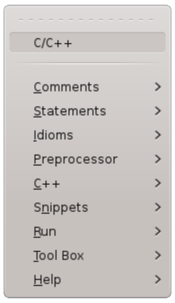
c.vim – C/C++ IDE – Screen Shots



C/C++ plugin root menu (version 6.1)

needs Vim version 7+

screen shots: gVim + plugins as C/C++ - IDE

Read the c.vim **help file** online

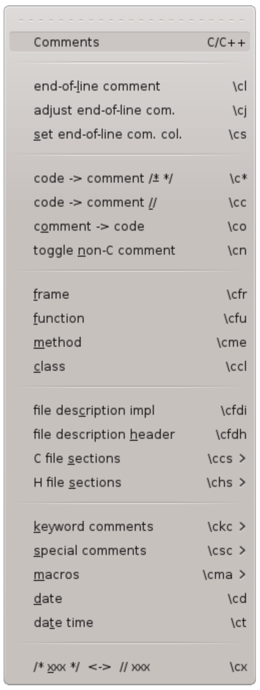
The key mappings of this plugin (PDF)

Plugin featured in the The Geek Stuff tutorial
Make Vim as Your C/C++ IDE Using c.vim Plugin

Similar plugins:
AWK-IDE
Bash-IDE
Git Support
Perl-IDE
Latex-IDE
Lua-IDE
Vim Script IDE

Submenus (1. level) | Submenus (2. level) | Generated Code

Menu *Comments*



Menu *Comments* : different types of comments, file section headers, commenting and uncommenting of marked areas

Many menu entries generate comments or commented code. The style of the comments can be toggled between C-style (/* ... */) and C++-style (// ...).
6 menu entries generate block comments. These comments are read from **template files**. These files can be written or changed by the user to fulfill special requirements (layout for a project or workgroup already exists, file headers / blocks have to be prepared for a documentation tool, ...).
The entry **C/C++-file header** generates a complete file header. File name and the date are looked up by the editor. The other information (author name, sign, ...) are taken from the configuration settings in c.vim.

```
/*
 * =====================================================================================
 *
 *       Filename:  test1.c
 *
 *    Description:
 *
 *        Version:  1.0
 *        Created:  20.08.2012 19:08:35
 *       Revision:  none
 *       Compiler:  gcc
 *
 *         Author:  Dr. Fritz Mehner (fgm), mehner.fritz@fh-swf.de
 *   Organization:  FH Südwestfalen, Iserlohn
 *
 * =====================================================================================
 */
```

This header is generated from the template below.

```
== Comments.file description impl == map:cfdi, shortcut:c, start, noindent ==
/*
 * =====================================================================================
 *
 *       Filename:  |FILENAME|
 *
 *    Description:  <CURSOR>
 *
 *        Version:  1.0
 *        Created:  |DATE| |TIME|
 *       Revision:  none
 *       Compiler:  gcc
 *
 *         Author:  |AUTHOR| (|AUTHORREF|), |EMAIL|
 *   Organization:  |ORGANIZATION|
 *
 * =====================================================================================
 */
```

The entry **H-File Sections -> All Sections, C** generates file section header for sections often used in header files.
C style

```
/* #####   HEADER FILE INCLUDES   ########################################## */

/* #####   EXPORTED MACROS   ############################################### */

/* #####   EXPORTED DATA TYPES   ########################################### */

/* #####   EXPORTED TYPE DEFINITIONS   ##################################### */

/* #####   EXPORTED VARIABLES   ############################################ */

/* #####   EXPORTED FUNCTION DECLARATIONS   ################################ */
```

C++ style

```
// #####   HEADER FILE INCLUDES   ##########################################

// #####   EXPORTED MACROS   ###############################################

// #####   EXPORTED DATA TYPES   ###########################################

// #####   EXPORTED TYPE DEFINITIONS   #####################################

// #####   EXPORTED VARIABLES   ############################################

// #####   EXPORTED FUNCTION DECLARATIONS   ################################
```

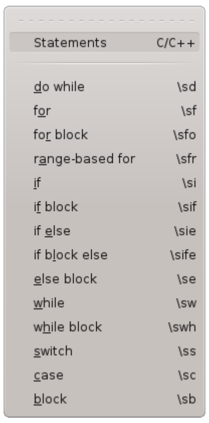
The entry **KEYWORD+Comm. -> //:BUG:** generates a special line end comment for commenting a bug. These comments are easily located by their key words (e.g. **:BUG:**). Date and author name are inserted by the editor.

Submenus (1. level)	Submenus (2. level)	Generated Code

```
// :BUG:04.03.2003:Mn:
```

These comments are not for the final version of a program, of course.

Menu *Statements*



Menu *Statements* : Insert C statements

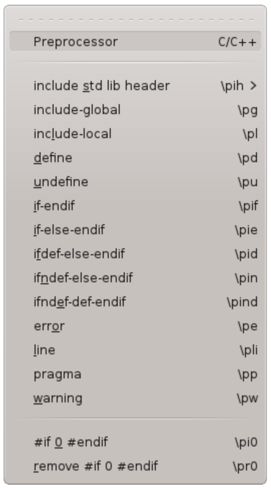
The entry **do { } while** generates an empty do-while-loop with a comment at the end. The cursor will be positioned between the brackets.

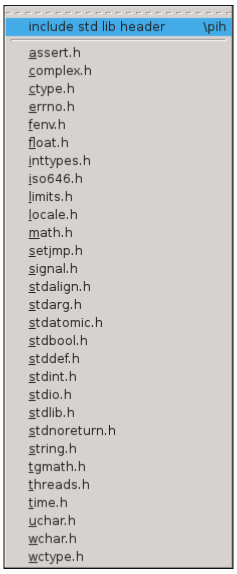
```
do
{

}
while (  );       // -----  end do-while  -----
```

The entry **switch** generates a switch-staement with 4 cases + default :

```
switch (  )
{

  case :
    break;

  case :
    break;

  case :
    break;

  case :
    break;

  default:
    break;
}       // -----  end switch  -----
```

Menu *Preprocessor*





Menu *Preprocessor* : Insert preprocessor statements

The menu entry **#ifndef #def #endif** generates an empty include guard. The macro name is suggested according to the file name.

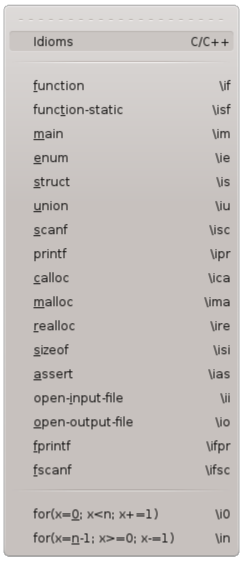
```
#ifndef  SHARED_MEM_1_INC
#define  SHARED_MEM_1_INC

#endif   /* ----- #ifndef SHARED_MEM_1_INC  ----- */
```

The menu entry **#if 0 #endif** inserts the lines

```
#if  0     /* ----- #if 0 : If0Label_1 ----- */

#endif     /* ----- #if 0 : If0Label_1 ----- */
```

In visual mode the marked block of lines will be surrounded by these lines. This is usually done to temporarily block out some code. The label names like **If0Label_1** are automatically inserted into the comments. The trailing numbers are automatically incremented.
The menu entry **remove #if #endif** removes such a construct if the cursor is in the middle of such a section or on one of the two enclosing lines. Nested constructs will be untouched.

Menu *Idioms*



Menu *Idioms* : Insert frequently used statements.

The entry **function** generates an empty function with a specified name (dialog). The curser will be positioned on the key word void.

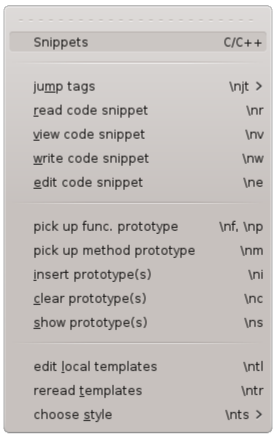
```
void
func33 (  )
{
  return ;
}       // ----------  end of function func33  ----------
```

The entry **open input file** generates the following statements which opens a file. The name of the file pointer can be specified in a dialog.

```
  FILE  *infile;                /* input-file pointer */
  char  *infile_file_name = ""; /* input-file name    */

  infile  = fopen( infile_file_name, "r" );
  if ( infile == NULL )
  {
    fprintf (stderr, "couldn't open file '%s'; %s\n",
        infile_file_name, strerror(errno) );
    exit (EXIT_FAILURE);
  }


  if( fclose(infile) == EOF )    /* close input file */
  {
    fprintf (stderr, "couldn't close file '%s'; %s\n",
        infile_file_name, strerror(errno) );
    exit (EXIT_FAILURE);
  }
```

Submenus (1. level)	Submenus (2. level)	Generated Code

Menu *Snippets*



Menu *Snippets* : Code snippets and prototypes

Read, write and edit your own code snippets in separate files in a separate snippet directory.

Pick up function prototype(s) from one or more lines. Insert the prototype(s) elsewhere.
Pick up prototypes:

Mark the following lines and choose pick up prototype :

```
double
f3_sub1 (                // comment  comment comment
/* */       double r1,   /**/ // the first /* ssssss */
            double r2 )  // the second
```

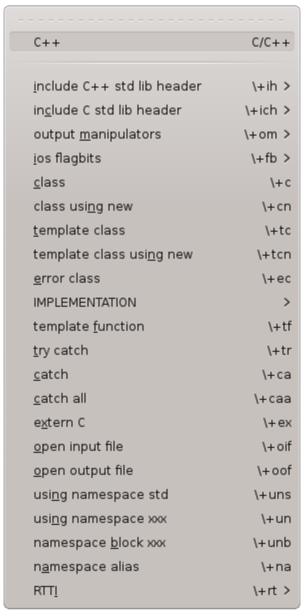
Now mark these lines and choose pick up prototype again:

```
int f4_sub1  (  int n1, /* multi
                           line
                           comment */

                int n2, int n3 /* comment3 */ )
```

The following lines will be inserted by insert prototype(s) :

```
double f3_sub1 ( double r1, double r2 );
int f4_sub1 ( int n1, int n2, int n3 );
```

Menu *C++*



Menu *C++* : empty classes, output manipulators and flagbits, open files, try-catch-blocks, ...

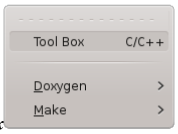
The entry **class** generates an empty class declaration. The name of the class (e.g. Cls) can be specified in a dialog.

```
class Cls
{
  public:

    // ====================  LIFECYCLE   =================================

    Cls (); // constructor

    // ====================  OPERATORS   =================================

    // ====================  OPERATIONS  =================================

    // ====================  ACCESS      =================================

    // ====================  INQUIRY     =================================

  protected:

  private:

}; // ----------  end of class  Cls  ----------
```

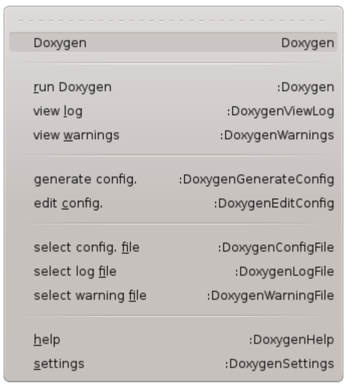
The entry **operator <<** generates an empty friend-function which overloads the standard I/O-operator. The name of the class (e.g. Cls) can be specified in a dialog.

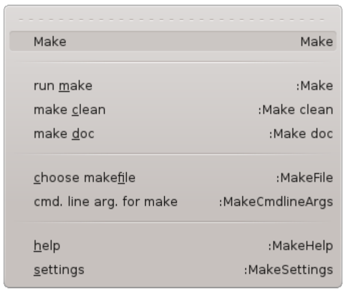
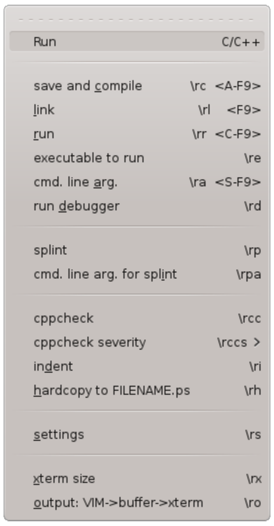
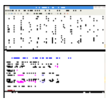
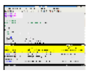
```
ostream &
operator << (ostream & os, const Cls & obj )
{
  os << obj. ;
  return os;
}       // -----  class Cls : end of friend function operator <<  -----
```

Menu *Tool Box*



Menu *Tool Box* : use configurable tool boxes


Doxygen Doxygen
run Doxygen :Doxygen
view log :DoxygenViewLog
view warnings :DoxygenWarnings
generate config. :DoxygenGenerateConfig
edit config. :DoxygenEditConfig
select config. file :DoxygenConfigFile
select log file :DoxygenLogFile
select warning file :DoxygenWarningFile
help :DoxygenHelp
settings :DoxygenSettings
control doxygen

Submenus (1. level)	Submenus (2. level)	Generated Code
		 Make Make run make :Make make clean :Make clean make doc :Make doc choose makefile :MakeFile cmd. line arg. for make :MakeCmdlineArgs help :MakeHelp settings :MakeSettings control **make(1)**
Menu *Run*  Run C/C++ save and compile \rc <A-F9> link \rl <F9> run \rr <C-F9> executable to run \re cmd. line arg. \ra <S-F9> run debugger \rd splint \rp cmd. line arg. for splint \rpa cppcheck \rcc cppcheck severity \rccs > indent \ri hardcopy to FILENAME.ps \rh settings \rs xterm size \rx output: VIM->buffer->xterm \ro		**Menu *Run* :** **compile, link, run a program** **set command line arguments,** **make hardcopy, redirect output, run splint** Tear off menus, tag list source code browsing, function preview and error list buffer.  Output directed into a Vim buffer (200x200 matrix, line length 1400+ characters) .  Run current buffer through the source code analyser splint and open an error window. Run current buffer through the source code analyser CodeCheck (Trademark of Abraxas Software, Inc.) and open an error window.

TOP

back to Sourceforge

Page created: October 29 2013 / Mail to: F.Mehner