

[*perlsupport.txt*](#)

Perl Support

Jan 13 2006

Perl Support

[*perl-support*](#) [*perlsupport*](#)

Plugin version 2.9
 for [Vim version 6.0](#) and above
 Fritz Mehner <mehner@fh-swf.de>

perl-support.vim implements a Perl-IDE for Vim/gVim. It is written to considerably speed up writing code in a consistent style. This is done by inserting complete statements, comments, idioms, code snippets, templates, comments and POD documentation. Reading perldoc is integrated. Syntax checking, running a script, starting a debugger and a profiler can be done with a keystroke. There are many additional hints and options which can improve speed and comfort when writing Perl.

| | | |
|--------|---------------------------------|--|
| 1. | Usage with GUI | perlsupport-usage-gvim |
| 1.1 | Menu 'Comments' | perlsupport-comments |
| 1.1.1 | Append aligned comments | perlsupport-aligned-comm |
| 1.1.2 | Code to comment | perlsupport-code-to-comm |
| 1.1.3 | Comment to code | perlsupport-comm-to-code |
| 1.1.4 | Comment templates | perlsupport-comm-templates |
| 1.1.5 | KEYWORD + comment | perlsupport-comm-keywords |
| 1.2 | Menu 'Statements' | perlsupport-statements |
| 1.2.1 | Normal mode, insert mode | perlsupport-stat-norm-ins |
| 1.2.2 | Visual mode | perlsupport-stat-visual |
| 1.2.3 | Code snippets | perlsupport-stat-snippets |
| 1.3 | Menu 'Idioms' | perlsupport-idioms |
| 1.3.1 | Stub subroutine | perlsupport-stub-sub |
| 1.3.2 | Opening files | perlsupport-open-files |
| 1.4 | Menu 'Regex' | perlsupport-regex |
| 1.5 | Menu 'CharCls' | perlsupport-charcls |
| 1.6 | Menu 'File-Tests' | perlsupport-filetests |
| 1.7 | Menu 'Spec-Var' | perlsupport-specvar |
| 1.8 | Menu 'POD' | perlsupport-pod |
| 1.8.1 | Menu 'invisible POD' | perlsupport-pod-invisible |
| 1.8.2 | Run podchecker | perlsupport-podchecker |
| 1.9 | Menu 'Run' | perlsupport-run |
| 1.9.1 | Run script | perlsupport-run-script |
| 1.9.2 | Check syntax | perlsupport-syntax-check |
| 1.9.3 | Command line arguments | perlsupport-cmdline-args |
| 1.9.4 | Debug | perlsupport-run-debug |
| 1.9.5 | Read perldoc | perlsupport-perldoc |
| 1.9.6 | Generate Perl module list | perlsupport-module-list-generation |
| 1.9.7 | Show installed Perl modules | perlsupport-module-list |
| 1.9.8 | Run perltidy | perlsupport-perltidy |
| 1.9.9 | Run SmallProf | perlsupport-profile |
| 1.9.10 | Run perlcritic | perlsupport-perlcritic |
| 1.9.11 | Save buffer with timestamp | perlsupport-timestamp |
| 1.9.12 | Hardcopy | perlsupport-hardcopy |
| 1.9.13 | Xterm size | perlsupport-xterm |
| 1.9.14 | Change Output Destination | perlsupport-output |
| 1.10 | Help | perlsupport-help |
| 2. | Usage without GUI | perlsupport-usage-vim |
| 3. | Hotkeys | perlsupport-hotkeys |
| 4. | Customization and configuration | perlsupport-customization |
| 4.1 | Files | perlsupport-custom-files |
| 4.2 | Global variables | perlsupport-custom-variables |
| 4.3 | The root menu | perlsupport-custom-root |
| 4.4 | Navigate through PODs | perlsupport-custom-navigate |
| 4.5 | Tabulator width | perlsupport-custom-tab |
| 5. | Template files and tags | perlsupport-tempfiles |
| 5.1 | Template files | perlsupport-tempfiles |
| 5.2 | Tags | perlsupport-tags |
| 6. | Perl Dictionary | perlsupport-dictionary |
| 7. | MS-Windows particularities | perlsupport-windows |
| 8. | Troubleshooting | perlsupport-troubleshooting |
| 9. | Release Notes | perlsupport-release-notes |
| 10. | Credits | perlsupport-credits |

How to add this help file to vim's help [|add-local-help|](#)

=====

1. USAGE WITH GUI (gVim)

[*perlsupport-usage-gvim*](#)

If the root menu 'Perl' is not visible call it with the entry "Load Perl Support" from the standard Tools-menu.
 The entry "Unload Perl Support" can be used to unload the Perl root menu.
 See also [|perlsupport-custom-root|](#).

1.1 MENU 'Comments'

[*perlsupport-comments*](#)

1.1.1 APPEND ALIGNED COMMENTS TO CONSECUTIVE LINES

[*perlsupport-aligned-comm*](#)

In NORMAL MODE the menu entry
 'Line End Comm.'
 will append a comment to the current line.

In VISUAL MODE this entry will append aligned comments to all marked lines.
 Marking the 3 lines

```
my $x11      = 11;
my $x1111    = 1111;

my $x11111111 = 11111111;
```

and choosing 'Line End Comm.' will yield

```
my $x11      = 11;          # |
my $x1111    = 1111;        #
my $x11111111 = 11111111;   #
```

The cursor position above is marked by '|' . Empty lines will be ignored.

The default starting column is 49 (= (8*6 or 4*12 or 2*24) + 1). This can be changed by setting a global variable in the file .vimrc , e.g. :

```
let g:Perl_LineEndCommColDefault = 45
```

The starting column can also be set by the menu entry 'Comments->Set End Comm. Col.' . Just position the cursor in an arbitrary column (column number is shown in the Vim status line) and choose this menu entry. This setting is buffer related.

1.1.2 CODE TO COMMENT

[*perlsupport-code-to-comm*](#)

A marked block will be changed into

```
#my $x11      = 11;
#my $x1111    = 1111;
#my $x11111111 = 11111111;
```

The menu entry works also for a single line. A single line needs not to be marked.

1.1.3 COMMENT TO CODE

[*perlsupport-comm-to-code*](#)

A marked block

```
#my $x1      = 1;
my $x11      = 11;
    my $x1111 = 1111;
    # my $x11111111 = 11111111;
```

will be changed into

```
my $x1      = 1;
my $x11      = 11;
    my $x1111 = 1111;
    my $x11111111 = 11111111;
```

The menu entry works also for a single line. A single line needs not to be marked.

1.1.4 COMMENT TEMPLATES

perlsupport-comm-templates

Frame comments, function descriptions and file header comments are read as templates from the appropriate files (see [perlsupport-tempfiles](#)).

1.1.5 KEYWORD+comment

perlsupport-comm-keywords

Preliminary line end comments to document (and find again) places where work will be resumed shortly, like

```
# :TODO:12.05.2004:Mn: <your comment>
```

Usually not meant for the final documentation.

1.2 MENU 'Statements'

perlsupport-statements

1.2.1 NORMAL MODE, INSERT MODE.

perlsupport-stat-norm-ins

An empty statement will be inserted and properly indented. The entry 'if{' will insert an if-statement:

```
if ( ) {
}
```

The opening curly brace is on the same line as the keyword introducing it. To place the brace on a new line set the appropriate global variable in .vimrc :

```
let g:Perl_BraceOnNewLine = "yes"
```

The default values is "no". This flag applies to all structural blocks, including subroutines.

1.2.2 VISUAL MODE.

perlsupport-stat-visual

The highlighted area

```
xxxxx
xxxxx
```

can be surrounded by one of the following statements ('|' marks the cursor position after insertion):

| | |
|--|---|
| <pre>do { xxxxx xxxxx } while (); # ----- end do-while -----</pre> | |
| <pre>for (; ;) { xxxxx xxxxx }</pre> | <pre>foreach () { xxxxx xxxxx } # ----- end foreach -----</pre> |
| <pre>if () { xxxxx xxxxx }</pre> | <pre>if () { xxxxx xxxxx } else { }</pre> |

| | |
|---|--|
| <pre> unless () { xxxxxx xxxxxx } </pre> | <pre> unless () { xxxxxx xxxxxx } else { } </pre> |
| <pre> until () { xxxxxx xxxxxx } </pre> | <pre> while () { xxxxxx xxxxxx } # ----- end while ----- </pre> |
| <pre> { xxxxxx xxxxxx } </pre> | |

The whole statement will be indented after insertion.

1.2.3 CODE SNIPPETS

perlsupport-stat-snippets

Code snippets are pieces of code which are kept in separate files in a special directory. File names are used to identify the snippets.

The snippet directory has to be created by the user

(\$HOME/.vim/codesnippets-perl is the default).

Snippets are managed with the 3 entries

```

Perl -> Statements -> read  code snippet
Perl -> Statements -> write code snippet
Perl -> Statements -> edit  code snippet

```

from the Statements submenu.

Creating a new snippet:

When nothing is marked, "write code snippet" will write the whole buffer to a snippet file, otherwise the marked area will be written to a file.

Insert a snippet:

Select the appropriate file from the snippet directory ("read code snippet"). The inserted lines will be indented.

Indentation / no indentation

Code snippets are normally indented after insertion. To suppress indentation add the file extension "ni" or "noindent" to the snippet file name, e.g.

```
parameter_handling.pl.noindent
```

1.3 MENU 'Idioms'

perlsupport-idioms

1.3.1 STUB SUBROUTINE

perlsupport-stub-sub

In normal mode the entry '**subroutine**' asks for a subroutine name and creates a stub subroutine with one parameter:

```

sub xxx {
  my ($par1) = @_;

  return ;
} # ----- end of subroutine xxx -----

```

In visual mode with a few lines marked this entry will enclose these lines in a subroutine and generate a call to this subroutine. The lines

```
print "x11 = $x11\n";
print "x22 = $x22\n";
print "x33 = $x33\n";
```

will be changed into

```
abc();

sub abc {
  my ($par1) = @_ ;
  print "x11 = $x11\n";
  print "x22 = $x22\n";
  print "x33 = $x33\n";
  return ;
} # ----- end of subroutine abc -----
```

The further adaption is left to the user.

1.3.2 OPENING FILES

perlsupport-open-files

All declarations beginning with **'my'** and the multiline statements (subroutine, open input file / output file / pipe) will be inserted below the current line. Everything else will be inserted at the cursor position.

The entries 'open input file', 'open output file' and 'open pipe' ask for the name of a file handle. After the insertion of the statements

```
my $INFILE_file_name = '';      # input file name

open my $INFILE, '<', $INFILE_file_name
    or die "$0 : failed to open input file $INFILE_file_name : $!\n";

close $INFILE
    or warn "$0 : failed to close input file $INFILE_file_name : $!\n";
```

the 'Idioms'-menu will be extended with an entry for the new handle:

```
<$INFILE>
```

After the insertion of the statements for an output file the 'Idioms'-menu will be extended by a new print statement for the new handle, e.g.

```
print $OUTFILE "\n";
```

Multi-line inserts will be indented after insertion.

1.4 MENU 'Regex'

perlsupport-regex

In NORMAL and INSERT MODE the shown items will be inserted at the cursor position.

In VISUAL MODE the following entries and all entries from the 'extended Regex' submenu will surround the marked area **'xxx'** :

```
( )      :      (xxx)
(|)      :      (xxx|)
[]       :      [xxx]
{}       :      {xxx}
{,}      :      {xxx,}
```

1.5 MENU 'CharCls'

perlsupport-charcls

1.6 MENU 'File-Tests'

perlsupport-filetests

1.7 MENU 'Spec-Var'

perlsupport-specvar

The entries from these menus will be inserted at the cursor position.

1.8 MENU 'POD' *perlsupport-pod*

Most entries insert POD commands below the cursor position, e.g.

```
=pod
```

```
=cut # back to Perl
```

The entries 'POD->html', 'POD->man', 'POD->text' call the appropriate translator which will generate the desired document from the current buffer.

The plugin taglist.vim (Yegappan Lakshmanan) can be expanded for POD navigation. See [|perlsupport-custom-navigate|](#).

1.8.1 MENU 'invisible POD' *perlsupport-pod-invisible*

These menu entries insert "invisible" POD sections as suggested in Damian Conway's book "Perl Best Practices", e.g.

```
=for Improvement: <keyword>
<single paragraph>
```

```
=cut # back to Perl
```

The text in the single paragraph will be ignored by the compiler and by a POD formatter. This can be used to embed extended pieces of internal documentation. For the paragraph to be invisible there must not be an empty line between =for ... and the following paragraph.

The four formatter names "Improvement", "Optimization", "Rationale", and "Workaround" are just suggestions. You can choose additional ones. The <keyword> is a short explanation which makes navigation with taglist easier. See [|perlsupport-custom-navigate|](#). Please **note** the colon after the "formatter name". It is needed for parsing this construct.

1.8.2 RUN PODCHECKER *perlsupport-podchecker*

The current buffer will be run through the application podchecker to check the syntax of the embedded POD or of a POD format documentation file (see podchecker(1) and Pod::Checker).

Podchecker always reports errors. Printing warnings can be turned on and off with the options -warnings/-nowarning . The default is to print warnings. To turn the warnings of put the following line in the file .vimrc :

```
let g:Perl_PodcheckerWarnings = "no"
```

1.9 MENU 'Run' *perlsupport-run*

1.9.1 RUN SCRIPT *perlsupport-run-script*

Running a Perl-script opens a window with name "Perl-Output" to display the script output. The buffer and its content will disappear when closing the window. If this window remains open it will be used for the next runs. If the script doesn't produce shell output the output window will not be opened (but you will see a message). There is no file behind the window Perl-Output but the content can be saved with a 'save as'.

1.9.2 CHECK SYNTAX *perlsupport-syntax-check*

The script is run as "perl -wc xxx.pl" with most warnings enabled to check the syntax.

The Perl script efm_perl.pl (from the VIM standard distribution with a minor improvement) is needed for checking the syntax of a file with a file name or a pathname containing blanks. Due to a weakness in the file name representation in the Perl output, messages have to be filtered in order to be processed correctly by the VIM quickfix system.
This script has to be executable under UNIX.

1.9.3 COMMAND LINE ARGUMENTS

perlsupport-cmdline-args

The entry 'command line arguments' calls an input dialog which asks for command line arguments. These arguments are forwarded to the script which is run by the 'run' entry. The arguments are kept until you change them. The arguments can contain pipes and redirections, e.g.

```
" infile.txt | sort -rn > result.txt"
```

The arguments belong to the current buffer (that is, each buffer can have its own arguments). The input dialog has a history.

If the buffer gets a new name with "save as" the arguments will now belong to the buffer with the new name.

1.9.4 DEBUG

perlsupport-run-debug

Start a debugger with the menu entry Run->debug or with hotkey F9. One of three debuggers can be started. The preference can be set with the variable g:Perl_Debugger (possible values: 'perl', 'ptkdb', 'ddd'). The default is 'perl'.

(1) perl

The script will be run as 'perl -d my-script.pl my-arguments' in an xterm.

(2) ptkdb

The debugger ptkdb will be started as an independent process. ptkdb is a Perl debugger using a Tk GUI. The module Devel::ptkdb and the Tk tool kit have to be installed.

(3) ddd

The data display debugger ddd is a graphical front end for GDB. It will be started as an independent process.

The debugger ddd is not available under MS-Windows.

The debugger starts in an separate xterm or is a separate GUI-application (e.g. ddd). This feature is therefore not available for Vim.

1.9.5 READ PERLDOC

perlsupport-perldoc

If a (key-)word is under the cursor the entry 'read perldoc' tries to look up the Perl documentation for this word using perldoc. If a whitespace is under the cursor the user will be asked for a keyword. Search order:

1. modules *-<-+
2. functions | |
3. FAQs +-->+

This sequence is organized as a ring. If you search for the same item in the module description (if any) again the plugin tries to look up a function description, then a FAQ and then the module description again.

On a UNIX platform errors produced by perldoc will be suppressed (a few module descriptions have POD errors!).

1.9.6 GENERATE PERL MODULE LIST

perlsupport-module-list-generation

The entry 'Run -> generate Perl module list' generates a text file (default: \$HOME/.vim/plugin/perl-modules.list) which contains one line for each Perl module installed on your machine:

```
...
Fcntl      (1.05)  - load the C Fcntl.h defines
File::Basename (2.72)  - split a pathname into pieces
File::CheckTree (4.3)   - run many filetest checks on a tree
File::Compare (1.1003) - Compare files or filehandles
File::Copy    (2.07)  - Copy files or filehandles
...
```

The module list is generated by the Perl script `$HOME/.vim/plugin/pmdesc3.pl` (based on `pmdesc2` by Aristotle, see [|perlsupport-credit|](#)). This script has to be executable under UNIX. The generation may take a while. `pmdesc3.pl` has a POD included; see file `doc/pmdesc3.text` .

1.9.7 SHOW INSTALLED PERL MODULES

perlsupport-module-list

The entry 'Run -> show installed Perl modules' loads the module list in a new window. The full documentation for that module can be opened in a `perldoc` help window using the hot keys `<Shift-F1>`, `\h` or `\rp` . Looking up help with `Shift-F1` works also in the `perldoc` help window. Vim (without GUI): only `\h` and `\rp` are working.

1.9.8 RUN PERLTIDY

perlsupport-perltidy

The buffer can be formatted with `perltidy`. If nothing is marked the whole buffer will be formatted. If a region is marked only this region will be formatted.

`Perltidy` has a lot of options. It is recommended to use a `.perltidyc` initialization file to define the preferred style (see 'man 1 perltidy'). See also [|perlsupport-troubleshooting|](#).

1.9.9 RUN SMALLPROF

perlsupport-profile

Profile the script in the current buffer using the profiler `Devel::SmallProf` (the module has to be installed, of course). The results will go to the file `smallprof.out` in the current directory. This file will be opened automatically. `Devel::SmallProf` (version 2.00_03) is controlled by 4 variables (default values shown here):

```
$DB::drop_zeros = 0;           # Do not show lines which were never called: 1
$DB::grep_format = 0;          # Output on a format similar to grep : 1
$DB::profile     = 1;           # Turn off profiling for a time: 0
%DB::packages    = ('main'=>1); # Only profile code in a certain package.
```

These variables can be put in a file called `.smallprof` in the current directory. See the module documentation for more information.

1.9.10 RUN PERLCRITIC

perlsupport-perlcrritic

"perlcrritic" is a Perl source code analyzer. It is the executable front-end to the `Perl::Critic` engine, which attempts to identify awkward, hard to read, error-prone, or unconventional constructs in your code. Most of the rules are based on Damian Conway's book *Perl Best Practices* (PBP). When run from the menu the current buffer will be saved and run through `perlcrritic`. The reported violations will be displayed in a separate quickfix error window. There are three formats:

FORMAT 1 (one line; short)

```
file | line | column | brief description
```

Example:

```
test.pl|23 col 1| Code before warnings are enabled
```

FORMAT 2 (one line; default)

```
file | line | column | brief description | explanation or page number in PBP
```

Example:

```
test.pl|23 col 1| Code before warnings are enabled. See page 431 of PBP
```

FORMAT 3 (multi-line)

```
file | line | column | brief description
policy
explanation or page number in PBP
full diagnostic
```

Example:

```
test.pl|23 col 1| Code before warnings are enabled
```



```
[Perl::Critic::Policy::TestingAndDebugging::RequirePackageWarnings]
See page 431 of PBP.
Using warnings is probably the single most effective way to improve the
quality of your code. This policy requires that the 'use warnings'
statement must come before any other statements except 'package',
'require', and other 'use' statements. Thus, all the code in the entire
package will be affected.
```

Format 2 is the default. This can be changed by setting the variable `g:Perl_Perlritic` in `.vimrc` to another value (1-3):

```
let g:Perl_PerlriticFormat = 3
```

Format 3 may be your choice if you do not have a copy of PBP at hand. It produces a lot of output.

The default configuration file for `perlritic` is `perlriticrc`. `perlritic` will look for this file in the current directory first, and then in your home directory. See the manual for more information (`'man perlritic'` or `'perlritic -man'`) especially how to influence the policies.

Consider using maps like

```
imap <silent> <F7> <Esc>:cp<CR>
imap <silent> <F8> <Esc>:cn<CR>
```

in your `.vimrc` file to jump over the error locations and make navigation easier (see also file customization.vimrc [|perlsupport-custom-files|](#)).

1.9.11 SAVE BUFFER WITH TIMESTAMP

[*perlsupport-timestamp*](#)

Save the current buffer into a new file. The filename gets a trailing timestamp. The format is `YYYYMMDD-HHMMSS`. This feature can be used to comfortably save different profiling results but it will work with any named buffer. If you do a lot of profiling you want to add timestamps automatically. To enable this feature put the following line into `.vimrc` :

```
let g:Perl_ProfilerTimestamp = "yes"
```

The default value is "no".

1.9.12 HARDCOPY

[*perlsupport-hardcopy*](#)

Generates a PostScript file from the whole buffer or from a marked region. On a MS-Windows system a printer dialog is displayed.

The print header contains date and time for the current locale. The definition used is

```
let s:Perl_Printheader = "%<%f%h%m%< %={strftime('%x %X')}} Page %N"
```

The current locale can be overwritten by changing the language, e.g.

```
:language C
```

or by setting a global variable in the file `.vimrc` , e.g. :

```
let g:Perl_Printheader = "%<%f%h%m%< %={strftime('%x %X')}} SEITE %N"
```

See `:h printhead` and `:h strftime()` for more details.

1.9.13 XTERM SIZE

[*perlsupport-xterm*](#)

The size of the xterm used for debugging ([|perlsupport-run-debug|](#)) or for running the script (below) can be set by this menu entry. The default is 80 columns with 24 lines.

This feature is not available under MS-Windows.

1.9.14 CHANGE OUTPUT DESTINATION

[*perlsupport-output*](#)

Running a Perl script can be done in three ways:

- (1) The script can be run from the command line as usual.
- (2) The output can be directed into a window with name "Perl-Output".
The buffer and its content will disappear when the window is closed and reused otherwise.
- (3) The script can be run in an xterm.

The output method can be chosen with the menu entry 'Run->output: ...'.
This menu has three states:

```
output: VIM->buffer->xterm
output: BUFFER->xterm->vim
output: XTERM->vim->buffer
```

The first (uppercase) item shows the current method. The default is '**vim**'.
This can be changed by setting the variable g:Perl_OutputGvim to another value.
Possible values are '**vim**', '**buffer**' and '**xterm**' .

Vim (non-GUI) : The output destination can be toggled between (1) and (2)
using the hotkey \ro .

The xterm defaults can be set in .vimrc by the variable g:Perl_XtermDefaults .
The default is "-fa courier -fs 12 -geometry 80x24" :

```
font name      : -fa courier
font size      : -fs 12
terminal size  : -geometry 80x24
```

See 'xterm -help' for more options. Xterms are not available under MS-Windows.

1.10 'help'

perlsupport-help

The root menu entry '**help**' shows this plugin help in a help window. The help tags must have been generated with
:helptags ~/.vim/doc

2. USAGE WITHOUT GUI (Vim)

perlsupport-usage-vim

The frequently used constructs can be inserted with key mappings. The mappings are also described in the document perl-hot-keys.pdf (reference card).

-- Load / Unload Perl Support -----

```
\lps    Load Perl Support    (The key mappings below are defined)
\ups    Unload Perl Support   (The key mappings below are undefined.)
```

-- Comments -----

```
\cl      Line End Comment
\cf      Frame Comment
\cu      Function Description
\ch      File Header
\ckb     Keyword comment BUG
\ckt     Keyword comment TODO
\ckr     Keyword comment TRICKY
\ckw     Keyword comment WARNING
\ckn     Keyword comment New_Keyword
\cc      code to comment
\co      comment to code
\cd      Date
\ct      Date & Time
\cv      vim modeline
```

-- Statements -----

```
\ad      do { } while
\af      for { }
\ao      foreach { }
\ai      if { }
\ae      if { } else { }
\au      unless { }
```

```
\an    unless { } else { }
\at    until { }
\aw    while { }
\a{    { }
```

-- Idioms -----

```
\dm    my $;
\dy    my $ = ;
\d,    my ( $, $ );
\d1    my @;
\d2    my @ = (,,);
\d3    my %;
\d4    my % = (=>,=>,);
\d5    my $regex_ = '';
\d6    my $regex_ = qr//;
\d7    $ =~ m//
\d8    $ =~ s///
\d9    $ =~ tr///
\dp    print "...\\n";
\df    printf ("...\\n");
\ds    subroutine
\di    open input file
\do    open output file
\de    open pipe
```

-- POSIX Character Classes -----

```
\la    [:alnum:]
\lh    [:alpha:]
\li    [:ascii:]
\lb    [:blank:]
\lc    [:cntrl:]
\ld    [:digit:]
\lg    [:graph:]
\ll    [:lower:]
\lp    [:print:]
\ln    [:punct:]
\ls    [:space:]
\lu    [:upper:]
\lw    [:word:]
\lx    [:xdigit:]
```

-- Run -----

| | | | |
|-----|--------------------------------|-----|--|
| \rr | update file, run script | see | perlsupport-run-script |
| \rs | update file, check syntax | see | perlsupport-syntax-check |
| \ra | set command line argument | see | perlsupport-cmdline-args |
| \rd | start debugger | see | perlsupport-run-debug |
| \re | make script executable (*) | | |
| \rp | read perldoc | see | perlsupport-perldoc |
| \ri | show installed Perl modules | see | perlsupport-module-list |
| \rg | generation Perl module list | see | perlsupport-module-list-generation |
| \ry | run perltidy | see | perlsupport-perltidy |
| \rm | run SmallProf | see | perlsupport-profile |
| \rc | run perlcritic | see | perlsupport-perlcritic |
| \rt | save buffer with timestamp | see | perlsupport-timestamp |
| \rh | hardcopy buffer to FILENAME.ps | see | perlsupport-hardcopy |
| \rk | settings and hotkeys | | |
| \rx | set xterm size (*) | see | perlsupport-xterm |
| \ro | change output destination | see | perlsupport-output |

-- Help -----

\h read perldoc see [perlsupport-perldoc](#)

(*) Linux/UNIX only

File perl-hot-keys.pdf contains a reference card for these key mappings.
Mult-inline inserts and code snippets will be indented after insertion.

The hotkeys are defined in the file type plugin perl.vim (part of this perl-support plugin package).

3. HOTKEYS

perlsupport-hotkeys

The following hotkeys are defined in normal, visual and insert mode:

```
Shift-F1  read perldoc   (for the word under the cursor)
      F9   start a debugger
      Alt-F9 run syntax check
      Ctrl-F9 run script
      Shift-F9 set command line arguments (buffer related)
```

These hotkeys are defined in the file type plugin ~/.vim/ftplugin/perl.vim .

Note for xterm users (Vim without GUI): The function key combinations Shift-Fx, Alt-Fx and Ctrl-Fx do not work. F9 is also not working to prevent unintentional use. Use hotkeys instead [|perlsupport-usage-vim|](#).

4. CUSTOMIZATION

perlsupport-customization

4.1 FILES

perlsupport-custom-files

| | |
|-------------------------|---|
| README.perlsupport | Installation, release notes. |
| codesnippets-perl/* | Some Perl code snippets as a starting point. |
| doc/perlsupport.txt | The help file for the local online help. |
| ftplugin/perl.vim | A file type plugin. Define hotkeys, create a local dictionary for each Perl file. |
| plugin/perl-support.vim | The Perl plugin for Vim/gVim. |
| plugin/efm_perl.pl | Perl script; reformats the error messages of the Perl interpreter |
| plugin/pmdesc3.pl | Perl script; generates a list of all installed Perl modules |
| plugin/wrapper.sh | The wrapper script for the use of an xterm. |

| | |
|--|-------------------------------------|
| plugin/templates/perl-file-header | +- Perl template files |
| plugin/templates/perl-frame | (see section TEMPLATE FILES below). |
| plugin/templates/perl-function-description | |
| plugin/templates/perl-module-header | +- |

| | |
|---------------------|---|
| wordlists/perl.list | A file used as dictionary for automatic word completion. This file is referenced in the file customization.vimrc . |
|---------------------|---|

The following files and extensions are for convenience only.
perl-support.vim will work without them.

| | |
|-----------------------------|--|
| rc/customization.vimrc | Additional settings I use in .vimrc: incremental search, tabstop, hot keys, font, use of dictionaries, ... The file is commented. Append it to your .vimrc if you like. |
| rc/customization.gvimrc | Additional settings I use in .gvimrc: hot keys, mouse settings, ... The file is commented. Append it to your .gvimrc if you like. |
| rc/customization.ctags | Additional settings I use in .ctags to enable navigation through POD with the plugin taglist.vim. |
| rc/customization.perltidyrc | Additional settings I use in .perltidyrc to enable customize perltidy. |
| rc/customization.SmallProf | Settings I use in .SmallProf . |
| doc/perl-hot-keys.pdf | Reference card for the key mappings. The mappings can be used with the non-GUI Vim, |

because the menus are not available.

4.2 GLOBAL VARIABLES

perlsupport-custom-variables

Several global variables are checked by the script to customize it:

| global variable | default value | tag (see below) |
|--------------------------------|--------------------------------------|-----------------|
| g:Perl_AuthorName | " " | AUTHOR |
| g:Perl_AuthorRef | " " | AUTHORREF |
| g:Perl_Email | " " | EMAIL |
| g:Perl_Company | " " | COMPANY |
| g:Perl_Project | " " | PROJECT |
| g:Perl_CopyrightHolder | " " | COPYRIGHTHOLDER |
| g:Perl_Template_Directory | root_dir.'plugin/templates/' | |
| g:Perl_Template_File | 'perl-file-header' | |
| g:Perl_Template_Module | 'perl-module-header' | |
| g:Perl_Template_Frame | 'perl-frame' | |
| g:Perl_Template_Function | 'perl-function-description' | |
| g:Perl_CodeSnippets | root_dir.'codesnippets-perl/' | |
| g:Perl_LoadMenus | 'yes' | |
| g:Perl_Dictionary_File | ' ' | |
| g:Perl_Root | '&Perl.' | |
| g:Perl_MenuHeader | 'yes' | |
| g:Perl_PerlModuleList | root_dir.'plugin/perl-modules.list' | |
| g:Perl_PerlModuleListGenerator | root_dir.'plugin/pmdesc3.pl' | |
| g:Perl_OutputGvim | "vim" | |
| g:Perl_XtermDefaults | "-fa courier -fs 12 -geometry 80x24" | |
| g:Perl_Debugger | "perl" | |
| g:Perl_ProfilerTimestamp | "no" | |
| g:Perl_LineEndCommColDefault | 49 | |
| g:Perl_BraceOnNewLine | "no" | |
| g:Perl_PodcheckerWarnings | "yes" | |
| g:Perl_PerlriticFormat | 2 | |

The variable root_dir will automatically be set to one of the following values:

\$HOME.'/.vim/' for Linux/Unix
\$VIM.'/vimfiles/' for MS-Windows

1. group: Defines the text which will be inserted for the tags when a template is read in (see also TEMPLATE FILES AND TAGS below).

```
g:Perl_AuthorName      : author name
g:Perl_AuthorRef       : author reference (e.g. acronym)
g:Perl_Email           : email address
g:Perl_Company         : name of the company / institution
g:Perl_Project         : project
g:Perl_CopyrightHolder : the copyright holder
```

2. group: Sets the template directory and the names of the template files (see below).

3. group:
 - g:Perl_CodeSnippets : The name of the code snippet directory (see below).
 - g:Perl_LoadMenus : Load menus and mappings ("yes", "no") at startup.
 - g:Perl_Dictionary_File : Path and filename of the Perl word list used for dictionary completion (see below).
 - g:Perl_Root : The name of the root menu entry of this plugin (see below).
 - g:Perl_MenuHeader : Switch submenu titles on/off.

4. group:
 - g:Perl_PerlModuleList : The name of the Perl module list (text file, see below).
 - g:Perl_PerlModuleListGenerator : The command line which starts the module list generation.
 - g:Perl_OutputGvim : when script is running output goes to the vim command line ("vim"), to a buffer ("buffer")

```

                                or to an xterm ("xterm").
g:Perl_XtermDefaults           : the xterm defaults
g:Perl_Debugger                : the debugger called by F9 (perl, ptkdb, ddd).
g:Perl_ProfilerTimestamp       : add time stamp to the profiler buffer name
g:Perl_LineEndCommColDefault   : default starting column for line end comments
g:Perl_BraceOnNewLine          : opening brace (not) on a new line
g:Perl_PodcheckerWarnings      : podchecker warnings on/off
g:Perl_PerlcriticFormat        : perlcritic error format

```

To override the defaults add appropriate assignments to .vimrc .
 Set at least some personal details into .vimrc by overriding some defaults.
 Here the minimal personalization (my settings as an example, of course):

```

let g:Perl_AuthorName      = 'Dr.-Ing. Fritz Mehner'
let g:Perl_AuthorRef       = 'Mn'
let g:Perl_Email           = 'mehner@fh-swf.de'
let g:Perl_Company         = 'FH SÃ¼dwestfalen, Iserlohn'

```

4.3 THE ROOT MENU

perlsupport-custom-root

The variable g:Perl_Root, if set (in .vimrc or in .gvimrc), gives the name of the single gVim root menu entry in which the Perl submenus will be put.
 The default is

```
'&Perl.'
```

Note the terminating dot. A single root menu entry is appropriate if the screen is limited or several plugins are in use.

If set to "", this single root menu entry will not appear. Now all submenus are put into the gVim root menu. Nice for a Perl-only-programmer and Perl courses.

4.4 NAVIGATE THROUGH PODs

perlsupport-custom-navigate

The plugin taglist.vim (Author: Yegappan Lakshmanan) is a source code browser plugin for Vim and provides an overview of the structure of source code files and allows you to efficiently browse through source code files for different programming languages. It is based on ctags (Exuberant Ctags, Darren Hiebert, <http://ctags.sourceforge.net>).

The file rc/customization.ctags is an extension for the configuration file of ctags. If appended to \$HOME/.ctags (the initialization file for ctags) taglist can show the structure of the included POD as an table of content.

The taglist navigation window for the module Eliza.pm starts like this:

```

Eliza.pm (/home/mehner)
subroutines
  Version
  new
  _initialize
  AUTOLOAD
  command_interface
  preprocess
  postprocess
  _testquit
  _debug_memory
  transform
  parse_script_data

POD
  NAME
  SYNOPSIS
  DESCRIPTION
  INSTALLATION
  USAGE
  MAIN DATA MEMBERS
  . %decomplist
  . %reasmblist
  . %reasmblist_for_memory

```

. . .

Now you can navigate through the embedded POD with a mouse click on these entries. To enable this feature

- 1) append rc/customization.ctags to \$HOME/.ctags (or create this file)
- 2) add the following lines to \$HOME/.vimrc :

```
"
"-----
" taglist.vim : toggle the taglist window
" taglist.vim : define the title texts for Perl
"-----
noremap <silent> <F11> <Esc><Esc>:Tlist<CR>
inoremap <silent> <F11> <Esc><Esc>:Tlist<CR>

let tlist_perl_settings='perl;c:constants;l:labels;s:subroutines;d:POD'
```

- 3) restart vim/gvim

The two maps will toggle the taglist window (hotkey F11) in all editing modes. The assignment defines the headings for the Perl sections in the taglist window.

IMPORTANT : The POD contents will not be displayed if the POD comes after an `__END__` token. Ctags (current version 5.5.4) does not parse beyond this token. You may therefore want not to use `__END__` in your own modules.

4.5 Tabulator width

perlsupport-custom-tab

The Perl Style Guide recommends a tabulator setting of 4. You can force this setting for all files with file type '`perl`' by uncommenting the two lines

```
"setlocal tabstop=4
"setlocal shiftwidth=4
```

in the file type plugin `~/.vim/ftplugin/perl.vim` .

5. TEMPLATE FILES AND TAGS

perlsupport-tempfiles

5.1 TEMPLATE FILES

Four menu entries generate block comments:

```
Perl -> Comments -> Frame Comm.
Perl -> Comments -> Function Descr.
Perl -> Comments -> File Header (.pl)
Perl -> Comments -> File Header (.pm)
```

The comments which will be inserted by these menu entries are read from template files:

| MENU ENTRY | FILE | GLOBAL VARIABLE |
|----------------------|---------------------------|--------------------------|
| Frame Comment | perl-frame | g:Perl_Template_Frame |
| Function Description | perl-function-description | g:Perl_Template_Function |
| File Header (.pl) | perl-file-header | g:Perl_Template_File |
| File Header (.pm) | perl-module-header | g:Perl_Template_Module |

The template files can be written or changed by the user to fulfill special requirements (layout for a project or work group already exists, file headers / blocks have to be prepared for a documentation tool, ...). They can hold not only comments but a complete file skeleton if this is necessary. So you may want to lay out your own templates.

5.2 TAGS

perlsupport-tags

The comments in these files do not have to be personalized but they can. The text can contain the following tags which are replaced by the appropriate information when the file is read in:

```
|AUTHOR|
|DATE|
|EMAIL|
|FILENAME|
|YEAR|

|AUTHORREF|
|COMPANY|
|COPYRIGHTHOLDER|
|PROJECT|
|TIME|

|CURSOR|
```

Each tag can occur more than once. The tag `|CURSOR|` may appear only once. The tag `|CURSOR|` will be the cursor position after the block is read in. There is no need to use any of these tags, some or all can be missing.

The template files can actually be links pointing to existing templates.

6. PERL DICTIONARY

perlsupport-dictionary

The file `perl.list` contains words used as dictionary for automatic word completion. This feature is enabled by default. The default word list is

```
$HOME/.vim/wordlists/perl.list
```

If you want to use an additional list `MyPerl.List` put the following line into `.vimrc` :

```
let g:Perl_Dictionary_File = "$HOME/.vim/wordlists/perl.list",
\ "$HOME/.vim/wordlists/MyPerl.List"
```

The right side is a comma separated list of files. **Note** the point at the end of the first line (string concatenation) and the backslash in front of the second line (continuation line).

You can use Vim's dictionary feature `CTRL-X`, `CTRL-K` (and `CTRL-P`, `CTRL-N`).

7. MS-Windows PARTICULARITIES

perlsupport-windows

The plugins should go into the directory structure below the local installation directory `$HOME/.vim/` for LINUX/UNIX and `$VIM/vimfiles/` for MS-Windows.

The values of the two variables can be found from inside Vim:

```
:echo $VIM
```

or

```
:echo $HOME
```

Configuration files:

```
LINUX/UNIX : $HOME/.vimrc and $HOME/.gvimrc
MS-Windows : $VIM/_vimrc and $VIM/_gvimrc
```

8. TROUBLESHOOTING

perlsupport-troubleshooting

* Script not executable.

- Script executable from the command line ?
 - Perl installation correct ?
 - PATH variable correct ?
 - Script set executable (file access permission under LINUX/UNIX) ?
 - Script syntax correct ?
 - Necessary modules installed ?
- * Some plugin features not present (e.g. hotkeys).
- Does the file type plugin \$HOME/.vim/ftplugin/perl.vim exist ?
- * Some hotkeys do not work.
- The hotkeys might be in use by your graphical desktop environment.
Under KDE Ctrl-F9 is the hotkey which let you switch to the 9. desktop.
The key settings can usually be redefined.
- * perltidy not running / messing up my file
- Unix/Linux: you have had a proper installation of perltidy, but now it does not work or messes up your file.
The start script '/usr/bin/perltidy' needs the module 'Perl::Tidy.pm'. Most likely you have updated Perl and the module can now longer be found. The easiest remedy is to reinstall perltidy. Check the installation with the command "perltidy -v" from the command line.

9. RELEASE NOTES

perlsupport-release-notes

See file README.perlsupport .

10. CREDITS

perlsupport-credits

David Fishburn <fishburn@ianymwhere.com> for the implementation of the single root menu and several suggestions for improving the customization and the documentation.

Ryan Hennig <hennig@amazon.com> improved the install script.

Aristotle, <http://qs321.pair.com/~monkads/> is the author of the script pmdesc2 which is the base of the script pmdesc3.pl.

David Fishburn contributed changes for the Windows platform and suggested to not let enter snippets and templates the list of alternate files.

The code snippet files pod-template-application.pl and pod-template-module.pl are taken from Damian Conway's book "Perl Best Practices".

vim:tw=78:noet:ts=2:ft=help:norl: