The phfcc package[1]

Philippe Faist *philippe.faist@bluewin.ch*

October 06, 2021

[1] *This document corresponds to phfcc v2.0, dated 2021/10/06. It is part of the phfqitltx package suite, see https://github.com/phfaist/phfqitltx.*

phfcc—A handy LaTeX package for inline commenting in collaborative LaTeX documents.

1 Introduction

When elaborating documents, especially collaborative documents with multiple authors, it is useful to leave inline comments and be able to mark text changes.

Often, authors will choose a text color and define a simple macro, like this:

```
\newcommand{\phf}[1]{{\color{blue}Ph.F.: #1}} % Philippe Faist's comments
```

This rudimentary approach has a few drawbacks. The main drawback is when highlighting long blocks of text. First of all, it can be hard to visually find the matching closing brace in your editor. Then, the synchronization between your viewer and the TeX source (`synctex`) will no longer point to the correct source line within the highlighted block of text; you will always be pointed at the beginning or the end of the entire block. The reason for this behavior is that the entire text is passed as a macro argument to `\phf`, so `synctex` only sees a single macro call. The fact that a long block of text is passed as a macro argument can also lead to other issues, such as errors with `\verb+...+`-type constructs.

Also, it is occasionally desirable to make the difference between sections of text that were added by an author, and an inline off-hand remark, which is not part of the text itself. For instance, with the above macro, we could enclose remarks in "`[`" and "`]`":

```
... \phf{Observe that $A\geq 0$ by definition.}
\phf{[I added this sentence because I felt that it wasn't
clear from the context that $A\geq 0$]} ...
```

to obtain:

... *(Ph.F.:) Observe that* $A \geq 0$ *by definition. (Ph.F.:) [I added this sentence because I felt that it wasn't clear from the context that* $A \geq 0$*]* ...

Observe how the off-hand remark blends with the text. Wouldn't it be nicer if it stood out clearly, and if the initials were perhaps *inside* the brackets, like so?

... *Ph.F: Observe that* $A \geq 0$ *by definition.* ***[(Ph.F.:) I added this sentence because I felt that it wasn't clear from the context that*** $A \geq 0$***]*** ...

Of course, we could have defined a different macro for the off-hand remark, say `\phfremark`. But typing a different macro name is not convenient, Wouldn't it be nicer if the off-hand remark could be typed with the syntax `\phf[...]`, using square brackets?

Also the initials don't have to be repeated at every macro call, it suffice to associate the color to the author perhaps once per page or so. They could also be displayed in the margin, to avoid cluttering the text.

Furthermore, to avoid the aforementioned issues with long blocks of text, we could imagine marking text with the syntax `\begin{phf} ... \end{phf}` or even `\phf ... \endphf`.

And while we're at it, wouldn't it be cool to mark text removals, as well? We could use the syntax `\phf*{text to be removed}`.

And because I definitely didn't have time to spare but still got distracted into programming LaTeX macros to implement the above features, I have the pleasure to present:

Introducing . . . the phfcc package.

With phfcc, you simply define your multi-functional commenting command using the `\phfMakeCommentingCommand` macro.

Let's start with a simple example. Suppose two authors, Albert Einstein and John Bell, are commenting on the same document. They might define the commenting commands `\AlE` and `\JB`:

```
\phfMakeCommentingCommand[initials={A.E.}]{AlE}
\phfMakeCommentingCommand[initials={J.B.}]{JB}
```

Suppose they edit a paragraph of text, with multiple rounds of editing by both authors; they mark their latest changes while leaving comments for the other author as follows:

```
Quantum mechanics developed quickly in the 1920's thanks to the
works of Schr\"odinger, Heisenberg, and Dirac, who drew inspiration
from earlier ideas by Planck \AlE{and Einstein} \AlE[@JB, you forgot
about me!].  Quantum mechanics appears to allow spooky action at a
distance via some strange thing called ``entanglement,'' \JB*{but
this can be explained with hidden variables}\JB{which has
revolutionized our understanding of local realism.} \JB[@Albert,
we talked about this...]
```

which gives:

Quantum mechanics developed quickly in the 1920's thanks to the works of Schrödinger, Heisenberg, and Dirac, who drew inspiration from earlier ideas by
A.E. *Planck and Einstein* [@JB, you forgot about me!]. *Quantum mechanics appears to allow spooky action at a distance via some strange thing called "entanglement,"* ~~*but this can be explained with hidden variables*~~ *which has revolutionized our*
J.B. *understanding of local realism.* [@Albert, we talked about this...]

Read on to learn more about how these commands can be used, what their main features are, how to customize them, and more.

As an alternative to phfcc, you can consider the package todonotes[1]. The commands defined in this package are designed for minimal effort by the commenter (e.g., they only have to type `\phf{...}` or `\phf[...]`). On the other hand todonotes offers pretty boxes and more flexibility.

[1] *https://ctan.org/pkg/todonotes*

2 Syntax of commenting commands

Here is the full syntax of the commenting commands generated with `\phfMakeCommentingCommand`. Suppose the command `\AlE` was defined by issuing `\phfMakeCommentingCommand[initials={A.E.}]{AlE}`. You can write:

A.E. `\AlE{some text here}` — some text here — typeset the given text in color, perhaps to mark an addition by author `AlE` to the document. By default, the initials associated with the command (here, "A.E.") will be issued in the margin once per page.

`\AlE[please revise this]` — [please revise this] — typeset an inline comment in a different font. This is meant for comments that are not a portion of document text itself but rather an inline remark that refers to the text around it.

You can use the `footcomments` style to have your comment typeset as a footnote. (See also subsection 3.1 below.)

`\AlE*{text to be removed}` — ~~*text to be removed*~~ — typeset the given text in a way to indicate that the text should be suppressed.

If you use LuaLaTeX, you can get proper text strikethrough accross entire chunks of text by using the argument `style={rmstrikethrough}` to `\phfMakeCommentingCommand`. (See also subsection 3.1 below.)

`\AlE!{important piece of text}` — **!!! important piece of text !!!** — typeset the given text marking it as important.

`\AlE![a really important comment]` — **[!!! a really important comment !!!]** — typeset the given comment and mark it as important.

In each of these cases (except for comments in square brackets), you can use the syntax `\AlE .... \endAlE` instead of specifying the argument using curly brackets:
`\AlE text here\endAlE` — text here
`\AlE! important\endAlE` — **!!! important !!!**
`\AlE* remove\endAlE` — ~~*remove*~~

For simple text, you can also use the `\begin`/`\end` syntax:
`\begin{AlE} text here\end{AlE}` — text here

TIP

Note that in contrast to optional arguments in many LaTeX macros, the argument is allowed to contain matching square brackets, for instance
`\AlE[inner brackets [like this] work as you'd expect]`
[inner brackets [like this] work as you'd expect]
In any case you can protect the argument as usual with curly braces:
`\AlE[{this is [weird]}]` [this is [weird]

■ 3 Defining your commenting command

\phfMakeCommentingCommand

Now let's see exactly how to define your customized \AlE command. The command \phfMakeCommentingCommand takes the following syntax:

\phfMakeCommentingCommand[⟨*key=value,key2=value2,...*⟩]{⟨*cmd name*⟩}

The {⟨*command name*⟩} should be given as a text name (e.g., {AlE}), and not as a macro. They key-value pairs may be one of the following:

color={⟨*color specification*⟩}

The color of the comments generated by the new commenting command. By default, a new suitable color is chosen for each new commenting command defined.

The color specification may be a name such as red, green, etc., but also a mixture like blue!40!green (which stands for 40% of blue and 60% of green). You can specify any argument you could specify to \colorlet from the xcolor package. (The xcolor package is automatically loaded.)

initials={⟨*your initials*⟩}

Your name initials, which will by typeset at the beginning of a comment (typically in a small box in the margin), allowing to identify different comment colors with different people. By default, the initials are set to the name of the command itself.

formatinitials=⟨*keyword or command*⟩

The command that will be used to format the initials. By default (formatinitials=default or formatinitials=margin), the initials are displayed on the margin of the paragraph the first time the change occurs on a given page [like this and like in the examples above]. You may also specify formatinitials=box to typeset the initials inline immediately before your annotation PhF[like this]. Specify formatinitials=nobox to remove the frame PhF[like this]. Or specify formatinitials=hide to hide the initials altogether [like this]. You can also specify formatinitials=footnote to hide the initials at the point of text but with a footnote to associate colored comments with the name (for this to work more nicely, specify your full name as argument to initials={...}) [Here is an example]. [See subsection 3.2 for customizing the formatting of the margin initials for the margin style or for changing the text in the footnote in the footnote style.]

PhF

You may also specify as a value to the formatinitials= option any LaTeX command, which will then be used to format the initials. The macro should accept a single argument, the initials.

The default initials formatting style (formatinitials=margin) requires the marginnote package, which is loaded automatically. If you do

Changes by Philippe Faist

not want to load the marginnote package, specify the package option `usemarginnote=false` and set a different `formatinitials=...` style.

`style={`⟨*style name(s)*⟩`}`

Load one or more existing style(s) for the commenting command. The styles must either be predefined, or must have been defined using `\phfDefineCommentingStyle`. Separate style names with a comma.

`font={`⟨*LaTeX commands*⟩`}`

PhF The font commands to invoke when typesetting annotations, regardless of type. By default, the font is unchanged other than having the author-specific color set. Also, the font is not reset, so if you type an annotation somewhere where the font is large, the annotation's font will match that font by default. (Different annotation types may set their own font, see below.)

`spacing=`⟨*length*⟩

Spacing to add around normal annotations. This amount of horizontal space is added before and after the annotation.

`begin={`⟨*LaTeX text and/or commands*⟩`}`

Stuff to typeset at the beginning of a general annotation. Nothing is typeset by default.

`end={`⟨*LaTeX text and/or commands*⟩`}`

Stuff to typeset at the end of a general annotation. Nothing is typeset by default.

`cfont={`⟨*LaTeX commands*⟩`}`

The font commands to invoke when typesetting an in-line comment. By default, use [the default sans serif typeface, like this].

`cspacing=`⟨*length*⟩

Spacing to add around comments. This amount of horizontal space is added before and after the comment.

`cbegin={`⟨*LaTeX text and/or commands*⟩`}`

Stuff to typeset at the beginning of a commment. By default, an opening square bracket (with some spacing adjustments).

`cend={`⟨*LaTeX text and/or commands*⟩`}`

Stuff to typeset at the end of a commment. By default, a closing square bracket (with some spacing adjustments).

`rmfont={`⟨*LaTeX commands*⟩`}`

The font commands to issue when typesetting text that is to be removed.

~~*By default, the text is in italics like this*~~ (If you're using LuaLaTeX, it's recommended to use the `rmstrikethrough` style for proper strikethrough text; see subsection 3.1.)

`rmspacing=`⟨*length*⟩

Spacing to add around text that is marked to be removed. This amount of horizontal space is added before and after.

`rmbegin={`⟨*LaTeX text and/or commands*⟩`}`

Stuff to typeset at the beginning of a piece of text to be removed. By default, a line overlapping with the text, to suggest removal.

`rmend={`⟨*LaTeX text and/or commands*⟩`}`

Stuff to typeset at the end of a piece of text to be removed. By default, a line overlapping with the text, to suggest removal.

`ifont={`⟨*LaTeX commands*⟩`}`

PhF The font commands to issue when typesetting important text. **!!! By default, the text is in larger boldface format like this !!!**

`ispacing=`⟨*length*⟩

Spacing to add around important text. This amount of horizontal space is added before and after.

`ibegin={`⟨*LaTeX text and/or commands*⟩`}`

Stuff to typeset at the beginning of a piece of important text. By default, three exclamation marks.

`iend={`⟨*LaTeX text and/or commands*⟩`}`

Stuff to typeset at the end of a piece of important text. By default, three exclamation marks.

After invoking `\phfMakeCommentingCommand`, the colors `xxxcolor`, `xxxrmcolor` and `xxxrmcolorlink` are automatically defined (where `xxx` is to be replaced by your commenting macro name). These are used respectively for a usual comment color, text to be removed, and for links in text to be removed. You may redefine these colors afterwards with `\colorlet` if you wish:

```
\colorlet{AlErmcolor}{gray}
```

`\phfDisableCommentingCommands` The command `\phfDisableCommentingCommands` disables all commenting commands, and causes them to emit an error. Use this command when approaching the final version of a long document to ensure that no commenting commands are left in the document.

3.1 Commenting Styles

Styles regroup a collection of definitions that alter the overall behavior and appearance of commenting commands issued with `\phfMakeCommentingCommand`. You can assign one or more styles to a commenting command definition by using the `style=` keyword:

```
\phfMakeCommentingCommand[style={footcomments}]{JD} % \JD
```

You can combine style names with commas, for instance:

```
\phfMakeCommentingCommand[style={rmstrikethrough,footcomments}]{JD} % \JD
```

The following styles are predefined:

`footcomments`

- The `footcomments` style arranges for in-line comments to be typeset as footnotes.

  ```
  \phfMakeCommentingCommand[style={footcomments}]{YY}

  For instance, \YY[This would be an example of a comment
  typeset as a footnote.] this text might be suitable to
  comment on.
  ```

  gives:

  "For instance, [2] this text might be suitable to comment on."

  This style can be convenient to place comments without interrupting the natural flow of the text.

  You can redefine `\phfccfootcommentwrapfnmark{..}` to customize the appearance of the footnote mark, and `\phfccfootcommenttextstyle` to get some handle on the commands issued within the footnote contents.

  ```
  \renewcommand\phfccfootcommentwrapfnmark[1]{[[#1]]}
  \renewcommand\phfccfootcommenttextstyle{\bfseries}
  ```

`rmstrikethrough`

- For LuaLaTeX users, the style `rmstrikethrough` is available to make text marked for removal drawn in strikethrough font, ~~*kind of like this*~~. You can cross out entire paragraphs of text which include display equations. Simply use:

  ZZ

  ```
  \phfMakeCommentingCommand[style=rmstrikethrough]{ZZ} % requires LuaLaTeX

  ... \ZZ*{text to be removed} ... or also ... \ZZ* more
  text that I want to remove\endZZ ...
  ```

YY [2] [This would be an example of a comment typeset as a footnote.]

The implementation uses the `lua-ul` package, which requires LuaLaTeX. To apply this style to all definitions, use `\phfSetDefaultCommentingStyle{style=rmstrikethrough}` (see below).

(We do not support strikethrough for regular LaTeX because we weren't able to find an implementation that worked reliably even for large blocks of text. For instance, using the `\sout` from the ulem package, you cannot strikethrough multiple paragraphs with displayed equations. Plus, to support the extended syntax of `\phfcc`'s commenting commands, we need the strikethrough to be implemented by a state change macro (like `\itshape`) or an environment, and not by a macro that accepts an argument.)

- We'll probably add more predefined styles in the future.

`\phfDefineCommentingStyle` You can define a style using `\phfDefineCommentingStyle`:

```
\phfDefineCommentingStyle{mystyle}{
  font={\fontfamily{phv}\selectfont}, % phv = Helvetica
  cfont={\bfseries\large},
  cbegin={{\ensuremath{\bigl\langle}}~},
  cend={{~\ensuremath{\bigr\rangle}}}
}
```

You can then define your commenting command invoking this style:

```
\phfMakeCommentingCommand[style=mystyle]{me}
```

me Then for example, (which was typeset using `\me{for example}`,) the code `\me[A comment]` becomes: ⟨ **A comment** ⟩.

Styles are loaded in the order they are specified (separate style names with commas). Style definitions may themselves contain `style=...` keys to load other styles. Styles are always loaded before user keys, regardless of where the '`style`' key appears in the definition. When styles are loaded, and when the users' keys are set, later keys are overwrite earlier ones.[3]

`\phfSetDefaultCommentingStyle` Instead of defining named styles which have to be invoked explicitly, you can also preset the style of all future calls of `\phfMakeCommentingCommand`. You can achieve this result with the `\phfSetDefaultCommentingStyle` command:

```
\phfSetDefaultCommentingStyle{
  font={\large}
}
% all annotations defined from here on will appear larger
\phfMakeCommentingCommand{AB}
```

[3] Except for an as-of-yet undocumented `startcmds` and `endcmds` keys which might be relevant if you'd like to develop a style which requires you to issue definitions at the beginning of every annotation.

You can also use \phfMakeCommentingCommand to set one or more default style(s) for all future commenting command definitions:

```
\phfSetDefaultCommentingStyle{
  style={rmstrikethrough,footcomments}
}
% all annotations defined from here on will have these styles set
\phfMakeCommentingCommand{AB}
```

3.2 Customizing the formatting of the initials label

For the formatinitials=margin *style:*

You can customize the appearance of the initials in the margin when using the formatinitials=margin initials style by redefining the following macros. Changes affect all margin labels (i.e., changes affect all commenting commands using the margin initials formatting style).

\phfccformatmargininitials The \phfccformatmargininitials is used to format the initials in the margin. You can redefine it if you'd like the initials to appear differently. Adapt the following, which contains the default definition, to your liking:

```
\renewcommand{\phfccformatmargininitials}[1]{%
  \fbox{\normalfont\sffamily\footnotesize #1}%
}
```

(Technical note: the \fboxsep length has been set to 1pt prior to calling this macro.)

\phfccmargininitialssep The \phfccmargininitialssep length is the minimal vertical separation between two initials labels. This is a length, so you should change it with \setlength:

```
\setlength{\phfccmargininitialssep}{2pt}
```

For the formatinitials=box *and* formatinitials=nobox *styles:*

\phfccformatboxinitials For the formatinitials=box and formatinitials=nobox initials style, you can redefine the command \phfccformatboxinitials to typeset the initials, like so:

```
\renewcommand{\phfccformatboxinitials}[1]{%
  \normalfont\sffamily\tiny#1%
}
```

For the `formatinitials=footnote` *style:*

`\phfCCChangesBy` When using the `formatinitials=footnote` initials formatting style, you may redefine `\phfCCChangesBy` to change the text in the footnote:

```
\renewcommand{\phfCCChangesBy}[1]{Changes in this color are by #1}
```

4 Package Options

This package requires the marginnote package to generate initials labels when using the `margin` initials style (the default style). If you do not want to load the marginnote package, you may use the package option `usemarginnote=false` (but then you cannot use the `margin` initials style):

```
\usepackage[usemarginnote=false]{phfcc}
```

5 Known limitations

There are some cases where it's hard to get everything right. The phfcc commands do their best to get stuff right and to avoid generating obscure LaTeX errors, but there might be times where not everything works as expected.

- If you place a comment inside certain contructs that are processed twice by LaTeX (e.g., a figure caption when using e.g. the caption package, etc.) then the margin initials will not appear even if the comment is the first of the page. This is because the margin label is produced the first time they are typeset, per page; if that happened to be in a temporary TeX box that was then discarded, the margin note gets discarded along with it.

  In the case of AMS equations, there's a hack that make the labels appear right. Feel free to send pull requests to address other cases.

  PhF [FIXME: A better solution might be to write to the AUX file at each comment, so that the information is written when the comment is actually typeset, and then process where to put margin labels on the second run of LaTeX.

  As a bonus, this would enable stuff like "list of comments" like the todonotes package does. Even though I don't think it's necessary for the standard use case of this package.]

- Initials in the margin might overlap in some edge cases. See my comments and footnotes in the implementation documentation below.

- Initials in the margin might not appear if the comment immediately follows a page break, because the comment might have been processed while LaTeX was on the previous page before deciding to start a new page. See my comments and footnotes in the implementation doc below.

6 Implementation

Load these internally required packages.

```
\RequirePackage{xkeyval}
\RequirePackage{kvoptions}
\RequirePackage{etoolbox}
\RequirePackage{xparse}
```

Ensure we have the xcolor package to manage text colors. We need xcolor and not color because we use `\colorlet`.

```
\RequirePackage{xcolor}
```

6.1 Set up default settings

`\phfCommentingDefault...` Provide sensible defaults for commenting formatting.

Bold or semibold CFont sounds like a good idea, but comments would seem more aggressive like that, so keep them normal by default.

```
\def\phfCommentingDefaultStartCmds{}
\def\phfCommentingDefaultEndCmds{}
\def\phfCommentingDefaultFont{}
\def\phfCommentingDefaultSpacing{0pt}
\def\phfCommentingDefaultBegin{}
\def\phfCommentingDefaultEnd{}
\def\phfCommentingDefaultCFont{\normalfont\sffamily}
\def\phfCommentingDefaultCSpacing{0.2em}
\def\phfCommentingDefaultCBegin{[\,}
\def\phfCommentingDefaultCEnd{\,]}
\def\phfCommentingDefaultRmFont{\small\itshape}%\itshape\notesmaller[0.9]}
\def\phfCommentingDefaultRmSpacing{.3em}
\def\phfCommentingDefaultRmBegin{%
  \mbox{}\vrule height .55ex depth -.45ex width 1.4em\relax\hspace*{0.2em}%
  \vrule height .55ex depth -.45ex width 0.6em\relax
  \hspace*{-2em}}
\def\phfCommentingDefaultRmEnd{%
  \hspace*{-2em}%
  \vrule height .55ex depth -.45ex width 0.6em\relax\hspace*{0.2em}%
  \vrule height .55ex depth -.45ex width 1.4em\relax}
\def\phfCommentingDefaultIFont{\large\bfseries}
\def\phfCommentingDefaultISpacing{0.25em}
\def\phfCommentingDefaultIBegin{!\hspace*{0.1em}!\hspace*{0.1em}!~}
\def\phfCommentingDefaultIEnd{~!\hspace*{0.1em}!\hspace*{0.1em}!}
```

A default list of colors for commenting, and a minimal tool to select the next available color. The macro `\phf@cc@usedcolors` stores a vertical-bar-separated list of color names that have been already used and which should not be used again.

```
\definecolor{phfcc0}{RGB}{0,148,240} % blue-y
\definecolor{phfcc1}{RGB}{242,108,13} % orange-brown-y
\definecolor{phfcc2}{RGB}{65,149,42} % green-y
\definecolor{phfcc3}{RGB}{128,55,134} % purple-y
\definecolor{phfcc4}{RGB}{0,129,129} % blue-green-y
\definecolor{phfcc5}{RGB}{148,7,24} % dark red / burgundy
\definecolor{phfcc6}{RGB}{160,120,0} % brownish
\definecolor{phfcc7}{RGB}{35,195,155} % aqua-ish
\csdef{phf@cc@presetcolor@0}{phfcc0}
\csdef{phf@cc@presetcolor@1}{phfcc1}
\csdef{phf@cc@presetcolor@2}{phfcc2}
\csdef{phf@cc@presetcolor@3}{phfcc3}
\csdef{phf@cc@presetcolor@4}{phfcc4}
\csdef{phf@cc@presetcolor@5}{phfcc5}
\csdef{phf@cc@presetcolor@6}{phfcc6}
\csdef{phf@cc@presetcolor@7}{phfcc7}
\def\phf@cc@usedcolors{|}
\def\phf@cc@nextcolor@#1{%
  \ifcsname phf@cc@presetcolor@#1\endcsname% try preset
    \edef\phf@tmp@testxx{%
      \noexpand\in@{|\csname phf@cc@presetcolor@#1\endcsname|}{\phf@cc@usedcolors}}%
    \phf@tmp@testxx
    \ifin@% color already used, try next
      \expandafter\phf@cc@nextcolor@\expandafter{\the\numexpr#1+1\relax}%
    \else
      \def\phf@cc@thecolor{\csname phf@cc@presetcolor@#1\endcsname}% good, use
    \fi
  \else% out of colors, fallback to red
    \def\phf@cc@thecolor{red}%
  \fi
}
```

Read the argument. If non-empty, set `\phf@cc@thecolor` to the argument value. If empty, choose the next available preset color and set `\phf@cc@thecolor` to that.

```
\def\phf@cc@getcolor#1{%
  \edef\phf@tmp@xyz{#1}%
  \if\relax\detokenize\expandafter{\phf@tmp@xyz}\relax
    \phf@cc@nextcolor@{0}%
  \else
    \edef\phf@cc@thecolor{#1}%
  \fi}
```

Define the keys for `\setkeys` with xkeyval package, and set the overall defaults.

```
\def\phfcc@hook@phfmkccstyle@setstyle@noop#1{}
\let\phfcc@hook@phfmkccstyle@setstyle\phfcc@hook@phfmkccstyle@setstyle@noop
\define@cmdkey{phfmkccstyle}{style}{\phfcc@hook@phfmkccstyle@setstyle{#1}}

\define@cmdkey{phfmkcc}{color}{}
```

```
\define@cmdkey{phfmkcc}{initials}{}
\define@cmdkey{phfmkcc}{formatinitials}{}
\newtoks\cmdKV@phfmkcc@startcmds
\newtoks\cmdKV@phfmkcc@endcmds
\define@key{phfmkcc}{startcmds}{%
  \cmdKV@phfmkcc@startcmds=\expandafter{\the\cmdKV@phfmkcc@startcmds#1}%
}
\define@key{phfmkcc}{endcmds}{%
  \edef\x{\unexpanded{#1}\the\cmdKV@phfmkcc@endcmds}%
  \cmdKV@phfmkcc@endcmds=\expandafter{\x}%
}
\define@cmdkey{phfmkcc}{font}{}
\define@cmdkey{phfmkcc}{spacing}{}
\define@cmdkey{phfmkcc}{begin}{}
\define@cmdkey{phfmkcc}{end}{}
\define@cmdkey{phfmkcc}{cfont}{}
\define@cmdkey{phfmkcc}{cspacing}{}
\define@cmdkey{phfmkcc}{cbegin}{}
\define@cmdkey{phfmkcc}{cend}{}
\define@cmdkey{phfmkcc}{rmfont}{}
\define@cmdkey{phfmkcc}{rmspacing}{}
\define@cmdkey{phfmkcc}{rmbegin}{}
\define@cmdkey{phfmkcc}{rmend}{}
\define@cmdkey{phfmkcc}{ifont}{}
\define@cmdkey{phfmkcc}{ispacing}{}
\define@cmdkey{phfmkcc}{ibegin}{}
\define@cmdkey{phfmkcc}{iend}{}
\define@cmdkey{phfmkcc}{groupcmd}{}
```

Factory defaults refer to the `\phfCommentingDefault***` commands.

```
\presetkeys{phfmkcc}{%
  color={},%
  initials={},%
  formatinitials={default},%
  startcmds={\phfCommentingDefaultStartCmds},%
  endcmds={\phfCommentingDefaultEndCmds},%
  font={\phfCommentingDefaultFont},%
  spacing={\phfCommentingDefaultSpacing},%
  begin={\phfCommentingDefaultBegin},%
  end={\phfCommentingDefaultEnd},%
  cfont={\phfCommentingDefaultCFont},%
  cspacing={\phfCommentingDefaultCSpacing},%
  cbegin={\phfCommentingDefaultCBegin},%
  cend={\phfCommentingDefaultCEnd},%
  rmfont={\phfCommentingDefaultRmFont},%
  rmspacing={\phfCommentingDefaultRmSpacing},%
  rmbegin={\phfCommentingDefaultRmBegin},%
  rmend={\phfCommentingDefaultRmEnd},%
  ifont={\phfCommentingDefaultIFont},%
  ispacing={\phfCommentingDefaultISpacing},%
```

```
  ibegin={\phfCommentingDefaultIBegin},%
  iend={\phfCommentingDefaultIEnd},%
  groupcmd={\@firstofone}%
}{}
```

6.2 Helpers and handlers for styles

\phfDefineCommentingStyle User interface for defining custom comment styles.

```
\newcommand\phfDefineCommentingStyle[2]{%
  \csdef{phfcc@style@#1}{#2}%
}
```

\phfSetDefaultCommentingStyle User interface for defining the default settings for all commenting commands declared from now on. Keep note of `style=` keywords separately (in `\phfcc@default@style`), because we cannot place them as `\presetkeys` in the main options family.

```
\def\phfcc@default@style{}
\def\phfcc@hook@phfmkccstyle@setstyle@savestyle#1{%
  \xdef\phfcc@default@style{#1}%
}
\newcommand\phfSetDefaultCommentingStyle[1]{%
  \let\phfcc@hook@phfmkccstyle@setstyle\phfcc@hook@phfmkccstyle@setstyle@savestyle
  \setkeys*{phfmkccstyle}{#1}%
  \let\phfcc@hook@phfmkccstyle@setstyle\phfcc@hook@phfmkccstyle@setstyle@noop
  \edef\x{\noexpand\presetkeys{phfmkcc}{\expandonce\XKV@rm}{}}%
  \x
}
```

6.2.1 Internal helpers for loading styles

\phfcc@expandstylekeys Helpers to expand style= instructions in a set of key=vals.

The helper macro `\phfcc@expandstylekeys` accepts two arguments. The argument `#1` is a list of style names and `#2` is a collection of key=value pairs. This macro computes all the key=value settings that should be applied from the style definitions, plus all the remaining user key=value settings. We will recursively look for style=... keywords so that styles can load other styles.

The final expanded key=value pairs is set in the token register `\phfcc@val@expanded@keyval`.

```
\newtoks\phfcc@val@expanded@keyval
\def\phfcc@expandstylekeys#1#2{%
  \def\phfcc@val@expanded@preexpandstyles{#1}%
  \phfcc@val@expanded@keyval={#2}%
  \loop
```

```
    \phfcc@expandstylekeys@once
    \def\phfcc@val@expanded@preexpandstyles{}%
  \ifnum\phfcc@val@expanded@needmore=1\repeat
}
```

The implementation of `\phfcc@expandstylekeys` proceeds by iterating single expansions until there are no more `style={}` values left. A single iteration is implemented here. The following macro reads its input from `\phfcc@val@expanded@preexpandstyles` and `\phfcc@val@expanded@keyval`, and it sets the macros `\phfcc@val@expanded@keyval` and `\phfcc@val@expanded@needmore`.

```
\def\phfcc@expandstylekeys@once{%
  \def\cmdKV@phfmkccstyle@style{}
  \edef\x{\noexpand\setkeys*{phfmkccstyle}{\the\phfcc@val@expanded@keyval}}%
  \x
```

The `style={...}` values is stored in the macro `\cmdKV@phfmkccstyle@style`. If it is empty, there is no style to set and we're done.

```
  \if\relax\detokenize\expandafter{\cmdKV@phfmkccstyle@style}\relax
    \def\phfcc@val@expanded@needmore{0}%
  \else
    \def\phfcc@val@expanded@needmore{1}%
  \fi
```

We might have one or more styles to expand. Iterate over the comma-separated list of styles and we'll append the collected style-specific key=value definitions in the macro `\phfcc@val@fullstylekeyvals` for now. Include also the predefined style names to expand before the user-specified styles.

```
  \def\phfcc@val@fullstylekeyvals{}%
  \edef\phfcc@tmp@thestylelist{%
    \phfcc@val@expanded@preexpandstyles,\cmdKV@phfmkccstyle@style}%
  \@for\phfcc@tmp@next:=\phfcc@tmp@thestylelist \do{%
```

At this point we're requested to expand the style whose name is stored in `\phfcc@tmp@next`. Collect the key=value definitions in the macro named `\phfcc@style@STYLENAME` or produce an error if such macro does not exist. Also, if the style provides a macro named `\phfcc@styleused@STYLENAME`, then invoke it at this point, in case the style would like to load any additional packages or execute any other preamble-time setup steps. In the following `\phfcc@tmp@stylekeyvals` is defined so that its immediate expansion yields the styles' key=value definitions.

```
      \ifcsname phfcc@style@\phfcc@tmp@next \endcsname
        \ifcsname phfcc@styleused@\phfcc@tmp@next \endcsname
          \csname phfcc@styleused@\phfcc@tmp@next \endcsname
        \fi
        \edef\phfcc@tmp@stylekeyvals{%
```

```
          \expandafter\expandonce\csname phfcc@style@\phfcc@tmp@next \endcsname}%
        \edef\phfcc@val@fullstylekeyvals{%
          \expandonce\phfcc@val@fullstylekeyvals,%
          \expandonce\phfcc@tmp@stylekeyvals}%
      \else
        \if\relax\detokenize\expandafter{\phfcc@tmp@next}\relax
        \else
          \PackageError{phfcc}{Requested commenting style `\phfcc@tmp@next' %
            is not defined}{}%
        \fi
      \fi
    }%
```

Now we can collect the definitions from the styles and merge them with the remaining non-style-related user key/values. The macro `\XKV@rm` is set by xkeyval to those key/values not processed by `\setkeys*`.

```
  \edef\x{\noexpand\phfcc@val@expanded@keyval={%
      \expandonce\phfcc@val@fullstylekeyvals,%
      \expandonce\XKV@rm}}%
  \x
}
```

6.2.2 The `rmstrikethrough` predefined style

`\phfcc@style@rmstrikethrough` The definition of the `rmstrikethrough` style is fairly straightforward.

```
\gdef\phfcc@styleused@rmstrikethrough{%
  \RequirePackage{lua-ul}
  \newunderlinetype\beginPhfccStrikeThrough{%
    \leaders\vrule height .55ex depth -.45ex}
  \gdef\phfcc@styleused@rmstrikethrough{}% auto-destruct/only run once
}
\def\phfcc@style@rmstrikethrough{%
  rmfont={\beginPhfccStrikeThrough},
  rmbegin={},
  rmend={},
  rmspacing={2pt}
}
```

6.2.3 The `footcomments` predefined style

`\phfcc@style@footcomments` The `footcomments` style uses `startcmds` and `endcmds` to collect material and then typeset it into a footnote.

```
\def\phfcc@style@footcomments{%
  startcmds={\phfcc@collectcommentfootnote},
  endcmds={\phfcc@endcollectcommentfootnote},
}
```

\phfccfootcommenttextstyle The macro \phfccfootcommenttextstyle can be redefined to customize the footnote text appearance.

```
\def\phfccfootcommenttextstyle{\footnotesize}
```

Now we define helpers to collect the contents of the footnote, and then typeset it.

\phfcc@collectcommentfootnote
\phfcc@endcollectcommentfootnote The entry point to these helpers are the macros \phfcc@collectcommentfootnote and \phfcc@endcollectcommentfootnote: They check whether this annotation is a comment and then call our main helpers if that's the case.

```
\def\phfcc@collectcommentfootnote{%
  \if@phfcc@iscomment
    \expandafter\phfcc@collectfootnote
  \fi}
\def\phfcc@endcollectcommentfootnote{%
  \if@phfcc@iscomment
    \expandafter\phfcc@endcollectfootnote
  \fi}
```

\phfcc@collectfootnote
\phfcc@endcollectfootnote
\phfcc@startfootnote The helper \phfcc@collectfootnote collects the material and builds the footnote. I had to copy some footnote-building code from latex.ltx unfortunately: We can't call \footnote{} because it requires a macro argument; we need begin/end macros to feed all the footnote contents and material as TEX parses it. Recall that LATEX's \footnote basically (1) steps the footnote counter (the counter name is stored in \@mpfn and the printable value is given by \thempfn) and prints the footnote mark (\@footnotemark), and (2) collects material in the \insert called \footins. The code relevant to (1) is collected in \phfcc@startfootnote. The code relevant to (2) is collected and adapted below between \phfcc@collectfootnote and \phfcc@endcollectfootnote.

```
\def\phfcc@collectfootnote{%
  \phfcc@startfootnote
  \insert\footins\bgroup%
    \reset@font\footnotesize
    \interlinepenalty\interfootnotelinepenalty
    \splittopskip\footnotesep
    \splitmaxdepth \dp\strutbox \floatingpenalty \@MM
    \hsize\columnwidth \@parboxrestore
    \protected@edef\@currentlabel{%
       \csname p@footnote\endcsname\@thefnmark
    }%
    \color@begingroup
    \@makefntext{%
      \rule\z@\footnotesep\@finalstrut\strutbox\phfcc@hackrevtex@skippar}%
    \ignorespaces
    \footnotesize
```

```
    \color{\phf@cc@val@cur color}%
    \phfccfootcommenttextstyle
}
\def\phfcc@endcollectfootnote{%
    \unskip
    \color@endgroup
  \egroup
}
\def\phfcc@startfootnote{%
  \stepcounter\@mpfn
  \protected@xdef\@thefnmark{\thempfn}%
  \phfccfootcommentwrapfnmark{\@footnotemark}%
}
\def\phfccfootcommentwrapfnmark#1{%
  [#1]%
}
```

Above, we used the class-provided `\@makefntext` macro to make comment footnotes look like the other document-provided footnotes, but we invoke that macro with dummy content because remember, we haven't collected the footnote contents yet. But some latex classes, like RevTEX, this macro add a paragraph; so we need to eliminate that paragraph break that would otherwise appear at the beginning of the footnote:

```
\def\phfcc@hackrevtex@skippar{%
  \@ifnextchar\par\@gobble\relax
}
```

6.3 Define styles for initials formatting

Define the implementations for the different initials formatting. These are by default the small boxes that appear in the margin that match color to comment owner.

Define the `margin` initials style as the default:

```
\def\phf@cc@formatinitialsstyle@default{\phf@cc@formatinitialsstyle@margin}
```

6.3.1 Define styles `hide`, `box`, `nobox`

`\phf@cc@formatinitialsstyle@hide`

Define the no-op `hide` style first.

```
\def\phf@cc@formatinitialsstyle@hide#1{}
```

`\phf@cc@formatinitialsstyle@box` `\phf@cc@formatinitialsstyle@nobox`

Define the `box`, and `nobox` styles. In either cases, the label can be customized by redefining `\phfccformatboxinitials`.

```
\def\phf@cc@formatinitialsstyle@box#1{%
```

```
  \hspace{0.25em}\relax
  \raisebox{1pt}{{\fboxsep=1pt\relax\fbox{\phfccformatboxinitials{#1}}}}%
  \hspace{0.25em}}
\def\phf@cc@formatinitialsstyle@nobox#1{%
  \hspace{0.25em}\relax
  {\phfccformatboxinitials{#1}}%
  \hspace{0.25em}}
```

\phfccformatboxinitials

```
\def\phfccformatboxinitials#1{%
  \normalfont\sffamily\tiny#1%
}
```

6.3.2 Define the style `footnote`

Define the `footnote` style. Note you can redefine `\phfCCChangesBy` to whatever you please, and it can even take the name as `#1` argument.

```
\def\phf@cc@formatinitialsstyle@footnote#1{%
  \ifcsname phf@cc@valfmtinitialsfootnote@intlspage@\phf@cc@val@cur @\roman{page}\endcsname
    % already displayed on this page, don't repeat
  \else
    \csgdef{phf@cc@valfmtinitialsfootnote@intlspage@\phf@cc@val@cur @\roman{page}}{1}%
    \def\phf@cc@tmp@zz{\gdef\@thefnmark{}\@footnotetext}%
    \expandafter\phf@cc@tmp@zz\expandafter{\expandafter{%
        \expandafter\color\expandafter{\phf@cc@val@cur color}\phfCCChangesBy{#1}}}%
  \fi
}
\robustify\phf@cc@formatinitialsstyle@footnote
\def\phfCCChangesBy{Changes by\ }
```

6.3.3 Define the style `margin`

\phf@cc@formatinitialsstyle@margin

Define the `margin` style now. This one is a little tricky.

```
\def\phf@cc@formatinitialsstyle@margin#1{%
```

First thing, we define an auxiliary global macro to remember whenever we've already displayed the margin note on a given page. If the label was already displayed on this page (as witnessed by the following test macro being defined), don't repeat the label display and skip altogether.[4]

```
  \ifcsname phf@cc@valfmtinitialsmargin@intlspage@\phf@cc@val@cur
```

[4] There are some edge cases here I'm not sure I want to spend the effort to fix. For instance, if the comment appears right after a page break, it might have been processed by TEX while on the previous page and this test might get the page number wrong. Oh well, the label was on last page already so you should know whose comment this is.

```
    @\roman{page}\endcsname
  \else
```

Check for any other reasons why we shouldn't emit a margin note (e.g. for some known two-pass environments); this is done by `\phf@cc@margin@ifchecks`. Then actually create the margin note with `\phf@cc@margin@emit`.

```
    \phf@cc@margin@ifchecks{%
      \phf@cc@margin@emit{#1}%
    }%
  \fi
}
\robustify\phf@cc@formatinitialsstyle@margin
```

`\phf@cc@margin@ifchecks` Detects if we're in a first pass in an AMS equation, for instance, to only emit the label in the second pass (actual typesetting).

```
\def\phf@cc@margin@ifchecks#1{%
  \phf@cc@ififexists{ifmeasuring@}{%
    \message{Detected first pass in environment, not emitting margin
      initials label this time.}%
  }{%
    \phf@cc@ififexists{if@inlabel}{%
      \message{Detected comment in latex item label, not emitting margin
        initials at this time to avoid messing up everything.}%
    }{%
      #1%
    }%
  }%
}
```

`\phf@cc@ififexists` A helper macro. The `\phf@cc@ififexists{`⟨*if-cs-name*⟩`}{`⟨*commands if true*⟩`}{`⟨*commands if false*⟩`}` macro checks if the command `\ifXXXX` exists, where the name `ifXXXX` is the first argument of `\phf@cc@ififexists`. If it exists, then it checks that "if" condition and executes `{`⟨*commands if true*⟩`}` or `{`⟨*commands if false*⟩`}` as appropriate. If the "if" macro doesn't exist, then executes `{`⟨*commands if false*⟩`}`.

```
\def\phf@cc@ififexists#1#2#3{%
  \ifcsname #1\endcsname
    \def\x{\phf@cc@ififexists@{#1}{#2}{#3}}%
  \else
    \def\x{#3}%
  \fi
  \x
}
\def\phf@cc@ififexists@#1#2#3{%
  \csname #1\endcsname
    #2%
  \else
```

```
    #3%
  \fi
}
```

At this point, we are all set to display the initials in the margin. It seems like there should be nothing wrong with simply emitting a `\marginpar`, but those don't work in footnotes and figure captions, where one would also like to be able to introduce comments. One could use the marginnote package instead, which is more reliable, but then we can get overlapping margin labels which are not very nice.

On the subject of `\marginnote` vs `\marginpar`: The strategy here is use `\marginnote` all the time. Originally I used `\marginpar`, but it doesn't work in footnotes and figure captions, where one often wants to comment things. It's really hard to detect all cases where `\marginpar` is problematic. (`\ifinner` detects minipages/equations, value of `\@captype` detects figure captions, but I didn't manage to detect footnotes reliably, and I anticipate other problematic situations coming up). So I use marginnote instead, which is more reliable. We could have some complex process to decide whether to use `\marginpar` or `\marginnote`, but only if we can find a really reliable algorithm. When collaborators are commenting on a document in the final hours before a deadline, we really don't want to generate obscure LaTeX errors because they happened to hit an edge case by placing a comment somewhere I hadn't anticipated. So for now, universally use `\marginnote`. If you don't want to load the marginnote package for whatever reason, use a different initials label style and specify the `usemarginnote=false` package option.

To avoid margin notes overlapping with marginnote, we use custom code with a simple algorithm to shift the margin notes vertically so that they don't overlap. The code is a bit ugly, but it seems to work and it doesn't rely on too specific hacks, so it is hopefully not too prone to edge cases.

`\phf@cc@margin@emit` This is the macro that actually creates the margin note. First we create a `\hbox` with the label so that we can measure its height. Then we defer to `\phf@cc@createmarginnote` to produce the margin note with the formatted initials.

```
\newbox\phf@cc@margin@labelbox
\newdimen\phf@cc@margin@labelboxhgt
\def\phf@cc@margin@emit#1{%
  \setbox\phf@cc@margin@labelbox=\hbox{\strut
    \expandafter\color\expandafter{\phf@cc@val@cur color}%
    \fboxsep=1pt\relax
    \phfccformatmargininitials{#1}}%
  \phf@cc@margin@labelboxhgt=\ht\phf@cc@margin@labelbox
  \phf@cc@createmarginnote
  \csgdef{phf@cc@valfmtinitialsmargin@intlspage@\phf@cc@val@cur%
    @\roman{page}}{1}%
}%
```

\phfccformatmargininitials The \phfccformatmargininitials{⟨*initials*⟩} can be redefined to change the appearance of the margin intials box.

```
\def\phfccformatmargininitials#1{%
  \fbox{\normalfont\sffamily\footnotesize #1}%
}
```

\phf@cc@createmarginnote The tricky thing with marginnote is that labels will overlap if they are emitted on the same line. Here we use a workaround: We check \pdflastypos and add some vertical spacing if we are too close to the last note's y position on the same page. But the issue is that \pdfsavepos and \pdflatexypos only work while shipping out, so basically all we can do is write the values to the AUX file and use them next time latex is run. Here we use the same mechanism as marginnote: define a new LaTeX label printing a \newphfccmarginnote in the AUX file with the relevant meta-info. We use the internal counters \@mn@thispage (current absolute page number) and \@mn@atthispage (number of note on current page) because we also need to track which notes are on which page, and there's no use duplicating all that code (I do hope those counters don't change, though).

```
\def\phf@cc@createmarginnote{%
  \marginnote[{\copy\phf@cc@margin@labelbox}]{\copy\phf@cc@margin@labelbox}[%
    \phf@cc@marginextractvshift]%
  \phf@cc@savepos
  \protected@write\@auxout{}{%
    \string\newphfccmarginnote{%
      {\@mn@thispage}%
      {\@mn@atthispage}%
      {\noexpand\number\phf@cc@lastypos sp}%
      {\number\phf@cc@margin@labelboxhgt sp}%
    }%
  }%
}
\ifcsname pdfsavepos\endcsname
  \let\phf@cc@savepos\pdfsavepos
  \let\phf@cc@lastypos\pdflastypos
\else
  \ifcsname savepos\endcsname
    \let\phf@cc@savepos\savepos
    \let\phf@cc@lastypos\lastypos
  \else
    \PackageError{phfcc}{Cannot find appropriate \string\pdfsavepos\space
      command, neither \string\pdfsavepos\space nor\string\savepos\space
      are defined.}{}
  \fi
\fi
```

\phf@cc@marginextractvshift The macro \phf@cc@marginextractvshift is responsible for extracting the required vertical shift from the corresponding label created from the information saved in the AUX file. *This macro is fully expandable, and is meant to be used as*

the value of the optional vertical-shift argument in `\marginnote`. The current note is identified by the internal marginnote counters `\@mn@thispage` (current absolute page number) and `\@mn@atthispage` (current note number on this page).

The label `\csname phfccmn@\@mn@thispage.\@mn@atthispage\endcsname` generated by `\newphfccmarginnote` expands to {{⟨*pdf y-pos of the point in text*⟩}{⟨*label box height*⟩}}{{⟨*requried vertical shift to adjust this note*⟩}}. The first group of fields are the two last arguments specified to `\newphfccmarginnote` written to the AUX file (the two first are already in the label/macro name), and the second group contains the values (only one currently) computed when calling `\newphfccmarginnote`.

Used as `\marginnote`'s vertical shift length. Expands to the vertical shift needed to correct the current note. Remember: no assignments in here, this command must be fully expandable.

```
\def\phf@cc@marginextractvshift{%
  \ifcsname phfccmn@\@mn@thispage.\@mn@atthispage\endcsname
    \expandafter\expandafter\expandafter\expandafter\expandafter
    \expandafter\expandafter\@firstofone
      \expandafter\expandafter\expandafter\@secondoftwo
        \csname phfccmn@\@mn@thispage.\@mn@atthispage\endcsname
  \else
    \z@
  \fi
}
```

`\newphfccmarginnote` The `\newphfccmarginnote` command defines the margin label in the AUX file. It's not meant as public API, it's only written in the AUX file to be re-read on next run. Here is where all the calculations are done.

In this macro we get the page number, y position, and box height of a margin note written by the previous run of LaTeX. We need to compute the vertical shift to avoid overlapping with other boxes.

The algorithm is pretty dumb. For each margin note, we loop over all already-processed margin notes on the same page. If an overlap is detected, we advance the y-position of the margin note according to the height of the existing overlapping margin note, and re-start the loop.

This is also the main entry point into our vertical shift calculating algorithm. Set the `\phf@cc@marginvshift` to zero, this is where the calculated total vertical shift will be stored. Then call our algorithm main routine unless we're the first note on the page.[5]

[5] There's an edge case here that I haven't dealt with: In the case of footnotes or figure captions, the notes can appear out of order in the AUX file. Our algorithm only checks for overlap with note numbers that are less than the current note number. So if note 1.1 appears in the AUX file after note 1.2, then first 1.2 is processed, no shift is added (it ignores the fact that 1.1 doesn't exist yet), then 1.1 is processed and thinks it's the first note so doesn't add a shift either. If this ends up really being a problem there are ways to fix this but now I don't feel it's worth the effort. So you might have a

```
\def\phf@cc@marginlastmarginnoteypos{}
\newdimen\phf@cc@marginvshift
\def\newphfccmarginnote#1{\newphfccmarginnote@#1}
\def\newphfccmarginnote@#1#2#3#4{%
  \phf@cc@marginvshift=\z@
    \ifnum#2>0\relax
      \phf@cc@margin@calculatevshift{#1}{#2}{#3}{#4}%
    \else\fi
  \csxdef{phfccmn@#1.#2}{%
    {{#3}{#4}}{\number\phf@cc@marginvshift sp}%
  }%
}
```

\phfccmargininitialssep The dimension \phfccmargininitialssep is the minimum vertical separation between two margin notes initials. Redefine this length to whatever you like (e.g., \phfccmargininitialssep=1mm\relax or \setlength{\phfccmargininitialssep}{1mm}).

```
\newdimen\phfccmargininitialssep
\phfccmargininitialssep=2\p@
```

\phf@cc@margin@calculatevshift This is our note vertical-shifting algorithm. Compute the vertical shift by looping through existing notes on the present page. The temporary counter \phf@cc@tmp@cntiter is the iteration counter and iterates over the notes on the present page up to the current note. (If a note with earlier note is undefined, skip it, this can happen if the note is in a footnote or figure caption; see earlier footnote.)

The total vertical shift that needs to be applied to the present note (positive shift = downwards shift) is stored in the dimen \phf@cc@marginvshift, to be picked up by \newphfccmarginnote. Here #1=absolute page no., #2=current note no., #3=current note y position, #4=current note box height.

```
\newcount\phf@cc@tmp@cntiter
\def\phf@cc@margin@calculatevshift#1#2#3#4{%
  \phf@cc@marginvshift=\z@
  \phf@cc@tmp@cntiter=0
  \phf@cc@margin@calculatevshift@{#1}{#2}{#3}{#4}%
}
\def\phf@cc@margin@calculatevshift@#1#2#3#4{%
  \ifnum\phf@cc@tmp@cntiter<#2\relax
    \ifcsname phfccmn@#1.\the\phf@cc@tmp@cntiter\endcsname
      \expandafter\let\expandafter
          \phf@cc@tmp\csname phfccmn@#1.\the\phf@cc@tmp@cntiter\endcsname
      \edef\phf@cc@tmp@otherypos{%
        \expandafter\expandafter\expandafter\@firstoftwo
          \expandafter\@firstoftwo \phf@cc@tmp}%
```

note in a figure caption somehow overlapping with another note if they happen to be in different environments and yet on the same line.

```
      \edef\phf@cc@tmp@otherhgt{%
        \expandafter\expandafter\expandafter\@secondoftwo
          \expandafter\@firstoftwo \phf@cc@tmp}%
      \edef\phf@cc@tmp@othervshift{%
        \expandafter\expandafter\expandafter\@firstofone
          \expandafter\@secondoftwo \phf@cc@tmp}%
      \ifdim\dimexpr\phf@cc@tmp@otherypos-\phf@cc@tmp@othervshift
            >\dimexpr#3-#4-\phf@cc@marginvshift\relax
        \ifdim\dimexpr\phf@cc@tmp@otherypos-\phf@cc@tmp@otherhgt-\phf@cc@tmp@othervshift
              <\dimexpr#3-\phf@cc@marginvshift\relax
```

At this point, an overlap is detected between the current note that we're adjusting (note number #2 on page #1) and an earlier processed note (note number `\phf@cc@tmp@cntiter` on the same page). So we shift the current note so that it appears below the other note. The vertical shift is accumulated in the `\phf@cc@marginvshift` dimen, this will be stored into the label after we're done computing. Note: the y-position reported by `\pdflastypos` is from the *bottom*, not the top, so "higher on the page than" is ">"; but a positive margin vertical shift in marginnote shifts the margin note *downwards*.

Then reset the iteration counter to zero, so that we re-loop over all existing notes on the same page to re-check (with the new vertical shift) that we didn't generate an overlap with any earlier note (e.g., could happen on a two-column page with comments in both columns).

```
          \@tempdima=\dimexpr
              \phfccmargininitialssep
              +(#3-\phf@cc@marginvshift)%
              -(\phf@cc@tmp@otherypos-\phf@cc@tmp@otherhgt-\phf@cc@tmp@othervshift)%
              \relax
          %\message{*** phfcc DEBUG: overlap detected of note #1.#2 with
          %  note #1.\the\phf@cc@tmp@cntiter, increasing vshift of note
          %  number #2 on page #1 by \the\@tempdima \space ***}%
          \advance\phf@cc@marginvshift \@tempdima
          \phf@cc@tmp@cntiter=\z@\relax
        \else\fi
      \else\fi
    \else\fi
    \advance\phf@cc@tmp@cntiter 1\relax
    \phf@cc@margin@calculatevshift@{#1}{#2}{#3}{#4}%
  \fi
}
```

6.4 The main `\phfMakeCommentingCommand` macro

`\phfMakeCommentingCommand` Implementation of `\phfMakeCommentingCommand`. See main package documentation for usage details. The inner command definitions are kinda tricky because of all the possible syntax options.

\phfMakeCommentingCommand[⟨*color=blue,...*⟩]{⟨*cmdname*⟩}

```
\newcommand\phfMakeCommentingCommand[2][]{%
  \cmdKV@phfmkcc@startcmds={}%
  \cmdKV@phfmkcc@endcmds={}%
  \edef\x{\noexpand\phfcc@expandstylekeys{\phfcc@default@style}{\unexpanded{#1}}}%
  \x
  %\setrmkeys{phfmkcc}%
  \edef\x{\noexpand\setkeys{phfmkcc}{\the\phfcc@val@expanded@keyval}}%
  \x
  \phf@cc@getcolor{\cmdKV@phfmkcc@color}%
  \colorlet{#2color}{\phf@cc@thecolor}%
  \edef\phf@tmp@xxappcmd{\noexpand\appto\noexpand\phf@cc@usedcolors{\phf@cc@thecolor|}}%
  \phf@tmp@xxappcmd
  \colorlet{#2rmcolor}{#2color!70!gray!55!white}%
  \colorlet{#2rmcolorlink}{blue!40!#2rmcolor}%
  \edef\phf@tmp@xxx{\cmdKV@phfmkcc@initials}%
  \if\relax\detokenize\expandafter{\phf@tmp@xxx}\relax
    \csedef{phf@cc@val@#2@initials}{#2}%
  \else
    \csedef{phf@cc@val@#2@initials}{\cmdKV@phfmkcc@initials}%
  \fi
  \ifcsname phf@cc@formatinitialsstyle@%
      \detokenize\expandafter{\cmdKV@phfmkcc@formatinitials}\endcsname%
    \expandafter\def\expandafter\phf@cc@tmp@xzx\expandafter{%
      \csname phf@cc@formatinitialsstyle@%
          \detokenize\expandafter{\cmdKV@phfmkcc@formatinitials}\endcsname}%
    \csedef{phf@cc@val@#2@formatinitials}{\expandonce\phf@cc@tmp@xzx}%
  \else
    \csedef{phf@cc@val@#2@formatinitials}{\expandonce\cmdKV@phfmkcc@formatinitials}%
  \fi
  \csedef{phf@cc@val@#2@startcmds}{\the\cmdKV@phfmkcc@startcmds}%
  \csedef{phf@cc@val@#2@endcmds}{\the\cmdKV@phfmkcc@endcmds}%
  %\message{***** STARTCMDS ARE \detokenize\expandafter{\the\cmdKV@phfmkcc@startcmds} *****}%
  %\message{***** ENDCMDS ARE \detokenize\expandafter{\the\cmdKV@phfmkcc@endcmds} *****}%
  \csedef{phf@cc@val@#2@font}{\expandonce\cmdKV@phfmkcc@font}%
  \csedef{phf@cc@val@#2@spacing}{\expandonce\cmdKV@phfmkcc@spacing}%
  \csedef{phf@cc@val@#2@begin}{\expandonce\cmdKV@phfmkcc@begin}%
  \csedef{phf@cc@val@#2@end}{\expandonce\cmdKV@phfmkcc@end}%
  \csedef{phf@cc@val@#2@cfont}{\expandonce\cmdKV@phfmkcc@cfont}%
  \csedef{phf@cc@val@#2@cspacing}{\expandonce\cmdKV@phfmkcc@cspacing}%
  \csedef{phf@cc@val@#2@cbegin}{\expandonce\cmdKV@phfmkcc@cbegin}%
  \csedef{phf@cc@val@#2@cend}{\expandonce\cmdKV@phfmkcc@cend}%
  \csedef{phf@cc@val@#2@rmfont}{\expandonce\cmdKV@phfmkcc@rmfont}%
  \csedef{phf@cc@val@#2@rmspacing}{\expandonce\cmdKV@phfmkcc@rmspacing}%
  \csedef{phf@cc@val@#2@rmbegin}{\expandonce\cmdKV@phfmkcc@rmbegin}%
  \csedef{phf@cc@val@#2@rmend}{\expandonce\cmdKV@phfmkcc@rmend}%
  \csedef{phf@cc@val@#2@ifont}{\expandonce\cmdKV@phfmkcc@ifont}%
  \csedef{phf@cc@val@#2@ispacing}{\expandonce\cmdKV@phfmkcc@ispacing}%
  \csedef{phf@cc@val@#2@ibegin}{\expandonce\cmdKV@phfmkcc@ibegin}%
  \csedef{phf@cc@val@#2@iend}{\expandonce\cmdKV@phfmkcc@iend}%
```

```
  \csedef{phf@cc@val@#2@groupcmd}{\expandonce\cmdKV@phfmkcc@groupcmd}%
```

Once all the information was gathered and stored, define the commenting command itself (as well as its `\end*` counterpart). Make them robust.

```
  \csdef{#2}{\phf@cc@do{#2}}%
  \expandafter\robustify\csname #2\endcsname%
  \csdef{end#2}{\phf@cc@end}%
  \expandafter\robustify\csname end#2\endcsname%
}
```

`\phfDisableCommentingCommands` This command causes any commenting command declared with `\phfMakeCommentingCommand` to emit an error.

```
\newif\ifphf@cc@disabled
\phf@cc@disabledfalse
\def\phfDisableCommentingCommands{%
  \phf@cc@disabledtrue
}
```

6.5 Implementation of the commenting commands

Overall, the implementation of a commenting command goes as follows. First, gather all the info about the command invocation (e.g., important? starred? comment?) and then ultimately call `\phf@cc@begin` and `\phf@cc@end` which execute the formatting options.

Within each comment, the following flags are set by the parsing mechanism.

```
\newif\if@phfcc@iscomment
\newif\if@phfcc@isremoved
\newif\if@phfcc@isimportant
```

Everything happens within `\begingroup`…`\endgroup`. The group is opened at `\phf@cc@do` (the commenting command itself), and is closed by `\phf@cc@end`.

`\phf@cc@begin` The helper macro `\phf@cc@begin` is responsible for issuing all the necessary definitions for the comment to be typeset correctly, including initials, depending on whether it is a regular text addition, a text removal comment, an important comment, a commenting comment, etc. It inspects `\if@phfcc@iscomment`, `\if@phfcc@isremoved`, and `\if@phfcc@isimportant` to determine the comment type.

This is not the main comment entry point, the main entropy point is `\phf@cc@do` and `\phf@cc@begin` is called internally from there. *Execution is already enclosed in a local TEX group; a `\begingroup` has already been issued by `\phf@cc@do`.*

```
\def\phf@cc@begin{%
```

First, we set the author's color so that by default everything we do will use that color.

```
\color{\phf@cc@val@cur color}%
```

Issuing `\leavevmode` here is annoying because it adds vertical space if the command is issued in vertical mode right before an equation or theorem; but it is needed immediately after section and paragraph headings so that the comment style (eg sans font along with color etc.) is not applied to the section/paragraph heading itself.

```
    \leavevmode% beurk, see above
```

The comment typesetting is constructed in layers, each time adding commands to execute at the start and corresponding commmands to execute at the end. The start commands are executed in the specified order, with the end commands in the reverse order at the end. We use the `\phf@cc@helper@pushgrpcmds` helper to this effect.

Introducing the necessary spacing, and issuing the initials. The amount of spacing will depend on the type of comment; `\phf@cc@val@spacing` will be set appropriately below when setting comment-type specific settings.

```
\phf@cc@helper@pushgrpcmds{%
  \csname phf@cc@val@\phf@cc@val@cur @startcmds\endcsname
  \hspace{\phf@cc@val@spacing}%
  \phf@cc@showinitials%
}{%
  \hspace{\phf@cc@val@spacing}%
  \csname phf@cc@val@\phf@cc@val@cur @endcmds\endcsname
}%
```

See what comment type we're dealing with, and set the appropriate settings.

First, check if we have a commenting-comment. If so, see whether or not it is additionally marked as important and issue the relevant commands. Also set the relevant spacing.

```
\if@phfcc@iscomment
  \if@phfcc@isimportant
    \phf@cc@helper@pushgrpcmdsX{%
      \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @font\endcsname
      \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @cfont\endcsname
      \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @ifont\endcsname
      \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @cbegin\endcsname
      \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @ibegin\endcsname
    }{%
      \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @iend\endcsname
      \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @cend\endcsname
    }%
    \edef\phf@cc@val@spacing{\csname phf@cc@val@\phf@cc@val@cur @ispacing\endcsname}%
```

```
    \else
      \phf@cc@helper@pushgrpcmdsX{%
        \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @font\endcsname
        \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @cfont\endcsname
        \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @cbegin\endcsname
      }{%
        \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @cend\endcsname
      }%
      \edef\phf@cc@val@spacing{\csname phf@cc@val@\phf@cc@val@cur @cspacing\endcsname}%
    \fi
  \else
```

Now check if we have an important comment (which is not a commenty-comment).

```
    \if@phfcc@isimportant
      \phf@cc@helper@pushgrpcmdsX{%
        \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @font\endcsname
        \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @ifont\endcsname
        \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @ibegin\endcsname
      }{%
        \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @iend\endcsname
      }%
      \edef\phf@cc@val@spacing{%
        \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @ispacing\endcsname%
      }%
    \else
```

Not a comment, not important. Could be a piece of text marked for removal.

```
      \if@phfcc@isremoved
        \phf@cc@helper@pushgrpcmdsX{%
          \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @font\endcsname
          \noexpand\color{\phf@cc@val@cur rmcolor}%
          \noexpand\colorlet{docnotelinkcolor}{\phf@cc@val@cur rmcolorlink}%
          \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @rmfont\endcsname
          \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @rmbegin\endcsname
        }{%
          \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @rmend\endcsname
        }%
        \edef\phf@cc@val@spacing{\csname phf@cc@val@\phf@cc@val@cur @rmspacing\endcsname}%
      \else
```

We've got just a plain, old, inline marked text.

```
        \phf@cc@helper@pushgrpcmdsX{%
          \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @font\endcsname
          \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @begin\endcsname
        }{%
          \expandafter\noexpand\csname phf@cc@val@\phf@cc@val@cur @end\endcsname
        }%
```

```
        \edef\phf@cc@val@spacing{\csname phf@cc@val@\phf@cc@val@cur @spacing\endcsname}%
      \fi
    \fi
  \fi
```

Actually issue these commmands and kick off the typesetting of the comment.

```
  \phf@cc@val@grpcmds
}
\def\phf@cc@helper@pushgrpcmds#1#2{%
  \appto\phf@cc@val@grpcmds{#1}%
  \preto\phf@cc@val@grpendcmds{#2}%
}
\def\phf@cc@helper@pushgrpcmdsX#1#2{% expands arguments with edef
  \edef\phfcc@tmp@x{\noexpand\phf@cc@helper@pushgrpcmds{#1}{#2}}%
  \phfcc@tmp@x
}
```

`\phf@cc@end` Here we execute closing commands and close the group. `\phf@cc@end` is the last macro executed when the user invokes a comment, in any case.

```
\def\phf@cc@end{%
  \phf@cc@val@grpendcmds
  \endgroup
}
```

`\phf@cc@showinitials` This helper macro prints out the initials, if any.

```
\def\phf@cc@showinitials{%
  \edef\phf@cc@tmpx{%
    \expandafter\expandonce\csname phf@cc@val@\phf@cc@val@cur @initials\endcsname}%
  \expandafter\notblank\expandafter{\phf@cc@tmpx}{%
    \csname phf@cc@val@\phf@cc@val@cur @formatinitials%
        \expandafter\endcsname\expandafter{\phf@cc@tmpx}}{}%
}
```

`\phf@cc@do` **The macro `\phf@cc@do` is the main entry point of a commenting command.** Actually, the user-instantiated commenting commands are an alias of this command, with the user macro name as first argument. So the `\AlE` command defined in the main documentation of this package was in fact defined as an alias to `\phf@cc@do{AlE}`. This is were everything starts.

```
\def\phf@cc@do#1{%
```

First, we need to check if commenting commands were disabled with `\phfDisableCommentingCommands`. If this is the case, generate an error.

```
  \ifphf@cc@disabled
    \PackageError{phfcc}{Commenting commands have been disabled
      with \string\phfDisableCommentingCommands.}{}%
  \fi
```

Now we start for real. First, we open the TEX group. Then we start parsing the invocation syntax.

```
\begingroup
  \@phfcc@iscommentfalse
  \@phfcc@isremovedfalse
  \@phfcc@isimportantfalse
```

The macros `\phf@cc@val@grp*cmds` store the commands to be executed in `\phf@cc@begin` and `\phf@cc@end`. By default, they are empty. They are filled by relevant code when parsing the command invocation.

By default, there are no commands at group begin (`\phf@cc@val@grpcmds`), no commands at group end (`\phf@cc@val@grpendcmds`), and no spacing on either side of the typeset comment (`\phf@cc@val@spacing`).

```
  \def\phf@cc@val@grpcmds{}%
  \def\phf@cc@val@grpendcmds{}%
  \def\phf@cc@val@spacing{0pt}%
```

Store the name of the commenting command so we can look up the relevant comment settings. This is the name that was given to the commenting command itself, e.g. `phf`.

```
  \edef\phf@cc@val@cur{#1}%
```

Here we start the parsing of the type of comment. First, check for a star (e.g., `\phf*{...}`) to see if we're dealing with a removed block of text.

```
  \@ifstar\phf@cc@star\phf@cc@nostar%
}
```

If we don't have a star, continue parsing to see if we have an important piece of text or comment (`\phf!` sytnax).

```
\def\phf@cc@nostar{\@ifnextchar!\phf@cc@important\phf@cc@grp}
```

At this point, we have determined whether the invocation is starred (removed text) or important. The next execution points are `\phf@cc@star`, `\phf@cc@important`, or `\phf@cc@grp`.

`\phf@cc@star` `\phf@cc@important` If starred or if important, set up the inner group execution commands to the corresponding settings, and defer to `\phf@cc@grp`.

```
\long\def\phf@cc@star{%
  \@phfcc@isremovedtrue
  \phf@cc@grp}
\long\def\phf@cc@important!{%
  \@phfcc@isimportanttrue
  \phf@cc@grp}
```

\phf@cc@grp The command \phf@cc@grp parses how the argument content is given. Is it a comment ("[...]"), a single LaTeX argument group ("{ ... }"), or should we expect it to end with \endXXX?

```
\def\phf@cc@grp{%
  \@ifnextchar[\phf@cc@comment\phf@cc@grpnocomment%]
}
\def\phf@cc@grpnocomment{%
  \@ifnextchar\bgroup{\phf@cc@grpwarg}{\phf@cc@begin}}
```

The main commenting command has already defined the \endXXX command as an alias of \phf@cc@end, so in the case of the syntax \AlE ... \endAlE we can call \phf@cc@begin directly.

Now at this point we have determined how the argument is given, and we are calling one of \phf@cc@comment, \phf@cc@grpwarg (with argument), or \phf@cc@begin directly.

\phf@cc@grpwarg Format a simple comment specified as a normal LaTeX block, i.e., provided as an argument to the commenting command. (Like \AlE{text here} or \AlE!{important text} or \AlE*{some stuff to remove}.)

```
\long\def\phf@cc@grpwarg#1{%
  \phf@cc@begin
  \csname phf@cc@val@\phf@cc@val@cur @groupcmd\endcsname
  {#1}%
  \phf@cc@end
}
```

\phf@cc@comment If the argument is a comment, set up the comment font, begin/end chars, and go. Use xparse to get the argument in square brackets to ensure that the argument itself can contain matched square brackets.

```
\NewDocumentCommand\phf@cc@comment{+O{}}{%
  \@phfcc@iscommenttrue
  \phf@cc@begin%
  #1%
  \phf@cc@end%
}
```

6.6 Process package options

Initialization code for kvoptions for our package options.

```
\SetupKeyvalOptions{
   family=phfcc,
   prefix=phfcc@opt@
}
\DeclareBoolOption[true]{usemarginnote}
```

Process package options.

```
\ProcessKeyvalOptions*
```

Load the marginnote package unless requested not to. This package is required for the `margin` initials style.

```
\ifphfcc@opt@usemarginnote
  \RequirePackage{marginnote}
\fi
```

Change History

v1.0

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Symbols

A

B

C

D

E

F

H

I

K

L

M

N

P

R