The docmfp package*

Peter Wilson†
Herries Press

2005/03/26

Abstract

The docmfp package extends the doc package to cater for documentation of non-LaTeX code, such as Metafont and Metapost code, or C or Java code.

Contents

1 Introduction

It is common practice to document LaTeX packages using the doc system [GMS94]. The docmfp package extends the doc package so that similar facilities are provided for the documentation of non-LaTeX code, such as Metafont and Metapost code, or code in other more common programming languages. For example, a single `.dtx` file can contain the documented sources of both the Metafont code for a new font, together with the documented LaTeX code for the accompanying package.

This manual is typeset according to the conventions of the LaTeX docstrip utility which enables the automatic extraction of the LaTeX macro source files [GMS94].

Section 2 describes the usage of the package. Commented source code for the package is in Section 3.

*This file (`docmfp.dtx`) has version number v1.2c, last revised 2005/03/26.

†`herries dot press at earthlink dot net`

2 The package

I have assumed that if you are reading this then you are familiar with the facilities provided by the doc package.

\DescribeRoutine The \DescribeRoutine{⟨*name*⟩} command is equivalent to the doc package \DescribeEnv{⟨*name*⟩} command, except that it is intended to introduce the description of a Metafont/post macro (or character or picture). It typesets ⟨*name*⟩ in the margin and also generates an index entry for ⟨*name*⟩.

routine The routine environment is equivalent to the doc package macro environment. It takes one argument, which is the name of the Metafont/post macro (or character or picture) that is being defined. It typesets the argument in the margin and makes an index entry for it.

\routinestring \routineheadname These two commands contain the texts that are used in indexing routine names. They can be changed via \renewcommand. Their default definitions are:

```
\newcommand{\routinestring}{\space(routine)}
\newcommand{\routineheadname}{routines:}
```

\DescribeVariable The \DescribeVariable{⟨*name*⟩} is like the doc \DescribeMacro{⟨*name*⟩} command, except that it is intended to introduce the description, and definition, of a variable or parameter. It typesets ⟨*name*⟩ in the margin and makes an index entry for it.

variable The variable environment is equivalent to the doc package macro environment. It takes one argument, which is the name of the variable or parameter that is being defined. It typesets the argument in the margin and makes an index entry for it.

\variablestring \variableheadname These two commands contain the texts that are used in indexing variable names. They can be changed via \renewcommand. Their default definitions are:

```
\newcommand{\variablestring}{\space(variable)}
\newcommand{\variableheadname}{variables:}
```

A routine or variable ⟨*name*⟩ can include the underscore and hash characters (i.e., _ and #), so that names like a_variable# can be used.

\Describe This is a generalization of the \Describe... commands.

\Describe{⟨*head*⟩}{⟨*flag*⟩}{⟨*name*⟩} typesets ⟨*name*⟩ in the margin and makes index entries for it. One entry will be a main entry as name flag, and the other will be a subsidiary entry of name under the main heading head.

Essentially, \DescribeVariable{name} is equivalent to:
\Describe{\variableheadname}{\variablestring}{name}

Code This is a generalization of the variable and routine environments.

\begin{Code}{⟨*head*⟩}{⟨*flag*⟩}{⟨*name*⟩} typesets ⟨*name*⟩ in the margin and makes index entries for it. One entry will be a main entry as name flag, and the other will be a subsidiary entry of name under the main heading head.

Essentially, \begin{routine}{name} is equivalent to:
\begin{Code}{\routineheadname}{\routinestring}{name}

For example, if you are documenting Java code, then you may wish to use commands like:
`\newcommand{\cvar}{class variables}`
`\newcommand{\fcvar}{ (variable)}`
`\newcommand{\ofield}{object fields}`
`\newcommand{\ffield}{ (field)}`
`\newcommand{\cmeth}{class methods}`
`\newcommand{\ometh}{object methods}`
`\newcommand{\meth}{ (method)}`
`\Describe{\cvar}{\fcvar}{...}` for class variables
`\Describe{\ofield}{\ffield}{...}` for object fields
`\begin{Code}{\cmeth}{\meth}{...}` for class methods
`\begin{Code}{\ometh}{\meth}{...}` for object methods.

2.1 Usage with .dtx and .ins files

I assume that the major use of the docmfp package will be in `.dtx` file(s) that will be processed via a corresponding `.ins` file. The ltxdoc class also automatically calls the doc package.

As an example, if the docmfp package was needed for the document you are now reading, then I would have started it off like this (but look at the start of the source for details that I ignore here):

```
%<*driver>
\documentclass{ltxdoc}
\usepackage{docmfp}
\EnableCrossrefs
\CodelineIndex
\setcounter{StandardModuleDepth}{1}
\begin{document}
  \DocInput{docmfp.dtx}
\end{document}
%</driver>
```

The source of this document also includes several calls of the docmfp commands, which I have commented out (use your editor to look for the occurences of the string `^^A` after this point). You can edit the source to include the `\usepackage{docmfp}` command and uncomment the docmfp commands if you want to sample the package in use.

The source of an `.ins` file might look like this:

```
%% file myfile.ins
\def\batchfile{myfile.ins}
\input docstrip.tex
\preamble
  Copyright and other notices
```

```
\endpreamble
\generateFile{myfile.drv}{t}{\from{myfile.dtx}{driver}}
\generateFile{mypackage.sty}{t}{\from{myfile.dtx}{pack}}

\endinput
```

By default, the documentation system will put an `\endinput` command at the end of each file it generates (`myfile.drv` and `mypackage.sty` in the example above). This is fine provided the generated files are to be processed by LaTeX which understands `\endinput`. If a generated file is to be processed by something that treats `\endinput` as an error, as Metafont/post will, then there is a problem.

`\usepostamble` `\empty` `\defaultpostamble`

The documentation system provides these three commands which can be used within an `.ins` file to either prevent or enable the addition of `\endinput` to the generated files. Extending the above example `.ins` file to include both Metafont and and LaTeX files we can have:

```
%% file myfile.ins
\def\batchfile{myfile.ins}
\input docstrip.tex
\preamble
  Copyright and other notices
\endpreamble
\generateFile{myfile.drv}{t}{\from{myfile.dtx}{driver}}
\usepostamble\empty                 % switch off writing \endinput
\generateFile{myfont.mf}{t}{\from{myfile.dtx}{font}}
\usepostamble\defaultpostamble      % switch on writing \endinput
\generateFile{mypackage.sty}{t}{\from{myfile.dtx}{pack}}

\endinput
```

`\usepreamble` `\defaultpreamble`

The documentation system provides these commands which can be used within an `.ins` file to either prevent or enable the addition of preamble information at the beginning of the generated files. The preamble information is in the form of LaTeX comment lines (i.e., lines starting with %). Other languages that you may wish to document probably have other different commenting conventions, in which cases it is desireable to inhibit the preamble output. Extending the above example `.ins` file to turn off the preamble for the driver file we can have:

```
%% file myfile.ins
\def\batchfile{myfile.ins}
\input docstrip.tex
\usepreamble\empty            % switch off all preamble info
\generateFile{myfile.drv}{t}{\from{myfile.dtx}{driver}}
\usepreamble\defaultpreamble  % switch on normal preambling
\preamble
  Copyright and other notices
\endpreamble
\usepostamble\empty                 % switch off writing \endinput
```

```
\generateFile{myfont.mf}{t}{\from{myfile.dtx}{font}}
\usepostamble\defaultpostamble      % switch on writing \endinput
\generateFile{mypackage.sty}{t}{\from{myfile.dtx}{pack}}

\endinput
```

There is no intrinsic reason why the use of this package should be limited to documenting Metafont/post code. It could just as well be used for documenting C, C++, Java, or practically any other kind of code.

3 The package code

Announce the name and version of the package, which requires LaTeX 2$_\varepsilon$.

```
⟨*usc⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{docmfp}[2004/05/14 v1.2b General coding extension to the doc package]

```

In order to try and avoid name clashes with other packages, each internal name will include the character string `m@fp`.

`\m@fpmakeuscoreletter` `\m@fpmakehashletter` `\Makem@fpPrivateLetters`

Metafont/post names can include underscores and hash characters. The special meanings of these have to be turned off.

```
\newcommand{\m@fpmakeuscoreletter}{\catcode`\_11\relax}
\newcommand{\m@fpmakehashletter}{\catcode`\#11\relax}
\newcommand{\Makem@fpPrivateLetters}{\m@fpmakeuscoreletter\m@fpmakehashletter}

```

`routine`

The `routine` environment code is similar to the doc package's `environment` code.

```
\def\routine{\begingroup
  \catcode`\\12
  \Makem@fpPrivateLetters \m@fpm@cro@ \iffalse}
\let\endroutine\endtrivlist

```

`variable`

The `variable` environment code is almost identical to the code for the `routine` environment.

```
\def\variable{\begingroup
  \catcode`\\12
  \Makem@fpPrivateLetters \m@fpm@cro@ \iftrue}
\let\endvariable\endroutine

```

`\m@fpm@cro@`

This command does all the work for both the `routine` and `variable` environments. The first part is a straight copy of the doc package `\m@cro@` command.

```
\long\def\m@fpm@cro@#1#2{\endgroup \topsep\MacroTopsep \trivlist
  \def\makelabel##1{\llap{##1}}%
```

```
\if@inlabel
  \let\@tempa\@empty \count@\macro@cnt
  \loop \ifnum\count@>\z@
    \edef\@tempa{\@tempa\hbox{\strut}}\advance\count@\m@ne \repeat
  \edef\makelabel##1{\llap{\vtop to\baselineskip
                            {\@tempa\hbox{##1}\vss}}}%
  \advance \macro@cnt \@ne
\else
  \macro@cnt\@ne
\fi
```

The rest of the code is for this package, and is a simplified and modified version of the corresponding code for `\m@cro@`.

```
\edef\@tempa{\noexpand\item[\noexpand\PrintMfpName{\string#2}]}%
\@tempa
\global\advance\c@CodelineNo\@ne
#1%
```

Do the indexing for the `variable` environment.

```
  \SpecialMainMfpIndex{#2}{\variablestring}{\variableheadname}\nobreak
\else
```

Do the indexing for the `routine` environment.

```
  \SpecialMainMfpIndex{#2}{\routinestring}{\routineheadname}\nobreak
\fi
```

and finish off the definition.

```
\global\advance\c@CodelineNo\m@ne
\ignorespaces}
```

`\routinestring` `\routineheadname` These two commands store the default indexing strings for routines.

```
\newcommand{\routinestring}{\space(routine)}
\newcommand{\routineheadname}{routines:}
```

`\variablestring` `\variableheadname` These two commands store the default indexing strings for variables.

```
\newcommand{\variablestring}{\space(variable)}
\newcommand{\variableheadname}{variables:}
```

`\Describe` This is a generic description macro, and is similar to those defined later.

```
\def\Describe{\leavevmode\@bsphack\begingroup\Makem@fpPrivateLetters
  \Describem@fp}
```

`\Describem@fp` The workhorse for `\Describe`. It processes the three apparent arguments to `\Describe`.

```
\def\Describem@fp#1#2#3{\endgroup
  \marginpar{\raggedleft\PrintMfpName{#3}}%
  \SpecialMfpIndex{#3}{#2}{#1}\@esphack\ignorespaces}
```

`Code` This is a generalization of the previous environments.

```
\def\Code{\begingroup
 %%  \catcode`\\12
  \Makem@fpPrivateLetters \m@fpm@c}
\let\endCode\endtrivlist

```

`\m@fpm@c` This is the workhorse for the `Code` environment and processes the 3 arguments apparently taken by the environment.

```
\long\def\m@fpm@c#1#2#3{\endgroup \topsep\MacroTopsep \trivlist
  \def\makelabel##1{\llap{##1}}%
  \if@inlabel
    \let\@tempa\@empty \count@\macro@cnt
    \loop \ifnum\count@>\z@
      \edef\@tempa{\@tempa\hbox{\strut}}%
      \advance\count@\m@ne \repeat
    \edef\makelabel##1{\llap{\vtop to\baselineskip
                               {\@tempa\hbox{##1}\vss}}}%
    \advance\macro@cnt\@ne
  \else
    \macro@cnt\@ne
  \fi
  \edef\@tempa{\noexpand\item[\noexpand\PrintMfpName{\string#3}]}%
  \@tempa
  \global\advance\c@CodelineNo\@ne
  \SpecialMainMfpIndex{#3}{#2}{#1}\nobreak
  \global\advance\c@CodelineNo\m@ne
  \ignorespaces}

```

`\PrintMfpName` This typesets the name of a Metafont/post routine or variable, or in general ⟨*name*⟩. If there is a pre-existing definition, then the package does not modify it.

```
\providecommand{\PrintMfpName}[1]{\strut \MacroFont \string #1\ }

```

Now for the command that writes out the index entries for the `routine` and `variable` environments. It is also used for the `Code` environment.

`\SpecialMainMfpIndex` The command `\SpecialMainMfpIndex{`⟨*name*⟩`}{`⟨*string*⟩`}{`⟨*heading*⟩`}` writes ⟨*name*⟩ to the `.idx` file, firstly as a 'main' entry (flagged with ⟨*string*⟩) and then as a subsidiary entry under ⟨*heading*⟩. Both entries are treated as definitional.

```
\newcommand{\SpecialMainMfpIndex}[3]{\@bsphack
```

Here is the main index entry.

```
  \special@index{%
    \string#1\actualchar
    \string\verb\quotechar*\verbatimchar\string#1\verbatimchar
    #2 \encapchar main}%
```

Here is the subsidiary index entry.

```
  \special@index{#3\levelchar
    \string#1\actualchar
    \string\verb\quotechar*\verbatimchar\string#1\verbatimchar
    \encapchar main}
  \@esphack}

```

`\DescribeRoutine` The command `\DescribeRoutine{`⟨*name*⟩`}` typesets a marginal heading and an index entry for the description of a routine called ⟨*name*⟩. It is based on the doc `\DescribeMacro` command.

```
\def\DescribeRoutine{\leavevmode\@bsphack\begingroup\Makem@fpPrivateLetters
  \Describem@fpRoutine}
```

`\Describem@fpRoutine` This is the macro that does the work for `\DescribeRoutine`.

```
\def\Describem@fpRoutine#1{\endgroup
  \marginpar{\raggedleft\PrintMfpName{#1}}%
  \SpecialMfpIndex{#1}{\routinestring}{\routineheadname}\@esphack\ignorespaces}

```

`\DescribeVariable` The command `\DescribeVariable{`⟨*name*⟩`}` typesets a marginal heading and an index entry for the description of a variable called ⟨*name*⟩. It is based on the `\DescribeRoutine` command.

```
\def\DescribeVariable{\leavevmode\@bsphack\begingroup\Makem@fpPrivateLetters
  \Describem@fpVariable}
```

`\Describem@fpVariable` This is the macro that does the work for `\DescribeVariable`.

```
\def\Describem@fpVariable#1{\endgroup
  \marginpar{\raggedleft\PrintMfpName{#1}}%
  \SpecialMfpIndex{#1}{\variablestring}{\variableheadname}\@esphack\ignorespaces}

```

`\SpecialMfpIndex` The command `\SpecialMfpIndex{`⟨*name*⟩`}{`⟨*string*⟩`}{`⟨*heading*⟩`}` writes ⟨*name*⟩ to the `.idx` file, firstly as a 'main' entry (flagged with ⟨*string*⟩) and then as a subsidiary entry under ⟨*heading*⟩. Both entries are treated as 'usages' of the ⟨*name*⟩.

```
\newcommand{\SpecialMfpIndex}[3]{\@bsphack
```

Here is the main index entry.

```
  \index{%
    \string#1\actualchar
    \string\verb\quotechar*\verbatimchar\string#1\verbatimchar
    #2 \encapchar usage}%
```

Here is the subsidiary index entry.

```
  \index{#3\levelchar
    \string#1\actualchar
    \string\verb\quotechar*\verbatimchar\string#1\verbatimchar
    \encapchar usage}
```

```
\@esphack}
```

`\check@checksum` After some experience with using docmfp for code that had no resemblence at all to LaTeX, I found that if there were no backslashes present, then doc whined about there being no checksum and said that the `\CheckSum` should be set to zero, which it was. This is the relevent code from the doc package.

```
\def\check@checksum{\relax
   \ifnum\check@sum=\z@
     \typeout{**********************************}%
     \typeout{* This macro file has no checksum!}%
     \typeout{* The checksum should be \the\bslash@cnt!}%
     \typeout{**********************************}%
   \else
     \ifnum\check@sum=\bslash@cnt
       \typeout{********************}%
       \typeout{* Checksum passed *}%
       \typeout{********************}%
     \else
       \PackageError{doc}{Checksum not passed
                  (\the\check@sum<>\the\bslash@cnt)}%
        {The file currently documented seems to be wrong.^^J%
         Try to get a correct version.}%
     \fi
   \fi
   \global\check@sum\z@}
```

For the purposes of the docmfp package this needs redefining as a zero check sum is acceptable.

```
\renewcommand{\check@checksum}{\relax
      \ifnum\check@sum=\bslash@cnt
        \typeout{********************}%
        \typeout{* Checksum passed *}%
        \typeout{********************}%
      \else
        \PackageError{docmfp}{Checksum not passed
                   (\the\check@sum<>\the\bslash@cnt)}%
         {The file currently documented seems to be wrong.^^J%
          Try to get a correct version.}%
      \fi
      \global\check@sum\z@}
```

The end of this package.

⟨/usc⟩

References

[GMS94] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley Publishing Company, 1994.

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.