The \texttt{mhsetup} package

Morten Høgholm (maintained by Lars Madsen and The LaTeX3 team)

2021/03/18

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

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

1 The new internal syntax

The \LaTeX{}3 package \texttt{ldcsetup} defines the command \texttt{\textbackslash 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 \texttt{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{\textbackslash MHInternalSyntaxOn} and \texttt{\textbackslash MHInternalSyntaxOff}. They are to be used only as a pair, because \texttt{\textbackslash MHInternalSyntaxOn} defines \texttt{\textbackslash 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 \texttt{amsmath} makes sure that commands like \textbackslash disallow 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{verbatim}
[\[v\] = 100 \[t\] = 200 \end{verbatim}

\footnote{This package has version number v1.4, last revised on 2021/03/18.}
\LaTeX{} will see the \texttt{[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}


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}{% 
\start@aligned{#1}\m@ne}
\newcommand{\aligned@a}[1][c]{% 
\langle\@testopt\alignsafe@testopt}
\alignedopt}{% 
\cr\egroup
\restorecolumn@}
\end{verbatim}

By applying our trick on the grabber function, we get a space obeying version:
This way a nested \texttt{aligned} environment is still safe from empty first cells.

\section{First bits of a new programming environment}

\subsection{The new internal syntax}

Almost copy of \texttt{InternalSyntaxOn}.

\begin{verbatim}
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}
\end{verbatim}

\subsection{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 set up a few helper functions.

\begin{verbatim}
\MH_let:NwN An alias for \texttt{\let}.
25 \let\MH_let:NwN \let
\end{verbatim}

\begin{verbatim}
\MH_let:cN This one takes a \texttt{\csname-\endcsname} name and \texttt{\let}s 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}
\end{verbatim}
\MH_let:cc This one takes a \cname\endcname name and \lets it to another \cname\endcname 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:nT \MH_if_boolean:nF

Sets up conditionals. For instance

\MH_new_boolean:n \MH_set_boolean_F:n \MH_set_boolean_T:n

\MH_if_boolean:nT \MH_if_boolean:nF \MH_if_boolean:nTF defines the boolean \langle name \rangle but also the conditional \MH_if_boolean_{\langle name \rangle}: to be used in the ordinary

\MH_if_boolean_{\langle name \rangle}: \langle true code \rangle \MH_else: \langle false code \rangle \MH_fi:

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

\MH_if_boolean:nTF\langle name \rangle\{(arg)\} which will execute the argument if the current value of the boolean is ‘true’ while

\MH_if_boolean:nF\langle name \rangle\{(arg)\} is the equivalent with ‘false’. Finally we have

\MH_if_boolean:nTF\{\langle true code \rangle\}{\langle false code \rangle}.

This is the interface I have used in this package.

Initially \MH_if_boolean_{\langle name \rangle}: is ‘false’. This can be changed by saying

\LaTeX: \MH_boolean_{\langle name \rangle}\_true: \textbf{or} \LaTeX: \MH_set_boolean_T:n\langle name \rangle

and changed back again by

\LaTeX: \MH_boolean_{\langle name \rangle}\_false: \textbf{or} \LaTeX: \MH_set_boolean_F:n\langle name \rangle

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

\def\MH_new_boolean:n #1{\expandafter@ifdefinable\csname MH_if_boolean_#1:1\endcsname{\@namedef{MH_boolean_#1\_true:}{\MH_let:cN{MH_if_boolean_#1:1}\iftrue}}\@namedef{MH_boolean_#1\_false:}{\MH_let:cN{MH_if_boolean_#1:1}\iffalse}\@nameuse{MH_boolean_#1\_false:}%;}}

\def\MH_set_boolean_F:n #1{\@nameuse{MH_boolean_#1\_false:}}
\MH_set_boolean:T:n \MH_if:boolean:nTF \MH_if:boolean:nT \MH_if:boolean:nF
\MH_if:w \MH_if_meaning:NN \MH_else:\MH_fi:\MH_if_num:w \MH_if_dim:w \MH_if_case:w \MH_or:
\MH_cs_to_str:N \MH_protected: \MH_setlength:dn \MH_addtolength:dn
\MH_if:w Copies of \TeX\ primitives.
\MH_if_meaning:NN \MH_else:\MH_fi:\MH_if_num:w \MH_if_dim:w \MH_if_case:w \MH_or:
\MH_cs_to_str:N Strip off the backslash of a macro name.
\MH_protected: \MH_setlength:dn \MH_addtolength:dn We might as well make use of some of the extended features from \eTeX. We use\dimexpr for some simple calculations as it saves a lot of the scanning that goes on inside \texttt{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 \eTeX\ we’ll just let our private macros be aliases for the less effective alternatives.
\ifundefined{\TeXversion}
{\MH_let:NwN \MH_protected:\relax
\def\MH_setlength:dn{\setlength}
\def\MH_addtolength:dn{\addtolength}
}
\else
{\MH_let:NwN \MH_protected:\protected
\def\MH_setlength:dn #1#2{#1=#2\dimexpr#1\relax\relax}
}
\fi
A way to make aliases with keyval. This will come in handy later.

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

Scanning for the next character but disallow spaces.

The code for the space sensitive peek ahead.
\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
}
\MH_group_align_safe_begin:
\MH_group_align_safe_end:
\def \MH_group_align_safe_begin: {\iffalse{\fi\ifnum0='}\fi}
\def \MH_group_align_safe_end: {\ifnum0='}\fi}

\MH_group_align_safe_begin:
\MH_group_align_safe_end: