

The hardwrap package

Kevin Godby

Will Robertson

2010/10/02

v0.1

Abstract

This package provides facilities for hard-wrapping text to a certain line width. The primary purpose is to make it easier for package authors to write informational messages for the console and log file; wrappers around `\PackageWarning` et al. are provided for this.

I	USER DOCUMENTATION	1	II	IMPLEMENTATION	5
1	Introduction	1	6	Required Packages	5
2	Wrapping text	1	7	Counters and variables	5
3	Wrapping log messages	2	8	Main procedure	6
4	Customizing the output	3	9	Utility Macros	9
5	Examples	3	10	Wrapping Log Messages	11

Part I USER DOCUMENTATION

§1 Introduction

The hardwrap package provides a macro for word-wrapping text. In addition, helper macros are available for package and document class authors to use in automatically wrapping informational, warning, and error messages.

§2 Wrapping text

The main function provided by this package is the `\HardWrap` command, which takes five arguments.

`\HardWrap { $\langle function \rangle$ } { $\langle width \rangle$ } { $\langle setup code \rangle$ } { $\langle newline \rangle$ } { $\langle text \rangle$ }`

This command will wrap $\langle text \rangle$ to a text block of $\langle width \rangle$ characters wide, inserting $\langle newline \rangle$ at the end of each line and processing the result with $\langle function \rangle$. The $\langle text \rangle$ is fully expanded before being hard-wrapped; while doing so, the $\langle setup code \rangle$ may be used to change local definitions for commands such as `\.`

Examples will be given in Section 5.

§3 Wrapping log messages

A common use case for the `\HardWrap` macro is to format the informational, warning, and error messages that are printed to the terminal and log file. In support of this, we’ve provided a simple interface for package and document class authors to do this.

`\GeneratePackageLogMacros[$\langle prefix \rangle$]{ $\langle package name \rangle$ }`
`\GenerateClassLogMacros [$\langle prefix \rangle$]{ $\langle class name \rangle$ }`

If the optional argument $\langle prefix \rangle$ is not given, it is set equal to $\langle package name \rangle$. These two commands will generate the following macros:

`\ $\langle prefix \rangle$ @info{ $\langle info \rangle$ }`
`\ $\langle prefix \rangle$ @info@noline{ $\langle info \rangle$ }`
`\ $\langle prefix \rangle$ @warning{ $\langle warning \rangle$ }`
`\ $\langle prefix \rangle$ @warning@noline{ $\langle warning \rangle$ }`
`\ $\langle prefix \rangle$ @error{ $\langle error \rangle$ }{ $\langle help \rangle$ }`

For instance, calling `\GeneratePackageLogMacros{mypackage}` will create macros called `\mypackage@info`, `\mypackage@warning`, etc. The arguments for the generated macros are the same as the arguments for `\PackageInfo{ $\langle package name \rangle$ }`, `\PackageWarning{ $\langle package name \rangle$ }`, etc. Additionally, info messages may be printed with `\ $\langle prefix \rangle$ @info@noline` in which L^AT_EX’s ‘on input line $\langle num \rangle$ ’ suffix is suppressed.

The `\GenerateClassLogMacros` command generates similar macros using `\ClassInfo{ $\langle class name \rangle$ }`, `\ClassWarning{ $\langle class name \rangle$ }`, etc.

Note that no punctuation is added after messages, unlike standard L^AT_EX. You are free to punctuate your messages as you wish.

These macros define `_` and `\.` locally inside these messages to mean, respectively, $\langle space \rangle$ and $\langle newline \rangle$. These redefinitions are stored in the macro `\HardWrapSetup`, which may be altered before executing `\Generate...LogMacros` to change the behaviour of the generated commands.

§4 Customizing the output

While `hardwrap` goes to some effort to determine the appropriate line lengths, you may wish to override the value found.

```
\setmaxprintline{<value>}
```

This macro takes an integer value which is subsequently used as the maximum line width allowed in the terminal output and log file. By default this value is 79.

§5 Examples

The command

```
\HardWrap{\PackageWarning{foobar}}{50}{\HardWrapSetup}{\MessageBreak}{%  
  Sed feugiat. Cum sociis natoque...;}
```

produces the following in the console output:

```
Package foobar Warning: Sed feugiat. Cum sociis natoque penatibus et magnis  
(foobar)               dis parturient montes, nascetur ridiculus mus. Ut  
(foobar)               pellentesque augue sed urna. Vestibulum diam eros,  
(foobar)               fringilla et, consectetur eu, nonummy id, sapien.  
(foobar)               Nullam at lectus. In sagittis ultrices mauris.  
(foobar)               Curabitur malesuada erat sit amet massa. Fusce  
(foobar)               blandit. Aliquam erat volutpat. Aliquam euismod.  
(foobar)               Aenean vel lectus. Nunc imperdiet justo nec  
(foobar)               dolor; on input line 102.
```

Compare this to that below without the manual wrapping; \TeX breaks lines at 79 characters without keeping words together: (e.g., ‘Vestibulum’ broken between lines two and three)

```
Package foobar Warning: Sed feugiat. Cum sociis natoque penatibus et magnis dis  
parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Ves  
tibulum diam eros, fringilla et, consectetur eu, nonummy id, sapien. Nullam at  
lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa.  
Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc  
imperdiet justo nec dolor; on input line 110.
```

The `\HardWrap` macro can also be useful when writing to an external file. For example, one may write

```
\newwrite\textfile
\immediate\openout\textfile=\jobname.txt\relax
\HardWrap{\immediate\write\textfile}{50}{\HardWrapSetup}{^^J}{%
  Sed feugiat. Cum sociis natoque...;}
\closeout\textfile
```

to write the text to a file after being hard-wrapped with carriage returns (^^J) after each line.

Part II IMPLEMENTATION

This is the package implementation.

§6 Required Packages

```
1 \RequirePackage{ifplatform}
```

§7 Counters and variables

`\hw@charcount`

`\hw@wordcount`

The `\hw@charcount` count holds the number of characters on the current line.
The `\hw@wordcount` count holds the number of characters in the current word.

```
2 \newcount\hw@charcount
3 \hw@charcount=-1\relax
4 \newcount\hw@wordcount
```

`\hw@currtext`

`\hw@currline`

`\hw@currword`

We store the current word, current line, and current wrapped text in the following macros:

```
5 \def\hw@currtext{}
6 \def\hw@currline{}
7 \def\hw@currword{}
```

`\hw@protected@newline`

This macro is called each time a line break is created. It typically holds `\MessageBreak` for log messages, but could be set to `\` for typeset text.

```
8 \protected\def\hw@protected@newline{}
```

`\hw@protected@space`

`\hw@expanding@space`

`\hw@kernel@space`

The `\hw@protected@space` definition of ‘space’ is designed to be switched for a real space later on using `\hw@kernel@space`. `\hw@expanding@space` is inserted into scratch variable as the ‘real’ space char.

```
9 \protected\def\hw@protected@space{ }
10 \def\hw@expanding@space{ }
11 \let\hw@kernel@space\space
```

`\hw@scanstop`

This is a ‘quark’ from `expl3` designed to delimit the scanning; it will never be executed, else an infinite loop results.

```
12 \def\hw@scanstop{\hw@scanstop}
```

§8 Main procedure

`\HardWrap`

Arguments: $\{\langle function \rangle\} \{\langle chars\ to\ wrap\ to \rangle\} \{\langle setup \rangle\} \{\langle newline \rangle\} \{\langle text \rangle\}$

This is the macro that does everything. Note that the `\space` is first made ‘protected’ and then restored again.

```
13 \newcommand\HardWrap[5]{%
14   \begingroup
15     \hw@maxprintline=#2\relax
16     \let\space\hw@protected@space
17     #3
18     \edef\@tempa{#5}%
19     \expandafter\hw@scan\@tempa\hw@scanstop
20     \def\hw@protected@newline{#4}
21     \let\space\hw@kernel@space
22     \@temptokena={#1}%
23     \expandafter\the\expandafter\@temptokena\expandafter{\hw@wrappedtext}
24   \endgroup
25 }
```

`\hw@scan`

Convenience wrapper for `\futurelet`.

```
26 \def\hw@scan{%
27   \futurelet\let@token\hw@process
28 }
```

`\hw@process`

The `\hw@process` macro contains the actual word-wrapping algorithm. The text is scanned token by token. Each token falls into one of three categories: (a) the stop token `\hw@scanstop`, (b) a space token, or (c) anything else.

```
29 \def\hw@process{%
```

If we encounter the `\hw@scanstop` token, then we've hit the end of the string. Swallow the stop token and stop processing.

```
30   \ifx\let@token\hw@scanstop\relax
31     \hw@process@end
32     \let\next\@gobble
```

If we find a space, add the word to the current line if it fits, otherwise insert a line break and put the word on its own line. Continue reading tokens.

```
33   \else
34     \ifx\let@token\@sptoken
35       \ifnum\numexpr(\hw@charcount+\hw@wordcount+1)\relax<\hw@maxprntline
36         \advance\hw@charcount by \hw@wordcount
37         \ifx\hw@currline\@empty
38           \protected@edef\hw@currline{\hw@currword}%
39         \else
40           \advance\hw@charcount by 1\relax % account for the space character
41           \protected@edef\hw@currline{\hw@currline\hw@expanding@space\hw@currword}%
42         \fi
43     \else
44       \hw@charcount=\hw@wordcount\relax
45       \protected@edef\hw@currtext{\hw@currtext\hw@currline\hw@protected@newline}%
46       \let\hw@currline\hw@currword
47     \fi
48     \hw@wordcount=1\relax
49     \let\hw@currword\@empty
50     \let\next\hw@dochar
```

`\hw@process`

(*cont.*)

If the token is neither the stop token nor a space, we'll just append it to the current word and continue reading tokens.

```
51   \else
52     \advance\hw@wordcount by 1\relax
53     \let\next\hw@dochar
54   \fi
55 \fi
56 \next
57 }
```

`\hw@dochar`

After a letter, the `\hw@dochar` macro just appends a token (non-space and non-stop token) to the current word. After a space token, however, the following argument could possibly be `\hw@scanstop`, so we need to special-case this branch. I have a feeling that a 'gobble-space' function is possible which would make this all a bit more elegant but this works for now.

```
58 \def\hw@dochar#1{%
59   \def\@tempa{#1}%
60   \ifx\@tempa\hw@scanstop
61     \hw@process@end
62   \else
63     \protected@edef\hw@currword{\hw@currword#1}%
64     \expandafter\hw@scan
65   \fi
66 }
```

`\hw@process@end`

The final stage of processing the text. We've just come to the end of the final word on the final line: add the word to the current line if it fits, otherwise insert a line break and put the word on its own line.

```
67 \def\hw@process@end{%
68   \ifnum\numexpr(\hw@charcount+\hw@wordcount+1)\relax<\hw@maxprintline\relax
69     \protected@edef\hw@wrappedtext{%
70       \hw@currtext
```


`\hw@process@end`

(*cont.*)

```
71     \ifx\hw@currline\@empty\else
72         \hw@currline\space
73     \fi
74     \hw@currword
75 }%
76 \else
77     \protected@edef\hw@wrappedtext{%
78         \hw@currtext\hw@currline\hw@protected@newline\hw@currword
79     }%
80 \fi
81 }
```

`\HardWrapSetup`

This is the command to use if you want to ‘special-case’ some meanings to be more appropriate inside message text. It is used by default for argument #3 in `\HardWrap`.

```
82 \def\HardWrapSetup{%
83     \let \ \hw@protected@space
84     \let \ \hw@protected@newline
85 }
```

§9 Utility Macros

`\hw@strlen`

A simple string-length macro.

```
86 \def\hw@strlen#1{%
87     \number\numexpr\hw@Ncharscan#1\hw@scanstop\relax
88 }
89 \def\hw@Ncharscan#1{%
90     \ifx#1\hw@scanstop
91         \expandafter\@gobble
92     \else
93         \expandafter\@firstofone
```

`\hw@strlen`

(*cont.*)

```
94 \fi
95 {+1\hw@Ncharscan}%
96 }
```

`\hw@maxprintline`

Some code to detect T_EX's *max_print_line* value. This doesn't work with MiK_TE_X (yet?), so we disable it under Windows always.

```
97 \newcount\hw@maxprintline
98 \ifwindows\else
99   \ifnum\pdfshellescape>0\relax
100     \hw@maxprintline=\@input"|kpsewhich -var-value=max_print_line"\relax
101   \fi
102 \fi
103 \ifnum\hw@maxprintline=0\relax % default
104   \hw@maxprintline=79\relax % default
105 \fi
```

`\setmaxprintline`

In case the code above borks the `\hw@maxprintline` value, the user can set it manually with the `\setmaxprintline` macro.

```
106 \newcommand*{\setmaxprintline}[1]{%
107   \hw@maxprintline=#1\relax
108 }
```

§10 Wrapping Log Messages

L^AT_EX informational, warning, and error messages are printed in the format:

```
Package <pkgname> Info: This is an informational message.  
<(<pkgname>)>          That spans multiple lines. The  
<(<pkgname>)>          \MessageBreak macro is used to split  
<(<pkgname>)>          the text across lines.  
←-----A-----> ←-----B----->  
←-----max_print_line----->
```

The maximum line length (*max_print_line*) is used by T_EX for all log file and terminal output. It defaults to 79 characters but may be changed by editing the `texmf.cnf` file.

The length of *A* is the sum of three values:

1. whether it's a class or package message: add 6 for class messages, and 8 for package messages;
2. the length of the package name;
3. the type of message: information (add 7), warning (add 10), or error (add 10).

The length of *B* is the difference between *max_print_line* and *A* plus one for the extra space between them. Note that the length of *B* for the warning and error text is the same.

`\hw@suffix`

This string is used as a suffix to L^AT_EX warnings and info messages to push the automatic 'on input line *<num>*' onto the next line. This makes writing grammatically correct messages somewhat easier.

```
109 \newcommand\hw@suffix{^^JThis message occurred}
```

`\GeneratePackageLogMacros`

`\GenerateClassLogMacros`

Shortcuts are provided for generating logging macros that automatically wrap the text provided to them. The `\GeneratePackageLogMacros` and `\GenerateClassLogMacros` calculate the various lengths of *B* appropriately.

<code>\GeneratePackageLogMacros</code>	<code>\GenerateClassLogMacros</code>	(cont.)
----------------------------------------	--------------------------------------	---------

```

110 \newcommand{\GeneratePackageLogMacros}[2][{}%
111   \hw@generate@logging@macros{package}{#1}{#2}%
112   {\hw@maxprintline-\hw@strlen{#2}-16}% info length
113   {\hw@maxprintline-\hw@strlen{#2}-19}% warning length
114 }

115 \newcommand{\GenerateClassLogMacros}[2][{}%
116   \hw@generate@logging@macros{class}{#1}{#2}%
117   {\hw@maxprintline-\hw@strlen{#2}-14}% info length
118   {\hw@maxprintline-\hw@strlen{#2}-17}% warning length
119 }

```

`\hw@generate@logging@macros`

And now for the code that generates all the logging macros. Arguments:

1. $\{\langle 'package' \text{ or } 'class' \rangle\}$
2. $\{\langle prefix \rangle\}$
3. $\{\langle package \text{ name} \rangle\}$
4. $\{\langle info \text{ message length} \rangle\}$
5. $\{\langle warning \text{ message length} \rangle\}$

The $\langle info \dots \rangle$ and $\langle warning \text{ message length} \rangle$ values correspond to the calculation of B as described above.

First of all, if the $\langle prefix \rangle$ is not specified then fall back to the $\langle package \text{ name} \rangle$:

```

120 \newcommand{\hw@generate@logging@macros}[5]{%
121   \def\@tempa{#2}\ifx\@tempa\@empty
122     \hw@generate@logging@macros@aux{#1}{#3}{#3}{#4}{#5}%
123   \else
124     \hw@generate@logging@macros@aux{#1}{#2}{#3}{#4}{#5}%
125   \fi
126 }

```

Finally, the main procedure. Info messages first:

```

127 \newcommand{\hw@generate@logging@macros@aux}[5]{%
128   \expandafter\edef\csname #2@info\endcsname##1{%
129     \noexpand\HardWrap

```

`\hw@generate@logging@macros` (cont.)

```

130     {\@nameuse{hw@#1@info}{#3}}
131     {\number\numexpr#4\relax}
132     {\unexpanded\expandafter{\HardWrapSetup}}
133     {\noexpand\MessageBreak}
134     {##1}%
135 }%
136 \expandafter\edef\csname #2@info@noline\endcsname##1{%
137     \noexpand\HardWrap
138     {\@nameuse{hw@#1@info@noline}{#3}}
139     {\number\numexpr#4\relax}
140     {\unexpanded\expandafter{\HardWrapSetup}}
141     {\noexpand\MessageBreak}
142     {##1}%
143 }%

```

Now warnings:

```

144 \expandafter\edef\csname #2@warning\endcsname##1{%
145     \noexpand\HardWrap
146     {\@nameuse{hw@#1@warning}{#3}}
147     {\number\numexpr#5\relax}
148     {\unexpanded\expandafter{\HardWrapSetup}}
149     {\noexpand\MessageBreak}
150     {##1}%
151 }%
152 \expandafter\edef\csname #2@warning@noline\endcsname##1{%
153     \noexpand\HardWrap
154     {\@nameuse{hw@#1@warning@noline}{#3}}
155     {\number\numexpr#5\relax}
156     {\unexpanded\expandafter{\HardWrapSetup}}
157     {\noexpand\MessageBreak}
158     {##1}%
159 }%

```

And finally errors.

In addition to the *<info>* and *<warning>* lengths, the `\PackageError` macro allows for additional text to be displayed when the user requests it. This text doesn't have anything prepended to each line, so the length of this text is the same as *max_print_line*.

```

160 \expandafter\edef\csname #2@error\endcsname##1##2{%

```

```

161 \noexpand\HardWrap
162   {\xdef\noexpand\hw@tempa}
163   {\number\numexpr#5\relax}
164   {\unexpanded\expandafter{\HardWrapSetup}}
165   {\noexpand\MessageBreak}
166   {\MessageBreak ##1}%
167 \noexpand\HardWrap
168   {\xdef\noexpand\hw@tempb}
169   {\the\hw@maxprintline}
170   {\unexpanded\expandafter{\HardWrapSetup}}
171   {\noexpand\MessageBreak}
172   {\MessageBreak ##2}%
173 \unexpanded{%
174   \@nameuse{hw@#1@error}{#3}{\hw@tempa}{\hw@tempb}%
175 }%
176 }%
177 }

```

Here are our wrappers for `\PackageInfo`, et al., which are used above to generalise the code a little. Note that these macros are `\protected`, which allows them to be used in an expanding context without a preceding `\noexpand`.

```

178 \protected\def\hw@class@info      #1#2{\ClassInfo   {#1}{#2\hw@suffix}}
179 \protected\def\hw@class@info@noline #1#2{\ClassInfo   {#1}{#2@gobbletwo}}
180 \protected\def\hw@class@warning    #1#2{\ClassWarning{#1}{#2\hw@suffix}}
181 \protected\def\hw@class@warning@noline #1#2{\ClassWarning{#1}{#2@gobbletwo}}
182 \protected\def\hw@class@error      #1#2{\ClassError   {#1}{#2}}

183 \protected\def\hw@package@info      #1#2{\PackageInfo {#1}{#2\hw@suffix}}
184 \protected\def\hw@package@info@noline #1#2{\PackageInfo {#1}{#2@gobbletwo}}
185 \protected\def\hw@package@warning    #1#2{\PackageWarning{#1}{#2\hw@suffix}}
186 \protected\def\hw@package@warning@noline #1#2{\PackageWarning{#1}{#2@gobbletwo}}
187 \protected\def\hw@package@error      #1#2{\PackageError {#1}{#2}}

```

Fin.