The **mhsetup** package*

Morten Høgholm

2007/12/03

**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.

The description below was made before the extensive changes made to the expl3 code available from the LaTeX Project website.

1 The new internal syntax

The \LaTeX3 package \texttt{ldcsetup} defines the command \texttt{//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 \texttt{@for@tempa:=@tempb\do{...}} will not work correctly either, thus we provide an alternative version here with the pair of commands \texttt{\MHInternalSyntaxOn} and \texttt{\MHInternalSyntaxOff}. They are to be used only as a pair, because \texttt{\MHInternalSyntaxOn} defines \texttt{\MHInternalSyntaxOff} so that it restores the category codes correctly.

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 \texttt{//} 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}]
```

\*This package has version number v1.2, last revised on 2007/12/03.
\LaTeX will see the \[v\] as an optional argument of \texttt{gathered} and use it. In this case the test inside \texttt{gathered} checks if it’s a \texttt{t} or \texttt{b} and if it’s neither it’ll choose \texttt{vcenter} internally. So you get no warning, only missing output. Another example, this time from the \texttt{empheq} package used with its \texttt{overload} option: If preceding spaces are allowed, the input

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

results in the rather strange error message

\begin{verbatim}
! Package keyval Error: a undefined.
\end{verbatim}

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

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

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 \texttt{control symbols} where \TeX obeys following spaces but there are rather few of them available. All is not lost however. In the \texttt{aligned} environment from \texttt{amsmath} (shown below) a command is used as argument grabber.

\begin{verbatim}
\newenvironment{aligned}{%\let@\testopt\alignsafe@\testopt
\aligned@a}
{\crcr\egroup\restorecolumn@}
\end{verbatim}

\section*{2}
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

\ProvidesPackage{mhsetup}%
[2007/12/03 v1.2 programming setup (MH)]

3.1 The new internal syntax

\MHInternalSyntaxOn
\MHInternalSyntaxOff

Almost copy of \InternalSyntaxOn.

\MHInternalSyntaxOn
\MHInternalSyntaxOff

\def\MHInternalSyntaxOn{
\edef\MHInternalSyntaxOff{%
\catcode'\noexpand\~=	he\catcode'\~\relax
\catcode'\noexpand\=	he\catcode'\ \relax
\catcode'\noexpand\^^I=	he\catcode'\^^I\relax
\catcode'\noexpand\@=	he\catcode'@\relax
\catcode'\noexpand\:=	he\catcode':'\relax
\catcode'\noexpand\_=	he\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
\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_{\textit{let}}: NwN \quad \text{An alias for } \texttt{let}.

25 \texttt{\textbackslash let}\MH_{\textit{let}}: NwN \texttt{\textbackslash let}

\MH_{\textit{let}}: cN \quad \text{This one takes a } \texttt{\textbackslash csname-\endcsname} \text{ name and } \texttt{\textbackslash lets} \text{ it to a single macro. We'll use this to setup our conditionals.}

26 \texttt{\def\MH_{\textit{let}}: cN \#1\#2{\expandafter\MH_{\textit{let}}: NwN \csname#1\endcsname#2}}

\MH_{\textit{let}}: cc \quad \text{This one has takes a } \texttt{\textbackslash csname-\endcsname} \text{ name and } \texttt{\textbackslash lets} \text{ it to a another } \texttt{\textbackslash csname-\endcsname} \text{ name. To be used in constructions with weird characters like * or alike in them and can take a } \texttt{\textbackslash global} \text{ prefix if wanted (we want that later on).}

28 \texttt{\def\MH_{\textit{let}}: cc \#1\#2{\expandafter\MH_{\textit{let}}: NwN\csname#1\expandafter\endcsname#2}}

\MH_{\textit{new boolean}}: n
\MH_{\textit{set boolean T}}: n
\MH_{\textit{set boolean F}}: n
\MH_{\textit{if boolean T}}: n
\MH_{\textit{if boolean F}}: n
\MH_{\textit{if boolean TF}}: n

\MH_{\textit{set boolean T}}: n\{\langle name\rangle\}
\MH_{\textit{set boolean F}}: n\{\langle name\rangle\}
\MH_{\textit{if boolean T}}: n\{\langle name\rangle\}{\langle arg\rangle}
\MH_{\textit{if boolean F}}: n\{\langle name\rangle\}{\langle arg\rangle}
\MH_{\textit{if boolean TF}}: n\{\langle name\rangle\}{\langle true code\rangle}{\langle false code\rangle}

Sets up conditionals. For instance

\texttt{\MH_{\textit{new boolean}}: n\{\langle name\rangle\}}

defines the boolean \langle name\rangle but also the conditional \texttt{\textbackslash if boolean_{\langle name\rangle}}: to be used in the ordinary

\texttt{\textbackslash if boolean_{\langle name\rangle}}:
\texttt{(true code)}
\texttt{\textbackslash else:}
\texttt{(false code)}
\texttt{\textbackslash fi:}

There is also a more “\texttt{\LaTeX}”-like’ interface available by using the commands

\texttt{\MH_{\textit{if boolean T}}: n\{\langle name\rangle\}}\{\langle arg\rangle\}

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

\texttt{\MH_{\textit{if boolean F}}: n\{\langle name\rangle\}}\{\langle arg\rangle\}

is the equivalent with ‘false’. Finally we have

\texttt{\MH_{\textit{if boolean TF}}: n\{\langle name\rangle\}{\langle true code\rangle}{\langle false code\rangle}}

This is the interface I have used in this package.

Initially \texttt{\textbackslash if boolean_{\langle name\rangle}}: is ‘false’. This can be changed by saying

\texttt{\LaTeX}: \texttt{\textbackslash boolean_{\langle name\rangle}.true:} or
\texttt{\texttt{L}\LaTeX}: \texttt{\MH_{\textit{set boolean T}}: n\{\langle name\rangle\}}

and changed back again by

\texttt{\LaTeX}: \texttt{\textbackslash boolean_{\langle name\rangle}.false:} or
\texttt{\texttt{L}\LaTeX}: \texttt{\MH_{\textit{set boolean F}}: n\{\langle name\rangle\}}
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 \LaTeXe\ package (well, maybe a little more is needed, but not much).

```latex
\def\MH_new_boolean:n #1\{
  \expandafter\@ifdefinable\csname if_boolean_#1:\endcsname{
    \@namedef{boolean_#1_true:}
      \iftrue
    \@namedef{boolean_#1_false:}
      \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:
  \if_num:w \if_dim:w \if_case:w \or:
  \MH_cs_to_str:N \MH_cs_to_str:N {\expandafter\@gobble\string}
```
We might as well make use of some of the extended features from \(\varepsilon\)-\TeX{}. We use \dimexpr for some simple calculations as it saves a lot of the scanning that goes on inside calc. The \texttt{protected} primitive comes in handy when we want to declare a robust command, that cannot be ‘robustified’ with \texttt{DeclareRobustCommand}. If we don’t have \(\varepsilon\)-\TeX{} we’ll just let our private macros be aliases for the less effective alternatives.

```latex
\@ifundefined{eTeXversion}{
\MH_keyval_alias_with_addon:nnnn\MH_keyval_alias:nnn\MH_use_choice_i:nnnn \MH_use_choice_ii:nnnn \MH_use_choice_iii:nnnn \MH_use_choice_iv:nnnn
\MH_nospace_ifnextchar:Nnn\MH_nospace_nextchar:\MH_nospace_testopt:nn\MH_nospace_protected_testopt:n
}{
\MH_setlength:dn\MH_addtolength:dn\MH_protected:relax
\def\MH_setlength:dn{\setlength}
\def\MH_addtolength:dn{\addtolength}
\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\MH_use_choice_i:nnnn \MH_use_choice_ii:nnnn \MH_use_choice_iii:nnnn \MH_use_choice_iv:nnnn
\MH_nospace_ifnextchar:Nnn\MH_nospace_nextchar:\MH_nospace_testopt:nn
\MH_nospace_protected_testopt:n
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).
\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:}
\long\def\MH_nospace_nextchar:{
\if_meaning:NN \@let@token\reserved@d
\MH_let:NwN \reserved@b\reserved@a
\def\reserved@b{#3}
\futurelet\@let@token\MH_nospace_nextchar:
}\fi:
\reserved@b
\MH_nospace_ifnextchar:}\MH_nospace_testopt:nn\MH_nospace_nextchar:nn
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@b{#2}
\def\reserved@b{#3}
\futurelet\@let@token\MH_nospace_nextchar:}
\if_meaning:NN \@let@token\reserved@d
\MH_let:NwN \reserved@b\reserved@a
\def\reserved@b{#1}
\futurelet\@let@token\MH_nospace_nextchar:}
\MH_nospace_testopt:nn\MH_nospace_ifnextchar:nnn
\MH_nospace_ifnextchar:nnn #1#2
\MH_nospace_ifnextchar:nnn {#1}
\MH_nospace_ifnextchar:nnn \MH_nospace_ifnextchar:nnn #1[[#2]]}
```
The code for the space sensitive peek ahead.

`\MH_group_align_safe_begin:`

\MH_group_align_safe_end:

```latex
\def \MH_group_align_safe_end: \ifnum0=`\fi
```

\MH_group_align_safe_begin:

\MH_group_align_safe_end: