{cprotect.sty}
\verbatim in \macro arguments*

Bruno Le Floch†

Released 2026/04/17

Contents

*This file describes version v1.0f, last revised 2026/04/17.

†E-mail: blflatex@gmail.com

1 Include \verb anywhere!

The cprotect package attempts to do something that should be impossible:[1] it allows you to put verbatim in footnotes[2], section titles, ... in a straightforward way. The section above was typeset using

```
\cprotect\section{Include\verb-\verb- anywhere!}
```

and the footnote was

```
...tes\cprotect\footnote{Like this one: \verb-!@#$%^&*()_+-.}
```

More generally, let us assume that you want to use verbatim text in the argument of some macro `\cs {`⟨*arg1*⟩`}`, and that this macro is normally allergic to verbatim. It probably has a good reason to be allergic, but using an auxiliary file, we can often solve the problem: if you want `{`⟨*arg1*⟩`}` to contain verbatim, just write `\cprotect\cs` or `\cprotect{\cs}` instead of `\cs`.[3] All the examples I give use very standard macros, but it should work with any macro.[4]

Nesting is supported: `\cprotect` cannot appear in a command argument unless this command is itself `\cprotect`ed:

```
\cprotect\section{Title \cprotect\emph{emphasized} word}
```

The first version of this package did not support macros with several arguments, nor macros with optional arguments. This new feature is accessed by giving an optional argument to `\cprotect`:

```
\cprotect[om]\section[Short title \verb-&^-]{Long title \verb-&^%$-}
```

[1] The UK TEX FAQ `http://www.tex.ac.uk/cgi-bin/texfaq2html` states:

> So `\verb` and `\begin{verbatim}` have to assume that they are getting the first look at the parameter text; if they aren't, TEX has already assigned category codes so that the verbatim command doesn't have a chance.

The argument is quite sound, but it can be circumvented if one can change category codes once assigned, which we do.

[2] Like this one: `!@#$%^&*()_+`.

[3] This solves most problems, for instance, verbatim text in section titles written as `\cprotect\section{`⟨*title*⟩`}` appears correctly in the table of contents. However, there are still bugs in capitalized headers.

[4] If you have a macro that works when it is not preceeded by `\cprotect`, but breaks down if you put `\cprotect` in front of it, I will be interested to know why.

Each `o` stands for an optional argument (within `[]`), and `m` stands for a mandatory argument (within `{}`). For example, one can write

```
\[
\cprotect[om]\sqrt[\verb-%c-]{\cprotect[mm]\frac{\verb-%a-}{\verb-%b-}}
\]
```

to typeset

$$\sqrt[\%c]{\frac{\%a}{\%b}}$$

In fact, the `[om]` argument specifier is optional.

Some commands will not accept reading from a file as a valid argument. For instance, the first argument of `\hyperlink` cannot be `\cprotect`-ed. Workaround: write `\cprotect{\hyperlink{`⟨*arg1*⟩`}}{`⟨*arg2*⟩`}`, in other words, `\cprotect` the combination of `\hyperlink` and its first argument. This trick can be tried when all else fails.

2 List of user commands

`\cprotect` Possibly the single most useful command is `\cprotect`, used as `\cprotect \foo {`⟨*arg1*⟩`}` or `\cprotect {`⟨*cslist*⟩`} {`⟨*arg1*⟩`}`. As described in the previous section, the first form behaves as `\foo {`⟨*arg1*⟩`}`, and the second as ⟨*cslist*⟩ `{`⟨*arg1*⟩`}`. The difference is that the argument `{`⟨*arg1*⟩`}` can now contain `\catcode` changes (e.g., induced by the `\verb` command and the `verbatim` environment). In fact, `{`⟨*arg1*⟩`}` is written to a file, and then read again as the argument of `\foo`. So using the ideas behind `\cprotect`, one could in principle build weird macros that read their arguments several times, with different `\catcode`s in effect each time. The e-TEX primitive `\scantokens` is most probably a better way to do these things, or you can try to use the experimental macro `\ReadVerbatimUntil`.

`\cMakeRobust` If you find yourself using a `\cprotect\foo` combination frequently, and you try to define `\bar` to mean `\cprotect\foo`, you will run into trouble. The technical point is that I made `\cprotect` "outer", which prevents it from being nested in the argument of another command.

However, the package provides `\icprotect`, which is a non-outer version of `\cprotect`. You can write

```
\outer\def\bar{\icprotect\foo}
```

(I like to make `\bar` outer, to make sure that it does not get called from inside a macro argument. It is not necessary.)

You can also use the typical

```
\let\oldfoo\foo
\outer\def\foo{\icprotect\oldfoo}
```

to redefine `\foo` itself instead of a new command `\bar`. The package provides a wrapper for this construction: `\cMakeRobust\foo` replaces `\foo` by a version which accepts verbatim. As a stupid example,

```
\newcommand{\expon}[1]{\mathrm{#1}^{#1}}
\cMakeRobust{\expon}
\(\expon{Hel\verb+|+o}\)
```

produces $\mathrm{Hel|o}^{Hel|o}$, and the verbatim is treated correctly.

`\cprotEnv` Some[5] environments do not really behave as environment in that they read their argument before acting on it. One such example is `amsmath`'s `align` environment. For these cases, use `\cprotEnv` as follows. Simply put `\cprotEnv \begin {`⟨*name*⟩`}` instead of `\begin {`⟨*name*⟩`}`. For example,

```
\cprotEnv\begin{align}
  x&=\begin{cases}
    1 &\text{if }\verb"@#%^&*"\\
    2 &\text{otherwise}
  \end{cases}
\end{align}
```

gives

$$x = \begin{cases} 1 & \text{if } \texttt{@\#\%\^{}\&*} \\ 2 & \text{otherwise} \end{cases} \tag{1}$$

Beware: the `align` environment does something weird at the measuring step, and `\cprotect` will not work very easily (see Section 4 on bugs below).

You can use `\CPTbegin` as a short-hand for `\cprotEnv\begin`, but this will fail when nesting the same environment twice (admittedly, this is rare): the inner nesting is distinguished by *precisely* the characters `\begin{`*name*`}`.

`\ReadVerbatimUntil` Finally, an odd half-deprecated macro. We use all of the auxiliary macros that are involved in its construction, but not `\ReadVerbatimUntil` anymore (the code has to be rewritten to make things clearer.

The aim of this command is to read a piece of text verbatim, until it reaches an end-marker. It then writes all that it has read to an auxiliary

[5] Todo: rethink `\cprotEnv`.

file for future use. The most naive approach has a major flaw when nesting is involved: when parsing `...{}` for instance, with an end-marker of `}`, we often wish to stop at the *second* closing bracket, not the first one. Thus, `\ReadVerbatimUntil` needs to be aware of a begin-marker. Also, for some applications we need to write things before and after the content read by `\ReadVerbatimUntil`. Finally, we want to do something once the file has been written, and possibly something before.

The syntax is thus `\ReadVerbatimUntil [`⟨*arg1*⟩`] {`⟨*arg2*⟩`} ^begin-text^endtext^begintag^endtag^`, followed by the text in which we look for `endtag`. The caret (`^`) can be any character. It delimits the four verbatim-like arguments of `\ReadVerbatimUntil`. The strings of letters `begintext` and `endtext` are pre- and ap-pended to the file. As mentionned before, `begintag` and `endtag` are used when reading the content, to determine how far `\ReadVerbatimUntil` should go. The mandatory argument `{`⟨*arg2*⟩`}` is executed once the whole text has been stored to a file. Typically, `{`⟨*arg2*⟩`}` involves reading the file one or more times, and acting on it in any way we like. The optional argument can be used for hacks such as changing where `cprotect` writes its files.

3 Technical points

[...] Will break if `^` does not have its usual catcode at the beginning and at the end of the `\cprotect`ed command.

Also, will break if `^^E` or `^^L` change catcodes. This choice of symbols can be changed by setting the options `gobbling-escape =` ⟨*letter*⟩, and `gobbling-letter =` ⟨*letter*⟩. The defaults are `gobbling-escape = E` and `gobbling-letter = L`.

4 Known bugs/limitations

Incompatibility with `\pagestyle{headings}`: when a chapter title is put as a header, it gets upper-cased. If you did `\cprotect\chapter{...}` as usual, the title has been stored to a file, but now, the name of the file is capitalized, and TEX cannot find it.

Issues with nesting of `\cprotect` in `align` environments: the `align` environment is being too clever with counting braces in its argument. But we do nasty things to braces, such as `\let\foo{`. This should in fact not be counted as a brace[6], but `align` has no way of knowing that. Thus, it expects a closing brace when there will be none. A workaround is to add `\iffalse}\fi` in the body of the align environment, after any

[6] In some sense, it defines `\foo` to be an opening brace.

\cprotect. Also, the align environment itself should be protected. Here is an example (three \cprotect, hence three closing braces):

```
\cprotEnv\begin{align*}
\sum\cprotect_{\verb"\k" = 1}\cprotect^{\verb"\N"} \verb"\k"
= \cprotect\frac{\verb"\N"(\verb"\N"+1)}{2} \iffalse}}}\fi
\end{align*}
```

$$\sum_{\k=1}^{\N} \k = \frac{\N(\N+1)}{2}$$

For commands with two or more arguments, it is only possible to put verbatim in one of the arguments (and the syntax is not great).

The argument of any command that is prefixed with \cprotect has to have balanced braces, *even when hidden inside verbatim environments, or behind a comment character*. For instance,

```
\cprotect\footnote{On the \verb:{: character}
```

would fail: \cprotect would see the brace in \verb:{:, and count it as an opening tag, which then has to be closed. This is most likely to lead \cprotect to gobble the whole file before complaining. Similarly,

```
\cprotect\footnote{On the \verb:!: %should it be }?
character}
```

would only gobble until the closing brace in %should it be }?, and I am not sure what error would be produced.

But we can use one ailment to cure the other! A correct way to typeset the first example is

```
\cprotect\footnote{On the \verb:{: character%}
}
```

if % is a comment character at the time it is read. A safer solution would be to use \iffalse}\fi instead of %}. It still requires to be sure that \ is an escape character when this piece of code is read, and can lead to problems if the previous token is \let for instance.

5 The code

1 ⟨*package⟩

5.1 Setting up

We first load a few packages

2 \RequirePackage{ifthen}
3 \RequirePackage{suffix}

We borrow the idea of quark from `expl3`: `\CPT@qend` expands to itself, useful for `\ifx` comparisons.

```
\def\CPT@qend{\CPT@qend}
\def\CPT@option@head#1=#2\CPT@qend{#1}
\def\CPT@option@tail#1=#2\CPT@qend{#2}
\DeclareOption*{%
  \ifthenelse{%
    \equal{gobbling-escape}{%
      \expandafter\CPT@option@head\CurrentOption=\CPT@qend}%
  }{%
    \edef\CPT@gobbling@escape{%
      \expandafter\CPT@option@tail\CurrentOption\CPT@qend
    }%
  }{%
    \ifthenelse{%
      \equal{gobbling-letter}{%
        \expandafter\CPT@option@head\CurrentOption=\CPT@qend}%
    }{%
      \edef\CPT@gobbling@letter{%
        \expandafter\CPT@option@tail\CurrentOption\CPT@qend
      }%
    }{%
      \PackageError{cprotect}{Unknown option \CurrentOption}{}%
    }%
  }%
}
\def\CPT@gobbling@escape{E}
\def\CPT@gobbling@letter{L}
\ProcessOptions\relax
```

Then we introduce the commands pertaining to writing files.[7]

We write files `\jobname-1.cpt`, `\jobname-2.cpt`, etc. in order.

```
\newwrite\CPT@WriteOut
\newcounter{CPT@WriteCount}
\edef\CPT@filename{\jobname.cpt}
\newcommand{\CPT@Write}[1]{%
  \immediate\openout\CPT@WriteOut=\CPT@filename%
  \newlinechar`\^^M%
  \immediate\write\CPT@WriteOut{#1}%
  \immediate\closeout\CPT@WriteOut%
  \expandafter\xdef\csname CPT@\CPT@filename\endcsname{%
    \noexpand\scantokens{#1}%
  }%
```

[7] To be rewritten. For the moment, we use a new file each time cprotect is used. Thus, many files. But this is needed if people want to nest cprotect's.

```
  %\expandafter\gdef\csname \string\CPT@\CPT@filename\expandafter\endcsname\expanda
  %  \expandafter\protect\csname CPT@\CPT@filename\endcsname}%
  %\expandafter\show\csname \string\CPT@\CPT@filename\endcsname%
  %\expandafter\show\csname CPT@\CPT@filename\endcsname%
}
```

The next command changes all catcodes to "other". It was adapted from `filecontents.sty`.

```
\newcommand{\makeallother}{%
  \count0=0\relax
  \loop
  \catcode\count0=12\relax
  \advance\count0 by 1\relax
  \ifnum\count0<256
  \repeat
}
```

`CPT@gobbling@escape` `CPT@gobbling@letter` e make the active character `^^L` of charcode 11 [letter...], and define it to expand to `\relax` (was hoping to make it invalid most of the time, but that fails) [...]. Also, failure if `^` is not of catcode "superscript".

`\CPT@escape@hat@hat@L` contains the string of characters `\^^L`, with `L` replaced by the value of `\CPT@gobbling@letter`.

`\CPT@hat@hat@E@hat@hat@L` contains the string of characters `^^E^^L` where `^^E` is the escape character, and `^^L` is the active charace.

We define `\CPT@hat@hat@E@hat@hat@L` to be empty: it will simply be used to gobble the initial space at the beginning of environments and hide the end-of-line token that eTEX inserts for every `\scantokens`.

```
{
  \catcode`\/=0
  /catcode`/\=12
  /catcode`/^=12
  /xdef/CPT@escape@hat@hat@L{\^^/CPT@gobbling@letter/space}
  /xdef/CPT@escape@hat@hat@E{\^^/CPT@gobbling@escape/space}
  /xdef/CPT@hat@hat@E@hat@hat@L{%
    ^^/CPT@gobbling@escape^^/CPT@gobbling@letter/space}
}
\expandafter\scantokens\expandafter{%
  \expandafter\catcode\expandafter`\CPT@escape@hat@hat@E=0}
\expandafter\scantokens\expandafter{%
  \expandafter\catcode\expandafter`\CPT@hat@hat@E@hat@hat@L=11}
\expandafter\scantokens\expandafter{%
  \expandafter\def\CPT@hat@hat@E@hat@hat@L{}}
```

5.2 \ReadVerbatimUntil

Both `\ReadVerbatimUntil` and its starred version (which we define using the `suffix.sty` package) take one optional argument `[`⟨*first-cs*⟩`]` and a mandatory argument `{`⟨*final-cs*⟩`}`. The ⟨*final-cs*⟩ is saved as `\CPT@commandatend` to be executed when we close the file (and the group).[8]

```
\newcommand\ReadVerbatimUntil[2][]{%
  \def\CPT@commandatend{#2}%
  \begingroup #1%
  \makeallother%
  \CPT@setup}
\WithSuffix\newcommand\ReadVerbatimUntil*[2][]{%
  \def\CPT@commandatend{#2}%
  \begingroup #1%
  \makeallother%
  \CPT@starsetup}
```

`\CPT@setup` reads the four "verbatim" arguments following `{`⟨*final-cs*⟩`}`, and stores them in this order as macros `\CPT@preText`, `\CPT@postText`, `\CPT@begin`, and `\CPT@end`. The macros which read each of these arguments need to be defined inside `\CPT@setup`, because I don't want any constraint on the delimiter. I could write a single macro that gobbles all four arguments at once, but this would require a crazy number of `\expandafter`s, so instead I do it one by one.

If the delimiter was given, say `^`, then we would define `\CPT@readBegin` as `\def \CPT@readBegin#1^{\def \CPT@begin{#1}\CPT@readEnd}`. But since `^` is not given explicitly, we need `\expandafter`s to expand it before the definition takes place. To avoid code repetition, I did it once and for all in the auxiliary macro `\CPT@def`. Note that a parameter of `##1` is somehow hidden inside `\CPT@def`, and that the `##1` inside the replacement text refer to the arguments of the `\CPT@read...` macros.

```
\newcommand{\CPT@def}[2]{\expandafter\def\expandafter#1%
  \expandafter##\expandafter1#2}
\newcommand{\CPT@setup}[1]{%
  \def\CPT@delimiter{#1}%
  \CPT@def\CPT@readPreText\CPT@delimiter{%
       \def\CPT@preText{##1}\CPT@readPostText}%
  \CPT@def\CPT@readPostText\CPT@delimiter{%
       \def\CPT@postText{##1}\CPT@readBegin}%
  \CPT@def\CPT@readBegin\CPT@delimiter{%
       \def\CPT@begin{##1}\CPT@readEnd}%
```

[8] It is not a straightforward `\aftergroup`, because I want this to be executed after another `\aftergroup` that comes later.

```
90   \CPT@def\CPT@readEnd\CPT@delimiter{%
91         \def\CPT@end{##1}\CPT@readContent}%
92   \CPT@readPreText%
93 }
94 \newcommand{\CPT@starsetup}[1]{\CPT@setup#1#1#1}
```

We also give the variant `\CPT@starsetup`, which has empty `\CPT@preText` and `\CPT@postText`.

When `\CPT@setup` is expanded, it will call `\CPT@readPreText`, `\CPT@readPostText`, `\CPT@readBegin`, and `\CPT@readEnd`, and finish with `\CPT@readContent`, which we describe now.

The counter `CPT@numB` will count the surplus of `begin-tag`s compared to `end-tag`s when we parse the text following `\CPT@readContent`. And `\CPT@store` is a macro that adds its argument to an other macro (I was too lazy to learn about token lists). The storage itself will be initialized later.

```
95 \newcounter{CPT@numB}
96 \newcommand{\CPT@store}[1]{\edef\CPT@storage{\CPT@storage#1}}
```

The macro `\CPT@readContent` is quite tricky: if the `begin-tag` and `end-tag` were one character, things would be easy: I would read one character at a time, and compare it to both `begin-tag` and `end-tag`, then either store it, and possibly increase or decrease `CPT@numB`, or decide that I am done if `CPT@numB` becomes negative.

Unfortunately, I want to use `\ReadVerbatimUntil` for environments, in which case the `begin-tag` is `\begin{myenv}` and the `end-tag` is `\end{myenv}`. So two options:

- code a standard string searching algorithm... I did not feel like it, but it might lead to a regexp package later on;
- use TEX's delimited parameters.

I did the latter, using `\CPT@def` again (we want to expand the string which delimits the parameter before doing the definition).

The details are ugly:

- gobble until the first `end-tag`,[9] and insert a fake `begin-tag`, as well as the quark guard `\CPT@qend`,
- inside what we gobbled, gobble `begin-tag`s until reaching the fake one (marked by the quark guard).
- continue until we have one more `end-tag` than `begin-tag`.

[9] This will fail for devious cases: if `begin-tag` is `abc` and `end-tag` is `bcd`, and `\CPT@readContent` is followed by `abcd...`: we will wrongly see `bcd` as an `end-tag`.

```
\newcommand{\CPT@readContent}{%
  \CPT@def\CPT@gobbleOneB\CPT@begin##2{%
    \ifx\CPT@qend##2\CPT@store{##1}\addtocounter{CPT@numB}{-1}%
    \else\CPT@store{##1\CPT@begin}\stepcounter{CPT@numB}%
    \expandafter\CPT@gobbleOneB\expandafter##2\fi}%
  %
  \CPT@def\CPT@gobbleUntilE\CPT@end{%
    \edef\CPT@tempi{##1\CPT@begin}%
    \expandafter\CPT@gobbleOneB\CPT@tempi\CPT@qend%
    \ifthenelse{\value{CPT@numB}<0}{%
      \CPT@store{\CPT@postText}%
      \CPT@Write{\CPT@storage}\endgroup%
      \CPT@commandatend%
    }{%
      \CPT@store{\CPT@end}\CPT@gobbleUntilE%
    }%
  }%
  \setcounter{CPT@numB}{0}%
  \def\CPT@storage{\CPT@preText}%
  \CPT@gobbleUntilE%
}
```

5.3 For macros: `\cprotect` and friends

`\cprotect` Equipped with `\ReadVerbatimUntil`, we are ready for the more practical macros. `\cprotect` cheats: it uses `{` and `}` as a `begin-tag` and `end-tag`. This works most of the time, but fails in cases such as those presented in Section 4 (in usual cases there are workarounds[10]).

We define `\cprotect` as `\outer`, as a check that it is the first to read its argument. A neat thing is that `\cprotect` will complain if it is nested inside anything, *except if the enclosing macro is itself* `\cprotect`*ed*.[11]

```
\outer\long\def\cprotect{\icprotect}
```

Normally, one should always use `\cprotect`. In some rare cases, it can be necessary to overcome the `\outer`ness of `\cprotect`. For this we provide `\icprotect`, where `i` stands for `i`nternal.

`\icprotect [`⟨*spec*⟩`] {`⟨`\cs`⟩`}` takes as a mandatory argument the control sequence `\cs` that we are protecting, and as a first, optional,

[10] But I could definitely not have the contents of this `.dtx` file as a (huge) footnote in some document: since `{` and `}` change `\catcode`s, it is unlikely that the numbers balance correctly.

[11] In particular, the author of this documentation had to use a devious trick (namely, have an ignored character in the midst of the name) to include even the name of the macro in the left margin.

argument the argument specification of `\cs`, in the `xparse` style: `m` for mandatory arguments, `o` for optionals, etc.[12] ***No spaces, i.e., `[om]`, not `[o m]`!***

```
\newtoks\CPT@commandatend@toks
\newcommand{\icprotect}[2][om]{%
  \def\CPT@argsig{#1}%
  \def\CPT@cs{#2}%
  \CPT@commandatend@toks{#2}%
  \def\CPT@commandatend{\CPT@read@args}% used by RVU.
  \CPT@commandatend%
}
\def\CPT@argsig@pop{%
  \edef\CPT@argsig{\expandafter\@gobble\CPT@argsig}%
}
```

Currently, `\CPT@read@args` is simply a synonym.

```
 %\newcommand\CPT@read@args{\CPT@read@m}
\newcommand\CPT@read@args{%
  \ifx\CPT@argsig\empty
  \expandafter\the\expandafter\CPT@commandatend@toks
  \else
  \expandafter\expandafter\expandafter\CPT@read@one
  \expandafter\CPT@argsig\expandafter\CPT@qend%
  \fi
}
\def\CPT@read@one#1#2\CPT@qend{%
  \def\CPT@argsig{#2}%
  \def\CPT@tempii{\csname CPT@read@#1\endcsname}% To make the \afterassignment simp
  \afterassignment\CPT@tempii\let\CPT@next=%
}
```

5.3.1 Mandatory arguments

First, we learn how to parse mandatory arguments. We define `\CPT@read@m` to read a mandatory (braced) argument. Note the use of `\bgroup` in `\CPT@read@m`: if the argument of `\cs` starts with an opening brace, it has been read early, and its `\catcode` will still be `1`.[13]

We check whether the next token is a brace or not:

- If it is, we discard the `{`, and the argument of `\cs` stops at the matching explicit closing brace `}`. (See `\CPT@read@mbeg` below, where `beg` stands for an opening brace.)

[12] Todo: list all the arguments that are supported: for the moment, `o` and `m`.

[13] I am not sure whether catcodes really matter in such an `\ifx`.

- If it is not, the argument of `\cs` is this token only, and there is no need for `\cprotection` (see `\CPT@read@mone` below).

```
\newcommand\CPT@read@m{%
  \ifx\CPT@next\bgroup%
  \expandafter\CPT@read@mbeg%
  \else%
  \expandafter\CPT@read@mone%
  \fi%
}
\def\CPT@read@mone{\CPT@cs\CPT@next}
```

Since `\ReadVerbatimUntil` makes everything into others, we need braces to be others when we define most macros in this Section. We thus need a few `\catcode` changes. Think of `{` → `(` and `}` → `)`.

```
\begingroup
  \catcode`\{=12 \catcode`\}=12
  \catcode`\(=1 \catcode`\)=2
  \gdef\CPT@other@bgroup({)
  \gdef\CPT@other@egroup(})
\endgroup
```

`\CPT@read@mbeg` to read a braced mandatory argument once the begin-group symbol has been read.

```
\def\CPT@read@mbeg{%
  \stepcounter{CPT@WriteCount}%
  \edef\CPT@filename{\jobname-\number\c@CPT@WriteCount.cpt}%
  \expandafter\expandafter\expandafter\CPT@commandatend@toks
  \expandafter\expandafter\expandafter{%
    \expandafter\the
    \expandafter\CPT@commandatend@toks
    % Input a file:
    \expandafter{%
      \expandafter\protect
      \expandafter\input
      \CPT@filename
      \relax
    }%
    % % Using \scantokens: requires '%' active.
    % \expandafter{%
    %   \csname \string\CPT@\CPT@filename\endcsname
    %   \expandafter\protect
    %   \expandafter\csname  CPT@\CPT@filename\endcsname
    % }%
  }%
  %\showthe\CPT@commandatend@toks%
```

We append {\protect \input ⟨*file name*⟩relax} at the end of the token list \CPT@commandatend@toks. Then we set \CPT@commandatend to be this token list. The role of \CPT@commandatend is to be executed once the matching } is found. Note the \protect, necessary for things to work well inside \section.

Since we \edefed, \CPT@filename contains the expanded version of the filename. We then use many \expandafters in order to expand the filename now: we are then sure that the right file will be input as an argument of \CPT@cs (this is the \cprotected command).

```
  \begingroup%
  \makeallother%
  \def\CPT@preText{}%
  \let\CPT@postText\CPT@hat@hat@E@hat@hat@L%
  \let\CPT@begin\CPT@other@bgroup%
  \let\CPT@end\CPT@other@egroup%
  \CPT@readContent%
}%
```

5.3.2 Optional (o) and delimited (d**) arguments

\CPT@read@o (for scanning standard optional arguments) is defined to be \CPT@read@d[].

```
\def\CPT@read@o{\CPT@read@d[]}
```

Now for delimited arguments, \CPT@read@d takes two arguments (the two delimiters). If the next token, \CPT@next is the same *character* as the opening delimiter #1, then we read the argument with \CPT@read@d@beg. Otherwise, we use \CPT@read@d@none.

```
\def\CPT@read@d#1#2{%
  \if\noexpand\CPT@next#1%
  \expandafter\CPT@read@d@beg%
  \else%
  \expandafter\CPT@read@d@none%
  \fi%
  {#1}{#2}%
}
\def\CPT@read@d@none#1#2{%
  \CPT@read@args\CPT@next%
}
```

\CPT@read@d@beg to read a delimited argument once (if) the beginning symbol \CPT@d@begin has been found. The begin-tag and end-tag used by \ReadVerbatimUntil and its friends are here \CPT@d@begin and \CPT@d@end. Errata: replaced by #1 and #2.

```
\def\CPT@read@d@beg#1#2{%
```

```
  \stepcounter{CPT@WriteCount}%
  \edef\CPT@filename{\jobname-\number\c@CPT@WriteCount.cpt}%
  \expandafter\expandafter\expandafter\CPT@commandatend@toks%
  \expandafter\expandafter\expandafter{%
    \expandafter\the%
    \expandafter\CPT@commandatend@toks%
    \expandafter #1%
      \expandafter\protect%
      \expandafter\input%
      \CPT@filename%
      \relax%
    #2%
  }%
```

Since we \edefed, \CPT@filename contains the expanded version of the filename. We then use many \expandafters in order to expand \CPT@d@begin, \CPT@d@end, and \CPT@filename the filename now: we are then sure that the right file will be input as an argument of \CPT@cs (this is the \cprotected command).

```
  \begingroup%
  \makeallother%
  \def\CPT@preText{}%
  \let\CPT@postText\CPT@hat@hat@E@hat@hat@L%
  \def\CPT@begin{#1}%
  \def\CPT@end{#2}%
  \CPT@readContent%
}%
```

Finally, the \cMakeRobust command is a mess, and could probably be improved, although... it works :). We make the new command \outer, and use the inner version \icprotect of \cprotect.

```
\newcommand{\cMakeRobust}[1]{%
  \def\CPT@cs@name{\expandafter\@gobble\string#1}%
  \expandafter\let\csname CPT@old@\CPT@cs@name\endcsname #1%
  \expandafter\outer\expandafter\def\csname\CPT@cs@name\endcsname{%
    \expandafter\icprotect\csname CPT@old@\CPT@cs@name\endcsname}%
}
```

5.4 For Environments: \cprotEnv\begin and \CPTbegin

\CPTbegin We introduce the command \CPTbegin, which has a behaviour close to the behaviour of \begin. Namely, \CPTbegin{env} gobbles its argument until it sees the matching \end, and it writes what it gobbled to a file. It then inputs the file between \begin{...} and \end{...}, as we can see from the definition of \CPT@commandatend.

As for the case of `\cprotect`, we need to setup the values of the four arguments of `\ReadVerbatimUntil` before reading the content. Since the `begin-tag` and `end-tag` depend on a parameter, it would be incredibly messy to try to expandafter the right things, so we define `\CPT@env@setup` to, well, setup the values of the four arguments of `\ReadVerbatimUntil`, and then skip directly to `\CPT@readContent`.

```
\newcommand{\CPTbegin}[1]{%
  \stepcounter{CPT@WriteCount}%
  \edef\CPT@filename{\jobname-\number\c@CPT@WriteCount.cpt}%
  \edef\CPT@commandatend{%
    \noexpand\begin\noexpand{\noexpand#1\noexpand}%
      \noexpand\expandafter\noexpand\protect%
      \noexpand\expandafter\noexpand\input \CPT@filename\relax%
    \noexpand\end\noexpand{\noexpand#1\noexpand}%
  }%
```

Here, we are defining `\CPT@commandatend` to be `\begin{#1}` `\expandafter` `\protect` `\expandafter` `\input` ⟨*filename*⟩ `\end{#1}`, where ⟨*filename*⟩ is `\CPT@filename`, fully expanded. Is there a cleaner way of doing that, given how far in the definition `\CPT@filename` is?

```
  \begingroup%
  \CPT@env@setup{#1}%
  \makeallother%
  \CPT@readContent%
}
```

As announced, `\CPT@env@setup`, defined with lots of catcode changes [*this paragraph is not up to date*]. Since the catcode of `\` changes, I need an extra escape character, which I take to be `/`. I use two groups so each group is opened and closed using the same escape character (this is technically irrelevant, but seems less messy).

```
\newcommand{\CPT@otherify}[1]{%
  \expandafter\expandafter\expandafter\@gobble
  \expandafter\string\csname #1\endcsname
}
\newcommand{\CPT@env@setup}[1]{%
  \def\CPT@temp{#1}%
  \edef\CPT@temp{\CPT@otherify{#1}}%
  \edef\CPT@temp{\expandafter\strip@prefix\meaning\CPT@temp}%
  \expandafter\CPT@env@setup@\expandafter{\CPT@temp}%
}
\def\CPT@env@setup@#1{%
  \let\CPT@preText\CPT@hat@hat@E@hat@hat@L%
  \let\CPT@postText\CPT@hat@hat@E@hat@hat@L%
  \edef\CPT@begin{\string\begin\string{#1\string}}%
```

```
256   \edef\CPT@end{\string\end\string{#1\string}}%
257 }
```

A final piece of code is needed so that \CPT-environments can be nested: in order to be able to nest, we need \ReadVerbatimUntil to be aware that our environment has been opened when \CPTbegin{foo} or \begin{foo} appear in the text. Given the rules for delimited parameters in TeX, this is quite difficult. A workaround is to replace \CPTbegin by something that ends in \begin shuch as \cprotEnv\begin as defined below. Thus, for nesting to work, you need to prepend every one of your env environments with \cprotEnv.

```
258 \def\cprotEnv\begin{\CPTbegin}
```

259 ⟨/package⟩

Change History

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.