBackgroundColor\relax
2400   \else\ifx\FancyVerbBackgroundColor@empty
2401     \let\FancyVerbBackgroundColor\relax
2402   \fi\fi}%
2403 \fvset{backgroundcolor=none}
2404 \define@key{FV}{bgcolor}{%
2405   \fvset{backgroundcolor=#1}}
2406 \ExplSyntaxOn
2407 \patchcmd{\FV@List}{%
2408   {\FV@ObeyTabsInit}%
2409   {\FV@ObeyTabsInit
2410     \ifx\FancyVerbBackgroundColor\relax
2411       \else
2412         \lineskip\z@
2413       \fi}%
2414   }{}%
2415 {\PackageError{fvextra}{%
2416   {Failed-to-patch-\detokenize{\FV@List}for-backgroundcolor}%
2417   {Failed-to-patch-\detokenize{\FV@List}for-backgroundcolor}}
2418 \ExplSyntaxOff

```

`backgroundcolorboxoverlap`

`bgcolorboxoverlap`

`\FV@backgroundcolorboxoverlap`

`\FV@bgcolorstrut`

`\FV@bgcoloroverlap`

```

2419 \define@key{FV}{backgroundcolorboxoverlap}{%
2420   \ifdim#1=0pt\relax
2421     \let\FV@backgroundcolorboxoverlap\relax
2422   \else
2423     \def\FV@backgroundcolorboxoverlap{#1}%
2424   \fi}
2425 \fvset{backgroundcolorboxoverlap=0.25pt}
2426 \define@key{FV}{bgcolorboxoverlap}{%
2427   \fvset{backgroundcolorboxoverlap=#1}}
2428 \newsavebox{\FV@bgcolorstructbox}
2429 \def\FV@bgcolorstrut{%
2430   \ifx\FancyVerbBackgroundColor\relax
2431     \FancyVerbBackgroundColorVPhantom\strut
2432   \else\ifx\FV@backgroundcolorboxoverlap\relax
2433     \FancyVerbBackgroundColorVPhantom\strut
2434   \else
2435     \savebox{\FV@bgcolorstructbox}{\hbox{\FancyVerbBackgroundColorVPhantom\strut}}%
2436     \vrule height \ht\FV@bgcolorstructbox
2437           depth \dimexpr\dp\FV@bgcolorstructbox+\FV@backgroundcolorboxoverlap\relax

```

```

2438         width Opt\relax
2439     \fi\fi}
2440 \def\FV@bgcoloroverlap{%
2441     \ifx\FV@backgroundcolorboxoverlap\relax
2442     \else
2443     \vspace{-\FV@backgroundcolorboxoverlap}%
2444     \fi}

```

backgroundcolorvphantom

bgcolorvphantom

\FancyVerbBackgroundColorVPhantom

```

2445 \define@key{FV}{backgroundcolorvphantom}{%
2446     \def\FancyVerbBackgroundColorVPhantom{#1}%
2447     \ifx\FancyVerbBackgroundColorVPhantom\FV@None
2448     \let\FancyVerbBackgroundColorVPhantom\relax
2449     \else\ifx\FancyVerbBackgroundColorVPhantom\@empty
2450     \let\FancyVerbBackgroundColorVPhantom\relax
2451     \fi\fi}
2452 \fvset{backgroundcolorvphantom=\vphantom{"Apgjy}}
2453 \define@key{FV}{bgcolorvphantom}{%
2454     \fvset{backgroundcolorvphantom=#1}}

```

backgroundcolorpadding

bgcolorpadding

\FancyVerbBackgroundColorPadding

```

2455 \let\FancyVerbBackgroundColorPadding\relax
2456 \def\FV@backgroundcolorpadding@none@framenotsingle{%
2457     \fvset{frame=none,framerule,rulecolor=none}}
2458 \def\FV@backgroundcolorpadding@dim@framenotsingle{%
2459     \fvset{frame=single,framerule=Opt,rulecolor=\FancyVerbBackgroundColor}}
2460 \define@key{FV}{backgroundcolorpadding}{%
2461     \def\FancyVerbBackgroundColorPadding{#1}%
2462     \ifx\FancyVerbBackgroundColorPadding\FV@None
2463     \let\FancyVerbBackgroundColorPadding\relax
2464     \else\ifx\FancyVerbBackgroundColorPadding\@empty
2465     \let\FancyVerbBackgroundColorPadding\relax
2466     \fi\fi
2467     \let\FV@Next\relax
2468     \ifx\FancyVerbBackgroundColorPadding\relax
2469     \ifx\FV@BeginListFrame\FV@BeginListFrame@Single
2470     \else
2471     \let\FV@Next\FV@backgroundcolorpadding@none@framenotsingle
2472     \fi
2473     \ifx\FV@LeftListFrame\FV@LeftListFrame@Single
2474     \else
2475     \let\FV@Next\FV@backgroundcolorpadding@none@framenotsingle
2476     \fi
2477     \ifx\FV@RightListFrame\FV@RightListFrame@Single
2478     \else
2479     \let\FV@Next\FV@backgroundcolorpadding@none@framenotsingle
2480     \fi
2481     \ifx\FV@endListFrame\FV@endListFrame@Single
2482     \else
2483     \let\FV@Next\FV@backgroundcolorpadding@none@framenotsingle

```

```

2484 \fi
2485 \FV@Next
2486 \fvset{framesep,fillcolor=none}%
2487 \else
2488 \ifx\FV@BeginListFrame\FV@BeginListFrame@Single
2489 \else
2490 \let\FV@Next\FV@backgroundcolorpadding@dim@framenotsingle
2491 \fi
2492 \ifx\FV@LeftListFrame\FV@LeftListFrame@Single
2493 \else
2494 \let\FV@Next\FV@backgroundcolorpadding@dim@framenotsingle
2495 \fi
2496 \ifx\FV@RightListFrame\FV@RightListFrame@Single
2497 \else
2498 \let\FV@Next\FV@backgroundcolorpadding@dim@framenotsingle
2499 \fi
2500 \ifx\FV@endListFrame\FV@endListFrame@Single
2501 \else
2502 \let\FV@Next\FV@backgroundcolorpadding@dim@framenotsingle
2503 \fi
2504 \FV@Next
2505 \fvset{framesep=#1,fillcolor=\FancyVerbBackgroundColor}%
2506 \fi}
2507 \define@key{FV}{bgcolorpadding}{\fvset{backgroundcolorpadding=#1}}

```

#### `\FV@BGColor@List`

Background color for environments based on Verbatim. Puts each line in a colorbox.

```

2508 \def\FV@BGColor@List#1{%
2509 \ifx\FancyVerbBackgroundColor\relax
2510 \expandafter\@firstoftwo
2511 \else
2512 \expandafter\@secondoftwo
2513 \fi
2514 {#1}%
2515 {\setlength{\FV@TmpLength}{\fboxsep}%
2516 \setlength{\fboxsep}{0pt}%
2517 \colorbox{\FancyVerbBackgroundColor}{%
2518 \setlength{\fboxsep}{\FV@TmpLength}%
2519 \rlap{\FV@bgcolorstrut#1}%
2520 \hspace{\linewidth}%
2521 \ifx\FV@RightListFrame\relax\else
2522 \hspace{-\FV@FrameSep}%
2523 \hspace{-\FV@FrameRule}%
2524 \fi
2525 \ifx\FV@LeftListFrame\relax\else
2526 \hspace{-\FV@FrameSep}%
2527 \hspace{-\FV@FrameRule}%
2528 \fi}%
2529 \hss}}

```

#### `\FV@BVerbatimBegin`

#### `\FV@BVerbatimEnd`

Reimplementation of BVerbatim macros to support bgcolor. Much of this follows the implementation of SaveVerbatim and \UseVerbatim.

Key values, formatting, and tabs must be configured immediately in the `bgcolor` case, so that the background color and other settings are available. `\FV@UseKeyValues` can be invoked multiple times, but that doesn't cause any issues since `\FV@UseKeyValues` applies keys and then redefines `\FV@KeyValues` to empty.

The definition of `\FV@BProcessLine` already accounts for `\FancyVerbBackgroundColorVPhantom`.

```

2530 \let\FV@BVerbatimBegin@NoBGColor\FV@BVerbatimBegin
2531 \let\FV@BVerbatimEnd@NoBGColor\FV@BVerbatimEnd
2532 \def\FV@BVerbatimBegin{%
2533   \begingroup
2534   \FV@UseKeyValues
2535   \FV@FormattingPrep
2536   \let\FV@FormattingPrep\relax
2537   \FV@ObeyTabsInit
2538   \let\FV@ObeyTabsInit\relax
2539   \ifx\FancyVerbBackgroundColor\relax
2540     \expandafter\FV@BVerbatimBegin@NoBGColor
2541   \else
2542     \expandafter\FV@BVerbatimBegin@BGColor
2543   \fi}
2544 \def\FV@BVerbatimEnd{%
2545   \ifx\FancyVerbBackgroundColor\relax
2546     \expandafter\FV@BVerbatimEnd@NoBGColor
2547   \else
2548     \expandafter\FV@BVerbatimEnd@BGColor
2549   \fi
2550 \endgroup}
2551 \def\FV@BVerbatimBegin@BGColor{%
2552   \gdef\FV@TheVerbatim{%
2553     \ifx\FV@boxwidth\relax
2554       \gdef\FV@boxwidth@tmp{Opt}%
2555       \def\FV@ProcessLine##1{%
2556         \sbox{\FV@LineBox}{\FV@BProcessLine{##1}}%
2557         \ifdim\wd\FV@LineBox>\FV@boxwidth@tmp\relax
2558           \xdef\FV@boxwidth@tmp{\the\wd\FV@LineBox}%
2559         \fi
2560         \expandafter\gdef\expandafter\FV@TheVerbatim\expandafter{%
2561           \FV@TheVerbatim\FV@ProcessLine{##1}}}%
2562     \else
2563       \def\FV@ProcessLine##1{%
2564         \expandafter\gdef\expandafter\FV@TheVerbatim\expandafter{%
2565           \FV@TheVerbatim\FV@ProcessLine{##1}}}%
2566     \fi}
2567 \def\FV@BVerbatimEnd@BGColor{%
2568   \ifx\FV@boxwidth\relax
2569     \let\FV@boxwidth\FV@boxwidth@tmp
2570     \global\let\FV@boxwidth@tmp\FV@Undefined
2571   \fi
2572   \setlength{\FV@TmpLength}{\fboxsep}%
2573   \ifx\FancyVerbBackgroundColorPadding\relax
2574     \setlength{\fboxsep}{Opt}%
2575   \else
2576     \setlength{\fboxsep}{\FancyVerbBackgroundColorPadding}%

```

```

2577 \fi
2578 \colorbox{\FancyVerbBackgroundColor}{%
2579   \setlength{\fboxsep}{\FV@TmpLength}%
2580   \FV@BVerbatimBegin@NoBGColor\FV@TheVerbatim\FV@BVerbatimEnd@NoBGColor}%
2581 \gdef\FV@TheVerbatim{}}

```

### 12.11.5 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`

```

2582 \define@key{FV}{highlightlines}{\def\FV@HighlightLinesList{#1}}%
2583 \fvset{highlightlines=}

```

`highlightcolor`

`\FV@HighlightColor`

Define color for highlighting. The default is `LightCyan`. A good alternative for a brighter color would be `LemonChiffon`.

```

2584 \define@key{FV}{highlightcolor}{\def\FancyVerbHighlightColor{#1}}%
2585 \let\FancyVerbHighlightColor\@empty
2586 \ifcsname definecolor\endcsname
2587 \ifx\definecolor\relax
2588 \else
2589   \definecolor{FancyVerbHighlightColor}{rgb}{0.878, 1, 1}
2590   \fvset{highlightcolor=FancyVerbHighlightColor}
2591 \fi\fi
2592 \AtBeginDocument{%
2593   \ifx\FancyVerbHighlightColor\@empty
2594     \ifcsname definecolor\endcsname
2595     \ifx\definecolor\relax
2596     \else
2597       \definecolor{FancyVerbHighlightColor}{rgb}{0.878, 1, 1}
2598       \fvset{highlightcolor=FancyVerbHighlightColor}
2599     \fi\fi
2600 \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.

```

2601 \def\FancyVerbHighlightLine#1{#1}

```

`\FV@HighlightLine`

This determines whether highlighting should be performed, and if so, which macro should be invoked.

```

2602 \def\FV@HighlightLine#1{%
2603   \@tempcnta=\c@FancyVerbLine
2604   \@tempcntb=\c@FancyVerbLine
2605   \ifcsname FV@HighlightLine:\number\@tempcnta\endcsname
2606     \advance\@tempcntb\m@ne

```

```

2607 \ifcsname FV@HighlightLine:\number\@tempcntb\endcsname
2608 \advance\@tempcntb\tw@
2609 \ifcsname FV@HighlightLine:\number\@tempcntb\endcsname
2610 \let\FV@HighlightLine@Next\FancyVerbHighlightLineMiddle
2611 \else
2612 \let\FV@HighlightLine@Next\FancyVerbHighlightLineLast
2613 \fi
2614 \else
2615 \advance\@tempcntb\tw@
2616 \ifcsname FV@HighlightLine:\number\@tempcntb\endcsname
2617 \let\FV@HighlightLine@Next\FancyVerbHighlightLineFirst
2618 \else
2619 \let\FV@HighlightLine@Next\FancyVerbHighlightLineSingle
2620 \fi
2621 \fi
2622 \else
2623 \let\FV@HighlightLine@Next\FancyVerbHighlightLineNormal
2624 \fi
2625 \FV@HighlightLine@Next{#1}%
2626 }

```

#### `\FancyVerbHighlightLineNormal`

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

```
2627 \def\FancyVerbHighlightLineNormal#1{#1}
```

#### `\FV@TmpLength`

```
2628 \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`  $\neq$  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.

```

2629 \def\FancyVerbHighlightLineFirst#1{%
2630 \setlength{\FV@TmpLength}{\fboxsep}%
2631 \setlength{\fboxsep}{0pt}%
2632 \colorbox{\FancyVerbHighlightColor}{%
2633 \setlength{\fboxsep}{\FV@TmpLength}%
2634 \rlap{\strut#1}%
2635 \hspace{\linewidth}%
2636 \ifx\FV@RightListFrame\relax\else
2637 \hspace{-\FV@FrameSep}%
2638 \hspace{-\FV@FrameRule}%
2639 \fi
2640 \ifx\FV@LeftListFrame\relax\else
2641 \hspace{-\FV@FrameSep}%
2642 \hspace{-\FV@FrameRule}%

```

```

2643     \fi
2644   }%
2645   \hss
2646 }

\FancyVerbHighlightLineMiddle
  A middle line in a multi-line range.
2647 \let\FancyVerbHighlightLineMiddle\FancyVerbHighlightLineFirst

\FancyVerbHighlightLineLast
  The last line in a multi-line range.
2648 \let\FancyVerbHighlightLineLast\FancyVerbHighlightLineFirst

\FancyVerbHighlightLineSingle
  A single line not in a multi-line range.
2649 \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.
2650 \def\FV@HighlightLinesPrep{%
2651   \ifx\FV@HighlightLinesList\@empty
2652   \else
2653     \let\FancyVerbHighlightLine\FV@HighlightLine
2654     \expandafter\FV@HighlightLinesPrep@i
2655     \fi}
2656 \def\FV@HighlightLinesPrep@i{%
2657   \renewcommand{\do}[1]{%
2658     \ifstrempy{##1}{\FV@HighlightLinesParse##1-\FV@Undefined}}%
2659   \expandafter\docsvlist\expandafter{\FV@HighlightLinesList}}
2660 \def\FV@HighlightLinesParse#1-#2\FV@Undefined{%
2661   \ifstrempy{#2}%
2662     {\FV@HighlightLinesParse@Single{#1}}%
2663     {\FV@HighlightLinesParse@Range{#1}#2\relax}}
2664 \def\FV@HighlightLinesParse@Single#1{%
2665   \expandafter\let\csname FV@HighlightLine:\detokenize{#1}\endcsname\relax}
2666 \newcounter{FV@HighlightLinesStart}
2667 \newcounter{FV@HighlightLinesStop}
2668 \def\FV@HighlightLinesParse@Range#1#2-{%
2669   \setcounter{FV@HighlightLinesStart}{#1}%
2670   \setcounter{FV@HighlightLinesStop}{#2}%
2671   \stepcounter{FV@HighlightLinesStop}%
2672   \FV@HighlightLinesParse@Range@Loop}
2673 \def\FV@HighlightLinesParse@Range@Loop{%
2674   \ifnum\value{FV@HighlightLinesStart}<\value{FV@HighlightLinesStop}\relax
2675     \expandafter\let\csname FV@HighlightLine:\arabic{FV@HighlightLinesStart}\endcsname\relax
2676     \stepcounter{FV@HighlightLinesStart}%
2677     \expandafter\FV@HighlightLinesParse@Range@Loop
2678   \fi}
2679 \g@addto@macro\FV@FormattingPrep@PreHook{\FV@HighlightLinesPrep}

\FV@StepLineNo@Patch@HighlightLine

```

Patch `\FV@StepLineNo` so that when `numberblanklines=false`, blank lines won't be highlighted. If the previous line is at the end of a highlighted range, then `\let \FV@HighlightLine:<n> to \FV@Undefined` to prevent further highlighting. Otherwise, leave everything as-is since the blank line(s) are within a highlighted range.

```

2680 \def\FV@StepLineNo@Patch@HighlightLine{%
2681   \ifcsname FV@HighlightLine:\number\c@FancyVerbLine\endcsname
2682     \@tempcnta=\c@FancyVerbLine
2683     \advance\@tempcnta\@ne
2684     \ifcsname FV@HighlightLine:\number\@tempcnta\endcsname
2685       \else
2686         \expandafter\let
2687           \csname FV@HighlightLine:\number\c@FancyVerbLine\endcsname\FV@Undefined
2688         \fi
2689       \fi}
2690 \patchcmd{\FV@StepLineNo}%
2691   {\ifx\FV@Line\empty}%
2692   {\ifx\FV@Line\empty\FV@StepLineNo@Patch@HighlightLine}%
2693   {}%
2694   {\PackageError{fvextra}%
2695     {Failed to patch \string\FV@StepLineNo\ to make highlightlines
2696     compatible with numberblanklines}%
2697   {Failed to patch \string\FV@StepLineNo\ to make highlightlines
2698     compatible with numberblanklines}}

```

## 12.12 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.

### 12.12.1 Options and associated macros

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

#### `\FV@SetToWidthNChars`

Set a dimen to the width of a given number of characters. This is used in setting several indentation-related dimensions.

```

2699 \newcount\FV@LoopCount
2700 \newbox\FV@NCharsBox
2701 \def\FV@SetToWidthNChars#1#2{%
2702   \FV@LoopCount=#2\relax
2703   \ifnum\FV@LoopCount>0
2704     \def\FV@NChars{}%
2705     \loop
2706       \ifnum\FV@LoopCount>0
2707         \expandafter\def\expandafter\FV@NChars\expandafter{\FV@NChars x}%
2708       \fi
2709       \advance\FV@LoopCount by -1
2710     \ifnum\FV@LoopCount>0
2711     \repeat

```



```

2712 \setbox\FV@NCharsBox\hbox{\FV@NChars}%
2713 #1=\wd\FV@NCharsBox
2714 \else
2715 #1=0pt\relax
2716 \fi
2717 }

```

### 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.

```

2718 \newbool{FV@breaklines}
2719 \define@booleankey{FV}{breaklines}%
2720 {\booltrue{FV@breaklines}%
2721 \let\FV@ListProcessLine\FV@ListProcessLine@Break}%
2722 {\boolfalse{FV@breaklines}%
2723 \let\FV@ListProcessLine\FV@ListProcessLine@NoBreak}
2724 \AtEndOfPackage{\fvset{breaklines=false}}

```

### \FV@BreakLinesLuaTeXHook

Fix hyphen handling under LuaTeX. `\automaticshyphenmode=2` would work for environments, but doesn't seem to work inline. Instead, the active hyphen is redefined to `\mbox{-}`.

This is needed before `\@noligs` is ever used, so it is placed in `\FV@FormattingPrep@PreHook`.

```

2725 \def\FV@BreakLinesLuaTeXHook{%
2726 \expandafter\def\expandafter\@noligs\expandafter{\@noligs
2727 \begingroup\lccode`~=`-\lowercase{\endgroup\def~}{\leavevmode\kern\z@\mbox{-}}}}
2728 \ifcsname directlua\endcsname
2729 \ifx\directlua\relax
2730 \else
2731 \FV@AddToHook\FV@FormattingPrep@PreHook\FV@BreakLinesLuaTeXHook
2732 \fi
2733 \fi

```

### \FV@BreakLinesIndentationHook

A hook for performing on-the-fly indentation calculations when `breaklines=true`. This is used for all `*NChars` related indentation. It is important to use `\FV@FormattingPrep@PostHook` because it is always invoked *after* any font-related settings.

```

2734 \def\FV@BreakLinesIndentationHook{}
2735 \g@addto@macro\FV@FormattingPrep@PostHook{%
2736 \ifFV@breaklines
2737 \FV@BreakLinesIndentationHook
2738 \fi}

```

### \FV@BreakIndent

#### \FV@BreakIndentNChars

Indentation of continuation lines.

```

2739 \newdimen\FV@BreakIndent
2740 \newcount\FV@BreakIndentNChars
2741 \define@key{FV}{breakindent}{%
2742 \FV@BreakIndent=#1\relax
2743 \FV@BreakIndentNChars=0\relax}
2744 \define@key{FV}{breakindentnchars}{\FV@BreakIndentNChars=#1\relax}
2745 \g@addto@macro\FV@BreakLinesIndentationHook{%
2746 \ifnum\FV@BreakIndentNChars>0

```

```

2747     \FV@SetToWidthNChars{\FV@BreakIndent}{\FV@BreakIndentNChars}%
2748     \fi}
2749     \fvset{breakindentnchars=0}

```

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

```

2750 \newbool{FV@breakautoindent}
2751 \define@booleankey{FV}{breakautoindent}%
2752   {\booltrue{FV@breakautoindent}}{\boolfalse{FV@breakautoindent}}
2753 \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.

```

2754 \define@key{FV}{breaksymbolleft}{\def\FancyVerbBreakSymbolLeft{#1}}
2755 \define@key{FV}{breaksymbol}{\fvset{breaksymbolleft=#1}}
2756 \fvset{breaksymbolleft=\tiny\ensuremath{\hookrightarrow}}

```

**\FancyVerbBreakSymbolRight**  
The right-hand symbol indicating a break.

```

2757 \define@key{FV}{breaksymbolright}{\def\FancyVerbBreakSymbolRight{#1}}
2758 \fvset{breaksymbolright={}}

```

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

```

2759 \newdimen\FV@BreakSymbolSepLeft
2760 \newcount\FV@BreakSymbolSepLeftNChars
2761 \define@key{FV}{breaksymbolsepleft}{%
2762   \FV@BreakSymbolSepLeft=#1\relax
2763   \FV@BreakSymbolSepLeftNChars=0\relax}
2764 \define@key{FV}{breaksymbolsep}{\fvset{breaksymbolsepleft=#1}}
2765 \define@key{FV}{breaksymbolsepleftnchars}{\FV@BreakSymbolSepLeftNChars=#1\relax}
2766 \define@key{FV}{breaksymbolsepnchars}{\fvset{breaksymbolsepleftnchars=#1}}
2767 \g@addto@macro\FV@BreakLinesIndentationHook{%
2768   \ifnum\FV@BreakSymbolSepLeftNChars>0
2769     \FV@SetToWidthNChars{\FV@BreakSymbolSepLeft}{\FV@BreakSymbolSepLeftNChars}%
2770     \fi}
2771 \fvset{breaksymbolsepleftnchars=2}

```

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

```

2772 \newdimen\FV@BreakSymbolSepRight
2773 \newcount\FV@BreakSymbolSepRightNChars
2774 \define@key{FV}{breaksymbolsepright}{%
2775   \FV@BreakSymbolSepRight=#1\relax
2776   \FV@BreakSymbolSepRightNChars=0\relax}
2777 \define@key{FV}{breaksymbolseprightnchars}{\FV@BreakSymbolSepRightNChars=#1\relax}
2778 \g@addto@macro\FV@BreakLinesIndentationHook{%
2779   \ifnum\FV@BreakSymbolSepRightNChars>0
2780     \FV@SetToWidthNChars{\FV@BreakSymbolSepRight}{\FV@BreakSymbolSepRightNChars}%

```

```

2781 \fi}
2782 \fvset{breaksymbolseprightnchars=2}
\FV@BreakSymbolIndentLeft
\FV@BreakSymbolIndentLeftNChars
    Additional left indentation to make room for the left break symbol.
2783 \newdimen\FV@BreakSymbolIndentLeft
2784 \newcount\FV@BreakSymbolIndentLeftNChars
2785 \define@key{FV}{breaksymbolindentleft}{%
2786 \FV@BreakSymbolIndentLeft=#1\relax
2787 \FV@BreakSymbolIndentLeftNChars=0\relax}
2788 \define@key{FV}{breaksymbolindent}{\fvset{breaksymbolindentleft=#1}}
2789 \define@key{FV}{breaksymbolindentleftnchars}{\FV@BreakSymbolIndentLeftNChars=#1\relax}
2790 \define@key{FV}{breaksymbolindentnchars}{\fvset{breaksymbolindentleftnchars=#1}}
2791 \g@addto@macro\FV@BreakLinesIndentationHook{%
2792 \ifnum\FV@BreakSymbolIndentLeftNChars>0
2793 \FV@SetToWidthNChars{\FV@BreakSymbolIndentLeft}{\FV@BreakSymbolIndentLeftNChars}%
2794 \fi}
2795 \fvset{breaksymbolindentleftnchars=4}
\FV@BreakSymbolIndentRight
\FV@BreakSymbolIndentRightNChars
    Additional right indentation to make room for the right break symbol.
2796 \newdimen\FV@BreakSymbolIndentRight
2797 \newcount\FV@BreakSymbolIndentRightNChars
2798 \define@key{FV}{breaksymbolindentright}{%
2799 \FV@BreakSymbolIndentRight=#1\relax
2800 \FV@BreakSymbolIndentRightNChars=0\relax}
2801 \define@key{FV}{breaksymbolindentrightnchars}{\FV@BreakSymbolIndentRightNChars=#1\relax}
2802 \g@addto@macro\FV@BreakLinesIndentationHook{%
2803 \ifnum\FV@BreakSymbolIndentRightNChars>0
2804 \FV@SetToWidthNChars{\FV@BreakSymbolIndentRight}{\FV@BreakSymbolIndentRightNChars}%
2805 \fi}
2806 \fvset{breaksymbolindentrightnchars=4}
    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.
2807 \newcommand{\FancyVerbBreakSymbolLeftLogic}[1]{%
2808 \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.
2809 \newcounter{FancyVerbLineBreakLast}

```

```

\FV@SetLineBreakLast
    Store the local line number for the last continuation line.
2810 \newcommand{\FV@SetLineBreakLast}{%
2811   \setcounter{FancyVerbLineBreakLast}{\value{linenumber}}}

\FancyVerbBreakSymbolRightLogic
    Only insert a right break symbol if not on the last continuation line.
2812 \newcommand{\FancyVerbBreakSymbolRightLogic}[1]{%
2813   \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.
2814 \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.
2815 \let\FancyVerbBreakStop\relax

\FV@Break@DefaultToken
    Macro that controls default token handling between \FancyVerbBreakStart
    and \FancyVerbBreakStop. Initially \let to \FV@Break@NBToken, which does not
    insert breaks. Later \let to \FV@Break@AnyToken or \FV@Break@BeforeAfterToken
    if breakanywhere or breakbefore/breakafter are in use.
2816 \let\FV@Break@DefaultToken\FV@Break@NBToken

FV@breakanywhere
    Allow line breaking (almost) anywhere. Set \FV@Break and \FV@EndBreak to
    be used, and \let \FV@Break@DefaultToken to the appropriate macro.
2817 \newbool{FV@breakanywhere}
2818 \define@booleankey{FV}{breakanywhere}%
2819   {\booltrue{FV@breakanywhere}%
2820     \let\FancyVerbBreakStart\FV@Break
2821     \let\FancyVerbBreakStop\FV@EndBreak
2822     \let\FV@Break@DefaultToken\FV@Break@AnyToken}%
2823   {\boolfalse{FV@breakanywhere}%
2824     \let\FancyVerbBreakStart\relax
2825     \let\FancyVerbBreakStop\relax
2826     \let\FV@Break@DefaultToken\FV@Break@NBToken}
2827 \fvset{breakanywhere=false}

breakanywhereinlinestretch
\FV@breakanywhereinlinestretch
\FV@ApplyBreakAnywhereInlineStretch
    Stretch glue to insert at potential breakanywhere break locations in inline
    contexts, to give better line widths and avoid overfull \hbox.
    \FV@UseInlineKeyValues invokes \FV@ApplyBreakAnywhereInlineStretch
    to redefine \FancyVerbBreakAnywhereBreak locally.
2828 \define@key{FV}{breakanywhereinlinestretch}{%
2829   \def\FV@breakanywhereinlinestretch{#1}%
2830   \ifx\FV@breakanywhereinlinestretch\FV@None
2831     \let\FV@breakanywhereinlinestretch\relax

```

```

2832 \else\ifx\FV@breakanywhereinlinestretch\@empty
2833 \let\FV@breakanywhereinlinestretch\relax
2834 \fi\fi}
2835 \fvset{breakanywhereinlinestretch=none}
2836 \def\FV@ApplyBreakAnywhereInlineStretch{%
2837 \ifx\FV@breakanywhereinlinestretch\relax
2838 \else
2839 \let\FancyVerbBreakAnywhereBreak@Orig\FancyVerbBreakAnywhereBreak
2840 \def\FancyVerbBreakAnywhereBreak{%
2841 \nobreak\hspace{Opt plus \FV@breakanywhereinlinestretch}%
2842 \FancyVerbBreakAnywhereBreak@Orig}%
2843 \fi}

```

### \FV@BreakBefore

Allow line breaking (almost) anywhere, but only before specified characters.

```

2844 \define@key{FV}{breakbefore}{%
2845 \ifstrempy{#1}%
2846 {\let\FV@BreakBefore\@empty
2847 \let\FancyVerbBreakStart\relax
2848 \let\FancyVerbBreakStop\relax
2849 \let\FV@Break@DefaultToken\FV@Break@NBToken}%
2850 }{\def\FV@BreakBefore{#1}%
2851 \let\FancyVerbBreakStart\FV@Break
2852 \let\FancyVerbBreakStop\FV@EndBreak
2853 \let\FV@Break@DefaultToken\FV@Break@BeforeAfterToken}%
2854 }
2855 \fvset{breakbefore={}}

```

### FV@breakbeforeinrun

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

```

2856 \newbool{FV@breakbeforeinrun}
2857 \define@booleankey{FV}{breakbeforeinrun}%
2858 {\booltrue{FV@breakbeforeinrun}}%
2859 {\boolfalse{FV@breakbeforeinrun}}%
2860 \fvset{breakbeforeinrun=false}

```

### \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@FormattingPrep@PreHook`, 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.

```

2861 \def\FV@BreakBeforePrep{%
2862   \ifx\FV@BreakBefore\@empty\relax
2863   \else
2864     \gdef\FV@BreakBefore@Def{%
2865       \begingroup
2866       \def\FV@BreakBefore@Process##1##2\FV@Undefined{%
2867         \expandafter\FV@BreakBefore@Process@i\expandafter{##1}%
2868         \expandafter\ifx\expandafter\relax\detokenize{##2}\relax
2869         \else
2870           \FV@BreakBefore@Process##2\FV@Undefined
2871         \fi
2872       }%
2873       \def\FV@BreakBefore@Process@i##1{%
2874         \g@addto@macro\FV@BreakBefore@Def{%
2875           \@namedef{FV@BreakBefore@Token\detokenize{##1}}{}}%
2876       }%
2877       \FV@EscChars
2878       \expandafter\FV@BreakBefore@Process\FV@BreakBefore\FV@Undefined
2879       \endgroup
2880       \FV@BreakBefore@Def
2881       \FV@BreakBeforePrep@PygmentsHook
2882     \fi
2883   }
2884 \def\FV@BreakBeforePrep@PygmentsHook{}
```

### `\FV@BreakAfter`

Allow line breaking (almost) anywhere, but only after specified characters.

```

2885 \define@key{FV}{breakafter}{%
2886   \ifstrempy{#1}%
2887   {\let\FV@BreakAfter\@empty
2888     \let\FancyVerbBreakStart\relax
2889     \let\FancyVerbBreakStop\relax
2890     \let\FV@Break@DefaultToken\FV@Break@NBToken}%
2891   {\def\FV@BreakAfter{#1}%
2892     \let\FancyVerbBreakStart\FV@Break
2893     \let\FancyVerbBreakStop\FV@EndBreak
2894     \let\FV@Break@DefaultToken\FV@Break@BeforeAfterToken}%
2895   }
2896 \fvset{breakafter={}}
```

### `FV@breakafterinrun`

Determine whether breaking after specified characters is always allowed after each individual character, or is only allowed after the last in a run of identical characters.

```

2897 \newbool{FV@breakafterinrun}
2898 \define@booleankey{FV}{breakafterinrun}%
2899 {\booltrue{FV@breakafterinrun}}%
```

```

2900 {\boolfalse{FV@breakafterinrun}}%
2901 \fvset{breakafterinrun=false}

```

### \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.

```

2902 \def\FV@BreakAfterPrep{%
2903   \ifx\FV@BreakAfter\@empty\relax
2904   \else
2905     \gdef\FV@BreakAfter@Def{}%
2906     \begingroup
2907     \def\FV@BreakAfter@Process##1##2\FV@Undefined{%
2908       \expandafter\FV@BreakAfter@Process@i\expandafter{##1}%
2909       \expandafter\ifx\expandafter\relax\detokenize{##2}\relax
2910       \else
2911         \FV@BreakAfter@Process##2\FV@Undefined
2912       \fi
2913     }%
2914     \def\FV@BreakAfter@Process@i##1{%
2915       \ifcsname FV@BreakBefore@Token\detokenize{##1}\endcsname
2916       \ifbool{FV@breakbeforeinrun}%
2917         {\ifbool{FV@breakafterinrun}%
2918          {}}%
2919         {\PackageError{fvextra}%
2920          {Conflicting breakbeforeinrun and breakafterinrun for "\detokenize{##1}"}%
2921          {Conflicting breakbeforeinrun and breakafterinrun for "\detokenize{##1}"}}%
2922         {\ifbool{FV@breakafterinrun}%
2923          {\PackageError{fvextra}%
2924           {Conflicting breakbeforeinrun and breakafterinrun for "\detokenize{##1}"}%
2925           {Conflicting breakbeforeinrun and breakafterinrun for "\detokenize{##1}"}}%
2926          {}}%
2927     \fi
2928     \g@addto@macro\FV@BreakAfter@Def{%
2929       \@namedef{FV@BreakAfter@Token\detokenize{##1}}{}}%
2930   }%
2931   \FV@EscChars
2932   \expandafter\FV@BreakAfter@Process\FV@BreakAfter\FV@Undefined
2933   \endgroup
2934   \FV@BreakAfter@Def
2935   \FV@BreakAfterPrep@PygmentsHook
2936 \fi
2937 }
2938 \def\FV@BreakAfterPrep@PygmentsHook{}

```

Now that `\FV@BreakBeforePrep` and `\FV@BreakAfterPrep` are defined, add them to `\FV@FormattingPrep@PreHook`, 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.



```

2939 \g@addto@macro\FV@FormattingPrep@PreHook{%
2940   \ifFV@pdfTeXinputenc
2941     \ifdefstring{\inputencodingname}{utf8}%
2942     {\let\FV@BreakBeforePrep\FV@BreakBeforePrep@UTF
2943      \let\FV@BreakAfterPrep\FV@BreakAfterPrep@UTF}%
2944     {}}%
2945 \fi
2946 \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.
2947 \define@key{FV}{breakanywheresymbolpre}{%
2948   \ifstrempy{#1}%
2949   {\def\FancyVerbBreakAnywhereSymbolPre{}}%
2950   {\def\FancyVerbBreakAnywhereSymbolPre{\hbox{#1}}}}
2951 \fvset{breakanywheresymbolpre={\,\footnotesize\ensuremath{\_ \rfloor}}}}

\FancyVerbBreakAnywhereSymbolPost
  The post-break symbol for breaks introduced by breakanywhere.
2952 \define@key{FV}{breakanywheresymbolpost}{%
2953   \ifstrempy{#1}%
2954   {\def\FancyVerbBreakAnywhereSymbolPost{}}%
2955   {\def\FancyVerbBreakAnywhereSymbolPost{\hbox{#1}}}}
2956 \fvset{breakanywheresymbolpost={}}

\FancyVerbBreakBeforeSymbolPre
  The pre-break symbol for breaks introduced by breakbefore.
2957 \define@key{FV}{breakbeforesymbolpre}{%
2958   \ifstrempy{#1}%
2959   {\def\FancyVerbBreakBeforeSymbolPre{}}%
2960   {\def\FancyVerbBreakBeforeSymbolPre{\hbox{#1}}}}
2961 \fvset{breakbeforesymbolpre={\,\footnotesize\ensuremath{\_ \rfloor}}}}

\FancyVerbBreakBeforeSymbolPost
  The post-break symbol for breaks introduced by breakbefore.
2962 \define@key{FV}{breakbeforesymbolpost}{%
2963   \ifstrempy{#1}%
2964   {\def\FancyVerbBreakBeforeSymbolPost{}}%
2965   {\def\FancyVerbBreakBeforeSymbolPost{\hbox{#1}}}}
2966 \fvset{breakbeforesymbolpost={}}

\FancyVerbBreakAfterSymbolPre
  The pre-break symbol for breaks introduced by breakafter.
2967 \define@key{FV}{breakaftersymbolpre}{%
2968   \ifstrempy{#1}%
2969   {\def\FancyVerbBreakAfterSymbolPre{}}%
2970   {\def\FancyVerbBreakAfterSymbolPre{\hbox{#1}}}}
2971 \fvset{breakaftersymbolpre={\,\footnotesize\ensuremath{\_ \rfloor}}}}

\FancyVerbBreakAfterSymbolPost
  The post-break symbol for breaks introduced by breakafter.
2972 \define@key{FV}{breakaftersymbolpost}{%
2973   \ifstrempy{#1}%
2974   {\def\FancyVerbBreakAfterSymbolPost{}}%
2975   {\def\FancyVerbBreakAfterSymbolPost{\hbox{#1}}}}

```



```

2976 \fvset{breakaftersymbolpost={}}
\FancyVerbBreakAnywhereBreak
    The macro governing breaking for breakanywhere=true.
2977 \newcommand{\FancyVerbBreakAnywhereBreak}{%
2978   \discretionary{\FancyVerbBreakAnywhereSymbolPre}%
2979   {\FancyVerbBreakAnywhereSymbolPost}{}}
\FancyVerbBreakBeforeBreak
    The macro governing breaking for breakbefore=true.
2980 \newcommand{\FancyVerbBreakBeforeBreak}{%
2981   \discretionary{\FancyVerbBreakBeforeSymbolPre}%
2982   {\FancyVerbBreakBeforeSymbolPost}{}}
\FancyVerbBreakAfterBreak
    The macro governing breaking for breakafter=true.
2983 \newcommand{\FancyVerbBreakAfterBreak}{%
2984   \discretionary{\FancyVerbBreakAfterSymbolPre}%
2985   {\FancyVerbBreakAfterSymbolPost}{}}
breaknonspaceingroup
FV@breaknonspaceingroup
    When inserting breaks, insert breaks within groups (typically { . . . } but depends
    on commandchars) instead of skipping over them. This isn't the default because it is
    incompatible with many macros since it inserts breaks into all arguments. For those
    cases, redefining macros to use \FancyVerbBreakStart . . . \FancyVerbBreakStop
    to insert breaks is better.
2986 \newbool{FV@breaknonspaceingroup}
2987 \define@booleankey{FV}{breaknonspaceingroup}%
2988   {\booltrue{FV@breaknonspaceingroup}}%
2989   {\boolfalse{FV@breaknonspaceingroup}}
2990 \fvset{breaknonspaceingroup=false}
breakpreferspaces
\FV@BreakHyphenation
    Adjust hyphenation settings for breaklines.
    When breakbefore, breakafter, or breakanywhere are in use, \finalhyphendemerits=0
    prevents the final "word" on a line from being put into a line segment by itself after
    line breaking. Otherwise, TEX tries to avoid a line break (hyphenation) within a
    "word" on the penultimate line segment, and may accomplish this by introducing
    an unnecessary line break at the space before the last "word."
    breakpreferspaces (\linepenalty) determines whether line breaks are prefer-
    entially inserted at normal spaces (breakcollapsespaces=true, showspaces=false)
    rather than at other locations allowed by breakbefore, breakafter, or breakanywhere.
2991 \newbool{FV@breakpreferspaces}
2992 \booltrue{FV@breakpreferspaces}
2993 \define@booleankey{FV}{breakpreferspaces}%
2994   {\booltrue{FV@breakpreferspaces}}%
2995   {\boolfalse{FV@breakpreferspaces}}
2996 \def\FV@BreakHyphenation{%
2997   \finalhyphendemerits=0\relax
2998   \ifbool{FV@breakpreferspaces}{\linepenalty=\@M\relax}}
2999 \g@addto@macro\FV@FormattingPrep@PreHook{\FV@BreakHyphenation}

```

## 12.12.2 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.

```
3000 \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.

```
3001 \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

```
3002 \let\FV@LineIndentChars\@empty
```

#### `\FV@GetLineIndent`

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

```
3003 \def\FV@GetLineIndent{%
3004   \@ifnextchar\FV@Sentinel
3005   {\FV@GetLineIndent@End}%
3006   {\ifx\@let@token\FV@FVSpaceToken
3007     \let\FV@Next\FV@GetLineIndent@Whitespace
3008     \else\ifx\@let@token\FV@FVTabToken
3009       \let\FV@Next\FV@GetLineIndent@Whitespace
3010       \else\ifcsname FV@PYG@Redefed\endcsname
3011         \ifx\@let@token\FV@PYG@Redefed
3012           \let\FV@Next\FV@GetLineIndent@Pygments
3013         \else
3014           \let\FV@Next\FV@GetLineIndent@End
3015         \fi
3016       \else
3017         \let\FV@Next\FV@GetLineIndent@End
3018       \fi\fi\fi
3019     \FV@Next}}
3020 \def\FV@GetLineIndent@End#1\FV@Sentinel{}
3021 \def\FV@GetLineIndent@Whitespace#1{%
3022   \expandafter\def\expandafter\FV@LineIndentChars\expandafter{\FV@LineIndentChars#1}%
3023   \FV@GetLineIndent}
3024 \def\FV@GetLineIndent@Pygments#1#2#3{%
3025   \FV@GetLineIndent#3}
```

### 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 12.10.4.) 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.

```
3026 \def\FV@TrueTab@UseWidth{%
3027   \@tempdima=\csname FV@TrueTab:Width\arabic{FV@TrueTabCounter}\endcsname sp\relax
3028   \stepcounter{FV@TrueTabCounter}%
3029   \hbox to\@tempdima{\hss\FV@TabChar}}
```

## Line scanning and break insertion macros

The strategy here is to scan through text token by token, inserting potential breaks at appropriate points. The final text with breaks inserted is stored in `\FV@BreakBuffer`, which is ultimately passed on to a wrapper macro like `\FancyVerbFormatText` or `\FancyVerbFormatInline`.

If user macros insert breaks via `\FancyVerbBreakStart... \FancyVerbBreakStop`, this invokes an additional scanning/insertion pass within each macro after expansion. The scanning/insertion only applies to the part of the expanded macros wrapped in `\FancyVerbBreakStart... \FancyVerbBreakStop`. At the time this occurs, during macro processing, text will already be wrapped in a wrapper macro like `\FancyVerbFormatText` or `\FancyVerbFormatInline`. That is, the built-in break insertion occurs before any typesetting, but user macro break insertion occurs during typesetting.

Token comparison is currently based on `\ifx`. This is sufficient for verbatim text but a comparison based on `\detokenize` might be better for cases when `commandchars` is in use. For example, with `commandchars` characters other than the curly braces `{}` might be the group tokens.

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, since with some settings breaks will interference with optional arguments.<sup>9</sup>

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

<sup>9</sup>Through 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.

to use `\let\FV@LastToken=...` so that `\let\FV@LastToken==` will work (so that the equals sign = won't break things).

### `FV@BreakBufferDepth`

Track buffer depth while inserting breaks. Some macros and command sequences require recursive processing. For example, groups `{...}` (with `commandchars` and `breaknonspaceingroup`), math, and nested `\FancyVerbBreakStart... \FancyVerbBreakStop`. Depth starts at zero. The current buffer at depth  $n$  is always `\FV@BreakBuffer`, with other buffers `\FV@BreakBuffer<n>` etc. named via `\csname` to allow for the integer.

```
3030 \newcounter{FV@BreakBufferDepth}
```

### `\FV@BreakBuffer@Append`

Append to `\FV@BreakBuffer`.

```
3031 \def\FV@BreakBuffer@Append#1{%
```

```
3032 \expandafter\def\expandafter\FV@BreakBuffer\expandafter{\FV@BreakBuffer#1}}
```

### `\FV@BreakBufferStart`

Create a new buffer, either at the beginning of scanning or during recursion. The single mandatory argument is the macro for handling tokens, which is `\let to \FV@Break@Token`. An intermediate `\FV@BreakBufferStart@i` is used to optimize `\ifx` comparisons for `\FV@BreakBufferStart` during scanning.

For recursion, `\FV@BreakBuffer<n>` and `\FV@Break@Token<n>` store the state (buffer and token handling macro) immediately prior to recursion with depth  $<n>$ .

```
3033 \def\FV@BreakBufferStart{%
```

```
3034 \FV@BreakBufferStart@i}
```

```
3035 \def\FV@BreakBufferStart@i#1{%
```

```
3036 \ifnum\value{FV@BreakBufferDepth}>0\relax
```

```
3037 \expandafter\let\csname FV@BreakBuffer\arabic{FV@BreakBufferDepth}\endcsname
```

```
3038 \FV@BreakBuffer
```

```
3039 \expandafter\let\csname FV@Break@Token\arabic{FV@BreakBufferDepth}\endcsname
```

```
3040 \FV@Break@Token
```

```
3041 \fi
```

```
3042 \def\FV@BreakBuffer{}%
```

```
3043 \let\FV@Break@Token=#1%
```

```
3044 \stepcounter{FV@BreakBufferDepth}%
```

```
3045 \let\FV@LastToken=\FV@Undefined
```

```
3046 \FV@Break@Scan}
```

### `FV@UserMacroBreaks`

Whether a user macro is inserting breaks, as opposed to `fvextra`'s standard scanning routine. When breaks come from `fvextra`, `\FV@BreakBufferStop` does nothing with `\FV@BreakBuffer` at buffer depth 0, since `\FV@InsertBreaks` handles buffer insertion. When breaks come from user macros, `\FV@BreakBufferStop` needs to insert `\FV@BreakBuffer` at buffer depth 0.

```
3047 \newbool{FV@UserMacroBreaks}
```

### `\FV@BreakBufferStop`

Complete the current buffer. The single mandatory argument is a wrapper macro for `\FV@BreakBuffer`'s contents (for example, insert recursively scanned group into braces `{...}`). If the mandatory argument is empty, no wrapper is used.

For `fvextra`'s standard scanning: If this is the main buffer (depth 0), stop scanning—which ultimately allows `\FV@BreakBuffer` to be handled by

`\FV@InsertBreaks`. For user macros: Insert `\FV@BreakBuffer` at buffer depth 0. Otherwise for both cases: Append the current buffer to the previous buffer, and continue scanning.

An intermediate `\FV@BreakBufferStop@i` is used to optimize `\ifx` comparisons for `\FV@BreakBufferStop` during scanning.

```

3048 \def\FV@BreakBufferStop{%
3049   \FV@BreakBufferStop@i}
3050 \def\FV@BreakBufferStop@i#1{%
3051   \addtocounter{FV@BreakBufferDepth}{-1}%
3052   \let\FV@LastToken=\FV@Undefined
3053   \ifnum\value{FV@BreakBufferDepth}<0\relax
3054     \PackageError{fvextra}%
3055       {Line break insertion error (extra \string\FancyVerbBreakStop?)}%
3056       {Line break insertion error (extra \string\FancyVerbBreakStop?)}%
3057     \def\FV@BreakBuffer{}%
3058     \fi
3059     \ifnum\value{FV@BreakBufferDepth}>0\relax
3060       \expandafter\@firstoftwo
3061     \else
3062       \expandafter\@secondoftwo
3063     \fi
3064     {\expandafter\FV@BreakBufferStop@ii\expandafter{\FV@BreakBuffer}{#1}}%
3065     {\ifbool{FV@UserMacroBreaks}%
3066       {\expandafter\let\expandafter\FV@BreakBuffer\expandafter\FV@Undefined\FV@BreakBuffer}%
3067       {}}}
3068 \def\FV@BreakBufferStop@ii#1#2{%
3069   \ifstrempy{#2}%
3070     {\FV@BreakBufferStop@iii{#1}}%
3071     {\expandafter\FV@BreakBufferStop@iii\expandafter{#2{#1}}}}
3072 \def\FV@BreakBufferStop@iii#1{%
3073   \expandafter\let\expandafter\FV@BreakBufferUpLevel
3074   \csname FV@BreakBuffer\arabic{FV@BreakBufferDepth}\endcsname
3075   \expandafter\def\expandafter\FV@BreakBuffer\expandafter{\FV@BreakBufferUpLevel#1}%
3076   \expandafter\let\expandafter\FV@Break@Token
3077   \csname FV@Break@Token\arabic{FV@BreakBufferDepth}\endcsname
3078   \FV@Break@Scan}

```

### `\FV@InsertBreaks`

This inserts breaks within text (`#2`) and stores the result in `\FV@BreakBuffer`. Then it invokes a macro (`#1`) on the result. That allows `\FancyVerbFormatInline` and `\FancyVerbFormatText` to operate on the final text (with breaks) directly, rather than being given text without breaks or text wrapped with macros that will (potentially recursively) insert breaks. (Breaks inserted by user macros are not yet present, though, since they are only inserted—potentially recursively—during macro processing.)

The initial `\ifx` skips break insertion when break insertion is turned off (`\FancyVerbBreakStart` is `\relax`).

The current definition of `\FV@Break@Token` is swapped for a UTF-8 compatible one under pdfTeX when necessary. In what follows, the default macros are defined after `\FV@Break`, since they make the algorithms simpler to understand. The more complex UTF variants are defined afterward.

```

3079 \def\FV@InsertBreaks#1#2{%
3080   \ifx\FancyVerbBreakStart\relax

```

```

3081 \expandafter\@firstoftwo
3082 \else
3083 \expandafter\@secondoftwo
3084 \fi
3085 {#1{#2}}%
3086 {\ifFV@pdfTeXinputenc
3087 \ifdefstring{\inputencodingname}{utf8}%
3088 {\ifx\FV@Break@DefaultToken\FV@Break@AnyToken
3089 \let\FV@Break@DefaultToken\FV@Break@AnyToken@UTF
3090 \else
3091 \ifx\FV@Break@DefaultToken\FV@Break@BeforeAfterToken
3092 \let\FV@Break@DefaultToken\FV@Break@BeforeAfterToken@UTF
3093 \fi
3094 \fi}%
3095 {}%
3096 \fi
3097 \setcounter{FV@BreakBufferDepth}{0}%
3098 \boolfalse{FV@UserMacroBreaks}%
3099 \FancyVerbBreakStart#2\FancyVerbBreakStop
3100 \setcounter{FV@BreakBufferDepth}{0}%
3101 \booltrue{FV@UserMacroBreaks}%
3102 \expandafter\FV@InsertBreaks@i\expandafter{\FV@BreakBuffer}{#1}}
3103 \def\FV@InsertBreaks@i#1#2{%
3104 \let\FV@BreakBuffer\FV@Undefined
3105 #2{#1}}

```

### **\FV@Break**

The entry macro for break insertion. Whatever is delimited (after expansion) by `\FV@Break... \FV@endBreak` will be scanned token by token/group by group, and accumulated (with any added breaks) in `\FV@BreakBuffer`. After scanning is complete, `\FV@BreakBuffer` will be inserted.

```

3106 \def\FV@Break{%
3107 \FV@BreakBufferStart{\FV@Break@DefaultToken}}

```

### **\FV@endBreak**

```

3108 \def\FV@endBreak{%
3109 \FV@BreakBufferStop{}}

```

### **\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.

The check against `\FV@BreakBufferStart` should typically not be necessary; it is included for completeness and to allow for future extensions and customization. `\FV@BreakBufferStart` is only inserted raw (rather than wrapped in `\FancyVerbBreakStart`) in token processing macros, where it initiates (or restarts) scanning and is not itself scanned.

```

3110 \begingroup
3111 \catcode`\$=3
3112 \gdef\FV@Break@Scan{%
3113 \@ifnextchar\FancyVerbBreakStart%
3114 {}%

```

```

3115 {\ifx\@let@token\FancyVerbBreakStop
3116   \let\FV@Break@Next\relax
3117   \else\ifx\@let@token\FV@Break@BufferStart
3118     \let\FV@Break@Next\relax
3119     \else\ifx\@let@token\FV@Break@BufferStop
3120       \let\FV@Break@Next\relax
3121       \else\ifx\@let@token$
3122         \let\FV@Break@Next\FV@Break@Math
3123         \else\ifx\@let@token\bgroup
3124           \let\FV@Break@Next\FV@Break@Group
3125           \else
3126             \let\FV@Break@Next\FV@Break@Token
3127             \fi\fi\fi\fi\fi
3128           \FV@Break@Next}}
3129 \endgroup

```

### `\FV@Break@Math`

Grab an entire math span, and insert it into `\FV@Break@Buffer`. Due to grouping, this works even when math contains things like `\text{${x$}}`. After dealing with the math span, continue scanning.

```

3130 \begingroup
3131 \catcode`\$=3%
3132 \gdef\FV@Break@Math$#1${%
3133   \FV@Break@BufferStart{\FV@Break@NBToken}#1\FV@Break@BufferStop{\FV@Break@MathTemplate}}
3134 \gdef\FV@Break@MathTemplate#1{#1$}
3135 \endgroup

```

### `\FV@Break@Group`

Grab the group, and insert it into `\FV@Break@Buffer` (as a group) before continuing scanning.

```

3136 \def\FV@Break@Group#1{%
3137   \ifstrempy{#1}%
3138     {\FV@Break@Buffer@Append{}}%
3139     \FV@Break@Scan}%
3140   {\ifbool{FV@breaknonspaceingroup}%
3141     {\FV@Break@BufferStart{\FV@Break@DefaultToken}%
3142       #1\FV@Break@BufferStop{\FV@Break@GroupTemplate}}%
3143     {\FV@Break@BufferStart{\FV@Break@NBToken}%
3144       #1\FV@Break@BufferStop{\FV@Break@GroupTemplate}}}}
3145 \def\FV@Break@GroupTemplate#1{#1}

```

### `\FV@Break@NBToken`

Append token to buffer while adding no breaks (NB) and reset last token.

```

3146 \def\FV@Break@NBToken#1{%
3147   \FV@Break@Buffer@Append{#1}%
3148   \let\FV@LastToken=\FV@Undefined
3149   \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%
    \FV@BreakBuffer@Append...
  \else
...
\endgroup

```

---

```

3150 \def\FV@Break@AnyToken#1{%
3151   \ifx\FV@FVSpaceToken#1\relax
3152     \expandafter\@firstoftwo
3153   \else
3154     \expandafter\@secondoftwo
3155   \fi
3156   {\let\FV@LastToken=#1\FV@BreakBuffer@Append{#1}\FV@Break@Scan}%
3157   {\ifx\FV@LastToken\FV@FVSpaceToken
3158     \expandafter\@firstoftwo
3159   \else
3160     \expandafter\@secondoftwo
3161   \fi
3162   {\let\FV@LastToken=#1%
3163     \FV@BreakBuffer@Append{#1}\FV@Break@Scan}%
3164   {\let\FV@LastToken=#1%
3165     \FV@BreakBuffer@Append{\FancyVerbBreakAnywhereBreak#1}\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.

The `\csname @let@token\endcsname` prevents issues if `\@let@token` is ever `\else` or `\fi`.

```

3166 \def\FV@Break@BeforeAfterToken#1{%
3167   \ifcsname FV@BreakBefore@Token\detokenize{#1}\endcsname
3168     \let\FV@Break@Next\FV@Break@BeforeTokenBreak
3169   \else
3170     \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
3171       \let\FV@Break@Next\FV@Break@AfterTokenBreak
3172     \else
3173       \let\FV@Break@Next\FV@Break@BeforeAfterTokenNoBreak
3174     \fi

```



```

3175 \fi
3176 \FV@Break@Next{#1}%
3177 }
3178 \def\FV@Break@BeforeAfterTokenNoBreak#1{%
3179 \FV@BreakBuffer@Append{#1}%
3180 \let\FV@LastToken=#1%
3181 \FV@Break@Scan}
3182 \def\FV@Break@BeforeTokenBreak#1{%
3183 \ifbool{FV@breakbeforeinrun}%
3184 {\ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
3185 \ifx#1\FV@FVSpaceToken
3186 \FV@BreakBuffer@Append{\FancyVerbSpaceBreak}%
3187 \else
3188 \FV@BreakBuffer@Append{\FancyVerbBreakBeforeBreak}%
3189 \fi
3190 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan
3191 \def\FV@RescanToken{#1}%
3192 \else
3193 \ifx#1\FV@FVSpaceToken
3194 \FV@BreakBuffer@Append{\FancyVerbSpaceBreak#1}%
3195 \else
3196 \FV@BreakBuffer@Append{\FancyVerbBreakBeforeBreak#1}%
3197 \fi
3198 \let\FV@Break@Next\FV@Break@Scan
3199 \let\FV@LastToken=#1%
3200 \fi}%
3201 {\ifx#1\FV@LastToken\relax
3202 \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
3203 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan
3204 \def\FV@RescanToken{#1}%
3205 \else
3206 \FV@BreakBuffer@Append{#1}%
3207 \let\FV@Break@Next\FV@Break@Scan
3208 \let\FV@LastToken=#1%
3209 \fi
3210 \else
3211 \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
3212 \ifx#1\FV@FVSpaceToken
3213 \FV@BreakBuffer@Append{\FancyVerbSpaceBreak}%
3214 \else
3215 \FV@BreakBuffer@Append{\FancyVerbBreakBeforeBreak}%
3216 \fi
3217 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan
3218 \def\FV@RescanToken{#1}%
3219 \else
3220 \ifx#1\FV@FVSpaceToken
3221 \FV@BreakBuffer@Append{\FancyVerbSpaceBreak#1}%
3222 \else
3223 \FV@BreakBuffer@Append{\FancyVerbBreakBeforeBreak#1}%
3224 \fi
3225 \let\FV@Break@Next\FV@Break@Scan
3226 \let\FV@LastToken=#1%
3227 \fi
3228 \fi}%

```

```

3229 \FV@Break@Next}
3230 \def\FV@Break@BeforeTokenBreak@AfterRescan{%
3231 \expandafter\FV@Break@AfterTokenBreak\FV@RescanToken}
3232 \def\FV@Break@AfterTokenBreak#1{%
3233 \let\FV@LastToken=#1%
3234 \@ifnextchar\FV@FVSpaceToken%
3235 {\ifx#1\FV@FVSpaceToken
3236 \expandafter\@firstoftwo
3237 \else
3238 \expandafter\@secondoftwo
3239 \fi
3240 {\FV@Break@AfterTokenBreak@i{#1}}%
3241 {\FV@Break@Buffer@Append{#1}%
3242 \FV@Break@Scan}}%
3243 {\FV@Break@AfterTokenBreak@i{#1}}}
3244 \def\FV@Break@AfterTokenBreak@i#1{%
3245 \ifbool{FV@breakafterinrun}%
3246 {\ifx#1\FV@FVSpaceToken
3247 \FV@Break@Buffer@Append{#1\FancyVerbSpaceBreak}%
3248 \else
3249 \FV@Break@Buffer@Append{#1\FancyVerbBreakAfterBreak}%
3250 \fi
3251 \let\FV@Break@Next\FV@Break@Scan}%
3252 {\ifx\@let@token#1\relax
3253 \FV@Break@Buffer@Append{#1}%
3254 \let\FV@Break@Next\FV@Break@Scan
3255 \else
3256 \expandafter\ifx\csname @let@token\endcsname\bgroup\relax
3257 \FV@Break@Buffer@Append{#1}%
3258 \let\FV@Break@Next\FV@Break@AfterTokenBreak@Group
3259 \else
3260 \ifx#1\FV@FVSpaceToken
3261 \FV@Break@Buffer@Append{#1\FancyVerbSpaceBreak}%
3262 \else
3263 \FV@Break@Buffer@Append{#1\FancyVerbBreakAfterBreak}%
3264 \fi
3265 \let\FV@Break@Next\FV@Break@Scan
3266 \fi
3267 \fi}%
3268 \FV@Break@Next
3269 }
3270 \def\FV@Break@AfterTokenBreak@Group#1{%
3271 \ifstrempy{#1}%
3272 {\FV@Break@Buffer@Append{}}%
3273 \@ifnextchar\FV@LastToken%
3274 {\FV@Break@Scan}%
3275 {\ifx\FV@LastToken\FV@FVSpaceToken
3276 \FV@Break@Buffer@Append{\FancyVerbSpaceBreak}%
3277 \else
3278 \FV@Break@Buffer@Append{\FancyVerbBreakAfterBreak}%
3279 \fi
3280 \FV@Break@Scan}}%
3281 {\ifx\FV@LastToken\FV@FVSpaceToken
3282 \FV@Break@Buffer@Append{\FancyVerbSpaceBreak}%

```

```

3283 \else
3284 \FV@BreakBuffer@Append{\FancyVerbBreakAfterBreak}%
3285 \fi
3286 \FV@Break@Group{#1}}

```

### 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.

Create macros conditionally:

```

3287 \ifFV@pdfTeXinputenc
\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@FormattingPrep@PreHook (checks are inserted into it after the non-UTF macro definitions).
3288 \def\FV@BreakBeforePrep@UTF{%
3289 \ifx\FV@BreakBefore@empty\relax
3290 \else
3291 \gdef\FV@BreakBefore@Def{%
3292 \begingroup
3293 \def\FV@BreakBefore@Process##1{%
3294 \ifcname FV@U8:\detokenize{##1}\endcsname
3295 \expandafter\let\expandafter\FV@Break@Next\cname FV@U8:\detokenize{##1}\endcsname
3296 \let\FV@UTF@octets@after\FV@BreakBefore@Process@ii
3297 \else
3298 \ifx##1\FV@Undefined
3299 \let\FV@Break@Next@gobble
3300 \else
3301 \let\FV@Break@Next\FV@BreakBefore@Process@i
3302 \fi
3303 \fi
3304 \FV@Break@Next##1%
3305 }%
3306 \def\FV@BreakBefore@Process@i##1{%
3307 \expandafter\FV@BreakBefore@Process@ii\expandafter{##1}}%
3308 \def\FV@BreakBefore@Process@ii##1{%
3309 \g@addto@macro\FV@BreakBefore@Def{%
3310 \@namedef{FV@BreakBefore@Token\detokenize{##1}}{}}%
3311 \FV@BreakBefore@Process
3312 }%
3313 \FV@EscChars
3314 \expandafter\FV@BreakBefore@Process\FV@BreakBefore\FV@Undefined
3315 \endgroup
3316 \FV@BreakBefore@Def
3317 \FV@BreakBeforePrep@PygmentsHook
3318 \fi

```

```

3319 }
\FV@BreakAfterPrep@UTF
3320 \def\FV@BreakAfterPrep@UTF{%
3321 \ifx\FV@BreakAfter\@empty\relax
3322 \else
3323 \gdef\FV@BreakAfter@Def{%
3324 \begingroup
3325 \def\FV@BreakAfter@Process##1{%
3326 \ifcsname FV@U8:\detokenize{##1}\endcsname
3327 \expandafter\let\expandafter\FV@Break@Next\csname FV@U8:\detokenize{##1}\endcsname
3328 \let\FV@UTF@octets@after\FV@BreakAfter@Process@ii
3329 \else
3330 \ifx##1\FV@Undefined
3331 \let\FV@Break@Next\@gobble
3332 \else
3333 \let\FV@Break@Next\FV@BreakAfter@Process@i
3334 \fi
3335 \fi
3336 \FV@Break@Next##1%
3337 }%
3338 \def\FV@BreakAfter@Process@i##1{%
3339 \expandafter\FV@BreakAfter@Process@ii\expandafter{##1}}%
3340 \def\FV@BreakAfter@Process@ii##1{%
3341 \ifcsname FV@BreakBefore@Token\detokenize{##1}\endcsname
3342 \ifbool{FV@breakbeforeinrun}%
3343 {\ifbool{FV@breakafterinrun}%
3344 {}%
3345 {\PackageError{fvextra}%
3346 {Conflicting breakbeforeinrun and breakafterinrun for "\detokenize{##1}"}%
3347 {Conflicting breakbeforeinrun and breakafterinrun for "\detokenize{##1}"}}}%
3348 {\ifbool{FV@breakafterinrun}%
3349 {\PackageError{fvextra}%
3350 {Conflicting breakbeforeinrun and breakafterinrun for "\detokenize{##1}"}%
3351 {Conflicting breakbeforeinrun and breakafterinrun for "\detokenize{##1}"}}}%
3352 {}}%
3353 \fi
3354 \g@addto@macro\FV@BreakAfter@Def{%
3355 \@namedef{FV@BreakAfter@Token\detokenize{##1}}{}}%
3356 \FV@BreakAfter@Process
3357 }%
3358 \FV@EscChars
3359 \expandafter\FV@BreakAfter@Process\FV@BreakAfter\FV@Undefined
3360 \endgroup
3361 \FV@BreakAfter@Def
3362 \FV@BreakAfterPrep@PygmentsHook
3363 \fi
3364 }
\FV@Break@AnyToken@UTF

```

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

```

3365 \def\FV@Break@AnyToken@UTF#1{%

```

```

3366 \ifcsname FV@U8:\detokenize{#1}\endcsname
3367 \expandafter\let\expandafter\FV@Break@Next\csname FV@U8:\detokenize{#1}\endcsname
3368 \let\FV@UTF@octets@after\FV@Break@AnyToken@UTF@i
3369 \else
3370 \let\FV@Break@Next\FV@Break@AnyToken@UTF@i
3371 \fi
3372 \FV@Break@Next{#1}%
3373 }
3374 \def\FV@Break@AnyToken@UTF@i#1{%
3375 \def\FV@CurrentToken{#1}%
3376 \ifx\FV@CurrentToken\FV@ActiveSpaceToken\relax
3377 \expandafter\@firstoftwo
3378 \else
3379 \expandafter\@secondoftwo
3380 \fi
3381 {\let\FV@LastToken\FV@CurrentToken
3382 \FV@BreakBuffer@Append{#1}\FV@Break@Scan}%
3383 {\ifx\FV@LastToken\FV@ActiveSpaceToken
3384 \expandafter\@firstoftwo
3385 \else
3386 \expandafter\@secondoftwo
3387 \fi
3388 {\let\FV@LastToken\FV@CurrentToken
3389 \FV@BreakBuffer@Append{#1}\FV@Break@Scan}%
3390 {\let\FV@LastToken\FV@CurrentToken
3391 \FV@BreakBuffer@Append{\FancyVerbBreakAnywhereBreak#1}\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.

```

3392 \def\FV@Break@BeforeAfterToken@UTF#1{%
3393 \ifcsname FV@U8:\detokenize{#1}\endcsname
3394 \expandafter\let\expandafter\FV@Break@Next\csname FV@U8:\detokenize{#1}\endcsname
3395 \let\FV@UTF@octets@after\FV@Break@BeforeAfterToken@UTF@i
3396 \else
3397 \let\FV@Break@Next\FV@Break@BeforeAfterToken@UTF@i
3398 \fi
3399 \FV@Break@Next{#1}%
3400 }
3401 \def\FV@Break@BeforeAfterToken@UTF@i#1{%
3402 \ifcsname FV@BreakBefore@Token\detokenize{#1}\endcsname
3403 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@UTF
3404 \else
3405 \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
3406 \let\FV@Break@Next\FV@Break@AfterTokenBreak@UTF
3407 \else
3408 \let\FV@Break@Next\FV@Break@BeforeAfterTokenNoBreak@UTF
3409 \fi

```

```

3410 \fi
3411 \FV@Break@Next{#1}%
3412 }
3413 \def\FV@Break@BeforeAfterTokenNoBreak@UTF#1{%
3414 \FV@Break@Buffer@Append{#1}%
3415 \def\FV@LastToken{#1}%
3416 \FV@Break@Scan}
3417 \def\FV@Break@BeforeTokenBreak@UTF#1{%
3418 \def\FV@CurrentToken{#1}%
3419 \ifbool{FV@breakbeforeinrun}%
3420 {\ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
3421 \ifx\FV@CurrentToken\FV@ActiveSpaceToken
3422 \FV@Break@Buffer@Append{\FancyVerbSpaceBreak}%
3423 \else
3424 \FV@Break@Buffer@Append{\FancyVerbBreakBeforeBreak}%
3425 \fi
3426 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan@UTF
3427 \def\FV@RescanToken{#1}%
3428 \else
3429 \ifx\FV@CurrentToken\FV@ActiveSpaceToken
3430 \FV@Break@Buffer@Append{\FancyVerbSpaceBreak#1}%
3431 \else
3432 \FV@Break@Buffer@Append{\FancyVerbBreakBeforeBreak#1}%
3433 \fi
3434 \let\FV@Break@Next\FV@Break@Scan
3435 \def\FV@LastToken{#1}%
3436 \fi}%
3437 {\ifx\FV@CurrentToken\FV@LastToken\relax
3438 \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
3439 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan@UTF
3440 \def\FV@RescanToken{#1}%
3441 \else
3442 \FV@Break@Buffer@Append{#1}%
3443 \let\FV@Break@Next\FV@Break@Scan
3444 \def\FV@LastToken{#1}%
3445 \fi
3446 \else
3447 \ifcsname FV@BreakAfter@Token\detokenize{#1}\endcsname
3448 \ifx\FV@CurrentToken\FV@ActiveSpaceToken
3449 \FV@Break@Buffer@Append{\FancyVerbSpaceBreak}%
3450 \else
3451 \FV@Break@Buffer@Append{\FancyVerbBreakBeforeBreak}%
3452 \fi
3453 \let\FV@Break@Next\FV@Break@BeforeTokenBreak@AfterRescan@UTF
3454 \def\FV@RescanToken{#1}%
3455 \else
3456 \ifx\FV@CurrentToken\FV@ActiveSpaceToken
3457 \FV@Break@Buffer@Append{\FancyVerbSpaceBreak#1}%
3458 \else
3459 \FV@Break@Buffer@Append{\FancyVerbBreakBeforeBreak#1}%
3460 \fi
3461 \let\FV@Break@Next\FV@Break@Scan
3462 \def\FV@LastToken{#1}%
3463 \fi

```

```

3464     \fi}%
3465   \FV@Break@Next}
3466 \def\FV@Break@BeforeTokenBreak@AfterRescan@UTF{%
3467   \expandafter\FV@Break@AfterTokenBreak@UTF\expandafter{\FV@RescanToken}}
3468 \def\FV@Break@AfterTokenBreak@UTF#1{%
3469   \def\FV@LastToken{#1}%
3470   \@ifnextchar\FV@FVSpaceToken%
3471   {\ifx\FV@LastToken\FV@ActiveSpaceToken
3472     \expandafter\@firstoftwo
3473   \else
3474     \expandafter\@secondoftwo
3475   \fi
3476   {\FV@Break@AfterTokenBreak@UTF@i{#1}}%
3477   {\FV@Break@Buffer@Append{#1}}%
3478   \FV@Break@Scan}}%
3479   {\FV@Break@AfterTokenBreak@UTF@i{#1}}}}
3480 \def\FV@Break@AfterTokenBreak@UTF@i#1{%
3481   \ifbool{FV@breakafterinrun}%
3482   {\ifx\FV@LastToken\FV@ActiveSpaceToken
3483     \FV@Break@Buffer@Append{#1\FancyVerbSpaceBreak}%
3484   \else
3485     \FV@Break@Buffer@Append{#1\FancyVerbBreakAfterBreak}%
3486   \fi
3487   \let\FV@Break@Next\FV@Break@Scan}%
3488   {\FV@Break@Buffer@Append{#1}}%
3489   \expandafter\ifx\csname @let@token\endcsname\bgroup\relax
3490   \let\FV@Break@Next\FV@Break@AfterTokenBreak@Group@UTF
3491   \else
3492     \let\FV@Break@Next\FV@Break@AfterTokenBreak@UTF@ii
3493   \fi}%
3494   \FV@Break@Next}
3495 \def\FV@Break@AfterTokenBreak@UTF@ii#1{%
3496   \ifcsname FV@U8:\detokenize{#1}\endcsname
3497     \expandafter\let\expandafter\FV@Break@Next\csname FV@U8:\detokenize{#1}\endcsname
3498     \let\FV@UTF@octets@after\FV@Break@AfterTokenBreak@UTF@ii
3499   \else
3500     \def\FV@NextToken{#1}%
3501     \ifx\FV@LastToken\FV@NextToken
3502     \else
3503       \ifx\FV@LastToken\FV@ActiveSpaceToken
3504         \FV@Break@Buffer@Append{\FancyVerbSpaceBreak}%
3505       \else
3506         \FV@Break@Buffer@Append{\FancyVerbBreakAfterBreak}%
3507       \fi
3508     \fi
3509     \let\FV@Break@Next\FV@Break@Scan
3510   \fi
3511   \FV@Break@Next#1}
3512 \def\FV@Break@AfterTokenBreak@Group@UTF#1{%
3513   \ifstrempy{#1}%
3514   {\FV@Break@Buffer@Append{}}%
3515   \@ifnextchar\bgroup
3516   {\ifx\FV@LastToken\FV@ActiveSpaceToken
3517     \FV@Break@Buffer@Append{\FancyVerbSpaceBreak}%

```

```

3518     \else
3519         \FV@BreakBuffer@Append{\FancyVerbBreakAfterBreak}%
3520     \fi
3521     \FV@Break@Group}%
3522     {\FV@Break@AfterTokenBreak@Group@UTF@i}}%
3523     {\ifx\FV@LastToken\FV@ActiveSpaceToken
3524         \FV@BreakBuffer@Append{\FancyVerbSpaceBreak}%
3525     \else
3526         \FV@BreakBuffer@Append{\FancyVerbBreakAfterBreak}%
3527     \fi
3528     \FV@Break@Group{#1}}}
3529 \def\FV@Break@AfterTokenBreak@Group@UTF@i#1{%
3530 \ifcsname FV@U8:\detokenize{#1}\endcsname
3531     \expandafter\let\expandafter\FV@Break@Next\csname FV@U8:\detokenize{#1}\endcsname
3532     \let\FV@UTF@octets@after\FV@Break@AfterTokenBreak@Group@UTF@i
3533 \else
3534     \def\FV@NextToken{#1}%
3535     \ifx\FV@LastToken\FV@NextToken
3536     \else
3537         \ifx\FV@LastToken\FV@ActiveSpaceToken
3538             \FV@BreakBuffer@Append{\FancyVerbSpaceBreak}%
3539         \else
3540             \FV@BreakBuffer@Append{\FancyVerbBreakAfterBreak}%
3541         \fi
3542     \fi
3543     \let\FV@Break@Next\FV@Break@Scan
3544 \fi
3545 \FV@Break@Next#1}

```

End the conditional creation of the pdfTeX UTF macros:

```

3546 \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`.

```

3547 \def\FV@makeLineNumber{%
3548     \hss
3549     \FancyVerbBreakSymbolLeftLogic{\FancyVerbBreakSymbolLeft}%
3550     \hbox to \FV@BreakSymbolSepLeft{\hfill}%
3551     \rlap{\hskip\linewidth
3552         \hbox to \FV@BreakSymbolSepRight{\hfill}%
3553         \FancyVerbBreakSymbolRightLogic{\FancyVerbBreakSymbolRight}%

```



```

3554 \FV@SetLineBreakLast
3555 }%
3556 }

```

### \FV@RaggedRight

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

```

3557 \def\FV@RaggedRight{%
3558 \let\\@centercr
3559 \@rightskip\@flushglue\rightskip\@rightskip\leftskip\z@skip\parindent\z@}

```

### \FV@LineWidth

This is the effective line width within a broken line.

```

3560 \newdimen\FV@LineWidth

```

### \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 `\FV@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.

```

3561 \def\FV@SaveLineBox#1{%
3562 \savebox{\FV@LineBox}{%
3563 \advance\FV@LineWidth by -\FV@BreakIndent
3564 \hbox to \FV@BreakIndent{\hfill}%
3565 \ifbool{FV@breakautoindent}%
3566 {\let\FV@LineIndentChars\@empty
3567 \FV@GetLineIndent#1\FV@Sentinel
3568 \savebox{\FV@LineIndentBox}{\FV@LineIndentChars}%
3569 \hbox to \wd\FV@LineIndentBox{\hfill}%
3570 \advance\FV@LineWidth by -\wd\FV@LineIndentBox
3571 \setcounter{FV@TrueTabCounter}{0}}%
3572 }%
3573 \ifdefempty{\FancyVerbBreakSymbolLeft}{%
3574 {\hbox to \FV@BreakSymbolIndentLeft{\hfill}%
3575 \advance\FV@LineWidth by -\FV@BreakSymbolIndentLeft}%
3576 \ifdefempty{\FancyVerbBreakSymbolRight}{%
3577 {\advance\FV@LineWidth by -\FV@BreakSymbolIndentRight}%
3578 \parbox[t]{\FV@LineWidth}{%
3579 \FV@RaggedRight
3580 \leftlinenumbers*
3581 \begin{internallinenumbers*}%
3582 \let\makeLineNumber\FV@makeLineNumber
3583 \noindent\hspace*{-\FV@BreakIndent}%

```

```

3584     \ifdefempty{\FancyVerbBreakSymbolLeft}{}{-%
3585         \hspace*{-\FV@BreakSymbolIndentLeft}}}%
3586     \ifbool{FV@breakautoindent}%
3587         {\hspace*{-\wd\FV@LineIndentBox}}}%
3588         {}%
3589     \FV@BreakByTokenAnywhereHook
3590     \strut\FV@InsertBreaks{\FancyVerbFormatText}{#1}\nobreak\strut
3591     \end{internallinenumbers*}
3592 }%
3593 \ifdefempty{\FancyVerbBreakSymbolRight}{}{-%
3594     {\hbox to \FV@BreakSymbolIndentRight{\hfill}}}%
3595 }%
3596 }
3597 \def\FV@BreakByTokenAnywhereHook{}

```

#### `\FV@ListProcessLine@Break`

This macro is based on the original `\FV@ListProcessLine` and follows it as closely as possible. `\FV@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.

```

3598 \expandafter\let\expandafter\FV@iffp\csname fp_compare:nNnTF\endcsname
3599 \def\FV@ListProcessLine@Break#1{%
3600     \hbox to \hsize{%
3601         \kern\leftmargin
3602         \hbox to \linewidth{%
3603             \FV@LineWidth\linewidth
3604             \ifx\FV@RightListFrame\relax\else
3605                 \advance\FV@LineWidth by -\FV@FrameSep
3606                 \advance\FV@LineWidth by -\FV@FrameRule
3607             \fi
3608             \ifx\FV@LeftListFrame\relax\else
3609                 \advance\FV@LineWidth by -\FV@FrameSep
3610                 \advance\FV@LineWidth by -\FV@FrameRule
3611             \fi
3612             \ifx\FV@Tab\FV@TrueTab
3613                 \let\FV@TrueTabSaveWidth\FV@TrueTabSaveWidth@Save
3614                 \setcounter{FV@TrueTabCounter}{0}%
3615             \fi
3616             \sbox{\FV@LineBox}{%
3617                 \let\FancyVerbBreakStart\relax
3618                 \let\FancyVerbBreakStop\relax

```

```

3619 \FancyVerbFormatLine{%
3620   \FancyVerbHighlightLine %<-- Default definition using \rlap breaks breaking
3621   {\FV@ObeyTabs{\FancyVerbFormatText{#1}}}}}%
3622 \ifx\FV@Tab\FV@TrueTab
3623   \let\FV@TrueTabSaveWidth\relax
3624 \fi
3625 \ifnum
3626   \FV@iffp{\the\wd\FV@LineBox}>{\FV@LineWidth}%
3627   {1}% width greater than line
3628   {\FV@iffp{\the\wd\FV@LineBox}<{0}{1}{0}}% width overflows
3629   >0
3630 \relax
3631 \setcounter{FancyVerbLineBreakLast}{0}%
3632 \ifx\FV@Tab\FV@TrueTab
3633   \let\FV@Tab\FV@TrueTab@UseWidth
3634   \setcounter{FV@TrueTabCounter}{0}%
3635 \fi
3636 \FV@SaveLineBox{#1}%
3637 \ifdefempty{\FancyVerbBreakSymbolRight}{-}{%
3638   \let\FV@SetLineBreakLast\relax
3639   \setcounter{FV@TrueTabCounter}{0}%
3640   \FV@SaveLineBox{#1}}}%
3641 \FV@LeftListNumber
3642 \FV@LeftListFrame
3643 \FV@BGColor@List{%
3644   \FancyVerbFormatLine{%
3645     \FancyVerbHighlightLine{\usebox{\FV@LineBox}}}}}%
3646 \FV@RightListFrame
3647 \FV@RightListNumber
3648 \ifx\FV@Tab\FV@TrueTab@UseWidth
3649   \let\FV@Tab\FV@TrueTab
3650 \fi
3651 \else
3652   \let\FancyVerbBreakStart\relax
3653   \let\FancyVerbBreakStop\relax
3654   \FV@LeftListNumber
3655   \FV@LeftListFrame
3656   \FV@BGColor@List{%
3657     \FancyVerbFormatLine{%
3658       \FancyVerbHighlightLine{%
3659         \parbox[t]{\FV@LineWidth}{%
3660           \noindent\strut\FV@ObeyTabs{\FancyVerbFormatText{#1}}\strut}}}}}%
3661   \FV@RightListFrame
3662   \FV@RightListNumber
3663 \fi}%
3664 \hss}\FV@bgcoloroverlap\baselineskip\z@\lineskip\z@}

```

## 12.13 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.

```

3665 \def\FV@BreakBeforePrep@Pygments#1{%
3666   \ifcsname FV@BreakBefore@Token@backslashchar\endcsname
3667     \@namedef{FV@BreakBefore@Token#1Zbs }{ }%
3668   \fi
3669   \ifcsname FV@BreakBefore@Token\FV@underscorechar\endcsname
3670     \@namedef{FV@BreakBefore@Token#1Zus }{ }%
3671   \fi
3672   \ifcsname FV@BreakBefore@Token@charlb\endcsname
3673     \@namedef{FV@BreakBefore@Token#1Zob }{ }%
3674   \fi
3675   \ifcsname FV@BreakBefore@Token@charrb\endcsname
3676     \@namedef{FV@BreakBefore@Token#1Zcb }{ }%
3677   \fi
3678   \ifcsname FV@BreakBefore@Token\detokenize{^}\endcsname
3679     \@namedef{FV@BreakBefore@Token#1Zca }{ }%
3680   \fi
3681   \ifcsname FV@BreakBefore@Token\FV@ampchar\endcsname
3682     \@namedef{FV@BreakBefore@Token#1Zam }{ }%
3683   \fi
3684   \ifcsname FV@BreakBefore@Token\detokenize{<}\endcsname
3685     \@namedef{FV@BreakBefore@Token#1Zlt }{ }%
3686   \fi
3687   \ifcsname FV@BreakBefore@Token\detokenize{>}\endcsname
3688     \@namedef{FV@BreakBefore@Token#1Zgt }{ }%
3689   \fi
3690   \ifcsname FV@BreakBefore@Token\FV@hashchar\endcsname
3691     \@namedef{FV@BreakBefore@Token#1Zsh }{ }%
3692   \fi
3693   \ifcsname FV@BreakBefore@Token@percentchar\endcsname
3694     \@namedef{FV@BreakBefore@Token#1Zpc }{ }%
3695   \fi
3696   \ifcsname FV@BreakBefore@Token\FV@dollarchar\endcsname
3697     \@namedef{FV@BreakBefore@Token#1Zdl }{ }%
3698   \fi
3699   \ifcsname FV@BreakBefore@Token\detokenize{-}\endcsname
3700     \@namedef{FV@BreakBefore@Token#1Zhy }{ }%
3701   \fi
3702   \ifcsname FV@BreakBefore@Token\detokenize{'}\endcsname
3703     \@namedef{FV@BreakBefore@Token#1Zsq }{ }%
3704   \fi
3705   \ifcsname FV@BreakBefore@Token\detokenize{"}\endcsname
3706     \@namedef{FV@BreakBefore@Token#1Zdq }{ }%
3707   \fi

```

```

3708 \ifcsname FV@BreakBefore@Token\FV@tildechar\endcsname
3709 \namedef{FV@BreakBefore@Token#1Zti }{-}%
3710 \fi
3711 \ifcsname FV@BreakBefore@Token\detokenize{@}\endcsname
3712 \namedef{FV@BreakBefore@Token#1Zat }{-}%
3713 \fi
3714 \ifcsname FV@BreakBefore@Token\detokenize{[]}\endcsname
3715 \namedef{FV@BreakBefore@Token#1Zlb }{-}%
3716 \fi
3717 \ifcsname FV@BreakBefore@Token\detokenize{[]}\endcsname
3718 \namedef{FV@BreakBefore@Token#1Zrb }{-}%
3719 \fi
3720 }

```

### \FV@BreakAfterPrep@Pygments

```

3721 \def\FV@BreakAfterPrep@Pygments#1{%
3722 \ifcsname FV@BreakAfter@Token\@backslashchar\endcsname
3723 \namedef{FV@BreakAfter@Token#1Zbs }{-}%
3724 \fi
3725 \ifcsname FV@BreakAfter@Token\FV@underscorechar\endcsname
3726 \namedef{FV@BreakAfter@Token#1Zus }{-}%
3727 \fi
3728 \ifcsname FV@BreakAfter@Token\@charlb\endcsname
3729 \namedef{FV@BreakAfter@Token#1Zob }{-}%
3730 \fi
3731 \ifcsname FV@BreakAfter@Token\@charrb\endcsname
3732 \namedef{FV@BreakAfter@Token#1Zcb }{-}%
3733 \fi
3734 \ifcsname FV@BreakAfter@Token\detokenize{^}\endcsname
3735 \namedef{FV@BreakAfter@Token#1Zca }{-}%
3736 \fi
3737 \ifcsname FV@BreakAfter@Token\FV@ampchar\endcsname
3738 \namedef{FV@BreakAfter@Token#1Zam }{-}%
3739 \fi
3740 \ifcsname FV@BreakAfter@Token\detokenize{<}\endcsname
3741 \namedef{FV@BreakAfter@Token#1Zlt }{-}%
3742 \fi
3743 \ifcsname FV@BreakAfter@Token\detokenize{>}\endcsname
3744 \namedef{FV@BreakAfter@Token#1Zgt }{-}%
3745 \fi
3746 \ifcsname FV@BreakAfter@Token\FV@hashchar\endcsname
3747 \namedef{FV@BreakAfter@Token#1Zsh }{-}%
3748 \fi
3749 \ifcsname FV@BreakAfter@Token\@percentchar\endcsname
3750 \namedef{FV@BreakAfter@Token#1Zpc }{-}%
3751 \fi
3752 \ifcsname FV@BreakAfter@Token\FV@dollarchar\endcsname
3753 \namedef{FV@BreakAfter@Token#1Zdl }{-}%
3754 \fi
3755 \ifcsname FV@BreakAfter@Token\detokenize{-}\endcsname
3756 \namedef{FV@BreakAfter@Token#1Zhy }{-}%
3757 \fi
3758 \ifcsname FV@BreakAfter@Token\detokenize{'}\endcsname
3759 \namedef{FV@BreakAfter@Token#1Zsq }{-}%

```

```

3760 \fi
3761 \ifcsname FV@BreakAfter@Token\detokenize{"}\endcsname
3762 \namedef{FV@BreakAfter@Token#1Zdq }{ }%
3763 \fi
3764 \ifcsname FV@BreakAfter@Token\FV@tildechar\endcsname
3765 \namedef{FV@BreakAfter@Token#1Zti }{ }%
3766 \fi
3767 \ifcsname FV@BreakAfter@Token\detokenize{@}\endcsname
3768 \namedef{FV@BreakAfter@Token#1Zat }{ }%
3769 \fi
3770 \ifcsname FV@BreakAfter@Token\detokenize{[]}\endcsname
3771 \namedef{FV@BreakAfter@Token#1Zlb }{ }%
3772 \fi
3773 \ifcsname FV@BreakAfter@Token\detokenize{[]}\endcsname
3774 \namedef{FV@BreakAfter@Token#1Zrb }{ }%
3775 \fi
3776 }

```

### 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.

```

3777 \newbool{FV@breakbytoken}
3778 \define@booleankey{FV}{breakbytoken}%
3779 {\booltrue{FV@breakbytoken}}%
3780 {\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`.

```

3781 \newbool{FV@breakbytokenanywhere}
3782 \define@booleankey{FV}{breakbytokenanywhere}%
3783 {\booltrue{FV@breakbytokenanywhere}\booltrue{FV@breakbytoken}}%
3784 {\boolfalse{FV@breakbytokenanywhere}\boolfalse{FV@breakbytoken}}

```

### \FancyVerbBreakByTokenAnywhereBreak

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

```

3785 \def\FancyVerbBreakByTokenAnywhereBreak{\allowbreak{}}

```

### texcomments

`texcomments` retokenizes the contents of Pygments comment tokens using the catcodes in place before the verbatim context.

```

3786 \newbool{FV@texcomments}
3787 \define@booleankey{FV}{texcomments}%
3788 {\booltrue{FV@texcomments}}%
3789 {\boolfalse{FV@texcomments}}

```

### \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` for `minted`, `\PYG<style_name>` for `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@FormattingPrep@PreHook`, to make the actual changes at the appropriate time.

```
3790 \def\VerbatimPygments#1#2{%
3791   \FV@AddToHook{\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 `\ifcsname`, because `#2` will be a macro, and will typically be created with `\csname... \endcsname` which will `\let` the macro to `\relax` if it doesn’t already exist.

`\FV@PYG@Redefed` is `\let` to the Pygments macro that appears literally (after redefinition), so that it can be detected elsewhere to allow for special processing, such as in `breakautoindent`.

Escape (`esc`) tokens are always retokenized. Comment (`c`) tokens are retokenized if the `texcomments` option is true. Reference for token abbreviations: <https://github.com/pygments/pygments/blob/master/pygments/token.py>.

```
3792 \def\FV@VerbatimPygments#1#2{%
3793   \ifx#2\relax
3794     \let\FV@PYG@orig=#1\relax
3795   \else
3796     \let\FV@PYG@orig=#2\relax
3797   \fi
3798   \let#1\FV@PYG
```

```

3799 \ifbool{FV@breaklines}%
3800 {\edef\FV@PYG@name{\expandafter\FV@DetokMacro@StripSpace\detokenize{#1}}%
3801 \def\FV@BreakBeforePrep@PygmentsHook{%
3802 \expandafter\FV@BreakBeforePrep@Pygments\expandafter{\FV@PYG@name}}%
3803 \def\FV@BreakAfterPrep@PygmentsHook{%
3804 \expandafter\FV@BreakAfterPrep@Pygments\expandafter{\FV@PYG@name}}%
3805 \ifbool{FV@breakbytoken}%
3806 {\ifbool{FV@breakbytokenanywhere}%
3807 {\def\FV@BreakByTokenAnywhereHook{%
3808 \def\FV@BreakByTokenAnywhereBreak{%
3809 \let\FV@BreakByTokenAnywhereBreak\FancyVerbBreakByTokenAnywhereBreak}}%
3810 \def\FV@PYG@retoked##1##2{%
3811 \FV@BreakByTokenAnywhereBreak
3812 \leavevmode\hbox{\FV@PYG@orig{##1}{##2}}}%
3813 {\def\FV@PYG@retoked##1##2{%
3814 \leavevmode\hbox{\FV@PYG@orig{##1}{##2}}}}%
3815 {\def\FV@PYG@retoked##1##2{%
3816 \ifbool{FV@PYG@doretok}%
3817 {\FV@PYG@orig{##1}{##2}}%
3818 {\FV@PYG@orig{##1}{\FancyVerbBreakStart##2\FancyVerbBreakStop}}}}}%
3819 {\let\FV@PYG@retoked\FV@PYG@orig}%
3820 \let\FV@PYG@Redefed\FV@PYG}
3821 \let\FV@BreakByTokenAnywhereBreak\relax
3822 \def\FV@DetokMacro@StripSpace#1 {#1}
3823 \newbool{FV@PYG@doretok}
3824 \begingroup
3825 \catcode\+=12
3826 \gdef\FV@PYG@setretok#1+{%
3827 \ifx\relax#1\@empty
3828 \else
3829 \ifstrequal{#1}{c}%
3830 {\ifbool{FV@texcomments}{\booltrue{FV@PYG@doretok}}{}}%
3831 {\ifstrequal{#1}{esc}{\booltrue{FV@PYG@doretok}}{}}%
3832 \expandafter\FV@PYG@setretok
3833 \fi}
3834 \endgroup
3835 \def\FV@PYG#1#2{%
3836 \expandafter\FV@PYG@i\expandafter{\detokenize{#1}}{#2}}
3837 \def\FV@PYG@i#1#2{%
3838 \begingroup
3839 \boolfalse{FV@PYG@doretok}%
3840 \FV@PYG@setretok#1+\relax+%
3841 \ifbool{FV@PYG@doretok}%
3842 {\FVExtraDetokenizeVArg{%
3843 \FVExtraRetokenizeVArg{\FV@PYG@retoked{#1}}{\FancyVerbRestoreCodes}}{#2}}%
3844 {\FV@PYG@retoked{#1}{#2}}%
3845 \endgroup}%

```