

# 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 Projektverwaltung

Die Dokumentation für [Kapitel 3](#) auf Seite 55 und ?? auf Seite ?? sind offizielle Dokumentationen der CodeBlocks Wiki-Seite und nur in englischer Sprache verfügbar.

Die nachfolgende Abbildung zeigt den Aufbau der CodeBlocks Oberfläche.

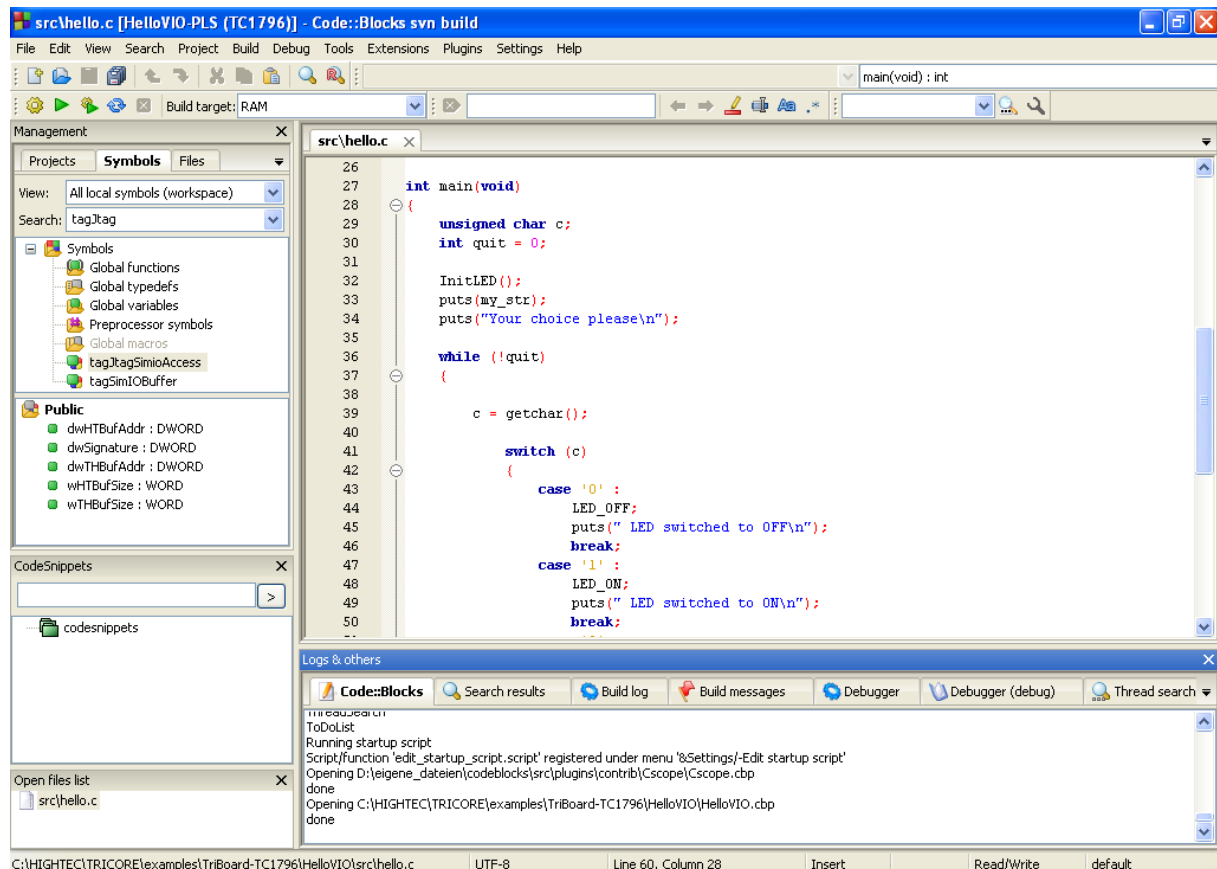


Abbildung 1.1: IDE CodeBlocks

**Management** Diese Fenster enthält die Ansicht 'Projects', im nachfolgenden als Projektansicht bezeichnet. In dieser werden die in CodeBlocks aktuell geöffneten Projekte angezeigt. In dem Management Fenster erhält man im Reiter 'Symbols' die Anzeige von Symbolen, Variablen etc.

**Editor** In der obigen Abbildung ist eine Quelle `hello.c` mit Syntaxhighlighting im Editor geöffnet.

**Open files list** Zeigt die Liste der im Editor geöffneten Dateien an, hier `hello.c`.

**CodeSnippets** Lässt sich über das Menü 'View' → 'CodeSnippets' anzeigen. Hier können Textbausteine, Verknüpfungen auf Dateien und Urls verwaltet werden.

**Logs & others** Fenster zur Ausgabe von Suchergebnisse, Logmeldung eines Compilers etc.

Die Statusbar gibt einen Überblick der folgenden Einstellungen:

- Vollständiger Pfad einer im Editor angezeigten Datei.
- Im Editor wird das vom Betriebssystem standardmäßig verwendete Character Encoding übernommen und mit der Einstellung **default** angezeigt.
- Aktuelle Zeilen- und Spaltennummer der Cursorposition im Editor.
- Über die Tastatur eingestellte Modus für das Einfügen von Text (Insert oder Overwrite).
- Aktuelle Status einer Datei. Für eine geänderte Datei steht der Eintrag auf **Modified** ansonsten ist dieser Eintrag leer.
- Die für eine Datei eingestellte Berechtigung. Eine schreibgeschützte Datei wird als **Read only** in der Statusbar gekennzeichnet. Im Fenster 'Open files list' werden solche mit einem Schloss als Icon Overlay markiert.

**Hinweis:**

Im aktiven Editor kann über das Kontextmenü Properties im Reiter 'General' die Option 'File is read-only' aktiviert werden. Dies bewirkt das die Datei nur innerhalb CodeBlocks schreibgeschützt ist, jedoch bleibt die Lese- und Schreibberechtigung auf der Festplatte unverändert.

- Falls CodeBlocks mit der Kommandozeilenoption `--personality=<profile>` gestartet wird, steht in der Statusbar das aktuell verwendete Profil, ansonsten wird hier **default** angezeigt. Die Einstellungen von CodeBlocks werden in der gleichnamigen Konfigurationsdatei `<personality>.conf` gespeichert.

CodeBlocks bietet eine sehr flexible und umfassende Projektverwaltung. Der folgende Text geht nur auf einige Besonderheiten der Projektverwaltung ein.

## 1.1 Projektansicht

In CodeBlocks werden Quellen und die Einstellungen für den Buildprozess in einer Projektdatei `<name>.cbp` gespeichert. Ein Projekt besteht typischerweise aus C/C++ Quellen und zugehörige Header Dateien. Ein neues Projekt legen Sie am einfachsten an, indem Sie das Menü 'File' → 'Project' ausführen und einen Wizard auswählen. Anschließend können Sie im Management Fenster über das Kontextmenü 'Add files' Dateien zum Projekt hinzufügen. In CodeBlocks werden die Projektdateien abhängig von ihrer Dateiendung in Kategorien verwalten. Die voreingestellten Kategorien sind für

**Sources** Unter der Kategorie **Sources** werden Quellen z.B. mit den Endungen `*.c;*.cpp`; aufgelistet.

**ASM Sources** Unter der Kategorie **ASM Sources** werden Quellen z.B. mit den Endungen `*.s;*.S;*.ss;*.asm` aufgelistet.

**Headers** Unter der Kategorie **Headers** werden Dateien z.B. mit den Endungen `*.h`; angezeigt.

**Resources** Unter der Kategorie **Resources** werden z.B. Dateien `*.res`; `*.xrc`; für die Beschreibung von Layout von wxWidgets Fenster gelistet. Für die Anzeigen dieser Dateitypen dient im Manangement Fenster der Reiter 'Resources'.

Die Einstellungen für Typen und Kategorien von Dateien können über das Kontextmenü 'Project tree' → 'Edit file types & categories' angepasst werden. Dabei können auch eigene Kategorien für Dateiendungen angelegt werden. Wenn Sie z.B. Linkerskripte mit der Endung `*.ld` unter der Kategorie **Linkerscript** anzeigen möchten, legen Sie einfach eine neue Kategorie an.

**Hinweis:**

Wenn Sie im Kontextmenü 'Project tree' → 'Categorize by file types' deaktivieren, wird die Anzeige in Kategorien aufgehoben und die Dateien erscheinen wie sie im Dateisystem abgelegt sind.

## 1.2 Notizen für Projekte

In CodeBlocks können zu Projekten sogenannte Notes hinterlegt werden. Diese sollten eine Kurzbeschreibung oder Hinweise für das jeweilige Projekt enthalten. Durch Anzeige dieser Information beim Öffnen des Projektes bekommen andere Bearbeiter einen schnellen Überblick. Die Anzeige von Notes kann bei den Properties eines Projektes im Reiter Notes aktiviert bzw. deaktiviert werden.

## 1.3 Projektvorlagen

CodeBlocks wird mit einer Vielzahl von Projektvorlagen ausgeliefert, die beim Anlegen eines neuen Projektes angezeigt werden. Es ist aber auch möglich, eigene Vorlagen zu speichern und somit eigene Vorgaben für Compilerschalter, wie zu verwendete Optimierung, maschinenspezifische Schalter etc. in Vorlagen zusammenzufassen. Diese werden im Verzeichnis **Dokumente und Einstellungen\<user>\Anwendungsdaten\codeblocks\UserTemplates** abgelegt. Wenn die Vorlagen für alle Benutzer zugänglich sein sollen, müssen die Vorlagen in zugehöriges Verzeichnis der CodeBlocks Installation kopiert werden. Diese Vorlagen erscheinen dann beim nächsten Start von CodeBlocks unter 'New' → 'Project' → 'User templates'.

**Hinweis:**

Die verfügbaren Vorlagen im Project Wizard können durch Auswahl mit der rechten Maustaste bearbeitet werden.

## 1.4 Projekte aus Build Targets erstellen

In Projekten ist es notwendig unterschiedliche Varianten eines Projektes vorzuhalten. Varianten werden als Build Target bezeichnet. Diese unterscheiden sich in der Regel durch unterschiedliche Compileroptionen, Debug-Information und Auswahl von Dateien. Ein Build Target kann auch in ein eigenständiges Projekt ausgelagert werden, dafür selektieren Sie in 'Project' → 'Properties' den Reiter 'Build Targets' die Variante und wählen Sie Schaltfläche 'Create project from target' (siehe [Abbildung 1.2](#) auf Seite 4).

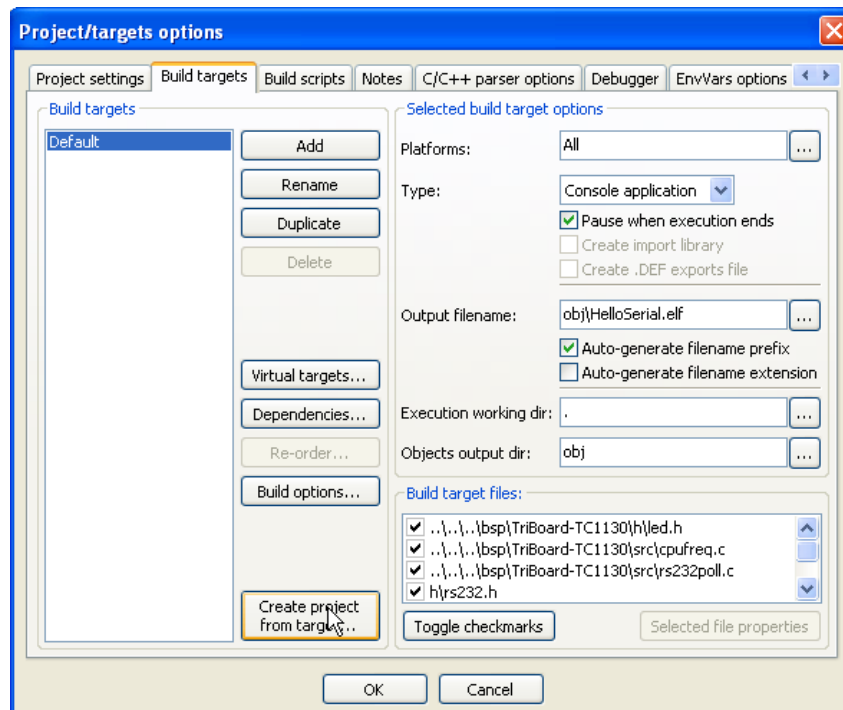


Abbildung 1.2: Build Targets

## 1.5 Virtual Targets

Mit sogenannten Virtual Targets können Projekte in CodeBlocks weiter strukturiert werden. Eine häufige Projektstruktur besteht aus zwei Build Targets. Einem Target 'Debug' mit Debuginformation und einem anderen Target 'Release' ohne diese Information. Durch Hinzufügen von Virtual Targets unter 'Project' → 'Properties' → 'Build Targets' können einzelne Build Targets zusammengefasst werden. So kann zum Beispiel ein Virtual Target 'All' die Targets Debug und Release gleichzeitig erzeugen. Die Virtual Targets werden auch in der Symbolleiste des Compilers unter Build Targets angezeigt.

## 1.6 Pre- und Postbuild Schritte

CodeBlocks ermöglicht es, weitere Arbeitsschritte vor oder nach der Compilierung eines Projektes durchzuführen. Die Arbeitsschritte werden als Prebuilt bzw. Postbuilt Step bezeichnet. Typische Postbuilt Steps sind:

- Erzeugung eines Intel Hexformats aus einem fertigen Objekt
- Manipulation von Objekten mit `objcopy`
- Generierung von Dumpdateien mit `objdump`

### Beispiel

Erzeugung einer Disassembly aus einem Objekt unter Windows. Die Umlenkung in eine Datei erfordert den Aufruf der `cmd` mit der Option `/c`.

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

Ein weiteres Beispiel für ein Postbuilt Step kann die Archivierung eines Projektes sein. Hierzu erstellen Sie ein Build Target 'Archive' und tragen im Postbuilt Step folgende Anweisung ein

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

Mit diesem Befehl wird das aktive Projekt und seine Quellen, Header und Objekte als Zip-Datei gepackt. Dabei werden über die Built-in Variablen `$(PROJECT_NAME)` und `$(TODAY)`, der Projektname und das aktuelle Datum extrahiert (siehe [Abschnitt 3.2](#) auf Seite 56). Im Verzeichnis des Projektes liegt dann nach Ausführen des Targets 'Archive' die gepackte Datei.

In dem Verzeichnis `share/codeblocks/scripts` finden Sie einige Beispiele für Skripte. Ein Skript kann über das Menü 'Settings' → 'Scripting' hinzugefügt und in ein Menü eingetragen werden. Wenn Sie ein Skript z.B. `make_dist` über ein Menü ausführen, werden alle Dateien, die zum einem aktiven Projekt gehören in ein Archiv `<project>.tar.gz` komprimiert.

## 1.7 Hinzufügen von Scripts in Build Targets

CodeBlocks bieten die Möglichkeit, Aktionen die vom Benutzer in Menüs ausgeführt werden, auch in Skripten zu verwenden. Mit dem Skript entsteht somit ein zusätzlicher Freiheitsgrad um die Generierung Ihres Projektes zu steuern.

### Hinweis:

Ein Skript kann auch bei einem Build Target angegeben werden.

## 1.8 Workspace und Project Dependencies

In CodeBlocks können Sie mehrere Projekte geöffnet halten. Durch speichern der geöffneten Projekte über 'File' → 'Save workspace' werden diese in einem Arbeitsbereich unter `<name>.workspace` zusammengefasst. Wenn Sie beim nächsten Start von CodeBlocks den Arbeitsbereich `<name>.workspace` öffnen erscheinen wieder alle Projekte.

Komplexe Softwaresysteme bestehen aus Komponenten, die in unterschiedlichen CodeBlocks Projekten verwaltet werden. Des weiteren existieren bei der Generierung von solchen Softwaresystemen oftmals Abhängigkeiten zwischen diesen Projekten.

### Beispiel

Ein Projekt A enthält zentrale Funktionen, die auch anderen Projekten in Form einer Bibliothek zugänglich gemacht werden. Wenn nun diese Quellen eines Projektes geändert werden, muss die Bibliothek neu erzeugt werden. Damit die Konsistenz zwischen einem Projekt B, das die Funktionen verwendet und dem Projekt A, das die Funktionen implementiert, gewahrt bleibt, muss Projekt B von Projekt A abhängen. Die Information für die Abhängigkeit von Projekten wird im jeweiligen Workspace gespeichert, damit jedes Projekt weiterhin einzeln erzeugt werden kann. Durch die Verwendung von Abhängigkeiten kann auch die Reihenfolge bei der Generierung von Projekten gesteuert werden. Die Abhängigkeiten für Projekte werden über den Menüeintrag 'Project' → 'Properties' und Auswahl der Schaltfläche 'Project's dependencies' gesetzt.

## 1.9 Einbinden von Assembler Dateien

In der Projektansicht (Project View) im Fenster Management werden Assembler Dateien im Kategorie **ASM Sources** aufgeführt. Die Anzeige von Dateien und Kategorien kann vom Benutzer festgelegt werden (siehe [Abschnitt 1.1](#) auf Seite 2). Durch einen Rechtsklick einer der gelisteten Assembler Dateien erhält man ein Kontextmenü. Darin öffnet der Befehl 'Properties' ein neues Fenster. Klicken Sie darin auf den Reiter 'Build' und aktivieren Sie die beiden Felder 'Compile file' und 'Link file'. Wechseln Sie nun auf den Reiter 'Advanced' und führen Sie folgende Schritte durch:

1. 'Compiler variable' auf CC setzen
2. Den Compiler unter 'For this compiler' auswählen
3. 'Use custom command to build this file' anwählen
4. Inhalt im Fenster eingeben:

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

Dabei sind die CodeBlocks Variablen durch **\$** gekennzeichnet (siehe [Abschnitt 3.4](#) auf Seite 60). Diese werden automatisch ersetzt, so dass Sie lediglich die Assembleroption **<asopt>** durch Ihre Einstellungen ersetzen brauchen.

## 1.10 Editor und Hilfsmittel

### 1.10.1 Default Code

Durch vorgegebene Coding Rules im Unternehmen müssen Quelldateien einen einheitlichen Aufbau vorweisen. CodeBlocks bietet die Möglichkeit, beim Anlegen von neuen C/C++ Quellen und Header einen vorgegebenen Inhalt am Anfang einer Datei automatisiert einzufügen. Die vorgegebene Inhalt wird als Default Code bezeichnet. Die Einstellung hierfür kann unter 'Settings' → 'Editor' Default Code vorgenommen werden. Wenn eine neue Datei angelegt wird, dann wird eine Makroauflösung von Variablen, die zum Beispiel über das Menü 'Settings' → 'Global variables' definiert wurden, durchgeführt. Eine neue Datei erzeugen Sie über das Menü 'File' → 'New' → 'File'.

#### Beispiel

```

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

```

## 1.10.2 Abbreviation

Durch Definition von Abkürzung in CodeBlocks kann einiges an Schreibarbeit und Zeit gespart werden. Hierzu werden in 'Settings' → 'Editor' sogenannte Abbreviations unter dem Namen <name> angelegt, die über das Tastenkürzel Ctrl-J aufgerufen werden (siehe [Abbildung 1.3](#) auf Seite 7).

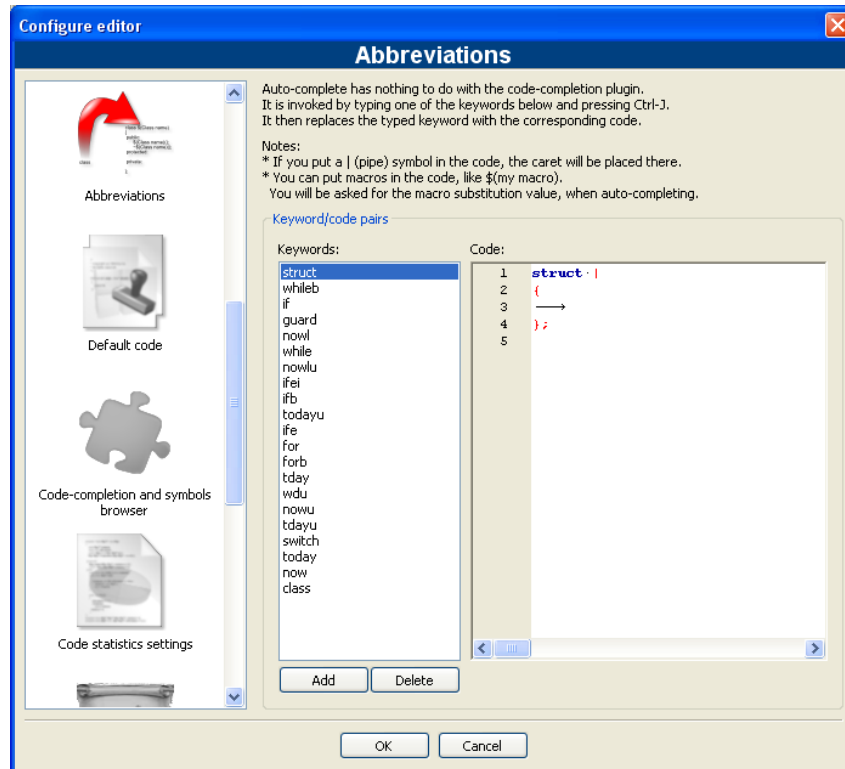


Abbildung 1.3: Definition von Abkürzungen

Durch Einfügen von Variablen \$(NAME) in den Abkürzungen ist auch eine Parametrisierung möglich.

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

Bei Aufruf der Abkürzung <name> im Quelltext und Ausführen von Ctrl-J, wird der Inhalt der Variablen abgefragt und eingefügt.

## 1.10.3 Personalities

CodeBlocks Einstellungen werden als Anwendungsdaten im Verzeichnis `codeblocks` in einer Datei <user>.conf gespeichert. Diese Konfigurationsdatei enthält Informationen wie beispielsweise zuletzt geöffnete Projekte, Einstellungen für Editor, Anzeige von Symbolleisten etc. Standardmäßig ist die Personality 'default' eingestellt, so dass die Konfiguration in der Datei `default.conf` abgelegt ist. Wenn CodeBlocks mit dem Parameter `--personality=myuser` in der Kommandozeile aufgerufen wird, werden die Einstellungen



in der Datei `myuser.conf` gespeichert. Falls das Profil nicht bereits existiert, wird es automatisch angelegt. Durch diese Vorgehensweise können für unterschiedliche durchzuführende Arbeitsschritte auch zugehörige Profile gespeichert werden. Wenn Sie CodeBlocks mit dem zusätzlichen Parameter `--personality=ask` starten erscheint ein Auswahldialog für die verfügbaren Profile.

**Hinweis:**

Der Name des aktuell verwendeten Profils/Personality wird rechts in der Statusbar angezeigt.

### 1.10.4 Konfigurationsdateien

Die Einstellungen für CodeBlocks werden im Profil `default.conf` im Ordner `codeblocks` in Ihren Anwendungsdaten gespeichert. Bei Verwendung von personalities (siehe [Unterabschnitt 1.10.3](#) auf Seite 7) werden die Konfiguration in der Datei `<personality>.conf` abgelegt.

Mit dem Werkzeug `cb_share.conf`, aus dem CodeBlocks Installationsverzeichnis, können diese Einstellungen verwaltet und gesichert werden.

Falls Sie Standardeinstellung für mehrere Benutzer eines PCs vorgeben möchten, muss die Konfigurationsdatei `default.conf` im Ordner `\Dokumente und Einstellungen\Default User\Anwendungsdaten\codeblocks` abgelegt sein. Beim ersten Start von CodeBlocks werden die Voreinstellungen aus 'Default User' in die Anwendungsdaten der aktuellen Benutzers kopiert.

Zur Erzeugung einer portablen Version von CodeBlocks auf einem USB-Stick gehen Sie wie folgt vor. Kopieren Sie die CodeBlocks Installation auf einen USB-Stick und legen Sie die Konfigurationsdatei `default.conf` in dieses Verzeichnis. Die Konfiguration wird als globale Einstellung verwendet. Bitte achten Sie darauf, dass die Datei schreibbar sein muss, damit Änderungen in der Konfiguration auch gespeichert werden können.

### 1.10.5 Navigieren und Suchen

In CodeBlocks existieren unterschiedliche Möglichkeiten zum schnellen Navigieren zwischen Dateien und Funktionen. Eine typische Vorgehensweise ist das Setzen von Lesezeichen (Bookmarks). Durch Betätigen des Tastenkürzel (Ctrl-B) wird ein Lesezeichen in einer Quelldatei gesetzt bzw. gelöscht. Mit (Alt-PgUp) wird zum vorherigen Lesezeichen gesprungen und mit (Alt-PgDn) zum nächsten gewechselt.

In der Projektansicht können Sie durch Auswählen eines Projektes oder im gesamten Workspace über das Kontextmenü 'Find file' in einem Dialog einen Dateinamen angeben. Dieser wird anschließend in der Projektansicht markiert und durch Eingabe mit Return im Editor geöffnet (siehe [Abbildung 1.4](#) auf Seite 9).

Für das schnelle Navigieren zwischen Header/Quelle Dateien bietet CodeBlocks folgende Möglichkeiten

1. Cursor auf Zeile setzen wo Header inkludiert wird und über Kontextmenü 'open include file' öffnen (siehe [Abbildung 1.5](#) auf Seite 9)

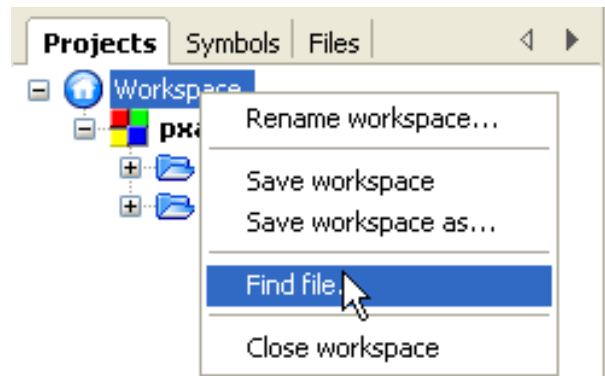


Abbildung 1.4: Suche nach Dateien

2. Umschalten zwischen Quelle und Header über Kontextmenü 'Swap header/source'
3. Markieren eines Begriffes z.B. eines Defines in einer Datei und Aufruf des Kontextmenü 'Find declaration'

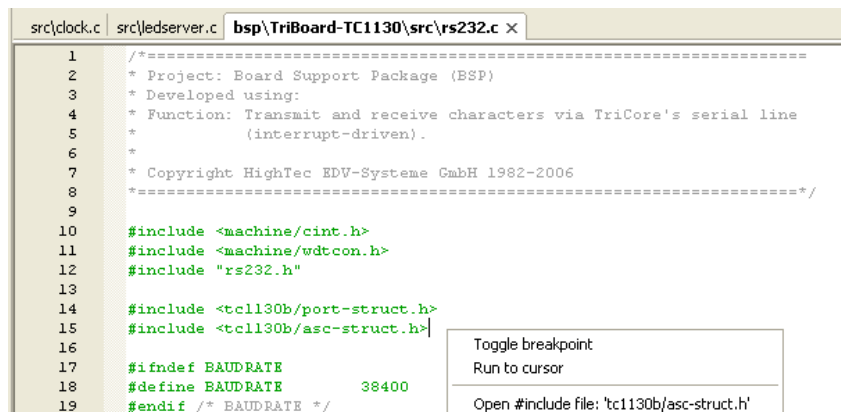


Abbildung 1.5: Öffnen einer Header Datei

CodeBlocks bietet verschiedene Möglichkeiten für die Suche in einer Datei oder in Verzeichnissen. Mit dem 'Search' → 'Find' (Ctrl-F) oder 'Find in Files' (Ctrl-Shift-F) öffnet sich der Dialog für die Suche.

Eine weitere komfortable Funktion bietet das Tastenkürzel Alt-G und Ctrl-Alt-G. Der sich öffnende Dialog erlaubt die Auswahl von Dateien/Funktionen und springt anschließend an die Implementierung der Funktion (siehe [Abbildung 1.6](#) auf Seite 10) bzw. öffnet die ausgewählte Datei. Als Eingabe werden auf Wildcards \* oder ? etc. für eine inkrementelle Suche unterstützt.

#### Hinweis:

Mit dem Tastenkürzel Ctrl-PAGEUP springen Sie an die vorherige Funktion und mit Ctrl-PAGEDOWN zur nächsten Funktion.

Wenn Sie sich im Editor Fenster befinden, öffnet sich mit Ctrl-Tab ein zusätzliches Open Files Dialog und es kann zwischen den Einträgen der zu öffnenden Dateien gewechselt werden. Bei gedrückter Ctrl-Taste kann eine Auswahl auf unterschiedliche Weise erfolgen:

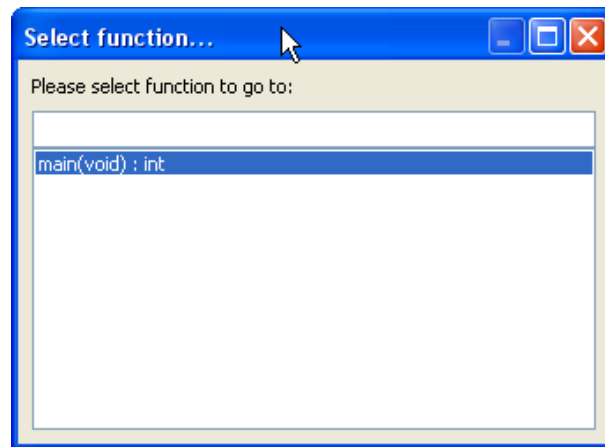


Abbildung 1.6: Suche nach Funktionen

1. Mit der linken Maustaste einen Eintrag anklicken und es öffnet sich die Datei.
2. Betätigen der Tab-Taste wechselt zwischen den Einträgen. Durch Loslassen der Ctrl-Taste wird die ausgewählte Datei geöffnet.
3. Wenn die Maus über die Einträge bewegt werden, dann erscheint die aktuelle Auswahl farblich hervorgehoben. Beim Loslassen wird die ausgewählte Datei geöffnet.
4. Wenn der Mauszeiger außerhalb der farblich Hervorhebung steht, dann kann über das Mausrad eine Auswahl getroffen werden. Beim Loslassen der Ctrl-Taste wird die ausgewählte Datei geöffnet.

Eine häufige Arbeitsweise bei der Entwicklung von Software ist jedoch, dass man sich durch ein Satz von Funktion hangelt, die in unterschiedlichen Dateien implementiert sind. Durch das Plugin Browse Tracker zeigt mit dem Fenster 'Browsed Tabs' eine Liste in der Reihenfolge wie Dateien selektiert wurden. Somit können Sie komfortabel zwischen den Aufrufen navigieren (siehe [Abschnitt 2.8](#) auf Seite 40).

In CodeBlocks aktivieren Sie die Anzeige von Zeilennummern in 'Settings' → 'General Settings' im Feld 'Show line numbers'. Mit dem Tastenkürzel Ctrl-G oder über das Menü 'Search' → 'Goto line' springen Sie an die gewünschte Zeile.

**Hinweis:**

Sie können auch im Editor einen Begriff mit gedrückter Ctrl Taste markieren und dann über das Kontextmenü nach diesem Begriff z.B. in Goolge suchen.

### 1.10.6 Symbolansicht

Für das Navigieren über Funktionen oder Variablen bietet das Management Fenster in CodeBlocks eine Baumansicht für Symbole von C/C++ Quellen. Dabei lässt sich der Gültigkeitsbereich (Scope) der Ansicht auf die aktuelle Datei oder Projekt oder den gesamten Arbeitsbereich einstellen.

**Hinweis:**

Wenn Sie einen Suchbegriff bzw. Symbolnamen in die Eingabemaske Search des Symbol Browsers eingeben, erhalten Sie, bei vorhandenen Suchtreffern, eine gefilterte Ansicht von Symbolen.

Für die Kategorien der Symbole existieren folgende Kategorien.

**Global functions** Listet die Implementierung von globalen Funktionen.

**Global typedefs** Listet die Verwendung von **typedef** Definitionen.

**Global variables** Zeigt die Symbole von globalen Variablen an.

**Preprocessor symbols** Auflistung der mit **#define** erzeugten Präprozessor Direktiven.

**Global macros** Listet Makros von Präprozessor Direktiven auf.

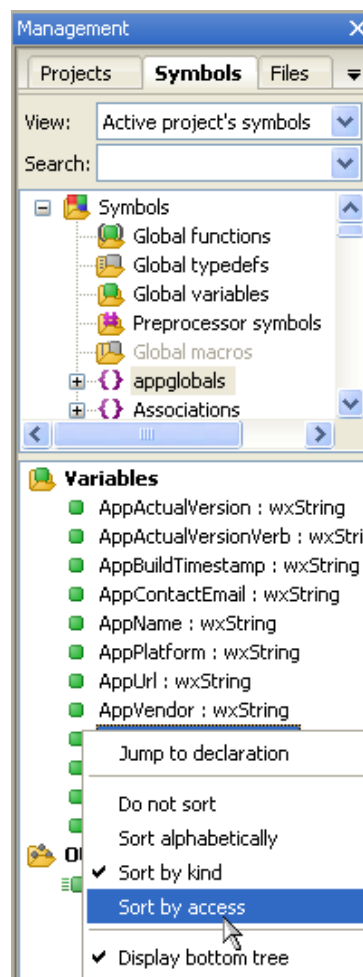


Abbildung 1.7: Symbolansicht

Strukturen und Klassen werden unterhalb im 'bottom tree' angezeigt und die Sortierung kann über das Kontextmenü geändert werden. Wenn eine Kategorie mit der Maus ausgewählt wird, erscheinen die gefundenen Symbole in dem unteren Teil des Fensters (siehe

Abbildung 1.7 auf Seite 11). Ein Doppelklick auf das Symbol öffnet die Datei, wo das Symbol definiert bzw. die Funktion implementiert ist und springt an die zugehörige Zeile. Die Symbolansicht wird beim Speichern einer Datei aktualisiert. Eine Auto-Refresh des Symbolbrowsers ohne Speichern ist über das Menü 'Settings' → 'Editor' → 'Code Completion' aktivierbar (siehe Abbildung 1.8 auf Seite 12). Bei Projekten mit vielen Symbolen wird die Performance innerhalb CodeBlocks beeinträchtigt.

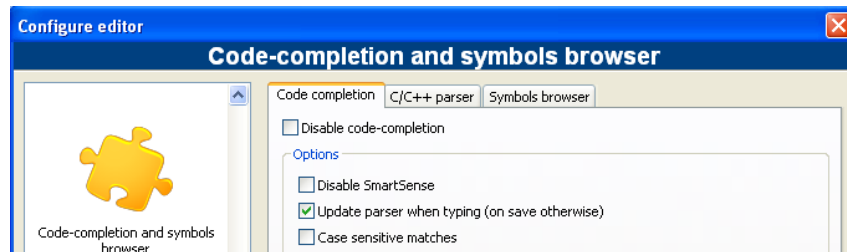


Abbildung 1.8: Aktivieren von Echtzeit-Parsing

**Hinweis:**

Im Editor können Sie über das Kontextmenü 'Insert Class method declaration implementation' bzw. 'All class methods without implementation' sich auch die Liste der Klassen anzeigen lassen.

### 1.10.7 Einbinden von externen Hilfen

Die Entwicklungsumgebung CodeBlocks unterstützt das Einbinden von externen Hilfen über das Menü 'Settings' → 'Environment'. Fügen Sie ein Manual Ihrer Wahl im chm Format in 'Help Files' hinzu und wählen Sie die Einstellung 'this is the default help file' (siehe Abbildung 1.9 auf Seite 13). Dabei steht im Eintrag \$(keyword) als Platzhalter für einen Begriff der im Editor markiert wird. Nun können Sie in CodeBlocks in einer geöffneten Quelldatei eine Funktion mit der Maus durch Doppelklick markieren und anschließend die Hilfe mit F1 aufrufen und erhalten die zugehörige Dokumentation.

Wenn Sie mehrere Hilfedateien einbinden, können Sie im Editor einen Begriff markieren und anschließend über das Kontextmenü 'Locate in' die Hilfedatei auswählen, in der CodeBlocks suchen soll.

In CodeBlocks werden auch die Hilfe mit man pages unterstützt. Hier fügen Sie einen neuen Eintrag 'man' ein und geben den Pfad wie folgt an.

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

CodeBlocks bietet auch einen 'Embedded HTML Viewer', hiermit können einfache HTML-Dateien in CodeBlocks angezeigt und für Suchen genutzt werden. Konfigurieren Sie einfach den Pfad der HTML-Datei, die durchsucht werden soll und aktivieren Sie die Option 'Open this file with embedded help viewer' in dem Menü 'Settings' → 'Environment' → 'Help Files'.

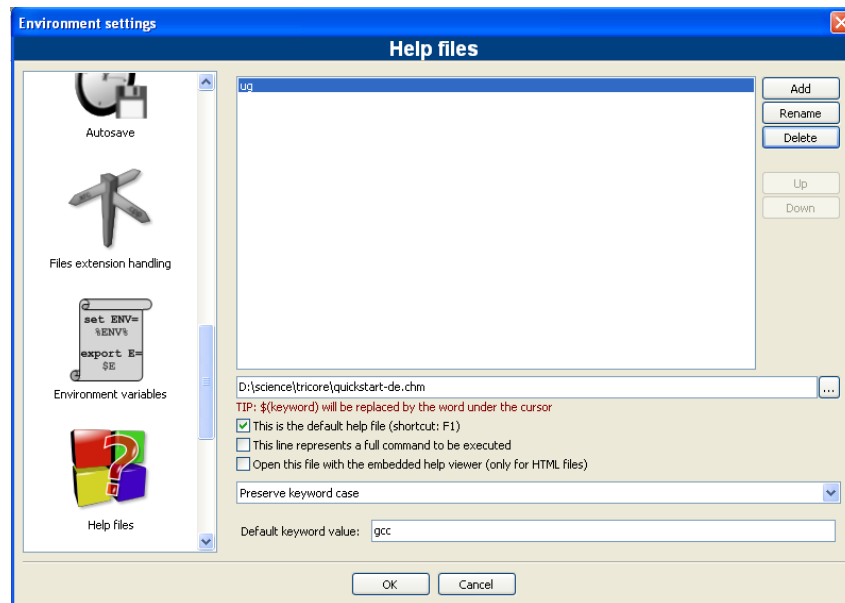


Abbildung 1.9: Einstellungen für Hilfe

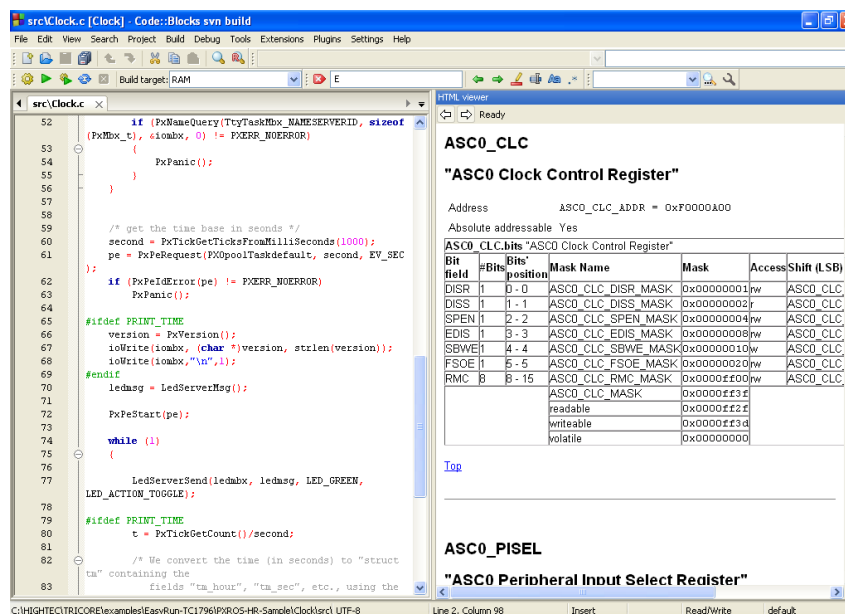


Abbildung 1.10: Embedded HTML Viewer

**Hinweis:**

Wenn Sie eine HTML-Datei im File Explorer mit einem Doppelklick öffnen (siehe [Abschnitt 2.7](#) auf Seite 36) dann wird der Embedded Html Viewer gestartet, solange für HTML Dateien keine andere Zuordnung im file extension handler vorgenommen wurde.

### 1.10.8 Einbinden von externen Werkzeugen

Die Einbindung von externen Tools ist in CodeBlocks unter dem Menüeintrag 'Tools' → 'Configure Tools' → 'Add' vorgesehen. Für die Übergabeparameter der Tools kann auch auf Built-in Variables (see [Abschnitt 3.2](#) auf Seite 56) zugegriffen werden. Des weiteren existieren für das Starten von externen Anwendungen unterschiedliche Arten (Launching options). Je nach Option werden die extern gestarteten Anwendung beim Beenden von CodeBlocks gestoppt. Falls die Anwendungen auch beim Beenden von CodeBlocks geöffnet bleiben sollen, ist die Option 'Launch tool visible detached' einzustellen.

## 1.11 Tips zum Arbeiten mit CodeBlocks

In diesem Kapitel werden Ihnen einige nützliche Einstellungen in CodeBlocks vorgestellt.

### 1.11.1 Änderungen im Editor verfolgen

CodeBlocks bietet die Möglichkeit geänderte Stellen eines Quellcodes im Vergleich zu einer Vorversion mit Hilfe von seitlich angebrachten Revisionsbalken automatisch sichtbar zu machen. Dabei werden Änderungen mit einem gelben Balken und gespeicherte Änderungen mit einem grünen Balken dargestellt (siehe [Abbildung 1.11](#) auf Seite 14). Das Navigieren zwischen den einzelnen Änderungen ist über das Menü 'Search' → 'Goto next changed line' oder 'Search' → 'Goto previous changed line' möglich. Standardmäßig sind hierfür die Tastenkürzel Ctrl-F3 und Ctrl-Shift-F3 konfiguriert.

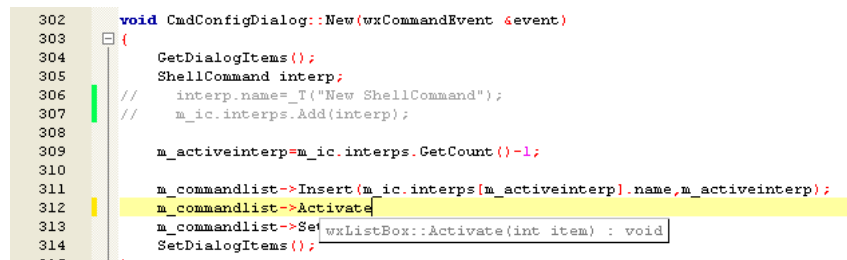


Abbildung 1.11: Verfolgen von Änderungen

Diese Feature kann unter 'Settings' → 'Editor' → 'Margins and caret' mit der Checkbox 'Use Changebar' aktiviert bzw. deaktiviert werden.

#### Hinweis:

Wenn eine geänderte Datei geschlossen wird, geht die Änderungsinformation für Undo/Redo und die Revisionsbalken verloren. Über das Menü 'Edit' → 'Clear changes history' oder das zugehörige Kontextmenü kann auch bei geöffneter Datei die Änderungshistorie gelöscht werden.

### 1.11.2 Datenaustausch mit anderen Anwendungen

Zwischen CodeBlocks und anderen Anwendungen können Daten dynamisch ausgetauscht werden. Diese Interprozess Kommunikation wird unter Windows auf DDE (Dynamic Data

Exchange) und unter anderen Betriebssystemen auf eine TCP Kommunikation zwischen den Anwendungen abgebildet.

Über diese Schnittstelle können an CodeBlocks Kommandos mit der folgenden Syntax weitergegeben werden.

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

Als Kommandos stehen zur Verfügung:

|          |   |
|----------|---|
| Open     | Mit dem folgenden Kommando<br><br><pre>[Open("d:\temp\test.txt")]</pre><br>wird der Parameter, hier die Datei mit absolutem Pfad, innerhalb einer CodeBlocks Instanz geöffnet oder bei Bedarf eine erste Instanz gestartet. |
| OpenLine | Das Kommando öffnet eine Datei mit spezifizierter Zeilennummer in einer CodeBlocks Instanz. Diese Zeilennummer wird mit :line angegeben.<br><br><pre>[OpenLine("d:\temp\test.txt:10")]</pre>                                |
| Raise    | Setzt den Fokus auf die CodeBlocks Instanz. Hier darf kein Parameter angegeben werden.  |

### 1.11.3 Konfiguration von Umgebungsvariablen

Die Konfiguration für ein Betriebssystem wird durch sogenannte Umgebungsvariablen festgelegt. Zum Beispiel enthält die Umgebungsvariablen PATH den Pfad auf einen installierten Compiler. Das Betriebssystem geht diese Umgebungsvariable von vorne nach hinten durch, d.h. die Einträge am Ende werden als letztes durchsucht. Wenn nun unterschiedliche Versionen eines Compilers oder anderer Anwendungen installiert sind, können nun folgende Situationen auftreten:

- Die falsche Version einer Software wird aufgerufen
- Installierte Softwarepakete stören sich gegenseitig

Es könnte zum Beispiel notwendig sein, dass für unterschiedliche Projekte unterschiedliche Versionen eines Compilers oder anderer Werkzeugen vorgeschrieben sind. Eine Möglichkeit ist die Umgebungsvariablen in der Systemsteuerung jeweils für ein Projekt zu ändern. Diese Vorgehensweise ist jedoch fehleranfällig und nicht flexibel. Für diese Anforderung bietet CodeBlocks eine elegante Lösung. Es lassen sich hier unterschiedliche Konfigurationen von Umgebungsvariablen erstellen, die nur intern in CodeBlocks verwendet werden. Zusätzlich kann zwischen diesen Konfiguration umgeschaltet werden. Die [Abbildung 1.12](#) auf Seite 16 zeigt den Eingabedialog, den Sie über das Menü 'Settings' → 'Environment' und Auswahl von 'Environment Variables' erhalten. Eine Konfiguration wird über die Schaltfläche 'Create' erzeugt. Die Übernahme der hinzugefügten Umgebungsvariablen erfolgt durch Bestätigen des OK Knopfes. Das Aktivieren einer Konfiguration erfolgt über den Knopf Set Now.



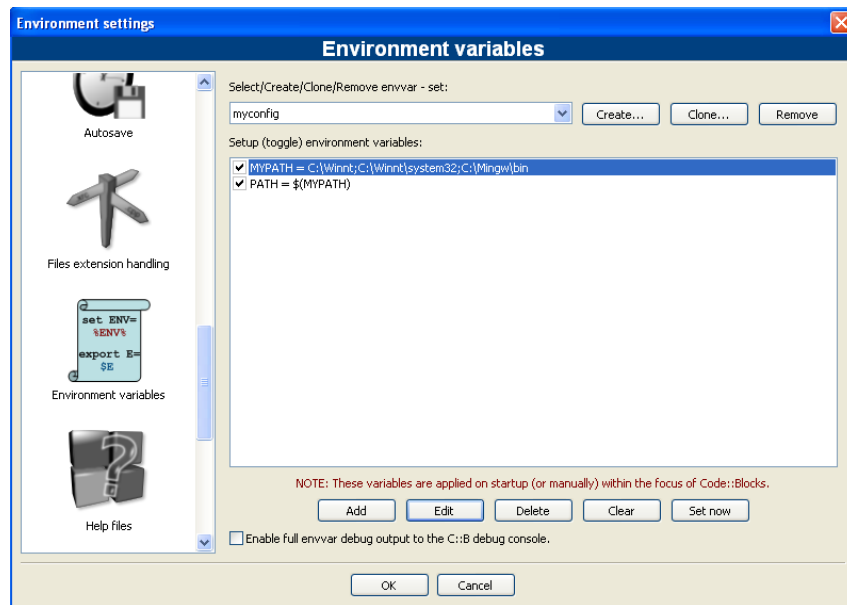


Abbildung 1.12: Umgebungsvariablen

Der Zugriff und der Gültigkeitsbereich auf die hier erstellten Umgebungsvariablen ist auf CodeBlocks begrenzt. Sie können diese Umgebungsvariablen wie auch andere CodeBlocks Variablen über `$(NAME)` expandieren.

#### Hinweis:

Eine Konfiguration von Umgebungsvariable lässt sich pro Projekt im Kontextmenü 'Properties' im Reiter 'EnvVars options' selektieren.

### Beispiel

Sie können die verwendete Umgebung in einem postbuild Step (siehe [Abschnitt 1.6](#) auf Seite 4) in einer Datei `<project>.env` schreiben und zu Ihrem Projekt archivieren.

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

oder unter Linux

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

### 1.11.4 Umschalten zwischen Perspektiven

Abhängig von der Aufgabenstellung ist es sinnvoll unterschiedliche Konfigurationen oder Ansichten in CodeBlocks zu haben und diese zu speichern. Standardmäßig werden die Einstellungen wie z.B. Anzeige von Symbolleisten, Layout etc. in der Konfigurationsdatei `default.conf` gespeichert. Durch die Verwendung der Kommandozeilenoption `--personality=ask` beim Start von CodeBlocks kann zwischen verschiedenen Einstellungen umgeschaltet werden. Neben dieser globalen Einstellung besteht jedoch häufig auch der Wunsch während einer Session zwischen Ansichten für Fenster und Symbolleisten zu wechseln. Zwei typische Szenarien sind z.B. das Editieren von Dateien oder das Debuggen eines Projektes.

Damit der Anwender nicht ständig mit dem Öffnen und Schließen von Fenstern, Symbolleisten etc. beschäftigt ist, bietet CodeBlocks einen Mechanismus um unterschiedliche Perspektiven zu speichern bzw. diese umzuschalten. Über das Menü 'View' → 'Perspectives' → 'Save current' und Eingabe eines Namens <name> wird eine Perspektive gespeichert. Über 'Settings' → 'Editor' → 'Keyboard shortcuts' → 'View' → 'Perspectives' → '<name>' kann ein Tastenkürzel hierfür angegeben werden. Durch diese Vorgehensweise können Sie nun einfach zwischen den Ansichten über Ihre Tastenkürzel wechseln.

**Hinweis:**

Ein weiteres Anwendungsbeispiel ist das Editieren einer Datei im Full Screen Modus ohne Symbolleisten. Hierfür können Sie sich auch eine Perspektive z.B. 'Full' anlegen und ein Tastenkürzel vergeben.

### 1.11.5 Umschalten zwischen Projekten

Wenn mehrere Projekte oder Dateien gleichzeitig geöffnet sind, so will der Benutzer häufig zwischen den Projekten und Dateien schnell wechseln können. CodeBlocks stellt hierfür eine Reihe an Shortcuts zur Verfügung.

**Alt-F5** Aktiviert vorheriges Projekt aus der Projektansicht.

**Alt-F6** Aktiviert nachfolgendes Projekt aus der Projektansicht.

**F11** Wechselt im Editor zwischen einer Quelldatei <name>.cpp und der zugehörigen Header Datei <name>.h

### 1.11.6 Erweitere Einstellung für Compiler

Beim Buildprozess eines Projektes werden die Ausgaben des Compilers in Fenster Messages im Reiter Build Log ausgegeben. Wenn Sie an detaillierten Information interessiert sind, kann die Ausgabe erweitert werden. Dazu wählen Sie unter 'Settings' → 'Compiler and Debugger' im Reiter 'Other Settings'.

Achten Sie darauf, dass beim Eintrag Selected Compiler der gewünschte Compiler eingestellt ist. Die Einstellung 'Full command line' im Feld Compiler Logging gibt die vollständige Information im Build Log aus. Zusätzlich kann diese Ausgabe in eine HTML-Datei geloggt werden. Hierzu ist die Einstellung 'Save build log to HTML file when finished' erforderlich. Des weiteren bietet CodeBlocks eine Fortschrittsanzeige des Buildprozesses im Fenster Build Log. Diese aktivieren Sie mit dem Einstellung 'Display build progress bar'.

### 1.11.7 Zoom im Editor

CodeBlocks bietet einen sehr leistungsfähigen Editor. Eine Besonderheit ist, dass Sie innerhalb einer geöffneten Datei die Darstellung vergrößern und verkleinern können. Wenn Sie eine Maus mit einem Scrollrad haben, halten Sie einfach die Ctrl-Taste gedrückt und scrollen im Editor über das Rad nach vorne oder hinten.

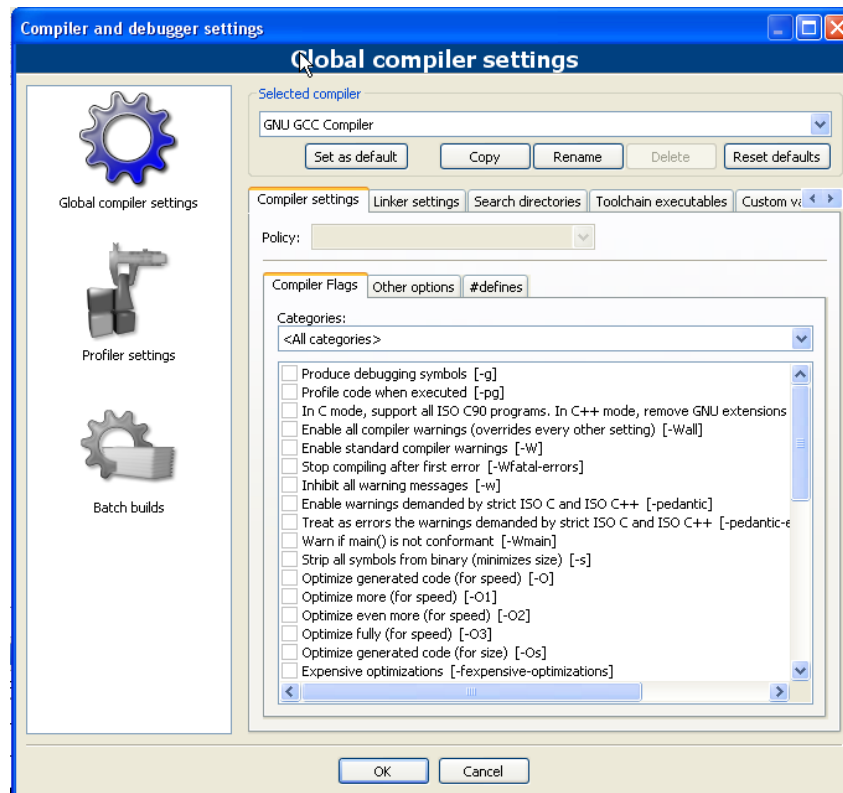


Abbildung 1.13: Einstellung von Detailinformationen

**Hinweis:**

Mit dem Tastenkürzel `Ctrl-Numepad-/` oder mit 'Edit' → 'Special commands' → 'Zoom' → 'Reset' können Sie auf die ursprüngliche Schriftgröße umschalten.

**1.11.8 Wrap Mode**

Für das Editieren von Textdateien z.B. `*.txt` innerhalb CodeBlocks ist es nützlich den Eingabetext automatisch auf die Textbreite des Editorfensters in mehrere Zeilen umbrechen zu lassen. Diese Funktionalität wird 'Word wrap' genannt und diese lässt sich im Menü 'Settings' → 'Editor' → 'Other Options' und mit der Checkbox 'Word wrap' aktivieren. Durch die Home/Pos 1 Taste springt der Cursor an den Anfang und mit End/Ende an das Ende der umgebrochenen Zeilen. Durch die Einstellung 'Settings' → 'Editor' → 'Other Options' und dem Setzen von 'Home key always move to caret to first column' wird erreicht, dass die Home/Pos 1 bzw. End/Ende Taste der Cursor an den Zeilenanfang bzw. an das Ende der aktuellen Zeile springt. Falls jedoch ein Sprung des Cursors an den Zeilenanfang des Abschnitts gewünscht ist, muss stattdessen die Tastenkombination 'Atl-Home/Pos 1' gedrückt werden. Entsprechendes gilt für 'Alt-End/Ende'.

**1.11.9 Select Modes im Editor**

CodeBlocks unterstützt im Editor verschiedene Modes für Selektion und Einfügen von Text.

1. Mit der linken Maustaste kann ein Text innerhalb des Editors selektiert werden und dann die Maustaste gelöst werden. Nun kann der Benutzer mit dem Mausekranz innerhalb des aktiven Editors scrollen und durch Betätigen der mittleren Maustaste, wird der ehemals selektierte Text an die aktuelle Cursorposition eingefügt. Dieses Verhalten funktioniert pro Datei und die Selektion kann als Zwischenablage pro Datei verstanden werden.
2. Durch Halten der 'ALT' Taste wird der Block-Select Modus aktiviert und ein Rechteck kann mit der linken Maustaste aufgezogen werden und dann kopiert bzw. eingefügt werden. Dies ist zum Beispiel nützlich, wenn nur einige Spalten eines Array markiert und kopiert werden sollen.
3. Im Menü 'Settings' → 'Editor' → 'Margins und Caret' können sogenannte 'Virtual Spaces' aktiviert werden. Diese Option bewirkt, dass im Block Select Modus eine Auswahl auch innerhalb einer leeren Zeilen beginnen oder enden kann.
4. Im Menü 'Settings' → 'Editor' → 'Margins und Caret' kann 'Multiple Selection' aktiviert werden. Bei gedrückter Ctrl-Taste können dann mit der linken Maustaste mehrere Textpassagen im Editor selektiert werden. Die Selektionen können mit Ctrl-C bzw. Ctrl-X in der Zwischenablage hintereinander angehängt und mit Ctrl-V an die gewünschte Stelle eingefügt werden. Zudem kann im 'multiple Selection' Modus die Option 'Enable typing (and deleting)' aktiviert werden. Dies kann z.B. nützlich sein, wenn Sie an mehreren Stellen eine Präprozessor Direktive wie `#ifdef` einfügen wollen oder z.B. ein Text an mehreren Stellen überschreiben oder ersetzen wollen.

**Hinweis:**

Beachten Sie, dass die meisten Linux Fenstermanager diese Tastenkombination für das Verschieben von Fenster vordefinieren. Deshalb muss dieses Verhalten des Fenstermanager geändert werden, damit der Block select mode unter Linux funktioniert.

### 1.11.10 Code folding

CodeBlocks unterstützt ein sogenanntes Folding für Quellen. Hiermit lassen sich zum Beispiel Funktionen zusammenklappen. Ein Folding Punkt erkennen Sie im Editor als Minussymbol im linken Seitenrand. Hier wird auch der Beginn und das Ende eines Folding Punktes durch eine vertikale Linie gekennzeichnet. Wenn Sie mit der linken Maustaste auf das Minussymbol klicken wird der entsprechende Abschnitt eingeklappt bzw. ausgeklappt. Sie können über das Menü 'Edit' → 'Folding' einstellen wie eingeklappt werden soll. Im Editor wird ein eingeklappte Codestelle durch eine durchgehende horizontale Linie dargestellt.

**Hinweis:**

Der Stil für das Folding und eine Grenze für die Folding-Tiefe kann im Menü 'Settings' → 'Editor' → 'Folding' geändert werden.

Neben dem Folding für Funktionen kann die Funktionalität auch für Präprozessor Direktiven eingestellt werden. Aktivieren Sie hierfür die Option 'Fold preprocessor commands' im Menü 'Settings' → 'Editor' unter dem Eintrag Folding.

Eine weitere Möglichkeit ist benutzerdefinierte Folding Punkte zu definieren, indem ein Kommentarzeichen durch eine geöffnete Klammer den Anfang und ein Kommentar mit schließender Klammer das Ende markiert.

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

**Hinweis:**

Bei eingeschaltetem Folding passiert es häufig, dass bei engem Seitenrand der Anwender anstatt die gewünschte Zeile zu markieren, den Folding point selektiert. In diesem Modus ist es deshalb ratsam die Zeilennummern im Editor einzublenden und dann mit der linken Maustaste in den linken Seitenrand neben der Anzeige der Zeilennummern eine oder mehrere Zeile zu markieren.

### 1.11.11 Auto complete

CodeBlocks parst beim Öffnen eines Projektes die 'Search directories' die für einen Compiler oder Projekt eingestellt wurden und die im Projekt befindlichen Quellen und Header. Des weiteren werden auch die Keywords der zugehörigen Lexerdateien geparkt. Die aus dem Parsen gewonne Information über Symbole kann für die sogenannte Auto completion genutzt werden, wenn diese in den Einstellungen des Editors für CodeBlocks aktiviert ist. Die Auto completion können Sie im Editor über das Tastenkürzel Ctrl-Space ausführen. Im Menü 'Settings' → 'Editor' → 'Syntax highlighting' können eigene keywords zum Lexer hinzugefügt werden.

### 1.11.12 Find broken files

Wenn eine Datei auf der Festplatte gelöscht wurde, jedoch in der Projektbeschreibung `<project>.cbp` noch enthalten ist, dann wird diese Datei als 'broken file' mit einem unterbrochenem Symbol in der Project View von CodeBlocks angezeigt. Das Entfernen einer Datei sollte in der Projekt View mit dem Kontextmenü 'Remove file from project' vorgenommen werden.

Bei größeren Projekten mit vielen Unterordnern kann die Suchen nach 'broken files' sehr schwierig werden. CodeBlocks bietet jedoch mit dem Plugin ThreadSearch (siehe [Abschnitt 2.6](#) auf Seite 32) eine einfache Lösung. Wenn Sie in ThreadSearch einen Suchbegriff eingeben und als Option 'Project files' oder 'Workspace files' wählen, wird ThreadSearch alle Dateien eines Projektes durchsuchen; falls ein 'broken file' im Projekt oder Workspace vorkommt, wird als Fehler diese Datei gemeldet.

### 1.11.13 Einbinden von Bibliotheken

In den Buildoption eines Projektes können Sie unter 'Linker Settings' im Eintrag 'Link libraries' über die Schaltfläche 'Add' verwendete Bibliotheken hinzufügen. Dabei können

Sie entweder den absoluten Pfad zur Bibliothek durchsuchen oder nur den Namen ohne den Prefix `lib` und die Dateierweiterung angeben.

### Beispiel

Für eine Bibliothek `<path>\libs\lib<name>.a` geben Sie einfach `<name>` an. Der Linker mit den jeweiligen Suchpfaden für die Bibliotheken bindet diese dann korrekt ein.

#### Hinweis:

Eine weitere Möglichkeit wie Sie Bibliotheken einbinden können, beschreibt [Abschnitt 2.10](#) auf Seite 41.

## 1.11.14 Linkreihenfolge von Objekten

Beim Compilierung werden aus Quellen `name.c/cpp` werden Objekte `name.o` erzeugt. Der Linker bindet die einzelnen Objekten zu einer Anwendung `name.exe` oder für den Embedded Bereiche `name.elf`. In einigen Fällen ist es wünschenswert die Reihenfolge für das Binden von Objekten vorzugeben. In CodeBlocks kann dies durch die Vergabe von sogenannten Prioritäten erzielt werden. Stellen Sie für eine Datei über das Kontextmenü 'Properties' im Reiter Build die Priorität ein. Dabei führt eine geringe Priorität des Objekts dazu, dass es zu erst gebunden wird.

## 1.11.15 Autosave

CodeBlocks bietet die Möglichkeit Projekte und Quelldateien automatisch zu speichern bzw. eine Sicherungskopie anzulegen. Diese Funktionalität wird im Menü 'Settings' → 'Environment' → 'Autosave' eingestellt. Dabei sollte als 'Save to .save file' als Methode für das Erstellen einer Sicherungskopie eingestellt werden.

## 1.11.16 Einstellen für Dateizuordnungen

In CodeBlocks können Sie zwischen verschiedenen Arten der Behandlung von Dateierweiterungen wählen. Die Einstellungen erhalten Sie über 'Settings' → 'Files extension handling'. Sie können entweder die von Windows zugeordneten Anwendungen (open it with the associated application) für entsprechende Dateierweiterungen verwenden oder für jede Dateierweiterungen die Einstellungen so ändern, dass entweder ein benutzerdefiniertes Programm (launch an external program) gestartet wird oder die Datei in Editor von CodeBlocks geöffnet wird (open it inside Code::Blocks editor).

#### Hinweis:

Wenn ein benutzerdefiniertes Programm für eine Dateierweiterung gewählt wird, sollte die Einstellung 'Disable Code::Blocks while the external program is running' deaktiviert werden, da sonst beim Öffnen dieser Dateien CodeBlocks beendet wird.

## 1.12 CodeBlocks in der Kommandozeile

Die IDE CodeBlocks kann auch ohne grafische Oberfläche in der Kommandozeile ausgeführt werden. Dabei stehen unterschiedliche Schalter zur Verfügung um den Buildprozess eines Projektes zu steuern. Da CodeBlocks somit skriptfähig ist, kann die Erzeugung von Executables in eigene Arbeitsabläufe integriert werden.

```
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** Alle Plugins werden beim Start deaktiviert.

**> <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

Auch wenn man eine IDE wie CodeBlocks überwiegend mit der Maus bedient, erweisen sich dennoch Tastenkombinationen immer wieder als hilfreich, um die Arbeit zu vereinfachen und zu beschleunigen. In nachstehender Tabelle sind einige verfügbare Tastenkombinationen zusammengefasst.

### 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 dient dem Einrücken, Formatieren und 'Verschönern' für C, C++, C# Quellen. Es kann verwendet werden um unterschiedliche Coding Rules für CodeBlocks einzustellen.

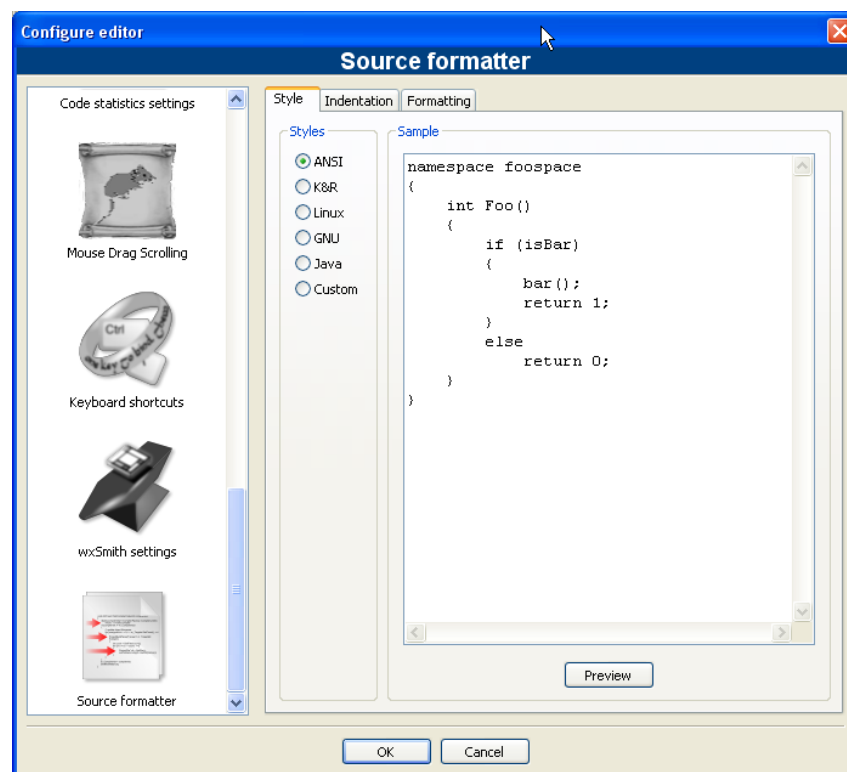


Abbildung 2.1: Formatierung für Quellcode

Wenn Quellen eingerückt werden, tendieren Programmierer dazu sowohl Leerzeichen als auch Tabulatoren einzusetzen, um die gewünschte Einrückung zu erzielen. Darüberhinaus gibt es auch Editoren die standardmäßig Tabulatoren durch eine feste Anzahl von Leerzeichen ersetzen. Andere Editoren versuchen den Code durch Einfügen von White space lesbarer zu machen selbst wenn der Code Tabulatoren enthält.

Da die Anzeige der Leerzeichen für jeden Tabulator durch die Einstellungen im Editor bestimmt ist, wirft dies immer ein Problem auf, wenn Programmierer unterschiedliche Editoren verwenden. Selbst bei größter Sorgfalt für die Formatierung der Quelle kann das Editieren durch andere Programmieren mit unterschiedlichen Editoren oder Einstellungen schnell Problemen verursachen.

Um diesen Problem Rechnung zu tragen, wurde Artistic Style entwickelt - ein Filter, der automatisch Ihre C / C++ / C# einrückt und formatiert.

**Hinweis:**

Durch Kopieren von Code z.B aus dem Internet oder aus einem Manual wird in CodeBlocks der Code automatisch an die ausgewählten Coding-Rules angepasst, indem Sie den Text markieren und das Plugin über das Menü 'Plugins' → 'Source code formatter' ausführen.

## 2.2 CodeSnippets

Das Plugin CodeSnippets ermöglicht es Textbausteine und Verknüpfungen auf Dateien in einer Baumansicht nach Kategorien zu strukturieren. Die Bausteine dienen dazu, häufig verwendete Dateien oder Konstrukte in Textbausteine abzulegen und zentral zu verwalten. Stellen Sie sich vor eine Reihe von häufig verwendeten Quelldateien sind im Dateisystem in unterschiedlichen Ordnern abgelegt. Im Fenster CodeSnippets können Sie nun Kategorien und darunter Verknüpfungen auf die gewünschten Dateien erstellen. Damit können Sie den Zugriff auf die Dateien unabhängig von der Ablage im Dateisystem verwalten und ohne das Dateisystem zu durchsuchen schnell zwischen diesen Dateien navigieren.

**Hinweis:**

Sie können auch CodeBlocks Variablen oder Umgebungsvariablen verwenden, um Links im CodeSnippets Browser zu parametrisieren z.B. `$(VARNAME)/name.pdf`.

Die Liste der Textbausteine und Verknüpfungen können im CodeSnippets Fenster mit der rechten Maustaste über das Kontextmenü 'Save Index' gespeichert werden. Die dabei erzeugte Datei `codesnippets.xml` befindet sich anschließend in Ihren **Dokumente und Einstellungen\Anwendungsdaten** im Ordner `codeblocks`. Unter Linux wird diese Information im HOME-Verzeichnis im Ordner `.codeblocks` abgelegt. Die Konfigurationsdateien von CodeBlocks werden beim nächsten Start geladen. Falls Sie den Inhalt von CodeSnippets an einen anderen Ort speichern möchten, selektieren Sie den Eintrag 'Save Index As'. Zum Laden dieser Datei wählen Sie beim nächsten Start von CodeBlocks 'Load Index File' oder stellen das Verzeichnis in dem Kontextmenü 'Settings' unter 'Snippet Folder' ein. Diese Einstellungen werden in der zugehörigen Datei `codesnippets.ini` in den Anwendungsdaten hinterlegt.

Das Einfügen einer Kategorie geschieht über das Menü 'Add SubCategory'. In einer Kategorie können Snippets (Textbausteine) oder File Links (Verknüpfungen) liegen. Ein Textbaustein wird mit dem Kontextmenü über 'Add Snippet' angelegt. Indem Sie einen Text im CodeBlocks Editor markieren und anschließend bei gedrückter linker Maustaste per Drag and Drop auf den Textbaustein ziehen, wird der Inhalt in den Textbaustein eingefügt. Wenn Sie einen selektierten Text auf eine Kategorie ziehen wird in diesem Ordner automatisch ein Textbaustein mit dem Namen 'New snippet' erzeugt und es öffnet sich der Properties Dialog. Durch einen Doppelklick auf den neu eingefügten Eintrag oder durch Auswahl von 'Edit Text' öffnet sich ein eigenständiger Editor zum Bearbeiten des Inhaltes.

Die Ausgabe eines Textbausteines in CodeBlocks erfolgt über das Kontextmenü 'Apply' oder durch Drag und Drop in den Editor. Die Inhalte eines Snippets können auch in

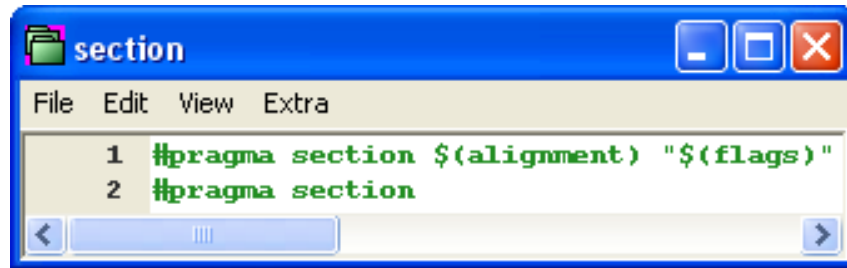


Abbildung 2.2: Bearbeiten eines Textbausteins

andere Anwendungen gezogen werden. Im CodeSnippets Browser können Sie auch per Drag and Drop einen Eintrag in eine andere Kategorie kopieren.

Textbausteine sind darüberhinaus auch über Variablen `<name>`, die über `$(name)` zugegriffen werden, parametrisierbar (siehe [Abbildung 2.2](#) auf Seite 27). Die Abfrage für die Werte der Variablen erfolgt über ein Eingabefeld, wenn der Textbaustein mit dem Kontextmenü 'Apply' aufgerufen wird.

Neben den Textbausteinen können auch Verknüpfungen auf Dateien angelegt werden. Wenn Sie zuvor einen Textbaustein angelegt haben und anschließend das Kontextmenü 'Properties' auswählen, selektieren Sie mit der Schaltfläche 'Link target' das Ziel der Verknüpfung. Eine Verknüpfung kann auch über das Kontextmenü 'Convert to FileLink' erzeugt werden. Dieser Schritt wandelt den Textbaustein automatisch in eine Verknüpfung auf eine Datei um. In CodeSnippets werden Textbausteine mit einem T-Symbol und Verknüpfungen auf eine Datei mit einem F-Symbol und Urls mit einem U-Symbol gekennzeichnet. Falls Sie die in Codesnippets markierte Datei (Verknüpfung) öffnen möchten selektieren Sie im Kontextmenü 'Open File' oder halten Sie die 'Alt' Taste gedrückt und machen ein Doppelklick auf die Datei.

**Hinweis:**

In Textbausteine können auch Urls angegeben werden z.B. <http://www.codeblocks.org>. Die Url kann wahlweise über das Kontextmenü 'Open Url' oder per Drag and Drop in Ihrem gewohnten Webbrowser geöffnet werden.

Falls Sie diese Einstellung vorgenommen haben, dann wird wenn Sie z.B. einen Verknüpfung auf eine pdf-Datei aus der Codesnippets Ansicht öffnen automatisch ein pdf-Viewer gestartet. Dieses Vorgehen ermöglicht dem Benutzer Dateien, die über das Netzwerk verteilt liegen, wie z.B. CAD Daten, Schaltpläne, Dokumentation etc. als Verknüpfung einfach über die gewohnten Anwendungen zuzugreifen. Der Inhalt der Codesnippets wird in der Datei `codesnippets.xml` und die Konfiguration in der Datei `codesnippets.ini` in Ihren Anwendungsdaten gespeichert. In dieser ini Datei wird z.B. der Ablageort der Datei `codesnippets.xml` hinterlegt.

CodeBlocks unterstützt die Verwendung von unterschiedlichen Profilen. Diese werden als personalities bezeichnet. Wenn Sie CodeBlocks mit der Kommandozeilen Option `--personality=<profil>` starten, wird entweder ein neues angelegt oder ein existierendes Profil verwendet. Die

Einstellungen werden dann statt in `default.conf` in der Datei `<personality>.conf` in den Anwendungsdaten gespeichert. Das Plugin Codesnippets speichert seine Einstellungen dann in der Datei `<personality>.codesnippets.ini`. Wenn nun Sie in den Settings von Codesnippets über 'Load Index File' einen neuen Inhalt `<name.xml>` laden, wird dies in der zugehörigen ini Datei hinterlegt. Der Vorteil von dieser Vorgehensweise ist, dass Sie zu unterschiedlichen Profilen auch unterschiedliche Konfigurationen für Textbausteine und Verknüpfungen verwalten können.

Für das Navigieren zwischen den Kategorien und Snippets bietet das Plugin eine zusätzliche Suchfunktion. Hierbei lässt sich auch der Gültigkeitsbereich (Scope) für die Suche auf Snippets, Categories oder Snippets and Categories einstellen. Durch Eingabe des gewünschten Suchbegriffes wird automatisch der zugehörige Eintrag in der Ansicht ausgewählt. Die [Abbildung 2.3](#) auf Seite 28 zeigt eine typische Ansicht im CodeSnippets Fenster.

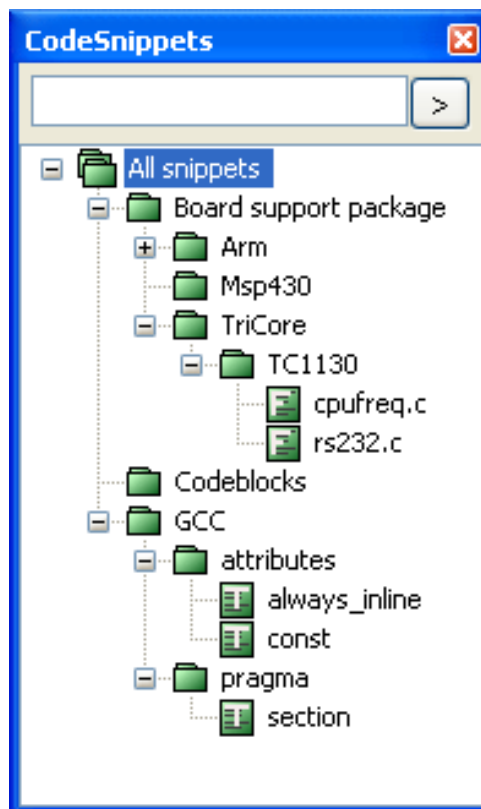


Abbildung 2.3: Ansicht von CodeSnippets

**Hinweis:**

Bei Verwendung von umfangreichen Textbausteine sollte deren Inhalt über 'Convert to File Link' in Dateien ausgelagert werden, um die Speicherauslastung im System zu reduzieren. Beim Löschen von Textbausteine und Verknüpfungen werden diese in den Ordner `.trash` verschoben bzw. bei gedrückter Shift-Taste direkt gelöscht.

## 2.3 Incremental Search

Für eine effiziente Suche in geöffneten Dateien bietet CodeBlocks die sogenannte Incremental Search Methode. Über das Menü 'Search' → 'Incremental Search' oder das Tastenkürzel Ctrl-I wird diese Suchmethode für eine geöffnete Datei eingeleitet. Dabei wird dann automatisch der Focus auf die Suchmaske der zugehörigen Werkzeugleiste gesetzt. Wenn Sie mit der Eingabe eines Begriffes beginnen, wird abhängig von dem Vorkommen der Hintergrund der Suchmaske hinterlegt. Sobald ein Treffer im aktiven Editor gefunden wird, erscheint diese Stelle farblich markiert. Standardmäßig wird der aktuelle Treffer grün hervorgehoben. Die Einstellungen hierfür können im Menü 'Settings' → 'Editor' → 'Incremental Search' geändert werden (siehe ?? auf Seite ??). Durch Betätigen der Return Taste wird zum nächsten Vorkommen des Suchbegriffes gesprungen. Mit Shift-Return kann zum vorherigen Vorkommen navigiert werden. Diese Funktionalität wird jedoch bei Suchen mit regulären Ausdrücken in Scintilla nicht unterstützt.

```
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;
}
... ..
```

Wird der Suchbegriff in der aktiven Datei jedoch nicht gefunden, wird dies durch rotes Hinterlegen der Suchmaske signalisiert.

**ESC** Verlässt den Incremental Search Modus.

**ALT-DELETE** Löscht den Inhalt für die Eingabe von Incremental Search.

Die Icons in der Werkzeugleiste von Incremental Search sind wie folgt zu verstehen:



Löschen des Textes innerhalb der Suchmaske der Incremental Search Werkzeugleiste.



, Navigation zwischen den Vorkommen eines Suchbegriffes.



Dieser Knopf bewirkt, dass nicht nur der aktuelle gefundene Suchbegriff im Editor sondern auch weitere Vorkommnisse farblich hervorgehoben werden.



Mit der Aktivierung dieser Option wird nur innerhalb eines selektierten Textes im

Editor gesucht.



Bewirkt, dass die Suche von Groß-/Kleinschreibung abhängt.



Der Suchbegriff wird als regulärer Ausdruck interpretiert.

**Hinweis:**

Die standardmäßigen Einstellungen dieser Werkzeugleiste sind in 'Settings' → 'Editor' → 'Incremental Search' konfigurierbar.

## 2.4 ToDo List

Für komplexe Software-Projekte, an denen unterschiedliche Benutzer arbeiten, hat man häufig die Anforderung, dass zu erledigende Arbeiten von unterschiedlichen Usern umzusetzen sind. Für dieses Problem bietet CodeBlocks eine ToDo List. Diese Liste, zu öffnen unter 'View' → 'To-Do list', enthält die zu erledigenden Aufgaben mit Prioritäten, Typ und zuständige User. Dabei kann die Ansicht nach zu erledigenden Aufgaben nach Benutzer und/oder Quelldatei gefiltert werden. Eine Sortierung nach Spalten erhält der Benutzer durch Anklicken der jeweiligen Spaltenüberschrift.

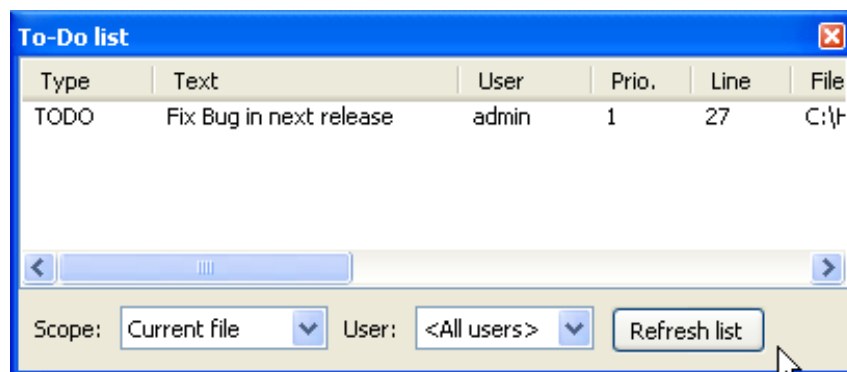


Abbildung 2.4: Anzeige der ToDo List

**Hinweis:**

Die To-Do Liste kann auch direkt in der Message Console angezeigt werden, indem Sie die Einstellung 'Include the To-Do list in the message pane' im Menü 'Settings' → 'Environment' auswählen.

Ein Todo lässt sich bei geöffneten Quellen in CodeBlocks über die rechte Maustaste 'Add To-Do item' hinzufügen. Im Quellcode wird ein entsprechender Kommentar an der ausgewählten Quellzeile eingefügt.

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

Beim Hinzufügen eines To-Do erhalten Sie einen Eingabedialog mit folgenden Einstellungen (siehe [Abbildung 2.5](#) auf Seite 31).

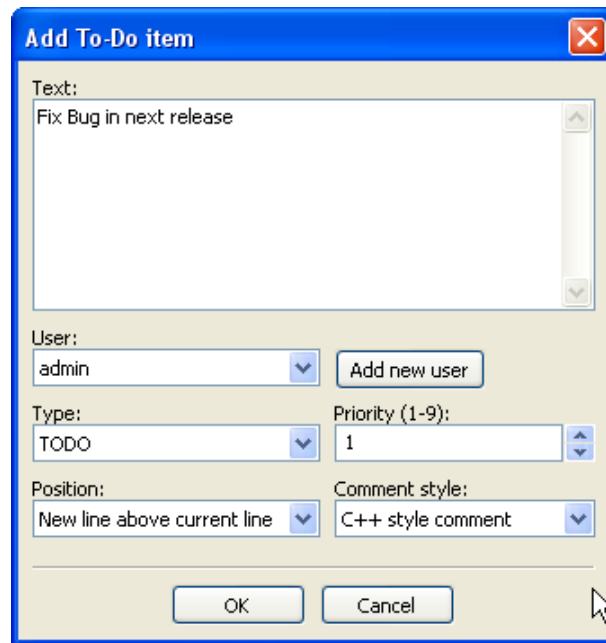


Abbildung 2.5: Dialog für Eingabe von ToDo

**User** Username <user> im Betriebssystem. Hierbei können auch Aufgaben für andere Benutzer angelegt werden. Dabei muss der zugehörige Benutzername über Add new user hinzugefügt werden. Die Zuordnung eines Todo geschieht dann über die Auswahl der unter User aufgelisteten Einträge.

**Hinweis:**

Beachten Sie, dass die User nichts mit den in CodeBlocks verwendeten Personalities zu tun haben.

**Type** Standardmäßig ist der Typ auf Todo eingestellt.

**Priority** Die Wichtigkeit von Aufgaben können in CodeBlocks durch Prioritäten (Wertebereich: 1 - 9) gewichtet werden.

**Position** Einstellung ob der Kommentar vor, nach oder exakt an der Stelle des aktuell befindlichen Cursor eingefügt werden soll.

**Comment Style** Auswahl der Formatierung für Kommentare (zum Beispiel doxygen).

## 2.5 Source Code Exporter

Oft ergibt sich die Notwendigkeit, den Quelltext in andere Anwendungen oder in Emails zu übernehmen. Beim schlichten Kopieren des Textes geht jedoch die Formatierung verloren, was den Text sehr unübersichtlich macht. Die Export Funktion in CodeBlocks schafft hier Abhilfe. Über 'File' → 'Export' kann ein gewünschtes Dateiformat für die Exportdatei ausgewählt werden. Danach übernimmt das Programm den Dateinamen und das Zielverzeichnis der geöffneten Quelldatei und schlägt diesen als Name zum speichern vor. Die jeweilige Dateierweiterung wird durch das Exportformat bestimmt. Es stehen folgende Formate zur Verfügung.



**html** Ein textbasiertes Format, das in einem Web-Browser oder Anwendungen zur Textverarbeitung angezeigt werden kann.

**rtf** Das Rich Text Format ist ein textbasiertes Format, das sich in Programmen zur Textverarbeitung wie Word oder OpenOffice öffnen lässt.

**odt** Open Document Text Format ist ein standardisiertes Format, dass von Sun und O'Reilly festgelegt wurde. Dieses Format kann von Word, OpenOffice und anderen Textverarbeitungsprogrammen eingelesen werden.

**pdf** Das Portable Document Format kann mit Anwendungen wie Acrobat Reader geöffnet werden.

## 2.6 Thread Search

Über das Menu 'Search' → 'Thread Search' lässt sich das entsprechende Plugin als Tab in der Messages Console ein- und ausblenden. In CodeBlocks kann mit diesem Plugin eine Vorschau für das Auftreten einer Zeichenkette in einer Datei, Workspace oder Verzeichnis angezeigt werden. Dabei wird die Liste der Suchergebnisse in der rechten Seite der ThreadSearch Console angezeigt. Durch Anklicken eines Eintrages in der Liste wird auf der linken Seite eine Vorschau angezeigt. Durch einen Doppelklick in der Liste wird die ausgewählte Datei im CodeBlocks Editor geöffnet.

### Hinweis:

Die Einstellung von zu durchsuchenden Dateieindungen voreingestellt ist und eventuell angepasst werden muss.

### 2.6.1 Features

ThreadSearch plugin bietet folgende Funktionalität

- Mehrfache Suche in Dateien
- Interner Editor zur Anzeige einer Vorschau der Suchergebnisse
- Öffnen der Datei im Editor
- Kontextmenü 'Find occurrences' um Suche in Dateien nach dem Wort unter dem aktuellen Cursor zu starten.

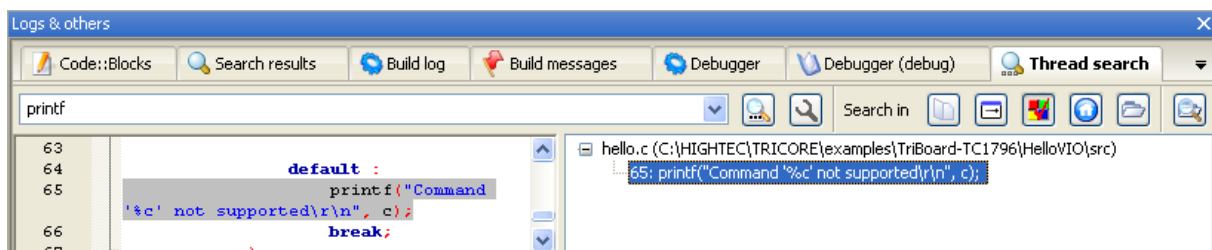


Abbildung 2.6: Thread Search Panel

## 2.6.2 Verwendung

1. Konfigurieren Sie Ihre Einstellungen für die Suche (see [Abbildung 2.7](#) auf Seite 34)

Nach dem das Plugin installiert wurde gibt es vier Arten die Suche zu starten.

- a) Eingabe oder Auswahl eines Wortes in der Combo Box ein und Bestätigen Sie Ihre Eingabe mit Return oder drücken Sie den Search Knopf im Thread Search Panel in der Message Console.
- b) Eingabe oder Auswahl eines Wortes in der Symbolleiste Search combo box und Bestätigen Sie Ihre Eingabe mit Return oder drücken Sie den Search Knopf.
- c) Wählen Sie ein 'Wort' im aktiven Editor und wählen Sie im Kontextmenü 'Find occurrences'.
- d) Selektieren Sie Search/Thread search um den ausgewählten Begriff im aktiven Editor zu finden.

**Hinweis:**

Eintrag 1, 2 und 3 erscheint nur bei entsprechenden Konfiguration von Thread Search.

2. Erneutes Betätigen des Search Knopfes bricht die Suche ab.
3. Durch Anklicken eines Eintrages in der Liste der Suchergebnisse wird auf der linken Seite eine Vorschau angezeigt.
4. Durch Doppelklick eines Eintrages in der Liste der Suchergebnisse wird die zugehörige Datei geöffnet und an die gesuchte Stelle gesprungen.

## 2.6.3 Einstellungen

Der Knopf 'Options' öffnet den Dialog für die Konfiguration des ThreadSearch plugin (see [Abbildung 2.7](#) auf Seite 34):

1. Knopf 'Options' in dem Reiter Thread Search der Message Console.
2. Knopf 'Options' der Symbolleiste Thread Search.
3. Menü 'Settings' → 'Environment' und Eintrag Thread search in der linken Spalte wählen.

**Hinweis:**

Eintrag 1, 2 und 3 erscheint nur bei entsprechenden Konfiguration von Thread Search.

Sie können Filter für die Suche von Dateien konfigurieren.

- Project und Workspace checkboxes schließen sich gegenseitig aus.
- Suchpfad kann bearbeitet werden oder über den Knopf 'Select' konfiguriert werden.
- Maske von Dateieindungen, die durch ';' getrennt sind. Zum Beispiel: \*.cpp;\*.c;\*.h.

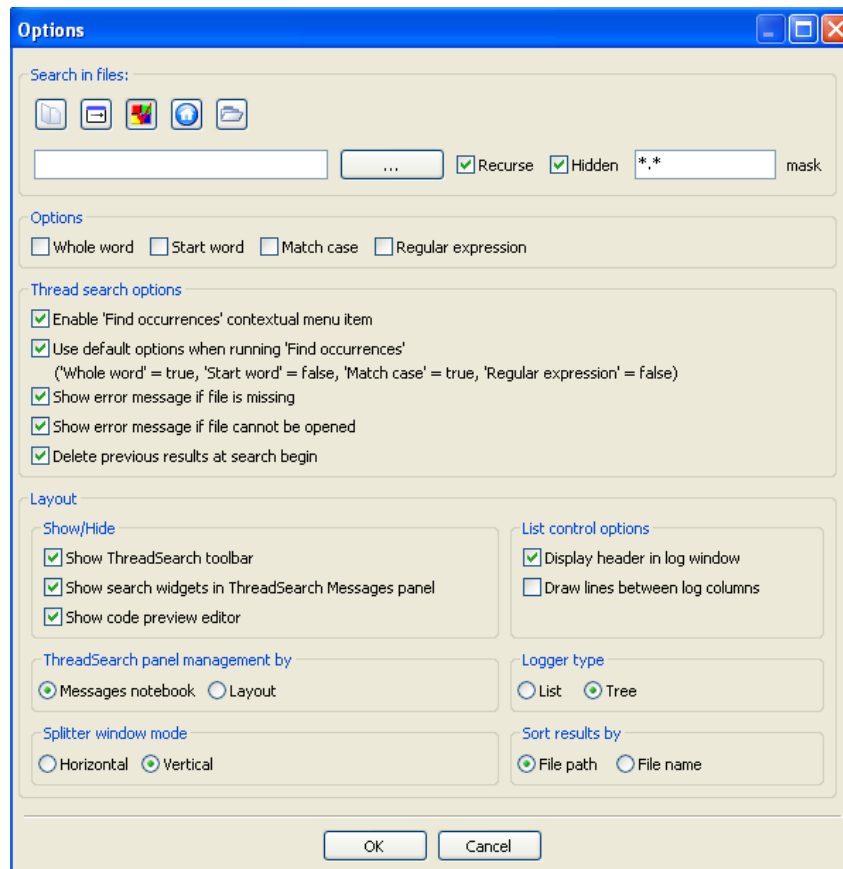


Abbildung 2.7: Konfiguration von Thread Search

### 2.6.4 Optionen

**Whole word** Diese Einstellung gibt in den Suchergebnisse nur die Begriffe zurück, die exakt dem Eintrag für die Suche entsprechen.

**Start word** Sucht alle Begriffe die mit Eintrag der Suche beginnen..

**Match case** Berücksichtigt Groß- und Kleinschreibung bei der Suche.

**Regular expression** Regulärer Ausdruck für eine Suche.

#### Hinweis:

Um nach reguläre Ausdrücken wie `n` suchen zu können, muss in 'Settings' → 'Editor' → 'General Settings' der Eintrag 'Use Advanced RegEx searches' aktiviert sein.

### 2.6.5 Konfiguration von Thread search

**Enable 'Find occurrences contextual menu item'** Fügt den Eintrag 'Find occurrences of 'Focused word' im Kontextmenü im Editor hinzu.

**Use default options when running 'Find occurrences'** Diese Einstellung übernimmt die

voreingestellte Konfiguration für das Kontextmenü 'Find occurrences'. Standardmäßig ist die Einstellung 'Whole word' und 'Match case' aktiv.

**Delete previous results at search begin** Bei der Suche mit ThreadSearch und der Einstellung 'Tree View' werden die Suchergebnisse hierarchisch angezeigt, d.h.

- der erste Knoten ist der Suchbegriff selbst
- darunter werden die Dateien, die den Suchbegriff enthalten, gelistet
- innerhalb der Dateiliste wird die Zeilennummer und der zugehörige Inhalt, wo der Suchbegriff gefunden wurde, angezeigt

Bei einer Suche nach mehreren Begriffen wird die Liste schnell unübersichtlich, deshalb bietet diese Einstellung die Möglichkeit vorangegangene Suchergebnisse beim Start einer Suche zu löschen.

**Hinweis:**

In der Anzeige der Suchergebnisse können auch einzelne oder alle Einträge über das Kontextmenü 'Delete item' bzw. 'Delete all items' gelöscht werden.

## 2.6.6 Layout

**Display header in log window** Der Name der Dateien wird in den Suchergebnissen angezeigt.

**Hinweis:**

Wenn diese Option deaktiviert ist, können die Spaltenbreite nicht mehr verändert werden, belegen jedoch Platz.

**Draw lines between columns** Anzeigen von Linien zwischen den Spalten im List Mode.

**Show ThreadSearch toolbar** Anzeige der Symbolleiste für das Thread Search plugin.

**Show search widgets in ThreadSearch Messages panel** Mit dieser Einstellung werden nur das Fenster für die Suchergebnisse und der Editor für die Vorschau angezeigt. Die Anzeige aller anderen Elementen für das Thread Search Plugin wird unterdrückt.

**Show code preview editor** Code preview kann entweder in den Thread Search Optionen deaktiviert werden oder durch einen Doppelklick auf die Trennlinie zwischen Code Preview und der Ausgabe der Suchergebnissen versteckt werden. In den Optionen kann die Vorschau wieder aktiviert werden.

## 2.6.7 Panel Management

Für das Verwalten des ThreadSearch Fenster stehen zwei Alternativen zur Auswahl. Mit der Einstellung 'Message Notebook' wird das ThreadSearch Fenster in der Message Konsole andockt. Mit der Einstellung 'Layout' können Sie das Fenster aus der Message Konsole lösen und als freies Fenster anordnen.

### 2.6.8 Logger Type

Für die Ansicht der Suchergebnisse existieren zwei Ansichten. Mit der Einstellung 'List' werden alle Einträge untereinander angezeigt. Der andere Mode 'Tree' zeigt die Suchergebnisse in einer Baumansicht an. Dabei werden Suchergebnisse aus einer Datei in einem Knoten zusammengefasst.

### 2.6.9 Splitter Window Mode

Der Benutzer kann eine horizontale oder vertikale Teilung der Fenster für die Vorschau und die Ausgabe von Suchergebnissen angeben.

### 2.6.10 Sort Search Results

Die Ansicht für die Suchergebnisse lässt sich sortieren nach Pfad oder Dateiname.

## 2.7 FileManager und PowerShell Plugin

Der File Explorer [Abbildung 2.8](#) auf Seite [37](#) ist im FileManager Plugin enthalten. In [Abbildung 2.8](#) auf Seite [37](#) ist der Aufbau des File Explorers dargestellt. Der File Explorer erscheint als Tab 'Files' im Management Fenster.

Das oberste Eingabefeld dient zur Angabe des Pfades. Die History der letzten Einträge erhalten Sie durch Mausklick auf die Schaltfläche neben dem Eingabefeld. Dadurch öffnet sich das Listenfeld, in dem der entsprechende Eintrag ausgewählt werden kann. Die Pfeil-nach-oben-Schaltfläche rechts daneben schiebt die Anzeige der Verzeichnisstruktur um eins nach oben.

Im Feld 'Wildcard' können Sie Filter für die Anzeige von Dateien angeben. Mit einer leeren Eingabe oder \* werden alle Dateien angezeigt. Ein Eintrag \*.c;\*.h zeigt z.B. nur C-Quellen und Headerdateien an. Durch Öffnen des Listenfeldes kann wiederum eine History der letzten Einträge zur Auswahl angezeigt werden.

Durch Mausklick mit gedrückter Shift-Taste kann ein Block von Dateien und Verzeichnissen ausgewählt werden, durch Mausklick mit gedrückter Ctrl-Taste können mehrere einzelne Dateien und Verzeichnisse ausgewählt werden.

Für die Auswahl eines oder mehrerer Verzeichnisse im File Explorer stehen Ihnen über das Kontextmenü folgende Operationen zur Verfügung:

**Make Root** Definiert das aktuell ausgewählte Verzeichnis als Hauptverzeichnis.

**Add to Favorites** Setzt ein Lesezeichen für das Verzeichnis und speichert es als Favorit ab. Durch diese Auswahl können Sie schnell zwischen häufig verwendeten Verzeichnissen, z.B. auf unterschiedlichen Netzlaufwerken, springen.

**New File** Erzeugt eine neue Datei in ausgewählten Verzeichnis.

**New Directory** Legt ein neues Unterverzeichnis im ausgewählten Ordner an.

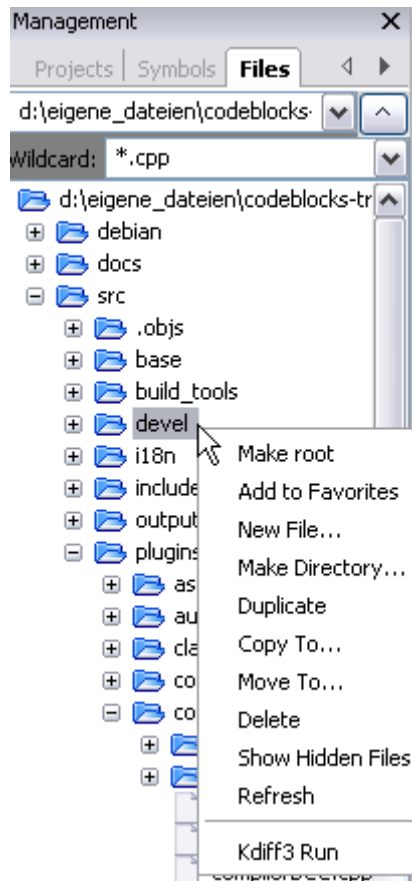


Abbildung 2.8: Ansicht des Dateimanagers

Für die Auswahl von Dateien und Verzeichnissen stehen im Kontextmenü folgende gemeinsamen Befehle zur Verfügung.

**Duplicate** Macht eine Kopie und benennt die Kopie um.

**Copy To** Es öffnet sich ein Dialog in dem Sie das Zielverzeichnis für das Kopieren auswählen können.

**Move To** Verschiebt Auswahl an die gewünschte Stelle.

**Delete** Löscht die ausgewählten Ordner/Dateien.

**Show Hidden Files** Aktiviert bzw. deaktiviert die Anzeige von versteckten Systemdateien. Wenn die Einstellung aktiviert ist, erscheint ein Haken vor dem Eintrag im Kontextmenü.

**Refresh** Aktualisiert die Anzeige im Verzeichnisbaum.

Folgende Einträge sind nur für die Auswahl ein oder mehrerer Dateien gültig.

**Open in CB Editor** Öffnet ausgewählte Datei im CodeBlocks Editor.

**Rename** Benennt die Datei um.

**Add to active project** Fügt die Datei oder Dateien zum aktiven Projekt hinzu.

**Hinweis:**

Die im File-Explorer ausgewählten Dateien oder Verzeichnisse sind über die Variable `mpaths` im Shell Extension Plugin verfügbar.

Über den Menübefehl 'Settings' → 'Environment' → 'PowerShell' können benutzerdefinierte Funktionen erstellt werden. In der Eingabemaske des PowerShell wird mit der Schaltfläche 'New' eine neue Funktion angelegt, die frei benannt werden kann. Im Feld ShellCommand Executable wird das auszuführende Programm angegeben, im unteren Feld können dem auszuführenden Programm zusätzliche Parameter übergeben werden. Durch Auswahl der Funktion im Kontextmenü oder PowerShell-Menü wird die eingetragene Aktion für die markierten Dateien oder Verzeichnisse ausgeführt. Die Ausgabe wird dabei auf ein eigenes Shell-Window umgelenkt.

Als Beispiel wird für den Eintrag mit dem Namen 'SVN' ein zugehöriger Menüeintrag in 'PowerShell' → 'SVN' und im Kontextmenü des File-Explorers hinzugefügt. Hierbei bedeutet `$file` die Datei, welche im FileExplorer markiert ist, und `$mpath` die markierten Dateien oder Verzeichnisse.

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

Dieser sowie jeder weitere Befehl erzeugt ein Untermenü, in diesem Fall 'Extensions' → 'SVN' → 'Add'. Das Kontextmenü wird entsprechend erweitert. Der Aufruf des Kontextmenüs führt das SVN-Kommando `add` für die ausgewählte(n) Datei(en)/Verzeichnis(se) aus.

TortoiseSVN ist ein weit verbreitetes SVN Programm, das im Explorer als context menu integriert ist. Das Programm `TortoiseProc.exe` von TortoiseSVN kann auch in the Kommandozeile gestartet werden und zeigt einen Dialog an, der zur Eingabe durch den Benutzer dient. Somit können die Befehle, die im Kontextmenü im Explorer zugänglich sind auch in der Kommandozeile ausgeführt werden. Dies ermöglicht diese Funktionalität sehr einfach als Shell extension in CodeBlocks einzubauen. Zum Beispiel wird die folgende Eingabe

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

eine im File Explorer von CodeBlocks ausgewählte Datei gegen die SVN base Version verglichen. Siehe hierzu [Abbildung 2.9](#) auf Seite 39 wie dieses Kommando als Shell extension verfügbar wird.

**Hinweis:**

Für Dateien, die unter SVN stehen, werden im File Explorer overlay icons angezeigt, wenn diese über das Kontextmenü 'View' → 'SVN Decorators' aktiviert wurden.

**Beispiel**

Sie können den File-Explorer auch verwendet um Unterschiede zwischen verschiedenen Dateien oder Verzeichnisse anzeigen zu lassen. Dabei gehen Sie wie folgt vor.

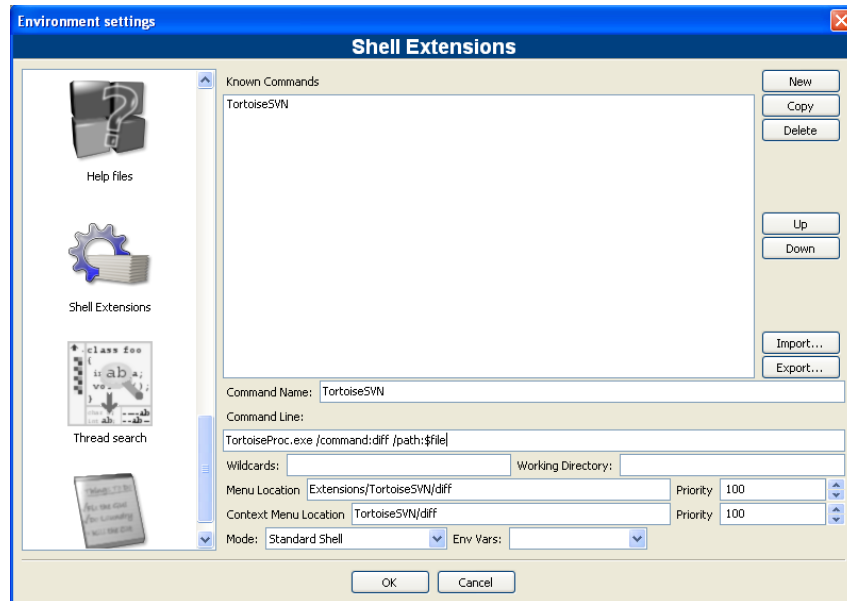


Abbildung 2.9: Hinzufügen von Aktionen für Kontextmenü

1. Fügen Sie im 'Settings' → 'Environment' → 'PowerShell' den Namen ein, der später im Menü ShellExtensions bzw. im Kontextmenü erscheinen soll.
2. Geben Sie den absoluten Pfad des Diff-Programms an (z.B. kdiff3). Dies wird über die Variable `$interpreter` zugegriffen.
3. Fügen Sie im unteren Feld den Eintrag:

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

In diesem Aufruf wird über die Variable `$mpaths` die im File-Explorer selektierten Dateien oder Verzeichnisse zugegriffen. Somit können Sie einfach ausgewählte Dateien oder Verzeichnisse gegeneinander vergleichen.

Als Übergabeparameter eines Befehl einer PowerShell unterstützt auch den Zugriff der in CodeBlocks verfügbaren Variablen (siehe [Abschnitt 3.2](#) auf Seite 56).

|                            |  |
|----------------------------|--|
| <code>\$interpreter</code> | Aufzurufendes Programm.                      |
| <code>\$fname</code>       | Name der Datei ohne Endung.                  |
| <code>\$fext</code>        | Dateiendung der ausgewählten Datei.          |
| <code>\$file</code>        | Dateiname mit Endung.                        |
| <code>\$relfile</code>     | Dateiname ohne Pfadangabe.                   |
| <code>\$dir</code>         | Ordnername mit Pfadangabe.                   |
| <code>\$reldir</code>      | Ordnername ohne Pfadanabe.                   |
| <code>\$path</code>        | Absoluter Pfad.                              |
| <code>\$relpath</code>     | Relativer Pfad einer Datei oder Verzeichnis. |



\$mpaths            Liste der ausgewählten Dateien oder Ordner.

\$inputstr{<msg>}    Zeichenkette die durch eine Eingabeaufforderung eingegeben wird.

\$parentdir        Übergeordnetes Verzeichnis (../)

**Hinweis:**

Die Einträge für Shell extension sind auch als Kontextmenü im Editor verfügbar.

## 2.8 Browse Tracker

Browse Tracker ist ein Plugin um zwischen kürzlich geöffneten Dateien in CodeBlocks zu navigieren. Dabei wird die Liste der kürzlich geöffneten Dateien in einer History gespeichert. Im Menü 'View' → 'Browse Tracker' → 'Clear All' können Sie die History löschen.

Das Fenster 'Browsed Tabs' zum Navigieren in dieser Listen erhalten Sie über das Menü 'View' → 'Browse Tracker' mit dem Eintrag 'Backward Ed/Forward Ed' oder über das Tastenkürzel Alt-Left/Alt-Right. Das Browse Tracker Menü ist auch über die Rechte Maustaste als Kontextmenü zugänglich. Die Marker werden in der Layout-Datei layout file <projectName>.bma gespeichert.

Eine häufige Arbeitsweise bei der Entwicklung von Software ist, dass man sich durch ein Satz von Funktion hangelt, die in unterschiedlichen Dateien implementiert sind. Durch das Plugin BrowseTracks können Sie somit komfortabel zwischen den Aufrufen in unterschiedlichen Dateien navigieren.

Das Plugin erlaubt auch Browse Marker in jeder Datei innerhalb des CodeBlocks Editor zu setzen. Die Cursor Position wird für jede Datei gespeichert. Das Setzen eines Markers innerhalb einer Datei ist wahlweise über das Menü 'View' → 'Browse Tracker' → 'Set BrowseMarks' oder durch einen Klick mit der linken Maustaste bei gehaltener Ctrl Taste möglich. Der Marker ist durch ... im linken Seitenrand gekennzeichnet. Über das Menü 'View' → 'Browse Tracker' → 'Prev Mark/Next Mark' oder das Tastenkürzel Alt-up/Alt-down kann zwischen den Marker innerhalb einer Datei gesprungen werden. Dabei werden die Marker beim Navigieren in der Reihenfolge angesprungen wie diese gesetzt wurden. Falls Sie die Marker innerhalb einer Datei nach Zeilennummern sortiert durchlaufen möchten, wählen Sie einfach das Menü 'View' → 'Browse Tracker' → 'Sort BrowseMark'.

Mit dem 'Clear BrowseMark' wird ein Marker in der ausgewählten Zeile gelöscht. Falls ein Marker für ein Zeile gesetzt ist, kann bei gehaltener linker Maustaste (1/4 Sekunde) und betätigen der Ctrl Taste der Marker für diese Zeile gelöscht werden. Mit dem Aufruf 'Clear All BrowseMarks' oder mit Ctrl-left Klick werden alle Marker innerhalb einer Datei zurückgesetzt.

Die Einstellungen für das Plugin können im Menü 'Settings' → 'Editor' → 'Browse Tracker' verändert werden.

**Mark Style** Standardmäßig werden Browse Marks durch ... im Seitenrand gekennzeichnet. Mit der Einstellung 'Book\_Marks' werden Browse Marks wie Bookmarks durch

einen blauer Pfeil im Rand dargestellt. Mit Hide werden die Darstellung von Browse Marks unterdrückt.

**Toggle Browse Mark key** Das Setzen oder Löschen von Marker kann entweder durch eine Klick mit der linken Maustaste oder bei gleichzeitig gehaltener Ctrl-Taste geschehen.

**Toggle Delay** Die Zeitspanne, die eine linke Maustaste gedrückt gehalten sein muss um in den Browse Marker Modus zu wechseln.

**Clear All BrowseMarks** Löschen aller Marker bei gehaltener Ctrl Taste entweder über einen einfachen Klick oder einen Doppelklick mit der linken Maustaste.

Die Konfiguration für das Plugin wird in den Anwendungsdaten in der Datei `default.conf` gespeichert. Bei der Verwendung einer Personality wird die Konfiguration aus der Datei `<personality>.conf` gelesen.

## 2.9 SVN Unterstützung

Eine Unterstützung für die SVN Versionskontrolle bietet das CodeBlocks Plugin TortoiseSVN. Im Menü 'TortoiseSVN' → 'Plugin settings' lässt sich im Reiter 'Integration' einstellen, wo die benutzerdefinierbaren SVN-Befehlen zur Verfügung stehen sollen.

**Menu integration** Fügt einen Eintrag TortoiseSVN mit Einstellmöglichkeiten in die Menüleiste ein.

**Project manger** Aktiviert die TortoiseSVN Befehle im Kontextmenü des Project Managers.

**Editor** Aktiviert die TortoiseSVN Befehle im Kontextmenü des Editors.

In den Plugin Settings lässt sich zusätzlich im Integration Dialog 'Edit main menu' beziehungsweise 'Edit popup menu' konfigurieren welche SVN Kommandos im Menü bzw. Kontextmenü ausgeführt werden können.

### Hinweis:

Der File Explorer in CodeBlocks bietet für die Anzeige des SVN Status unterschiedliche Farben als Icon Overlays. Auch hier ist das Kontextmenü für TortoiseSVN zugänglich.

## 2.10 LibFinder

Wenn Sie Bibliotheken in einer Anwendungen verwenden, muss Ihr Projekt so eingestellt werden, dass es nach diesen Bibliotheken sucht und diese anschließend benutzen kann. Dieser Vorgang kann zeitaufwändig und nervend sein, da jede Bibliothek unter Umständen durch unterschiedliche Arten von Option eingebunden werden muss. Des weiteren hängen die Einstellungen vom Host-Betriebssystem ab, was zu Inkompatibilitäten im Projekt für die Verwendung unter Unix und Windows führt.

LibFinder stellt folgende Kernfunktionalitäten zur Verfügung:

- Suche nach Bibliotheken die bereits auf einem System installiert sind
- Einbinden von Bibliotheken in Ihr Projekt und somit wird ein Projekt mit nur wenigen Mausklicks plattformunabhängig

### 2.10.1 Suche nach Bibliotheken

Die Suche nach Bibliotheken ist über das Menü 'Plugins' → 'Library finder' erreichbar. Der Sinn besteht in der Suche nach Bibliotheken, die bereits auf Ihrem System installiert sind. Das Ergebnis der Suche wird in der Libfinder Datenbank gespeichert. Diese Ergebnisse werden nicht in den CodeBlocks Projektdateien gespeichert. Die Suche startet mit dem Aufruf des Dialogs für die Angabe von Suchpfaden. LibFinder scannt diese Verzeichnisse rekursiv. Falls Sie nicht ganz sicher sind, wo sich die Bibliotheken befinden, ist auch die Angabe eines allgemeinen Pfades möglich. Sie können auch ganze Laufwerke angeben – in diesem Fall wird die Suche länger dauern aber voraussichtlich werden dann alle Bibliotheken gefunden (siehe [Abbildung 2.10](#) auf Seite 42).

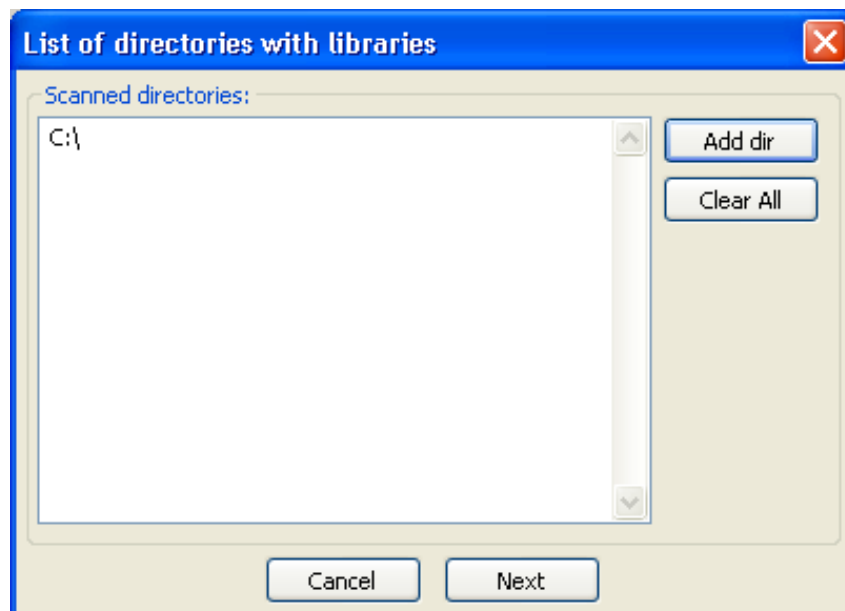


Abbildung 2.10: Liste für Suchpfade

Wenn LibFinder nach Bibliotheken sucht, verwendet es spezielle Regeln um das Vorhandensein von Bibliotheken zu erkennen. Jeder Satz an Regeln ist in einer xml Datei abgelegt. Derzeit unterstützt LibFinder die Suche von wxWidgets 2.6/2.8, CodeBlocks SDK and GLFW – die Liste wird zukünftig erweitert werden.

**Hinweis:**

Für nähere Informationen wie eine Unterstützung für weitere Arten von Bibliotheken eingefügt werden kann, lesen Sie bitte die Datei `src/plugins/contrib/lib_finder/lib_finder/readme.txt` im Quellverzeichnis von CodeBlocks.

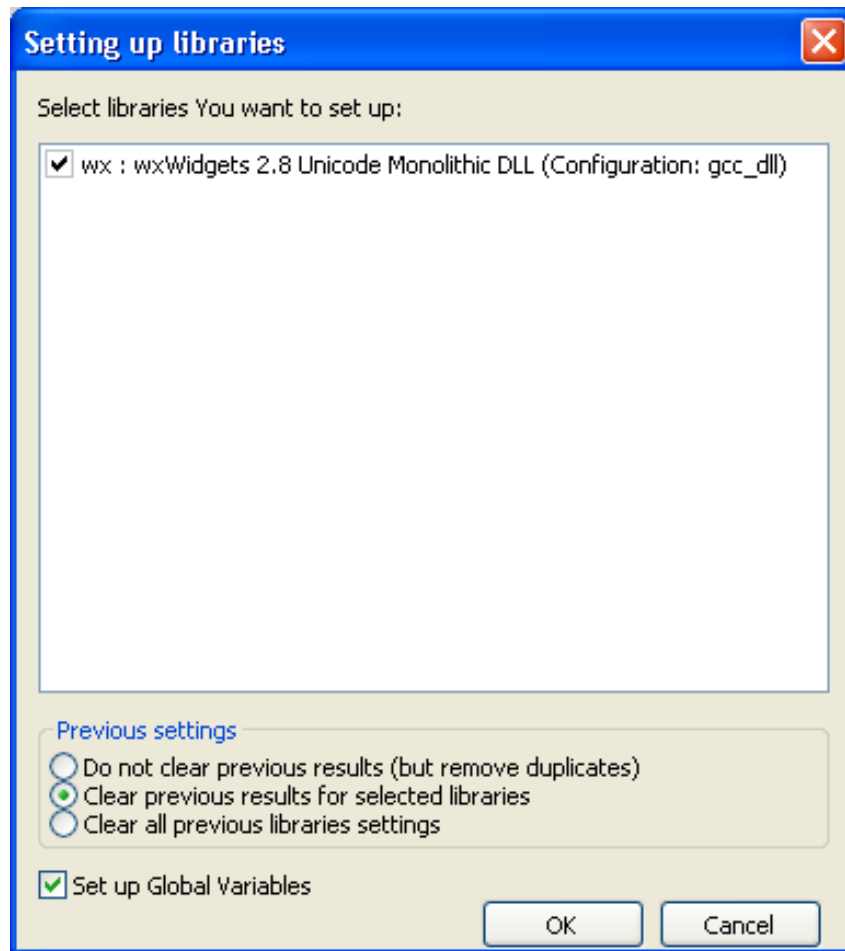


Abbildung 2.11: Suchergebnisse

Nach der Suche, zeigt Libfinder die Suchergebnisse an (siehe [Abbildung 2.11](#) auf Seite 43).

In der Liste wählen Sie dann die Bibliotheken aus, die in der Libfinder Datenbank gespeichert werden sollen. Beachten Sie das jede Bibliothek mehrere gültige Konfigurationen haben kann und die Einstellungen aus vorhergehenden Suchen für die Erzeugung eines Projektes dominieren.

Mit den nachfolgenden Einstellungen lässt sich konfigurieren, wie mit den Ergebnissen aus vorhergehenden Suchen umgegangen wird.

**Do not clear previous results** Diese Option funktioniert wie ein Update eines existierenden Ergebnis – es fügt neue hinzu und aktualisiert bereits bestehende. Die Option ist nicht ratsam.

**Second option (Clear previous results for selected libraries)** Löscht alle Suchergebnisse für Bibliotheken, die vor der Suche ausgewählt wurden. Die Verwendung dieser Option wird empfohlen.

**Clear all previous library settings** wenn diese Option ausgewählt ist, wird die LibFinder Datenbank aufgeräumt bevor neue Suchergebnisse eingefügt werden. Dies ist sinnvoll wenn Sie ungültige Einträge aus der LibFinder Datenbank entfernen wollen.

Eine weitere Alternative in diesem Dialog ist die Einstellung 'Set up Global Variables'. Wenn diese Option ausgewählt ist, versucht LibFinder automatisch die globalen Variablen zu konfigurieren und den Umgang mit den Bibliotheken zu erleichtern.

Wenn Sie pkg-config auf Ihrem System installiert haben (ist meist auf Linux Systemen installiert), wird LibFinder auch die Bibliotheken aus diesem Tool verwenden. Es ist keine weitere Suche erforderlich, da diese beim Start von CodeBlocks automatisch geladen werden.

## 2.10.2 Einbinden von Bibliotheken in Projekten

LibFinder fügt in Project Properties einen weiteren Reiter 'Libraries' ein – diese Reiter zeigt die Bibliotheken an, die im Projekt verwendet werden und LibFinder bekannt sind. Um Bibliotheken in ein Projekt einzufügen, wählen Sie einfach einen Eintrag im rechten Ausschnitt und Klicken Sie den < Knopf. Das Entfernen einer Bibliothek aus einem Projekt geschieht durch Auswahl eines Eintrages im linken Ausschnitt und einen Klick auf den > Knopf (siehe [Abbildung 2.12](#) auf Seite 44).

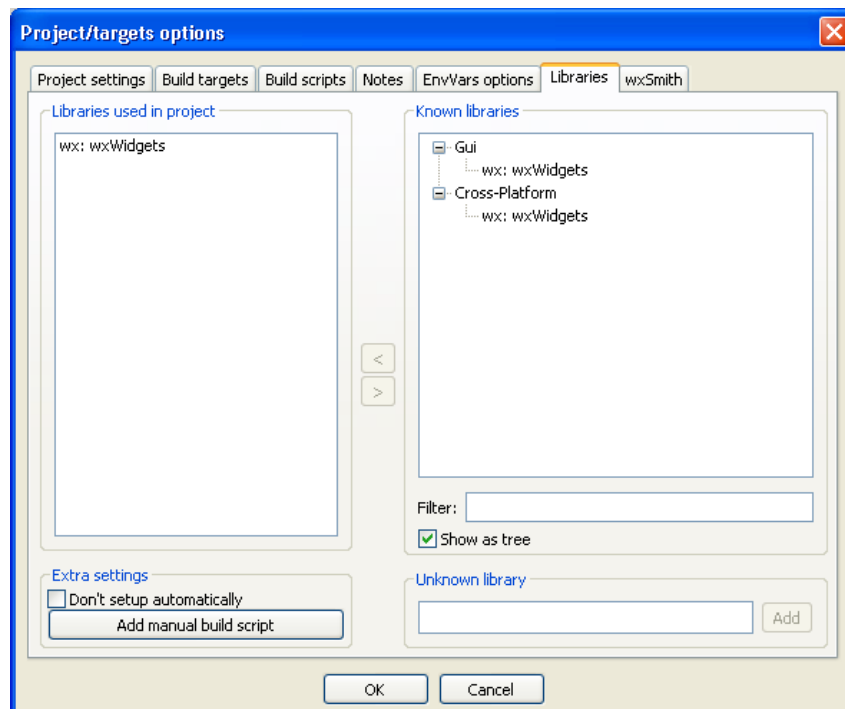


Abbildung 2.12: Project configuration

Die Anzeige von Bibliotheken, die LibFinder bekannt sind, kann gefiltert werden. Die Checkbox 'Show as Tree' erlaubt das Umschalten zwischen kategorisiert und nicht kategorisierter Ansicht.

Wenn Sie Bibliotheken, die nicht in der LibFinder Datenbank verfügbar sind, einfügen wollen, wählen Sie den Eintrag 'Unknown Library'. Sie sollte für die Angabe der Bibliothek das übliche Kürzel verwenden (entspricht normalerweise dem globalen Variablennamen) oder den Name der Bibliothek in pkg-config. Eine Liste von empfohlen Shortcodes finden Sie auf [Global Variables](#). Die Verwendung dieser Option ist nur dann ratsam, wenn

ein Projekt auf unterschiedlichen Systemen erzeugt werden soll, wo die erforderlichen Bibliotheken existieren und durch LibFinder ermittelt werden können. Der Zugriff auf eine globale Variable innerhalb von CodeBlocks sieht wie folgt aus:

```
$ (#GLOBAL_VAR_NAME.lib)
```

Die Auswahl der Option 'Don't setup automatically' wird LibFinder anweisen die Bibliotheken nicht automatisch beim Kompilieren des Projektes einzubinden. In einem solchen Fall kann LibFinder aus einem Build Script ausgeführt werden. Ein Beispiel für ein solches Skript wird durch Auswahl des Menüs 'Add manual build script' erzeugt und dem Projekt hinzugefügt.

### 2.10.3 Verwendung von LibFinder und aus Wizards erzeugten Projekten

Wizards erzeugen Projekte, die nicht LibFinder nutzen. Die Verwendung des Plugins erfordert, dass der Benutzer die Einstellung in den Build options im Reiter 'Libraries' anpasst. Die Vorgehensweise sieht so aus, dass alle Bibliothek spezifische Einstellungen entfernt werden müssen und die benötigten Bibliotheken im Reiter 'Libraries' eingefügt werden.

Diese Art von Projekten werden somit unabhängig von dem verwendeten Betriebssystem. Solange nur Bibliotheken, die in der LibFinder Datenbank definiert wurden, verwendet werden, werden die Build Optionen eines Projektes automatisch aktualisiert, so dass die Einstellung auch für die plattformabhängigen Einstellungen von Bibliotheken funktionieren.

## 2.11 AutoVersioning

Ein Plugin zur Versionierung von Anwendungen, indem die Versions- und Buildnummer einer Anwendung jedesmal hochgezählt wird, wenn eine Änderung stattgefunden hat. Diese Information wird über einfach benutzbare Variablendeklarationen in der Datei `version.h` abgelegt. Des weiteren sind möglich: Übergaben im SVN Stil, ein Versionsschema Editor, ein Change Log Generator und ein Log Generator und vieles mehr ...

### 2.11.1 Einleitung

Die Idee dieses Plugins entstand bei Entwicklung von Software, die sich im frühen pre-alpha Status befand und eine Art von Versionsinformation benötigte. Beschäftigt durch die Erstellung von Code, blieb keine Zeit um die Versionsnummer zu pflegen, deshalb wurde ein Plugin entwickelt, dass diese Arbeit erledigt und nur minimaler Bedienereingriff erfordert.

### 2.11.2 Features

Hier finden Sie eine Liste von Features, die vom Plugin abgedeckt werden.

- Unterstützung für C und C++.

- Generiert und auto inkrementiert Versionsvariablen.
- Software status editor.
- Integrierter Schemeneditor für die Konfiguration wie automatische Hochzählen der 'version values' geschehen soll.
- Datum deklariert als Monat, Datum und Jahr.
- Ubuntu style version.
- Svn revision check.
- Change log generator.
- Funktioniert unter Windows und Linux.

### 2.11.3 Handhabung

Wählen Sie einfach das Menü 'Project' → 'Autoversioning' . Das Pop Up Fenster wie auf ?? auf Seite ?? erscheint.

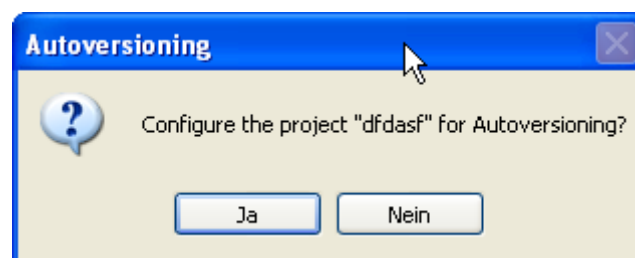


Abbildung 2.13: Configure project for Autoversioning

Wenn Sie den Dialog mit yes bestätigen, dann wird der Konfigurationsdialog von Autoversioning angezeigt.

Nachdem Sie Ihr Projekt für Autoversioning konfiguriert haben, werden die Einstellungen aus dem Eingabedialog im Projekt gespeichert und eine Datei `version.h` wird angelegt. Ab diesem Zeitpunkt wird bei jedem Aufruf des Menüs 'Project' → 'Autoversioning' der Konfigurationsdialog aufgerufen, um die Einstellung für Projektversion vorzunehmen, es sei denn Sie speichern die Änderungen des Plugins in Projektdatei.

### 2.11.4 Dialog notebook tabs

#### 2.11.4.1 Version Values

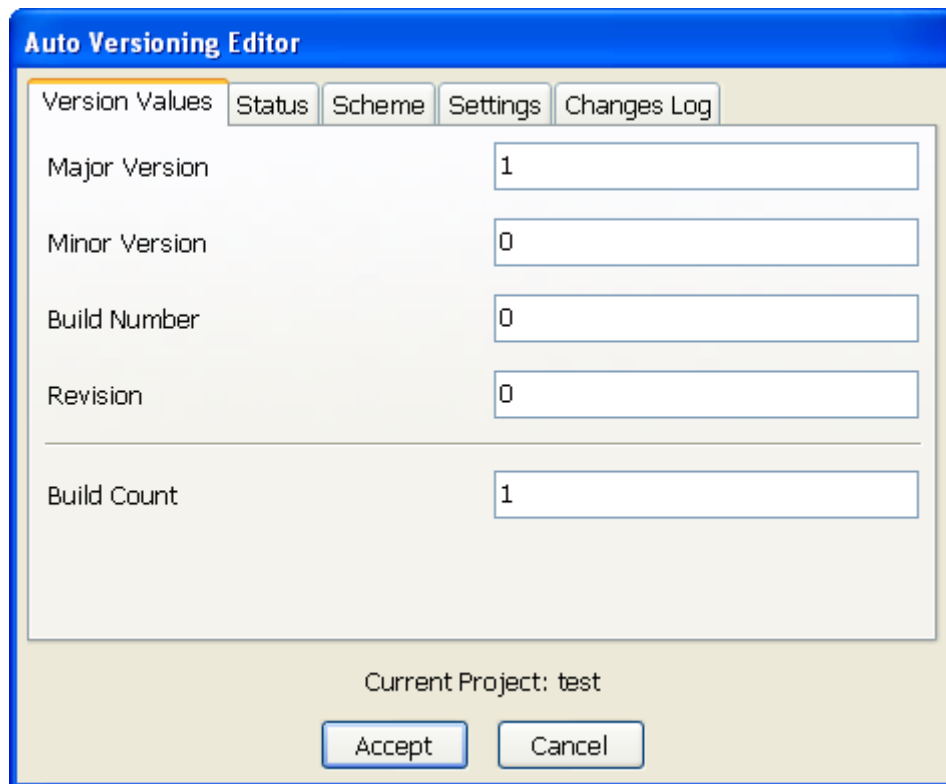
Hier können Sie einfach die zugehörigen Version Values eintragen oder Auswählen ob Auto Versioning diese für Sie hochzählt (siehe [Abbildung 2.14](#) auf Seite 47).

**Major** Wird um eins hochgezählt wenn die Minor Version ihr Maximum erreicht

**Minor** Wird um eins hochgezählt wenn die Anzahl von Build die Schranke build times überschreitet. Der Wert wird auch Null zurückgesetzt, nachdem die maximale Anzahl bereits erreicht wurde.

**Build Number** Gleichbedeutend mit Release und wird jedesmal wenn die Revision Nummer steigt um eins hochgezählt.

**Revision** Zählt die Revision zufallsartig hoch, wenn das Projekt geändert oder kompiliert wurde.



The image shows a dialog box titled "Auto Versioning Editor". It has five tabs: "Version Values", "Status", "Scheme", "Settings", and "Changes Log". The "Version Values" tab is active. Inside this tab, there are five input fields with labels to their left: "Major Version" (value: 1), "Minor Version" (value: 0), "Build Number" (value: 0), "Revision" (value: 0), and "Build Count" (value: 1). Below these fields, it says "Current Project: test". At the bottom of the dialog are two buttons: "Accept" and "Cancel".

Abbildung 2.14: Set Version Values

#### 2.11.4.2 Status

Einige Felder sind auf vordefiniert Werte voreingestellt (siehe [Abbildung 2.15](#) auf Seite 48).

**Software Status** Ein typisches Beispiel wäre v1.0 Alpha

**Abbreviation** Gleichbedeutend mit Software Status, aber in der Form: v1.0a

#### 2.11.4.3 Scheme

Hier stellen Sie ein, wie das Plugin die version values hochzählt (siehe [Abbildung 2.16](#) auf Seite 48).

**Minor maximum** Die obere Schranke für den Wert Minor. Wenn diese erreicht ist, wird Major hochgezählt und beim nächsten Kompilervorgang des Projektes wird Minor auf Null zurückgesetzt.

**Build Number maximum** Wenn der Wert erreicht wurde, wird beim nächsten Kompilervorgang der Wert auf Null zurückgesetzt. Die Einstellung 0 setzt das Maximum auf unendlich



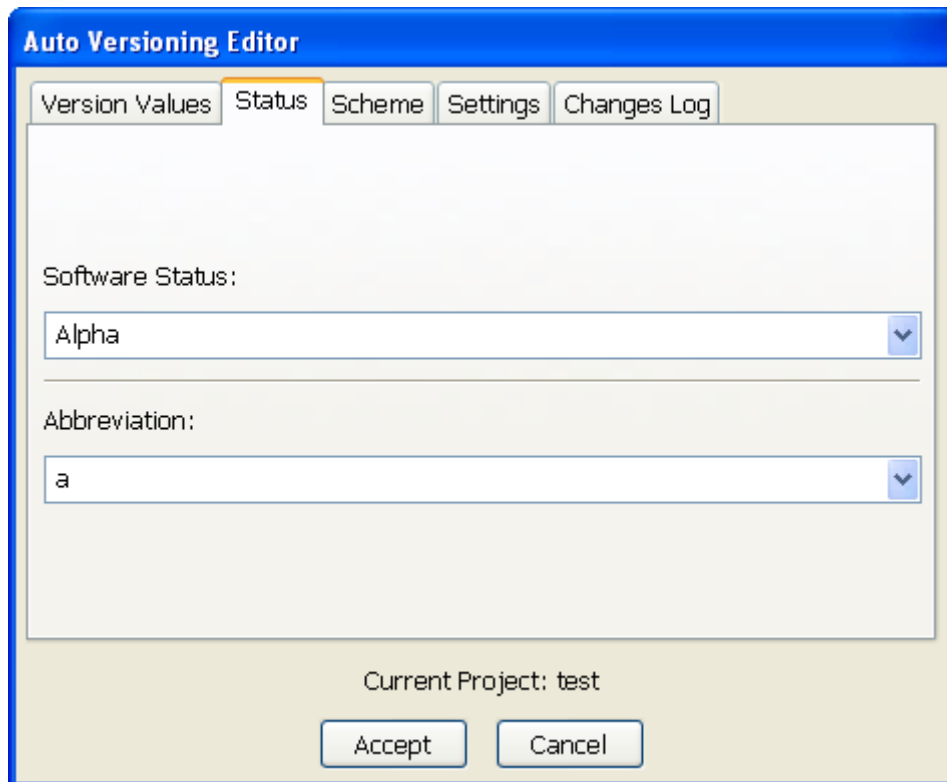


Abbildung 2.15: Setzen Status von Autoversioning

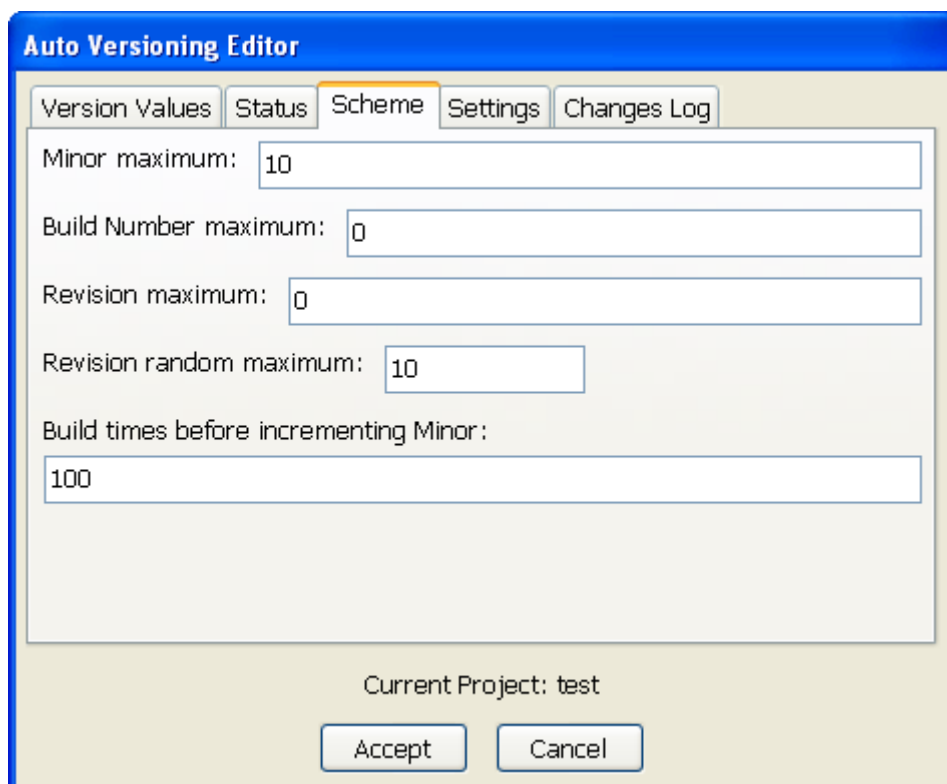


Abbildung 2.16: Scheme of autoversioning

**Revision maximum** Gleichbedeutend mit Maximum für Build Number maximum. Die Einstellung 0 setzt das Maximum auf unendlich

**Revision random maximum** Die Revisions Nummer wird durch Zufallszahlen hochgezählt. Eine Einstellung mit 1, wird die Revision um eins erhöhen.

**Build times before incrementing Minor** Nach Änderungen im Code und Kompilierung wird die Build History inkrementiert und wenn dieser Wert erreicht wird, dann wird der Minor Wert auch inkrementiert.

#### 2.11.4.4 Einstellungen

Hier können Sie einige Einstellungen für Auto Versioning vornehmen (siehe [Abbildung 2.17](#) auf Seite 49).

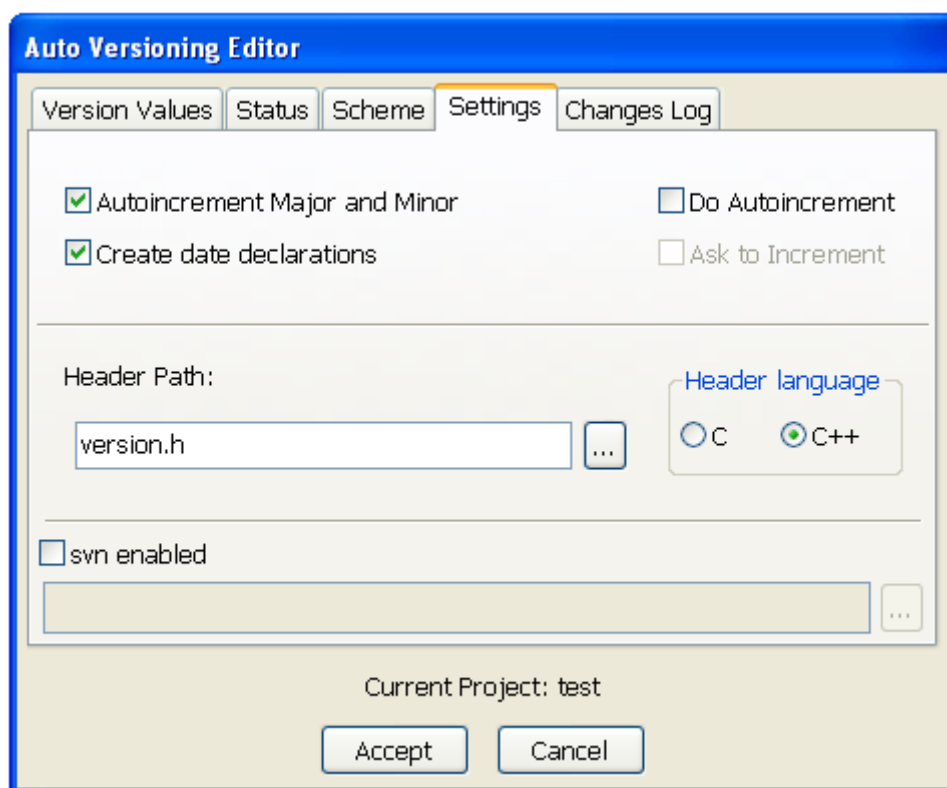


Abbildung 2.17: Settings von Autoversioning

**Autoincrement Major and Minor** Lassen Sie das Plugin nach diesem Schema den Wert inkrementieren. Wenn es nicht ausgewählt wurde, dann wird nur die Build Number und die Revision hochgezählt.

**Create date declarations** Erzeugt in der Datei `version.h` Einträge für Datum und Ubuntu style version.

**Do Auto Increment** Weist das Plugin an bei jeder Änderung noch vor dem Kompilervorgang zu inkrementieren.

**Header language** Einstellung der Sprache für Ausgabe in `version.h`

**Ask to increment** Wenn Do Auto Increment aktiv ist, wird vor dem Kompilervorgang bei Änderungen nachgefragt, ob hochgezählt werden soll.

**svn enabled** Sucht nach der SVN Revision und Datum im aktuellen Verzeichnis und erzeugt die zugehörigen Einträge in `version.h`

#### 2.11.4.5 Changes Log

Durch diese Einstellung wird die Eingabe für jegliche Änderung an einem Projekt in die Datei `ChangesLog.txt` generiert (siehe [Abbildung 2.18](#) auf Seite 50).

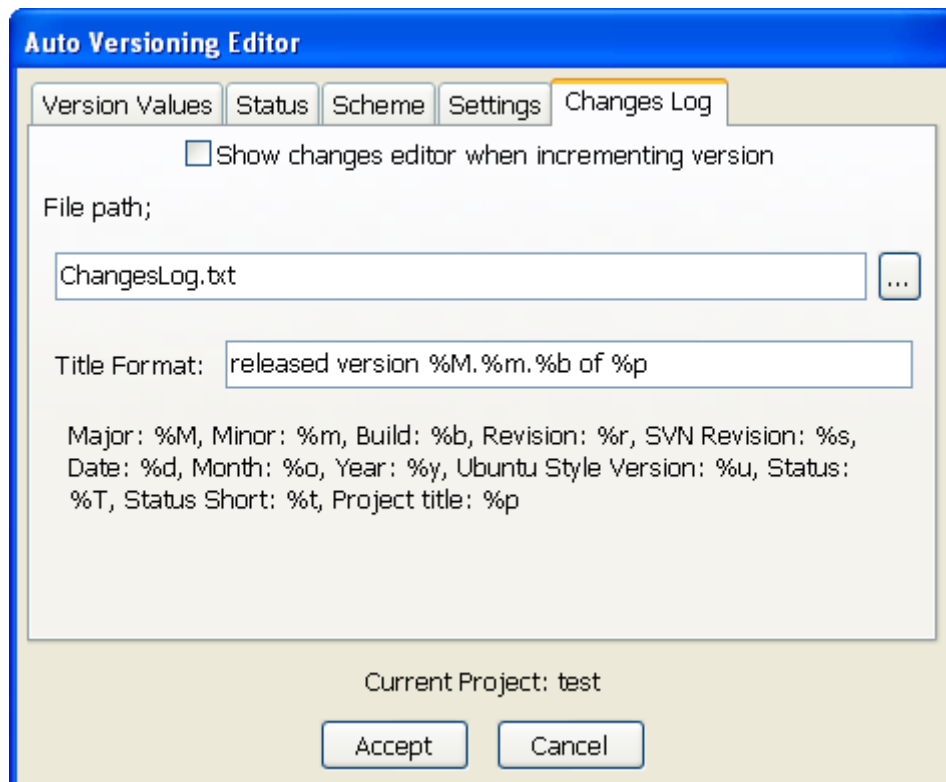


Abbildung 2.18: Changelog von Autoversioning

**Show changes editor when incrementing version** Ruft den Changes log editor auf, wenn die Version inkrementiert wird.

**Title Format** Format für Title mit einer Liste von vordefinierten Werten.

#### 2.11.5 Einbinden in den Quellen

Für die Verwendung der Variablen, die durch das Plugin erzeugt wurden, müssen Sie die Datei `#include <version.h>` in den Quellen einfügen. Ein Beispiel für eine Quelle könnte wie folgt aussehen:

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

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

### 2.11.5.1 Ausgabe von version.h

Die erzeugte Headerdatei könnte beispielsweise im C++ Mode wie folgt aussehen:

```
#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
```

Bei der Einstellung der Sprache C ergibt sich folgende Ausgabe ohne Namespaces:

```
#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

Dieser Dialog ist über das Menü 'Project' → 'Changes Log' erreichbar. Diese Dialog erscheint auch wenn die Einstellung 'Show changes editor' für das Inkrementierung der Version (Changes Log) besteht. Im Dialog werden die Liste von Änderungen nach Modifikation der Quellen eines Projektes eingegeben (siehe [Abbildung 2.19](#) auf Seite 52).

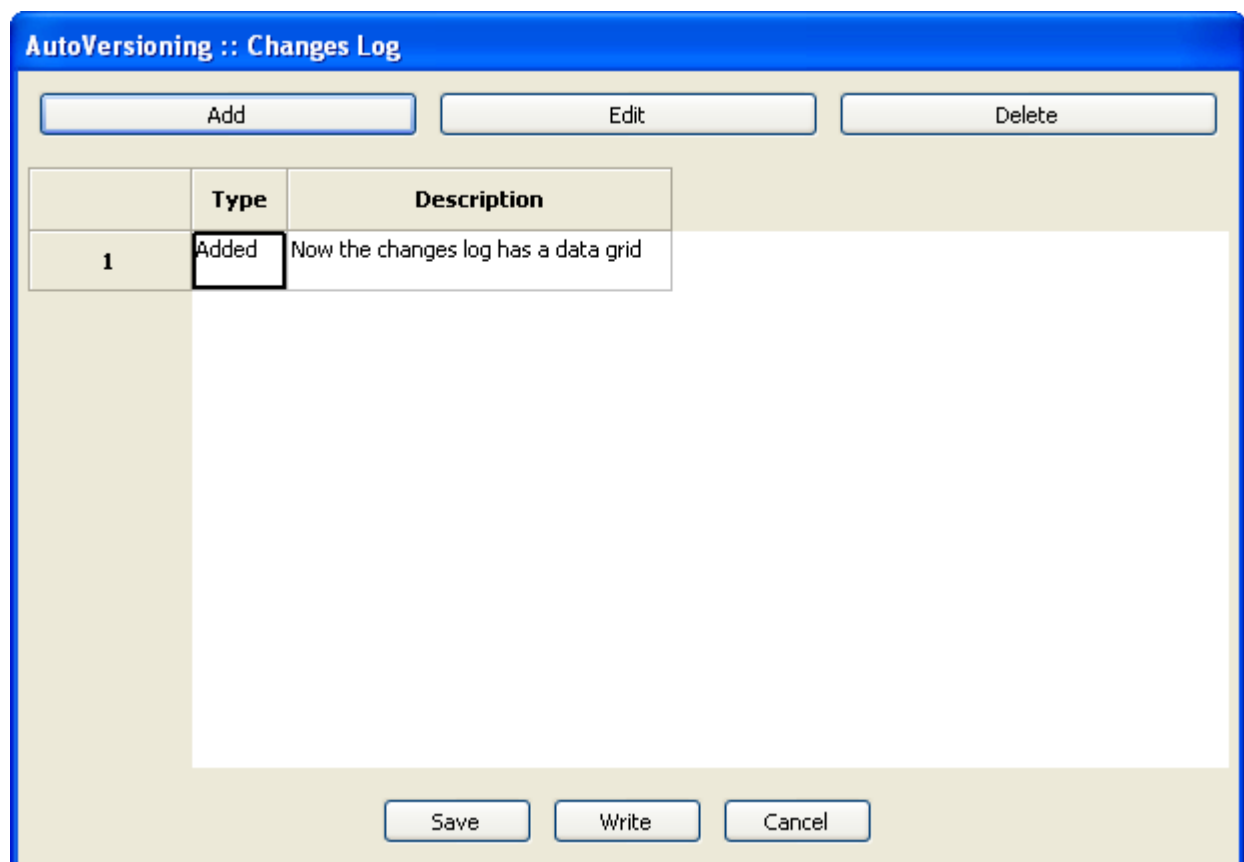


Abbildung 2.19: Changes for a project

### 2.11.6.1 Buttons Summary

**Add** Fügt eine Zeile in der Liste hinzu

**Edit** Editieren einer ausgewählte Zelle

**Delete** Löscht die ausgewählte Zeile aus der Liste

**Save** Speichert die aktuellen Daten temporär in der Datei (`changes.tmp`). Diese Information wird später für die Ausgabe in Changes Log verwendet

**Write** Speichert die Eingabe in der Changes Log Datei

**Cancel** Beendet den Dialog

Hier ein Beispiel für eine Datei `ChangesLog.txt`, die durch Auto Versioning erzeugt wurde.

```
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

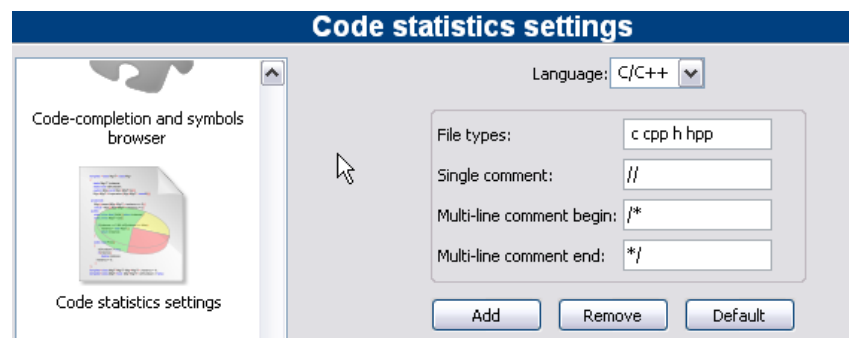


Abbildung 2.20: Konfiguration für Code Statistik

Anhand der Angaben in der Konfigurationsmaske ermittelt dieses einfache Plugin die Anteile von Code, Kommentaren und Leerzeilen für ein Projekt. Die Auswertung wird über das Menü 'Plugins' → 'Code statistics' durchgeführt.

## 2.13 Suche nach verfügbaren Quellencodes

Dieses Plugin ermöglicht es, einen Begriff im Editor zu markieren und über das Kontextmenü 'Search at Koders' in der Datenbank von [[↔Koders](#)] zu suchen. Dabei bietet der Eingabedialog zusätzlich die Möglichkeit, die Suche nach Programmiersprachen und Lizenzen zu filtern.

Durch diese Datenbanksuche finden Sie schnell Quellcode der aus anderen weltweiten Projekten von Universitäten, Consortiums und Organisationen wie Apache, Mozilla, Novell Forge, SourceForge und vielen mehr stammt und wiederverwendet werden kann, ohne dass jedes Mal das Rad neu erfunden werden muss. Bitte beachten Sie die jeweilige Lizenz des Quellcodes.

## 2.14 Code profiler

Eine einfache grafische Schnittstelle für das Profiler Programm GNU GProf.

## 2.15 Symbol Table Plugin

Diese Plugin ermöglicht die Suche von Symbolen in Objekten und Bibliotheken. Dabei werden die Optionen und der Pfad für das Kommandozeilen Programm nm über den Reiter Options konfiguriert.

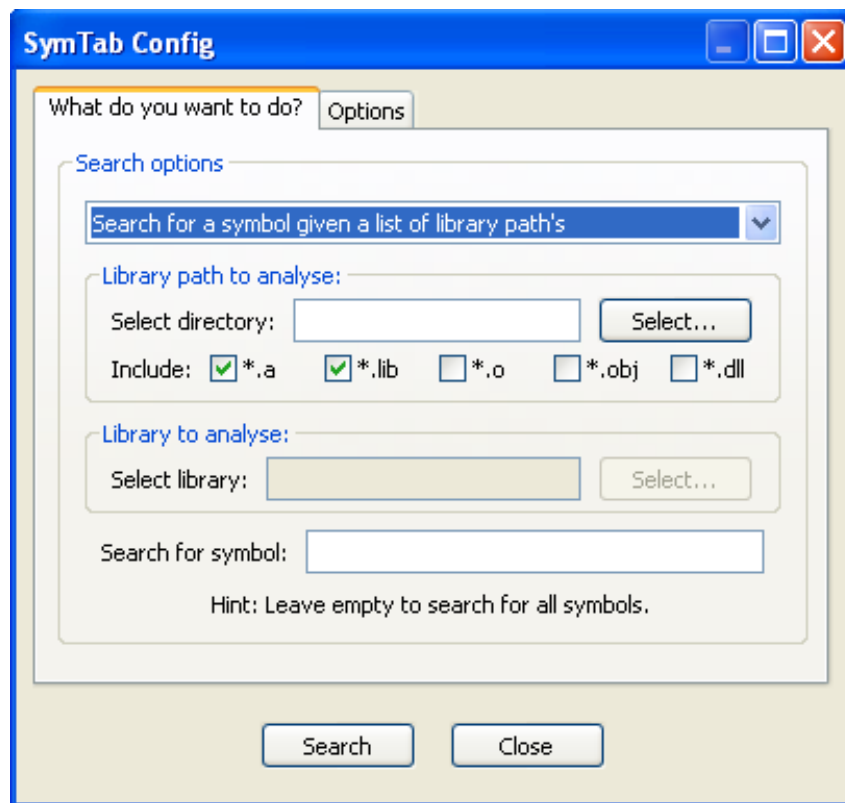


Abbildung 2.21: Konfiguration von Symbol Table

Mit der Schaltfläche 'Search' wird die Suche gestartet und die Ergebnisse des Programms NM werden in einem eigenen Fenster SymTabs Result angezeigt. Der Name des Objekts bzw. Bibliothek, die das Symbol enthalten ist unter dem Titel NM's Output gelistet.

## 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 [Abschnitt 3.2](#) auf Seite 56 for details).

**Command Macros** This type of variables is used for controlling the build process. For further information please refer to [Abschnitt 3.4](#) auf Seite 60.

**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



[Abschnitt 3.7](#) auf Seite 60 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.

**Hinweis:**

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 <b>plugins</b> 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

**Hinweis:**

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.).

**Hinweis:**

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 [Unterabschnitt 3.12.1](#) auf Seite 64 and [Abbildung 3.1](#) auf Seite 62 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

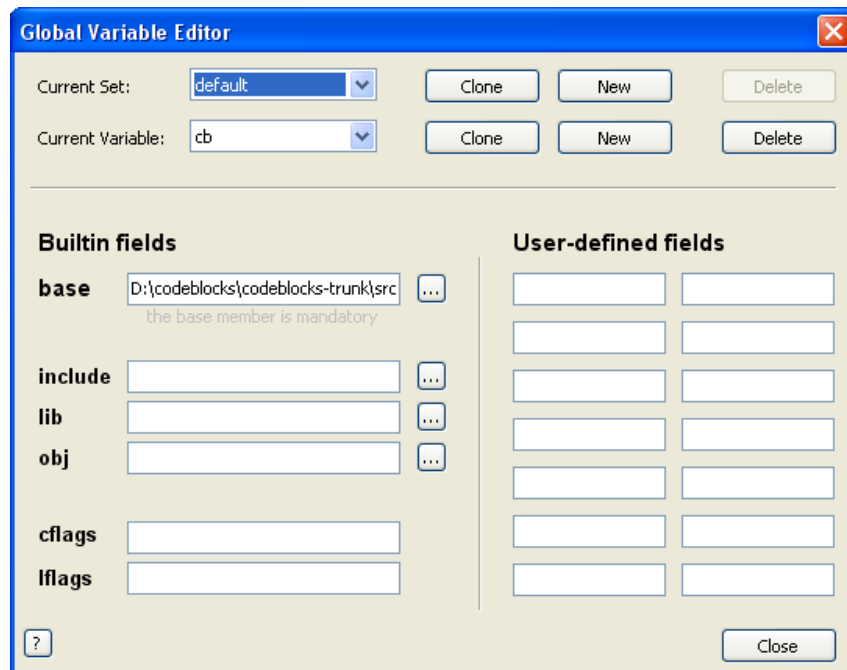


Abbildung 3.1: Global Variable Environment

- 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

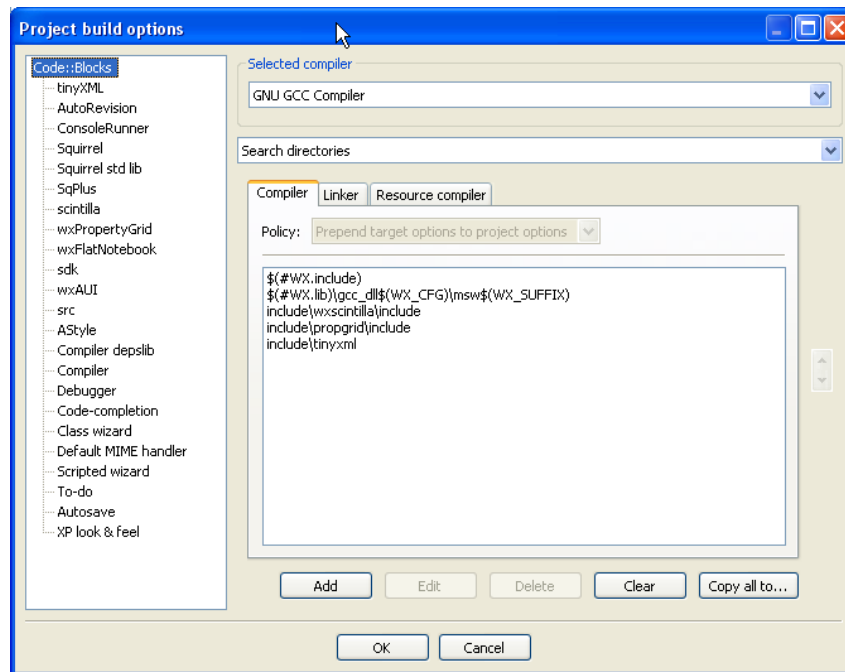


Abbildung 3.2: Global Variables

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