The mhsetup package*

Morten Høgholm

2004/10/10

Abstract

The mhsetup package provides tools for a LaTeX programming environment similar to the one described in expl3 on CTAN although not as extensive. It is a required part of both the mathtools and empheq packages.

1 The new internal syntax

The LaTeX3 package ldcsetup defines the command `\InternalSyntaxOn` which makes `_` and `:` letters and then automatically restores the category codes at the end of the package. This usually works fine but when you try to load amstext you will experience that TeX goes into an infinite loop. Packages containing code like `\@for\@tempa:=\@tempb\do{...}` will not work correctly either, thus we provide an alternative version here with the pair of commands `\MHInternalSyntaxOn` and `\MHInternalSyntaxOff`. They are to be used only as a pair, because `\MHInternalSyntaxOn` defines `\MHInternalSyntaxOff` so that it restores the category codes correctly.

`\MHInternalSyntaxOn`
`\MHInternalSyntaxOff`

2 Handling optional arguments

The standard behavior of scanning for optional arguments in LaTeX allows any number of spaces preceding the optional argument and that is not always good in math. For that reason amsmath makes sure that commands like `\\` disallows spaces before the optional argument but at the same time it fails to provide "safe" environments. What would you expect from the following input?

```
\[
  \begin{gathered}
    [v] = 100 \\
    [t] = 200
  \end{gathered}
\]
```

LaTeX will see the `[v]` as an optional argument of `gathered` and use it. In this case the test inside `gathered` checks if it's a `t` or `b` and if it's neither it'll choose

*This package has version number v1.0b, last revised on 2004/10/10.

`\vcenter` internally. So you get no warning, only missing output. Another example, this time from the `empheq` package used with its `overload` option: If preceding spaces are allowed, the input

```
\begin{gather}
  [a] = [b]
\end{gather}
```

results in the rather strange error message

```
! Package keyval Error: a undefined.
```

When using `\newcommand` etc. for defining commands and environments with optional arguments, the peek ahead is done by `\kernel@ifnextchar` (since LaTeX release 2003/12/01, else `\@ifnextchar`) and it is *hardwired at definition time* by `\@xargdef`. With the commands `\MHPrecedingSpacesOff` and `\MHPrecedingSpacesOn` `mhsetup` provides an interface to define commands and environments where the optional argument cannot have preceding spaces. You simply wrap them around the definitions:

`\MHPrecedingSpacesOff`
`\MHPrecedingSpacesOn`

```
\MHPrecedingSpacesOff
\newenvironment*{test}[1][default]{Start, arg: (#1)}{Ending.}
\MHPrecedingSpacesOn
\begin{test}
  [text]
\end{test}
\begin{test}[text]
\end{test}
```

Start, arg: (default) [text] Ending. Start, arg: (text) Ending.

It is of somewhat limited use in commands (control words in TeX terminology), because TeX discards the spaces. The exception is *control symbols* where TeX obeys following spaces but there are rather few of them available. All is not lost however. In the `aligned` environment from `amsmath` (shown below) a command is used as argument grabber.

```
\newenvironment{aligned}{%
  \let\@testopt\alignsafe@testopt
  \aligned@a
}{%
  \crcr\egroup
  \restorecolumn@
  \egroup
}
\newcommand{\aligned@a}[1][c]{\start@aligned{#1}\m@ne}
```

By applying our trick on the grabber function, we get a space obeying version:

```
\MHPrecedingSpacesOff
\renewcommand*\aligned@a[1][c]{\start@aligned{#1}\m@ne}
\MHPrecedingSpacesOn
```

This way a nested `aligned` environment is still safe from empty first cells.

3 First bits of a new programming environment

```
⟨*package⟩
```

3.1 The new internal syntax

\MHInternalSyntaxOn
\MHInternalSyntaxOff

Almost copy of \InternalSyntaxOn.

```
\def\MHInternalSyntaxOn{
  \edef\MHInternalSyntaxOff{%
  \catcode`\noexpand\~=\the\catcode`\~\relax
  \catcode`\noexpand\ =\the\catcode`\ \relax
  \catcode`\noexpand\^^I=\the\catcode`\^^I\relax
  \catcode`\noexpand\@=\the\catcode`\@\relax
  \catcode`\noexpand\:=\the\catcode`\:\relax
  \catcode`\noexpand\_=\the\catcode`\_\relax
  \endlinechar=\the\endlinechar\relax
  }%
  \catcode`\~=10\relax
  \catcode`\ =9\relax
  \catcode`\^^I=9\relax
  \makeatletter
  \catcode`\_=11\relax
  \catcode`\:=11\relax
  \endlinechar=` %
  \relax
}
\MHInternalSyntaxOn
\AtEndOfPackage{\MHInternalSyntaxOff}
```

3.2 Programming tools

The whole idea is to provide programming tools that are convenient but not yet widely available. I hope this'll be obsolete soon!

Firstly we setup a few helper functions.

\MH_let:NwN

An alias for \let.

```
\let\MH_let:NwN \let
```

\MH_let:cN

This one takes a \csname-\endcsname name and \lets it to a single macro. We'll use this to setup our conditionals.

```
\def\MH_let:cN #1#2{
  \expandafter\MH_let:NwN \csname#1\endcsname#2}
```

\MH_let:cc

This one has takes a \csname-\endcsname name and \lets it to a another \csname-\endcsname name. To be used in constructions with weird characters like * or alike in them and can take a \global prefix if wanted (we want that later on).

```
\def\MH_let:cc #1#2{
  \expandafter\MH_let:NwN\csname#1\expandafter\endcsname
  \csname#2\endcsname}
```

\MH_new_boolean:n
\MH_set_boolean_F:n
\MH_set_boolean_T:n
\MH_if_boolean:nTF
\MH_if_boolean:nT
\MH_if_boolean:nF

Sets up conditionals. For instance

\MH_new_boolean:n {⟨*name*⟩}

defines the boolean ⟨*name*⟩ but also the conditional `\if_boolean_`⟨*name*⟩`:` to be used in the ordinary

`\if_boolean_`⟨*name*⟩`:`
  ⟨*true code*⟩
`\else:`
  ⟨*false code*⟩
`\fi:`

There is also a more "LaTeX-like" interface available by using the commands

> `\MH_if_boolean:nT{`⟨*name*⟩`}{`⟨*arg*⟩`}`

which will execute the argument if the current value of the boolean is 'true' while

> `\MH_if_boolean:nF{`⟨*name*⟩`}{`⟨*arg*⟩`}`

is the equivalent with 'false'. Finally we have

> `\MH_if_boolean:nTF{`⟨*name*⟩`}{`⟨*true code*⟩`}{`⟨*false code*⟩`}`.

This is the interface I have used in this package.

Initially `\if_boolean_`⟨*name*⟩`:` is 'false'. This can be changed by saying

TeX:	`\boolean_`⟨*name*⟩`_true:` *or*
LaTeX:	`\MH_set_boolean_T:n{`⟨*name*⟩`}`

and changed back again by

TeX:	`\boolean_`⟨*name*⟩`_false:` *or*
LaTeX:	`\MH_set_boolean_F:n{`⟨*name*⟩`}`

And yes, we're also using alternative names for `\else` and `\fi` now. That way a simple search and replace will be all that is needed for this package to be a certified LaTeX3 package (well, maybe a little more is needed, but not much).

```
\def\MH_new_boolean:n #1{
  \expandafter\@ifdefinable\csname if_boolean_#1:\endcsname{
    \@namedef{boolean_#1_true:}
      {\MH_let:cN{if_boolean_#1:}\iftrue}
    \@namedef{boolean_#1_false:}
      {\MH_let:cN{if_boolean_#1:}\iffalse}
    \@nameuse{boolean_#1_false:}%
  }
}
\def\MH_set_boolean_F:n #1{ \@nameuse{boolean_#1_false:} }
\def\MH_set_boolean_T:n #1{ \@nameuse{boolean_#1_true:} }
\def\MH_if_boolean:nTF #1{
  \@nameuse{if_boolean_#1:}
    \expandafter\@firstoftwo
  \else:
    \expandafter\@secondoftwo
  \fi:
}
\def\MH_if_boolean:nT #1{
  \@nameuse{if_boolean_#1:}
    \expandafter\@firstofone
  \else:
    \expandafter\@gobble
  \fi:
}
\def\MH_if_boolean:nF #1{
  \@nameuse{if_boolean_#1:}
    \expandafter\@gobble
  \else:
    \expandafter\@firstofone
  \fi:
}
```

\if:w
\if_meaning:NN
\else:
\fi:
\if_num:w
\if_dim:w
\if_case:w
\or:

Copies of TeX primitives.

```
\@ifundefined{if:w}{\MH_let:NwN \if:w =\if}{}
\@ifundefined{if_meaning:NN}{\MH_let:NwN \if_meaning:NN =\ifx}{}
\@ifundefined{else:}{\MH_let:NwN \else:=\else}{}
\@ifundefined{fi:}{\MH_let:NwN \fi:=\fi}{}
\@ifundefined{if_num:w}{\MH_let:NwN \if_num:w =\ifnum}{}
\@ifundefined{if_dim:w}{\MH_let:NwN \if_dim:w =\ifdim}{}
\@ifundefined{if_case:w}{\MH_let:NwN \if_case:w =\ifcase}{}
\@ifundefined{or:}{\MH_let:NwN \or:=\or}{}
```

\MH_cs_to_str:N

Strip off the backslash of a macro name.

```
\def\MH_cs_to_str:N {\expandafter\@gobble\string}
```

\MH_protected:
\MH_setlength:dn
\MH_addtolength:dn

We might as well make use of some of the extended features from ε-TeX. We use `\dimexpr` for some simple calculations as it saves a lot of the scanning that goes on inside calc. The `\protected` primitive comes in handy when we want to declare a robust command, that cannot be 'robustified' with `\DeclareRobustCommand`. If we don't have ε-TeX we'll just let our private macros be aliases for the less effective alternatives.

```
\@ifundefined{eTeXversion}
  {
    \MH_let:NwN \MH_protected:\relax
    \def\MH_setlength:dn{\setlength}
    \def\MH_addtolength:dn{\addtolength}
  }
  {
    \MH_let:NwN \MH_protected:\protected
    \def\MH_setlength:dn #1#2{#1=\dimexpr#2\relax\relax}
    \def\MH_addtolength:dn #1#2{\advance#1 \dimexpr#2\relax\relax}
  }
```

\MH_keyval_alias_with_addon:nnnn
\MH_keyval_alias:nnn

A way to make aliases with keyval. This will come in handy later.

```
\def\MH_keyval_alias_with_addon:nnnn #1#2#3#4{
  \@namedef{KV@#1@#2}{\@nameuse{KV@#1@#3}#4}
  \@namedef{KV@#1@#2@default}{\@nameuse{KV@#1@#3@default}#4}}
\def\MH_keyval_alias:nnn #1#2#3{
  \MH_keyval_alias_with_addon:nnnn {#1}{#2}{#3}{}}
```

\MH_use_choice_i:nnnn
\MH_use_choice_ii:nnnn
\MH_use_choice_iii:nnnn
\MH_use_choice_iv:nnnn

I need to be able to pick up individual arguments in a list of four (similar to `\@secondoftwo`).

```
\def\MH_use_choice_i:nnnn #1#2#3#4{#1}
```

```
\def\MH_use_choice_ii:nnnn #1#2#3#4{#2}
\def\MH_use_choice_iii:nnnn #1#2#3#4{#3}
\def\MH_use_choice_iv:nnnn #1#2#3#4{#4}
```

\MH_nospace_ifnextchar:Nnn
\MH_nospace_nextchar:
\MH_nospace_testopt:nn
\MH_nospace_protected_testopt:n

Scanning for the next character but disallow spaces.

```
\long\def\MH_nospace_ifnextchar:Nnn #1#2#3{
  \MH_let:NwN\reserved@d=~#1
  \def\reserved@a{#2}
  \def\reserved@b{#3}
  \futurelet\@let@token\MH_nospace_nextchar:
}
\def\MH_nospace_nextchar:{
  \if_meaning:NN \@let@token\reserved@d
    \MH_let:NwN \reserved@b\reserved@a
  \fi:
  \reserved@b
}
\long\def\MH_nospace_testopt:nn #1#2{
  \MH_nospace_ifnextchar:Nnn[
    {#1}
    {#1[{#2}]}
}
\def\MH_nospace_protected_testopt:n #1{
  \if_meaning:NN \protect\@typeset@protect
    \expandafter\MH_nospace_testopt:nn
  \else:
    \@x@protect#1
  \fi:
}
```

\kernel@ifnextchar
\MH_kernel_xargdef:nwwn
\MH_nospace_xargdef:nwwn
\MHPrecedingSpacesOff
\MHPrecedingSpacesOn

The code for the space sensitive peek ahead.

```
\@ifundefined{kernel@ifnextchar}
  {\MH_let:NwN \kernel@ifnextchar \@ifnextchar}
  {}
\MH_let:NwN \MH_kernel_xargdef:nwwn \@xargdef
\long\def\MH_nospace_xargdef:nwwn #1[#2][#3]#4{
  \@ifdefinable#1{
     \expandafter\def\expandafter#1\expandafter{
          \expandafter
          \MH_nospace_protected_testopt:n
          \expandafter
          #1
          \csname\string#1\endcsname
          {#3}}
       \expandafter\@yargdef
          \csname\string#1\endcsname
           \tw@
           {#2}
           {#4}}}
\providecommand*\MHPrecedingSpacesOff{
  \MH_let:NwN \@xargdef \MH_nospace_xargdef:nwwn
}
\providecommand*\MHPrecedingSpacesOn{
  \MH_let:NwN \@xargdef \MH_kernel_xargdef:nwwn
}
⟨/package⟩
```