A couple of things involving environments

Will Robertson

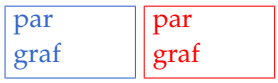
2008/06/18 v0.2

Abstract

This package provides two things, one for document authors and one for macro authors. For the document authors, a new method of defining environments that might be more convenient on occassion. And for the package writers, amsmath's `\collect@body` command, and a long version of the same, `\Collect@Body`.

1 Introduction

This packages provides new commands for defining environments. Here's a trivial example:




```
\NewEnviron{test}{%
  \fbox{\parbox{1.5cm}{\BODY}}
  \color{red}
  \fbox{\parbox{1.5cm}{\BODY}}}

\begin{test}
  par\par
  graf
\end{test}
```

`\RenewEnviron` can be used with the same syntax to redefine a preëxisting environment.

2 For the document author

LaTeX's standard method of defining environments looks like this (ignoring arguments for now):

`\newenvironment{`⟨*name*⟩`}{`⟨*pre code*⟩`}{`⟨*post code*⟩`}` .

The advantage to using environments is that their contents is not treated as a macro argument, so there are less restrictions on what can exist inside, and the processing can be more efficient for long pieces of document text.

The disadvantage of environments is that sometimes you really do want to collect up its body and apply some sort of command to the whole thing. This package provides a way to define such environments, and vo.2 of this package brings a new syntax:

`\NewEnviron{`⟨*name*⟩`}{`⟨*macro code*⟩`}[`⟨*final code*⟩`]` .

You saw an example in the introduction; the body of the environment is contained within the macro `\BODY`, and `[`⟨*final code*⟩`]` is the code executed at `\end{`⟨*name*⟩`}` (more on this later).

2.1 Environment arguments

If you want to use arguments to the environment, these are specified in the usual way:

`\NewEnviron{`⟨*name*⟩`}[`⟨*N. args*⟩`][`⟨*opt. arg.*⟩`]{`⟨*macro code*⟩`}[`⟨*final code*⟩`]` ,

where `{`⟨*macro code*⟩`}` has arguments `#1`, `#2`, ..., as per traditional LaTeX environment mandatory and optional arguments. Here's an example with two arguments; one optional argument (`#1`, which is `\today` if omitted) and one mandatory argument (`#2`):

Title
par
graf
(June 18, 2008)

Title
par
graf
(Yesterday)

```
\NewEnviron{test}[2][\today]{%
  \fbox{\parbox{3cm}{%
    \textbf{#2}\\
    \BODY\\
    (#1)}}}

\begin{test}{Title}
  par\par graf
\end{test}

\begin{test}[Yesterday]{Title}
  par\par graf
\end{test}
```

2.2 `[`⟨*final code*⟩`]`

This is the code executed at `\end{`⟨*name*⟩`}` of the environment. For the purposes of this package it is only designed (but is very useful indeed) for cleanup code such as space gobbling in the input text.

`\environfinalcode` This macro sets a default value for the `[`⟨*final code*⟩`]` (unless manually spec-

ified) in each subsequent environment created with `\NewEnviron`. The default is to define each new environment postfixed by `\ignorespacesafterend`, like this:

`\environfinalcode{\ignorespacesafterend}` .

Here's a silly example:


```
\environfinalcode{((finish))}
\NewEnviron{test}{\fbox{\parbox{3cm}{\BODY}}}
\begin{test}
  par\par
  graf
\end{test}
```

Careful, `\environfinalcode` cannot contain square brackets without first protecting them with braces (*e.g.*,

`\environfinalcode{[end]}`

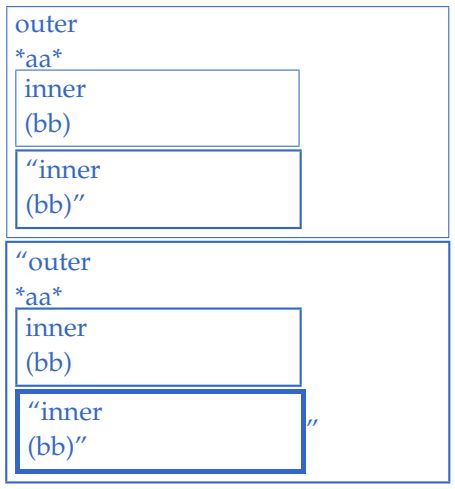
will not work but

`\environfinalcode{{[end]}}`

will). This is because the optional argument to `\NewEnviron` itself uses square brackets as argument delimiters.

3 Test

Here's an example to ensure everything that you'd think should work, in fact, does:




```
\NewEnviron{test}{%
  \fbox{\parbox{\linewidth-
    0.1cm*\currentgrouplevel}{\BODY}}
  \setlength\fboxrule{2\fboxrule}
  \fbox{\parbox{\linewidth-
    0.1cm*\currentgrouplevel}{``\BODY''}}}

\begin{test}
  outer\par
  \def\tmp#1{*#1*}%
  \tmp{aa}\par
  \begin{test}
    inner\par
    \def\tmp#1{(#1)}\tmp{bb}
  \end{test}
\end{test}
```

4 Backwards compatibility

In v0.1 of this package, the `\NewEnvironment` macro was used instead (described below). There were a number of limitations with the different approach that it used, the main ones being:

- Option processing occured with *every* instance of `#1` (which represented the environment body),
- Environment options were only available *after* the first instance of `#1`.

Please, please, don't use `\NewEnvironment`.

The code for this syntax is still loaded by this package for backwards compatibility, but in the next major release it will be moved to a conditional package option. Eventually, I'll probably delete it from the package altogether.

4.1 Previous code documentation

Just in case you need the documentation for the old syntax, here it is:

`\NewEnvironment{`⟨*name*⟩`}{`⟨*macro code*⟩`}` ,

where `{`⟨*macro code*⟩`}` has argument `#1` as everything inside the environment.

Now, this kind of environment definition makes collecting arguments a little cumbersome. Arguments are defined with a separate macro that 'gobbles up' the arguments inside the environment before the body is passed to `{`⟨*macro code*⟩`}`.

`\EnvironArgs{`⟨*name*⟩`}[`⟨*N. args*⟩`][`⟨*opt. arg.*⟩`]{`⟨*arg. macro code*⟩`}`

This follows the same syntax of defining a macro with several arguments and a possible optional argument at the beginning. Here's an example:

```
\NewEnvironment{test}{(#1)\tmp}
\EnvironArgs{test}[2][before]{\def\tmp{\par---#1/#2---}}
\begin{test}{after}
  par

  graf
\end{test}
```

(par graf) —before/after—

I've tried to ensure that whitespace is ignored at the appropriate places; without this additional code, there would be a space before 'par' and after 'graf' in the examples above.

Environments created with `\NewEnvironment` are ended by the command `\ignorespacesafterend`, which means that if they're used in a paragraph then the `\end{...}` command will gobble space after it. (If the environment is a paragraph on its own, there will be no difference.)

5 For the macro author

The amsmath package contains a macro that facilitates the functionality in the previous section, which package writers may wish to use directly. The canonical command is `\collect@body`, which I've also defined in `\long` form to be useable for multi-paragraph environments (`\Collect@Body`). Here's how it's used:

[hello
there]

```
\long\def\wrap#1{[#1]}
\newenvironment{test}{\Collect@Body\wrap}{}
\begin{test}
  hello

  there
\end{test}
```

And here's a crude example with environment arguments:

[—arg—
hello
there]

```
\long\def\wrap#1{[\arg#1]}
\def\arg#1{---#1---\par}
\newenvironment{test}{\Collect@Body\wrap}{}
\begin{test}{arg}
  hello

  there
\end{test}
```

File I
environ implementation

This is the package.

```
\ProvidesPackage{environ}[2008/06/18 v0.2 A new way to define environments]
```

Change History

v0.2

6 Begin

\environbodyname `{#1}`: control sequence

Changes the control sequence used to represent the environment body in its definition. Not to be used as a user command; but maybe one day it will be. Don't change this after defining any \NewEnviron environments!

```
\def\environbodyname#1{\def\env@BODY{#1}}
\environbodyname\BODY
```

\environfinalcode `{#1}`: code

This is the {⟨*code*⟩} that's executed by default at \end{⟨*env. name*⟩}:

```
\def\environfinalcode#1{%
  \def\env@finalcode{#1}}
\environfinalcode{\ignorespacesafterend}
```

\longdef@c LaTeX3-inspired shorthands.

```
\def\longdef@c#1{%
  \expandafter\long\expandafter\def\csname#1\endcsname}
```

\trim@spaces `{#1}`: tokens

Removes leading and trailing spaces from the (arbitrary) input. Thanks to Morten Høgholm.

```
\catcode`\Q=3
\long\def\trim@spaces#1{\romannumeral-`\q\trim@trim@\noexpand#1Q Q}
\long\def\trim@trim@#1 Q{\trim@trim@@#1Q}
\long\def\trim@trim@@#1Q#2{#1}
\catcode`\Q=11
```

7 \collect@body-related code

\collect@body Now, amsmath defines \collect@body for us. But that package may not be loaded, and we don't want to have to load the whole thing just for this one macro.

```
\unless\ifdefined\collect@body
  \newtoks\@emptytoks
  \newtoks\@envbody
  \def\collect@body#1{%
    \@envbody{\expandafter#1\expandafter{\the\@envbody}}%
    \edef\process@envbody{\the\@envbody\noexpand\end{\@currenvir}}%
    \@envbody\@emptytoks \def\begin@stack{b}%
    \begingroup
    \expandafter\let\csname\@currenvir\endcsname\collect@@body
    \edef\process@envbody{%
      \expandafter\noexpand\csname\@currenvir\endcsname}%
    \process@envbody
  }
  \def\push@begins#1\begin#2{%
    \ifx\end#2\else
      b\expandafter\push@begins
    \fi}
  \def\addto@envbody#1{%
    \global\@envbody\expandafter{\the\@envbody#1}}
  \def\collect@@body#1\end#2{%
    \edef\begin@stack{%
      \push@begins#1\begin\end \expandafter\@gobble\begin@stack}%
    \ifx\@empty\begin@stack
      \endgroup
      \@checkend{#2}%
      \addto@envbody{#1}%
    \else
      \addto@envbody{#1\end{#2}}%
    \fi
    \process@envbody}
\fi
```

\Collect@Body And now we define our own 'long' version.

```
\long\def\Collect@Body#1{%
  \@envbody{\expandafter#1\expandafter{\the\@envbody}}%
  \edef\process@envbody{\the\@envbody\noexpand\end{\@currenvir}}%
  \@envbody\@emptytoks \def\begin@stack{b}%
  \begingroup
  \expandafter\let\csname\@currenvir\endcsname\Collect@@Body
  \edef\process@envbody{%
```

```
    \expandafter\noexpand\csname\@currenvir\endcsname}%
  \process@envbody
}
\long\def\Push@Begins#1\begin#2{%
  \ifx\end#2\else
    b\expandafter\Push@Begins
  \fi}
\long\def\Addto@Envbody#1{%
  \global\@envbody\expandafter{\the\@envbody#1}}
\long\def\Collect@@Body#1\end#2{%
  \edef\begin@stack{%
    \Push@Begins#1\begin\end\expandafter\@gobble\begin@stack}%
  \ifx\@empty\begin@stack
    \endgroup
    \@checkend{#2}%
    \Addto@Envbody{#1}%
  \else
    \Addto@Envbody{#1\end{#2}}%
  \fi
  \process@envbody}
```

8 User-level syntax

`\NewEnviron` This is the new one.

```
\def\NewEnviron{%
  \let\env@newcommand\newcommand
  \let\env@newenvironment\newenvironment
  \env@NewEnviron}
\def\RenewEnviron{%
  \let\env@newcommand\renewcommand
  \let\env@newenvironment\renewenvironment
  \env@NewEnviron}
```

Input argument parsing The first optional argument:

```
\def\env@NewEnviron#1{%
  \@ifnextchar[
    {\env@new@i{#1}}
    {\env@new@iii{#1}{}}}
```

And the second:

```
\def\env@new@i#1[#2]{%
  \@ifnextchar[
    {\env@new@ii{#1}[#2]}
    {\env@new@iii{#1}{[#2]}}}
```

And the second: (cont.)

```
\def\env@new@ii#1[#2][#3]{%
  \env@new@iii{#1}{[#2][#3]}}
```

The final optional argument:

```
\long\def\env@new@iii#1#2#3{%
  \@temptokena={\env@new{#1}{#2}{#3}}%
  \@ifnextchar[{%
    \the\@temptokena
  }{%
    \expandafter\the\expandafter
      \@temptokena\expandafter[\env@finalcode]%
  }}
```

Environment creation code

`\env@new`

`{#1}`: name of the environment
`{#2}`: possible optional args (either '⟨*empty*⟩' or '`[N]`' or '`[N][default]`')
`{#3}`: environment code
`[#4]`: final code

```
\long\def\env@new#1#2#3[#4]{%
```

Define the new environment to Collect its body and execute `env@#1@parse` on it.

```
  \env@newenvironment{#1}{%
    \expandafter\Collect@Body\csname env@#1@parse\endcsname
  }{#4}
```

`env@#1@parse` executes the body twice: the first time to save the body while ignoring the arguments; and the second time to process the environment definition itself while ignoring the environment body:

```
  \longdef@c{env@#1@parse}##1{%
    \csname env@#1@save@env\endcsname##1\env@nil
    \csname env@#1@process\endcsname##1\env@nil}%
```

These must be defined on a per-environment basis in order to get the argument gobbling right: (because there are a variable number of arguments)

```
  \expandafter\env@newcommand
    \csname env@#1@save@env\endcsname#2{\env@save}%
  \expandafter\env@newcommand
    \csname env@#1@process\endcsname#2{#3\env@ignore}}
```

`\env@save`

If `\env@BODY` were variable, this macro would have to be saved for every environment definition individually; at the moment we just use a global definition. Use `\trim@spaces` to remove surrounding space:

```
\long\def\env@save#1\env@nil{%
```

```
  \expandafter\edef\env@BODY{%
    \unexpanded\expandafter
      \expandafter\expandafter{\trim@spaces{#1}}}}
```

This is the same as a `\@gobblenil` but long and less likely to exist in the environment body:

```
\long\def\env@ignore#1\env@nil{}
```

9 Backwards compatibility

`\NewEnvironment` `{#1}`: Environment name
`{#2}`: Macro definition applied to env. body
Here's our new environment definition macro. First of all wrap it up appropriately for new- or renew-

```
\newcommand\NewEnvironment{%
  \let\env@newenvironment\newenvironment
  \let\env@newcommand\newcommand
  \Make@Environment}
\newcommand\RenewEnvironment{%
  \let\env@newenvironment\renewenvironment
  \let\env@newcommand\renewcommand
  \Make@Environment}
```

And here we go:

```
\newcommand\Make@Environment[2]{%
```

Initial 'argument' parser, does nothing but remove leading space:

```
  \expandafter\let\csname env@args@#1\endcsname\ignorespaces
```

We use `\Collect@Body` to grab the argument (always `\long`)

```
  \env@newenvironment{#1}{%
    \expandafter\Collect@Body\csname env@@#1\endcsname}{\ignorespacesafterend}%
```

Now precede the env. body by the argument parsing command, which may or may not be defined in `\EnvironArgs` (and `\unskip` removes trailing space)

```
  \longdef@c{env@@#1}##1{%
    \csname env@@@#1\endcsname{%
      \csname env@args@#1\endcsname##1\unskip}}%
```

And then pass it all off to the environment macro (`#2`),

```
  \longdef@c{env@@@#1}##1{#2}}
```

`\EnvironArgs` `{#1}`: Environment name
`[#2]`: Number of arguments
`[#3]`: Optional argument

`{#4}`: Argument macro code
Tedious argument parsing:

```
\newcommand\EnvironArgs[1]{%
  \@ifnextchar[
    {\Env@Args{#1}}
    {\Env@Args{#1}[0]}}
```

Tedious argument parsing:

```
\long\def\Env@Args#1[#2]{%
  \@ifnextchar[
    {\Env@@@Args{#1}[#2]}
    {\Env@@Args{#1}[#2]}}
```

This is when there is no optional argument. In this case and the next, we simply define a command that is inserted when the argument body is processed (see `\NewEnvironment`). `\ignorespaces` removes leading space after the arguments.

```
\long\def\Env@@Args#1[#2]#3{%
  \expandafter\renewcommand\csname env@args@#1\endcsname[#2]{%
    #3\ignorespaces}}
```

Same as above when there is an optional argument:

```
\long\def\Env@@@Args#1[#2][#3]#4{%
  \expandafter\renewcommand\csname env@args@#1\endcsname[#2][#3]{%
    #4\ignorespaces}}
```