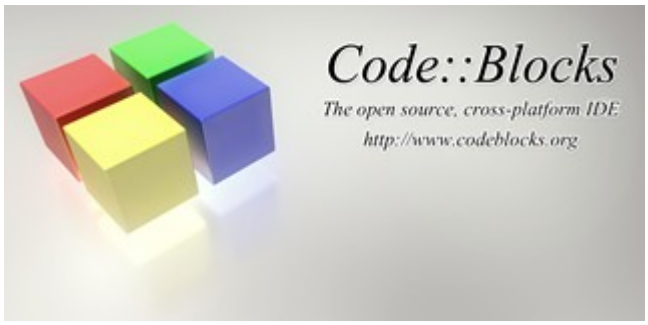


Verzeichnis

Entwicklungsumgebung

Diese Seiten sind eine Kurzanleitung zur Verwendung der C++-Entwicklungsumgebung Code::Blocks. Sie sollen echten Anfängern die ersten Schritte mit dieser plattformübergreifenden Entwicklungsumgebung (kurz: IDE von Integrated Development Environment) erleichtern.

Zur Erzeugung eines ablauffähigen Programms sind ein Compiler (mit Präprozessor), ein Linker und Bibliotheken erforderlich. Eine IDE ist zum Erstellen von Programmen nicht unbedingt erforderlich. Wer möchte, kann ein Programm auch mit Kommandos in einem Kommandozeilenfenster (manchmal auch DOS-Box genannt) erzeugen lassen. Eine IDE unterstützt einfach nur das Erstellen von Programmen. Dazu integriert sie typischerweise

- die Verwaltung von Programmierprojekten
- die Verwaltung von Suchpfaden für Dateien
- einen Editor mit Syntaxhighlighting (Hervorhebung oder farbliche Gestaltung von syntaktischen Elementen der Programmiersprache)
- das Starten von Compiler und Linker
- Fehlersuche auf Quelltextebene (Source-Level-Debugging)

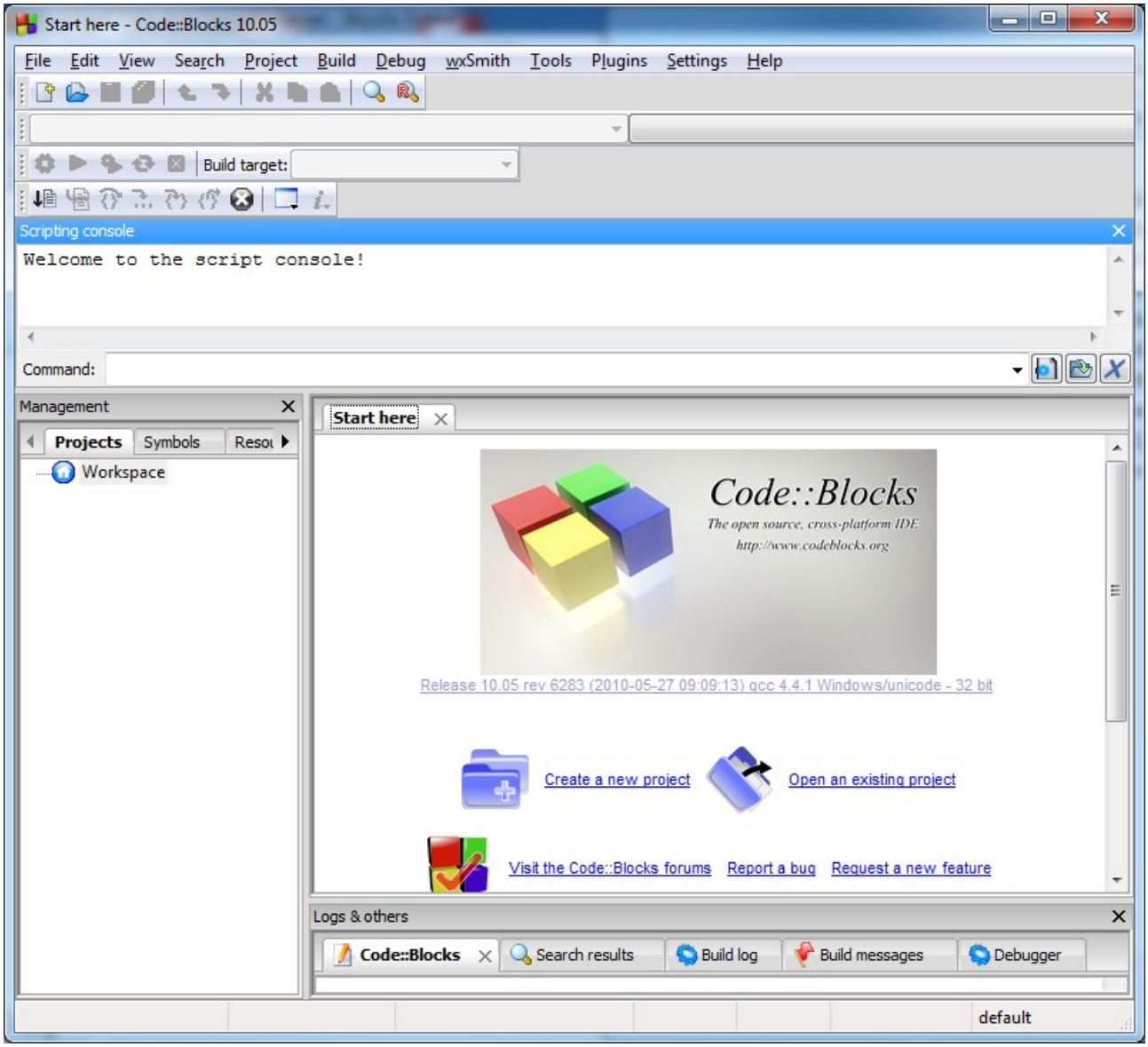


Abbildung 1: Erscheinungsbild der IDE Code::Blocks

Die von mir verwendete Installation läuft auf Windows und sie enthält MinGW. Dies ist der Minimale GNU Compiler für Windows. Sollten Sie mit einem Linux arbeiten, muss dort das entsprechende C++-Paket installiert sein. Code::Blocks ist genau genommen nur die IDE, die zum Erstellen von Programmen ein solches C++-Paket benötigt.

Für MacOS installieren Sie bitte eine dort lauffähige C++-IDE. Sollten Ihnen mehrere IDEs zur Verfügung stehen, wählen Sie eine möglichst einfache, die Source-Level-Debugging unterstützt!

Ballast

Diese Anleitung soll Sie als Anfänger mit den ersten erforderlichen Schritten zur erfolgreichen Erstellung eines Programms vertraut machen. Code::Blocks bietet einige fortgeschrittene Möglichkeiten, die IDE für eigene Zwecke effizient zu gestalten. Auch erstellt sie Protokolle, die einen Programmierer in größeren Projekten bei der Fehlersuche unterstützen. Alle diese Hilfsmittel sind gut und zweckmäßig, überfordern aber Anfänger. Deshalb lasse ich deren Erörterung weg und fokussiere ausschließlich die Dinge, die Sie für das Anfängerprogrammieren brauchen. Sie sollten nun das Teilfenster ***Scripting console*** mit Hilfe des Schließen-Kreuzes (rechts) schließen, damit die IDE etwas weniger Ballast für Sie bietet!

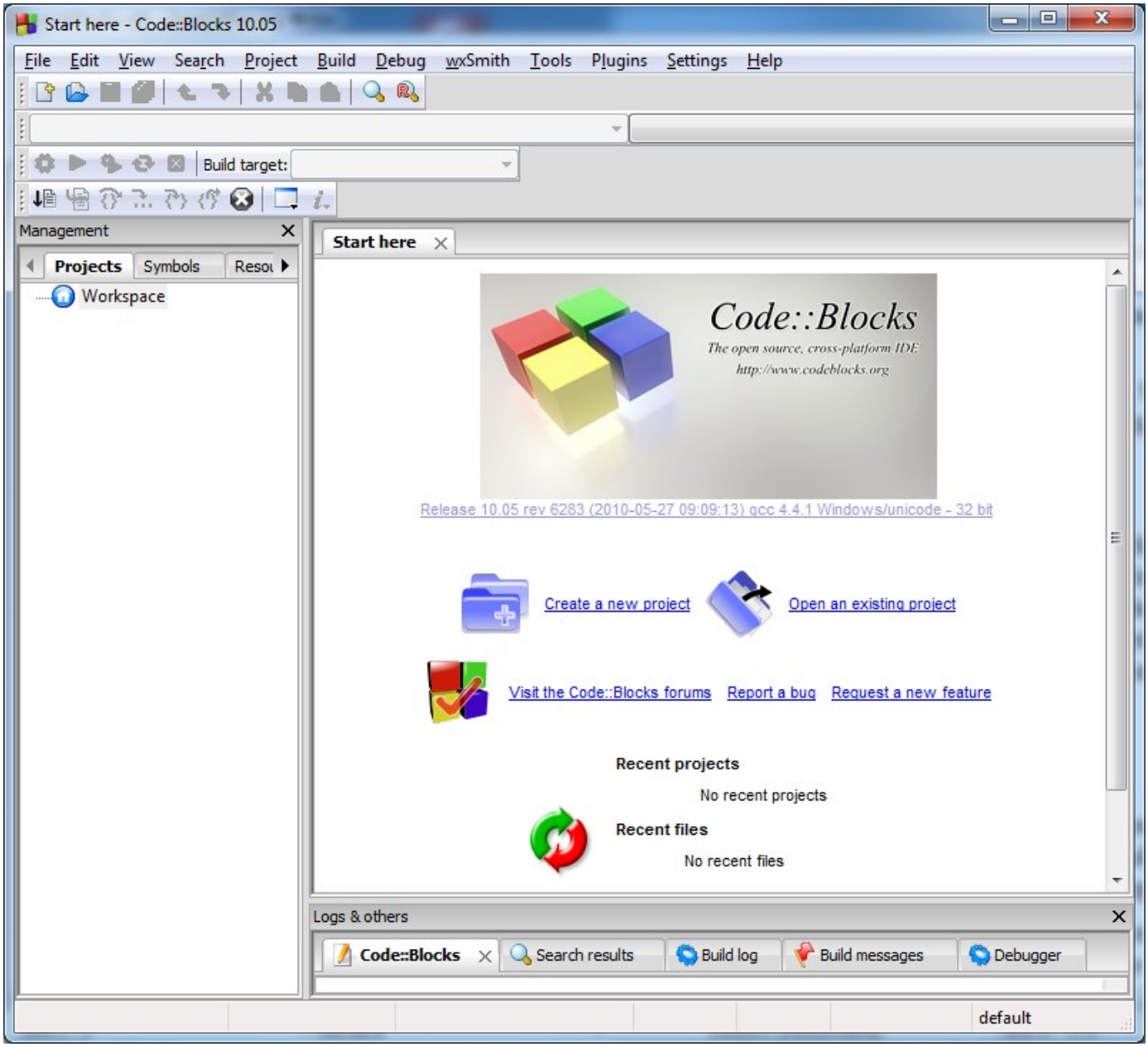


Abbildung 2: Die IDE ohne Einblendung der Scripting console

GUI-Anwendungen vs. Konsolenanwendungen

Eine Anwendung im IT-Sinn ist ein Programm, das ein Computerbenutzer bewusst einsetzt. Solche Anwendungen sind Archivierer, Textverarbeitungen, Tabellenkalkulationen, Bildbearbeitungen, Media-Abspieler, Web-Browser, FTP-Client, DVD-Brennen, Programmier-IDE, …

Es gibt viele Programme, die ein Benutzer unbewusst verwendet. Dazu gehören insbesondere Dienstprogramme, die im Hintergrund arbeiten und ohne die die heutige IT nicht funktionsfähig wäre. Beisp.:
Ein Benutzer/Anwender startet einen Web-Browser, um sich im WWW des Internet umzusehen. Das kann er nur dann erfolgreich, wenn auf seinem Computer ein Netzwerkdienst läuft, verschiedene Dienste des Internet-Service-Providers (ISP) fehlerfrei arbeiten und schließlich hinter der gewünschten Adresse ein Webserver seine Arbeit tut. Vielleicht sind noch ein LDAP-Server und ein Datenbank-Server beteiligt. Alle diese Dienste sind nicht wirklich sichtbar. Sie stellen aber die erforderliche Grundlage unserer IT dar.

Die allermeisten Anwendungen bieten dem Benutzer eine GUI-Mensch-Maschinen-Schnittstelle. GUI steht für Graphical User Interface. Nach dem Starten einer GUI-Anwendung erscheint mindestens ein Fenster auf der Arbeitsoberfläche (Desktop), das in der Regel eine Menüstruktur, eine Mauspalette, ein Schließfeld u.s.w. zur Verfügung stellt.

Konsolenanwendungen bieten keine grafischen Elemente. Sie können in der Regel nicht mit einer Maus bedient werden. Sie sind nicht chic. Sie besitzen aber einen entscheidenden Vorteil gegenüber GUI-Anwendungen: Sie sind insbesondere für Anfänger viel leichter zu programmieren.

Hype: Objektorientiertes Programmieren (OOP)

Mit C++ kann man sehr gut objektorientiert programmieren, man muss es aber nicht. Ich befürworte für Programmieranfänger das bottom-up-Verfahren. Dieses sieht vor, zuerst die in jedem Programm erforderlichen Grundlagen kennenzulernen und frühzeitig einfache Programme zu erstellen. Diese Grundlagen sind Programmstruktur, Einlesen und Ausgeben, Verwendung von Variablen und Objekten, Ablaufstrukturen, Modularisieren per Unterprogramme und einfache Datenstrukturen. Alle diese Dinge haben noch nichts mit OOP zu tun. Wer ernsthaft objektorientiert programmieren will, muss jedoch diese Grundlagen beherrschen.

GUI-Anwendungen werden in C++ fast ausschließlich objektorientiert erstellt – unter Verwendung von zumindest einer Bibliothek. Ein geeigneter Lernweg zum Programmieren von GUI-Anwendungen ist:

1. Programmiertechnische Grundlagen (s.o.) aneignen
2. Bibliotheken einsetzen
3. Klassen schreiben
4. Klassenhierarchie planen und aufbauen
5. GUI-Anwendungen programmieren

Es sollte spätestens jetzt plausibel sein, warum ich für Anfänger im Rahmen der programmiertechnischen Grundlagen ausschließlich Konsolenanwendungen vorsehe.

Erstes Programm in C++

Ein Programm ist das Ergebnis eines Programmierprojektes. Eine IDE verwaltet ein solches Projekt, indem sie dessen Bestandteile registriert. Das Projekt besteht in der Regel aus mehreren Quelltextdateien, die Bibliotheken verwenden, und aus sog. Headerdateien, die Beschreibungen zu den Bibliotheken enthalten. Die Projektverwaltung sorgt dafür, dass der Compiler die Quelltextdateien übersetzt und der Linker die benötigten Teile aus den Bibliotheken dem Programm hinzufügt. Zusätzlich kontrolliert sie den aktuellen Stand. Ist eine der Dateien neuer als das zuletzt erstellte Programm, gilt dieses als veraltet. Dann

wird ein Programmierer darauf hingewiesen, dass das Programm neu erzeugt werden sollte. Aus diesen Gründen sind beim Programmieren unter Verwendung einer IDE ein paar Formalien einzuhalten.

Praktische Arbeit mit der Entwicklungsumgebung

Beginnen wir nun ein erstes Programm, welches traditionsgemäß einen kurzen Text wie „Hallo!“ ausgibt. Zu diesem Zweck ist ein Projekt anzulegen:

1. Aktivieren Sie den Hauptmenüpunkt ***File***, darunter den Untermenüpunkt ***New*** und darunter ***Project...***, kurz notiert: ***File/New/Project...***!

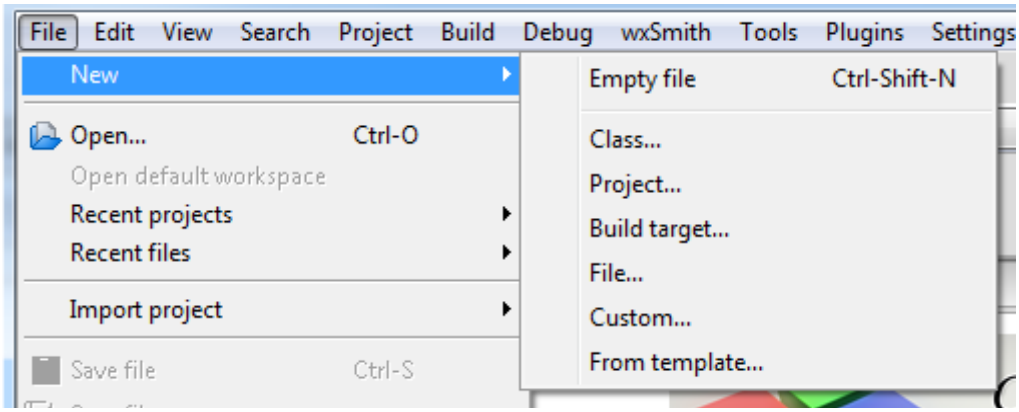


Abbildung 3: Anlegen eines Projektes

2. Daraufhin öffnet sich ein Dialogfenster, welches mehrere Schablonen (engl.: templates) zur Erstellung eines Programms anbietet.

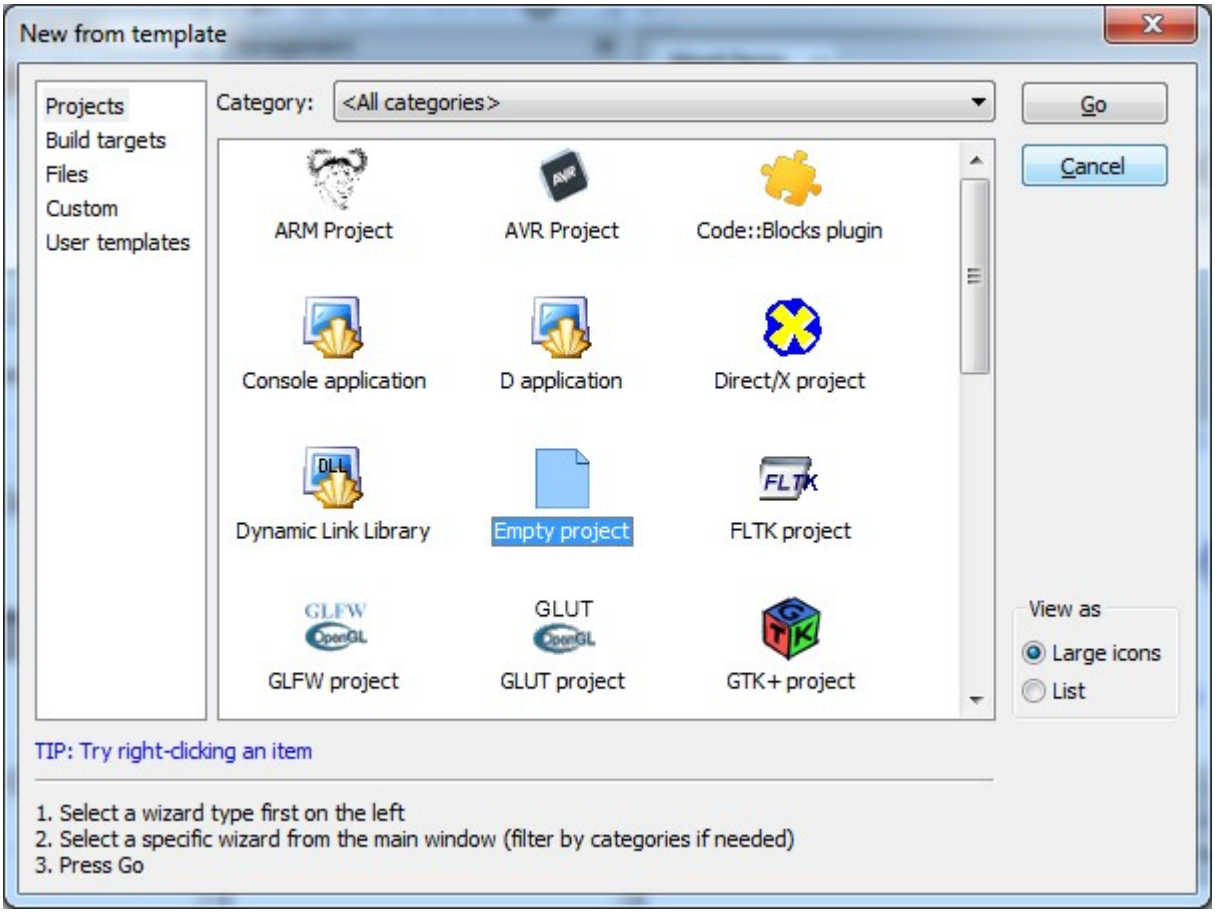


Abbildung 4: Projekttypen für verschiedene Erstellungsziele

3. Für eine Konsolenanwendung bieten sich zwei Projekttypen an:
   Console application (Konsolenanwendung) oder ***Empty project*** (leeres Projekt)

   Ein ***Empty project*** kommt ohne Ballast daher, weshalb Sie diesen Typ wählen sollten.
   Im linken Rahmen muss ***Project*** ausgewählt sein. Im rechten Rahmen wählen Sie ***Empty project*** und schließen dann den Dialog mit **Go**!

4. Dann erscheint ein eher unwichtiger Dialog zum sog. empty project wizard. Wenn Sie dieses Fenster in Zukunft nicht mehr sehen möchten, setzen Sie vor „Skip this page next time“ einen Haken! Schließlich bestätigen Sie Ihr Vorhaben mit **Next>**!

5. Im folgenden Dialogfenster ist unter **Project title** der Name für das Projekt einzutragen, hier **„hallo-01“**. Bei Bedarf können Sie unter **Folder to create project in** einen Projektpfad wählen, was ich nachdrücklich empfehle. Wählen Sie auf Ihrem eigenen Computer einen Pfad wie „D:\programmieren“ (Windows). Im Netzwerk oder auf Linux wählen Sie statt D:\ Ihr Benutzerverzeichnis! So finden Sie Ihre Programmierprojekte im Verzeichnisbaum am leichtesten. Die unteren beiden Eingabefelder werden auf Grund Ihres Projektnamens und Ihres Projektpfades automatisch gefüllt, was Sie getrost so lassen sollten. Schließen Sie den Dialog mit **Next**!

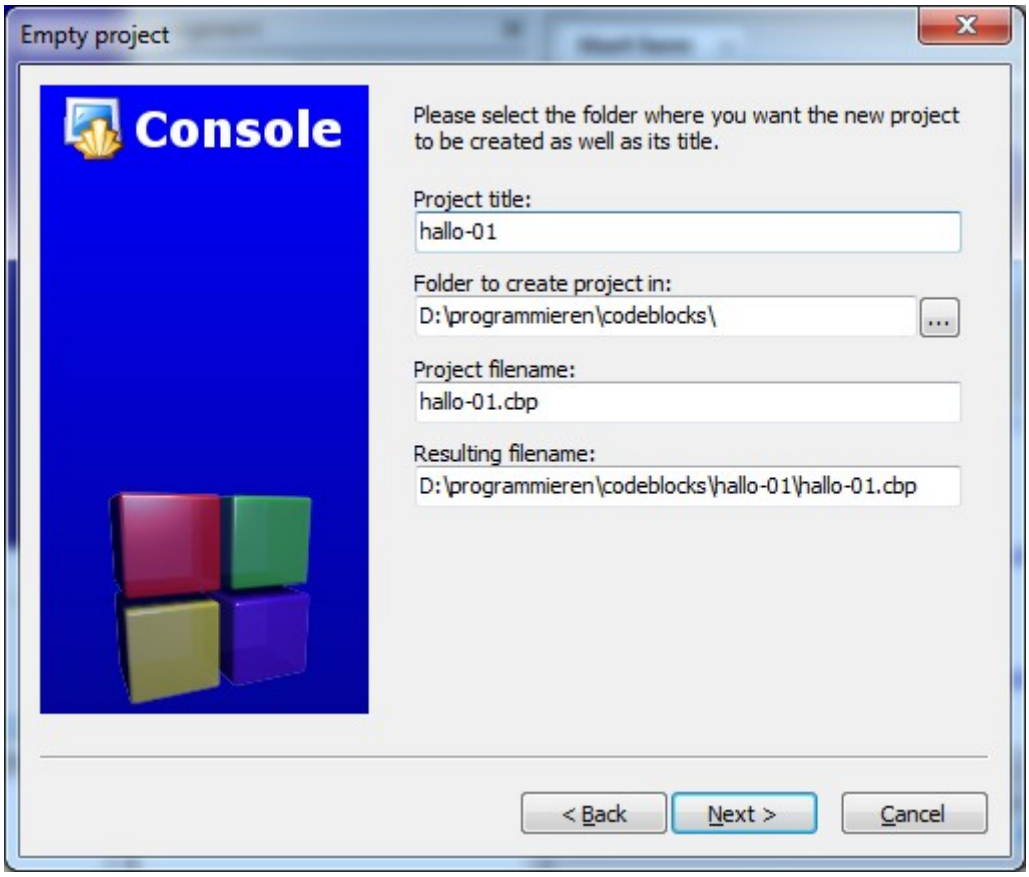


Abbildung 5: Projektname und Pfad

6. Am Ende dieser Prozedur erscheint noch ein letzter Dialog. Darin könnten Sie einen Compiler auswählen sowie die Konfigurationen für eine Debug- und eine Release-Version Ihres Programms abändern. Lassen Sie darin alles so wie es voreingestellt ist!

   Eine Debug-Version wird immer dann gebraucht, wenn man auf Quelltextebene eine Fehlersuche (Source Level Debugging) betreiben will. Außerdem können Sie damit genau beobachten, was in Ihrem Programm beim Ablauf passiert. Eine Release-Version ist kleiner und wird am Ende der Programmierphase ausgeliefert.

   Schließen Sie den Dialog mit **Finish**!

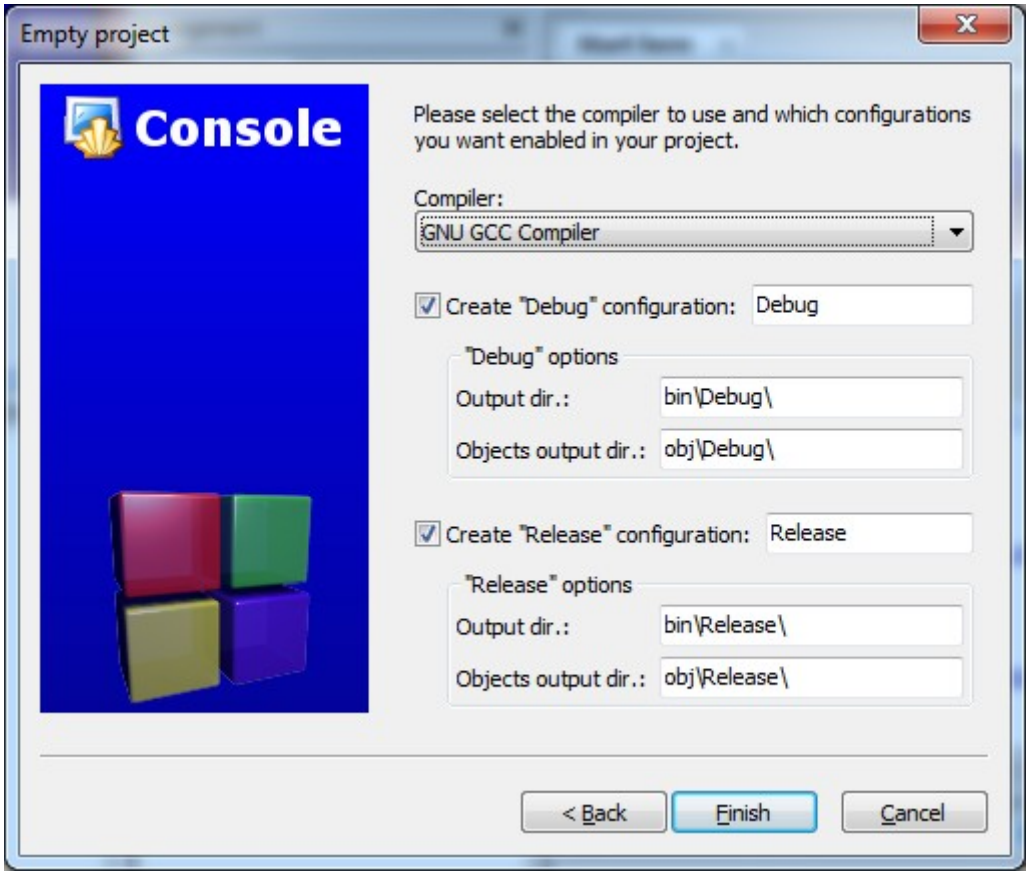


Abbildung 6: Compilerwahl, Debug- und Release-Konfigurationen

7. Nun liegt Ihnen ein (noch) leeres Projekt vor. Es ist in der Projektverwaltung unter Workspace eingetragen.

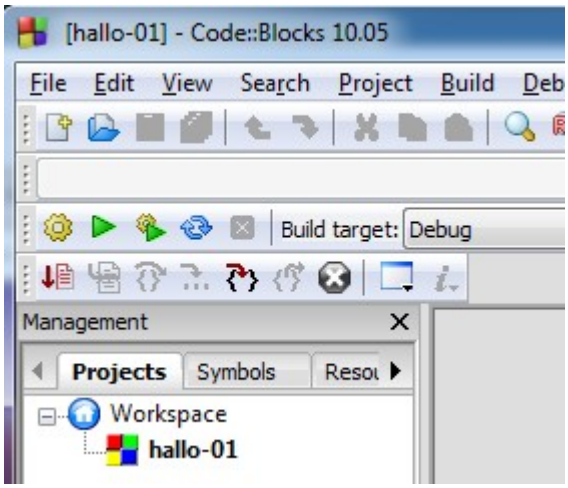


Abbildung 7: Projektverwaltung

8. Nach diesen Vorbereitungsarbeiten brauchen Sie noch eine Datei für den C++-Quelltext. Diese lassen Sie mit ***File/New/Empty file*** oder per Hotkey **Strg+Shift+N** erzeugen.

   Im erscheinenden Dialog werden Sie gefragt, ob Sie diese neue Datei dem aktiven Projekt hinzufügen möchten. Sie besitzen gegenwärtig nur ein Projekt und dies ist zugleich aktiv. Somit können Sie ohne nachzudenken mit **Ja** (oder Yes) bestätigen.

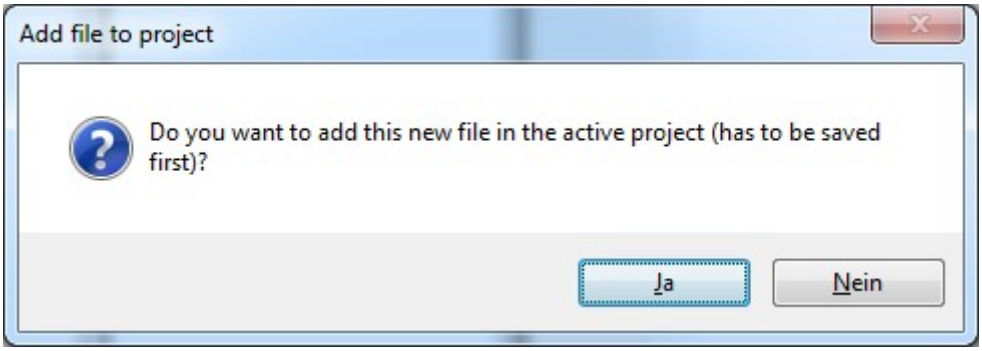


Abbildung 8: Datei dem aktiven Projekt hinzufügen?

9. Danach ist die Datei zu benennen. Ändern Sie den automatisch erzeugten Namen **„Untitled.c“** in einen Namen ab, der zum Projektnamen passt, hier **„hallo-01.cpp“**. Dies ist so lange sinnvoll, wie Ihre Projekte nur eine C++-Quelltextdatei beinhalten, was noch ziemlich lange der Fall sein wird. Schließlich lassen Sie die Datei speichern.

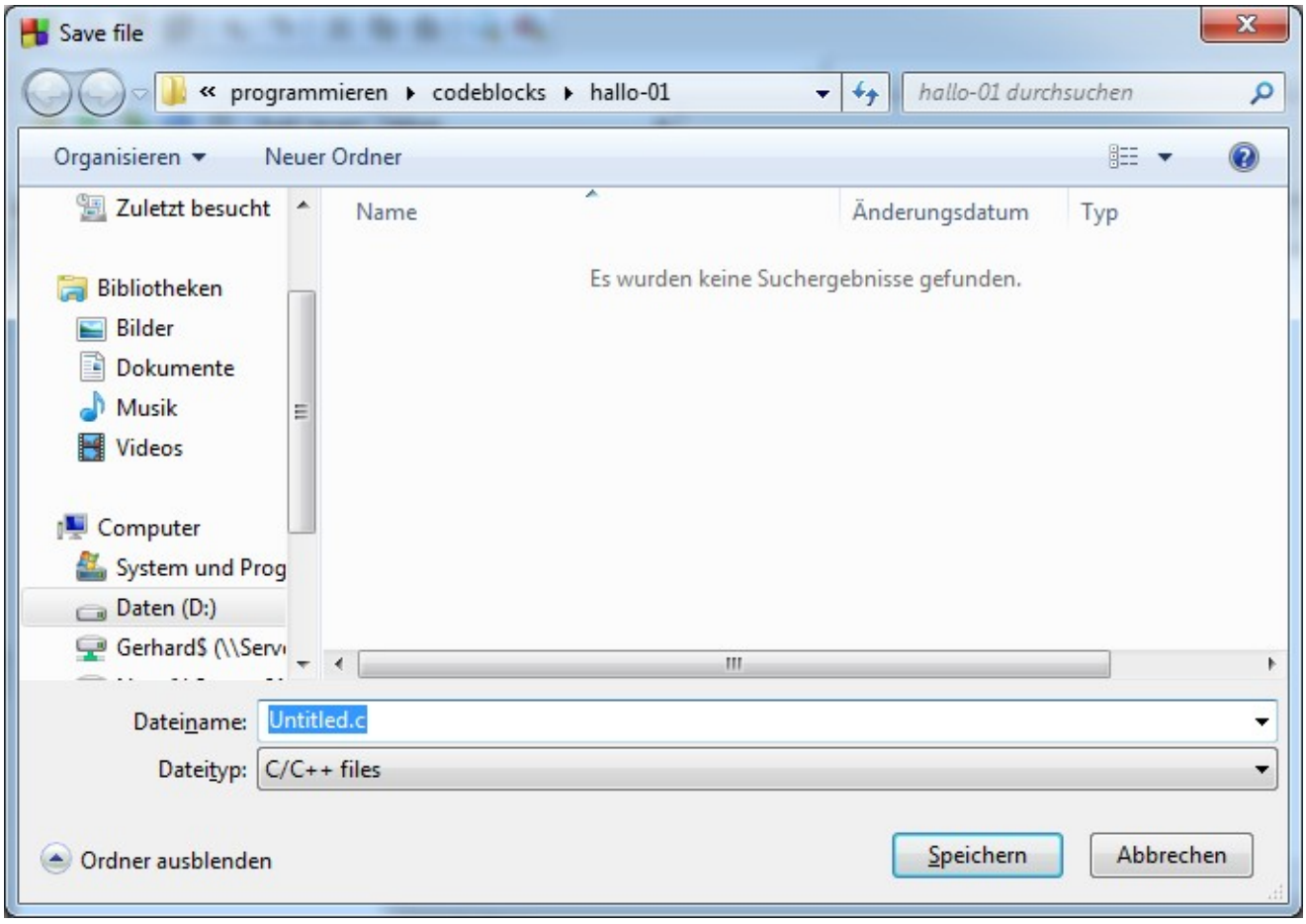


Abbildung 9: Eingabe eines Dateinamens für die C++-Datei

10. Der nächste Dialog gestattet es Ihnen, diese neue C++-Datei den beiden möglichen Versionen **Debug** und **Release** zuzuordnen. Beide sind bereits ausgewählt, was Sie auch so belassen und mit **Ok** beenden sollten.

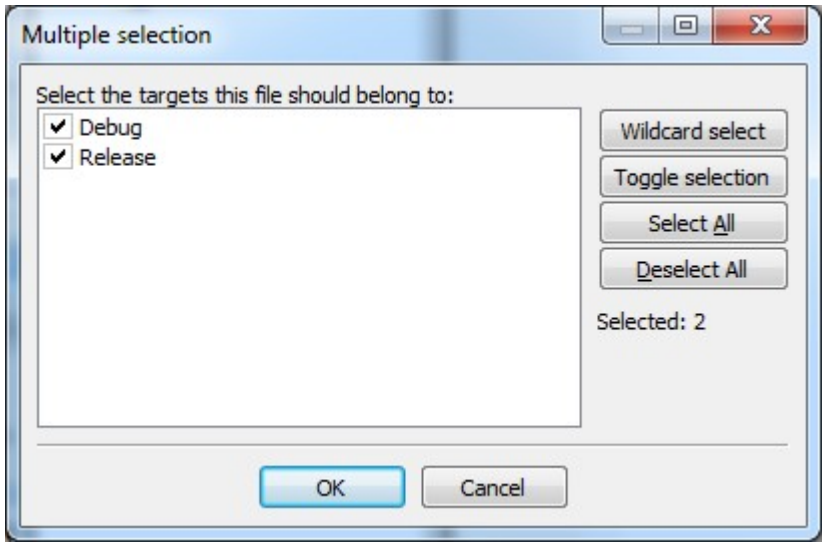


Abbildung 10: Debug- und Release-Version zuordnen

11. Nun liegt alles vor, um endlich mit dem Programmieren beginnen zu können. Fachleute nennen das Folgende allerdings nicht Programmieren sondern Codieren, deren Unterschiede hier noch nicht erörtert werden. Im Editierbereich gibt es ein Fenster mit dem zuvor gewählten Dateinamen, in welches Sie nun den unten folgenden Quelltext eingeben sollten!

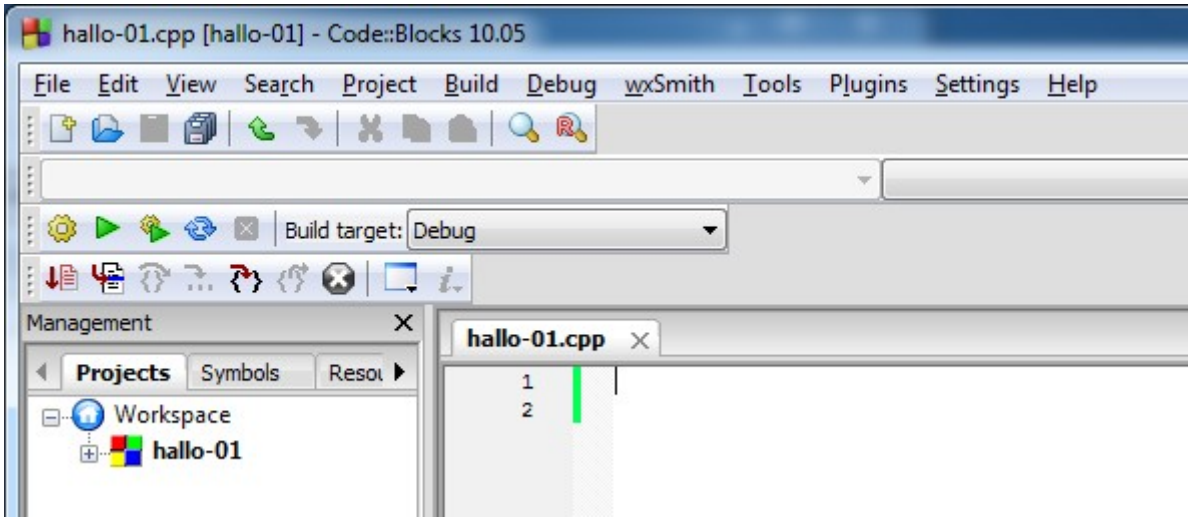


Abbildung 11: Quelltext in einem Editierfenster

Achten Sie genau auf Groß- und Kleinschreibung! Die Kommentare (// …) lassen Sie weg.

Einzugebender Quelltext:

```
#include <iostream>                              // Zeile 1
                                                 // Zeile 2
int main()                                       // Zeile 3
{                                                // Zeile 4
 std::cout << "Hallo, die Erste" << std::endl;   // Zeile 5
 return 0; // Das ist eine Null.                 // Zeile 6
}                                                // Zeile 7
```

12. Anfangserklärungen der obigen Quelltextzeilen

1. **#include** bedeutet, dass die dahinter stehende Datei in die Übersetzung Ihres Quelltextes einbezogen werden soll, hier die Datei **iostream**. Die beiden spitzen Klammern (< >) erklären, dass die Datei in Verzeichnissen gesucht werden soll, die in einer Such-Pfadliste stehen.
2. Eine Leerzeile soll dem Quelltext eine übersichtliche Struktur geben, erforderlich sind solche Leerzeilen allerdings nicht. Setzen Sie Leerzeilen sparsam und gezielt ein! Leerzeilen, die Sie nicht begründen können, sind überflüssig. Hier wird der eigentliche C++-Quelltext von der speziellen #-Anweisung (include) etwas abgesetzt.
3. Ein Programm benötigt einen eindeutigen Anfang, an welchem dessen Ausführung beginnt. Dies ist bei C++ die Hauptfunktion (Name: **main**). Hinter einem Funktionsnamen muss immer ein paar runder Klammern stehen, unabhängig davon, ob in den Klammern etwas steht oder – wie hier – nicht.

   Eine Funktion wird zumeist mit einem Wert beendet. Die main-Funktion liefert diesen Wert an das Betriebssystem bzw. die Shell, von der aus das Programm ausgeführt wurde. Damit wird der Shell ein auswertbarer Fehlercode mitgeteilt – s.a. Zeile 6.

   C++ ist eine weitestgehend typenstrenge Sprache. Jeder Wert muss von einem bestimmten Typ sein. Eine herkömmliche Shell kann nur ganzzahlige Werte verarbeiten. Der Typname für ganzzahlige Werte lautet **int** (von Integer).

   Dieser sog. Resultatstyp steht vor dem Funktionsnamen. Alles zusammen bildet den **Funktionskopf**. Dieser lautet hier: **int main()**
4. Unterhalb des Funktionskopfs folgt der **Funktionsrumpf**. Dessen Anfang und dessen Ende müssen gekennzeichnet werden. Der Anfang wird mit einer öffnenden geschweiften Klammer, das Ende mit einer schließenden gekennzeichnet. In HTML-Dateien dienen zu ähnlichen Zwecken Start- und Endtag.
5. Hier steht der eigentliche Quelltext dieses ersten Programms. C++ ist eine eher kleine Programmiersprache, die über kein Schlüsselwort für eine Ausgabe verfügt. Deshalb muss dafür die Datei iostream (s.a. Zeile 1) einbezogen werden. In dieser Datei wird dem Compiler das Stream-Objekt **cout** vorgestellt, mit Hilfe dessen in Konsolenanwendungen Ausgaben auf einfache Weise durchgeführt werden können – cout von **c**onsole **out**put.

   In C++ werden aus Gründen der Flexibilität Namensräume (namespaces) verwendet. Grundlegend ist der Namensraum **std** (von **st**an**d**ard), in welchem auch das Streamobjekt cout beheimatet ist. Damit der Compiler cout erkennen kann, muss man ihm mitteilen, dass cout ein Mitglied von std ist. Die Verbindung zwischen std und cout wird mit dem Scopeoperator (::) hergestellt. Der vollständige Name lautet somit **std::cout**.

   Dahinter steht der Ausgabeoperator << und hinter diesem der auszugebende Gegenstand. Das ist hier der Text „Hallo, die Erste“. Die beiden Anführungszeichen am Anfang und am Ende des Textes signalisieren dem Compiler, dass hier ein Text (genauer: Zeichenkettenwert) steht. Damit hinter diesem Text eine neue Zeile begonnen wird, folgt ein weiterer Ausgabeoperator und danach **std::endl**. endl bedeutet end of line und bewirkt in der Ausgabe eine Zeilenschaltung, d.h. den Beginn einer neuen Zeile. Auch endl gehört zum Namensraum std.
6. Mit **return** wird die Funktion (main) verlassen und hier der Wert 0 (Null) an die Shell geliefert. 0 bedeutet „Kein Fehler im Ablauf festgestellt.“
7. Mit der schließenden geschweiften Klammer wird der Funktionsrumpf geschlossen – das erste Programm ist hier zu Ende. Dieser Quelltext ist damit vollständig strukturiert und komplett.

Alle C++-Anweisungen werden mit einem Semikolon abgeschlossen. #include ist eine Steueranweisung und keine C++-Anweisung, deshalb fehlt hier das Semikolon.

13. Projekt aus der Projektverwaltung entfernen

Dazu müssen Sie das Projekt schließen. Dies gelingt mit **File/Close project**, worauf das aktive (fett gedruckte) Projekt geschlossen wird.

Alternativ können Sie mit der rechten Maustaste (RMT) auf den Projektnamen klicken und im kontextsensitiven Menü **Close project** wählen (Abb. rechts).

Mit Hilfe dieses Menüs sind noch einige weitere Aktionen möglich, die sich immer auf das per RMT angeklickte Projekt beziehen. Wichtig dabei ist, den **Projektnamen** mit dem Code::Blocks-Symbol davor anzuklicken und nicht etwa **Workspace** oder **Sources**.

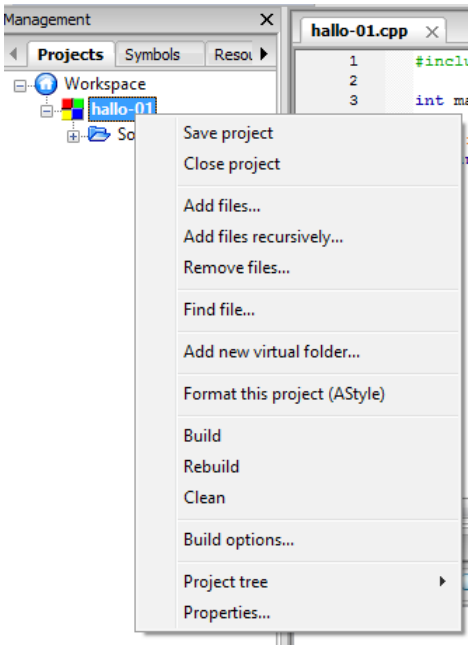


Abbildung 12: kontextsensitives Menü per RMT

14. Vorhandenes Projekt laden/öffnen

Sollten Sie Code::Blocks noch mit dem ersten Projekt (hallo-01) vor sich haben, schließen Sie die IDE nun bitte! Starten Sie Code::Blocks erneut! Ihr zuletzt bearbeitetes Projekt taucht in der Projektverwaltung unter Workspace nicht mehr auf. Es gibt drei Möglichkeiten das Projekt zu laden, um es vielleicht weiter zu bearbeiten.

1. Auf der Startseite von Code::Blocks (Fenster Start here) wird der komplette Pfad der Projektdatei unter **Recent projects** als Verweis aufgeführt. In unserem Fall heißt die Projektdatei hallo-01.cbp. Die Endung cbp kommt vermutlich von Code::Blocks Project. Sie ist eine lesbare XML-Datei – XML = Extensible Markup Language. Um diesen Pfad zu sehen, müssen Sie vielleicht den Scrollbalken des Fensters nach unten ziehen.

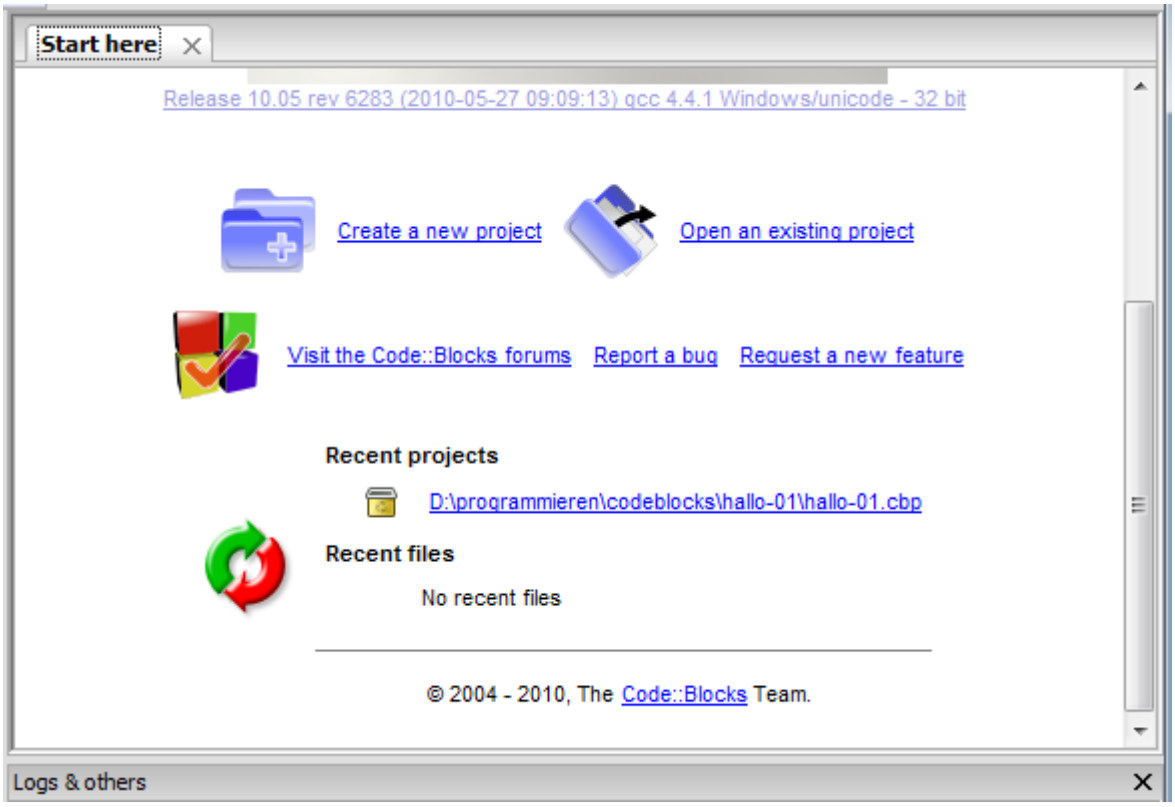


Abbildung 13: Start here Fenster mit Recent projects

Um das Projekt **hallo-01** zu laden, brauchen Sie nur auf diesen Verweis zu klicken.

Damit Sie die anderen beiden Möglichkeiten auch testen können, sollten Sie jeweils vorher das Projekt hallo-01 schließen!

2. Der oben genannte Pfad ist auch unter **File/Recent projects** zu finden. Klicken Sie diesen einfach an und das Projekt wird geladen. Dort können Sie auch per **Clear history** die aufgezeichnete Projektliste löschen. Danach wird Ihnen unter Recent projects keine Projektliste mehr angezeigt, bzw. diese Liste ist leer. Dann bleibt Ihnen noch die folgende Möglichkeit.
3. Hierfür müssen Sie wissen, wo Sie das gesuchte Projekt (die cbp-Datei) gespeichert haben. Deshalb ist es sehr zweckmäßig, sich bereits zu Anfang einen Ordner für alle Code::Blocks-Projekte anzulegen.

   Sie wählen im Menü **File/Open...** oder drücken als Hotkey **Str+O**. Nun öffnet sich ein Dialog zum Öffnen einer Datei. Darin navigieren Sie zu Ihrem Code::Blocks-Verzeichnis, wählen das gewünschte Projektverzeichnis, darin die cbp-Datei und öffnen diese.

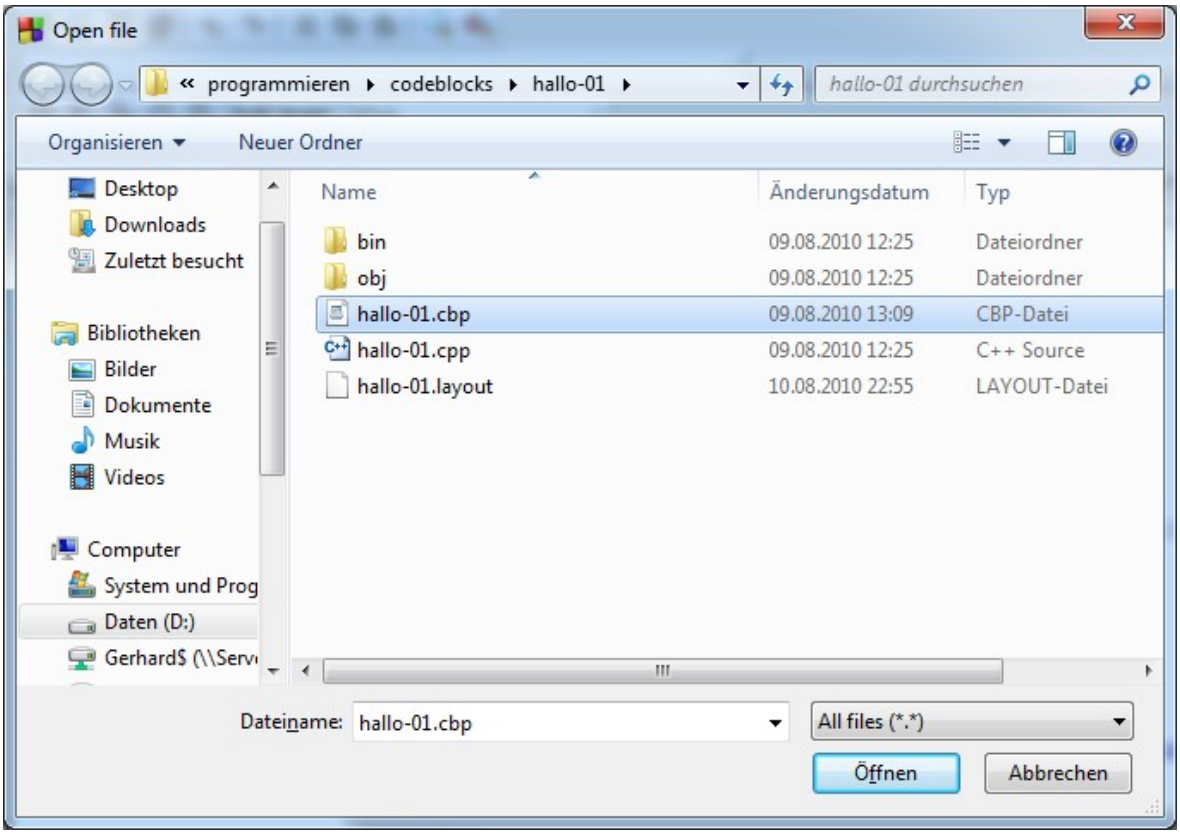


Abbildung 14: Projektdatei öffnen

15. Teil-Verzeichnisbaum zu einem Projekt

Auf Grund der Standard-Projektverwaltung werden die verschiedenen am Projekt beteiligten Dateien in einem Verzeichnis und darin angelegten Unterverzeichnissen abgelegt. Zu einem Projekt gehören in der Regel mehrere/viele Quelltextdateien – allerdings nicht in der Anfangsphase des Programmierens. Es ist guter Stil, alle diese Dateien innerhalb des Projektverzeichnisses abzulegen, bei Bedarf allenfalls in einem Geschwisterverzeichnis dazu. Dann kann man ein komplettes Projekt sehr einfach transferieren, indem man das komplette Verzeichnis kopiert oder verschiebt. Die Projektdatei enthält, so weit möglich, zum Projektverzeichnis relative Pfadangaben. Eine ganz schlechte Idee ist es, eine Datei direkt von

einem USB-Stick oder einer anderen Partition dem Projekt hinzuzufügen.

Sehen wir uns einmal das gut sortierte Projektverzeichnis zu hallo-01 an!

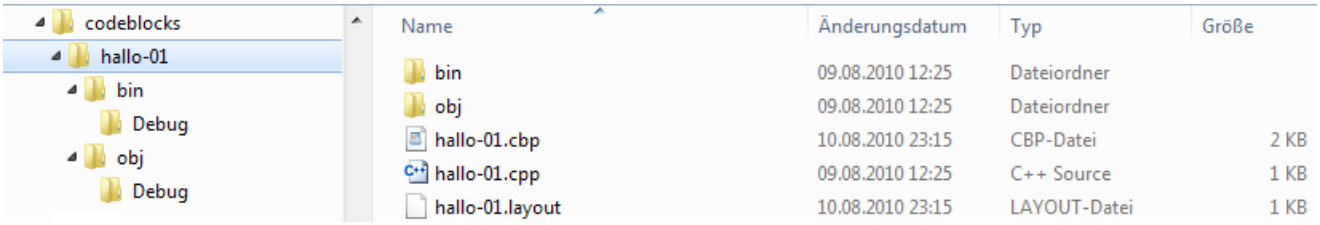


Abbildung 15: Teil-Verzeichnisbaum des Projektes hallo-01

Im linken Teilfenster ist der Teil-Verzeichnisbaum zu sehen, rechts sind die beiden Unterverzeichnisse bin und obj sowie die zum Projekt gehörenden Dateien aufgelistet. Vor der Übersetzung umfasst mein Projektverzeichnis 1,34 kB, danach wird es knapp 1 MB groß sein.

16. Programm übersetzen und laufen lassen

Für das ganz schnelle (Miss-)Erfolgserlebnis gibt es eine kleine Mauspalette.

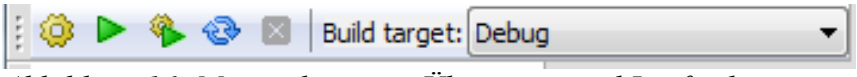


Abbildung 16: Mauspalette zum Übersetzen und Laufen lassen

Symbol	Bedeutung
	Build: Eine ausführbare Datei erzeugen, d.h. Quelltexte übersetzen lassen und Linken. Der Linker bindet Objektdateien und Teile aus Bibliotheken zum Programm zusammen.
	Run: Das erzeugte Programm wird ausgeführt.
	Build and run: Programm erzeugen und laufen lassen
	Rebuild: Das Programm neu erzeugen lassen, wobei alle Quelltextdateien neu übersetzt werden. Das braucht man allenfalls bei einem Versionswechsel.

Hinter **Build target** kann man zwischen Debug- und Release-Version des Programms wählen.

Nachdem Sie das Programm haben erzeugen und laufen lassen, erscheint hier ein Konsolenfenster mit der programmierten Ausgabe.

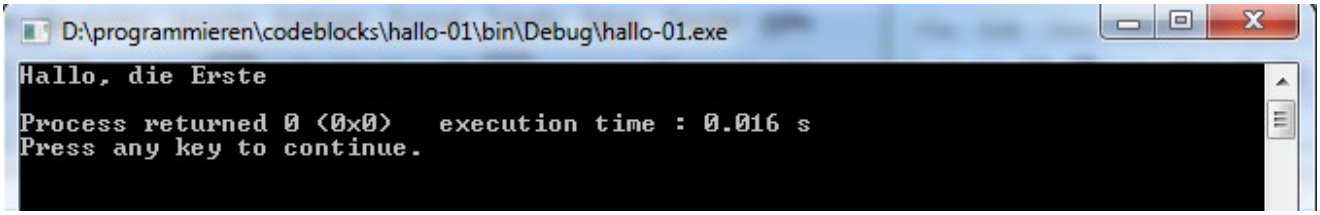


Abbildung 17: Die Ausgabe des Programms hallo-01

„Process returned 0 (0x0) … Press any key to continue“ fügt Code::Blocks hinzu. Diese Ausgabe erscheint nicht, wenn Sie das Programm direkt aus einem Konsolenfenster starten.

Alternativ zur Mauspalette kann man das Build-Menü oder auch das kontextsensitive Menü verwenden.

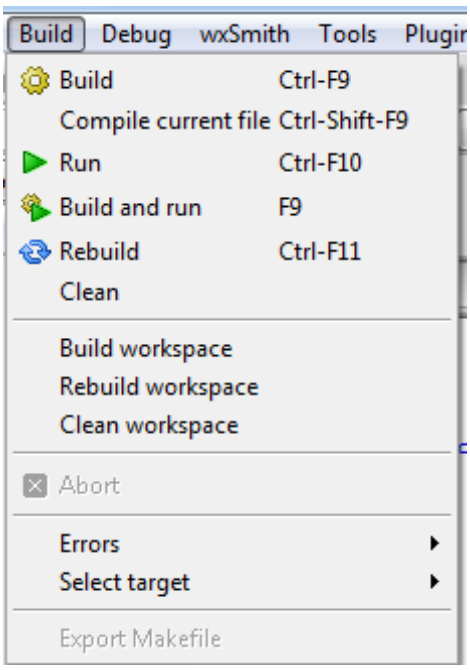


Abbildung 19: Das Build-Menü

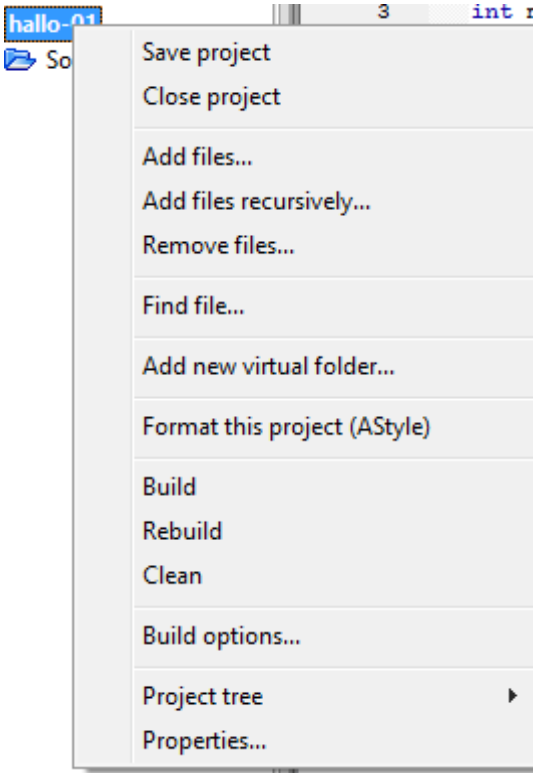


Abbildung 18: Kontextsensitives Menü

Das Build-Menü bietet zusätzlich die Möglichkeit, die aktuell gewählte Datei compilieren zu lassen. Dies kann hilfreich sein, um in dieser Datei nach Syntaxfehlern zu suchen.

17. Die erzeugten Dateien

Wenn Sie nun das Projektverzeichnis durchsuchen, werden Sie zwei neue Dateien finden. Diese wurden per Build oder Rebuild erzeugt und liegen relativ zum Projektordner (hallo-01):

- **obj\Debug\hallo-01.o**
  Das ist die Objektdatei zu Ihrer Quelltextdatei hallo-01.cpp. Diese wurde vom Compiler erzeugt. Zu jeder Quelltextdatei eines Projektes wird eine Objektdatei erzeugt. Objektdateien enthalten zwar ausführbaren Code, sie sind aber noch nicht ausführbar, weil noch Teile fehlen.
- **bin\Debug\hallo-01.exe**
  Diese Datei wird vom Linker erzeugt, indem er alle Objektdateien und Teile aus Bibliotheken zu einem ablauffähigen Programm zusammenbindet. Sollten Sie ein Linux verwenden, dürfte hier die Endung .exe im Dateinamen fehlen. Dieser Suffix ist Windows-typisch.

Wenn Sie nun die Größe des Projektverzeichnisses ermitteln, werden Sie feststellen, dass es sich auf etwa 943 kB vergrößert hat - und das bei diesem sehr kleinen Programm. Im Vergleich zu zuvor 1,34 kB ist das ziemlich viel. Wenn Sie ein solches Projekt per Email oder per USB-Stick übertragen wollen und auf dem Zielsystem steht ebenfalls Code::Blocks zur Verfügung, brauchen Sie weder die Objekt- noch die ausführbare Programmdatei. Sie können also diese Dateien löschen und das Projektverzeichnis archivieren. Mein 7z-Archiv umfasst 870 Byte und enthält alles, was zum erneuten Erzeugen des Programms per Code::Blocks gebraucht wird.

18. Das Löschen der nicht unbedingt benötigten Dateien können Sie auch Code::Blocks überlassen.

    Das erreichen Sie mit Build/**Clean** oder per RMT auf den Projektnamen klicken und im kontextsensitiven Menü **Clean** wählen. Unter Build können Sie sogar per **Clean workspace** alle im Arbeitsbereich (workspace) enthaltenen Projekte bereinigen. Bisher haben wir erst ein Projekt darin, was sich aber noch ändern soll.

19. Ein weiteres Projekt dem Arbeitsbereich hinzufügen

Gehen Sie so vor wie unter 1. bis 6. beschrieben! File/Open...

Wählen Sie hier bitte den Projektnamen **hallo-02**! Danach befinden sich in der Projektverwaltung unter Workspace zwei Projekte: hallo-01 und **hallo-02**

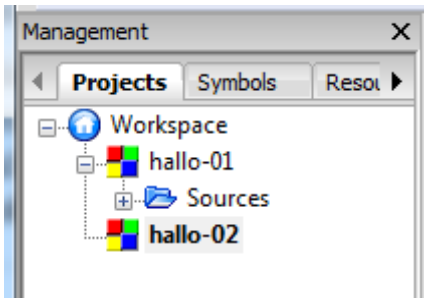


Abbildung 20: Zwei Projekte

Nun ist statt hallo-01 das Projekt hallo-02 fettgedruckt. Das fettgedruckte Projekt ist aktiv. Wenn Sie ein Programm erzeugen lassen (Build), wird immer das aktive Projekt übersetzt und gelinkt. Allerdings braucht es dazu noch eine C++ Quelltextdatei. Erzeugen Sie zu diesem Zweck die leere C++-Datei **hallo-02.cpp** (**File/New/Empty file** oder **Strg+Shift+N**)!

Das nun zu schreibende Programm soll den Text **„Hallo, die Zweite“** ausgeben. Dazu empfehle ich Ihnen, den Quelltext nicht vom ersten Projekt zu kopieren, sondern zwecks Übung neu einzugeben, allerdings mit kleinen Änderungen:

- Setzen Sie unter die #include-Anweisung eine Leerzeile und darunter den folgenden Text: **using namespace std;**
  Dies bewirkt, dass Sie im weiteren Quelltext vor cout und vor endl nicht mehr std:: schreiben müssen, weil der Compiler automatisch im Namensraum std nach diesen Elementen sucht – und findet.
- Notieren Sie also zur Ausgabe nur noch cout << “Hallo, die Zeite“ << endl;
- Setzen Sie hinter return einmal einen anderen Wert als 0, beispielsweise 12!

20. Schauen wir uns einmal das Syntax-Highlighting des fertigen Quelltextes an!

```
#include <iostream>

using namespace std;

int main()
{
 cout << "Hallo, die Zweite" << endl;
 return 12;
}
```

Abbildung 21: Syntax-Highlighting

- **#include** … wird grün dargestellt. Dies gilt für alle Anweisungen, die mit einem # beginnen. Solche Anweisungen werden vor der eigentlichen Übersetzung ausgeführt. Eine #include-Anweisung bewirkt, dass die aufgeführte Datei komplett an dieser Stelle eingefügt wird. Zu unserem kleinen Quelltext wird eine temporäre Datei erzeugt, die aus dem Inhalt der Datei iostream und unserem darunter folgenden Quelltext zusammengesetzt ist. Diese temporäre Datei wird vom Compiler übersetzt. Die #-Anweisungen sind eigentlich keine C++-Anweisungen, weshalb sie am Ende kein Semikolon besitzen.

- **using namespace** wird dunkelblau und fettgedruckt dargestellt. Daran erkennen Sie Wortsymbole von C++. Das sind Wörter, die in C++ eine festgelegte Bedeutung haben. **int** und **return** sind auch solche Wortsymbole. Wenn Sie sich einmal vertippen und beispielsweise namesace schreiben, wird dieses Wort nicht dunkelblau und fett dargestellt sondern einfach in schwarz. Daran ist schnell zu erkennen, dass etwas nicht stimmt.
- Namen wie cout und endl werden einfach schwarz dargestellt. Sie haben in C++ keine feste Bedeutung, d.h. der Compiler kann sie nur verarbeiten, wenn sie ihm vorher erklärt wurden. Das geschieht in der Datei iostream. main ist insofern eine Ausnahme, weil dieser Name für die Hauptfunktion reserviert ist.
- Operatoren wie ( ), <<, {} werden rot dargestellt, ebenso das Semikolon zum Abschließen einer C++-Anweisung.
- Zahlenwerte werden violett dargestellt, Zeichenkettenwerte (“Hallo, die Zweite“) in einem helleren Blau.

21. Der Fehlercode eines C++-Programms

Wenn Sie dieses Programm ablaufen lassen, wird nach unserer programmierten Ausgabe der Text Process returned 12 … ausgegeben, was normalerweise einen Fehlercode signalisiert. Wenn stattdessen eine 0 geliefert wird, bedeutet dies normalerweise „Kein Fehler festgestellt“. Dies ist der Grund dafür, dass wir die main-Funktion mit return 0; beenden.

22. Ein anderes Programm ablaufen lassen

Wenn Sie im Arbeitsbereich mehrere Projekte verwalten, kann es vorkommen, dass Sie ein anderes Programm ablaufen lassen möchten – also nicht das zuletzt erstellte. Dazu ist das aktive Projekt zu wechseln, d.h. dasjenige Projekt ist zu aktivieren, dessen Programm ablaufen soll.

Beispiel:

hallo-02 ist gerade das aktive Projekt, Sie möchten aber **hallo-01.exe** aus der IDE starten. Klicken Sie dazu mit der RMT auf den Projektnamen hallo-01 und wählen Sie im kontextsensitiven Menü den Punkt ***Activate project*** ! Dieser Menüpunkt steht übrigens nur bei nicht aktiven Projekten zur Verfügung. Wenn Sie nun auf den grünen Pfeil (Run) klicken, wird hallo-01.exe gestartet.

23. Einen Arbeitsbereich benennen und speichern

Wenn Sie über längere Zeit mehrere Projekte in einem Arbeitsbereich verwalten wollen, ist es hilfreich, diesem Arbeitsbereich einen vielsagenden Namen zu geben und ihn abzuspeichern. Dahinter steckt, wie bei den Projektdateien, eine XML-Datei.

Klicken Sie mit der RMT auf den Arbeitsbereich (Workspace) und wählen Sie im kontextsensitiven Menü ***Rename workspace...***! Im sich öffnenden Dialog können Sie den gewünschten Namen eingeben.

Benennen Sie den Arbeitsbereich in „Anfang“ um! (ohne Anführungszeichen)

Nun sollten Sie den Arbeitsbereich speichern. Öffnen Sie wieder das kontextsensitive Menü und wählen Sie ***Save workspace as...***! Speichern Sie den Arbeitsbereich an geeigneter Stelle, beispielsweise unter **D:\programmieren\Anfang** – und nicht wie vorgegeben in Ihrem Profil! Dann wird im Verzeichnis D:\programmieren die Datei **Anfang.workspace** abgelegt und in die History-Liste unter Recent projects (Startseite „Start here“) eingetragen.

In einer späteren Sitzung können Sie einfach diesen Arbeitsbereich öffnen, wodurch Sie zugleich alle darin enthaltenen Projekte öffnen. Sie können aber auch die Projekte einzeln verwenden. Ein Arbeitsbereich ist einfach ein Behälter für Projekte, der Ihnen die Projektverwaltung erleichtern kann. Ist die Recent projects-Liste leer, können Sie den Arbeitsbereich per File/Open bzw. (Strg+O) suchen und öffnen.

24. Einrückungen einstellen

Einrückungen sind immer dann für uns Menschen hilfreich, wenn ein Text strukturiert ist. In einem strukturierten Text erkennen wir an Hand der Einrückungen schneller und sicherer, welche Strukturen vorliegen und wo evtl. eine Struktur einen Fehler aufweist. Strukturierte Texte gibt es beispielsweise in XML-, (X)HTML-, CSS- und Programmquelltextdateien.

Gewöhnen Sie sich frühzeitig eine sorgfältige Einrücktechnik an! Sie erleichtert erheblich die Fehlersuche. Für Einrückungen sollten Sie statt Leerzeichen Tabstopps verwenden, weil diese in ihrer Wirkung konfigurierbar sind. Die Standardeinstellung für Tabstopps ist leider nicht sehr gut. Ich halte vier Zeichen Abstand pro Tabstopp für übertrieben und Leerzeichen sollten nicht eingesetzt werden. Das sollten Sie nun ändern!

Wählen Sie im Hauptmenü ***Settings/Editor...***! Es öffnet sich (mal wieder ,-) ein Dialogfenster.

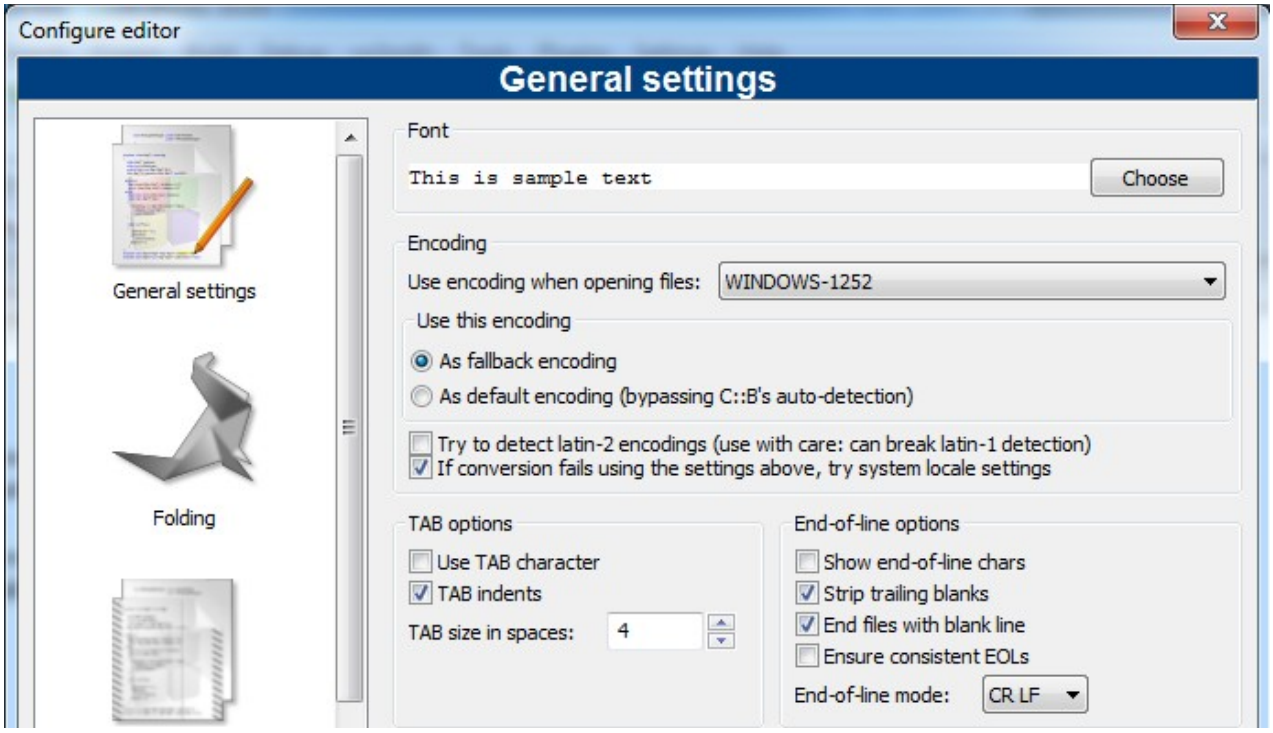
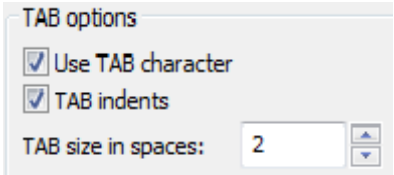


Abbildung 22: Einstellungen für den Editor



Sie finden etwa in der Mitte den kleinen Abschnitt **Tab options**. Darunter sollten Sie **Use TAB charachter** aktivieren und **TAB size in spaces** auf 2 reduzieren – siehe Abbildung rechts.

Bei Bedarf können Sie in diesem Dialog noch viele Einstellungen Ihren Bedürfnissen anpassen. Ich empfehle Ihnen aber, dies nicht zu übertreiben, weil ich Ihnen bei Programmierproblemen sonst vielleicht schlechter weiterhelfen kann.

Nun sollten Sie mit Code::Blocks arbeiten können. Vermutlich werden Sie im weiteren Verlauf hin und wieder vor einem Bedienungsproblem stehen. Dann lesen Sie noch einmal in dieser Anleitung und/oder fragen jemanden Ihres Vertrauens.

Viel Erfolg!

Sonderthemen

S1: Eine Bibliothek verwenden

Es gibt viele C++-Bibliotheken, die man in eigenen Projekten verwenden kann. Bibliotheken können dynamisch oder statisch sein. Für mehr Informationen bei Wikipedia einfach nach „Programmbibliothek“ suchen.

Dynamische Bibliothek

Eine dynamische Bibliothek wird auch „Dynamic Link Library“ (.dll, Windows) oder „Shared Object“ (.so, Linux) genannt. Eine solche Bibliothek wird nur **zur Laufzeit** eines Programms bei Bedarf in den Arbeitsspeicher geladen (daher „dll“) – und sie steht allen Programmen zur Verfügung, welche diese Bibliothek benutzen (daher „so“). Weiterhin kann eine neuere Version der Bibliothek verwendet werden, ohne das Programm neu übersetzen lassen zu müssen. Wenn man allerdings ein solches Programm ohne die dynamische Bibliothek weitergibt, ist es auf dem anderen System ohne diese Bibliothek nicht lauffähig.

Statische Bibliothek

Eine statische Bibliothek stellt Dinge wie Funktionen, Konstanten, Klassen, Objekte zur Verfügung, die **zur Übersetzungszeit** vom Linker diesem Programm hinzugefügt werden. Das fertige Programm ist also „all inclusive“. Es ist entsprechend größer und kann bei Weitergabe auf ein geeignetes Zielsystem dort ausgeführt werden.

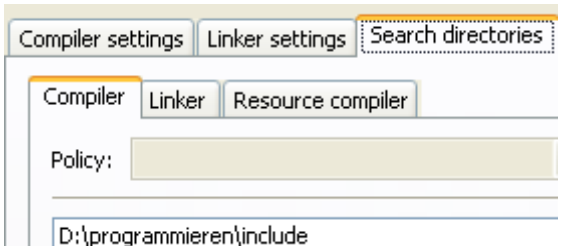
Eine Statische Bibliothek in einem Code::Blocks-Projekt einbinden

Zu diesem Zweck brauchen Sie die fertige Bibliotheksdatei, beispielsweise „libeichelsdoerfer.a“, und zumindest eine Headerdatei, beispielsweise „eichelsdoerfer.h“, die Sie per #include in Ihren Quelltext einbeziehen können. Ich empfehle folgende Vorgehensweise, welche die weitere Arbeit erleichtert:

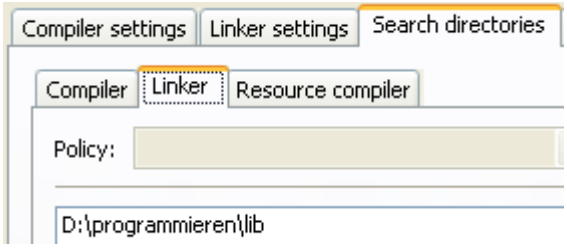
1. Erstellen Sie ein Verzeichnis „include“, beispielsweise D:\programmieren\include. In dieses Verzeichnis kopieren oder verschieben Sie die Headerdatei (eichelsdoerfer.h)!
2. Erstellen Sie ein Verzeichnis „lib“, beispielsweise D:\programmieren\lib. Dort hinein kopieren oder verschieben Sie die Bibliotheksdatei (libeichelsdoerfer.a)!
3. Nun müssen Sie in den Code::Blocks-Einstellungen der IDE noch mitteilen, wo nach Headerdateien und Bibliotheksdateien zusätzlich gesucht werden soll (Suchpfade).

   Wählen Sie dazu im Hauptmenü **Settings/Compiler and debugger …**! Im sich öffnenden Dialogfenster wählen Sie unter „Global compiler settings“ die Karte „Search directories“!

   In der Unterkarte **Compiler** fügen Sie per **Add** den Pfad zum obigen Include-Verzeichnis hinzu. Danach wählen Sie die Unterkarte **Linker** und fügen dort den Pfad zum obigen Bibliotheksverzeichnis hinzu (siehe Abbildungen rechts)! Der Linker-Pfadeintrag mag nicht nötig sein, schadet aber auch nicht.



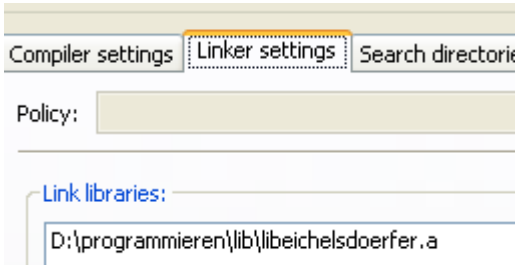
Include-Pfad für den Compiler



Bibliothekspfad für den Linker

Auf diese Weise können Compiler und Linker die Headerdateien (hinter #include) und die Bibliotheksdateien finden.

4. Nun braucht der Linker noch Informationen über die zusätzlich zu berücksichtigenden Bibliotheken. Im selben Dialog wählen Sie die Karte „Linker settings“ und fügen dort per **Add** die zu benutzende Bibliothek hinzu!



Bibliothek für den Linker eintragen

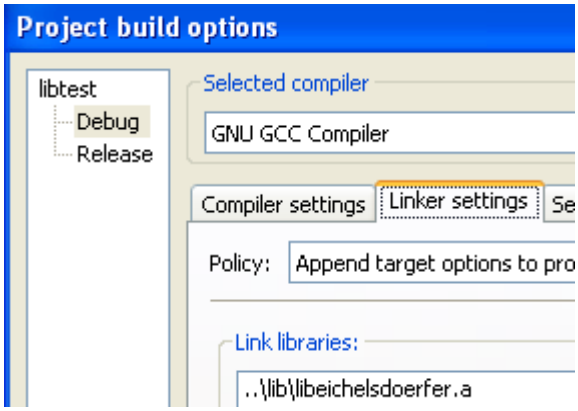
Projektkonfiguration als Alternative zur IDE-Konfiguration

Die oben beschriebenen Einträge sind in der Entwicklungsumgebung konfiguriert. Das hat zur Folge, dass alle Projekte davon profitieren. Sie gelten allerdings ausschließlich auf dem System, in welchem Code::Blocks auf diese Weise konfiguriert wurde. Bei einem Transfer auf ein anderes System mit anderer Code::Blocks-Konfiguration kann ein solches Projekt nicht einfach so übersetzt werden, das Projekt ist nicht „all inclusive“.

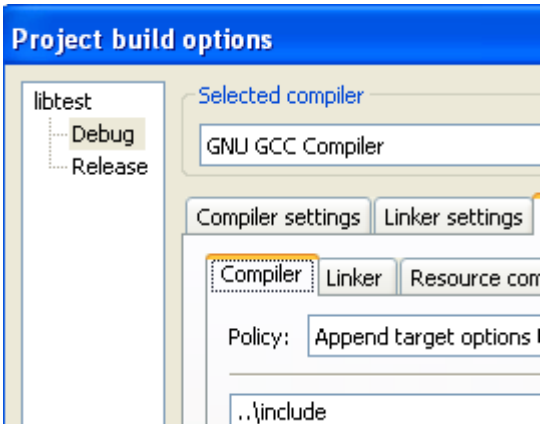
Diese Vorgehensweise ist also etwas für bequeme Menschen. Professioneller ist es, jedes Projekt einzeln zu konfigurieren. Die Informationen werden dann in der Projektdatei (XML) abgelegt und werden beim Transfer weitergereicht. Außerdem können die Pfade relativ (..\include) eingetragen werden, weshalb Bibliotheken und Projekte in einem Archiv komplett und übersetzbar übertragen werden können.

Dazu gehen Sie wie folgt vor:

Mit RMT auf den Projektnamen klicken oder für das aktuelle Projekt im Hauptmenü Project anklicken. Danach „Build options ...“ wählen. Im sich öffnenden Dialogfenster finden Sie die beiden Karten „Linker settings“ und „Search directories“, wie in der IDE-Konfiguration oben. Tragen Sie die Pfade und die benötigten Bibliotheksdateien entsprechend ein!



Bibliothek im Projekt eintragen



Include-Suchpfad im Projekt eintragen

Wenn Sie bei allen Projekten so vorgehen, brauchen Sie die oben beschriebene IDE-Konfiguration nicht durchzuführen. Anfänger besitzen aber oft nicht den erforderlichen Überblick und vergessen schon mal die Projektkonfiguration. Dann ist die IDE-Konfiguration weniger frustrierend ;-)

In meinem Austauschordner steht ein Archiv „eichlib.zip“ zum Herunterladen zur Verfügung. Darin liegen ein geeignet konfiguriertes Projekt „libtest“, das include-Verzeichnis mit der Headerdatei „eichelsdoerfer.h“ und das lib-Verzeichnis mit der statischen Bibliothek „libeichelsdoerfer.a“. Die Bibliothek ist allerdings sehr klein. Sie stellt ausschließlich die Funktion „runden()“ zum Runden einer double-Zahl zur Verfügung. Aber vielleicht wächst sie noch …

Wenn Sie nun noch wissen möchten, wie man eine dynamische Bibliothek einbinden kann, fragen Sie mich danach ;-)