



# User Reference Manual

Last change: 2011-12-04 10:42

Erich Frühstück

E<sub>rich</sub> F<sub>rühstück</sub> E<sub>ntwicklungs</sub> U<sub>mgebung</sub>

Copyright (C) 2001 Erich Frühstück.  
This documentation is part of EFEU.

EFEU is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

EFEU is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with EFEU; see the file COPYING. If not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.

# Preface

This manual describes the different commands coming with EFEU. The current version lists all commands, but some descriptions may be very brief. The work on this documentation is still in progress.

The single manpages are created by the commands itself with the option `--help`. So the command syntax is always up to date.

Here is a brief overview of the most important tools of EFEU:

`shmkmf`

create Makefile from Config.make.

`esh`

is a command interpreter with a syntax similar to C/C++.

`mksource`

is a C source generator.

`efeudoc`

is a document generator with different output formats (LaTeX, HTML, roff, ...).

Erich Frühstück  
Wördern, December 2011

# Contents

dircmp – compare directories . . . . .	4
dircpy – copy directory . . . . .	5
efeu-magic – update magic file for efile . . . . .	6
efeucc – run C compiler with efeu specific parameters . . . . .	7
efeudoc – document generator . . . . .	8
efeuscript – install script file . . . . .	20
efile – file command with efeu extensions . . . . .	21
esh – EFEU command interpreter . . . . .	22
findgrep – search pattern in files of directories . . . . .	35
htmlindex – create index.html for current directory . . . . .	36
mdcat – concatenate data cubes . . . . .	37
mdcmp – compare two data cubes . . . . .	38
mdcreate – create data cube . . . . .	40
mddiag – diagonalize axis of data cube . . . . .	42
mddiff – show differences of data cubes . . . . .	44
mdfile – show structure of data cube . . . . .	46
mdfunc – data cube function . . . . .	47
mdmul – matrix multiplication . . . . .	49
mdpaste – paste data cubes with regard to axis . . . . .	51
mdpermut – reorder axis of data cube . . . . .	53
mdprint – print data cube . . . . .	55
mdread – read data cube . . . . .	58
mdround – round values of data cube . . . . .	60
mdselect – select part of data cube . . . . .	62
mdsum – accumulate axis of data cube . . . . .	64
mdterm – evaluate data cube expression . . . . .	66
mksource – program generator . . . . .	68
mksrclist – create source list . . . . .	70

pp2dep – create dependence list from cpp output . . . . .	71
shmkmf – create Makefile from shell . . . . .	72
shmkmf-cflags – determine c-flags for use of external libraries . . . . .	74
shmkmf-config – configuration tool for shmkmf . . . . .	75
src2doc – create documentation from source . . . . .	78
tex2ps – process TeX-document . . . . .	80

**NAME**

dircmp – compare directories

**SYNOPSIS**

```
dircmp [ --help[=type] ] [ --version ] [ -l ] [ -b ] [ -d ] [ -c ] [ -v ] dir1 dir2
```

**DESCRIPTION**

The command `dircmp` compares files in *dir1* with files in *dir2*. The processing is recursive. The comparison is not symmetric: it's okay for *dir1* to contain less files than *dir2*.

The following options and arguments are accepted from command `dircmp`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeuDoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

`-l` show all different characters (`cmp -l`)

`-b` Ignore changes in the amount of white space.

`-d` use `diff` instead of `cmp`

`-c` use `diff -c` instead of `cmp`

`-v` list all performed comparisons

*dir1*

first directory

*dir2*

second directory

**EXAMPLES**

The following lines shows the usage of `dircmp`:

```
mkdir save
cp files save
...
dircmp save .
```

**SEE ALSO**

`cmp(1)`, `diff(1)`, `dircpy(1)`.

**COPYRIGHT**

Copyright (C) 2001 Erich Frühstück

**NAME**

dircpy – copy directory

**SYNOPSIS**

```
dircpy [ --help[=type] ] [ --version ] src tg [ find-options ]
```

**DESCRIPTION**

The command `dircpy` uses `find` and `cpio` to copy the directory tree *src* to *tg/src*. The target files gets the same file modification times then the original files. The access time of the source files is kept unchanged. Files are only overwritten, if the coresponding source files have a newer modifikation time.

**SEE ALSO**

`find(1)`, `cpio(1)`, `dircmp(1)`.

**COPYRIGHT**

Copyright (C) 2001 Erich Frühstück

**NAME**

efeu-magic – update magic file for efile

**SYNOPSIS**

```
efeu-magic [ --help[=type] ] [ --version ] dir
```

**DESCRIPTION**

The following options and arguments are accepted from command `efeu-magic`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

*dir* directory with magic file

**SEE ALSO**

efile(1), file(1), magic(5).

**COPYRIGHT**

Copyright (C) 2000 Erich Frühstück

**NAME**

efeucc – run C compiler with efeu specific parameters

**SYNOPSIS**

```
efeucc [ --help[=type] ] [ --version ] [ arg(s) ]
```

**DESCRIPTION**

The command `efeucc` sets some default compiler options for `cc` and expands the include path and `LD_RUN_PATH` according to the EFEU top directory. Just have a look to the command `/efeucc` to see the settings.

**SEE ALSO**

`cc(1)`.

**NOTES**

This command was created by `/home/efeu/www/efeu-3.3-1/src/config/efeu/efeucc.gencmd` on installing EFEU.

**NAME**

efeudoc – document generator

**SYNOPSIS**

```
efeudoc [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ]
[ -I dir ] [ -T type ] [ -C cfg ] [ -H hdr ] [ -O ] [ -u ] [ -r name ] [ -d ] [ -t ] [ -l ] [ -p ]
[ --pdf ] [ -h ] [ -o aus ] file(s)
```

**DESCRIPTION**

The command `efeudoc` is a document generator with different output formats (LaTeX, HTML, roff, ...). The EFEU interpreter language (see *esh(1)*) is used in commands, so you can directly integrate data evaluation or create tables from data cubes directly in your document.

The document processor `efeudoc` uses a plain text input format that allows you to write the document similar to the desired output and tries to use controls as less as possible.

To demonstrate this, the following example shows you the construction of a bullet list:

The code

```
Text before list.
```

```
*      List entry with a paragraph containing
      more than one line of text to show the behavior
      of indentation.
*      Another list entry
```

```
      This is the second paragraph of this entry.
```

```
Text after list.
```

creates the output:

Text before list.

- List entry with a paragraph containing more than one line of text to show the behavior of indentation.
- Another list entry
  - This is the second paragraph of this entry.

Text after list.

**Options**

The following options and arguments are accepted by `efeudoc`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp

send postscript data to `lpr`

`--version`

show version information

```

--info[=entry]
    show command information

--debug[=mode]
    set debug level for command See for details.

--verbose
    set debug level to .info.

-I dir
    append dir to the search path for configuration files. The default setting is "./home/efe/ww/efe-3"

-T type
    document type, ? gives a listing.

-C cfg
    configuration file

-H hdr
    evaluates hdr before any input file.

-O
    suppress document head

-u
    Use UTF8 coding in sources

-r name
    create target rule for make

-d
    create dependence

-t
    terminal output

-l
    LaTeX document

-p
    PostScript document over LaTeX

--pdf
    PDF document over LaTeX

-h
    html document

-o aus
    output file

file(s)
    input file(s)

```

## DOCUMENT SYNTAX

This section needs to be written

## COMMANDS

This sections contains a list of all predefined commands of `efeudoc`.

The following notation is used in command syntax:

- If there are different methods of parameter handling, all of them are given. Every calling variant starts with the command name.
- Any argument in brackets like `[arg]` or braces like `{arg}` is optional and may be omitted. The brackets and braces are part of the syntax.

- Any argument without braces in the same line as the command name is terminated by the next linefeed.
- Indented dots in a line means that the argument of the previous line may be continued with indented lines.
- Any argument in its own line is ended by the next empty line.

**\addflag**

document flags

The command `\addflag` expands the flags for document processing. The interpretation of the flags depends of the actual output driver.

**\addsrc**

expand source list with single file

`\addsrc file`

**\addsrclist**

expand source list with files of directory tree

`\addsrclist dir`

**\appendix**

appendix

`\appendix [short form] title`

The command `\appendix` starts a new appendix.

**\author**

list of authors

**\bfitem**

boldface item

`\bfitem tag`

The command starts a new description entry with a bold face item *tag*.

**\bibitem**

bibliography entry

**\bibliography**

start bibliography

**\br**

line break

The command `\br` causes a line break.

**\caption**

figure or table caption

`\caption [short form] title`

The command sets the title for a figure or table.

**\chap**

chapter

`\chap [toc entry] title`

...

**\cite**

cite bibliography entry

**\cmdinclude**

include script file

```
\cmdinclude "name" \cmdinclude <name>
```

**\code**

code environment

The command **\code** starts the code environment. Inside this environment, the default font is typewriter.

**\config**

load configuration file

```
\config name
```

The command **\config** loads the configuration file *name*.

**\Copyright**

caption COPYRIGHT

**\date**

set date

The command **\date** sets the date for the document.

**\delsrc**

remove files from source list

```
\delsrc pattern
```

**\Description**

caption DESCRIPTION

**\Diagnostics**

caption DIAGNOSTICS

**\docfile**

get path of document file

```
\docfile name  
\docfile {name}
```

The command searches the file *name* relating to the current document path and returns the path name if found.

**\end**

end current environment

The command **\end** closes the current environment.

**\endall**

end all environments

The command **\endall** closes all open environments.

**\endlist**

end list

The command **\endlist** closes the current list environment.

**\Environment**

caption ENVIRONMENT

**\Errors**

caption ERRORS

**\eval**

evaluate expression

```

\eval expr
\eval {expr}

```

The command `\eval` evaluates *expr* and inserts the result at current position.

Example: `\eval{3*5}` results in 15.

**\Example**

caption EXAMPLE

**\Examples**

caption EXAMPLES

**\exercise**

Exercise

```

\exercise

```

**\fig**

figure

```

\fig [pos]

```

The command `\fig` starts a new figure environment.

**\Files**

caption FILES

**\getsrc**

insert files of source list

```

\getsrc [cmd] pattern list

```

**\geval**

evaluate expression global

```

\geval expr
\geval {expr}

```

This command acts like `\eval` but new variables are created global.

**\hang**

hanging paragraphs

The command `\hang` starts a region with hanging paragraphs.

**\head**

headline in manual pages

```

\head [toc entry] title

```

...

**\hmode**

start paragraph, horizontal mode

```

\hmode

```

The command `\hmode` starts a new paragraph, if none is opened.

`\if`  
conditional block

`\include`  
include file

`\include [flags] name`

The command `\include` inserts file *name* at current position. The following control flags could be used:

`verbatim`

verbatim text without interpretation of special characters. Tabulators are replaced by spaces.

`latex`

plain LaTeX text.

`html`

plain HTML text

`man`

plain manpage text

`ignore`

load text but ignore output

`eval`

evaluate file with EFEU command interpreter

`geval`

evaluate file with EFEU command interpreter, new variables are created global.

`\index`  
index entry

`\index [entry]{text}`

The command `\index` creates an index entry *entry* to the text *text*. If *entry* is missing, *text* would be used.

`\intro`  
introduction

The command `\intro` starts the introduction

`\item`  
roman item

`\item tag`

The command starts a new description item with a roman tag.

`\ititem`  
italic item

`\ititem tag`

The command starts a new description entry with a italic item *tag*.

`\label`  
set label

`\label {name}`

`\langpar`  
language dependent paragraph

`\langpar`  
*par*

Experimental, may be changed.

`\latex`  
 L<sup>A</sup>T<sub>E</sub>X code

`\latex [alt] {code} \latex [alt] code`

The command allows to insert latex code *code* for the latex driver. All other drivers use *alt* instead.

`\lmark`  
 bibliographical reference

The command `\lmark` creates a bibliography mark.

`\lof`  
 list of figures

The command `\lof` creates a list of tables.

`\lot`  
 list of tables

The command `\lot` creates a list of figures.

`\margin`  
 marginal note

`\margin [left] {note}`

`\mark`  
 footnote mark

The command `\mark` creates a footnote mark.

`\math`  
 mathematic typesetting

`\math {formular}`

`\mchap`  
 manpage chapter

`\mchap [number] title`

...

`\mkmf`  
 create makefile and insert

`\mkmf imakefile`

The command `\mkmf` calls `mkmf` with the given Imakefile and inserts the created Makefile in verbatim mode.

`\mksource`  
 create sourcefile

`\mksource file`

The command `\mksource` calls `mksource` with the given script and includes the created files in verbatim mode.

`\mpage`  
 manual page entry

`\mpage` [*num*] *name*

The command `\mpage` starts a new manpage with name *name* for section *num*. This command is automatic inserted in every file witch prefix starts with a digit.

`\mref`  
manual page reference

`\mref` {*arg*}

`\ms_example`  
execute `mksource` example

`\ms_example` *name example script* `\end`

The command `\ms_example` calls `mksource` with the given example script and includes it together with all created files in verbatim mode.

`\n` line feed

`\n`

The command `\n` writes a linefeed back to the input stream.

`\Name`  
Name section of manpage with caption NAME

`\Name` [toc entry] *title*

...

`\Name` [toc entry]

*one line title*

`\newpage`  
page break

The command `\newpage` causes a page break.

`\note`  
footnote

`\note` [*mark*] {*note*}

`\Notes`  
caption NOTES

`\package`  
LaTeX packages

The command `\package` loades the LaTeX package *name* if the LaTeX driver is used.

`\par`  
end paragraph, vertical mode

`\par`

The command `\par` finishes the current paragraph.

`\para`  
paragraph

`\part`  
part

`\part [toc entry] title`

...

`\pipe`

include output of command

`\pipe [flags] cmd ...`

The command `\pipe` inserts the output of the command `cmd` at current position. The flags are the same as in `\include`.

`\pop`

remove command table

The command `\pop` removes the lowest table from the table stack.

`\pref`

page ref

`\preface`

preface

The command `\preface` starts the preface

`\printindex`

index

The command `\printindex` creates the index.

`\printsrlist`

print source list

`\printsrlist fmt`

`\proto`

function prototype

`\proto prototype`

The command `\proto` shows the prototype of a function of the EFEU interpreter language.

`\push`

new command table

The command `\push` creates a new table for commands, macros and vars and pushes it to the table stack.

`\quote`

quoting

The command `\quote` starts a region with quoted (indented) text.

`\ref`

refer label

`\relax`

idle command

the command `\relax` does nothing

`\rem`

comment

`\rem comment`

The command `\rem` writes a comment to the output file.

**\sec**

section

**\sec** [*toc entry*] *title*

...

**\SeeAlso**

caption SEE ALSO

**\thead**

subheadline in manual pages

**\thead** [*toc entry*] *title*

...

**\spage**

partial page

**\spage** [*margin*] *flags*

The command **\spage** starts a partial page, obsolete.

**\src2doc**

create manual page from source

**\src2doc** [*flags*] *pattern list*

**\srcinclude**

include files of source list

**\srcinclude** *pattern list*

**\ssec**

subsection

**\ssec** [*toc entry*] *title*

...

**\style**

document style

The command **\style** sets the style of the document.

**\Synopsis**

caption SYNOPSIS

**\t** tabulator key

**\t**

The command **\t** writes a tabulator key back to the input stream.

**\tab**

tabular

**\tab** [*pos*] *fmt*  
*tab lines*

The command **\tab** starts a tabular with vertical adjustment *pos* and column format *fmt*. The tabular ends by the first empty line.

**\table**

table

**\table** [*pos*]

The command `\table` starts a new table environment.

#### `\thedata`

get date

The command `\thedata` fetches the actual setting of the date.

#### `\title`

document title

```
\title [toc entry] title
      subtitle
```

If the document is not started, the command sets the document title, else it starts a new chapter. The optional argument *toc entry* is only used in the later case. The *subtitle* is ignored in the later case.

#### `\toc`

table of contents

The command `\toc` creates a table of contents.

#### `\ttitem`

typewriter item

The command starts a new description entry with a typewriter item *tag*.

#### `\url`

reference to uniform resource locator

```
\url |url| {label}
```

#### `\vref`

combined reference (page and value)

#### `\vspace`

vertical adjustment

```
\vspace offset
```

The command `\vspace` closes the current paragraph and performs a vertical shift of *offset* lines.

#### `\Warnings`

caption WARNINGS

## MACROS

This sections contains a list of all predefined macros of `efeudoc`.

#### `@Copyright`

impressum

```
@Copyright (year)
```

#### `@if`

conditional expression

```
@if (expr, ifpart, elsepart)
```

#### `@MSEExample`

include mksource example

```
@MSEExample (name,text)
```

The makro `@MSEExample` includes a `mksource` example script and the target file(s), where *name* is the name of the script and *text* is the separating text between source and target file(s).

**@PRM**

reference to the *Programmer's Reference Manual*

**@TODO**

note for incomplete paragraph

**ENVIRONMENT****APPLPATH**

path for configuration files.

**LANG**

locale information

**SEE ALSO**

*esh(1)*.

**COPYRIGHT**

Copyright (C) 1999 Erich Frühstück

**NAME**

efeuscript – install script file

**SYNOPSIS**

```
efeuscript [ --help[=type] ] [ --version ] [ -u ] [ -g ] [ -e ] [ -r ] [ -s /expr/repl/ ] [ -c name ]
          src tg
```

**DESCRIPTION**

The command completes the script *src* with a interpreter key (#!). The complete path of the interpreter is discovered.

If the script contains already an interpreter key, it is only checked and not modified.

The following options and arguments are accepted from command **efeuscript**:

**--help**[=*type*]

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for **efeudoc**

man

output nroff/troff source for **man**

lp send postscript data to **lpr**

**--version**

show version information

**-u** only user gets exec permission

**-g** only user and group gets exec permission

**-e** use /usr/bin/env to start interpreter

**-r** use exec to start interpreter

**-s** /*expr*/*repl*/

use sed to replace *expr* with *repl*. This Option may be repeated.

**-c** *name*

name of interpreter, default cmd

*src* path of source script

*tg* path of target source

**COPYRIGHT**

Copyright (C) 2001 Erich Frühstück

**NAME**

efile – file command with efeu extensions

**SYNOPSIS**

```
efile [ --help[=type] ] [ --version ] file(s)
```

**DESCRIPTION**

**efile** implements the **file** command in a efeu specific environment. For files with suffix **.gz** the first Block is uncompressed and analysed by **file**.

The following options and arguments are accepted from command **efile**:

**--help[=*type*]**

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for **efeudoc**

man

output nroff/troff source for **man**

lp

send postscript data to **lpr**

**--version**

show version information

*file(s)*

list of files to determine file types

**SEE ALSO**

efeu-magic(1), file(1), magic(5).

**COPYRIGHT**

Copyright (C) 2001, 2008 Erich Frühstück

**NAME**

esh – EFEU command interpreter

**SYNOPSIS**

```
esh [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -I dir ]
    [ -D name=val ] [ -c string ] [ -E ] [ file ] [ arg(s) ]
```

**DESCRIPTION**

The command `esh` evaluates scripts in the syntax of the EFEU interpreter language. The syntax of the language is similar to C/C++. If you are familiar with this language(s), you would easily learn to use this interpreter.

`esh` accepts comments in the style of C/C++ and uses a preprocessor similar to C/C++. See later in this document. Lines starting with `#!` are not interpreted by `esh`. If the script is executable and the first line is

```
#!path
```

where *path* is the full path name of the command `esh`, you can use it like an ordinary command. I prefer the following variation, which is independent from the installation place of `esh`:

```
#!/usr/bin/env esh
```

Expressions are terminated either by a semicolon or a linefeed, tabs and spaces are skipped. An expression may also end if there is no right operator following a term. In this case and if the next character is a punctuation character, it is used as termination key, else a space is used. On some places, e.g. inside the argument list of a function, a linefeed does not terminate the expression and is skipped like tabs or spaces.

In the outermost level (outside any block or function body) every statement is evaluated immediately after parsing. If an expression is not terminated by a semicolon, the result is written to standard output followed by the terminating character.

For example: The line

```
3 * 5 4 + 7 $ 2 - 1; 4 + 1
```

is split into the 4 expressions

```
3 * 5 terminated by space
4 + 7 terminated by $
2 - 1 terminated by ;
4 + 1 terminated by linefeed
```

and the output is

```
15 11$5
```

in outermost level.

If `esh` is called without script name or if the script name is a single minus, commands are read from standard input. If standard input and standard output is connected to a terminal, `esh` runs interactive and `readline` is used for reading lines from terminal. Readline control keys are active and `!` at beginning of a new line is used as control key to run history and system commands.

The use of `readline` in interactive mode and the automatic display of results in the outermost level allows to use `esh` as a comfortable desk calculator.

The complete EFEU interpreter language is implemented with C library functions. `esh` is a simple command which uses this functions. The interpreter shares data pointers directly with C functions. You can add your own functions and types to the interpreter. So it is easy to use this language for configuration files or to test functions with it.

If EFEU is build with shared libraries (e.g. on Linux) you can expand `esh` at run time. If shared libraries are not available, you can take a copy of `esh.c` and add your extensions there.

## Options

Options placed after the script name are interpreted by the script. The options `-?` and `--help` show you the syntax of the script.

The following options and arguments are accepted by `esh`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

`--info[=entry]`

show command information

`--debug[=mode]`

set debug level for command See for details.

`--verbose`

set debug level to `.info`.

`-I dir`

append *dir* to the search path for script headers. The default setting is `"/home/efeu/www/efeu-3.3-1/`

`-D name=val`

define macro *name* with value *val*

`-c string`

process commands from *string*.

*file* name of the script file.

*arg(s)*

script parameters.

`-E` preprocess only, do not evaluate commands.

## PREPROCESSOR

The EFEU interpreter language uses a preprocessor similar to C. The preprocessor is built in as a filter and can be used independent of the interpreter. The preprocessor evaluates the input per

line and not per file. So you can create or change variables which may be used later in conditional directives.

### Including files

Files are included by the `#include` directive. In the first step, macro substitution is performed on the rest of the line starting with `#include`. Afterwards, the following cases are possible:

`#include <name>`

This variant searches files in a list of directories, specified by the variable `IncPath`. Characters after `<` are silently ignored.

`#include expr`

First, `expr` is evaluated and converted to string. If the resulting string does not start with `<`, the current directory is searched before any other directory in `IncPath`.

`#include "name"`

This is only a special form of the above clause. The file `name` is first searched in the current directory.

In esh, the following construction is legal:

```
str header = paste("/", "SubDir", "MyHeader");
#include "<" + header + ">"
```

The variable `header` is defined in the outermost level, so it is immediately executed and can be used in the following `#include` directive. Adding `<` and `>` avoids searching the current directory (if `IncPath` does not include the current directory).

### Conditionals

A conditional begins with a 'conditional directive': `#if`, `#ifdef` or `#ifndef` and ends with `#endif`. A conditional block may contain `#elif` and `#else` directives. Conditional blocks may be nested.

The directives

`#ifdef name`

and

`#ifndef name`

are used to test macro definitions. No macro substitution is performed on this directive lines.

The simplest form of a conditional is:

```
#if expr
Input lines seen if expr is true.
#endif
```

A more complex conditional may look like this:

```
#if expr1
Input lines seen if expr1 is true.
#elif expr2
Input lines seen if expr2 is true and expr1 is false.
#else
Input lines seen if neither expr1 or expr2 is true.
#endif
```

As seen in the section ‘Including files’, you can use any variable or function in *expr* previously declared in the outermost level.

## Macros

Macros are defined with the `#define` directive. It has two forms:

`#define name replacement`  
defines a macro without arguments.

`#define name(args) replacement`  
defines a macro with arguments. The left parenthesis must follow immediately the name of the macro.

The name of a macro must start with an alphabetic or underline character and may contain only alphanumeric or underline characters.

In `esh` macros are rarely used. In most of all places, variables and functions are the better solution. Normally they are only used to protect header files for multiple inclusions.

A macro could be removed with the `#undef` directive.

## EXPRESSIONS

Constants and variables could be joined with operators to single expressions.

The next tables show the available operators of `esh`. They are sorted by descending priority. Operators not separated by a line have the same priority.

Prefix operators		
<code>::</code>	global	<code>::<i>name</i></code>
<code>++</code>	pre increment	<code>++<i>lvalue</i></code>
<code>--</code>	pre decrement	<code>--<i>lvalue</i></code>
<code>~</code>	complement	<code>~<i>expr</i></code>
<code>!</code>	not	<code>!<i>expr</i></code>
<code>-</code>	unary minus	<code>-<i>expr</i></code>
<code>+</code>	unary plus	<code>+<i>expr</i></code>
<code>{</code>	list grouping	<code>{ <i>expr</i> [, <i>expr</i> ] }</code>
<code>(</code>	grouping	<code>( <i>expr</i> )</code>
<code>[</code>	expression	<code>[ <i>expr</i> ]</code>
<code>()</code>	cast (type conversion)	<code>(<i>type</i>) <i>expr</i></code>

The list grouping operator creates a list of values. In opposite to C/C++, the use of this operator is not restricted to assigning data in variable declarations.

The expression operator parses an expression without evaluation. This expression may be stored in a variable or passed as function argument for later evaluation.

Postfix operators		
++	post increment	<i>lvalue++</i>
--	post decrement	<i>lvalue--</i>
::	scope resolution	<i>type::name</i>
::	variable selection	<i>vartab::name</i>
.	member selection	<i>expr.name</i>
[]	sub scripting	<i>expr[expr]</i>
()	function call	<i>expr(list)</i>

In esh, you have access to a table of variables and you can create your own variable tables. The scope resolution operator is used to get a member of this table. You can apply the operator to any type, that could be converted in the type `VarTab`.

Arithmetic operators		
*	multiply	<i>expr * expr</i>
/	division	<i>expr / expr</i>
%	modulo (remainder)	<i>expr % expr</i>
+	add (plus)	<i>expr + expr</i>
-	subtract	<i>expr - expr</i>
<<	shift left	<i>expr &lt;&lt; expr</i>
>>	shift right	<i>expr &gt;&gt; expr</i>

Comparison operators		
<	less than	<i>expr &lt; expr</i>
<=	less than or equal	<i>expr &lt;= expr</i>
>	greater than	<i>expr &gt; expr</i>
>=	greater than or equal	<i>expr &gt;= expr</i>
==	equal	<i>expr == expr</i>
!=	not equal	<i>expr != expr</i>

Bit wise operators		
&	bit wise AND	<i>expr &amp; expr</i>
^	bit wise exclusive OR	<i>expr ^ expr</i>
	bit wise inclusive OR	<i>expr   expr</i>

Logical operators		
&&	logical AND	<i>expr &amp;&amp; expr</i>
	logical OR	<i>expr    expr</i>

The right operand of a logical operator is only evaluated, if the resulting value is not determined by the left operand. So in

```

false && expr
true || expr

```

the right operand *expr* is never evaluated.

Conditional and range operator		
<code>? :</code>	conditional operator	<i>cond ? expr1 : expr2</i>
<code>:</code>	range operator	<i>start : end [ : step ]</i>

The range operator creates a list of variables, starting by *start* and ending by *end*. For numbers, the value of *step* must be positive. The default value is 1. Wrong use of this operator may result in an infinite loop. There is no range operator in C/C++.

Assign operators		
<code>=</code>	simple assignment	<i>lvalue = expr</i>
<code>*=</code>	multiply and assign	<i>lvalue *= expr</i>
<code>/=</code>	divide and assign	<i>lvalue /= expr</i>
<code>%=</code>	modulo and assign	<i>lvalue %= expr</i>
<code>+=</code>	add and assign	<i>lvalue += expr</i>
<code>-=</code>	subtract and assign	<i>lvalue -= expr</i>
<code>&lt;&lt;=</code>	shift left and assign	<i>lvalue &lt;&lt;= expr</i>
<code>&gt;&gt;=</code>	shift right and assign	<i>lvalue &gt;&gt;= expr</i>
<code>&amp;=</code>	AND and assign	<i>lvalue &amp;= expr</i>
<code>^=</code>	exclusive OR and assign	<i>lvalue ^= expr</i>
<code> =</code>	inclusive OR and assign	<i>lvalue  = expr</i>

Assign operators are right associative.

List separator		
<code>,</code>	list separator	<i>expr , expr</i>

The comma `,` in esh is used as list separator like python, not as comma operator like C/C++. So *a, b, ..., n* returns a list containing *a, b, ..., n*.

If you do not use the return value (in the most common use), there is no difference between the comma operator and the list separator.

## Loops

`while (cond) expr`

As long as *cond* is true, *expr* is executed.

`do expr while (cond)`

The expression *expr* is executed and repeated as long as *cond* is true.

`for (a; cond; b) expr`

First, *a* is evaluated. As long as *cond* is true, *expr* and then *b* is evaluated. Either *a*, *cond* or *b* may be omitted, *a* and *b* must be simple expressions.

`for (name in list) expr`

The representative *name* is set to each element of the list in turn, and *expr* is evaluated each time. If *list* consists of a single element and its type is convertible to `List_t`, the result of the conversion is used instead.

In any case of loop, the statement `break` breaks out of the loop and the statement `continue` starts the next cycle.

## Conditionals

```
if (cond) expr1
```

If the expression *cond* is true, *expr1* is executed.

```
if (cond) expr1 else expr2
```

If the expression *cond* is true, *expr1* is executed, else *expr2* is executed.

### Switch statement

The syntax of a `switch` statement is:

```
switch (expr)
{
  label:
      cmdlist
  label:
      cmdlist
  ...
}
```

where *label* may be `case val` or `default`. The expression *val* is evaluated on parsing and not on executing the switch statement. The value of *expr* is compared with all labels in order of definition. If the comparison is true, all following statements until `break`, `continue`, `return` or the end of the switch block is reached, are executed. If none of the labels compares with *expr*, all statements after `default` (if present) are executed.

In opposite to C, any data type is allowed in the switch statement as long as the operator `==` is defined. In particular you can use strings in switch statements.

### Grouping

Braces are used to group expressions to a block. A block has no return value. Every block has two tables of variables. The less visible table is created by parsing the command lines, the more visible table on evaluating the block. Every expression following the keyword `static` is executed immediately after parsing. So type declaration with the prefix `static` creates variables in the less visible table. The use of `static` is not restricted to declarations.

## DECLARATIONS AND CONSTANTS

In opposite to other interpreter languages, variables must be declared like C/C++ before use. A declaration may be placed anywhere in the source and has a return value (the value of the declared variable).

For example:

```
int x;
double a, b;
x = (int y = 5);
```

declares first the integer variable *x* and the double variables *a* and *b*. Afterwards the integer variable *y* is declared with value 5 and the result (the value 5) is assigned to *x*. It is allowed to declare a variable more than once with the same type. All but the first declarations are converted to a assignment statement.

Every predefined data type in the interpreter language has a corresponding data type in C. The EFEU interpreter language does not have pointers, but data types may be represented by pointer types in C.

The interpreter distinguishes between lvalues and constants. An lvalue is anything, that could stand on the left side of an assignment. Typical lvalues are variables. The result of an expression or a function call may be an lvalue or not.

### Integral types

Like in C/C++, there are several different integral types. The interpreter supports the exact-width integer types as defined in C99. They are `int8_t`, `int16_t`, `int32_t`, `int64_t`, `uint8_t`, `uint16_t`, `uint32_t` and `uint64_t`. The following table shows additional integral types and their representation in C.

esh type	C type
<code>bool</code>	<code>int</code>
<code>int</code>	<code>int</code>
<code>unsigned</code>	<code>unsigned int</code>
<code>varint</code>	<code>int64_t</code>
<code>varsize</code>	<code>uint64_t</code>

The keyword `unsigned` is a type name and not a type qualifier like in C. The data types `varint` and `varsize` have a variable length binary representation in data files.

The syntax of integral constants is like C/C++. The keywords `true` (integral value 0) and `false` (integral value 1) are used for boolean values.

### Floating number types

Esh uses the types `float` and `double` like C/C++. Floating number constants are always of type `double`. All arithmetic is done by `double`, `float` is defined for completeness and to save space for large data fields, where precision is not so important.

### Characters and strings

A character in esh has the type `char` and its code value is considered as unsigned.

Strings are completely different implemented in `esh` than in C. They are not fields of type `char`, they have the data type `str`. If you assign a string to a value or use it as function argument, the whole string (not the address) is copied. Copying strings is always done with memory allocation and there is a built in garbage collection for it (and generally for all dynamic allocated objects).

Character constants are delimited by single quotes, string constants by double quotes. The backslash is used as escape character like in C.

String constants may contain linefeed. In esh, two strings next to each other are not concatenated like in C/C++. You need the add operator `+` to do this.

For long string constants there exists the keyword `string`, which is used in the following form:

```
string !
  contents of string
!
```

There must be a newline after `!` in the starting line and `!` must be the first character in the last line. A so defined string always contains a linefeed at end. The backslash is no longer used as escape character with one exception: protecting a `!` at beginning of line to be interpreted as string termination. This construction of strings may be used anywhere inside an expression.

Comments are skipped and preprocessor directives are interpreted inside this string definitions.

For example: you can write

```
str s = string !
#include "file"
!;
```

to get the file `file` included in the string `s`.

Note, that in EFEU (and so in esh) null strings (character pointer to NULL) can be used like ordinary strings and there is a difference between null strings and empty strings (strings containing only the terminating 0) are handled. The EFEU libraries contains tools for handling dynamic allocated strings and you can mix them with constant strings. The memory allocation tools in EFEU knows, if the memory of a string could be freed.

### Pointer types

Data types, which are implemented by pointers, assign only the pointer address and do not copy the data (strings are an exception). But they use a reference counter for garbage collection. If the pointer is no more used, the whole memory allocated to this pointer is freed. All this types are subclasses of the type `_Ref_`.

The type `_Ref_` and all other types with a pointer representation in C are subclasses of `_Ptr_`. This is also the type of the constant `NULL`.

Types which starts and ends with the underline symbol are reserved for internal use. Normally, you don't declare variables of this type. But this types may be used for arguments in virtual functions, e.g. to distinguish between the constant `NULL` and a string initialized with `NULL`.

### Lists

Lists are a ordered set of any data variables. They have the data type `List_t`. There are three ways to get lists in the EFEU interpreter language:

1. with the list grouping operator: `{ 3, 5 }`. You can not use it at begin of a statement because `{` is also used for grouping expressions.
2. with the list operator: `3, 5`. Note the low priority of the list operator. You need parentheses to use it in terms. You get only lists of at least two elements.
3. with the range operator: `3 : 5 : 2`. All members of the list have the same type.

Any object of type `List_t` have the two components:

`obj`

returns the first entry of the list or `NULL` for empty lists.

`next`

returns the sub list starting with the next element or an empty list, if there are no more elements.

In absence of pointers in esh, you may use `List_t` as substitution.

### Data fields

Data fields may be declared in one of the two possible forms:

`type name[dim]`

declares `name` as a field of type `type`.

`type[dim] name`

declares `name` as a scalar of type `type[dim]`.

In the first case and if the data field is initialized by a list of values, `dim` may be omitted and the number of elements in the list determines the size of the field. In the second case, a new type is implicit created. The field size is necessary.

In the second case 0 or a missing value of `dim` indicates a variable length array. The data field is implicit enlarged as you use a higher index. Data types of the form `type[]` are subclasses of `EfiVec`.

If you have more than one dimension, a declaration of the form

```
type name[n0][n1]...[nk];
```

is translated into

```
type[nk]...[n1 name][n0];
```

because data fields can only have one dimension, but there is no limit on creating vector types. It is clear that only *n0* may be omitted.

Data fields are always packed into a object of type `EfiVec` on use. A data field can always be converted to a list and you can assign values to a data field with a list. If the list has less elements than the data field, only the corresponding elements are changed.

EFEU provides you the following data types for a more powerful handling of data than ordinary data fields:

#### `TimeSeries`

are dynamic fields of type double with a time index.

#### `mdmat`

holds a data cube of any type with unlimited number of dimensions, only restricted by available memory. EFEU contains a lot of tools for handling such data cubes.

### Creating new data types

The simplest way to create a new type is `typedef`, as in

```
typedef int myint;
```

The new type `myint` is created as subclass of `int`, not as alias.

Structures are created with the `struct` statement. The syntax is

```
struct type [: base [ name ]] { type declarations }
```

like in C++. If *base* is defined, *type* is created as subtype of *base*. Only one base type may be specified.

The following two types

```
struct T1 {
    int a;
    int b;
}

struct T2 : int a {
    int b;
}
```

have the same components, but `T2` can be used as representation of an integer.

Any previously defined type can be used in this form of type declaration. Any structure type could be converted to a list and any list with corresponding elements could be converted to a structure type.

Example for a more complex structure:

```
struct MyDataType {
    int i;
```

```

    double d;
    str s;
    int v[10];
};

```

The EFEU interpreter supports enumeration types. The syntax is

```
enum type [: base [ name ]] { identifier list }
```

with a comma separated list of identifiers with optionally assignment values. If *base* is defined, *type* is created as subtype of *base*, else as subtype of `_Enum_`. The enumeration keys are bound to the enumeration type. They may be used immediately after declaration.

The following statement:

```
enum Color { Red, Green = 5, Blue };
```

creates an enumeration type with name `Color` and identifiers `Color::Red`, `Color::Green` and `Color::Blue`. The corresponding integer values are 0, 5 and 6. For every enumeration type, converts from/to `int/str` are created. So the following assignments are all valid:

```

Color c1 = "Red";
Color c2 = 0;
str s = Color::Red;
int n = Color::Red;

```

The function `enumlist(type)` returns a list of all valid identifiers of the enumeration type *type* or an empty list, if *type* has no identifiers or is not an enumeration type.

## FUNCTION DECLARATION

A function declaration in esh has the general form

```
type name ( arglist )
      expr
```

Normally *expr* is a block structure, but in esh you can also use a single (but not empty) expression. If the function does not return any value, use the special type `void`.

The following function declarations are equivalent:

```

int f (int x) x + 1;
int f (int x) return x + 1;
inline int f (int x) { return x + 1; }

```

In esh, the keyword `inline` has nothing to do with optimization, but with visibility. A inline function sees all variable tables as in the line where it is called. All functions defined with a single expression are default of type inline.

Here is a example where inline functions are needed:

```

inline str f (str fmt)
{
    return psub(fmt);
}

{
    str x = "foo";
    f("x = $(x)");
}

```

The function `psub` substitutes parameter according to a format string. If `f` is not inline, `psub` does not see the variable `x` and the substitution `$(x)` would fail.

Functions have the type `Func` and you can use it like function pointers in C. Typing the function name in outermost level gives you the prototype of the function.

As in C++ function arguments may have default values. The general form of a function argument is:

```
type [ & ] name [ = value ]
```

The `&` key indicates that the argument must be an lvalue. A tilde `...` stands for a variable list of arguments. Inside the function this list could be referenced under the reserved name `va_list`.

Behind the most operators stands a function with the name of the operator. You can use either `operator "name"` or `operatorname<space>` for such function names. For example: `operator+` is the name of the addition function. Function names of prefix operators have an additional `()` in the name to distinguish between postfix and binary operators. So `operator+()` is the name of the unary plus.

### Virtual functions

As in C++, you can overload functions with different argument lists. The keyword `virtual` is used to declare virtual functions. They have the data type `VirFunc`. Any function can be converted to a virtual function.

You can convert a virtual function to a regular function with a prototype cast:

```
Func f = operator+ (int a, int b);;
```

Now you can use `f` to add two integer values. Note the two semicolons: The first is part of the prototype, the second terminates the expression and may be replaced by a linefeed.

### Type bound functions

Functions may be bound to an specific type. They have the general form

```
type btype::name [ & ] ( arglist )
      expr
```

The `&` after the function name indicates that it can only be used for lvalues. A bounded function is called

```
obj.name(args)
```

where `obj` is an object of type `btype`. Object bound functions have the type `ObjFunc` and may be virtual or not.

All assignment operators are bounded functions. In bounded functions you can use `this` to refer to the corresponding object.

### Special Functions

Functions which have the same name as a type, defines constructors and converters.

Constructors have the form:

```
virtual type type ( arglist )
```

The special form

```
type type ()
```

is called the copy constructor.

The declaration

```
type type (void)
```

is a normal constructor without arguments.

Converters have the form:

```
tg_type src_type ()
```

with an empty argument list. The source data is referred under the name `this`. If the target type is `void` the function defines the destructor of the type.

Because of internal garbage collection, there is normally no need on copy constructor and destructor. You must be very carefully in defining this functions, because you get an infinite recursive call if an object of the type is copied inside the function.

## MISSING SOMETHING

If you are missing something in the documentation, you may get it from esh. The option `--info` provide an interface to builtin information, just enter the command `esh --info`. If you are running esh in interactive mode, you can get the information with the function call `Info()`.

If you want to know how a function is used, just enter the function name and the prototype is displayed. For a data type *type* you can call the function *type.info()* to get additional information's.

`global`

is the table of global variables

`local`

is the table of local variables. In outermost level `local` is identical to `global`.

`str whatis (.)`

give you information about the argument. returns the type of an variable.

`void vtabstack (int = 0, IO = iostd)`

show the current stack of the variable tables.

`List_t typelist ()`

give you a list of all data types.

## ENVIRONMENT

`APPLPATH`

path for configuration files.

`LANG`

locale information

`ESHPATH`

define extra directories for searching script headers.

## COPYRIGHT

Copyright (C) 1994, 2001 Erich Frühstück

**NAME**

findgrep – search pattern in files of directories

**SYNOPSIS**

```
findgrep [ --help[=type] ] [ --version ] [ grep-options ] [ pat | -e pat | -F pat | -f file ] dir  
[ find-options ]
```

**DESCRIPTION**

The command `findgrep` search patterns with `grep` in a list of files collected by `find`. To avoid environment space overflow `xargs` is used to execute `grep`. The grep options are passed through and not checked.

Find is called with the flags `-type f` and `-print`. If supported, `-print` is replaced by `-print0`. This allows to handle files with special characters like spaces.

**SEE ALSO**

`grep(1)`, `find(1)`, `xargs(1)`.

**COPYRIGHT**

Copyright (C) 2001 Erich Frühstück

**NAME**

htmlindex – create index.html for current directory

**SYNOPSIS**

```
htmlindex [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ]  
          [ name ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `htmlindex`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

`--info[=entry]`

show command information

`--debug[=mode]`

set debug level for command See for details.

`--verbose`

set debug level to `.info`.

*name*

output file

**ENVIRONMENT**

`APPLPATH`

path for configuration files.

`LANG`

locale information

**COPYRIGHT**

Copyright (C) 2001 Erich Frühstück

**NAME**

mdcat – concatenate data cubes

**SYNOPSIS**

```
mdcat [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -T title ]
      [ -n name ] [ -m map ] [ -o out ] [ file(s) ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdcat`:

`--help[=type]`  
 create command usage. The optional parameter *type* determines the formatting and output of the description.

term  
 display on terminal (default)

raw  
 output raw data for `efeudoc`

man  
 output nroff/troff source for `man`

lp  
 send postscript data to `lpr`

`--version`  
 show version information

`--info[=entry]`  
 show command information

`--debug[=mode]`  
 set debug level for command See for details.

`--verbose`  
 set debug level to `.info`.

`-T title`  
 set title of output file

`-n name`  
 name of axis, default X

`-m map`  
 create file mapping

`-o out`  
 output file

*file(s)*  
 input file(s)

**ENVIRONMENT**

`APPLPATH`  
 path for configuration files.

`LANG`  
 locale information

**COPYRIGHT**

Copyright (C) 1997, 2001 Erich Fröhstück

**NAME**

mdcmp – compare two data cubes

**SYNOPSIS**

```
mdcmp [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -F file ]
      [ -E expr ] [ -f func ] [ -e expr ] [ -r ] [ -v var ] [ -T title ] file1 file2 { name=var } [ aus ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdcmp`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

`--info[=entry]`

show command information

`--debug[=mode]`

set debug level for command See for details.

`--verbose`

set debug level to `.info`.

`-F file`

execute command lines in file *file*

`-E expr`

evaluate expression *expr*

`-f func`

compare function

`-e expr`

compare expression

`-r` remove singular axis

`-v var`

select type components

`-T title`

set title of output file

*file1*

name of first data cube

*file2*  
name of second data cube

*name=var*  
selection parameter

*aus*  
output file

## **ENVIRONMENT**

**APPLPATH**  
path for configuration files.

**LANG**  
locale information

## **COPYRIGHT**

Copyright (C) 1997, 2001 Erich Fröhstück

**NAME**

mdcreate – create data cube

**SYNOPSIS**

```
mdcreate [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ]
        [ -v value ] [ -T title ] [ -m map ] type { name=var } [ aus ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdcreate`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

`--info[=entry]`

show command information

`--debug[=mode]`

set debug level for command See for details.

`--verbose`

set debug level to `.info`.

`-v value`

default value

`-T title`

set title of output file

*type*

type of data

`-m map`

create file mapping

*name=var*

list of axis

*aus*

name of output file

**ENVIRONMENT**

**APPLPATH**

path for configuration files.

**LANG**

locale information

**COPYRIGHT**

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mddiag – diagonalize axis of data cube

**SYNOPSIS**

```
mddiag [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -r ]
      [ -v var ] [ -T title ] [ -x axis ] [ -m map ] ein { name=var } [ aus ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mddiag`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

`--info[=entry]`

show command information

`--debug[=mode]`

set debug level for command See for details.

`--verbose`

set debug level to `.info`.

`-r` remove singular axis

`-v var`

select type components

`-T title`

set title of output file

`-x axis`

name of axis

`-m map`

create file mapping

*ein* input file

*name=var*

selection parameter

*aus*

output file

**ENVIRONMENT**

**APPLPATH**

path for configuration files.

**LANG**

locale information

**COPYRIGHT**

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mddiff – show differences of data cubes

**SYNOPSIS**

```
mddiff [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ]
      [ -E expr ] [ -F file ] [ -r ] [ -v var ] [ -p prec ] file1 file2 { name=var } [ out ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mddiff`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

`--info[=entry]`

show command information

`--debug[=mode]`

set debug level for command See for details.

`--verbose`

set debug level to `.info`.

`-E expr`

evaluate expression *expr*

`-F file`

execute command lines in file *file*

`-r` remove singular axis

`-v var`

select type components

`-p prec`

precision of floating numbers

*file1*

name of first data cube

*file2*

name of second data cube

*name=var*

selection parameter

*out* name of output file

## **ENVIRONMENT**

### **APPLPATH**

path for configuration files.

### **LANG**

locale information

## **COPYRIGHT**

Copyright (C) 1998, 2001 Erich Frühstück

**NAME**

mdfile – show structure of data cube

**SYNOPSIS**

```
mdfile [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -a ]
      [ -s ] [ -h ] [ -d ] [ -X ] [ -x ] [ -l ] [ file(s) ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdfile`:

`--help[=type]`  
 create command usage. The optional parameter *type* determines the formatting and output of the description.

term  
     display on terminal (default)

raw  
     output raw data for `efeudoc`

man  
     output nroff/troff source for `man`

lp   send postscript data to `lpr`

`--version`  
 show version information

`--info[=entry]`  
 show command information

`--debug[=mode]`  
 set debug level for command See for details.

`--verbose`  
 set debug level to `.info`.

`-a`   status of all files

`-s`   show structure of data cube

`-h`   show type and header

`-d`   show description

`-X`   short axis list

`-x`   verbose axis list

`-l`   long listing, equivalent to `-hdx`

*file(s)*  
 files

**ENVIRONMENT**

`APPLPATH`  
 path for configuration files.

`LANG`  
 locale information

**COPYRIGHT**

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mdfunc – data cube function

**SYNOPSIS**

```
mdfunc [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -F file ]
      [ -E expr ] [ -r ] [ -m map ] [ -v var ] [ -T title ] [ --neg ] [ --not ] [ --cpl ] [ --rnd ]
      [ -f func ] [ -t type ] [ -e expr ] [ -x list ] [ -p prec ] file { name=var } [ out ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdfunc`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

`--info[=entry]`

show command information

`--debug[=mode]`

set debug level for command See for details.

`--verbose`

set debug level to `.info`.

`-F file`

execute command lines in file *file*

`-E expr`

evaluate expression *expr*

`-r` remove singular axis

`-m map`

create file mapping

`-v var`

select type components

`-T title`

set title of output file

`--neg`

negation

`--not`

Boolean not

**--cpl**  
bitwise complement

**--rnd**  
round to nearest integer

**-f *func***  
function/operator

**-t *type***  
data type

**-e *expr***  
expression

**-x *list***  
list of column axis, default "#-1"

**-p *prec***  
floating number precision

***file*** input file

***name=var***  
selection parameter

***out*** output file

## ENVIRONMENT

**APPLPATH**  
path for configuration files.

**LANG**  
locale information

## COPYRIGHT

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mdmul – matrix multiplication

**SYNOPSIS**

```
mdmul [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -F file ]
      [ -E expr ] [ -r ] [ -m map ] [ -v var ] [ -T title ] [ -n dim ] [ -x axis ] [ -a name=var ]
      [ -b name=var ] file1 file2 { name=var } [ out ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdmul`:

- `--help[=type]`  
create command usage. The optional parameter *type* determines the formatting and output of the description.
- term  
display on terminal (default)
- raw  
output raw data for `efeudoc`
- man  
output nroff/troff source for `man`
- lp send postscript data to `lpr`
- `--version`  
show version information
- `--info[=entry]`  
show command information
- `--debug[=mode]`  
set debug level for command See for details.
- `--verbose`  
set debug level to `.info`.
- `-F file`  
execute command lines in file *file*
- `-E expr`  
evaluate expression *expr*
- `-r` remove singular axis
- `-m map`  
create file mapping
- `-v var`  
select type components
- `-T title`  
set title of output file
- `-n dim`  
number of common axis
- `-x axis`  
name of common axis

*-a name=var*  
selection parameter for first input file

*-b name=var*  
selection parameter for second input file

*file1*  
name of first data cube

*file2*  
name of second data cube

*name=var*  
selection parameter for both data files

*out* output file

## ENVIRONMENT

**APPLPATH**  
path for configuration files.

**LANG**  
locale information

## COPYRIGHT

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mdpaste – paste data cubes with regard to axis

**SYNOPSIS**

```
mdpaste [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -r ]
        [ -m map ] [ -v var ] [ -T title ] [ -s ] [ -u ] [ -e expr ] [ -n name ] [ -o aus ] { name=var }
        file(s) ...
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdpaste`:

- `--help[=type]`  
create command usage. The optional parameter *type* determines the formatting and output of the description.
- term  
display on terminal (default)
- raw  
output raw data for `efeudoc`
- man  
output nroff/troff source for `man`
- lp send postscript data to `lpr`
- `--version`  
show version information
- `--info[=entry]`  
show command information
- `--debug[=mode]`  
set debug level for command See for details.
- `--verbose`  
set debug level to `.info`.
- `-r` remove singular axis
- `-m map`  
create file mapping
- `-v var`  
select type components
- `-T title`  
set title of output file
- `-s` sort labels of axis
- `-u` sort labels of axis, use only last of repeated labels
- `-e expr`  
expression for comparing, default is "cmp(a,b)".
- `-n name`  
name of axis for pasting, default is #1
- `-o aus`  
output file

*name=var*  
selection parameter

*file(s) ...*  
list of input files.

## ENVIRONMENT

**APPLPATH**  
path for configuration files.

**LANG**  
locale information

## COPYRIGHT

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mdpermut – reorder axis of data cube

**SYNOPSIS**

```
mdpermut [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -r ]
        [ -v var ] [ -T title ] file { name=var } [ out ] [ name(s) ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdpermut`:

- `--help[=type]`  
create command usage. The optional parameter *type* determines the formatting and output of the description.
- `term`  
display on terminal (default)
- `raw`  
output raw data for `efeudoc`
- `man`  
output nroff/troff source for `man`
- `lp` send postscript data to `lpr`
- `--version`  
show version information
- `--info[=entry]`  
show command information
- `--debug[=mode]`  
set debug level for command See for details.
- `--verbose`  
set debug level to `.info`.
- `-r` remove singular axis
- `-v var`  
select type components
- `-T title`  
set title of output file
- file* input file
- name=var*  
selection parameter
- out* output file
- name(s)*  
name of axis to order first

**ENVIRONMENT**

**APPLPATH**  
path for configuration files.

**LANG**  
locale information

**COPYRIGHT**

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mdprint – print data cube

**SYNOPSIS**

```
mdprint [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ]
[ -F file ] [ -E expr ] [ -M mode ] [ --ascii ] [ --csv ] [ --tex ] [ --tab ] [ --data ] [ --sc ]
[ --std ] [ --ps ] [ --psq ] [ --lp ] [ --lpq ] [ --mark ] [ --nomark ] [ -h ] [ -b ] [ -H ]
[ -t ] [ -z ] [ -L loc ] [ --locale loc ] [ -f font ] [ -l width ] [ -3 ] [ -w width ] [ -p prec ]
[ -d delim ] [ -g ] [ -e ] [ -x list ] [ -s list ] [ -r ] [ -R list ] [ -v var ] [ -T title ] [ -P name=list ]
[ --paste name=list ] [ --paste-delim key ] [ -S list ] [ --split list ] file { name=var }
[ out ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdprint`:

- `--help[=type]`  
create command usage. The optional parameter *type* determines the formatting and output of the description.
- `term`  
display on terminal (default)
- `raw`  
output raw data for `efeudoc`
- `man`  
output nroff/troff source for `man`
- `lp` send postscript data to `lpr`
- `--version`  
show version information
- `--info[=entry]`  
show command information
- `--debug[=mode]`  
set debug level for command See for details.
- `--verbose`  
set debug level to `.info`.
- `-F file`  
execute command lines in file *file*
- `-E expr`  
evaluate expression *expr*
- `-M mode`  
set output mode, default is "std"
- `--ascii`  
prepare as ascii file
- `--csv`  
prepare for excel
- `--tex`  
prepare for LaTeX

`--tab`  
prepare for LaTeX tabular environment

`--data`  
print only data values

`--sc`  
output sc spreadsheet

`--std`  
fixed column alignment

`--ps`  
output PostScript file

`--psq`  
output PostScript file in landscape

`--lp`  
print data cube

`--lpq`  
print data cube in landscape

`--mark`  
mark column marker as comment

`--nomark`  
suppress column marker

`-h` show header only

`-b` show data only

`-H` show complete header

`-t` show only title in header

`-z` suppress 0 values

`-L loc, --locale loc`  
set locale for number format to *loc*

`-f font`  
font size in point

`-l width`  
width of line label

`-3` align exponent to multiples of 3

`-w width`  
field width for values

`-p prec`  
floating number precision

`-d delim`  
delimiter of lists

`-g` use universal float format

`-e` use exponential float format

**-x** *list*  
mark axis as column

**-s** *list*  
accumulate list of axis

**-r** remove singular axis

**-R** *list*  
remove singular axis from List *list*

**-v** *var*  
select type components

**-T** *title*  
set title of output file

**-P** *name=list*, **--paste** *name=list*  
paste list *list* of axis to new axis *name*

**--paste-delim** *key*  
delimiter for pasting axis, default: \_

**-S** *list*, **--split** *list*  
split data matrix by axis *list*.

*file* input file

*name=var*  
selection parameter

*out* output file

## ENVIRONMENT

**APPLPATH**  
path for configuration files.

**LANG**  
locale information

## COPYRIGHT

Copyright (C) 1997, 2001 Erich Fröhstück

**NAME**

mdread – read data cube

**SYNOPSIS**

```
mdread [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ]
[ -x name ] [ -y name ] [ -t type ] [ -M ] [ -L loc ] [ --locale loc ] [ --sc ] [ -r ] [ -m map ]
[ -v var ] [ -T title ] file { name=var } [ out ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdread`:

- `--help[=type]`  
create command usage. The optional parameter *type* determines the formatting and output of the description.
- term  
display on terminal (default)
- raw  
output raw data for `efeudoc`
- man  
output nroff/troff source for `man`
- lp send postscript data to `lpr`
- `--version`  
show version information
- `--info[=entry]`  
show command information
- `--debug[=mode]`  
set debug level for command See for details.
- `--verbose`  
set debug level to `.info`.
- `-x name`  
column axis name(s)
- `-y name`  
row axis name(s)
- `-t type`  
data type
- `-M` check magic number
- `-L loc, --locale loc`  
set locale for number format to *loc*
- `--sc`  
convert sc spreadsheet
- `-r` remove singular axis
- `-m map`  
create file mapping
- `-v var`  
select type components

*-T title* set title of output file  
*file* input file  
*name=var* selection parameter  
*out* output file

## ENVIRONMENT

**APPLPATH**  
path for configuration files.

**LANG**  
locale information

## COPYRIGHT

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mdround – round values of data cube

**SYNOPSIS**

```
mdround [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -r ]
        [ -m map ] [ -v var ] [ -T title ] [ -s val ] [ -m val ] file { name=var } [ out ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdround`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp

send postscript data to `lpr`

`--version`

show version information

`--info[=entry]`

show command information

`--debug[=mode]`

set debug level for command See for details.

`--verbose`

set debug level to `.info`.

`-r` remove singular axis

`-m map`

create file mapping

`-v var`

select type components

`-T title`

set title of output file

`-s val`

adjust data sum to value *val*

`-m val`

scalar multiplication with value *val*

*file* input file

*name=var*

selection parameter

*out* output file

**ENVIRONMENT**

**APPLPATH**

path for configuration files.

**LANG**

locale information

**COPYRIGHT**

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mdselect – select part of data cube

**SYNOPSIS**

```
mdselect [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ]
        [ -F file ] [ -E expr ] [ -r ] [ -m map ] [ -s list ] [ -v var ] [ -T title ] ein { name=var } [ aus ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdselect`:

- `--help[=type]`  
create command usage. The optional parameter *type* determines the formatting and output of the description.
- `term`  
display on terminal (default)
- `raw`  
output raw data for `efeudoc`
- `man`  
output nroff/troff source for `man`
- `lp` send postscript data to `lpr`
- `--version`  
show version information
- `--info[=entry]`  
show command information
- `--debug[=mode]`  
set debug level for command See for details.
- `--verbose`  
set debug level to `.info`.
- `-F file`  
execute command lines in file *file*
- `-E expr`  
evaluate expression *expr*
- `-r` remove singular axis
- `-m map`  
create file mapping
- `-s list`  
accumulate list of axis
- `-v var`  
select type components
- `-T title`  
set title of output file
- ein* input file
- name=var*  
selection parameter

*aus*  
output file

## **ENVIRONMENT**

**APPLPATH**  
path for configuration files.

**LANG**  
locale information

## **COPYRIGHT**

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mdsum – accumulate axis of data cube

**SYNOPSIS**

```
mdsum [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -r ]
      [ -v var ] [ -T title ] in out { name=var } [ name(s) ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdsum`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

`--info[=entry]`

show command information

`--debug[=mode]`

set debug level for command See for details.

`--verbose`

set debug level to `.info`.

`-r` remove singular axis

`-v var`

select type components

`-T title`

set title of output file

*in* input file name

*out* output file name

*name=var*

selection parameter

*name(s)*

name of axis to cumulate

**ENVIRONMENT**

**APPLPATH**

path for configuration files.

**LANG**

locale information

**COPYRIGHT**

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mdterm – evaluate data cube expression

**SYNOPSIS**

```
mdterm [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -F file ]
      [ -E expr ] [ --add ] [ --sub ] [ --mul ] [ --div ] [ --min ] [ --max ] [ -f func ] [ -t type ]
      [ -e expr ] [ -p prec ] [ -r ] [ -m map ] [ -v var ] [ -T title ] file1 file2 { name=var } [ out ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `mdterm`:

- `--help[=type]`  
create command usage. The optional parameter *type* determines the formatting and output of the description.
- `term`  
display on terminal (default)
- `raw`  
output raw data for `efeudoc`
- `man`  
output nroff/troff source for `man`
- `lp` send postscript data to `lpr`
- `--version`  
show version information
- `--info[=entry]`  
show command information
- `--debug[=mode]`  
set debug level for command See for details.
- `--verbose`  
set debug level to `.info`.
- `-F file`  
execute command lines in file *file*
- `-E expr`  
evaluate expression *expr*
- `--add`  
cell by cell addition
- `--sub`  
cell by cell subtraction
- `--mul`  
cell by cell multiplication
- `--div`  
cell by cell division
- `--min`  
cell minimum
- `--max`  
cell maximum

- f** *func*  
cell by cell function/operator
- t** *type*  
data type
- e** *expr*  
expression
- p** *prec*  
floating number precision
- r** remove singular axis
- m** *map*  
create file mapping
- v** *var*  
select type components
- T** *title*  
set title of output file
- file1*  
name of first data cube
- file2*  
name of second data cube
- name=var*  
selection parameter
- out* output file

## ENVIRONMENT

- APPLPATH**  
path for configuration files.
- LANG**  
locale information

## COPYRIGHT

Copyright (C) 1997, 2001 Erich Frühstück

**NAME**

mksource – program generator

**SYNOPSIS**

```
mksource [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ]
[ -I dir ] [ -s ] [ -V ] [ -v ] [ -l ] [ -L name ] [ -r ] [ -x ] [ --all name ] [ --clean name ]
[ --depend name ] [ --lock ] [ --unlock ] [ -d ] [ -D name ] [ -n name ] [ -h ] [ -c ] [ -H
hdr ] [ -C src ] [ -t ] [ -T doc ] name
```

**DESCRIPTION**

The following options and arguments are accepted from command `mksource`:

- `--help[=type]`  
create command usage. The optional parameter *type* determines the formatting and output of the description.
- term  
display on terminal (default)
- raw  
output raw data for `efeudoc`
- man  
output nroff/troff source for `man`
- lp send postscript data to `lpr`
- `--version`  
show version information
- `--info[=entry]`  
show command information
- `--debug[=mode]`  
set debug level for command See for details.
- `--verbose`  
set debug level to `.info`.
- `-I dir`  
appends *dir* to the search path for scripts.
- `-s` silent execution
- `-V` little verbose
- `-v` verbose
- `-l` create list of targets
- `-L name`  
create list of targets for *name*
- `-r` create rule
- `-x` no pseudo targets for make
- `--all name`  
target name
- `--clean name`  
clean target name

**--depend** *name*  
depend target name

**--lock**  
lock file creation

**--unlock**  
unlock file creation

**-d** make dependence

**-D name**  
make dependence for target *name*

**-n name**  
base name for created files

**-h** create header

**-c** create source

**-H hdr**  
create header with name *hdr*

**-C src**  
create source with name *src*

**-t** (obsolete) create TeX documentation, lock other outputs

**-T doc**  
(obsolete) create TeX documentation *doc*, lock other outputs

*name*  
script

## ENVIRONMENT

**APPLPATH**  
path for configuration files.

**LANG**  
locale information

## COPYRIGHT

Copyright (C) 1994 Erich Fröhstück

**NAME**

mksrclist – create source list

**SYNOPSIS**

```
mksrclist [ --help[=type] ] [ --version ] [ -f ] [ -x ] [ -e pat ] top [ name ]
```

**DESCRIPTION**

The command `mksrclist` creates a list of files under *top*.

The following options and arguments are accepted from command `mksrclist`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

`-f` force creation of output file

`-x` consider source generation with `mksource`.

`-e pat`

ignore files which matches pattern *pat*

*top* top directory of sources

*name*

name of output file to update

**SEE ALSO**

`shmkmf(1)`, `d2m.smh(7)`.

**DIAGNOSTICS**

The command `mksrclist` returns 2 on an error in creating the source list. If the source list is new or it has changed, the return code is 1. Otherwise the return code is 0.

**COPYRIGHT**

Copyright (C) 2000 Erich Frühstück

**NAME**

pp2dep – create dependence list from cpp output

**SYNOPSIS**

```
pp2dep [ --help[=type] ] [ --version ] [ -l ] [ -x pattern ] [ target(s) ]
```

**DESCRIPTION**

The command `pp2dep` greps the names of included files from the output of the C preprocessor and creates a dependend list for given targets.

The following options and arguments are accepted from command `pp2dep`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp

send postscript data to `lpr`

`--version`

show version information

`-l` list only local files

`-x pattern`

Exclude files which match *pattern*.

*target(s)*

list of targets

The next lines show the typical use of `pp2dep` in a Makefiles:

```
file.o: file.c
    $(CC) -c file.c

depend::
    $(CC) -E -c file.c | pp2dep -l file.o >> Makefile
```

**SEE ALSO**

*cc(1)*, *make(1)*, *shmkmf(1)*.

**NAME**

shmkmf – create Makefile from shell

**SYNOPSIS**

```
shmkmf [ --help[=type] ] [ --version ] [ -v ] [ -i ] [ -p ] [ -c string ] [ -x ] [ -r arg ] [ -t top ]
      [ -C dir ] [ -I dir ] [ config ] [ makefile ]
```

**DESCRIPTION**

The command `shmkmf` creates a Makefile from the script `Config.make`. The following options and arguments are accepted from command `shmkmf`:

The following options and arguments are accepted by `shmkmf`:

- `--help[=type]`  
create command usage. The optional parameter *type* determines the formatting and output of the description.
- `term`  
display on terminal (default)
- `raw`  
output raw data for `efeudoc`
- `man`  
output nroff/troff source for `man`
- `lp` send postscript data to `lpr`
- `--version`  
show version information
- `-v` write extra comments to the makefile
- `-i` read commands from stdin
- `-p` write rules to stdout
- `-c string`  
read commands from *string*
- `-x` do not create extra rules and comments
- `-r arg`  
create Makefile in a temporary directory and run `make` with the given argument *arg*.
- `-t top`  
sets `TOP` to *top* (default: `.`)
- `-C dir`  
expand config path with *dir*
- `-I dir`  
expand include path with *dir*
- config*  
name of configuration file (default: `Config.make`)
- makefile*  
name of target file (default: `Makefile`)

For normal use, [shmkmf](#) is called without arguments.

**Predefined variables**

IncludePath

Search Path for include files.

Config

Name of Configuration file.

Makefile

Name of target file.

CC

Name of C-compiler

All predefined functions to generate make-rules starts with 'mf\_'. Names starting with '\_' are reserved for internal use.

**SEE ALSO**

[ccmkmf\(1\)](#).

**NAME**

shmkmf-cflags – determine c-flags for use of external libraries

**SYNOPSIS**

```
shmkmf-cflags [ --help[=type] ] [ --version ] [ -m[name] ] [ -I[path] ] [ -L[path] ] [ -a ] [ -o ]
               [ -l lib ] [ name ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `shmkmf-cflags`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp

send postscript data to `lpr`

`--version`

show version information

`-m[name]`

define macro name *name* with found flags.

`-I[path]`

path to search header files (default: `/usr/include`).

`-L[path]`

path to search libraries (default: `/lib:/usr/lib`).

`-a`

try next search only if last search was successful.

`-o`

try next search only if last search failed.

`-l lib`

search for library corresponding to the regular expression `^lib(lib)\.(a|so)$`. On a success, print out the `-lname` flag for linking the library. If the library is in an other place than `/lib` or `/usr/lib` an `-Ldir` flag is also generated.

*name*

search directory *dir* which contains header *name*, where *name* may contain directory parts as in `X11/X.h`. If *dir* is different from `/usr/include`, print out the option `-Idir`.

**COPYRIGHT**

Copyright (C) 2008 Erich Frühstück

**NAME**

shmkmf-config – configuration tool for shmkmf

**SYNOPSIS**

```
shmkmf-config [ --help[=type] ] [ --version ] [ -v ] [ -x ] [ -s ] [ -c name ] [ -i hdr ] [ -I hdr ]
               [ -r hdr ] [ -f flgs ] cmd [ arg(s) ]
```

**DESCRIPTION**

The command `shmkmf-config` is used to determine system specific parameters. The normal use is to process a C header template with special test instructions. The instructions are isolated by `awk` and are evaluated by a call to `shmkmf-config`.

Some tests create c-sources and checks the compilation status. If the `-x` flag is set, the created program is executed and its output is inserted.

The following options and arguments are accepted from command `shmkmf-config`:

**--help[=*type*]**

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp

send postscript data to `lpr`

**--version**

show version information

**-v** display this help and exit

**-x** execute compiled command

**-s** read function body from standard input

**-c** *name*

C-Compiler

**-i** *hdr*

header for source

**-I** *hdr*

header for source if exists

**-r** *hdr*

header for shmkmf

**-f** *flgs*

compiler flags

*cmd*

command to perform

*arg(s)*

command specific arguments

The first argument determines the action of `shmkmf-config`. If it is `file` or `update`, a template file is evaluated, otherwise a special test is performed.

Every occurrence of `@SRC@` in the template file would be replaced by the name of the template file and `@CMD@` by `shmkmf-config`. The following keywords at start of a line are accepted:

`@set flags`

sets options for the following calls to `shmkmf-config`.

`@add flags`

expands the options for the following calls to `shmkmf-config`.

`@include file`

if header `file` exists, a include directive is created and the options for the following calls to `shmkmf-config` are expanded to include it.

`@eval args`

calls `shmkmf-config` with the given arguments.

`@cflags args`

calls `shmkmf-cflags` with the given arguments.

`@beg args`

...

`@end`

calls `shmkmf-config` with `-s` and the the given arguments. All lines between `@beg` and `@end` are send to `shmkmf-config`.

The following commands are performed by the command `shmkmf-config`

`file [name]`

inserts system specific parameters in file `name`

`update src tg`

performs `shmkmf-config` file on `src` and overwrites `tg` only if `src` is newer than `tg` or `tg` has changed.

`check expr`

check compilation/execution of `expr` without any additional message.

`report expr`

reports status of compilation/execution of `expr`.

`error message expr`

create `#error` directive with *message*, if compilation/execution fails.

`success` *tname* *expr*

if compilation/execution of *expr* is successful, define *tname* with 1, else with 0.

`failure` *tname* *expr*

if compilation/execution of *expr* failed, define *tname* with 1, else with 0.

`include` *name*

includes header *name* if available.

`proto` *proto*

check compatibility of prototype *proto*.

`typedef` *type* *decl*

if *type* is not defined, declare it with *decl*.

#### SEE ALSO

*shmkmf(1)*, *shmkmf-eflags(1)*.

#### COPYRIGHT

Copyright (C) 2002, 2008 Erich Frühstück

**NAME**

src2doc – create documentation from source

**SYNOPSIS**

```
src2doc [ --help[=type] ] [ --version ] [ --info[=entry] ] [ --debug[=mode] ] [ --verbose ] [ -t ]
        [ --ps ] [ -c ] [ -n name ] [ -m mode ] [ -s num ] [ -i fmt ] [ -g ] [ -L lang ] [ -l ] [ -a ] src
        [ out ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `src2doc`:

- `--help[=type]`  
create command usage. The optional parameter *type* determines the formatting and output of the description.
  - term  
display on terminal (default)
  - raw  
output raw data for `efeudoc`
  - man  
output nroff/troff source for `man`
  - lp  
send postscript data to `lpr`
- `--version`  
show version information
- `--info[=entry]`  
show command information
- `--debug[=mode]`  
set debug level for command See for details.
- `--verbose`  
set debug level to `.info`.
- `-t` terminal output
- `--ps`  
PostScript output
- `-c` insert program code
- `-n name`  
output name
- `-m mode`  
execution mode
- `-s num`  
manpage section
- `-i fmt`  
format for header file
- `-g` global header files
- `-L lang`  
set language

**-l** list execution modes

**-a** list alias table

*src* input file

*out* output file

## ENVIRONMENT

**APPLPATH**

path for configuration files.

**LANG**

locale information

## COPYRIGHT

Copyright (C) 2000 Erich Frühstück

**NAME**

tex2ps – process TeX-document

**SYNOPSIS**

```
tex2ps [ --help[=type] ] [ --version ] [ -q ] [ -n count ] [ -r res ] src [ ps ]
```

**DESCRIPTION**

The following options and arguments are accepted from command `tex2ps`:

`--help[=type]`

create command usage. The optional parameter *type* determines the formatting and output of the description.

term

display on terminal (default)

raw

output raw data for `efeudoc`

man

output nroff/troff source for `man`

lp send postscript data to `lpr`

`--version`

show version information

`-q` landscape

`-n count`

number of formatting runs

`-r res`

resolution

*src* source

*ps* target

**COPYRIGHT**

Copyright (C) 1993, 2001 Erich Frühstück