The hardwrap package

Will Robertson
⟨will.robertson@latex-project.org⟩

Kevin Godby
⟨kevin@godby.org⟩

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.

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. This package requires ε-TEX.

§2 Wrapping text

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

```
\HardWrap {⟨function⟩} {⟨width⟩} {⟨setup code⟩} {⟨newline⟩} {⟨text⟩}
```

This command will wrap ⟨*text*⟩ to a text block of ⟨*width*⟩ characters wide, inserting ⟨*newline*⟩ at the end of each line and processing the result with ⟨*function*⟩. The ⟨*text*⟩ is fully expanded before being hard-wrapped; while doing so, the ⟨*setup code*⟩ may be used to change local command definitions. Inside ⟨*text*⟩, you may use `\\` to force a new line and `\␣` to insert a space.

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[⟨prefix⟩]{⟨package name⟩}
\GenerateClassLogMacros   [⟨prefix⟩]{⟨class name⟩}
```

If the optional argument ⟨*prefix*⟩ is not given, it is set equal to ⟨*package name*⟩. These two commands will generate the following macros:

```
\⟨prefix⟩@info{⟨info⟩}
\⟨prefix⟩@info@noline{⟨info⟩}
\⟨prefix⟩@warning{⟨warning⟩}
\⟨prefix⟩@warning@noline{⟨warning⟩}
\⟨prefix⟩@error{⟨error⟩}{⟨help⟩}
```

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{`⟨*package name*⟩`}`, `\PackageWarning{`⟨*package name*⟩`}`, *etc*. Additionally, info messages may be printed with `\`⟨*prefix*⟩`@info@noline` in which LaTeX's '`on input line` ⟨*num*⟩' suffix is suppressed.

The `\GenerateClassLogMacros` command generates similar macros using `\ClassInfo{`⟨*class name*⟩`}`, `\ClassWarning{`⟨*class name*⟩`}`, *etc*.

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

As with the `\HardWrap` command, `\␣` and `\\` are defined locally inside these messages to mean, respectively, ⟨*space*⟩ and ⟨*newline*⟩.

Additional redefinitions are stored in the macro `\HardWrapSetup`, which may be altered before executing `\Generate...LogMacros` to change the behaviour of the generated commands. By default, `\HardWrapSetup` is defined as

```
\def\HardWrapSetup{%
  \def\MessageBreak{\\}%
  \def\newline{\\}%
}
```

§4 Customizing the output

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

Read on if you're curious what's behind the curtain.

```
⟨*package⟩
```

§6 Preliminaries

Intentionally kept to a minimum.

```
\IfFileExists{ifplatform.sty}{%
  \RequirePackage{ifplatform}
}{%
  \newif\ifwindows
  \IfFileExists{/dev/null}{\windowsfalse}{\windowstrue}
}
```

\hw@charcount, \hw@wordcount

These hold, respectively, the number of characters on the current line and the second the number of characters in the current word.

```
\newcount\hw@charcount
\newcount\hw@wordcount
```

\hw@currtext, \hw@currline, \hw@currword

Used to store the current word, current line, and current wrapped text.

```
\def\hw@currtext{}
\def\hw@currline{}
\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.

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

\hw@protected@space, \hw@expanding@space

The `\hw@protected@space` definition of 'space' is designed to be switched for a real space later on using `\hw@expanding@space`, which is also inserted into scratch variables as the 'real' space char.

`\hw@protected@space`, etc. (*cont.*)

```
\protected\def\hw@protected@space{ }
\let\hw@expanding@space\space
```

`\hw@insert@newline`

This is a placeholder to show where manual newlines are inserted. (It will never be executed.)

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

`\hw@scanstop`

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

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

§7 Utility Macros

`\hw@strlen`, `\hw@strlen@of`

A simple string-length macro.

```
\def\hw@strlen#1{%
  \numexpr0\hw@Ncharscan#1\hw@scanstop\relax
}
\def\hw@Ncharscan#1{%
 \ifx#1\hw@scanstop
   \expandafter\@gobble
 \else
   \expandafter\@firstofone
 \fi
 {+1\hw@Ncharscan}%
}
\def\hw@strlen@of#1{%
  \expandafter\hw@strlen\expandafter{#1}%
}
```

`\hw@maxprintline`

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

```
\newcount\hw@maxprintline
\ifwindows\else
```

\hw@maxprintline (*cont.*)

```
  \ifnum\pdfshellescape>0\relax
    \hw@maxprintline=\@@input"|kpsewhich -var-value=max_print_line"\relax
  \fi
\fi
\ifnum\hw@maxprintline=0\relax
  \hw@maxprintline=79\relax % default
\fi
```

\setmaxprintline

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

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

§8 Main procedure

\HardWrap

Arguments:

1. {⟨*function*⟩}
2. {⟨*chars to wrap to*⟩}
3. {⟨*setup*⟩}
4. {⟨*newline*⟩}
5. {⟨*text*⟩}

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

```
\newcommand*\HardWrap[5]{%
  \begingroup
    \hw@maxprintline=#2\relax
```

Replacements for user commands:

```
    \let\space\hw@protected@space
    \def\ {\space}%
    \let\\\hw@insert@newline
```

To avoid problems with repeated edef with arbitrary csnames:

```
    \let\noexpand\string
```

\HardWrap (*cont.*)

Execute the custom setup, and then fully expand the text to be wrapped, turning `\protected` macros into strings. (Fully `\protected` macros will still be actual control sequences at this point.)

```
    #3%
    \begingroup
      \let\protect\string
      \xdef\@tempa{#5}%
    \endgroup
```

Now scan over the text token by token, transforming it into an intermediate representation of fully wrapped text. Then fully expand this intermediate for into its final form, ready to be processed by the input function `#1`.

```
    \expandafter\hw@scan\@tempa\hw@scanstop
    \def\\{\hw@protected@newline}%
    \def\hw@protected@newline{#4}%
    \let\space\hw@expanding@space
    \@temptokena={#1}%
    \expandafter\the\expandafter\@temptokena\expandafter{\hw@wrappedtext}%
  \endgroup
}
```

\hw@scan

Convenience wrapper for `\futurelet`.

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

\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: (1) the stop token `\hw@scanstop`, (2) a space token, (3) a newline insertion, or (4) anything else.

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

2. 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.

`\hw@process` (*cont.*)

3. If we find an explicit 'newline', we process it much as if it were a space and the current word is the last one that can fit on the line. To continue, skip the actual token that is the 'newline' and then start scanning again.

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

```
\def\hw@process{%
  \ifx\let@token\hw@scanstop\relax
    \hw@process@end
    \let\next\@gobble
  \else\ifx\let@token\@sptoken
    \hw@process@space
    \let\next\hw@dochar
  \else\ifx\let@token\hw@insert@newline
    \hw@process@messagebreak
    \def\next{\expandafter\hw@dochar\@gobble}%
  \else
    \let\next\hw@dochar
  \fi\fi\fi
  \next
}
```

`\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.

```
\def\hw@dochar#1{%
  \protected\def\@tempa{#1}%
  \ifx\@tempa\hw@scanstop
    \let\next\hw@process@end
  \else\ifx\@tempa\hw@insert@newline
    \hw@process@messagebreak
    \let\next\hw@scan
  \else
    \edef\hw@currword{\hw@currword #1}%
    \let\next\hw@scan
```

\hw@dochar (*cont.*)

```
  \fi\fi
  \next
}
```

\hw@process@space

```
\def\hw@process@space{%
  \hw@wordcount=\hw@strlen@of\hw@currword\relax
  \ifnum\numexpr(\hw@charcount+\hw@wordcount)\relax<\hw@maxprintline
    \advance\hw@charcount by \hw@wordcount
    \ifx\hw@currline\@empty
      \edef\hw@currline{\hw@currword}%
    \else
      \advance\hw@charcount by 1\relax % account for the space character
      \edef\hw@currline{\hw@currline\hw@expanding@space\hw@currword}%
    \fi
  \else
    \hw@charcount=\hw@wordcount\relax
    \edef\hw@currtext{\hw@currtext\hw@currline\hw@protected@newline}%
    \let\hw@currline\hw@currword
  \fi
  \let\hw@currword\@empty
}
```

\hw@process@messagebreak

```
\def\hw@process@messagebreak{%
  \hw@wordcount=\hw@strlen@of\hw@currword\relax
  \ifnum\numexpr(\hw@charcount+\hw@wordcount)<\hw@maxprintline
    \edef\hw@currtext{%
      \hw@currtext
      \ifx\hw@currline\@empty\else
        \hw@currline\space
      \fi
      \hw@currword\hw@protected@newline
    }%
    \hw@charcount=0\relax
    \let\hw@currline\@empty
  \else
    \edef\hw@currtext{\hw@currtext\hw@currline\hw@protected@newline}%
    \hw@charcount=\hw@wordcount
    \let\hw@currline\hw@currword
```

`\hw@process@messagebreak` (*cont.*)

```
  \fi
  \let\hw@currword\@empty
}
```

`\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.

```
\def\hw@process@end{%
  \ifnum\numexpr(\hw@charcount+\hw@wordcount)<\hw@maxprintline
    \edef\hw@wrappedtext{%
      \hw@currtext
      \ifx\hw@currline\@empty\else
        \hw@currline\space
      \fi
      \hw@currword
    }%
  \else
    \edef\hw@wrappedtext{%
      \hw@currtext\hw@currline\hw@protected@newline\hw@currword
    }%
  \fi
}
```

`\HardWrapSetup`

This is the command to use if you want to 'special-case' some meanings to be more appropriate inside message text. When using `\GeneratePackageLogMacros`, it is used by default for argument `#3` in `\HardWrap`.

```
\def\HardWrapSetup{%
  \def\MessageBreak{\\}%
  \def\newline{\\}%
}
```

§9 Wrapping Log Messages

LaTeX 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 TeX 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 LaTeX warnings and info messages to push the automatic 'on input line ⟨*num*⟩' onto the next line. This makes writing grammatically correct messages somewhat easier.

```
\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.

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

\GeneratePackageLogMacros, etc. (*cont.*)

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

\hw@generate@logging@macros

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

1. {⟨*'package' or 'class'*⟩}
2. {⟨*prefix*⟩}
3. {⟨*package name*⟩}
4. {⟨*info message length*⟩}
5. {⟨*warning message length*⟩}

The ⟨*info...*⟩ and ⟨*warning message length*⟩ values correspond to the calculation of B as described above.

First of all, if the ⟨*prefix*⟩ is not specified then fall back to the ⟨*package name*⟩:

```
\newcommand{\hw@generate@logging@macros}[5]{%
  \def\@tempa{#2}\ifx\@tempa\@empty
    \hw@generate@logging@macros@aux{#1}{#3}{#3}{#4}{#5}%
  \else
    \hw@generate@logging@macros@aux{#1}{#2}{#3}{#4}{#5}%
  \fi
}
```

Finally, the main procedure. Info messages first:

```
\newcommand{\hw@generate@logging@macros@aux}[5]{%
  \expandafter\edef\csname #2@info\endcsname##1{%
    \noexpand\HardWrap
      {\@nameuse{hw@#1@info}{#3}}
      {\number\numexpr#4\relax}
      {\unexpanded\expandafter{\HardWrapSetup}}
      {\noexpand\MessageBreak}
      {##1}%
  }%
  \expandafter\edef\csname #2@info@noline\endcsname##1{%
    \noexpand\HardWrap
      {\@nameuse{hw@#1@info@noline}{#3}}
      {\number\numexpr#4\relax}
      {\unexpanded\expandafter{\HardWrapSetup}}
      {\noexpand\MessageBreak}
      {##1}%
  }%
```

Now warnings:

```
  \expandafter\edef\csname #2@warning\endcsname##1{%
    \noexpand\HardWrap
      {\@nameuse{hw@#1@warning}{#3}}
      {\number\numexpr#5\relax}
      {\unexpanded\expandafter{\HardWrapSetup}}
      {\noexpand\MessageBreak}
      {##1}%
  }%
  \expandafter\edef\csname #2@warning@noline\endcsname##1{%
    \noexpand\HardWrap
      {\@nameuse{hw@#1@warning@noline}{#3}}
      {\number\numexpr#5\relax}
      {\unexpanded\expandafter{\HardWrapSetup}}
      {\noexpand\MessageBreak}
      {##1}%
  }%
```

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*.

```
  \expandafter\edef\csname #2@error\endcsname##1##2{%
    \noexpand\HardWrap
      {\xdef\noexpand\hw@tempa}
      {\number\numexpr#5\relax}
      {\unexpanded\expandafter{\HardWrapSetup}}
      {\MessageBreak}
      {\MessageBreak ##1}%
    \noexpand\HardWrap
      {\xdef\noexpand\hw@tempb}
      {\the\hw@maxprintline}
      {\unexpanded\expandafter{\HardWrapSetup}}
      {\MessageBreak}
      {\MessageBreak ##2}%
```

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

```
    \unexpanded{%
      \@nameuse{hw@#1@error}{#3}{\hw@tempa}{\hw@tempb}%
    }%
  }%
}
```

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.

```
\protected\def\hw@class@info          #1#2{\ClassInfo   {#1}{#2\hw@suffix}}
\protected\def\hw@class@info@noline   #1#2{\ClassInfo   {#1}{#2\@gobbletwo}}
\protected\def\hw@class@warning       #1#2{\ClassWarning{#1}{#2\hw@suffix}}
\protected\def\hw@class@warning@noline#1#2{\ClassWarning{#1}{#2\@gobbletwo}}
\protected\def\hw@class@error         #1#2{\ClassError  {#1}{#2\@gobble}}

\protected\def\hw@package@info          #1#2{\PackageInfo   {#1}{#2\hw@suffix}}
\protected\def\hw@package@info@noline   #1#2{\PackageInfo   {#1}{#2\@gobbletwo}}
\protected\def\hw@package@warning       #1#2{\PackageWarning{#1}{#2\hw@suffix}}
\protected\def\hw@package@warning@noline#1#2{\PackageWarning{#1}{#2\@gobbletwo}}
\protected\def\hw@package@error         #1#2{\PackageError  {#1}{#2\@gobble}}

⟨/package⟩
```

Part III TEST SUITE

```
⟨*testsuite⟩

\begin{qstest}{basics}{}
\HardWrap{\xdef\TMP}{50}{}{NEWLINE}{aaa bbb ccc}
\Expect*{\TMP}{aaa bbb ccc}
\end{qstest}

\begin{qstest}{newline}{}
\HardWrap{\xdef\TMP}{50}{}{NEWLINE}{aaa \\ bbb ccc}
\Expect*{\TMP}{aaa NEWLINE bbb ccc}
\end{qstest}

\begin{qstest}{noexpand/protect/string}{}
\HardWrap{\xdef\TMP}{100}{}{NEWLINE}{%
  \string\section\ and \protect\subsection\
  and \noexpand\paragraph\ and the rest}
```

```
\Expect*{\TMP}
  *{\string\section\space and \string\subsection\space
    and \string\paragraph\space and the rest}
\end{qstest}
```

```
\begin{qstest}{ensure no wayward spaces}{}
\newbox\myboxa
\newbox\myboxb
\setbox\myboxa=\hbox{xx}
\setbox\myboxb=\hbox{x%
    \HardWrap{\xdef\TMP}{50}{}{NEWLINE}{\LIPSUM}%
  x}
\Expect*{\the\wd\myboxa}*{\the\wd\myboxb}
\end{qstest}
```

⟨/testsuite⟩