The collect package*

Jonathan Sauer
jonathan.sauer@gmx.de

2004/09/10

Abstract

This file describes the collect package that makes it possible to collect text for later use.

Contents

1 Introduction

Suppose you are writing the specification of a programming language. Then you will surely insert definitions of the grammar (i.e. using the syntax package). Most

*This document corresponds to collect.sty v0.9, dated 2004/09/10.

likely you will insert the grammar for i.e. loops, conditions et cetera in the appropriate chapter, but for easy implementation of a parser, you will want to include the complete grammar as an appendix, so that one does not have to collect the complete grammar from the bits in each chapter.

Of course you could do this by hand using the copy-and-paste facility of your text editor. But this is cumbersome and errorprone if you modify the grammar afterwards, i.e. for a second version.

This package provides the `collect` environment that typesets its contents and collects it for later use as well.

In other situations you will want to save the contents of an environment in a macro. Then the `collectinmacro` environment should do the trick.

2 Usage

General note If an environment exists in a normal and a starred form (i.e. `collect` and `collect*`), then the former does not typeset the environment contents, while the latter typesets it. Note, too, that the normal and the starred version of an environment often have a different number of parameters.

Stripping of spaces All environments strip any leading space before the start of the text and any trailing space after the end of the text. Trailing `\par`s (i.e. resulting from a blank line at the end of the environment) are *not* removed.

2.1 `collect`

You can define several *collections*. Each collection can be used independently to collect text across the document. You define a collection using `\definecollection`, use it in the `collect` and `collect*` environment and typeset it using `\includecollection`.

Note The name of the collection is used as the suffix of the file that stores the collection (the complete name is ⟨*jobname*⟩`.`⟨*name)*⟩. So you should not name a collection `tex`, `log`, `dvi`, `pdf` and so on.

Note You can only include a collection after completely collecting it. If you include it and continue to collect afterwards, the collection will be cleared after its inclusion.

This can be used to recycle a collection: Use it in the first part of your document to collect text, typeset this text using `\includecollection` and then reuse the collection in the second part of your document to collect new text which will finally be typeset using `\includecollection` a second time.

`\definecollection` Usage: `\definecollection {`⟨*name*⟩`}`.
Defines a collection of the name ⟨*name*⟩.

`collect*` Usage: `\begin{collect*} {`⟨*name*⟩`} {`⟨*before*⟩`} {`⟨*after*⟩`} {`⟨*beforecol*⟩`} {`⟨*aftercol*⟩`}` ... `\end{collect*}`.

Collects the text inside the environment in the collection ⟨*name*⟩. Then typesets the text. ⟨*name*⟩ must have been defined using `\definecollection`.

⟨*before*⟩ and ⟨*after*⟩ are tokens inserted before and after the environment contents in the collection as well as the typeset text.

⟨*beforecol*⟩ and ⟨*aftercol*⟩ are tokens inserted before and after the environment contents only in the collection. This can be used to insert a section heading (⟨*beforecol*⟩) and a backref to the section where the text has been collected (⟨*aftercol*⟩).

Note The `collect*` environment uses a temporary file ⟨*jobname*⟩`.tmp` to temporarily store the environment contents and include it afterwards.[1]

`collect` Usage: `\begin{collect}` `{`⟨*name*⟩`}` `{`⟨*beforecol*⟩`}` `{`⟨*aftercol*⟩`}` ... `\end{collect}`.

Collects the text inside the environment in the collection ⟨*name*⟩. *Does not* typeset the text (therefore, the two parameters ⟨*before*⟩ and ⟨*after*⟩ are missing); except for that, this environment is identical to `collect*`.

`\includecollection` Usage: `\includecollection` `{`⟨*name*⟩`}`. Includes the collection ⟨*name*⟩. Afterwards, the collection is reset.

You can include the collection *before* it is actually collected in the same way you can include a table of contents at the beginning of your document. Simply say `\includecollection` and then use the `collect*` or `collect` environment to collect text. Then run LaTeX on your file *twice*. (the first time to collect the text, the second time to put it in the document using `\includecollection`)

2.2 Other environments

`collectinmacro` Usage: `\begin{collectinmacro}` `{`⟨*macro*⟩`}` `{`⟨*before*⟩`}` `{`⟨*after*⟩`}` ... `\end{collectinmacro}`.

Collects the contents of an environment inside a macro ⟨*macro*⟩ without typesetting it. If ⟨*macro*⟩ has been defined prior to using this environment, its previous meaning is lost. The new definition is global.

⟨*before*⟩ and ⟨*after*⟩ are tokens inserted before and after the environment contents in the macro.

Important note This environment differs slightly from `collect` and `collect*`: Some macros, i.e. `\verb`, do not work correctly when ⟨*macro*⟩ is used later on.[2]

3 Examples

3.1 `collect`

```
\begin{collect*}{tst}{Before\par}{\par After}{Before file\par}{\par After file}
This is a test
\end{collect*}
```

[1] This way, catcode changes inside the environment are honored.

[2] As the catcodes have already been set when collecting the contents and cannot be changed afterwards, at least not without eTeX.

This results in the following text typeset directly:

```
Before
This is a test
After
```

And following text is typeset when the collection `tst` is included using `\includecollection{tst}`:

```
Before file
Before
This is a test
After
After file
```

3.2 Other environments

```
\begin{collectinmacro}{\collectedtext}{Before\par}{\par After}
This is a test
\end{collectinmacro}
```

This results in no text typeset directly and the following text stored in the macro `\collectedtext`:

```
Before\par This is a test\par After
```

4 Creating your own environment

You can create your own environment based on any of the environments in this package, i.e. to create a `grammarpart` environment to typeset part of a grammar.

You can for example say:

```
\definecollection{gra}

\newenvironment{grammarpart}[1]{%
  \@nameuse{collect*}{gra}{%
    \emph{Start of grammar `#1'}\par%
  }{%
    \par\emph{End of grammar}\par%
  }{%
    \subsection{#1}%
  }{%
    \par See some section in the text.
  }%
}{%
  \@nameuse{endcollect*}%
}
```

This will create a `grammarpart` environment with one parameter, the part of the grammar defined. (i.e. 'Identifiers') At the start of the environment the text '*Start of grammar* ⟨*part*⟩' will be typeset, followed by the grammar (the contents of the environment), and finally a line '*End of grammar*' will be added.

When including the collected grammar parts using `\includecollection` each grammar part will be prefixed by a subsection heading bearing the grammar part as its title and suffixed by 'See some section in the text.' (this could be extended to include a backref using `\ref`⟨*label*⟩)

Note If you use any of the environments inside a custom environment to afterwards collect text with this custom environment (as in the example of the `grammarpart` environment above), you *must not* begin and end it using `\begin` and `\end`, or the environment contents will not be collected correctly. You must use `\@nameuse{`⟨*environment*⟩`}` and `\@nameuse{end`⟨*environment*⟩`}` as in the example above, or, if the environment is not starred, `\`⟨*environment*⟩ and `\end`⟨*environment*⟩.

Note You still can use the environments of this package inside other environments (i.e. a `itemize` environment) without any problems.

5 Notes

- If you use any of the environments inside your own environment, note that the end of the environment is executed, but not included in the collected text! (that's why the the environments `collect*` and `collectinmacro` have two parameters for including text before and after the environment, ⟨*before*⟩ and ⟨*after*⟩).

6 Implementation

6.1 Main environments and macros

6.1.1 `collect`

`\definecollection` Usage: `\definecollection {`⟨*name*⟩`}`

Defines a collection ⟨*name*⟩.

This macro allocates a `\newwrite CE@@`⟨*name*⟩`@out` and defines a `\newif CE@@`⟨*name*⟩`open`.

```
\newcommand{\definecollection}[1]{%
  \@ifundefined{CE@@#1@out}{%
    \expandafter\newwrite\csname CE@@#1@out\endcsname%
    \expandafter\newif\csname ifCE@@#1@open\endcsname%
    \csname CE@@#1@openfalse\endcsname%
  }{%
    \PackageError{collect}{Collection `#1' has already %
```

```
      been defined}{\@ehc}%
  }%
}
```

`collect*` Usage: `\begin{collect*}` {⟨*name*⟩} {⟨*before*⟩} {⟨*after*⟩} {⟨*beforecol*⟩} {⟨*aftercol*⟩} ... `\end{collect*}`.

```
\newenvironment{collect*}[5]{%
  \global\toks@{}%
  \def\CE@file{#1}%
  \def\CE@preenv{#2}%
  \def\CE@postenv{#3}%
  \def\CE@prefileenv{#4}%
  \def\CE@postfileenv{#5}%
  \CE@get@env@body@start%
}{%
```

What are we doing? We make sure the collection is open, then we save ⟨*beforecol*⟩, ⟨*before*⟩, the collected environment contents, ⟨*after*⟩ and ⟨*aftercol*⟩ in the collection file.

```
  \CE@ensure@opened{\CE@file}%
  \edef\@tempa{\csname CE@@\CE@file @out\endcsname}%
  \immediate\write\@tempa{\CE@meaning\CE@prefileenv}%
  \immediate\write\@tempa{\CE@meaning\CE@preenv}%
  \immediate\write\@tempa{\the\toks@}%
  \immediate\write\@tempa{\CE@meaning\CE@postenv}%
  \immediate\write\@tempa{\CE@meaning\CE@postfileenv}%
```

Now we repeat the same thing, but with the temporary file and writing only ⟨*before*⟩, the collected environment contents and ⟨*after*⟩. Then we include the temporary file.

Why so complicated, as we have the contents of the environment in `\toks@`? Because the catcodes might not be correct, i.e. if `\verb` is used inside the environment. So we have to read the environment contents again, which without eTEX is only possible using a temporary file.

```
  \immediate\openout\CE@tmp@out=\jobname.tmp%
  \immediate\write\CE@tmp@out{\CE@meaning\CE@preenv}%
  \immediate\write\CE@tmp@out{\the\toks@}%
  \immediate\write\CE@tmp@out{\CE@meaning\CE@postenv}%
  \immediate\closeout\CE@tmp@out%
  \@input{\jobname.tmp}%
  \par%
}
```

`collect` Usage: `\begin{collect}` {⟨*name*⟩} {⟨*beforecol*⟩} {⟨*aftercol*⟩} ... `\end{collect}`.

```
\newenvironment{collect}[3]{%
  \global\toks@{}%
  \def\CE@file{#1}%
  \def\CE@prefileenv{#2}%
```

```
  \def\CE@postfileenv{#3}%
  \CE@get@env@body@start%
}{%
```

As this environment, contrary to `collect*`, does not typeset its contents, we only write to the collection file:

```
  \CE@ensure@opened{\CE@file}%
  \edef\@tempa{\csname CE@@\CE@file @out\endcsname}%
  \immediate\write\@tempa{\CE@meaning\CE@prefileenv}%
  \immediate\write\@tempa{\the\toks@}%
  \immediate\write\@tempa{\CE@meaning\CE@postfileenv}%
}
```

`\includecollection` Usage: `\includecollection {`⟨*name*⟩`}`.

Includes the collection ⟨*name*⟩ which must have been defined previously using `\definecollection`. Afterwards, the collection is cleared.

```
\newcommand{\includecollection}[1]{%
  \CE@ensure@closed{#1}%
  \@input{\jobname.#1}%
}
```

6.1.2 Other environments

`collectinmacro` Usage: `\begin{collectinmacro} {`⟨*macro*⟩`} {`⟨*before*⟩`} {`⟨*after*⟩`} ... \end{collectinmacro}`.

Collects the contents of an environment inside a macro.

```
\newenvironment{collectinmacro}[3]{%
  \def\CE@destmacro{#1}%
  \def\CE@postenv{#3}%
```

We initialize the result with ⟨*before*⟩:

```
  \toks@{#2}%
  \CE@get@env@body@start%
}{%
```

We add ⟨*after*⟩:

```
  \toks@\expandafter\expandafter\expandafter{%
    \expandafter\the\expandafter\toks@\CE@postenv}%
```

Finally we globally define ⟨*macro*⟩ to contain the collected contents:

```
  \expandafter\expandafter\expandafter%
    \gdef\expandafter\CE@destmacro\expandafter{\the\toks@}%
  \toks@{}%
}
```

6.2 Internal environments and macros

We allocate a new `\newwrite` for the processing of a temporary file:

```
\newwrite\CE@tmp@out
```

`\CE@get@env@body@start` Starts the collecting of the contents of an environment. (the environment starts immediately after the macro)

```
\def\CE@get@env@body@start{%
  \let\@tempa\CE@get@env@body%
```

We may have to gobble leading spaces, therefore we check the first character in the environment:

```
  \futurelet\@tempb\CE@get@env@body@start@%
}
```

`\CE@get@env@body@start@` Support macro for `\CE@get@env@body@start`. Checks if the next token is a space, then calls `\CE@get@env@body@start@@`. Otherwise, the collecting of the environment contents is started.

```
\def\CE@get@env@body@start@{%
```

`\@sptoken` contains a single space and is defined in ltdefns.dtx:

```
  \ifx\@tempb\@sptoken%
    \expandafter\CE@get@env@body@start@@%
  \else%
    \expandafter\CE@get@env@body%
  \fi%
}
```

`\CE@get@env@body@start@@` Support macro for `\CE@get@env@body@start@`. Gobbles up any space following the macro, then start the collecting of the environment contents using `\CE@get@env@body`.

```
\def\CE@get@env@body@start@@{%
  \afterassignment\CE@get@env@body%
  \let\@tempb= %
}
```

`\CE@get@env@body` Usage: `\CE@get@env@body {`⟨*body*⟩`} \end {`⟨*envname*⟩`}`.

To ensure proper initialization, this macro should not be called directly; instead `\CE@get@env@body@start` should be called.

First we change the catcode of `Q` to 3 (math switch) in order to have a really unique character for parameter matching later on.[3] We do this in a group in order to easily restore the catcode later on and make all macro definitions global:

```
\bgroup
\catcode`\Q=3
```

Now we begin the macro:

```
\long\gdef\CE@get@env@body#1\end#2{%
```

Right at the beginning of the macro, we are at an `\end` (and `#1` contains the contents prior to it). We check if it ends the current environment (`#2` contains the name of the environment ended):

```
  \def\@tempb{#2}%
  \ifx\@tempb\@currenvir%
```

[3] Taken from 'Around the bend 15'

Yes, it ends the current environment. We add the contents to `\toks@` and are done. But we do not add the contents directly as there may be a trailing space left (multiple spaces have been collapsed into one space by TEX). So we call `\CE@get@env@body@` using delimited parameters (note that `Q` has catcode 3, therefore a normal `Q` is not matched).

What exactly is going on here? Suppose we have the text

```
Hello World_
```

(the `_` denotes the trailing space). Then `\CE@get@env@body@` is called like this:

```
\CE@get@env@body@Hello World Q Q
```

`\CE@get@env@body@` matches the parameters like this: `#1` is 'Hello World'; leaving `_Q` (added by the call from `\CE@get@env@body`) in the input. `\CE@get@env@body@` then calls `\CE@get@env@body@@` using `#1` and `Q`, resulting in the call:

```
\CE@get@env@body@@Hello WorldQ Q
```

`\CE@get@env@body@@` matches its parameter like this: `#1` is 'Hello World' again, and `#2` is the second `Q`.

But what happened to the space between the two `Q`s? As `#2` is not a delimited parameter, TEX skips spaces after matching the first `Q` until it reaches the second `Q`, thus gobbling up the space inbetween.[4]

Now suppose we have the text

```
Hello World
```

without any trailing space. Then `\CE@get@env@body@` is called like this:

```
\CE@get@env@body@Hello WorldQ Q
```

`\CE@get@env@body@` matches its parameters like this: `#1` is 'Hello WorldQ'. `\CE@get@env@body@` then calls `\CE@get@env@body@@` using `#1` and `Q`, resulting in the call:

```
\CE@get@env@body@@Hello WorldQQ
```

`\CE@get@env@body@@` matches its parameter like this: `#1` is 'Hello World' without the trailing 'Q', and `#2` is the second `Q`. The only difference to the situation described above is the missing space between the two `Q`s.

[4] c.f. TEXbook chapter 20.

Remember that the ‘Q’ is always ‘Q’ with catcode 3, thus no ‘Q’ in the environment contents is matched.

```
    \CE@get@env@body@#1Q Q%
    \def\@tempa{\end{#2}}%
  \else
```

No, it ends another environment. We add the contents as well as the `\end`⟨*environment*⟩ to `\toks@`. Then we continue collecting:

```
    \toks@\expandafter{\the\toks@#1\end{#2}}%
  \fi%
  \@tempa%
}
```

`\CE@get@env@body@` Support macro for `\CE@get@env@body`. See `\CE@get@env@body` for explanations.

```
\long\gdef\CE@get@env@body@#1 Q{%
  \CE@get@env@body@@#1Q%
}
```

`\CE@get@env@body@@` Support macro for `\CE@get@env@body@`. See `\CE@get@env@body` for explanations.

```
\long\gdef\CE@get@env@body@@#1Q#2{%
  \toks@\expandafter{\the\toks@#1}%
}
```

Finally we end the group, thus restoring the catcode of `Q`:

```
\egroup
```

`\CE@meaning` Usage: `\CE@meaning {`⟨*macro*⟩`}`.
Expands to the meaning of ⟨*macro*⟩.

```
\long\def\CE@meaning#1{%
  \expandafter\strip@prefix\meaning#1%
}
```

`\CE@ensure@opened` Usage: `\CE@ensure@opened {`⟨*name*⟩`}`.
Ensures that the file for collection ⟨*name*⟩ is opened.

```
\def\CE@ensure@opened#1{%
  \@ifundefined{ifCE@@#1@open}{%
    \PackageError{collect}{Collection `#1' has not been defined}{\@ehc}%
  }{%
    \csname ifCE@@#1@open\endcsname\else%
      \expandafter\immediate\expandafter\openout%
        \csname CE@@#1@out\endcsname=\jobname.#1%
      \expandafter\global\csname CE@@#1@opentrue\endcsname%
    \fi%
  }%
}
```

`\CE@ensure@closed` Usage: `\CE@ensure@closed {`⟨*name*⟩`}`.

Ensures that the file for collection ⟨*name*⟩ is closed.

```
\def\CE@ensure@closed#1{%
  \@ifundefined{ifCE@@#1@open}{%
    \PackageError{collect}{Collection `#1' has not been defined}{\@ehc}%
  }{%
    \csname ifCE@@#1@open\endcsname%
      \expandafter\immediate\expandafter\closeout%
        \csname CE@@#1@out\endcsname%
      \expandafter\global\csname CE@@#1@openfalse\endcsname%
    \fi%
  }%
}
```

Index

Numbers written in italic refer to the page where the corresponding entry is described, the ones underlined to the code line of the definition, the rest to the code lines where the entry is used.