

# The mhsetup package\*

Morten Høgholm

2007/12/03

## Abstract

The `mhsetup` package provides tools for a  $\text{\LaTeX}$  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  $\text{\LaTeX}$ 3 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  $\text{\TeX}$  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  $\text{\LaTeX}$  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}
```

---

\*This package has version number v1.2, last revised on 2007/12/03.

`\]`

L<sup>A</sup>T<sub>E</sub>X 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 `\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 L<sup>A</sup>T<sub>E</sub>X 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
\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 T<sub>E</sub>X terminology), because T<sub>E</sub>X discards the spaces. The exception is *control symbols* where T<sub>E</sub>X 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
}%
\crr\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

```

1 <package>
2 \ProvidesPackage{mhsetup}%
3 [2007/12/03 v1.2 programming setup (MH)]

```

#### 3.1 The new internal syntax

```

\MHInternalSyntaxOn Almost copy of \InternalSyntaxOn.
\MHInternalSyntaxOff
4 \def\MHInternalSyntaxOn{
5   \edef\MHInternalSyntaxOff{%
6     \catcode'\noexpand\~=\the\catcode'\~\relax
7     \catcode'\noexpand\ =\the\catcode'\ \relax
8     \catcode'\noexpand\^^I=\the\catcode'\^^I\relax
9     \catcode'\noexpand\@=\the\catcode'\@\relax
10    \catcode'\noexpand\:=\the\catcode'\:\relax
11    \catcode'\noexpand\_=\the\catcode'\_\relax
12    \endlinechar=\the\endlinechar\relax
13  }%
14  \catcode'\~=10\relax
15  \catcode'\ =9\relax
16  \catcode'\^^I=9\relax
17  \makeatletter
18  \catcode'\_=11\relax
19  \catcode'\:=11\relax
20  \endlinechar=' %
21  \relax
22 }
23 \MHInternalSyntaxOn
24 \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.
25 \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.
26 \def\MH_let:cN #1#2{
27   \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).
28 \def\MH_let:cc #1#2{
29   \expandafter\MH_let:NwN\csname#1\expandafter\endcsname
30   \csname#2\endcsname}

\MH_new_boolean:n  Sets up conditionals. For instance
\MH_set_boolean_F:n      \MH_new_boolean:n {<name>}
\MH_set_boolean_T:n
\MH_if_boolean:nTF  defines the boolean <name> but also the conditional \if_boolean_<name>: to be
\MH_if_boolean:nT      used in the ordinary
\MH_if_boolean:nF
\if_boolean_<name>:
  <true code>
\else:
  <false code>
\fi:

```

There is also a more “L<sup>A</sup>T<sub>E</sub>X-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 L<sup>A</sup>T<sub>E</sub>X3 package (well, maybe a little more is needed, but not much).

```

31 \def\MH_new_boolean:n #1{
32   \expandafter\ifdefinable\csname if_boolean_#1:\endcsname{
33     \namedef{boolean_#1_true:}
34       {\MH_let:cN{if_boolean_#1:}\iftrue}
35     \namedef{boolean_#1_false:}
36       {\MH_let:cN{if_boolean_#1:}\iffalse}
37     \nameuse{boolean_#1_false:}%
38   }
39 }
40 \def\MH_set_boolean_F:n #1{ \nameuse{boolean_#1_false:} }
41 \def\MH_set_boolean_T:n #1{ \nameuse{boolean_#1_true:} }
42 \def\MH_if_boolean:nTF #1{
43   \nameuse{if_boolean_#1:}
44   \expandafter\@firstoftwo
45   \else:
46   \expandafter\@secondoftwo
47   \fi:
48 }
49 \def\MH_if_boolean:nT #1{
50   \nameuse{if_boolean_#1:}
51   \expandafter\@firstofone
52   \else:
53   \expandafter\@gobble
54   \fi:
55 }
56 \def\MH_if_boolean:nF #1{
57   \nameuse{if_boolean_#1:}
58   \expandafter\@gobble
59   \else:
60   \expandafter\@firstofone
61   \fi:
62 }

\if:w Copies of TEX primitives.
\if_meaning:NN 63 \@ifundefined{if:w}{\MH_let:NwN \if:w=\if}{\fi}
\else: 64 \@ifundefined{if_meaning:NN}{\MH_let:NwN \if_meaning:NN=\ifx}{\fi}
\fi: 65 \@ifundefined{else:}{\MH_let:NwN \else:=\else}{\fi}
\if_num:w 66 \@ifundefined{fi:}{\MH_let:NwN \fi:=\fi}{\fi}
\if_dim:w 67 \@ifundefined{if_num:w}{\MH_let:NwN \if_num:w=\ifnum}{\fi}
\if_case:w 68 \@ifundefined{if_dim:w}{\MH_let:NwN \if_dim:w=\ifdim}{\fi}
\or: 69 \@ifundefined{if_case:w}{\MH_let:NwN \if_case:w=\ifcase}{\fi}
70 \@ifundefined{or:}{\MH_let:NwN \or:=\or}{\fi}

\MH_cs_to_str:N Strip off the backslash of a macro name.
71 \def\MH_cs_to_str:N {\expandafter\@gobble\string}

```

`\MH_protected:` We might as well make use of some of the extended features from  $\varepsilon$ -TeX. We use  
`\MH_setlength:dn` `\dimexpr` for some simple calculations as it saves a lot of the scanning that goes on  
`\MH_addtolength:dn` 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  $\varepsilon$ -TeX we’ll just let our private macros be aliases for the less  
effective alternatives.

```

72 \ifundefined{eTeXversion}
73 {
74   \MH_let:NwN \MH_protected:\relax
75   \def\MH_setlength:dn{\setlength}
76   \def\MH_addtolength:dn{\addtolength}
77 }
78 {
79   \MH_let:NwN \MH_protected:\protected
80   \def\MH_setlength:dn #1#2{#1=\dimexpr#2\relax\relax}
81   \def\MH_addtolength:dn #1#2{\advance#1 \dimexpr#2\relax\relax}
82 }

```

`\MH_keyval_alias_with_addon:nnnn` A way to make aliases with keyval. This will come in handy later.  
`\MH_keyval_alias:nnn`

```

83 \def\MH_keyval_alias_with_addon:nnnn #1#2#3#4{
84   \@namedef{KV@#1@#2}{\@nameuse{KV@#1@#3}#4}
85   \@namedef{KV@#1@#2@default}{\@nameuse{KV@#1@#3@default}#4}}
86 \def\MH_keyval_alias:nnn #1#2#3{
87   \MH_keyval_alias_with_addon:nnnn {#1}{#2}{#3}{}}

```

`\MH_use_choice_i:nnnn` I need to be able to pick up individual arguments in a list of four (similar to  
`\MH_use_choice_ii:nnnn` `\@secondoftwo`).  
`\MH_use_choice_iii:nnnn`  
`\MH_use_choice_iv:nnnn`

```

88 \def\MH_use_choice_i:nnnn #1#2#3#4{#1}
89 \def\MH_use_choice_ii:nnnn #1#2#3#4{#2}
90 \def\MH_use_choice_iii:nnnn #1#2#3#4{#3}
91 \def\MH_use_choice_iv:nnnn #1#2#3#4{#4}

```

`\MH_nospace_ifnextchar:Nnn` Scanning for the next character but disallow spaces.  
`\MH_nospace_nextchar:`  
`\MH_nospace_testopt:nn`  
`\MH_nospace_protected_testopt:n`

```

92 \long\def\MH_nospace_ifnextchar:Nnn #1#2#3{
93   \MH_let:NwN\reserved@d=~#1
94   \def\reserved@a{#2}
95   \def\reserved@b{#3}
96   \futurelet\@let@token\MH_nospace_nextchar:
97 }
98 \def\MH_nospace_nextchar:{
99   \if_meaning:NN \@let@token\reserved@d
100   \MH_let:NwN \reserved@b\reserved@a
101   \fi:
102   \reserved@b
103 }
104 \long\def\MH_nospace_testopt:nn #1#2{
105   \MH_nospace_ifnextchar:Nnn[
106     {#1}
107     {#1[{#2}]}

```

```

108 }
109 \def\MH_nospace_protected_testopt:n #1{
110   \if_meaning:NN \protect\@typeset@protect
111   \expandafter\MH_nospace_testopt:nn
112   \else:
113     \@x@protect#1
114   \fi:
115 }

\kernel@ifnextchar The code for the space sensitive peek ahead.
\MH_kernel_xargdef:nwn 116 \@ifundefined{kernel@ifnextchar}
\MH_nospace_xargdef:nwn 117 {\MH_let:NwN \kernel@ifnextchar \@ifnextchar}
\MHPrecedingSpacesOff 118 {}
\MHPrecedingSpacesOn 119 \MH_let:NwN \MH_kernel_xargdef:nwn \@xargdef
120 \long\def\MH_nospace_xargdef:nwn #1[#2][#3]#4{
121   \@ifdefinable#1{
122     \expandafter\def\expandafter#1\expandafter{
123       \expandafter
124       \MH_nospace_protected_testopt:n
125       \expandafter
126       #1
127       \csname\string#1\endcsname
128       {#3}}
129     \expandafter\@yargdef
130     \csname\string#1\endcsname
131     \tw@
132     {#2}
133     {#4}}
134 \providecommand*\MHPrecedingSpacesOff{
135   \MH_let:NwN \@xargdef \MH_nospace_xargdef:nwn
136 }
137 \providecommand*\MHPrecedingSpacesOn{
138   \MH_let:NwN \@xargdef \MH_kernel_xargdef:nwn
139 }

\MH_group_align_safe_begin:
\MH_group_align_safe_end: 140 \def \MH_group_align_safe_begin: {\iffalse{\fi\ifnum0='}\fi}
141 \def \MH_group_align_safe_end: {\ifnum0='{}\fi}

142 \</package>

```