

monofill.sty

Alignment with Plain Text or Monospaced Characters*

Uwe Lück[†]

March 29, 2012

Abstract

`monofill.sty` addresses horizontal alignment with plain text as in the result of L^AT_EX's `\listfiles`, extending the `longnamefilelist` package. It may also be useful for alignment in typesetting monospaced characters as in figure tables, for simulating a typewriter, or for code listings.—The implementation also has “philosophical aspects,” avoiding use of a counter register.

Contents

1	Features and Usage	2
1.1	Summary of Features	2
1.2	“Philosophical aspects”	2
1.3	Installing and Calling	2
1.4	Examples	3
1.4.1	Typewriter	3
1.4.2	Figures	3
1.4.3	Screen Output	4
2	Package File Header (Legalize)	4
3	User Commands	4

*This document describes version [v0.1a](#) of `monofill.sty` as of 2012/03/29.

[†]<http://contact-ednotes.sty.de.vu>

1	FEATURES AND USAGE	2
4	Internal Commands	5
4.1	Tools	5
4.2	Field Declaration	6
4.3	Checking Field	6
4.4	Trying Alignment	7
5	Package Option	8
6	\endinput and Version HISTORY	8
7	Credit	8

1 Features and Usage

1.1 Summary of Features

A command `\MFfieldtemplate` sets the maximum width of a “field” using a template, with an optional argument for the “filler” token. Then `\MFleftinfield` and `\MFrightinfield` types given (one-line) text and adds “filler” tokens to the left or right, until the entire number of tokens es the number of characters in the associated template. So this is a kind of analogue to `\settowidth{\mylength}{\langle template \rangle}`, `\makebox[\mylength][l]{\langle text \rangle}`, and `\makebox[\mylength][r]{\langle text \rangle}` intended for plain text output, without typesetting. See Sec. 3 for details.

1.2 “Philosophical aspects”

The package also has “philosophical” aspects: 1. Apart from the declaration of the width of a “field”, everything is **expandable** (thinking of application with `blog.sty` of the `morehype` bundle) and thus is a kind of functional programming. 2. Actually, **no counter** is used, and we seem to *count without* using the *concept of “number.”* Rather, we (a) just generate a new list from a given one such that both have the same length and (b) compare the lengths of two lists—both (a) and (b) without *determining* the length (which would be a *number*) of any list.

1.3 Installing and Calling

The file `monofill.sty` is provided ready, installation only requires putting it somewhere where `TEX` finds it (which may need updating the filename data base).¹

Below the `\documentclass` line(s) and above `\begin{document}`, you load `monofill.sty` (as usually) by

```
\usepackage{monofill}
```

¹<http://www.tex.ac.uk/cgi-bin/texfaq2html?label=inst-wlcf>

For certain uses such as with `fileinfo`, the package is better loaded by

```
\RequirePackage{monofill}
```

1.4 Examples

1.4.1 Typewriter

With both

```
\MFfieldtemplate[\MFspace]{tt}{leftright}
```

and

```
\MFfieldtemplate[\MFenspace]{tt}{leftright}
```

followed by

```
\begin{quotation}\tt\noindent
!leftright!\!
!\MFleftinfield{left}{tt}!\!
!\MFrightinfield{right}{tt}!\!
!\MFrightinfield{rightleft}{tt}!
\end{quotation}
```

I get

```
!leftright!
!left      !
!   right!
!rightleft!
```

1.4.2 Figures

Similarly, with `\MFfieldtemplate[\MFenspace]{figs}{0000}` and

```
\begin{quote}\noindent
\MFrightinfield{1}{figs} is one,\!
\MFrightinfield{10}{figs} is ten,\!
\MFrightinfield{100}{figs} is hundred,\!
\MFrightinfield{1000}{figs} is thousand.
\end{quote}
```

I get

```
1 is one,
10 is ten,
100 is hundred,
1000 is thousand.
```

1.4.3 Screen Output

Finally, try

```
\MFfieldtemplate{screen}{0000}
  \typeout{\MFrightinfield{1}{screen} is one,}
  \typeout{\MFrightinfield{10}{screen} is ten,}
  \typeout{\MFrightinfield{100}{screen} is hundred,}
  \typeout{\MFrightinfield{1000}{screen} is thousand.}
\typein{OK?}
```

It works, believe me.

2 Package File Header (Legalize)

```
1 \NeedsTeXFormat{LaTeX2e}[1994/12/01]
2 \ProvidesPackage{monofill}[2012/03/29 v0.1a monospace alignment (UL)]
3
4 %% Copyright (C) 2012 Uwe Lueck,
5 %% http://www.contact-ednotes.sty.de.vu
6 %% -- author-maintained in the sense of LPPL below --
7 %%
8 %% This file can be redistributed and/or modified under
9 %% the terms of the LaTeX Project Public License; either
10 %% version 1.3c of the License, or any later version.
11 %% The latest version of this license is in
12 %% http://www.latex-project.org/lppl.txt
13 %% We did our best to help you, but there is NO WARRANTY.
14 %%
15 %% Please report bugs, problems, and suggestions via
16 %%
17 %% http://www.contact-ednotes.sty.de.vu
```

3 User Commands

`\MFfieldtemplate[<fill-element>]{<field>}{<template>}`

determines the width of fields with id *<field>* to be the same as of *<template>*:

```
18 \newcommand*{\MFfieldtemplate}[3][\MFfillelement]{%
```

`\@bg` delimits the “background” or “filler list”. The field id is stored at the end ahead.

```
19 \MF@make@bg#1#3\MF@store@field@bg\@bg{#2}}
```

`\MF@make@bg` is defined in Sec. 4.2.

`\MFfillelement` is the default for *<fill-element>*, defined to be (like) `\space` here:

```
20 \newcommand*{\MFfillelement}{\let\MFfillelement\space
```

$\langle fill\text{-}element \rangle$ must be a “single item” (that \TeX converts into a single token, due to our comparison mechanism), so for using somewhat more complex $\langle complex \rangle$ than $\backslash\text{space}$,

```
\renewcommand*{\MFfillelement}{\langle complex \rangle}
```

must be used instead of the optional argument.—It was very hard for me with *typesetting*, what finally worked were $\backslash\text{MFspace}$ and $\backslash\text{MFenspace}$ as alternative optional arguments. It is fine for half-quad spaces such as characters with \tt figures with more Computer Modern fonts:

```
21 \newcommand*{\MFspace}{\mbox{ }}
22 % \newcommand*{\MFenspace}{\leavevmode\enspace}
23 \newcommand*{\MFenspace}{\mbox{\enspace}}
```

```
\MFleftinfield{\langle text \rangle}{\langle field \rangle}
```

returns $\langle text \rangle$, followed by $\langle fill\text{-}elements \rangle$ to get as many elements (characters) as the $\langle template \rangle$ associated with $\langle field \rangle$:

```
24 \newcommand*{\MFleftinfield}{\MF@check@field l}
```

$\backslash\text{MFrightinfield}\{\langle text \rangle\}\{\langle field \rangle\}$ returns the $\langle fill\text{-}elements \rangle$ before giving $\langle text \rangle$:

```
25 \newcommand*{\MFrightinfield}{\MF@check@field r}
```

$\backslash\text{MF@check@field}$ is defined in Sec. 4.3.

4 Internal Commands

4.1 Tools

We test arguments $\langle arg \rangle$ on emptiness by $\backslash\text{MF@if@empty}\{\langle arg \rangle\}\{\langle yes \rangle\}\{\langle no \rangle\}$:

```
26 \newcommand*{\MF@if@empty}[1]{%
27   \ifx\MF@store@field@bg#1\MF@store@field@bg
28     \expandafter\@firstoftwo
29   \else
30     \expandafter\@secondoftwo
31   \fi}
```

$\backslash\text{MF@field}$ stores the name space for filling jobs:

```
32 \newcommand*{\MF@field}{\MF@field:}
```

4.2 Field Declaration

`\MF@make@bg` essentially builds a list of as many filler elements as the template has characters, using a loop macro `\MF@make@bg`. The current list of filler elements is delimited by `\@bg`.

```
33 \def\MF@make@bg#1#2#3\MF@store@field@bg{%
34     \MF@if@empty{#3}%
```

First case: #2 is the last template element. We run `\MF@store@field@bg` with an additional filler element:²

```
35         {\MF@store@field@bg#1}%
```

Second case: the filler list gets an additional element, and the loop repeats:

```
36         {\MF@make@bg#1#3\MF@store@field@bg#1}%
37     }
```

`\MF@store@field@bg<background>\@bg{<field>}` essentially stores the filler list (“`<background>`”), or more precisely ...

```
38 \def\MF@store@field@bg#1\@bg#2{%
```

Here is the **only assignment** when the macros run: a command

```
\MF@field:<field>{\<text>}
```

is defined.³

```
39 \namedef{\MF@field#2}##1{%
40     \MF@reduce@bg##1\rest@t#1\rest@f{##1}{#2}}}
```

4.3 Checking Field

`\MF@check@field{<align>}{<text>}{<field>}` runs `\MF@field:<field>{\<text>}` from above, provided the latter has been defined (by `\MFfieldtemplate`). The `<align>` command is appended.

```
41 \newcommand*{\MF@check@field}[3]{%
42     \@ifundefined{\MF@field#3}%
43     {\PackageError{field "#3" not defined}%
44     {use \string\MFfieldtemplate}}%
```

With v0.1, I thought about errors and warnings properly only more below ...

```
45         {\MF@field@undeclared{#2}{#3}}%
46         {\csname\MF@field#3\endcsname{#2}#1}}
```

²Another run of `\MF@make@bg` fails ...

³This is the common, confusing way to describe such situations. Actually, the definition assigns a macro meaning to a “named token” whose name is “`\MF@field:<field>`”. *Typing* `\MF@field:<field>` won’t work.

`\MF@field@undeclared{<text>}{<field>}` just outputs `<text>`.

```
47 \newcommand*{\MF@field@undeclared}[2]{#1}
```

A proper message is problematic in **pure expansion** as on screen or in `.log` files. Package option `\fake-undefined` (Sec. 5) offers another “cheap” solution. (TODO)

4.4 Trying Alignment

`\MF@reduce@bg<r-text>\rest@t<r-fill>\rest@f{<text>}{<field>}{<align>}`

is invoked by that `\MF@field:<field>` that `\MF@store@field@bg` defines as above. It takes away one element both from the (remaining) `<text>` (delimited by `\rest@t`) and the filler list (delimited by `\rest@f`). The full `<text>` has been stored ahead.

```
48 \def\MF@reduce@bg#1#2\rest@t#3#4\rest@f{%
49     \MF@if@empty{#2}%
50     {\MF@if@empty{#4}%
```

When we have removed the last elements of both lists at the same time, we just return `<text>`:

```
51         \@firstofthree
```

When we have removed the last element of `<text>`, and there still is a filler element, we perform the alignment:

```
52         {\MF@fine@align{#4}}}%
53     {\MF@if@empty{#4}%
```

When we have removed the last filler element, and a `<text>` element is still present, we return `<text>`, maybe together with a warning:

```
54         \MF@bad@align
```

When neither `#1` nor `#3` have been the last elements in their lists, we run `\MF@reduce@bg` on the remaining lists:

```
55         {\MF@reduce@bg#2\rest@t#4\rest@f}}}
```

`\@firstofthree{<use>}{<skip>}{<skip>}` may be known or not ...

```
56 \long\def\@firstofthree#1#2#3{#1}
```

`\MF@fine@align{<filler>}{<text>}{<field>}{<align>}` ...

```
57 \newcommand*{\MF@fine@align}[4]{\if r#4#1#2\else#2#1\fi}
```

`\MF@bad@align{<text>}{<field>}{<align>}`

at present is similar to `\@firstofthree`. In a future package version, we may add some warning or so for cases where it is useful—while it is not useful to write *code* for warnings to screen and `.log` (the originally intended use of the package). We offer a “cheap” possibility of throwing some error by package option `\fake-undefined`—see Sec. 5

```
58 \newcommand*{\MF@bad@align}[3]{#1}
```

Actually, in v0.1 `\MF@check@field` appends the $\langle field \rangle$ argument hoping it could be used in a warning.

5 Package Option

With applications like `\listfiles`, it may be useful to get an “undefined” error where the name of the undefined command is a kind of “secret message” ...

```
59 \DeclareOption{fake-undefined}{%
```

#1 is $\langle text \rangle$ and will be output, #2 is $\langle field \rangle$, cf. above.

```
60 \def\MF@field@undeclared#1#2{#1\monofillFieldUndeclared}
61 \def\MF@bad@align#1#2#3{#1\monofillFieldTooSmall}}
62 \ProcessOptions
```

6 \endinput and Version HISTORY

```
63 \endinput
```

VERSION HISTORY

```
64 v0.1    2012/03/18   started
65         2012/03/19   completed
66 v0.1a   2012/03/29   doc.: \medbreak (fix); \strong
67
```

7 Credit

The package actually is motivated by good ideas of Martin Muench’s about extending the `longnamefilelist` package.