

CodeBlocks

Manual

Version 1.1

Thanks to the CodeBlocks team:

Anders F. Björklund (afb), Biplab Kumar Modak (biplab), Bartomiej wiecki (byo), Paul A. Jimenez (ceniza), Koa Chong Gee (cyberkoa), Daniel Orb (daniel2000), Lieven de Cock (killerbot), Yiannis Mandravellos (mandrav), Mispunt (mispunt), Martin Halle (morten-macfly), Jens Lody (jens), Jerome Antoine (dje), Damien Moore (dmoore), Pecan Heber (pecan), Ricardo Garcia (rickg22), Thomas Denk (thomasdenk), tiwag (tiwag)

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation.

1 CodeBlocks Project Management

The instructions for [chapter 3](#) on page 53 and ?? on page ?? are official documentations of the CodeBlocks Wiki site and available in english only.

The below illustration shows the design of the CodeBlocks user interface.

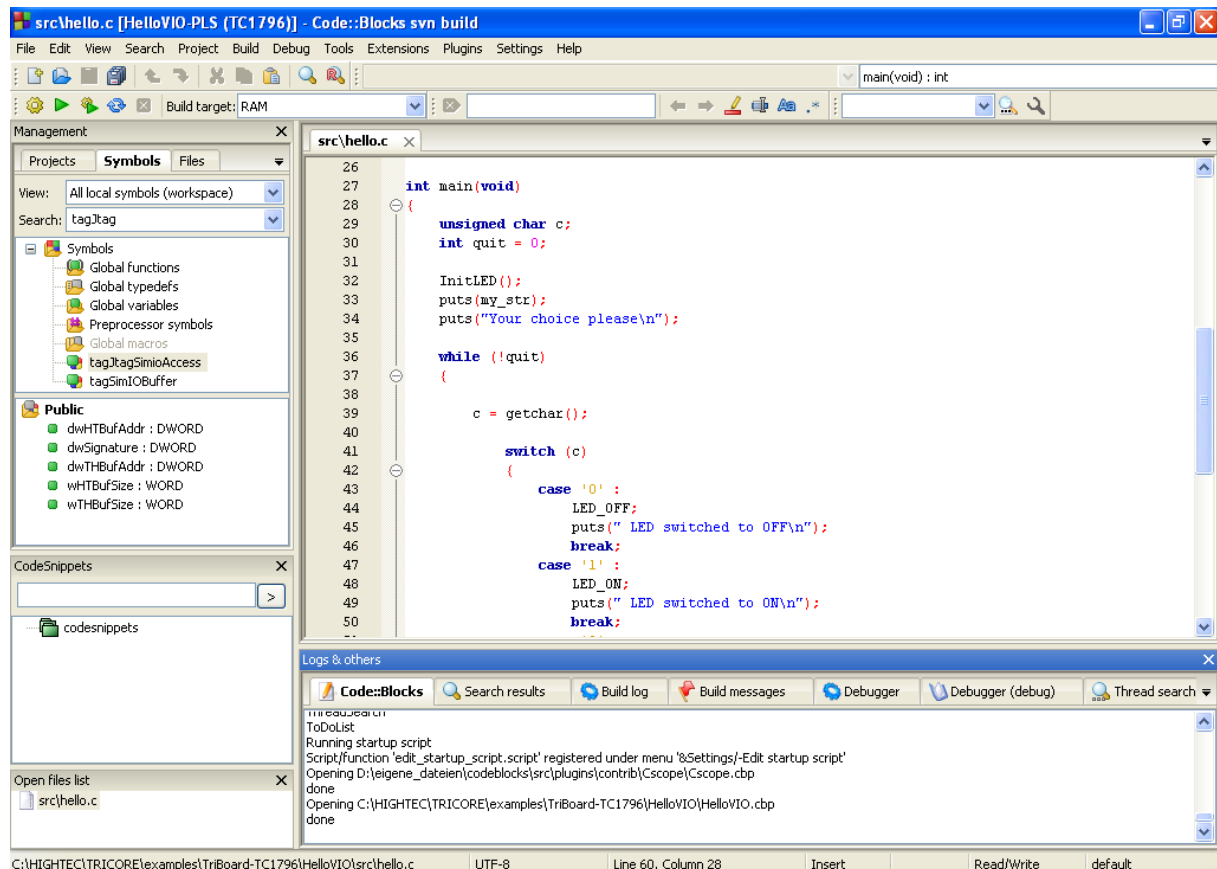


Figure 1.1: IDE CodeBlocks

Management This window contains the interface 'Projects' which will in the following text be referred to as the project view. This view shows all the projects opened in CodeBlocks at a certain time. The 'Symbols' tab of the Management window shows symbols, variables etc..

Editor In the above illustration, a source named `hello.c` is opened with syntax highlighting in the editor.

Open files list shows a list of all files opened in the editor, in this example: `hello.c`.

CodeSnippets can be displayed via the menu 'View' → 'CodeSnippets'. Here you can manage text modules, links to files and links to urls.

Logs & others. This window is used for outputting search results, log messages of a compiler etc..

The status bar gives an overview of the following settings:

- Absolute path of an opened file in the editor.
- The editor uses the default character encoding of your host operating system. This setting will be displayed with **default**.
- Row and column number of the current cursor position in the editor.
- The configured keyboard mode for inserting text (Insert or Overwrite).
- Current state of a file. A modified file will be marked with **Modified** otherwise this entry is empty.
- The permission of a file. A file with read only settings will display **Read only** in the status bar. In the window 'Open files list' these files will be emphasised with a lock as icon overlay.

Note:

In the active editor the user can select the context menu properties. In the appearing dialog in the tab 'General' the option 'File is read-only' can be selected. This option will result in a read-only access of the corresponding file within CodeBlocks, but the original read and write attributes of the file on the filesystem are not modified.

- If you start CodeBlocks with the command line option `--personality=<profile>` then the status bar will show the currently used profile, otherwise **default** will be shown. The settings of CodeBlocks are stored in the corresponding configuration file `<personality>.conf`.

CodeBlocks offers a very flexible and comprehensive project management. The following text will address only some of the features of the project management.

1.1 Project View

In CodeBlocks, the sources and the settings for the build process are stored in a project file `<name>.cbp`. C/C++ sources and the corresponding header files are the typical components of a project. The easiest way to create a new project is executing the command 'File' → 'Project' and selecting a wizard. Then you can add files to the project via the context menu 'Add files' in the Management window.

CodeBlocks governs the project files in categories according to their file extensions. These are the preset categories:

Sources includes source files with the extensions `*.c;*.cpp;.`

ASM Sources includes source files with the extensions `*.s;*.S;*.ss;*.asm`.

Headers includes, among others, files with the extension `*.h;.`

Resources includes files for layout descriptions for wxWidgets windows with the extensions `*.res;*.xrc;.` These file types are shown in the 'Resources' tab of the Management window.

The settings for types and categories of files can be adjusted via the context menu 'Project tree' → 'Edit file types & categories'. Here you can also define custom categories for file extensions of your own. For example, if you wish to list linker scripts with the *.ld extension in a category called Linkerscript, you only have to create the new category.

Note:

If you deactivate 'Project tree' → 'Categorize by file types' in the context menu, the category display will be switched off, and the files will be listed as they are stored in the file system.

1.2 Notes for Projects

In CodeBlocks, so-called notes can be stored for a project. These notes should contain short descriptions or hints for the corresponding project. By displaying this information during the opening of a project, other users are provided with a quick survey of the project. The display of notes can be switched on or off in the Notes tab of the Properties of a project.

1.3 Project Templates

CodeBlocks is supplied with a variety of project templates which are displayed when creating a new project. However, it is also possible to store custom templates for collecting your own specifications for compiler switches, the optimisation to be used, machine-specific switches etc. in templates. These templates will be stored in the **Documents and Settings\<user>\Application Data\codeblocks\UserTemplates** directory. If the templates are to be open to all users, they have to be copied to a corresponding directory of the CodeBlocks installation. These templates will then be displayed at the next startup of CodeBlocks under 'New' → 'Project' → 'User templates'.

Note:

The available templates in the Project Wizard can be edited by selection via right-click.

1.4 Create Projects from Build Targets

In projects it is necessary to have different variants of the project available. Variants are called Build Targets. They differ with respect to their compiler options, debug information and/or choice of files. A Build Target can also be outsourced to a separate project. To do so, click 'Project' → 'Properties', select the variant from the tab 'Build Targets' and click the 'Create project from target' button (see [Figure 1.2](#) on page 4).

1.5 Virtual Targets

Projects can be further structured in CodeBlocks by so-called Virtual Targets. A frequently used project structure consists of two Build Targets, one 'Debug' Target which

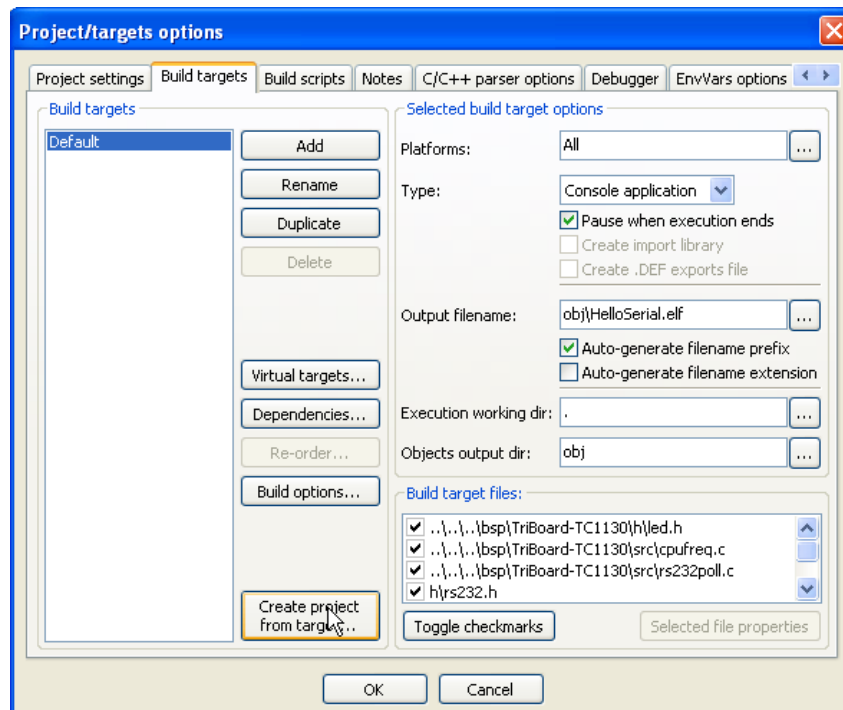


Figure 1.2: Build Targets

contains debug information and one 'Release' Target without this information. By adding Virtual Targets via 'Project' → 'Properties' → 'Build Targets' individual Build Targets can be combined. For example, a Virtual Target 'All' can create the Targets Debug and Release simultaneously. Virtual Targets are shown in the symbol bar of the compiler under Build Targets.

1.6 Pre- and Postbuild steps

CodeBlocks makes it possible to perform additional operations before or after compiling a project. These operations are called Prebuilt or Postbuilt Steps. Typical Postbuilt Steps are:

- Creating an Intel Hexformat from a finished object
- Manipulating objects by `objcopy`
- Generating dump files by `objdump`

Example

Creating a Disassembly from an object under Windows. Piping to a file requires calling `cmd` with the `/c` option.

```
cmd /c objdump -D name.elf > name.dis
```

Archiving a project can be another example for a Postbuilt Step. For this purpose, create a Build Target 'Archive' and include the following instruction in the Postbuilt Step:

```
zip -j9 $(PROJECT_NAME)_$(TODAY).zip src h obj $(PROJECT_NAME).cbp
```

With this command, the active project and its sources, header and objects will be packed as a zip file. In doing so, the Built-in variables `$(PROJECT_NAME)` and `$(TODAY)`, the project name and the current date will be extracted (see [section 3.2](#) on page 54). After the execution of the Target 'Archive', the packed file will be stored in the project directory.

In the `share/codeblocks/scripts` directory you will find some examples for scripts. You can add a script via menu 'Settings' → 'Scripting' and register in a menu. If you execute e.g. the script `make_dist` from the menu then all files belonging to a project will be compressed in an archive `<project>.tar.gz`.

1.7 Adding Scripts in Build Targets

CodeBlocks offers the possibility of using menu actions in scripts. The script represents another degree of freedom for controlling the generation of your project.

Note:

A script can also be included at a Build Target.

1.8 Workspace and Project Dependencies

In CodeBlocks, multiple projects can be open. By saving open projects via 'File' → 'Save workspace' you can collect them in a single workspace under `<name>.workspace`. If you open `<name>.workspace` during the next startup of von CodeBlocks, all projects will show up again.

Complex software systems consist of components which are managed in different CodeBlocks projects. Furthermore, with the generation of such software systems, there are often dependencies between these projects.

Example

A project A contains fundamental functions which are made available to other projects in the form of a library. Now, if the sources of this project are modified, then the library has to be rebuilt. To maintain consistency between a project B which uses the functions and project A which implements the functions, project B has to depend on project A. The necessary information on the dependencies of projects is stored in the relevant workspace, so that each project can be created separately. The usage of dependencies makes it also possible to control the order in which the projects will be generated. The dependencies for projects can be set via the selecting the menu 'Project' → 'Properties' and then clicking the 'Project's dependencies' button.

1.9 Including Assembler files

In the Management window of the Project View, Assembler files are shown in the **ASM Sources** category. The user can change the listing of files in categories (see [section 1.1](#) on page 2). Right-clicking one of the listed Assembler files will open a context menu. Select 'Properties' to open a new window. Now select the 'Build' tab and activate the two fields

'Compile file' and 'Link file'. Then select the 'Advanced' tab and execute the following steps:

1. Set 'Compiler variable' to CC
2. Select the compiler under 'For this compiler'
3. Select 'Use custom command to build this file'
4. In the window, enter:

```
$compiler $options $includes <asopts> -c $file -o $object
```

The CodeBlocks variables are marked by \$ (see [section 3.4](#) on page 58). They are set automatically so that you only have to replace the Assembler option <asopt> by your own settings.

1.10 Editor and Tools

1.10.1 Default Code

The company's Coding Rules require source files to have a standard design. CodeBlocks makes it possible to include a predefined content at the beginning of a file automatically when creating new C/C++ sources and headers. This predefined content is called default code. This setting can be selected under 'Stettings' → 'Editor' Default Code. If you create a new file then a macro expansion of variables, e.g. defined via menu 'Settings' → 'Global variables' , is performed. A new file can be created via the menu 'File' → 'New' → 'File' .

Example

```

/*****
 * Project: $(proejct)
 * Function:
 *****/
 * $Author: mario $
 * $Name: $
 *****/
 *
 * Copyright 2007 by company name
 *
 *****/

```

1.10.2 Abbreviation

A lot of typing can be saved in CodeBlocks by defining abbreviation. This is done by selecting 'Settings' → 'Editor' and defining the abbreviations under the name <name>, which can then be called by the keyboard shortcut Ctrl-J (see [Figure 1.3](#) on page 7).

Parametrisation is also possible by including variables \$(NAME) in the abbreviations.

```

#ifndef $(Guard token)
#define $(Guard token)
#endif // $(Guard token)

```

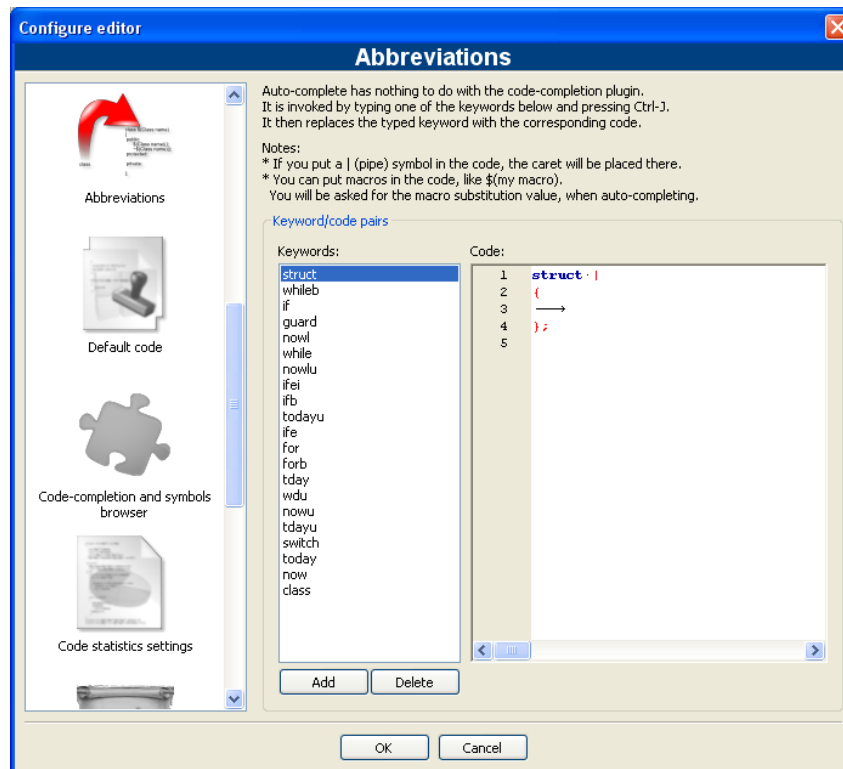


Figure 1.3: Defining abbreviations

When performing the abbreviation `<name>` in the source text and performing Ctrl-J, the content of the variable is requested and included.

1.10.3 Personalities

CodeBlocks settings are saved as application data in a file called `<user>.conf` in the `codeblocks` directory. This configuration file contains information such as the last opened projects, settings for the editor, display of symbol bars etc. By default, the 'default' personality is set so that the configuration is stored in the file `default.conf`. If CodeBlocks is called from the command line with the parameter `--personality=myuser`, the settings will be stored in the file `myuser.conf`. If the profile does not exist already, it will automatically be created. This procedure makes it possible to create the corresponding profiles for different work steps. If you start CodeBlocks from the command line with the additional parameter `--personality=ask`, a selection box will be displayed for all the available profiles.

Note:

The name of the current profile/personality is displayed in the right corner of the status bar.

1.10.4 Configuration Files

The CodeBlocks settings are stored in the `default.conf` profile in the `codeblocks` directory of your Application Data. When using personalities (see [subsection 1.10.3](#) on page 7),

the configuration details will be stored in the `<personality>.conf` file.

The tool `cb_share_conf`, which can be found in the CodeBlocks installation directory, is used for managing and storing these settings.

If you wish to define standard settings for several users of a computer, the configuration file `default.conf` has to be stored in the directory `\Documents and Settings\Default User\Application Data\codeblocks`. During the first startup, CodeBlocks will copy the presettings from 'Default User' to the application data of the current users.

To create a portable version of CodeBlocks on a USB stick, proceed as follows. Copy the CodeBlocks installation to a USB stick and store the configuration file `default.conf` in this directory. The configuration will be used as a global setting. Please take care that the file is writeable, otherwise changes of the configuration cannot be stored.

1.10.5 Navigate and Search

In CodeBlocks there are different ways of quick navigation between files and functions. Setting bookmarks is a typical procedure. Via the shortcut Ctrl-B a bookmark is set or deleted in the source file. Via Alt-PgUp you can jump to the previous bookmark, and via Alt-PgDn you can jump to the next bookmark.

If you select the workspace or a project in the workspace in the project view you will be able to search for a file in the project. Just select 'Find file' from the context menu, then type the name of the file and the file will be selected. If you hit return this file will be opened in the editor (see [Figure 1.4](#) on page 8).

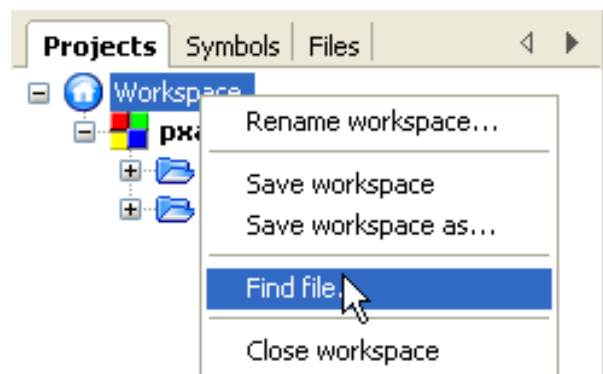


Figure 1.4: Searching for files

In CodeBlocks you can easily navigate between header/source files like:

1. Set cursor at the location where a header file is include and open this file via the context menu 'open include file' (see [Figure 1.5](#) on page 9)
2. Swap between header and source via the context menu 'Swap header/source'
3. Select e.g. a define in the editor and choose 'Find declaration' from the context menu to open the file with its declaration.

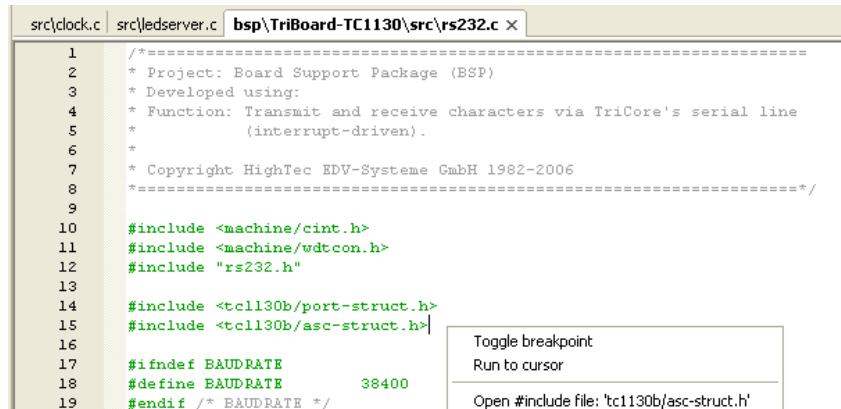


Figure 1.5: Opening of a header file

CodeBlocks offers several ways of searching within a file or directory. The dialogue box for searching is opened via 'Search' → 'Find' (Ctrl-F) or 'Find in Files' (Ctrl-Shift-F).

Alt-G and Ctrl-Alt-G are another useful functions. The dialogue which will open on using this shortcut, lets you select files/functions and then jumps to the implementation of the selected function (see Figure 1.6 on page 9) or opens the selected file in the editor. You may use wildcards like * or ? etc. for an incremental search in the dialog.

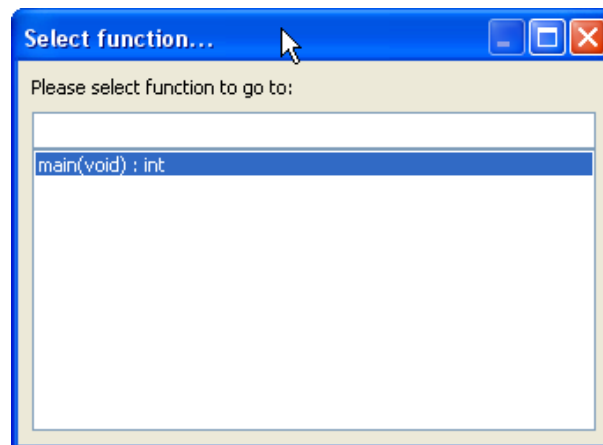


Figure 1.6: Search for functions

Note:

With the Ctrl-PgUp shortcut you can jump to the previous function, and via Ctrl-PgDn you can jump to the next function.

In the editor, you can open a new Open Files dialog via Ctrl-Tab and you can switch between the listed entries. If the Ctrl-key is pressed, then a file can be selected in different ways:

1. If you select an entry with the left mouse button, then the selected file will be opened.

2. If you press the Tab-key you will switch between the listed entries. Releasing the Ctrl-key will open the selected file.
3. If you move the mouse over the listed entries, then the current selection will be highlighted. Releasing the Ctrl-key will open the selected file.
4. If the mouse pointer is outside the highlighted selection, then you can use the mouse-wheel to switch between the entries. Releasing the Ctrl-key will open the selected file.

A common procedure when developing software is to struggle with a set of functions which are implemented in different files. The Browse Tracker plugin will help you solve this problem by showing you the order in which the files were selected. You can then comfortably navigate the function calls (see [section 2.8](#) on page 38).

The display of line numbers in CodeBlocks can be activated via 'Settings' → 'General Settings' in the field 'Show line numbers'. The shortcut Ctrl-G or the menu command 'Search' → 'Goto line' will help you jump to the desired line.

Note:

If you hold the Ctrl key and then select text in the CodeBlocks editor you can perform e.g. a Google search via the context menu.

1.10.6 Symbol view

The CodeBlocks Management window offers a tree view for symbols of C/C++ sources for navigating via functions or variables. As the scope of this view, you can set the current file or project, or the whole workspace.

Note:

Entering a search term or symbol names in the 'Search' input mask of the Symbol Browser results in a filtered view of the symbols if any hits occurred.

The following categories exist for the symbols:

Global functions Lists the implementation of global functions.

Global typedefs Lists the use of **typedef** definitions.

Global variables Displays the symbols of global variables.

Preprocessor symbols Lists the pre-processor directives created by **#define**.

Global macros Lists macros of pre-processor directives.

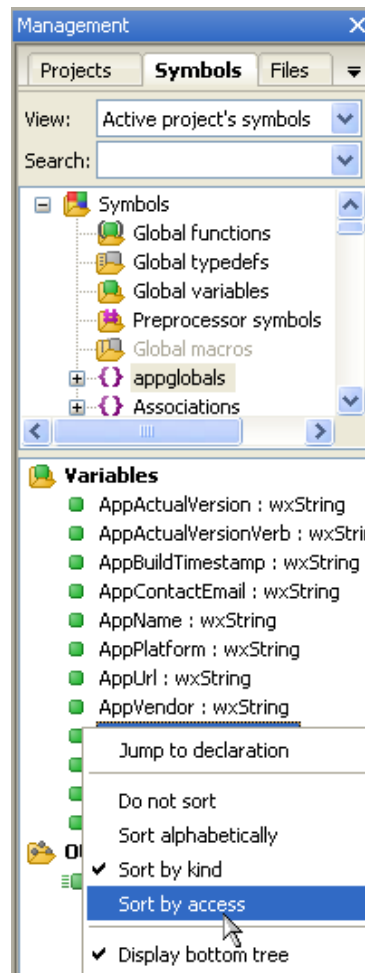


Figure 1.7: Symbol view

Structures and classes are displayed in the 'bottom tree' and the sort sequence can be modified via the context menu. If a category is selected by mouse-click, the found symbols will be displayed in the lower part of the window (see [Figure 1.7](#) on page 11). Double-clicking the symbol will open the file in which the symbol is defined or the function implemented, and jumps to the corresponding line. An auto-refresh of the symbol browser without saving a file, can be activated via the menu 'Settings' → 'Editor' → 'Code Completion' (see [Figure 1.8](#) on page 11). For projects with many symbols the performance within CodeBlocks will be affected.

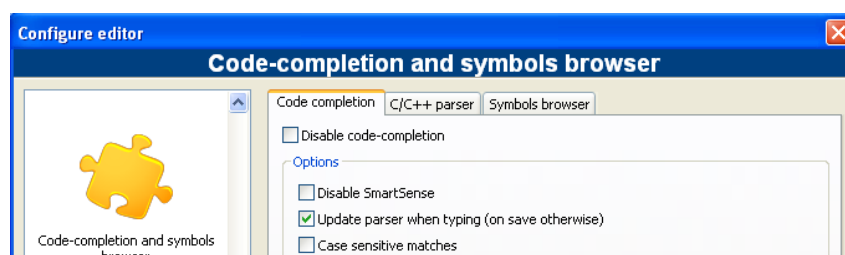


Figure 1.8: Enable real-time parsing

Note:

In the editor, a list of the classes can be displayed via the context menus 'Insert Class method declaration implementation' or 'All class methods without implementation' .

1.10.7 Including external help files

The CodeBlocks development environment supports the inclusion of external help files via the menu 'Settings' → 'Environment' . Include the manual of your choice in the chm format in 'Help Files' select 'this is the default help file' (see [Figure 1.9](#) on page 12). The entry \$(keyword) is a placeholder for a select item in your editor. Now you can select a function in an opened source file in CodeBlocks by mouse-click, and the corresponding documentation will appear while pressing F1.

If you have included multiple help files, you can select a term in the editor and choose a help file from the context menu 'Locate in' for CodeBlocks to search in.

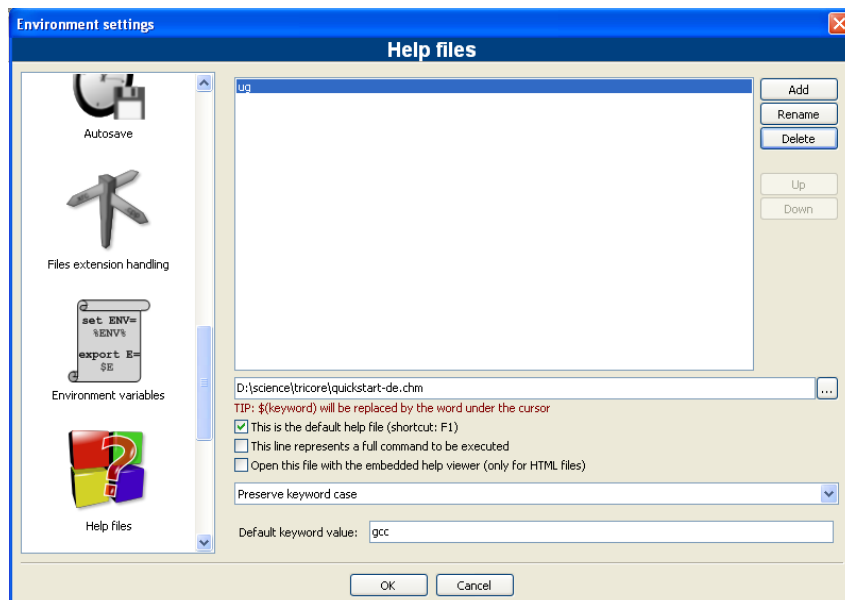


Figure 1.9: Settings for help files

In CodeBlocks you can add even support for man pages. Just add a entry 'man' and specify the path as follows.

```
man:/usr/share/man
```

CodeBlocks provides an 'Embedded HTML Viewer', which can be used to display simple html file and find keywords within this file. Just configure the path to the html file, which should be parsed and enable the checkbox 'Open this file with embedded help viewer' via the menu 'Settings' → 'Environment' → 'Help Files' .

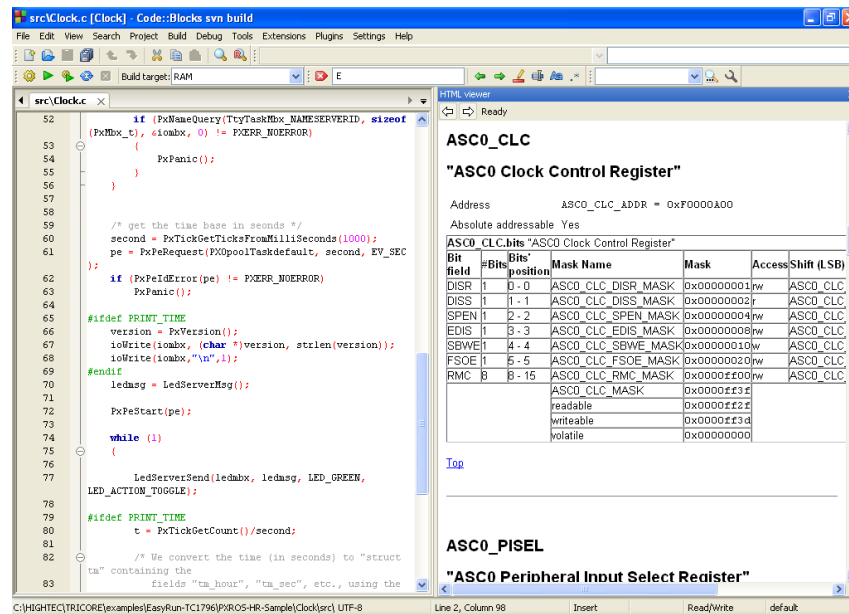


Figure 1.10: Embedded HTML Viewer

Note:

If you select a html file with a double-click within the file explorer (see [section 2.7](#) on page 34) then the embedded html viewer will be started, as long as no association for html files is made in file extensions handler.

1.10.8 Including external tools

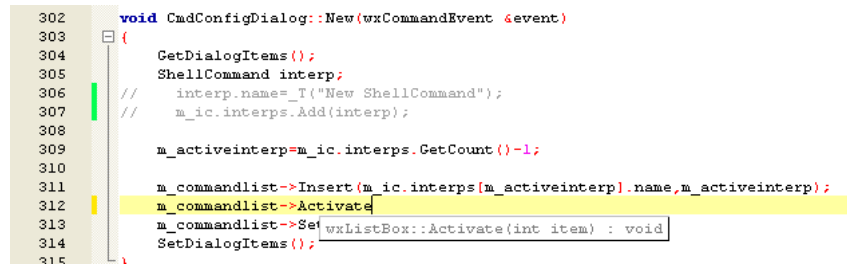
Including external tools is possible in CodeBlocks via 'Tools' → 'Configure Tools' → 'Add'. Built-in variables (see [section 3.2](#) on page 54) can also be accessed for tool parameters. Furthermore there are several kinds of launching options for starting external applications. Depending on the option, the externally started applications are stopped when CodeBlocks is quit. If the applications are to remain open after quitting CodeBlocks, the option 'Launch tool visible detached' must be set.

1.11 Tips for working with CodeBlocks

In this chapter we will present some useful settings in CodeBlocks.

1.11.1 Tracking of Modifications

CodeBlocks provides a feature to track modifications within a source file and to show a bar in the margin for the changes. Modifications are marked with a yellow changebar and modifications that are already saved will use a green changebar (see [Figure 1.11](#) on page 14). You can navigate between your changes via the menu 'Search' → 'Goto next changed line' or 'Search' → 'Goto previous changed line'. The same functionality is also accessible via the shortcuts Ctrl-F3 and Ctrl-Shift-F3.



```

302 void CmdConfigDialog::New(wxCommandEvent &event)
303 {
304     GetDialogItems();
305     ShellCommand interp;
306     // interp.name=_T("New ShellCommand");
307     // m_ic.interps.Add(interp);
308
309     m_activeinterp=m_ic.interps.GetCount()-1;
310
311     m_commandlist->Insert(m_ic.interps[m_activeinterp].name,m_activeinterp);
312     m_commandlist->Activate
313     m_commandlist->Set(wxListBox::&Activate(int item) : void
314     SetDialogItems();
315 }

```

Figure 1.11: Tracking of modifications

This feature can be enabled or disabled with the checkbox 'Use Changebar' in the menu 'Settings' → 'Editor' → 'Margins and caret'.

Note:

If a modified file is closed, then the changes history like undo/redo and changebars get lost. Via the menu 'Edit' → 'Clear changes history' or the corresponding context menu you are able to clear the changes history even if the file is kept open.

1.11.2 Data Exchange with other applications

Data can be exchanged between CodeBlocks and other applications. For this interprocess communication DDE (Dynamic Data Exchange) is used for windows and under different operating systems it is a TCP based communication.

With this interface different commands with the following syntax can be sent to a CodeBlocks instance.

```
[<command> ("<parameter>")]
```

These commands are currently available:

| | |
|----------|--|
| Open | <p>The command</p> <pre>[Open ("d:\temp\test.txt")]</pre> <p>uses the parameter, in our case it is a file specified with an absolute path, and opens it in an existing CodeBlocks instance or starts a first instance if required.</p> |
| OpenLine | <p>This command opens a file at a given line number in a CodeBlocks instance. The line number is specified with : line.</p> <pre>[OpenLine ("d:\temp\test.txt:10")]</pre> |
| Raise | <p>Set the focus to the CodeBlocks instance. A parameter must not be passed.</p> |

1.11.3 Configuring environmental variables

The configuration for an operating system is specified by so-called environmental variables. The environmental variable PATH for example contains the path to an installed compiler.

The operating system will process this environmental variable from beginning to end, i.e. the entries at the end will be searched last. If different versions of a compiler or other applications are installed, the following situations can occur:

- An incorrect version of a software is called
- Installed software packages call each other

So it might be the case that different versions of a compilers or other tools are mandatory for different projects. One possibility in such a case is to change the environmental variables in the system control for every project. However, this procedure is error-prone and not flexible. For this requirement, CodeBlocks offers an elegant solution. Different configurations of environmental variables can be created which are used only internally in CodeBlocks. Additionally, you can switch between these configurations. The [Figure 1.12](#) on page 15 shows the dialogue which you can open via 'Environment Variables' under 'Settings' → 'Environment'. A configuration is created via the 'Create' button.

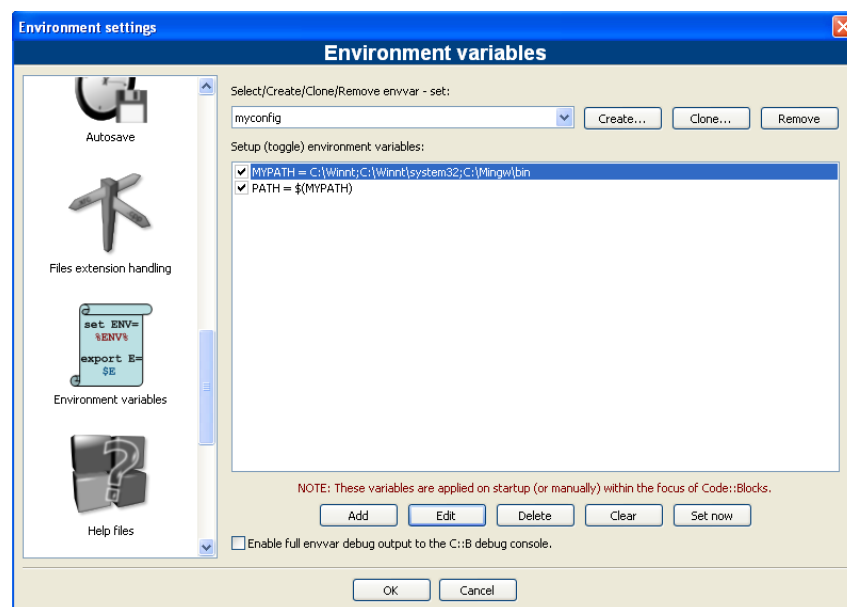


Figure 1.12: Environmental variables

Access and scope of the environmental variables created here, is limited to CodeBlocks. You can expand these environmental variables just like other CodeBlocks variables via \$(NAME).

Note:

A configuration for the environmental variable for each project can be selected in the context menu 'Properties' of the 'EnvVars options' tab.

Example

You can write the used environment into a postbuild Step (see [section 1.6](#) on page 4) in a file <project>.env and archive it within your project.


```
cmd /c echo \%PATH\% > project.env
```

or under Linux

```
echo \$PATH > project.env
```

1.11.4 Switching between perspectives

Depending on the task in hand, it can be useful to have different configurations or views in CodeBlocks and to save these configurations/views. By default, the settings (e. g. show/hide symbol bars, layout, etc.) are stored in the `default.conf` configuration file. By using the command line option `--personality=ask` during the start of CodeBlocks, different settings can be selected. Apart from this global setting, a situation might occur where you wish to switch between different views of windows and symbol bars during a session. Editing files and debugging projects are two typical examples for such situations. CodeBlocks offers a mechanism for storing and selecting different perspectives to prevent the user from frequently having to open and close windows and symbol bars manually. To save a perspective, select the menu 'View' → 'Perspectives' → 'Save current' and enter a name at <name>. The command 'Settings' → 'Editor' → 'Keyboard shortcuts' → 'View' → 'Perspectives' → '<name>' allows a keyboard shortcut to be defined for this process. This mechanism makes it possible to switch between different views by simply using hot keys.

Note:

Another example is editing a file in Full Screen mode without symbol bars. You can create a perspective such as 'Full' and assign a hot key for this purpose.

1.11.5 Switching between projects

If several projects or files are opened at the same time, the user needs a way to switch quickly between the projects or files. CodeBlocks has a number of shortcuts for such situations.

Alt-F5 Activates the previous project from the project view.

Alt-F6 Activates the next project from the project view.

F11 Switches within the editor between a source file <name>.cpp and the corresponding header file <name>.h

1.11.6 Extended settings for compilers

During the build process of a project, the compiler messages are displayed in the Messages window in the Build Log tab. If you wish to receive detailed information, the display can be extended. For this purpose click 'Settings' → 'Compiler and Debugger' and select 'Other Settings' in the drop-down field.

Take care that the correct compiler is selected. The 'Full command line' setting in the Compiler Logging field outputs the complete information in the Build Log. In addition,

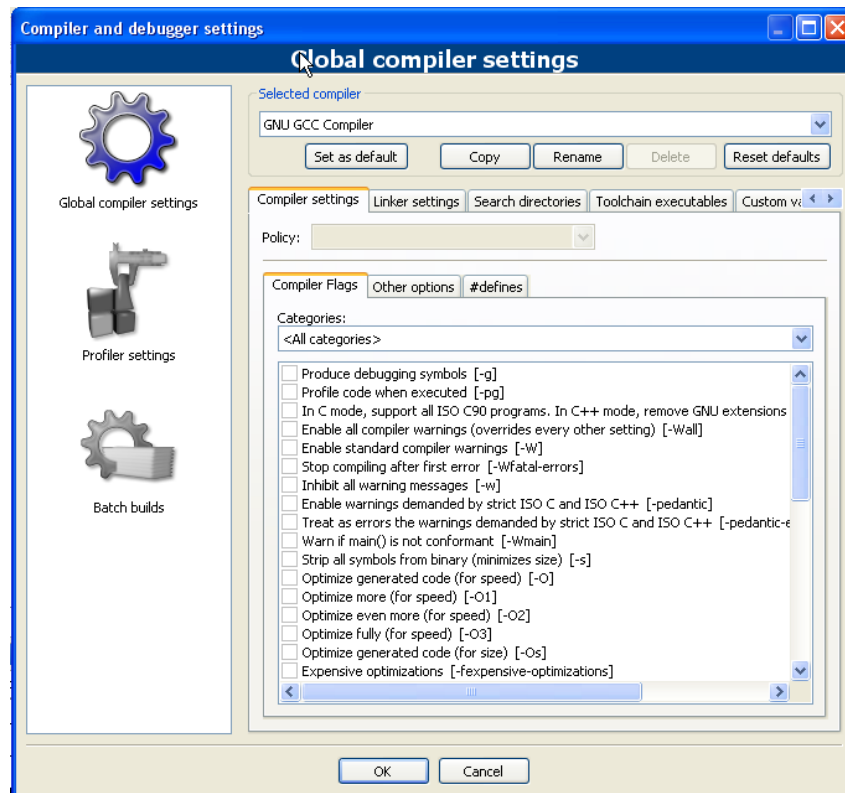


Figure 1.13: Setting detail information

this output can be logged in a HTML file. For this purpose select 'Save build log to HTML file when finished'. Furthermore, CodeBlocks offers a progress bar for the build process in the Build Log window which can be activated via the 'Display build progress bar' setting.

1.11.7 Zooming within the editor

CodeBlocks offers a very efficient editor. This editor allows you to change the size in which the opened text is displayed. If you use a mouse with a wheel, you only need to press the Ctrl key and scroll via the mouse wheel to zoom in and out of the text.

Note:

With the shortcut Ctrl-Numepad-/ or with the menu 'Edit' → 'Special commands' → 'Zoom' → 'Reset' the original font size of the active file in the editor is restored.

1.11.8 Wrap Mode

When editing text files, e. g. *.txt, within CodeBlocks, it might be useful to have the text wrapped, meaning long lines will be displayed in several lines on the screen so that they can be properly edited. The 'Word wrap' function can be activated via 'Settings' → 'Editor' → 'Other Options' or by setting the checkbox 'Word wrap'. The Home and End keys position the cursor at the beginning or end of wrapped lines respectively. When setting 'Settings' → 'Editor' → 'Other Options' and 'Home key always move to caret to

first column', the cursor will be positioned at the beginning or end of the current line respectively, if the Home or End keys are pressed. If positioning the cursor at the beginning of the first line of the current paragraph is desired, the key combination 'Alt-Home' is to be used. The same applies analogously for 'Alt-End' for positioning the cursor at the end of the last line of the current paragraph.

1.11.9 Select modes in editor

CodeBlocks supports different modes for selecting or pasting of strings.

1. With the left mouse button a text in the active editor can be selected and then the mouse button can be released. With the mouse wheel the user can scroll to a position. If the middle mouse button is pressed then the formerly selected text will be inserted. This feature is available per file and can be seen a clipboard per file.
2. Pressing the 'ALT' key will activate the so-called block-select mode and a rectangle selection can be raised with the left mouse button. If the Alt key is released this selection can be copied or pasted. This feature is helpful if you want to select some columns e.g. of an array and copy and paste the content.
3. In the menu 'Settings' → 'Editor' → 'Margins und Caret' so-called 'Virtual Spaces' can be activated. This option enables that a selection in the block select mode can start or end within an empty line.
4. In the menu 'Settings' → 'Editor' → 'Margins und Caret' the 'Multiple Selection' can be activated. While holding the Ctrl-key the user can select different lines in the active editor via the left mouse button. The selections will be appended in the clipboard via the shortcut Ctrl-C or Ctrl-X. Ctrl-V will insert the content at the current cursor position. An additional option called 'Enable typing (and deleting)' can be activated for multiple selections. This feature is useful if you want to add a pre-processor directive like `#ifdef` at different source lines or if you want to overwrite or replace a text at several positions.

Note:

Most Linux window managers use ALT-LeftClickDrag to move a window, so you will have to disable this window manager behavior first for block select to work.

1.11.10 Code folding

CodeBlocks supports so called code folding. With this feature you can fold e.g. functions within the CodeBlocks editor. A folding point is marked by minus symbol in the left margin of the editor view. In the margin the beginning and the end of a folding point is visible as vertical line. If you click the minus symbol with the left mouse button the code snippet will be folded or unfolded. Via the menu 'Edit' → 'Folding' you can select the folding. In the editor you see folded code as continuous horizontal line.

Note:

The folding style and the folding depth limit can be configured via menu 'Settings' → 'Editor' → 'Folding' .

CodeBlocks provides the folding feature also for preprocessor directives. To enable this feature select 'Fold preprocessor commands' via the menu 'Settings' → 'Editor' in the folding entry.

Another possibility is to set user defined folding points. The start of folding point is entered as comment with a opening bracket and the end is marked with a comment with a closing bracket.

```
//{  
code with user defined folding  
//}
```

1.11.11 Auto complete

If you open a project in CodeBlocks the 'Search directories' of your compiler and the project, the sources and headers of your project are parsed. In addition the keywords of the corresponding lexer file are parsed. The parse information is used for the auto complete feature in CodeBlocks. Please check the settings for the editor if this feature is enabled. The auto completion is accessible with the shortcut Ctrl-Space. Via the menu 'Settings' → 'Editor' → 'Syntax highlighting' you can add user defined keywords to your lexer.

1.11.12 Find broken files

If a file is removed from disk, but is still included in the project file `<project>.cbp`, then this 'broken file' will be shown a broken symbol in the project view. You should use the menu 'Remove file from project' instead of deleting files.

In large projects with a lot of subdirectories the search for broken files can be time consuming. CodeBlocks offers with the plug-in ThreadSearch (see [section 2.6](#) on page 30) a simple solution for this problem. If you enter a search expression in ThreadSearch and select the option 'Project files' or 'Workspace files', then ThreadSearch will parse all files that are included in a project or workspace. If a broken file is found ThreadSearch will issue an error with the missing file.

1.11.13 Including libraries

In the build options of a project, you can add the used libraries via the 'Add' button in the 'Link libraries' entry of the 'Linker Settings'. In doing so, you can either use the absolute path to the library or just give the name without the `lib` prefix and file extension.

Example

For a library called `<path>\libs\lib<name>.a`, just write `<name>`. The linker with the corresponding search paths will then include the libraries correctly.

Note:

Another way to include libraries is documented in [section 2.10](#) on page [39](#).

1.11.14 Object linking order

During compiling, objects `name.o` are created from the sources `name.c/cpp`. The linker then binds the individual objects into an application `name.exe` or for the embedded systems `name.elf`. In some cases, it might be desirable to predefine the order in which the objects will be linked. In CodeBlocks, this can be achieved by assigning priorities. In the context menu 'Properties', you can define the priorities of a file in the Build tab. A low priority will cause the file to be linked earlier.

1.11.15 Autosave

CodeBlocks offers ways of automatically storing projects and source files, or of creating backup copies. This feature can be activated in the menu 'Settings' → 'Environment' → 'Autosave'. In doing so, 'Save to .save file' should be specified as the method for creating the backup copy.

1.11.16 Settings for file extensions

In CodeBlocks, you can choose between several ways of treating file extensions. The settings dialogue can be opened via 'Settings' → 'Files extension handling'. You can either use the applications assigned by Windows for each file extension (open it with the associated application), or change the setting for each extensions in such a way that either a user-defined program will start (launch an external program), or the file will be opened in the CodeBlocks editor (open it inside Code::Blocks editor).

Note:

If a user-defined program is assigned to a certain file extension, the setting 'Disable Code::Blocks while the external program is running' should be deactivated because otherwise CodeBlocks will be closed whenever a file with this extension is opened.

1.12 CodeBlocks at the command line

IDE CodeBlocks can be executed from the command line without a graphic interface. In such a case, there are several switches available for controlling the build process of a project. Since CodeBlocks is thus scriptable, the creation of executables can be integrated into your own work processes.

```
codeblocks.exe /na /nd --no-splash-screen --built <name>.cbp --target='Release'
```

<filename> Specifies the project `*.cbp` filename or workspace `*.workspace` filename. For instance, `<filename>` may be `project.cbp`. Place this argument at the end of the command line, just before the output redirection if there is any.

`--file=<filename>[:line]`
Open file in Code::Blocks and optionally jump to a specific line.

`/h, --help` Shows a help message regarding the command line arguments.

`/na, --no-check-associations`
Don't perform any file association checks (Windows only).

`/nd, --no-dde` Don't start a DDE server (Windows only).

`/ni, --no-ipc` Don't start an IPC server (Linux and Mac only).

`/ns, --no-splash-screen`
Hides the splash screen while the application is loading.

`/d, --debug-log`
Display the debug log of the application.

`--prefix=<str>`
Sets the shared data directory prefix.

`/p, --personality=<str>, --profile=<str>`
Sets the personality to use. You can use `ask` as the parameter to list all available personalities.

`--rebuild` Clean and build the project or workspace.

`--build` Build the project or workspace.

`--target=<str>`
Sets target for batch build. For example `--target='Release'`.

`--no-batch-window-close`
Keeps the batch log window visible after the batch build is completed.

`--batch-build-notify`
Shows a message after the batch build is completed.

`--safe-mode` All plugins are disabled on startup.

`> <build log file>`
Placed in the very last position of the command line, this may be used to redirect standard output to log file. This is not a codeblock option as such, but just a standard DOS/*nix shell output redirection.

1.13 Shortcuts

Even if an IDE such as CodeBlocks is mainly handled by mouse, keyboard shortcuts are nevertheless a very helpful way of speeding up and simplifying work processes. In the below table, we have collected some of the available keyboard shortcuts.

1.13.1 Editor

| Function | Shortcut Key |
|-------------------------------|-------------------|
| Undo last action | Ctrl-Z |
| Redo last action | Ctrl-Shift-Z |
| Swap header / source | F11 |
| Comment highlighted code | Ctrl-Shift-C |
| Uncomment highlighted code | Ctrl-Shift-X |
| Auto-complete / Abbreviations | Ctrl-Space/Ctrl-J |
| Toggle bookmark | Ctrl-B |
| Goto previous bookmark | Alt-PgUp |
| Goto next bookmark | Alt-PgDown |

This is a list of shortcuts provided by the CodeBlocks editor component. These shortcuts cannot be rebound.

| | |
|---|---------------|
| Create or delete a bookmark | Ctrl-F2 |
| Go to next bookmark | F2 |
| Select to next bookmark | Alt-F2 |
| Find selection. | Ctrl-F3 |
| Find selection backwards. | Ctrl-Shift-F3 |
| Find matching preprocessor conditional, skipping nested ones. | Ctrl-K |

1.13.2 Files

| Function | Shortcut Key |
|-------------------------------|----------------------------|
| New file or project | Ctrl-N |
| Open existing file or project | Ctrl-O |
| Save current file | Ctrl-S |
| Save all files | Ctrl-Shift-S |
| Close current file | Ctrl-F4/Ctrl-W |
| Close all files | Ctrl-Shift-F4/Ctrl-Shift-W |

1.13.3 View

| Function | Shortcut Key |
|----------------------------------|--------------|
| Show / hide Messages pane | F2 |
| Show / hide Management pane | Shift-F2 |
| Activate prior (in Project tree) | Alt-F5 |
| Activate next (in Project tree) | Alt-F6 |

1.13.4 Search

| Function | Shortcut Key |
|----------------------------|---------------|
| Find | Ctrl-F |
| Find next | F3 |
| Find previous | Shift-F3 |
| Find in files | Ctrl-Shift-F |
| Replace | Ctrl-R |
| Replace in files | Ctrl-Shift-R |
| Goto line | Ctrl-G |
| Goto next changed line | Ctrl-F3 |
| Goto previous changed line | Ctrl-Shift-F3 |
| Goto file | Alt-G |
| Goto function | Ctrl-Alt-G |
| Goto previous function | Ctrl-PgUp |
| Goto next function | Ctrl-PgDn |
| Goto declaration | Ctrl-Shift-. |
| Goto implementation | Ctrl-. |
| Open include file | Ctrl-Alt-. |

1.13.5 Build

| Function | Shortcut Key |
|----------------------|---------------|
| Build | Ctrl-F9 |
| Compile current file | Ctrl-Shift-F9 |
| Run | Ctrl-F10 |
| Build and Run | F9 |
| Rebuild | Ctrl-F11 |

2 Plugins

2.1 Astyle

Artistic Style is a source code indenter, source code formatter, and source code beautifier for the C, C++, C# programming languages. It can be used to select different styles of coding rules within CodeBlocks.

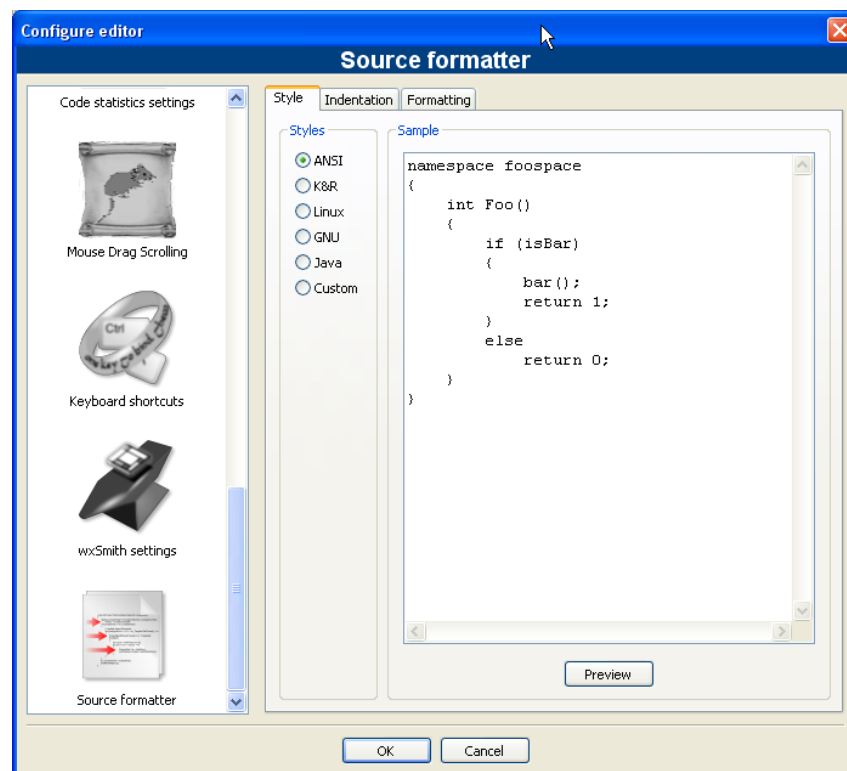


Figure 2.1: Formatting your source code

When indenting source code, we as programmers have a tendency to use both spaces and tab characters to create the wanted indentation. Moreover, some editors by default insert spaces instead of tabs when pressing the tab key, and other editors have the ability to prettify lines by automatically setting up the white space before the code on the line, possibly inserting spaces in a code that up to now used only tabs for indentation.

Since the number of space characters shown on screen for each tab character in the source code changes between editors, one of the standard problems programmers are facing when moving from one editor to another is that code containing both spaces and tabs that was up to now perfectly indented, suddenly becomes a mess to look at when changing to another editor. Even if you as a programmer take care to **ONLY** use spaces or tabs, looking at other people's source code can still be problematic.

To address this problem, Artistic Style was created - a filter written in C++ that automatically re-indent and re-formats C / C++ / C# source files.

Note:

When copying code, for example from the internet or a manual, this code will automatically be adapted to the coding rules in CodeBlocks.

2.2 CodeSnippets

The CodeSnippets plug-in makes it possible to structure text modules and links to files according to categories in a tree view. The modules are used for storing often used files and constructs in text modules and managing them in a central place. Imagine the following situation: A number of frequently used source files are stored in different directories of the file system. The CodeSnippets window provides the opportunity to create categories, and below the categories, links to the required files. With these features, you can control the access to the files independently from where they are stored within the file system, and you can navigate quickly between the files without the need to search the whole system.

Note:

You can use CodeBlocks variables or environment variables in file links e.g. `$(VARNAME)/name.pdf` to parametrise a link in the CodeSnippets browser.

The list of text modules and links can be stored in the CodeSnippets window by right-clicking and selecting 'Save Index' from the context menu. The file `codesnippets.xml` which will be created by this procedure, can then be found in the `codeblocks` subdirectory of your `Documents and Settings\Application data` directory. Under Linux, this information is stored in the `.codeblocks` subdirectory of your HOME directory. The CodeBlocks configuration files will be loaded during the next start-up. If you wish to save the content of CodeSnippets at a different location, select the 'Save Index As' entry. To load this file, select 'Load Index File' during the next start-up of CodeBlocks or include the directory in the 'Settings' context menu under 'Snippet Folder'. The settings are saved in the corresponding file `codesnippets.ini` in your application data.

For including a category, use the 'Add SubCategory' menu. A category can contain Snippets (text modules) or File Links. A text module is created via the 'Add Snippet' command in the context menu. The content is integrated into the text module as 'New snippet' by selecting the text passage in the CodeBlocks editor and dragging and dropping it onto the module and the properties dialog pops up. Double-clicking the newly included entry or selecting 'Edit Text' will open an editor for the content.

Output of a text module is handled in CodeBlocks via the context menu command 'Apply' or by dragging and dropping into the editor. Under Windows, the contents of a Snippet can also be dragged and dropped into other applications. In the CodeSnippets Browser you can copy a selected item with drag and drop to a different category.

Beyond this, text modules can be parametrised by `<name>` variables which can be accessed via `$(name)` (see [Figure 2.2](#) on page 26). The values of the variables can be retrieved in an entry field if the text module is called via the context menu command 'Apply'.

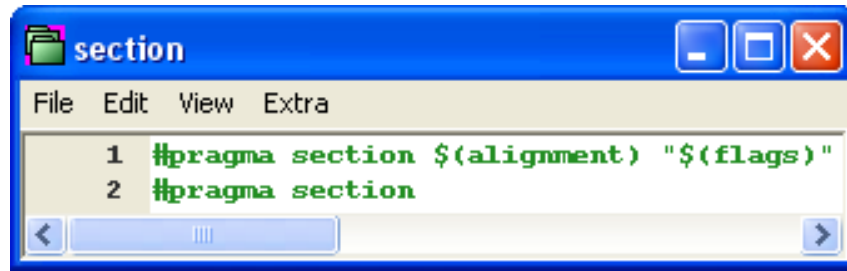


Figure 2.2: Editing a text module

Besides the text modules, links to files can also be created. If, after having created a text module, you click the context menu command 'Properties', then you can select the link target by clicking the 'Link target' button. This procedure will automatically convert the text module into a link to a file. In CodeSnippets, all text modules will be marked by a T symbol, links to a file by an F symbol and urls by an U symbol. If you want to open a selected file (link) in the codesnippets view just select the context menu 'Open File' or hold the 'Alt' key and make a double click on the file.

Note:

You can add even url (e.g. <http://www.codeblocks.org>) in text modules. The url can be opened using the context menu 'Open Url' or using drag and drop to your favorite web browser.

With this setting, if open a link to a pdf file from the codesnippets view a pdf viewer will be started automatically. This method makes it possible for a user to access files which are spread over the whole network, such as cad data, layouts, documentations etc., with the common applications, simply via the link. The content of the codesnippets is stored in the file `codesnippets.xml`, the configuration is stored in the file `codesnippets.ini` in your `application data` directory. This ini file will, for example, contain the path of the file `codesnippets.xml`.

CodeBlocks supports the usage of different profiles. These profiles are called personalities. Starting CodeBlocks with the command line option `--personality=<profile>` will create a new or use an existing profile. Then the settings will not be stored in the file `default.conf`, but in `<personality>.conf` in your `application data` directory instead. The Codesnippets plugin will then store its settings in the file `<personality>.codesnippets.in`. Now, if you load a new content `<name.xml>` in the Codesnippets settings via 'Load Index File', this content will be stored in the corresponding ini file. The advantage of this method lies in the fact that in case of different profiles, different configurations for text modules and links can be managed.

The plug-in offers an additional search function for navigating between the categories and Snippets. The scope for searching Snippets, categories or Snippets and categories can be adjusted. By entering the required search expression, the corresponding entry is automatically selected in the view. Figure 2.3 on page 27 shows a typical display in the CodeSnippets window.

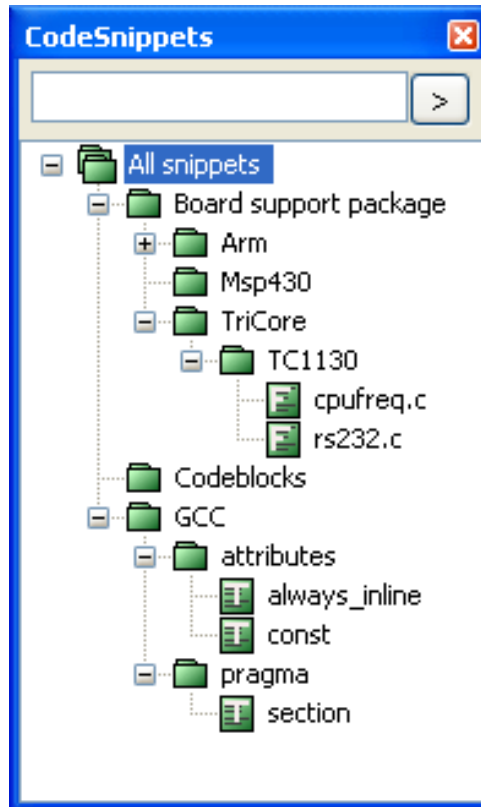


Figure 2.3: CodeSnippets View

Note:

When using voluminous text modules, the content of these modules should be saved in files via 'Convert to File Link' in order to reduce memory usage within the system. If you delete a codesnippet or file link it will be moved to the category `.trash`; if you hold the Shift key the item will be deleted.

2.3 Incremental Search

For an efficient search in open files, CodeBlocks provides the so-called Incremental Search. This search method is initiated for an open file via the menu 'Search' → 'Incremental Search' or by the keyboard shortcut Ctrl-I. The focus is then automatically set to the search mask of the corresponding toolbar. As soon as you begin entering the search term, the background of the search mask will be adjusted in accordance with the occurrence of the term. If a hit is found in the active editor, the respective position in the text is marked in colour. By default the current hit will be highlighted in green. This setting can be changed via 'Settings' → 'Editor' → 'Incremental Search' (see ?? on page ??). Pressing the Return key induces the search to proceed to the next occurrence of the search string within the file. With Shift-Return the previous occurrence can be selected. This functionality is not supported by Scintilla if the incremental search uses regular expressions.

```

m_pToolbar->EnableTool(X

if (m_pControl != 0)
{
    m_SearchText=m_pText;
    m_pToolbar->EnableTo
    m_pToolbar->EnableTo
    m_NewPos=m_pControl-;
    m_OldPos=m_NewPos;
}
else
{
    m_pToolbar->EnableTo
    m_pToolbar->EnableTo
}
...

```

If the search string cannot be found within the active file, this fact is highlighted by the background of the search mask being displayed in red.

ESC Leave the Incremental Search modus.

ALT-DELETE Clear the input of the incremental search field.

The icons in the Incremental Search toolbar have the following meanings:



Deleting the text within the search mask of the Incremental Search toolbar.



, Navigating between the occurrences of a search string.



Clicking this button results in all the occurrences of the search string within the editor being highlighted in colour, instead of only the initial occurrence.



Activating this option restricts the search to the text passage marked within the editor.



This option means a case sensitive search is performed.



Regular expression can be used in the input field of incremental search.

Note:

The standard settings of this toolbar can be configured in 'Settings' → 'Editor' → 'Incremental Search'.

2.4 ToDo List

In complex software projects, where different users are involved, there is often the requirement of different tasks to be performed by different users. For this purpose, CodeBlocks

offers a Todo List. This list can be opened via 'View' → 'To-Do list', and contains the tasks to be performed, together with their priorities, types and the responsible users. The list can be filtered for tasks, users and/or source files. A sorting by columns can be achieved by clicking the caption of the corresponding column.

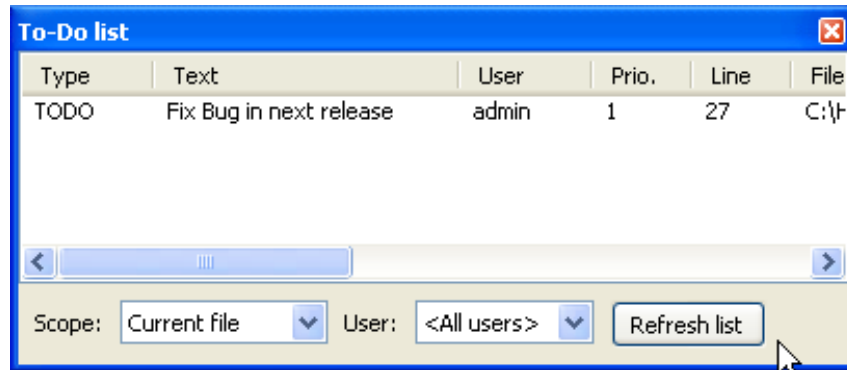


Figure 2.4: Displaying the ToDo List

Note:

The To-Do list can be docked in the message console. Select the option 'Include the To-Do list in the message pane' via the menu 'Settings' → 'Environment'.

If the sources are opened in CodeBlocks, a Todo can be added to the list via the context menu command 'Add To-Do item'. A comment will be added in the selected line of the source code.

```
// TODO (user#1#): add new dialog for next release
```

When adding a To-Do, a dialogue box will appear where the following settings can be made (see [Figure 2.5](#) on page 30).

User User name <user> in the operating system. Tasks for other users can also be created here. In doing so, the corresponding user name has to be created by Add new. The assignment of a Todo is then made via the selection of entries listed for the User.

Note:

Note that the Users have nothing to do with the Personalities used in CodeBlocks.

Type By default, type is set to Todo.

Priority The importance of tasks can be expressed by priorities (1 - 9) in CodeBlocks.

Position This setting specifies whether the comment is to be included before, after or at the exact position of the cursor.

Comment Style A selection of formats for comments (e.g. doxygen).

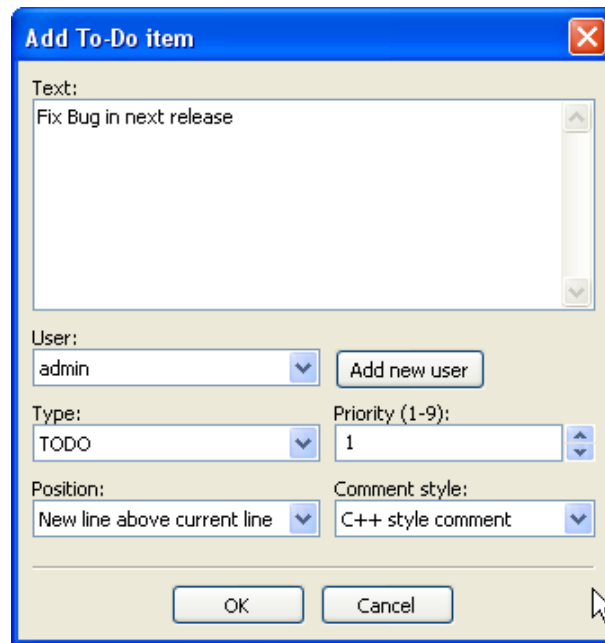


Figure 2.5: Dialogue for adding a ToDo

2.5 Source Code Exporter

The necessity occurs frequently of transferring source code to other applications or to e-mails. If the text is simply copied, formatting will be lost, thus rendering the text very unclear. The CodeBlocks export function serves as a remedy for such situations. The required format for the export file can be selected via 'File' → 'Export'. The program will then adopt the file name and target directory from the opened source file and propose these for saving the export file. The appropriate file extension in each case will be determined by the export format. The following formats are available.

html A text-based format which can be displayed in a web browser or in word processing applications.

rtf The Rich Text format is a text-based format which can be opened in word processing applications such as Word or OpenOffice.

odt Open Document Text format is a standardised format which was specified by Sun and O'Reilly. This format can be processed by Word, OpenOffice and other word processing applications.

pdf The Portable Document Format can be opened by applications such as the Acrobat Reader.

2.6 Thread Search

Via the 'Search' → 'Thread Search' menu, the appropriate plug-in can be shown or hidden as a tab in the Messages Console. In CodeBlocks, a preview can be displayed for the occurrence of a character string in a file, workspace or directory. In doing so, the list of search results will be displayed on the right-hand side of the ThreadSearch Console. By

clicking an entry in the list, a preview is displayed on the left-hand side. By double-clicking in the list, the selected file is opened in the CodeBlocks editor.

Note:

The scope of file extensions to be included in the search, is preset and might have to be adjusted.

2.6.1 Features

ThreadSearch plugin offers the following features:

- Multi-threaded 'Search in files'
- Internal read-only editor to preview the results
- File open in editors notebook
- Contextual menu 'Find occurrences' to start a search in files with the word under cursor

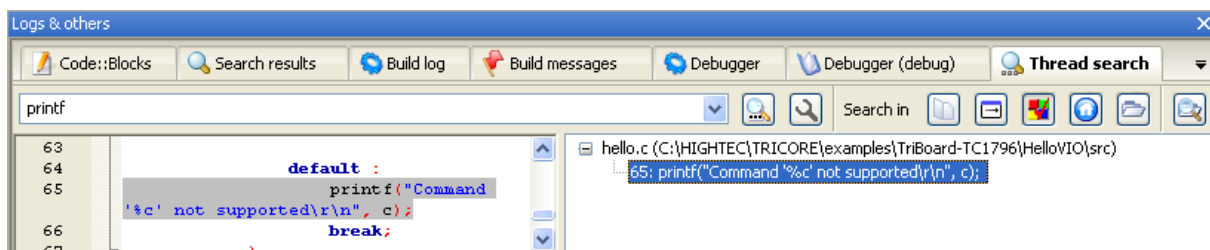


Figure 2.6: Thread Search Panel

2.6.2 Usage

1. Configure your search preferences (see [Figure 2.7](#) on page 32)

Once the plugin is installed, there are 4 ways to run a search:

- a) Type/Select a word in the search combo box and press enter or click on Search on the Thread search panel of the Messages notebook.
- b) Type/Select a word in the toolbar search combo box and press enter or click on Search button.
- c) Right click on any 'word' in active editor and click on 'Find occurrences'.
- d) Click on Search/Thread search to find the current word in active editor.

Note:

Items 1, 2 and 3 may not be available according to current configuration.

2. Click again on the search button to cancel current search.

3. A single click on a result item displays it on the preview editor at right location.
4. A double click on a result item opens or set an editor in editors notebook at right location.

2.6.3 Configuration

To access ThreadSearch plugin configuration panel click on (see [Figure 2.7](#) on page 32):

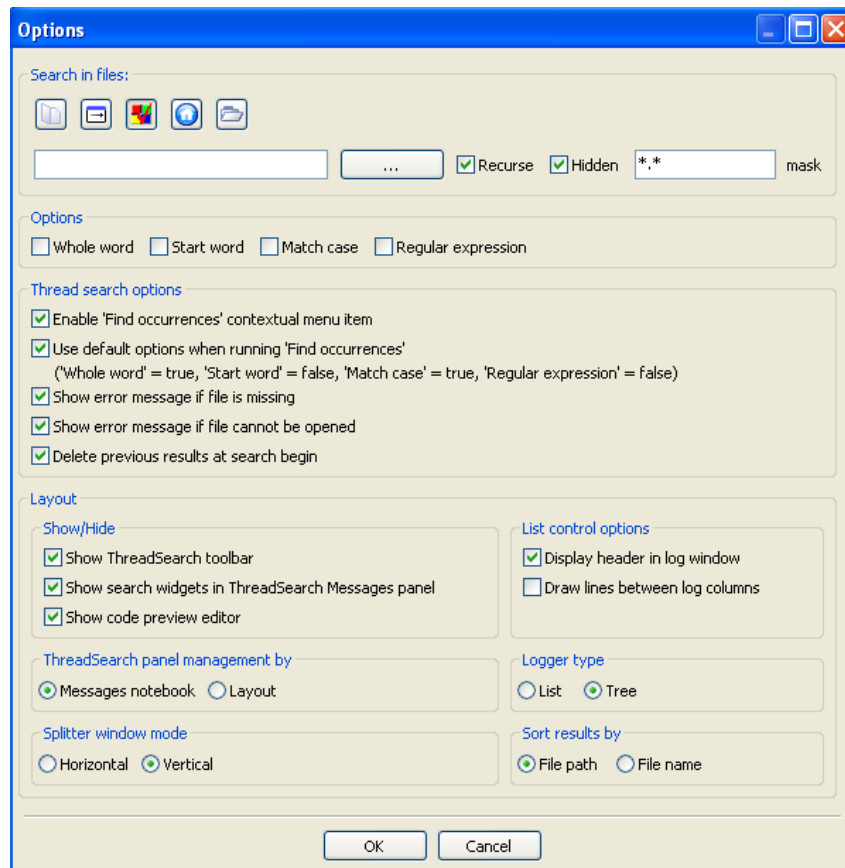


Figure 2.7: Configuration of Thread Search

1. Options button on Messages notebook Thread search panel.
2. Options button on Thread search toolbar.
3. Settings/Environment menu item and then on the Thread search item on the left columns.

Note:

Items 1, 2 and 3 may not be available according to current configuration.

Search in part defines the set of files that will be analysed.

- Project and Workspace checkboxes are mutually exclusive.

- Directory path can be edited or set with Select button.
- Mask is the set a file specifications separated by ';'. For example: *.cpp;*.c;*.h.

2.6.4 Options

Whole word if checked, line matches search expression if search expression is found with no alpha-numeric + '_' before and after.

Start word if checked, line matches search expression if search expression is found at the beginning of a word, ie no alpha-numeric + '_' before search expression.

Match case if checked, the search is case sensitive.

Regular expression the search expression is a regular expression.

Note:

If you want to search for regular expressions like n you will have to set the option 'Use Advanced RegEx searches' via the menu 'Settings' → 'Editor' → 'General Settings' .

2.6.5 Thread search options

Enable 'Find occurrences contextual menu item' If checked, the Find occurrences of 'Focused word' entry is added to the editor contextual menu.

Use default options when running 'Find occurrences' If checked, a set of default options is applied to the searches launched with the 'Find occurrences' contextual menu item. Per default option 'Whole word' and 'Match case' is enabled.

Delete previous results at search begin If ThreadSearch is configured with 'Tree View' then the search results will be listed hierarchically,

- the first node contains the search term
- above the files which contain the search term are listed
- within this list the line number and the corresponding content of the occurrence is displayed

If you search different terms the list will become confusing, therefore previous search results can be cleared at search begin using this option.

Note:

In the list of occurrences single items or all items can be deleted via the context menu 'Delete item' or 'Delete all items' .

2.6.6 Layout

Display header in log window if checked, the header are displayed in the results list control.

Note:

If unchecked, the columns are no longer resizable but space is spared.

Draw lines between columns Draws lines between columns in list mode.

Show ThreadSearch toolbar Display the toolbar of Thread Search plugin.

Show search widgets in ThreadSearch Messages panel If checked, only the results list control and the preview editor are displayed. All other search widgets are hidden (saves space).

Show code preview editor Code preview can be hidden either with this check box or with a double click on the splitter window middle border. This is where it can be shown again.

2.6.7 Panel Management

You can choose different modes how the ThreadSearch window is managed. With the setting 'Message Notebook' the ThreadSearch window will be a dockable window in the message panel. If you choose the setting 'Layout' you will be able to undock the window from the message panel and put it somewhere else.

2.6.8 Logger Type

The view of the search results can be displayed in different ways. The setting 'List' displays all occurrences as list. The other mode 'Tree' gathers all occurrences within a file as a node.

2.6.9 Splitter Window Mode

The user can configure a horizontal or vertical splitting of the preview window and the output window of the search results.

2.6.10 Sort Search Results

The view of the search results may be sorted by path or file name.

2.7 FileManager and PowerShell Plugin

The File Explorer [Figure 2.8](#) on page 35 is included in the FileManager plugin, and can be found in the 'Files' tab. The composition of the File Explorer is shown in [Figure 2.8](#) on page 35.

On top you will find a field for entering the path. By clicking the button at the end of this field, the drop-down field will list a history of the past entries which can be navigated via a scroll bar. The up arrow key on the right-hand side of the field moves up the directory structure one directory.

In the 'Wildcard' field you can enter a filter term for the file display. Leaving the field empty or entering * results in all files being displayed. Entering *.c;*.h, for example will

result in solely C sources and header files being displayed. Opening the pull-down field will, again, list a history of the last entries.

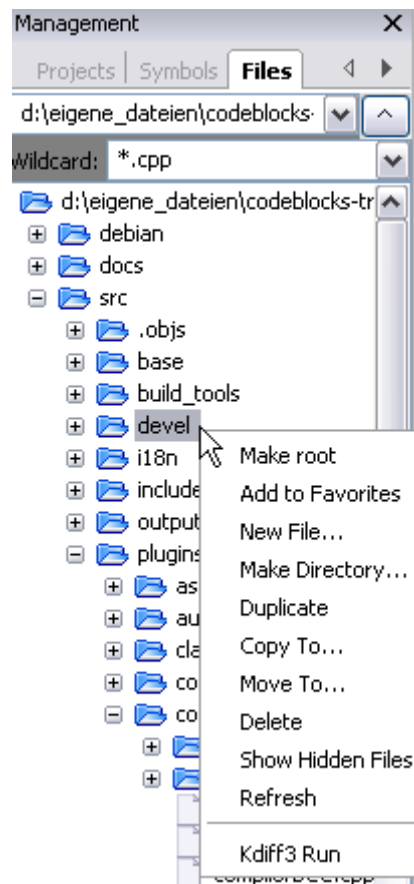


Figure 2.8: The file manager

Pressing the Shift key and clicking selects a group of files or directories, pressing the Ctrl key and clicking selects multiple separate files or directories.

The following operations can be started via the context menu if one or multiple directories are selected in the File Explorer:

Make Root defines the current directory as the root directory.

Add to Favorites sets a marker for the directory and stores it as a favourite. This function allows you to navigate quickly between frequently used directories, also on different network drives.

New File creates a new file in the selected directory.

New Directory creates a new subdirectory in the selected directory.

The following operations can be started via the context menu if one or multiple files or directories are selected in the File Explorer:

Duplicate copies a file/directory and renames it.

Copy To opens a dialogue for entering the target directory in which the copied file/directory is to be stored.

Move To moves the selection to the target location.

Delete deletes the selected files/directories.

Show Hidden Files activates/deactivates the display of hidden system files. When activated, this menu entry is checkmarked.

Refresh update the display of the directory tree.

The following operations can be started via the context menu if one or multiple files are selected in the File Explorer:

Open in CB Editor opens the selected file in the CodeBlocks editor.

Rename renames the selected file.

Add to active project adds the file(s) to the active project.

Note:

The files/directories selected in the File Explorer can be accessed in the PowerShell plugin via the `mpaths` variable.

User-defined functions can be specified via the menu command 'Settings' → 'Environment' → 'PowerShell'. In the PowerShell mask, a new function which can be named at random, is created via the 'New' button. In the 'ShellCommand Executable' field, the executable program is stated, and in the field at the bottom of the window, additional parameters can be passed to the program. By clicking the function in the context menu or the PowerShell menu, the function is started and will then process the selected files/directories. The output is redirected to a separate shell window.

For example a menu entry in 'PowerShell' → 'SVN' and in the context menu is created for 'SVN'. `$file` in this context means the file selected in the File Explorer, `$mpath` the selected files or directories (see [section 3.2](#) on page 54).

```
Add;$interpreter add $mpaths;;;
```

This and every subsequent command will create a submenu, in this case called 'Extensions' → 'SVN' → 'Add'. The context menu is extended accordingly. Clicking the command in the context menu will make the SVN command `add` process the selected files/directories.

TortoiseSVN is a widespread SVN program with integration in the explorer. The program `TortoiseProc.exe` of TortoiseSVN can be started in the command line and displays a dialogue to collect user input. So you can perform the commands, that are available as context menu in the explorer also in the command line. Therefore you can integrate it also a shell extension in CodeBlocks. For example the command

```
TortoiseProc.exe /command:diff /path:$file
```

will diff a selected file in the CodeBlocks file explorer with the SVN base. See [Figure 2.9](#) on page 37 how to integrate this command.

Note:

For files that are under SVN control the file explorer shows overlay icons if they are activated via menu 'View' → 'SVN Decorators'.

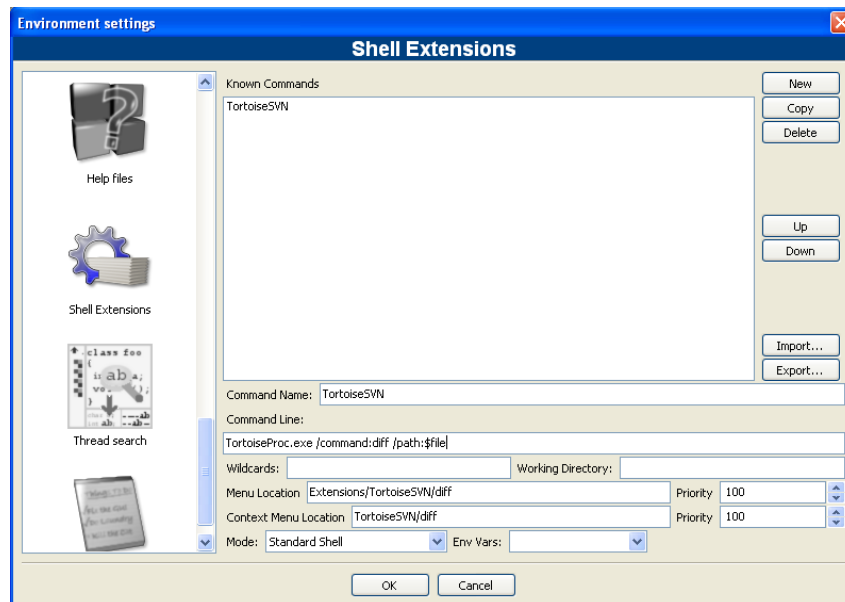


Figure 2.9: Add a shell extension to the context menu

Example

You can use the file explorer to diff files or directories. Follow these steps:

1. Add the name via menu 'Settings' → 'Environment' → 'PowerShell'. This is shown as entry in the interpreter menu and the context menu.
2. Select the absolute path of Diff executable (e.g. kdiff3). The program is accessed with the variable `$interpreter`.
3. Add parameters of the interpreter

```
Diff;$interpreter $mpaths;;;
```

This command will be executed using the selected files or directories as parameter. The selection is accessed via the variable `$mpaths`. This is an easy way to diff files or directories.

Note:

The plug-in supports the use of CodeBlocks variables within the shell extension.

| | |
|----------------------------|-------------------------------------|
| <code>\$interpreter</code> | Call this executable. |
| <code>\$fname</code> | Name of the file without extension. |
| <code>\$fext</code> | Extension of the selected file. |

| | |
|--------------------------------------|--|
| <code>\$file</code> | Name of the file. |
| <code>\$relfile</code> | Name of the file without path info. |
| <code>\$dir</code> | Name of the selected directory. |
| <code>\$reldir</code> | Name of directory without path info. |
| <code>\$path</code> | Absolute path. |
| <code>\$relpath</code> | Relative path of file or directory. |
| <code>\$mpaths</code> | List of current selected files or directories. |
| <code>\$inputstr{<msg>}</code> | String that is entered in a message window. |
| <code>\$parentdir</code> | Parent directory (<code>../</code>). |

Note:

The entries of shell extension are also available as context menu in the CodeBlocks editor.

2.8 Browse Tracker

Browse Tracker is a plug-in that helps navigating between recently opened files in CodeBlocks. The list of recent files is saved in a history. With the menu 'View' → 'Browse Tracker' → 'Clear All' the history is cleared.

With the window 'Browsed Tabs' you can navigate between the items of the recently opened files using the menu entry 'View' → 'Browse Tracker' → 'Backward Ed/Forward Ed' or the shortcut Alt-Left/Alt-Right. The Browse Tracker menu is also accessible as context menu. The markers are saved in the layout file `<projectName>.bmarks`

A common procedure when developing software is to struggle with a set of functions which are implemented in different files. The BrowseTracks plug-in will help you solve this problem by showing you the order in which the files were selected. You can then comfortably navigate the function calls.

The plug-in allows even browse markers within each file in the CodeBlocks editor. The cursor position is memorized for every file. You can set this markers using the menu item 'View' → 'Browse Tracker' → 'Set BrowseMarks' or with selecting a line with the left mouse button. A marker with ... is shown in the left margin. With the menu 'View' → 'Browse Tracker' → 'Prev Mark/Next Mark' or the shortcut Alt-up/Alt-down you can navigate through the markers within a file. If you want to navigate in a file between markers sorted by line numbers then just select the menu 'View' → 'Browse Tracker' → 'Sort BrowseMark'.

With the 'Clear BrowseMark' the marker in a selected line is removed. If a marker is set for a line, holding left-mouse button down for 1/4 second while pressing the Ctrl key will delete the marker for this line. Via the menu 'Clear All BrowseMarks' or with a Ctrl-left click on any unmarked line will reset the markers within a file.

The settings of the plug-in can be configure via the menu 'Settings' → 'Editor' → 'Browse Tracker' .

Mark Style Browse Marks are displayed per default as ... within the margin. With the setting 'Book_Marks' they will be displayed like Bookmarks as blue arrow in the margin. With hide the display of Browse Marks is suppressed.

Toggle Browse Mark key Markers can be set or removed either by a click with the left mouse button or with a click while holding the ctrl key.

Toggle Delay The duration of holding the left mouse button to enter the Browse Marker mode.

Clear All BrowseMarks while holding Ctrl key either by a simple or a double click with the left mouse button.

The configuration of the plug-in is stored in your application data directory in the file `default.conf`. If you use the personality feature of CodeBlocks the configuration is read from the file `<personality>.conf`.

2.9 SVN Support

The support of the version control system SVN is included in the CodeBlocks plugin TortoiseSVN. Via the menu 'TortoiseSVN' → 'Plugin settings' you can configure the accessible svn commands in the tab 'Integration' .

Menu integration Add an entry TortoiseSVN with different settings in the menu bar.

Project manger Activate the TortoiseSVN commands in the context menu of the project manager.

Editor Active the TortoiseSVN commands in the context menu of the editor.

In the plugin settings you can configure which svn commands are accessible via the menu or the context menu. The tab integration provides the entry 'Edit main menu' and 'Edit popup menu' to configure these commands.

Note:

The File Explorer in CodeBlocks uses different icon overlays for indicating the svn status. The TortoiseSVN commands are included here in the context menu.

2.10 LibFinder

If you want to use some libraries in your application, you have to configure your project to use them. Such configuration process may be hard and annoying because each library can use custom options scheme. Another problem is that configuration differs on platforms which result in incompatibility between unix and windows projects.

LibFinder provides two major functionalities:

- Searching for libraries installed on your system
- Including library in your project with just few mouse clicks making project platform-independent

2.10.1 Searching for libraries

Searching for libraries is available under 'Plugins' → 'Library finder' menu. It's purpose is to detect libraries installed on your system and store the results inside LibFinder's database (note that these results are not written into CodeBlocks project files). Searching starts with dialogue where you can provide set of directories with installed libraries. LibFinder will scan them recursively so if you're not sure you may select some generic directories. You may even enter whole disks – in such case searching process will take more time but it may detect more libraries (see [Figure 2.10](#) on page 40).

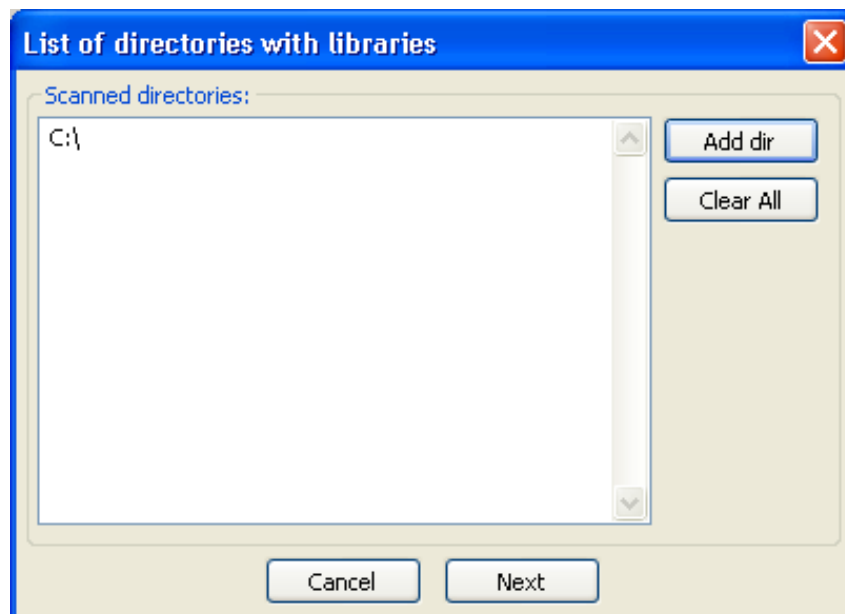


Figure 2.10: List of directories

When LibFinder scans for libraries, it uses special rules to detect presence of library. Each set of rules is located in xml file. Currently LibFinder can search for wxWidgets 2.6/2.8, CodeBlocks SDK and GLFW – the list will be extended in future.

Note:

To get more details on how to add library support into LibFinder, read `src/plugins/contrib/lib_finder/lib_finder/readme.txt` in CodeBlocks sources.

After completing the scan, LibFinder shows the results (see [Figure 2.11](#) on page 41).

In the list you check libraries which should be stored into LibFinder's database. Note that each library may have more than one valid configuration and settings added earlier are more likely to be used while building projects.

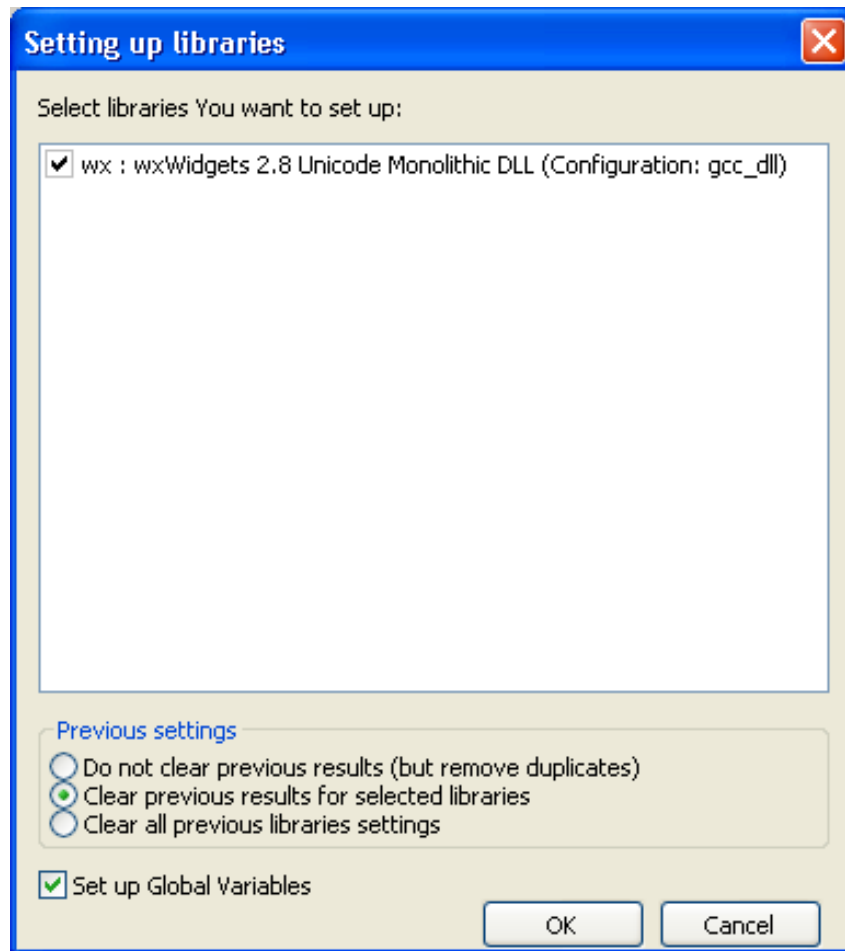


Figure 2.11: Search results

Below the list you can select what to do with results of previous scans:

Do not clear previous results This option works like an update to existing results – it adds new ones and updates those which already exist. This option is not recommended.

Second option (Clear previous results for selected libraries) will clear all results for libraries which are selected before adding new results. This is the recommended option.

Clear all previous library settings when you select this option, LibFinder's database will be cleared before adding new results. It's useful when you want to clean some invalid LibFinder's database.

Another option in this dialogue is 'Set up Global Variables' . When you check this option, LibFinder will try automatically configure Global Variables which are also used to help dealing with libraries.

If you have pkg-config installed on your system (it's installed automatically on most linux versions) LibFinder will also provide libraries from this tool. There is no need to perform scanning for them – they are automatically loaded when CodeBlocks starts.

2.10.2 Including libraries in projects

LibFinder adds extra tab in Project Properties 'Libraries' – this tab shows libs used in project and libs known in LibFinder. To add library into your project, select it in right pane and click < button. To remove library from project, select it on the left pane and click > button (see [Figure 2.12](#) on page 42).

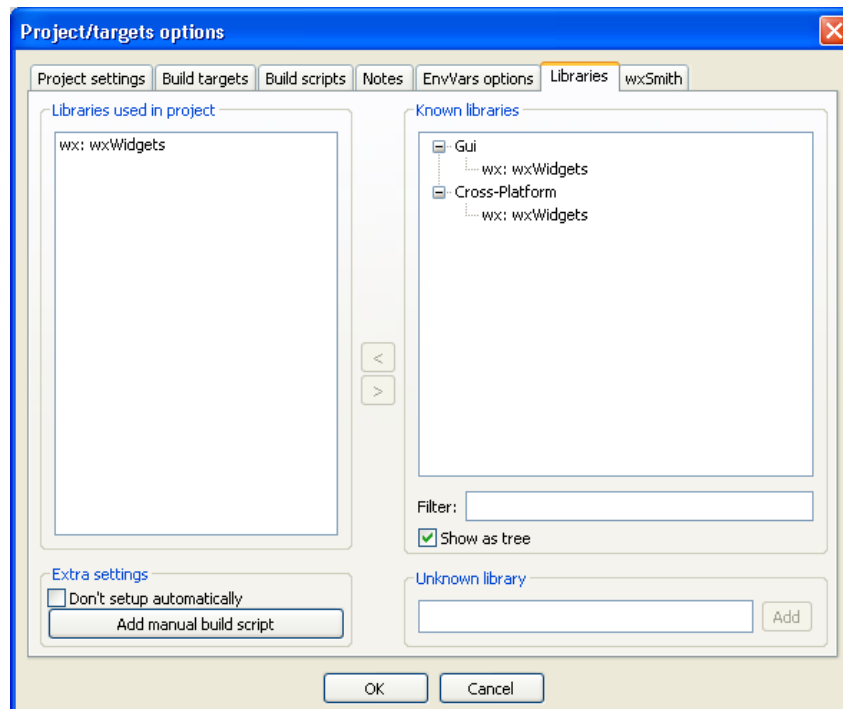


Figure 2.12: Project configuration

You can filter libraries known to LibFinder by providing search filter. The 'Show as Tree' checkbox allows to switch between categorized and uncategorized view.

If you want to add library which is not available in LibFinder's database, you may use 'Unknown Library' field. Note that you should enter library's shortcode (which usually matches global variable name) or name of library in pkg-config. List of suggested shortcodes can be found at [Global Variables](#). Using this option is recommended only when preparing project to be built on other machines where such library exists and is properly detected by LibFinder. You can access a global variable within CodeBlocks like:

```
$ (#GLOBAL_VAR_NAME.include)
```

Checking the 'Don't setup automatically' option will notify LibFinder that it should not add libraries automatically while compiling this project. In such case, LibFinder can be invoked from build script. Example of such script is generated and added to project by pressing 'Add manual build script'.

2.10.3 Using LibFinder and projects generated from wizards

Wizards will create projects that don't use LibFinder. To integrate them with this plugin, you will have to manually update project build options. This can be easily achieved by

removing all library-specific settings and adding library through 'Libraries' tab in project properties.

Such project becomes cross-platform. As long as used libs are defined in LibFinder's database, project's build options will be automatically updated to match platform-specific library settings.

2.11 AutoVersioning

An application versioning plug in that increments the version and build number of your application every time a change has been made and stores it in `version.h` with easy to use variable declarations. Also have a feature for committing changes a la SVN style, a version scheme editor, a change log generator and more ...

2.11.1 Introduction

The idea of the AutoVersioning plugin was made during the development of a pre-alpha software that required the version info and status. Been too busy coding, without time to maintain the version number, just decided to develop a plugin that could do the job with little intervention as possible.

2.11.2 Features

Here is the list of features the plugin covers summarized:

- Supports C and C++.
- Generates and auto increment version variables.
- Software status editor.
- Integrated scheme editor for changing the behavior of the auto incrementation of version values.
- Date declarations as month, date and year.
- Ubuntu style version.
- Svn revision check.
- Change log generator.
- Works on Windows and Linux.

2.11.3 Usage

Just go to 'Project' → 'Autoversioning' menu. A pop up window like this will appear:

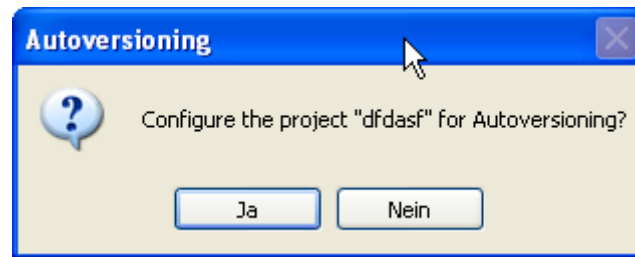


Figure 2.13: Configure project for Autoversioning

When hitting yes on the ask to configure message box, the main auto versioning configuration dialog will open, to let you configure the version info of your project.

After configuring your project for auto versioning, the settings that you entered on the configuration dialog will be stored on the project file, and a `version.h` file will be created. For now, every time that you hit the 'Project' → 'Autoversioning' menu the configuration dialog will popup to let you edit your project version and versioning related settings, unless you don't save the new changes made by the plugin to the project file.

2.11.4 Dialog notebook tabs

2.11.4.1 Version Values

Here you just enter the corresponding version values or let the auto versioning plugin increment them for you (see [Figure 2.14](#) on page 45).

Major Increments by 1 when the minor version reaches its maximum

Minor Increments by 1 when the build number pass the barrier of build times, the value is reset to 0 when it reach its maximum value.

Build Number (also equivalent to Release) - Increments by 1 every time that the revision number is incremented.

Revision Increments randomly when the project has been modified and then compiled.

2.11.4.2 Status

Some fields to keep track of your software status with a list of predefined values for convenience(see [Figure 2.15](#) on page 45).

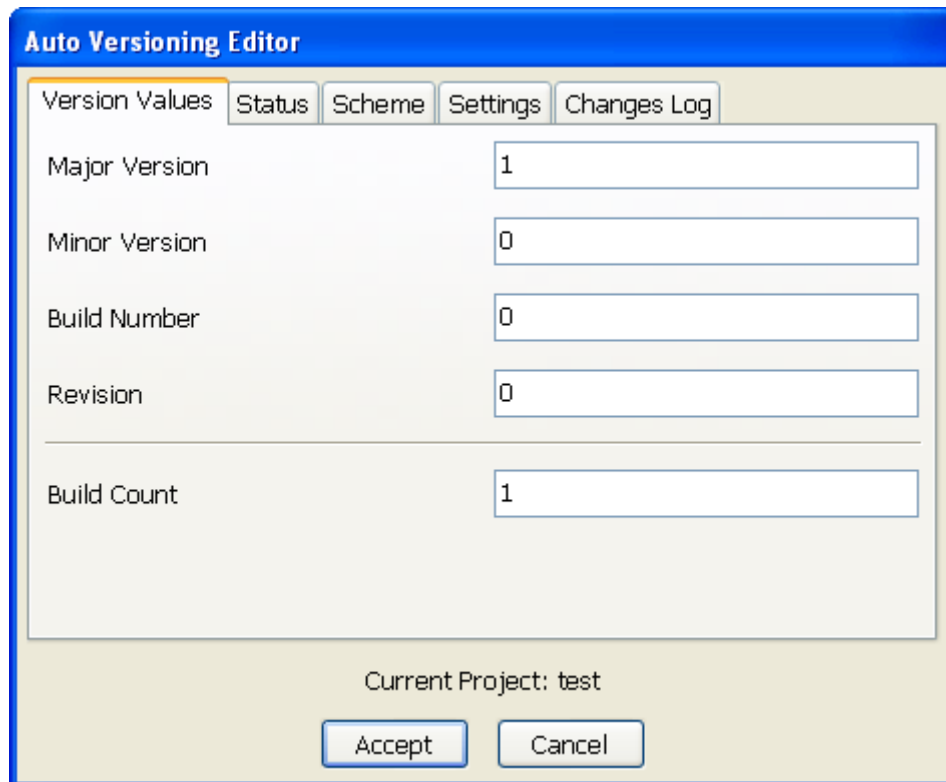
Software Status The typical example should be v1.0 Alpha

Abbreviation Same as software status but like this: v1.0a

2.11.4.3 Scheme

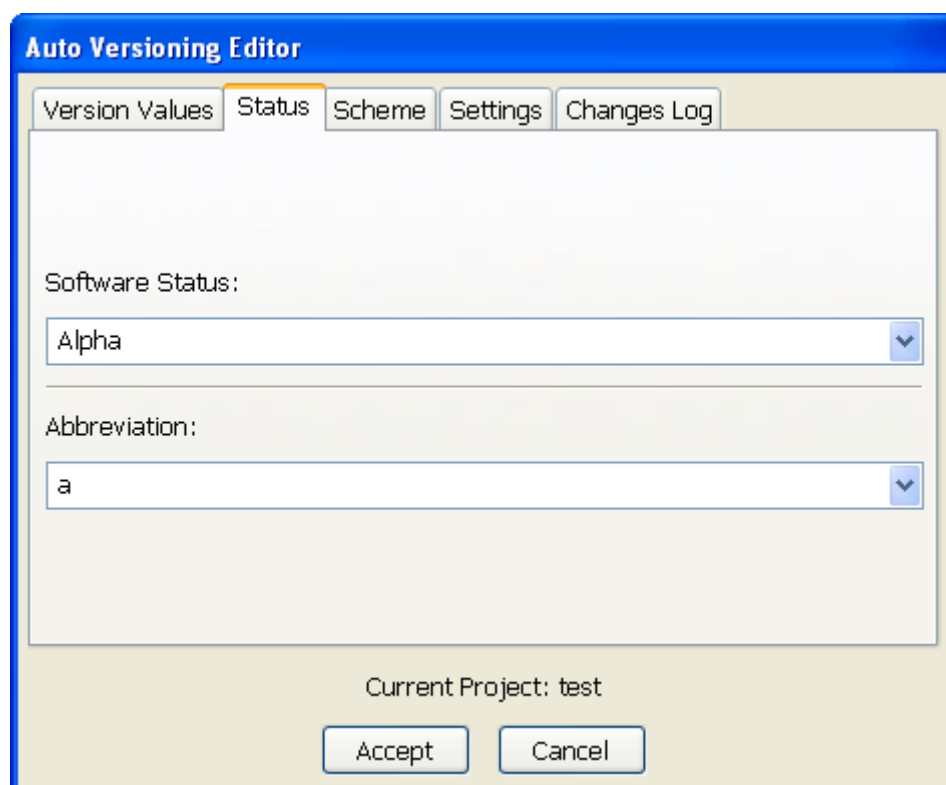
Lets you edit how the plugin will increment the version values (see [Figure 2.16](#) on page 46).

Minor maximum The maximum number that the Minor value can reach, after this value is reached the Major is incremented by 1 and next time project is compiled the Minor is set to 0.



The **Auto Versioning Editor** dialog box has a blue title bar and a light beige background. It features five tabs: **Version Values** (selected), **Status**, **Scheme**, **Settings**, and **Changes Log**. The **Version Values** tab contains five text input fields: **Major Version** (1), **Minor Version** (0), **Build Number** (0), **Revision** (0), and **Build Count** (1). At the bottom, it displays **Current Project: test** and **Accept** / **Cancel** buttons.

Figure 2.14: Set Version Values



The **Auto Versioning Editor** dialog box has a blue title bar and a light beige background. It features five tabs: **Version Values**, **Status** (selected), **Scheme**, **Settings**, and **Changes Log**. The **Status** tab contains two dropdown menus: **Software Status:** (Alpha) and **Abbreviation:** (a). At the bottom, it displays **Current Project: test** and **Accept** / **Cancel** buttons.

Figure 2.15: Set Status of Autoversioning

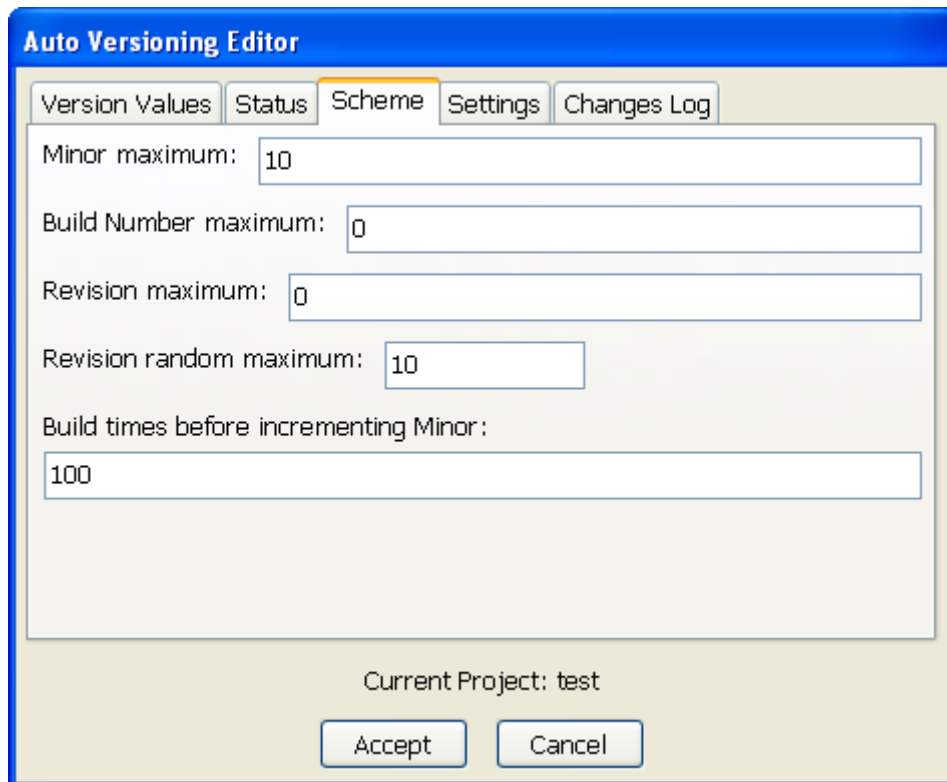


Figure 2.16: Scheme of autoversioning

Build Number maximum When the value is reached, the next time the project is compiled is set to 0. Put a 0 for unlimited.

Revision maximum Same as Build Number maximum. Put a 0 for unlimited

Revision random maximum The revision increments by random numbers that you decide, if you put here 1, the revision obviously will increment by 1.

Build times before incrementing Minor After successful changes to code and compilation the build history will increment, and when it reaches this value the Minor will increment.

2.11.4.4 Settings

Here you can set some settings of the auto versioning behavior (see [Figure 2.17](#) on page 47).

Autoincrement Major and Minor Lets the plugin increments this values by you using the scheme. If not marked only the Build Number and Revision will increment.

Create date declarations Create entries in the `version.h` file with dates and ubuntu style version.

Do Auto Increment This tells the plugin to automatically increment the changes when a modification is made, this incrementation will occur before compilation.

Header language Select the language output of `version.h`

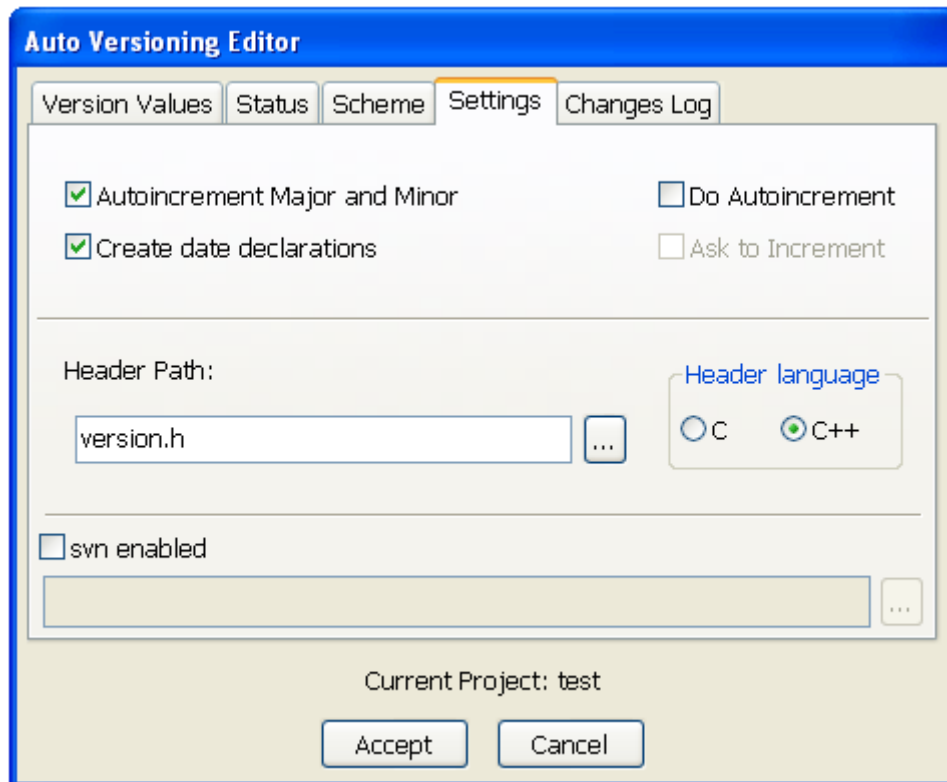


Figure 2.17: Settings of Autoversioning

Ask to increment If marked, Do Auto Increment, it ask you before compilation (if changes has been made) to increment the version values.

svn enabled Search for the svn revision and date in the current folder and generates the correct entries in `version.h`

2.11.4.5 Changes Log

This lets you enter every change made to the project to generate a `ChangesLog.txt` file (see [Figure 2.18](#) on page 48).

Show changes editor when incrementing version Will pop up the changes log editor when incrementing the version.

Title Format A format able title with a list of predefined values.

2.11.5 Including in your code

To use the variables generated by the plugin just `#include <version.h>`. An example code would be like the following:

```
#include <iostream>
#include "version.h"

void main() {
    std::cout<<AutoVersion::Major<<endl;
}
```

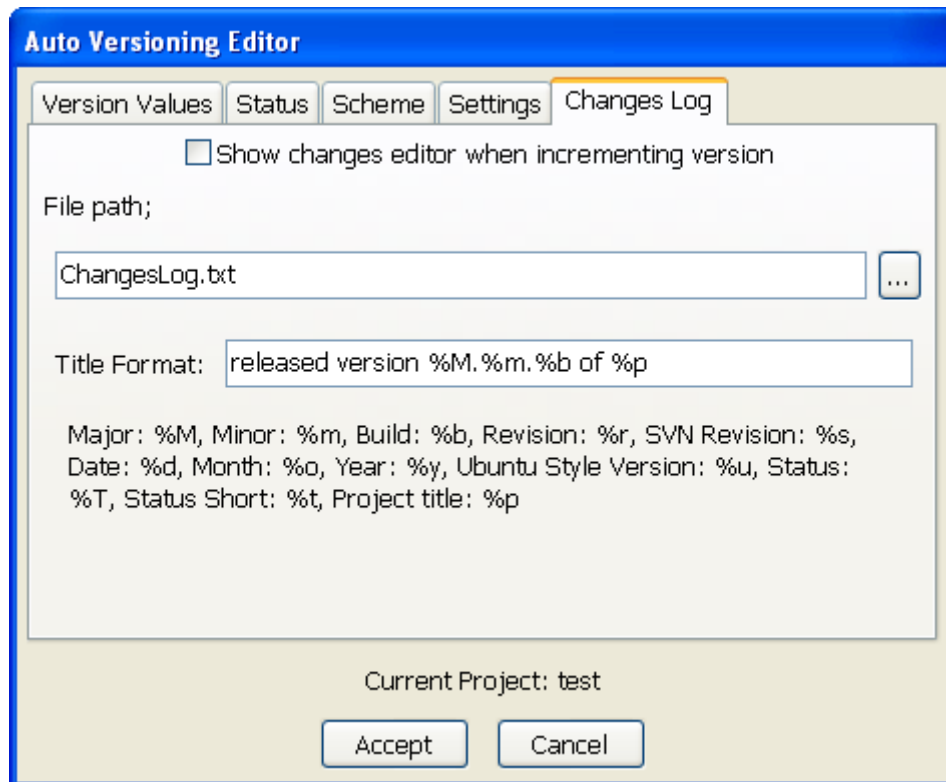



Figure 2.18: Changelog of Autoversioning

2.11.5.1 Output of version.h

The generated header file. Here is a sample content of the file on c++ mode:

```
#ifndef VERSION_H
#define VERSION_H

namespace AutoVersion{

    //Date Version Types
    static const char DATE[] = "15";
    static const char MONTH[] = "09";
    static const char YEAR[] = "2007";
    static const double UBUNTU_VERSION_STYLE = 7.09;

    //Software Status
    static const char STATUS[] = "Pre-alpha";
    static const char STATUS_SHORT[] = "pa";

    //Standard Version Type
    static const long MAJOR = 0;
    static const long MINOR = 10;
    static const long BUILD = 1086;
    static const long REVISION = 6349;

    //Miscellaneous Version Types
    static const long BUILDS_COUNT = 1984;
    #define RC_FILEVERSION 0,10,1086,6349
}
```

```
#define RC_FILEVERSION_STRING "0, 10, 1086, 6349\0"
static const char FULLVERSION_STRING[] = "0.10.1086.6349";

}
#endif //VERSION_h
```

On C mode is the same as C++ but without the namespace:

```
#ifndef VERSION_H
#define VERSION_H

//Date Version Types
static const char DATE[] = "15";
static const char MONTH[] = "09";
static const char YEAR[] = "2007";
static const double UBUNTU_VERSION_STYLE = 7.09;

//Software Status
static const char STATUS[] = "Pre-alpha";
static const char STATUS_SHORT[] = "pa";

//Standard Version Type
static const long MAJOR = 0;
static const long MINOR = 10;
static const long BUILD = 1086;
static const long REVISION = 6349;

//Miscellaneous Version Types
static const long BUILDS_COUNT = 1984;
#define RC_FILEVERSION 0,10,1086,6349
#define RC_FILEVERSION_STRING "0, 10, 1086, 6349\0"
static const char FULLVERSION_STRING[] = "0.10.1086.6349";

#endif //VERSION_h
```

2.11.6 Change log generator

This dialog is accessible from the menu 'Project' → 'Changes Log' . Also if checked Show changes editor when incrementing version on the changes log settings, the window will open to let you enter the list of changes after a modification to the project sources or an incrementation event (see [Figure 2.19](#) on page 50).

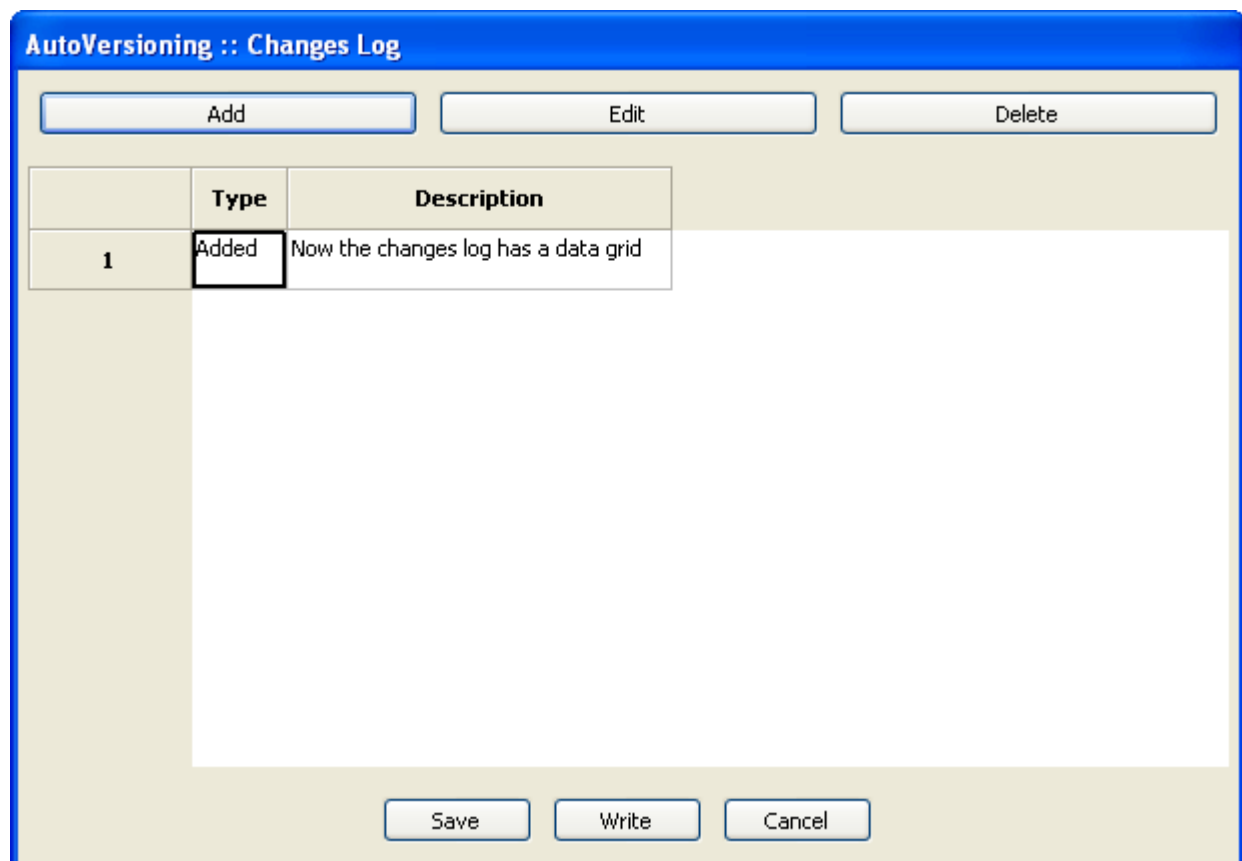


Figure 2.19: Changes for a project

2.11.6.1 Buttons Summary

Add Appends a row in to the data grid

Edit Enables the modification of the selected cell

Delete Removes the current row from the data grid

Save Stores into a temporary file (`changes.tmp`) the actual data for later processing into the changes log file

Write Process the data grid data to the changes log file

Cancel Just closes the dialog without taking any action

Here is an example of the output generated by the plugin to the `ChangesLog.txt` file:

```
03 September 2007
  released version 0.7.34 of AutoVersioning-Linux

  Change log:
    -Fixed: pointer declaration
    -Bug: blah blah

02 September 2007
  released version 0.7.32 of AutoVersioning-Linux
```

Change log:

- Documented some areas of the code
- Reorganized the code for readability

01 September 2007

released version 0.7.30 of AutoVersioning-Linux

Change log:

- Edited the change log window
- If the change log windows is leave blank no changes.txt is modified

2.12 Code statistics

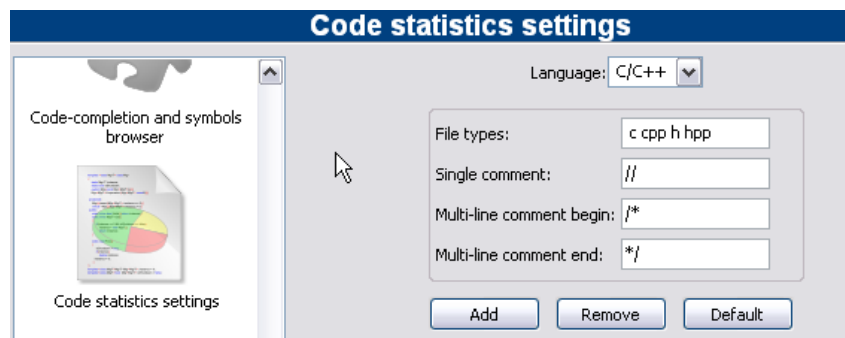


Figure 2.20: Konfiguration für Code Statistik

Based on the entries in the configuration mask, this simple plugin detects the proportions of code, commentaries and blank lines for a project. The evaluation is called via the menu command 'Plugins' → 'Code statistics'.

2.13 Searching Available Source Code

This plugin makes it possible to select a term within the editor and to search for this term via the context menu 'Search at Koders' in the [[↔Koders](#)] database. The dialogue offers the additional possibilities to of filtering for program languages and licences.

This database search will help you find source code originating from other world-wide projects of universities, consortiums and organisations such as Apache, Mozilla, Novell Forge, SourceForge and many others, which can be re-used without having to reinvent the wheel every time. Please observe the licence of the source code in each individual case.

2.14 Code profiler

A simple graphical interface to the GNU GProf Profiler.

2.15 Symbol Table Plugin

This plugin makes it possible to search for symbols in objects and libraries. The options and the path for the command line program nm are defined in the Options tab.

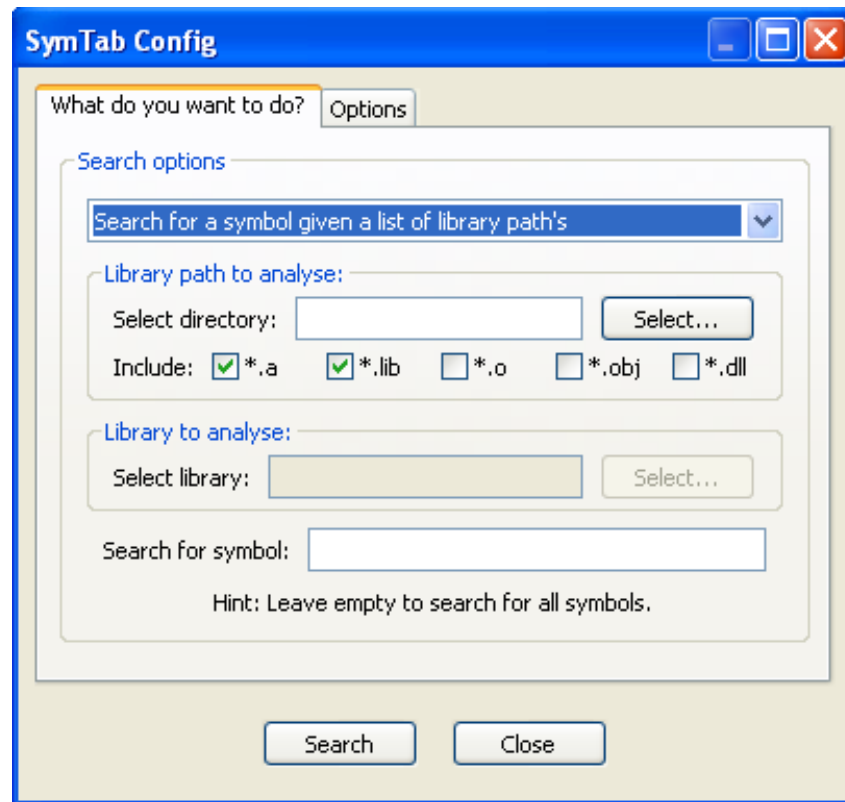


Figure 2.21: Configuring the Symbol Table

Clicking the 'Search' starts the search, the results of the NM program are displayed in a separate window called 'SymTabs Result'. The name of the objects or libraries containing the symbol are listed under the title 'NM's Output'.

3 Variable Expansion

CodeBlocks differentiates between several types of variables. These types serve the purpose of configuring the environment for creating a program, and at the same of improving the maintainability and portability. Access to the CodeBlocks variables is achieved via `$<name>`.

Environment Variable are set during the startup of CodeBlocks. They can modify system environment variables such as `PATH`. This can be useful in cases where a defined environment is necessary for the creation of projects. The settings for environment variables in CodeBlocks are made at 'Settings' → 'Environment' → 'Environment Variables' .

Builtin Variables are predefined in CodeBlocks, and can be accessed via their names (see [section 3.2](#) on page 54 for details).

Command Macros This type of variables is used for controlling the build process. For further information please refer to [section 3.4](#) on page 58.

Custom Variables are user-defined variables which can be specified in the build options of a project. Here you can, for example define your derivative as a variable `MCU` and assign a corresponding value to it. Then set the compiler option `-mcpu=$(MCU)`, and CodeBlocks will automatically replace the content. By this method, the settings for a project can be further parametrised.

Global Variables are mainly used for creating CodeBlocks from the sources or developments of wxWidgets applications. These variables have a very special meaning. In contrast to all others if you setup such a variables and share your project file with others that have **not** setup this GV CodeBlocks will ask the user to setup the variable. This is a very easy way to ensure the 'other developer' knows what to setup easily. CodeBlocks will ask for all path's usually necessary.

3.1 Syntax

CodeBlocks treats the following functionally identical character sequences inside pre-build, post-build, or build steps as variables:

- `$VARIABLE`
- `$(VARIABLE)`
- `${VARIABLE}`
- `%VARIABLE%`

Variable names must consist of alphanumeric characters and are not case-sensitive. Variables starting with a single hash sign (`#`) are interpreted as global user variables (see

section 3.7 on page 58 for details). The names listed below are interpreted as built-in types.

Variables which are neither global user variables nor built-in types, will be replaced with a value provided in the project file, or with an environment variable if the latter should fail.

Note:

Per-target definitions have precedence over per-project definitions.

3.2 List of available built-ins

The variables listed here are built-in variables of CodeBlocks. They cannot be used within source files.

3.2.1 CodeBlocks workspace

`$(WORKSPACE_FILENAME)`, `$(WORKSPACE_FILE_NAME)`, `$(WORKSPACEFILE)`, `$(WORKSPACEFILENAME)`
The filename of the current workspace project (.workspace).

`$(WORKSPACENAME)`, `$(WORKSPACE_NAME)`
The name of the workspace that is displayed in tab Projects of the Management panel.

`$(WORKSPACE_DIR)`, `$(WORKSPACE_DIRECTORY)`, `$(WORKSPACEDIR)`, `$(WORKSPACEDIRECTORY)`
The location of the workspace directory.

3.2.2 Files and directories

`$(PROJECT_FILENAME)`, `$(PROJECT_FILE_NAME)`, `$(PROJECT_FILE)`, `$(PROJECTFILE)`
The filename of the currently compiled project.

`$(PROJECT_NAME)`
The name of the currently compiled project.

`$(PROJECT_DIR)`, `$(PROJECTDIR)`, `$(PROJECT_DIRECTORY)`
The common top-level directory of the currently compiled project.

`$(ACTIVE_EDITOR_FILENAME)`
The filename of the file opened in the currently active editor.

`$(ACTIVE_EDITOR_LINE)`
Return the current line in the active editor.

`$(ACTIVE_EDITOR_COLUMN)`
Return the column of the current line in the active editor.

`$(ACTIVE_EDITOR_DIRNAME)`
the directory containing the currently active file (relative to the common top level path).

`$(ACTIVE_EDITOR_STEM)`
The base name (without extension) of the currently active file.

| | |
|--|---|
| <code>\$(ACTIVE_EDITOR_EXT)</code> | The extension of the currently active file. |
| <code>\$(ALL_PROJECT_FILES)</code> | A string containing the names of all files in the current project. |
| <code>\$(MAKEFILE)</code> | The filename of the makefile. |
| <code>\$(CODEBLOCKS)</code> , <code>\$(APP_PATH)</code> , <code>\$(APPPATH)</code> , <code>\$(APP-PATH)</code> | The path to the currently running instance of CodeBlocks. |
| <code>\$(DATAPATH)</code> , <code>\$(DATA_PATH)</code> , <code>\$(DATA-PATH)</code> | The 'shared' directory of the currently running instance of CodeBlocks. |
| <code>\$(PLUGINS)</code> | The plugins directory of the currently running instance of CodeBlocks. |
| <code>\$(TARGET_COMPILER_DIR)</code> | The compiler installation directory so-called master path. |

3.2.3 Build targets

| | |
|---|--|
| <code>\$(FOOBAR_OUTPUT_FILE)</code> | The output file of a specific target. |
| <code>\$(FOOBAR_OUTPUT_DIR)</code> | The output directory of a specific target. |
| <code>\$(FOOBAR_OUTPUT_BASENAME)</code> | The output file's base name (no path, no extension) of a specific target. |
| <code>\$(TARGET_OUTPUT_DIR)</code> | The output directory of the current target. |
| <code>\$(TARGET_OBJECT_DIR)</code> | The object directory of the current target. |
| <code>\$(TARGET_NAME)</code> | The name of the current target. |
| <code>\$(TARGET_OUTPUT_FILE)</code> | The output file of the current target. |
| <code>\$(TARGET_OUTPUT_BASENAME)</code> | The output file's base name (no path, no extension) of the current target. |
| <code>\$(TARGET_CC)</code> , <code>\$(TARGET_CPP)</code> , <code>\$(TARGET_LD)</code> , <code>\$(TARGET_LIB)</code> | The build tool executable (compiler, linker, etc) of the current target. |

3.2.4 Language and encoding

| | |
|---------------------------|---|
| <code>\$(LANGUAGE)</code> | The system language in plain language. |
| <code>\$(ENCODING)</code> | The character encoding in plain language. |

3.2.5 Time and date

| | |
|--|---|
| <code>\$ (TDAY)</code> | Current date in the form YYYYMMDD (for example 20051228) |
| <code>\$ (TODAY)</code> | Current date in the form YYYY-MM-DD (for example 2005-12-28) |
| <code>\$ (NOW)</code> | Timestamp in the form YYYY-MM-DD-hh.mm (for example 2005-12-28-07.15) |
| <code>\$ (NOW_L)</code> |] Timestamp in the form YYYY-MM-DD-hh.mm.ss (for example 2005-12-28-07.15.45) |
| <code>\$ (WEEKDAY)</code> | Plain language day of the week (for example 'Wednesday') |
| <code>\$ (TDAY_UTC)</code> , <code>\$ (TODAY_UTC)</code> , <code>\$ (NOW_UTC)</code> , <code>\$ (NOW_L_UTC)</code> , <code>\$ (WEEKDAY_UTC)</code> | These are identical to the preceding types, but are expressed relative to UTC. |
| <code>\$ (DAYCOUNT)</code> | The number of the days passed since an arbitrarily chosen day zero (January 1, 2009). Useful as last component of a version/build number. |

3.2.6 Random values

| | |
|--------------------------|---|
| <code>\$ (COIN)</code> | This variable tosses a virtual coin (once per invocation) and returns 0 or 1. |
| <code>\$ (RANDOM)</code> | A 16-bit positive random number (0-65535) |

3.2.7 Operating System Commands

The variable are substituted through the command of the operating system.

| | |
|-----------------------------|---------------------------|
| <code>\$ (CMD_CP)</code> | Copy command for files. |
| <code>\$ (CMD_RM)</code> | Remove command for files. |
| <code>\$ (CMD_MV)</code> | Move command for files. |
| <code>\$ (CMD_MKDIR)</code> | Make directory command. |
| <code>\$ (CMD_RMDIR)</code> | Remove directory command. |

3.2.8 Conditional Evaluation

```
$if(condition){true clause}{false clause}
```

Conditional evaluation will resolve to its true clause if

- condition is a non-empty character sequence other than 0 or false
- condition is a non-empty variable that does not resolve to 0 or false
- condition is a variable that evaluates to true (implicit by previous condition)

Conditional evaluation will resolve to its false clause if

- condition is empty
- condition is 0 or false
- condition is a variable that is empty or evaluates to 0 or false

Note:

Please do note that neither the variable syntax variants `%if (...)` nor `$(if)(...)` are supported for this construct.

Example

For example if you are using several platforms and you want to set different parameters depending on the operating system. In the following code the script commands of `[[]]` are evaluated and the `<command>` will be executed. This could be useful in a post-built step.

```
[[ if (PLATFORM == PLATFORM_MSW) { print (_T("cmd /c")); } else { print (_T("sh ")); } ]]
```

3.3 Script expansion

For maximum flexibility, you can embed scripts using the `[[]]` operator as a special case of variable expansion. Embedded scripts have access to all standard functionalities available to scripts and work pretty much like bash backticks (except for having access to CodeBlocks namespace). As such, scripts are not limited to producing text output, but can also manipulate CodeBlocks state (projects, targets, etc.).

Note:

Manipulating CodeBlocks state should be implemented rather with a pre-build script than with a script.

Example with Backticks

```
objdump -D `find . -name *.elf` > name.dis
```

The expression in backticks returns a list of all executables `*.elf` in any subdirectories. The result of this expression can be used directly by `objdump`. Finally the output is piped to a file named `name.dis`. Thus, processes can be automated in a simple way without having to program any loops.

Example using Script

The script text is replaced by any output generated by your script, or discarded in case of a syntax error.

Since conditional evaluation runs prior to expanding scripts, conditional evaluation can be used for preprocessor functionalities. Built-in variables (and user variables) are expanded after scripts, so it is possible to reference variables in the output of a script.

```
[[ print (GetProjectManager().GetActiveProject().GetTitle()); ]]
```

inserts the title of the active project into the command line.

3.4 Command Macros

| | |
|-----------------------------------|--|
| <code>\$compiler</code> | Access to name of the compiler executable. |
| <code>\$linker</code> | Access to name of the linker executable. |
| <code>\$options</code> | Compiler flags |
| <code>\$link_options</code> | Linker flags |
| <code>\$includes</code> | Compiler include paths |
| <code>\$c</code> | Linker include paths |
| <code>\$libs</code> | Linker libraries |
| <code>\$file</code> | Source file (full name) |
| <code>\$file_dir</code> | Source file directory without file name and file name extension. |
| <code>\$file_name</code> | Source file name without path info and file name extension. |
| <code>\$exe_dir</code> | Directory of executable without file name and file name extension. |
| <code>\$exe_name</code> | File name of executable without path and file name extension. |
| <code>\$exe_ext</code> | File name extension of executable without path and file name. |
| <code>\$object</code> | Object file |
| <code>\$exe_output</code> | Executable output file |
| <code>\$objects_output_dir</code> | Object Output Directory |

3.5 Compile single file

```
$compiler $options $includes -c $file -o $object
```

3.6 Link object files to executable

```
$linker $libdirs -o $exe_output $link_objects $link_resobjects $link_options $libs
```

3.7 Global compiler variables

3.8 Synopsis

Working as a developer on a project which relies on 3rd party libraries involves a lot of unnecessary repetitive tasks, such as setting up build variables according to the local file system layout. In the case of project files, care must be taken to avoid accidentally committing a locally modified copy. If one does not pay attention, this can happen easily for example after changing a build flag to make a release build.

The concept of global compiler variables is a unique new solution for CodeBlocks which addresses this problem. Global compiler variables allow you to set up a project once, with any number of developers using any number of different file system layouts being able to compile and develop this project. No local layout information ever needs to be changed more than once.

3.9 Names and Members

Global compiler variables in CodeBlocks are discriminated from per-project variables by a leading hash sign. Global compiler variables are structured; every variable consists of a name and an optional member. Names are freely definable, while some of the members are built into the IDE. Although you can choose anything for a variable name in principle, it is advisable to pick a known identifier for common packages. Thus the amount of information that the user needs to provide is minimised. The CodeBlocks team provides a list of recommended variables for known packages.

The member `base` resolves to the same value as the variable name uses without a member (alias).

The members `include` and `lib` are by default aliases for `base/include` and `base/lib`, respectively. However, a user can redefine them if another setup is desired.

It is generally recommended to use the syntax `$(#variable.include)` instead of `$(#variable)/include`, as it provides additional flexibility and is otherwise exactly identical in functionality (see [subsection 3.12.1](#) on page 62 and [Figure 3.1](#) on page 60 for details).

The members `cflags` and `lflags` are empty by default and can be used to provide the ability to feed the same consistent set of compiler/linker flags to all builds on one machine. CodeBlocks allows you to define custom variable members in addition to the built-in ones.

3.10 Constraints

- Both set and global compiler variable names may not be empty, they must not contain white space, must start with a letter and must consist of alphanumeric characters. Cyrillic or Chinese letters are not alphanumeric characters. If CodeBlocks is given invalid character sequences as names, it might replace them without asking.
- Every variable requires its base to be defined. Everything else is optional, but the base is absolutely mandatory. If you don't define a the base of a variable, it will not be saved (no matter what other fields you have defined).
- You may not define a custom member that has the same name as a built-in member. Currently, the custom member will overwrite the built-in member, but in general, the behaviour for this case is undefined.
- Variable and member values may contain arbitrary character sequences, subject to the following three constraints:
 - You may not define a variable by a value that references the same variable or any of its members

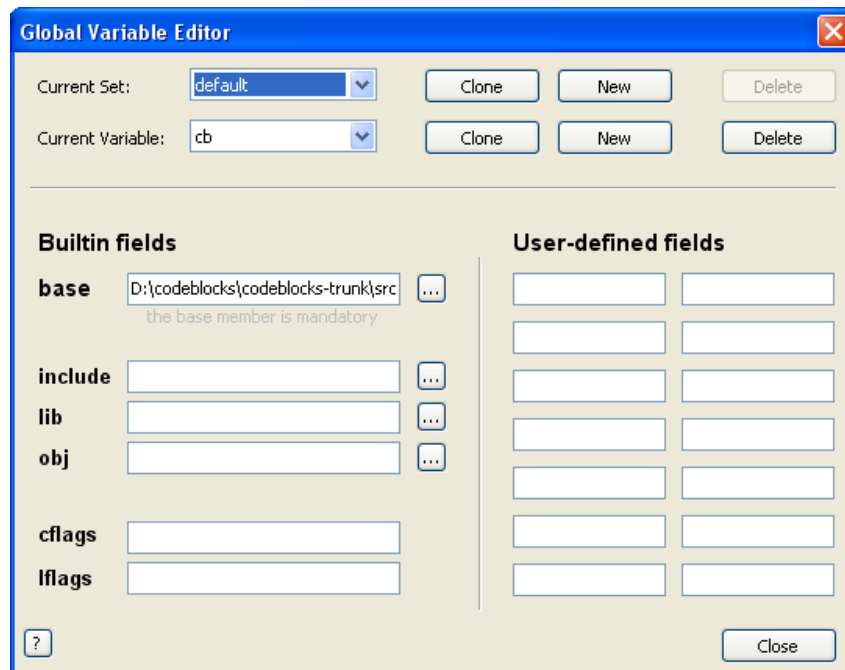


Figure 3.1: Global Variable Environment

- You may not define a member by a value that references the same member
- You may not define a member or variable by a value that references the same variable or member through a cyclic dependency.

CodeBlocks will detect the most obvious cases of recursive definitions (which may happen by accident), but it will not perform an in-depth analysis of every possible abuse. If you enter crap, then crap is what you will get; you are warned now.

Examples

Defining `wx.include` as `$(#wx)/include` is redundant, but perfectly legal. Defining `wx.include` as `$(#wx.include)` is illegal and will be detected by CodeBlocks. Defining `wx.include` as `$(#cb.lib)` which again is defined as `$(#wx.include)` will create an infinite loop.

3.11 Using Global Compiler Variables

All you need to do for using global compiler variables is to put them in your project! Yes, it's that easy.

When the IDE detects the presence of an unknown global variable, it will prompt you to enter its value. The value will be saved in your settings, so you never need to enter the information twice.

If you need to modify or delete a variable at a later time, you can do so from the settings

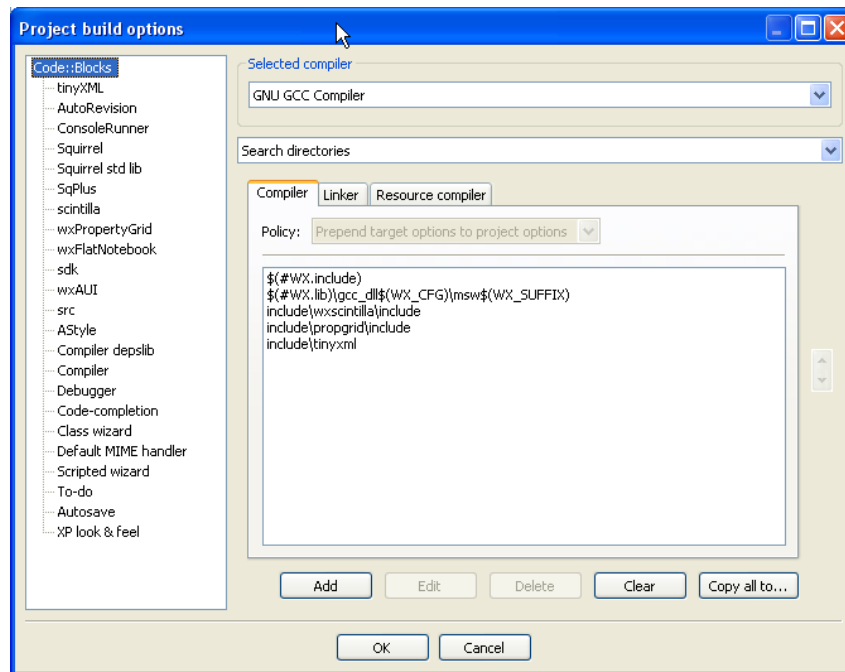


Figure 3.2: Global Variables

menu.

Example

The above image shows both per-project and global variables. `WX_SUFFIX` is defined in the project, but `WX` is a global user variable.

3.12 Variable Sets

Sometimes, you want to use different versions of the same library, or you develop two branches of the same program. Although it is possible to get along with a global compiler variable, this can become tedious. For such a purpose, CodeBlocks supports variable sets. A variable set is an independent collection of variables identified by a name (set names have the same constraints as variable names).

If you wish to switch to a different set of variables, you simply select a different set from the menu. Different sets are not required to have the same variables, and identical variables in different sets are not required to have the same values, or even the same custom members.

Another positive thing about sets is that if you have a dozen variables and you want to have a new set with one of these variables pointing to a different location, you are not required to re-enter all the data again. You can simply create a clone of your current set, which will then duplicate all of your variables.

Deleting a set also deletes all variables in that set (but not in another set). The `default` set is always present and cannot be deleted.

3.12.1 Custom Members Mini-Tutorial

As stated above, writing `$(#var.include)` and `$(#var)/include` is exactly the same thing by default. So why would you want to write something as unintuitive as `$(#var.include)`?

Let's take a standard Boost installation under Windows for an example. Generally, you would expect a fictional package ACME to have its include files under ACME/include and its libraries under ACME/lib. Optionally, it might place its headers into yet another subfolder called acme. So after adding the correct paths to the compiler and linker options, you would expect to **#include** `<acme/acme.h>` and link to `libacme.a` (or whatever it happens to be).

URL catalog

[↔7Z] 7z zip homepage.

<http://www.7-zip.org>

[↔BERLIOS] Codeblocks at berlios.

<http://developer.berlios.de/projects/codeblocks/>

[↔FORUM] Codeblocks forum.

<http://forums.codeblocks.org/>

[↔WIKI] Codeblocks wiki.

http://wiki.codeblocks.org/index.php?title=Main_Page/

[↔CODEBLOCKS] Codeblocks homepage.

<http://www.codeblocks.org/>

[↔GCC] GCC home page.

<http://gcc.gnu.org/>

[↔HIGHTEC] HighTec homepage.

<http://www.hightec-rt.com/>

[↔Koders] Koders homepage.

<http://www.koders.com/>

[↔TriCore] TriCore homepage.

<http://www.infineon.com/tricore/>

[↔TortoiseSVN] TriCore homepage.

<http://tortoisesvn.net/>

[↔Subversion] TriCore homepage.

<http://subversion.tigris.org/>

[↔Wxwidgets] WxWidgets homepage.

<http://www.wxwidgets.org/>

[↔Wxcode] WxCode homepage.

<http://wxcode.sourceforge.net/>

[↔Scripts] Scripting commands.

http://wiki.codeblocks.org/index.php?title=Scripting_commands/