Die Entwicklungsumgebung Eclipse CDT
Eine Einführung in die Bedienung von Eclipse zur Entwicklung von C++ Anwendungen

Gerd Hirsch ©2014 - 2017
Revision: 1273

18. Januar 2018

Inhaltsverzeichnis

Teil I

Einführung

1 Script

1.1 Ziele und Inhalte

Dieses Script soll in die Konzepte von Eclipse im Allgemeinen und in die Erstellung und das Debugging von C++ Programmen mit Eclipse im Speziellen einführen. Es ist als Workshop begleitendes Script und nicht zum Selbststudium konzipiert. Es werden nur die wesentlichen Dinge erklärt, die Erforschung der vielen unerwähnten Details wird dem geneigten Leser überlassen und ergeben sich im Laufe der praktischen Arbeit mit Eclipse. Soweit es möglich ist, werden entsprechende Literaturhinweise und Links in den Fußnoten angegeben. Beim Lesen sollte eine geöffnete Eclipse Installation mit den Beispielprojekten zur Verfügung stehen, um die beschriebenen Sichten und Arbeitsschritte sofort nachvollziehen zu können.

Im ersten Teil wird kurz auf die geschichtlichen Hintergründe, die Installation und die Dokumentation eingegangen, im zweiten Teil werden die Grundlagen und Bedienungskonzepte erklärt. Der dritte Teil beschreibt die Erzeugung und das Debuggen von C/C++ Projekten und die Themen rund um die embedded Entwicklung mit Eclipse.

Im Titel wird die Revision und das Datum der letzten Änderung angegeben. Die Revisionsnummer wird nur bei größeren Änderungen, z.B. ein neues Kapitel, geändert, sie bleibt bei kleineren Änderungen und Korrekturen gleich.

Einige Kapitel sind noch nicht erstellt oder noch nicht vollständig. Eine Sammlung der bisher geplanten Kapitel sind in section 13 auf Seite 92 beschrieben. In diesem Kapitel werden auch die letzten Änderungen angegeben.

Das Script lebt vom Feedback der Teilnehmer! Wenn Sie Fehler irgendwelcher Art entdecken oder Vorschläge zur Veränderung, Erweiterung des Scripts haben, senden Sie mir bitte eine E-Mail mit einem Hinweis.

1.2 Konventionen

Folgende Textformate werden in diesem Script verwendet:

- Shortcuts **Ctrl+z**
- kombinierte Shortcuts **Alt+Shift+Q, c**
- Auswahl eines Elements mit der rechten Maus **klick** bzw. **doppel klick**
- Menüauswahl *Window.Preferences*

- Kontext Menü *<Resourcename>.open* der Resourcename wird an manchen Stellen weggelassen
- wichtige Begriffe **Equinox**
- Sourcecode `#include <header.h> int i = 0;`

Das Plus (**+**) bei Shortcuts bedeutet, dass diese Tastenkombination gemeinsam gedrückt werden muss, das Komma (**,**) bedeutet, dass die vorherigen Tasten wieder freigegeben werden müssen.

Als *Cursor* wird normalerweise der Mauszeiger, als *Caret* der senkrechte Strich der Eingabeposition in einem Editor bezeichnet. Diese Begriffe werden häufig auch synonym verwendet, wenn aus dem Kontext hervorgeht, was gemeint ist.

Das Kontextmenü wird mit der rechten Maustaste auf einer Resource[1] geöffnet.

Drag´n Drop bezeichnet eine Interaktion mit der Maus. Als Drag wird das Auswählen eines graphischen Elements mit dem Mauszeiger, mit gedrückter linker Maustaste, bezeichnet.
Als Drop wird das loslassen der linken Maustaste über einem Zielfenster, das das Element verarbeiten kann, bezeichnet.
Als Drag and Drop wird das Ziehen des Elements von der Quelle zum Ziel bezeichnet. Der Mauszeiger verändert sich dabei entsprechend, um anzuzeigen, welche Aktion jeweils möglich ist.

Bei der ersten Verwendung eines neuen Begriffs ist meistens der englische Begriff mit einem Schrägstrich angefügt: Eigenschaft/property

1.3 Links and downloads

Dieses Script ist unter
http://www.gerdhirsch.de/downloads/Eclipse/EclipseCDTSchulung.pdf verfügbar.
Die Beispiele sind unter
http://www.gerdhirsch.de/downloads/Eclipse/EclipseWorkshop.zip verfügbar. Sie werden nach und nach aktualisiert.

Die aktuelle Version der Beispiele ist auf github verfügbar: https://github.com/GerdHirsch/EclipseWorkshop und https://github.com/GerdHirsch/VS_Application sowie VS_Sensor und VS_Kreuzung.

Wenn Sie Fehler entdecken oder Verbesserungs- oder Erweiterungsvorschläge haben, senden Sie diese bitte an meine e-mail Adresse.

[1] section 5.2 auf Seite 13

2 Equinox, ein Erweiterbares Framework

Der Ursprung von Eclipse liegt in der IDE VisualAge von IBM[2]. Eclipse basiert auf dem allgemeinen erweiterbaren Framework **Equinox**[3]. Equinox ist ein OSGi[4] konformes Framework. Auf dieser Basis sind Eclipse und weitere Features[5] für verschiedene Sprachen entwickelt:

- Eclipse (ist selbst ein Feature)
- Java
- C/C++ Development Toolkit (CDT)
  - Verschiedene Compiler (Tool Chain)
  - Cross Compiler
  - Remote Compiling / Debugging
- LaTeX
- weitere

Eclipse und die anderen Features werden durch eine wachsende Community kontinuierlich weiterentwickelt. Es lohnt sich, diese Entwicklung zu beobachten, um von den Erweiterungen und Verbesserungen zu profitieren. Für die Entwicklung von C/C++ Programmen ist besonders das Feature *Refactoring* hervorzuheben, da es hier noch große Potentiale auszuschöpfen gibt;-)

3 Installation

3.1 Voraussetzungen

Für Eclipse wird eine Java Runtime Umgebung benötigt. Diese kann unter `www.java.com/de/download/` bezogen werden. Eine genaue Installationsbeschreibung für die verschiedenen Betriebssysteme ist auf derselben Seite zu finden.

Für die CDT wird ein Compiler benötigt. Auf Unix Systemen ist dieser meistens schon vorhanden. Für Windows Systeme kann die GNU Toolchain über ein Posix konformes Subsystem wie
Cygwin: `http://www.cygwin.com/install.html` oder
MinGW: `http://sourceforge.net/projects/mingw/files/` installiert werden.

[2] [Bau10]
[3] Tag und Nacht Gleiche, `http://de.wikipedia.org/wiki/Equinox_(OSGi-Framework)`
[4] www.de.wikipedia.org/wiki/OSGi
[5] section 5.1 auf Seite 11

3.2 Eclipse

Eine aktuelle Version von Eclipse und die Vorgänger können unter
https://www.eclipse.org/downloads/ bezogen werden. Eine genaue Installationsbeschreibung für die verschiedenen Betriebssysteme ist auf derselben Seite zu finden.
Die aktuelle Version ist Eclipse Mars/CDT 8.7.

3.3 CDT C/C++ Development Toolkit

Das CDT ist ein Feature das in Eclipse installiert werden kann, wie es in section 5.1 auf Seite 11 beschrieben ist. Außerdem werden für CDT vorkonfigurierte Eclipse packaged zip Archive unter
http://www.eclipse.org/cdt/downloads.php als download angeboten.

Von der Website:
The CDT can either be installed as part of the Eclipse C/C++ IDE packaged zip file or installed into an existing Eclipse using the “Install New Software...” dialog and entering the p2 repository URLs.

3.4 RTC Rational Team Concert

Für das RTC kann ein Feature in Eclipse installiert werden kann, wie es in section 5.1 auf Seite 11 beschrieben ist. Das Feature kann über die Website
https://jazz.net/downloads/rational-team-concert/ bezogen werden.

Von der Website:
IBM Rational Team Concert integrates task tracking, source control and agile planning with continuous builds and a configurable process to adapt to the way you work.

3.5 Administration und Mirrors

Die Verwaltung der Features innerhalb von Organisationen kann über einen lokalen Mirror erfolgen. Die Archive der Plugins/Features werden dazu auf einem organisationsspezifischen Server gelegt und alle Verweise auf die Plugins der Eclipse Installationen beziehen sich auf diesen Server. Dadurch kann gewährleistet werden, dass alle mit denselben Versionen der Plugins arbeiten.

Auf diesem Server können auch die vordefinierten Shortcuts, Perspectives und weitere Einstellungen, die auf allen Arbeitsplätzen einheitlich zur Verfügung stehen sollten, abgelegt werden. Details und weitere Informationen zur Administration von Eclipse sind unter
http://wiki.eclipse.org/ und in den nachfolgenden Abschnitten zu finden.

4 Dokumentation zu Eclipse/CDT

Für Eclipse Luna existiert eine umfangreiche Online Help und Dokumentation unter
http://help.eclipse.org/luna/index.jsp. Für andere Versionen muss entsprechend der Name in der URL ausgetauscht werden.

Die Suche auf der Website kann themenbezogen gefiltert werden. Die Abbildung 1 zeigt die Website mit eingestelltem Filter für das CDT (*Scope: CDT*). Über den Link *Scope* wird der Dialog *Select Scope* geöffnet.

Im Vordergrund der Website ist der Auswahldialog *Select Scope* mit den Filtern *CDT* und *CDT Plug-in* und der Dialog *Edit Scope* zur Erstellung eines neuen Filters mit dem Filter *CDT Plug-in* mit ausgewählten Themen (Checked Boxes). Der Dialog wird über *New...* oder *Edit...* geöffnet.

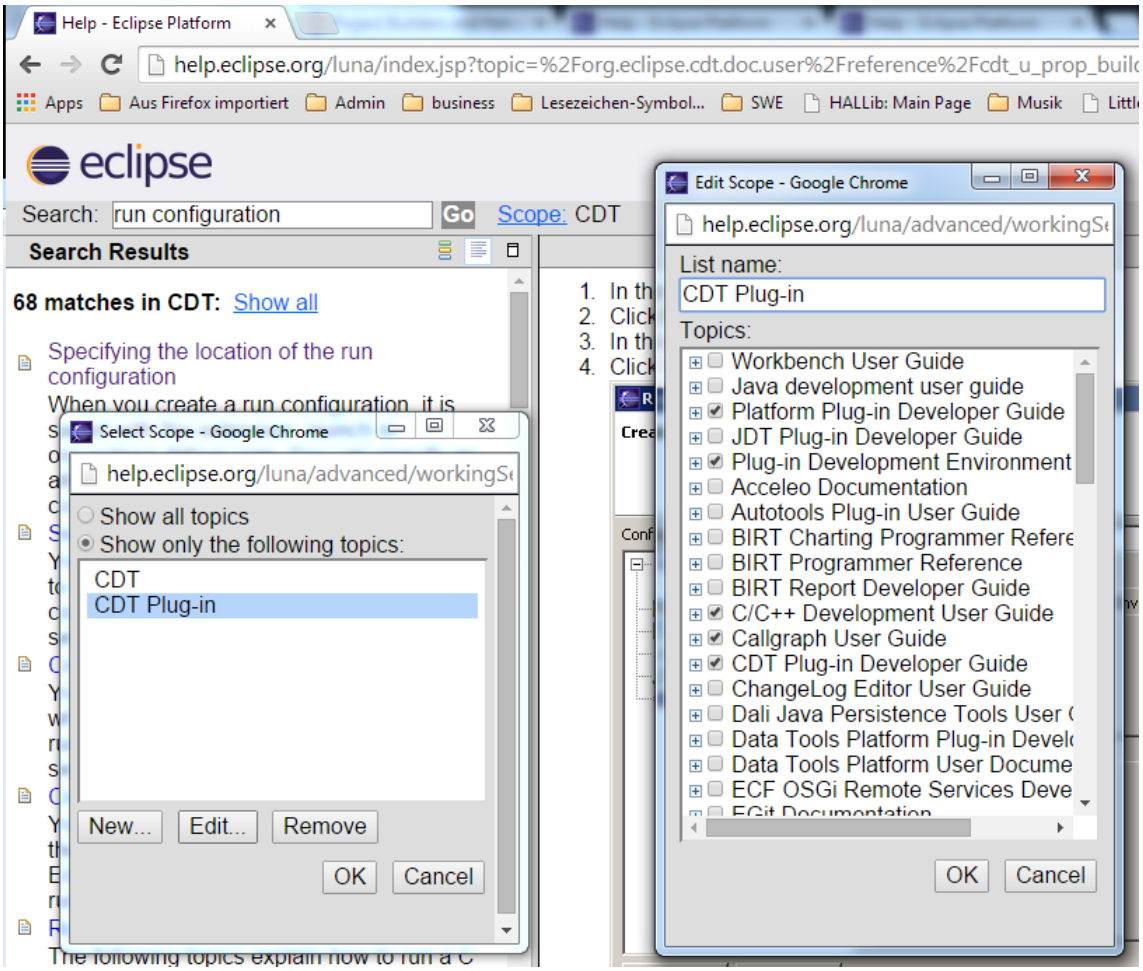

Abbildung 1: Die Eclipse Online Doku

Links zu Eclipse/CDT Themen:

- http://wiki.eclipse.org/index.php/CDT
- http://wiki.eclipse.org/CDT/User/FAQ
- http://help.eclipse.org/luna/index.jsp

Teil II

Grundlagen

5 Begriffe

Namen sind Schall und Rauch oder Nomen est Omen[6]?
Begriffe sind die Grundfesten unseres Denkens. Wer die Begriffe für die Dinge (von Eclipse) kennt, kann darüber nachdenken, sich austauschen, danach fragen und suchen.
In diesem Kapitel sollen die Begriffe und deren Beziehungen im Kontext von Eclipse vorgestellt werden.

5.1 Product, Feature und Plugin

Das Framework **Equinox** basiert auf **Plugins**, die durch XML Meta Informationen in der **Mainifest** Datei `plugin.xml` beschrieben werden.
Plugins sind die kleinste auslieferbare Einheit für Eclipse. Ein **Feature** sind mehrere Plugins, die zusammen als logische Einheit betrachtet werden können. Für ein Feature kann eine eigene *software license* definiert werden. Ein **Produkt** ist eine Anwendung, die auf der Eclipse *Rich Client Platform (RCP)* basiert. Ein Product wird auf der Basis von Features oder Plugins[7] definiert.
Die Meta Informationen beschreiben das Plugin, welche Module zu dem Plugin gehören und welche anderen Plugins benötigt werden, um das Plugin zu nutzen. Das Diagramm 1 auf der nächsten Seite zeigt diese Zusammenhänge in einem UML Klassendiagramm und Listing 1 auf der nächsten Seite zeigt eine einfache Manifest Datei für ein Plugin[8]. Plugins werden über das Menü *Help.Software Updates..* verwaltet und über die Update-Sites der Hersteller bezogen.

Die URL´s der installierten Komponenten können als XML File über *File.Export.Install.Installed Software Items to File* exportiert und wieder importiert werden. Beim Ex- und Import können die gewünschten Features ausgewählt werden. Beim Import werden bereits vorhandene URL´s nicht entfernt.

Welche Komponenten installiert sind, kann über den Dialog, der über *Help.About Eclipse.Installation Details* erreicht wird, abgefragt werden. Die installierten Komponenten können über diesen Dialog aktualisiert (*update...*) oder deinstalliert werden. Neue Komponenten können über *Help.Install New Software...* hinzugfügt werden, indem die URL der Update-Site angegeben wird.

[6] lat. der Name ist ein Zeichen http://de.wikipedia.org/wiki/Nomen_est_omen
[7] [Vog13a]
[8] http://wiki.eclipse.org/ Search for: “What is the plug-in manifest file”

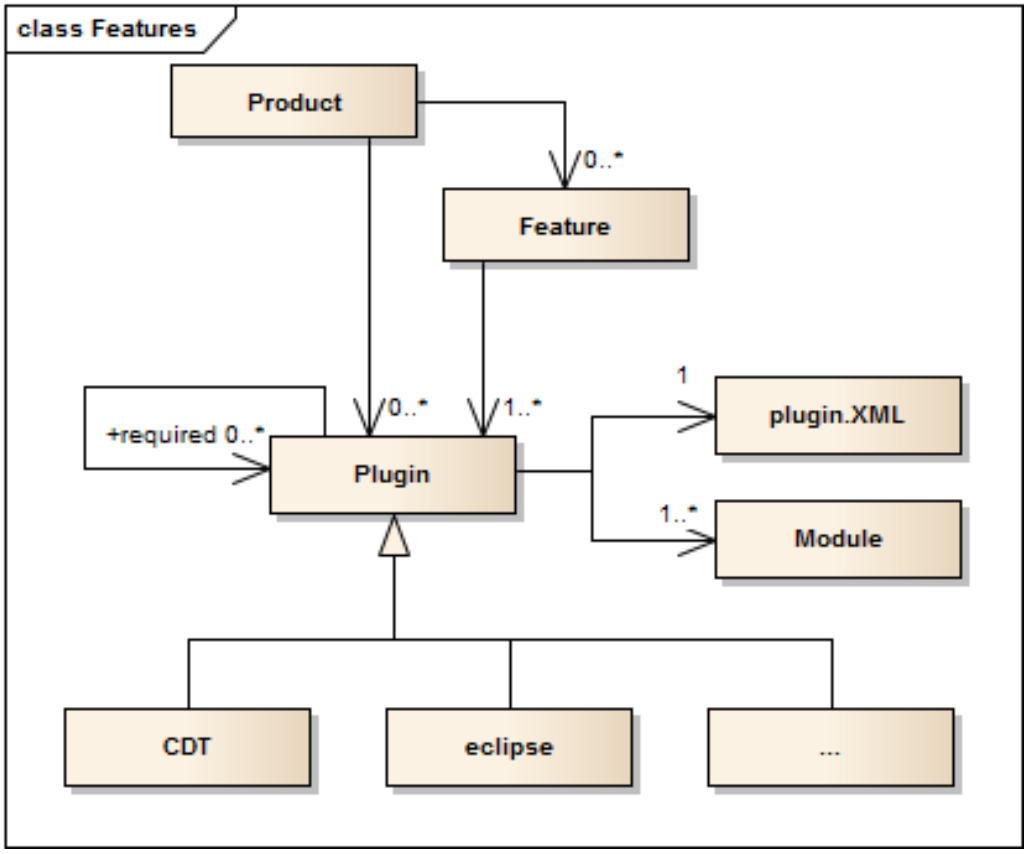

Diagramm 1: Features, Meta Informationen, Plugins und das CDT

Listing 1: Ein einfaches Manifest für ein Plugin

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?eclipse version="3.0"?>
3  <plugin id="com.xyz.myplugin" name="My Plugin" class="com.xyz.MyPlugin" version=
       "1.0">
4     <runtime>
5        <library name="MyPlugin.jar"/>
6     </runtime>
7     <requires>
8        <import plugin="org.eclipse.core.runtime"/>
9     </requires>
10 </plugin>
```

Das Plugin Konzept von Eclipse entspricht dem Bundle Konzept von OSGi.

Eclipse enthält einen *Update Manager*, der für die Verwaltung der installierten Komponenten verantwortlich ist. Mit der Komponente *director* des Update Managers können plug-ins über die Kommandozeile verwaltet werden[9]. Das Kommando in Listing 2 auf der nächsten Seite fügt die Komponenten egit, jgit, emf und mylyn der eclipse installation hinzu.

[9] http://blog.vogella.com/2012/04/04/installing-eclipse-features-via-the-command-line-with-the-p2-director/

Listing 2: Intalling Features via Commandline Arguments

```
eclipse/eclipse \
-application org.eclipse.equinox.p2.director \
-noSplash \
-repository \
http://download.eclipse.org/releases/juno \
-installIUs \
org.eclipse.egit.feature.group,\
org.eclipse.jgit.feature.group,\
org.eclipse.emf.sdk.feature.group,\
org.eclipse.mylyn_feature.feature.group
```

5.2 Instance, Workspace, Project und Resource

Ein **Workspace** ist ein Verzeichnis, in dem auf der obersten Ebene die Verzeichnisse der **Projekte** und darunter die projektspezifischen **Resourcen** verwaltet werden. Ein Workspace kann nur von einer **Eclipse Instance**[10] bearbeitet werden. Zu einer Eclipse Instanz kann ein oder mehrere Fenster/Window gehören. Das Hauptfenster von Eclipse ist in mehrere Subwindows aufgeteilt. Das Diagramm 2 zeigt diese Zusammenhänge in einem UML Klassendiagramm. Die

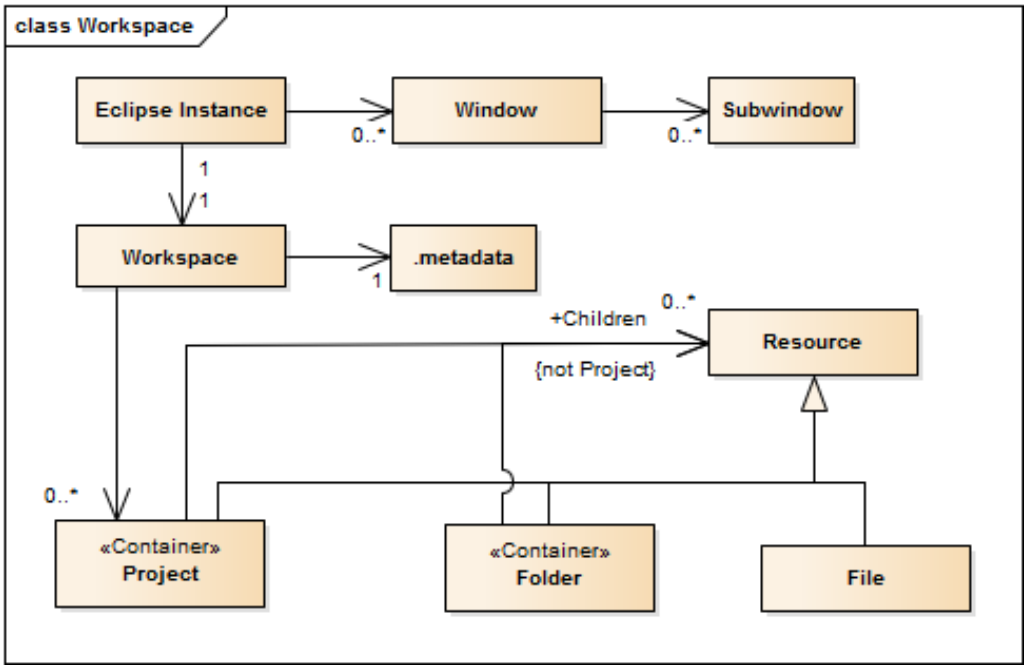

Diagramm 2: Eclipse Window, Workspace und Resourcen

Namen der Projektverzeichnisse entsprechen den Projektnamen. Projekte, Verzeichnisse / Folder und Files werden als Resourcen bezeichnet. Für Resourcen können jeweils spezifische Einstellungen vorgenommen werden. Verzeichnisse werden auch als *Container* bezeichnet. In diesen können weitere Folder oder Files verwaltet werden. Projekte sind mit einer oder mehreren **Nature**s assoziiert.

[10] ein ausgeführtes Programm

Eine Nature beschreibt den Zweck des Projekts und die dazu benötigten Werkzeuge, z.B. die Nature *LaTeX* definiert das Projekt als LaTeX Projekt.

Das Verzeichnis **.metadata** enthält die Einstellungen für diesen Workspace und die darin enthaltenen Projekte. Dieses Verzeichnis wird von Eclipse im Workspace angelegt, wenn ein Verzeichnis als Workspace ausgewählt wird.

In einem Workspace darf kein weiteres Workspace Verzeichnis und in einem Projektverzeichnis darf kein weiteres Projektverzeichnis existieren.

Beim Start von Eclipse kann über den Dialog in Abbildung 2 ein Verzeichnis als Workspace ausgewählt werden.

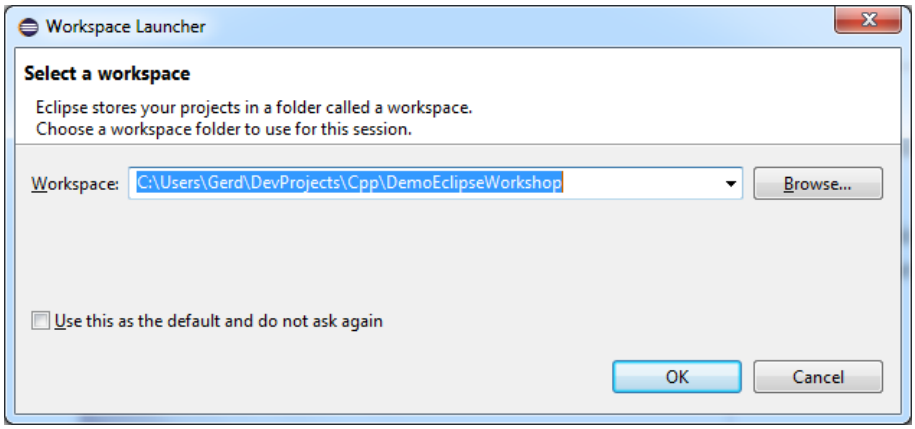

Abbildung 2: Der Workspace Auswahl Dialog beim Start von Eclipse

5.3 Perspective, View und Editor

Eine **Perspective** ist eine Auswahl verschiedener **Views** und Toolbars und deren Anordnung. Diese Auswahl kann frei gewählt und unter einem Namen gespeichert werden. Die aktuelle Perspective in Abbildung 3 ist *C/C++*, sie ist in der Toolbar hervorgehoben.

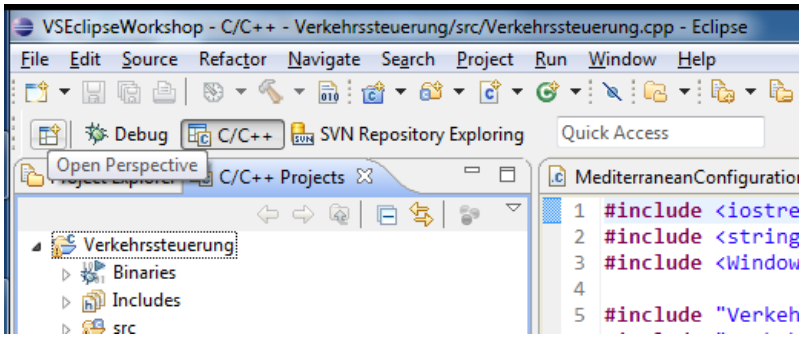

Abbildung 3: Die Toolbar mit ausgewählter Perspective C/C++

Das Kontextmenü der Perspective ermöglicht

- save as, speichert die Auswahl und Anordnung unter einem Namen

- Reset, überführt die Perspective in den letzten gespeicherten Zustand
- close, schließt die Perspective
- Show Text, zeigt nach dem Icon den Namen der Perspective

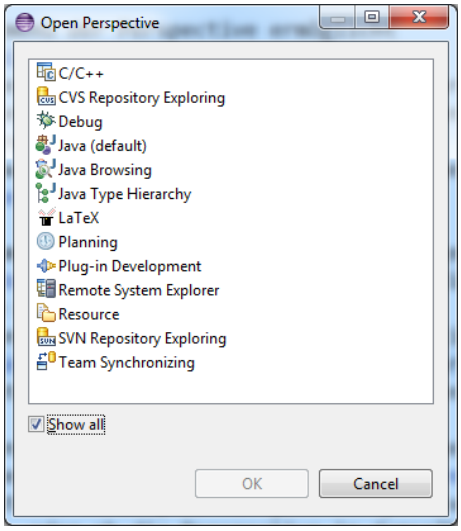

Abbildung 4: Dialog zur Auswahl einer Perspective

Eine Perspective kann über den Dialog in Abbildung 4 ausgewählt werden. Der Dialog kann über das Menü *Window.Open Perspective* oder über das entsprechende Toolbar Icon (links neben Debug) geöffnet werden.

Wird eine bereits geöffnete Perspective ausgewählt, wird die **letzte Anordnung** der Perspective wieder hergestellt. Wird eine Perspective ausgewählt, die noch nicht geöffnet ist, wird der **zuletzt gespeicherte Zustand** wieder hergestellt. Das kann ebenfalls über das Kontextmenü *Reset* einer geöffneten Perspective erreicht werden. Die geöffneten Editoren werden von Perspectiven nicht berücksichtigt.

Bestimmte Aktivitäten, wie z.B. Debugging, können mit einer Perspective verknüpft werden, in die automatisch gewechselt wird, wenn die Aktivität durchgeführt wird. Die Einstellung dafür ist nicht einheitlich und jeweils bei den Plugins, die die Aktivitäten ermöglichen, auf andere Weise möglich.
In einem Fenster/**Window** ist immer eine oder keine Perspective geöffnet. In *Window.Preferences.General.Perspectives* kann eingestellt werden, ob für jede Perspective ein neues Fenster geöffnet werden soll oder ob die Perspective in demselben Fenster geöffnet wird. Die Views und Editoren werden als Tabs in Subwindows dargestellt[11].

Das Diagramm 3 auf der nächsten Seite zeigt den Zusammenhang zwischen diesen Begriffen und Abbildung 5 auf der nächsten Seite zeigt ein Eclipse Window mit 4 Subwindows in denen die Views als Tabs verwaltet werden.
Links das Subwindow mit zwei Navigation Views, in der Mitte die Editoren, rechts daneben ein Subwindow mit der Outline View und der Make Target View und darunter ein Subwindow mit der Console und weiterer Views.

[11] mehr dazu in section 8 auf Seite 22

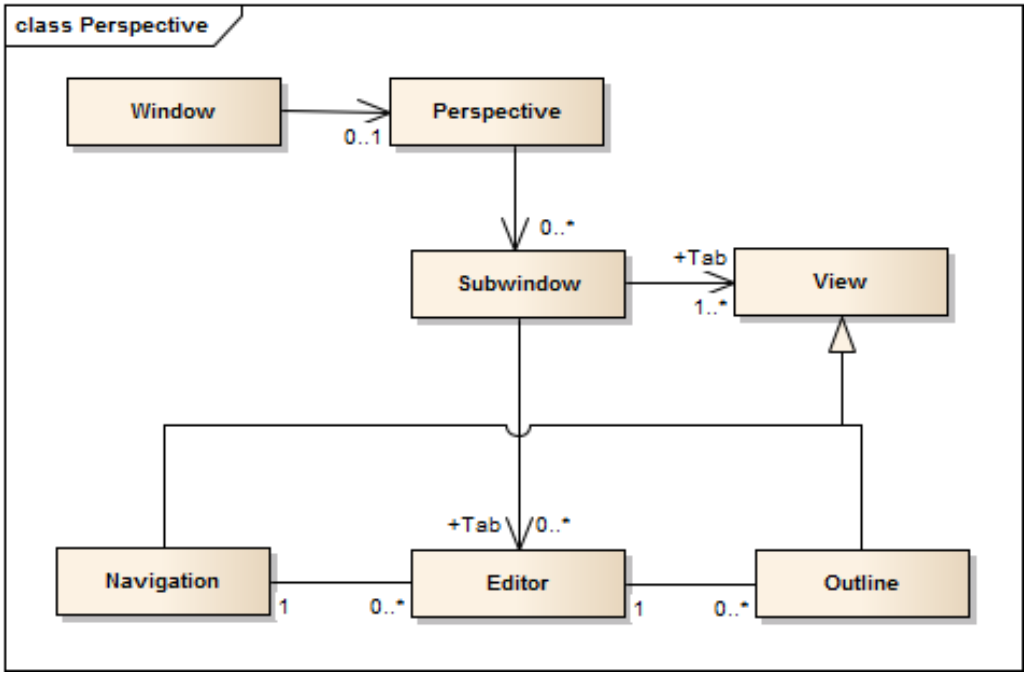

Diagramm 3: Eclipse Window, Perspectives, Views, Editors

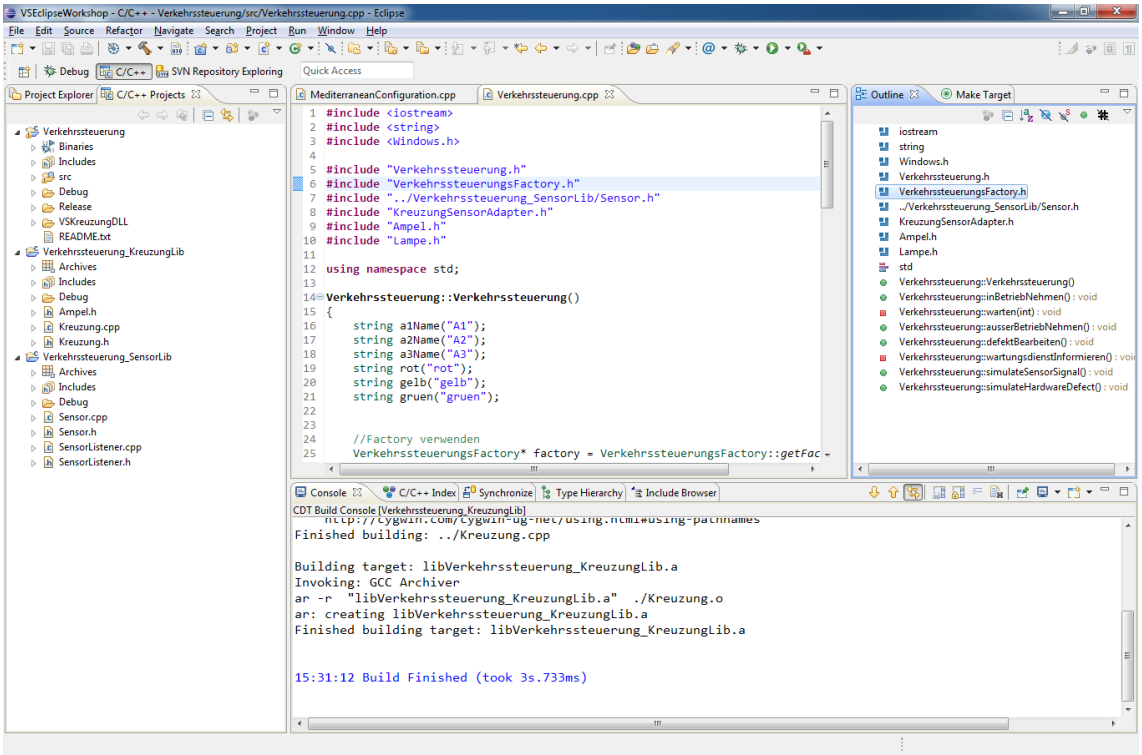

Abbildung 5: Subwindows und Views in einer Perspective

5.4 Working Sets

Die wichtigsten Working Sets sind Resource- und für die Software Entwicklung Breakpoint[12] Working Sets. Sie gruppieren Resourcen oder Breakpoints. In ver-

[12] siehe section 10.7.2 auf Seite 59

schiedenen Views und beim Suchen können Working Sets als Filter für den angezeigten Inhalt verwendet werden.

Im Pulldown Menü des Project Explorers wird über *Select Working Set* ein vorhandenes Working Set ausgewählt. In dem Dialog besteht auch die Möglichkeit Neue zu definieren oder Vorhandene zu bearbeiten. In Abbildung 6 ist der Auswahldialog und der Dialog zur Bearbeitung eines Working Sets dargestellt.

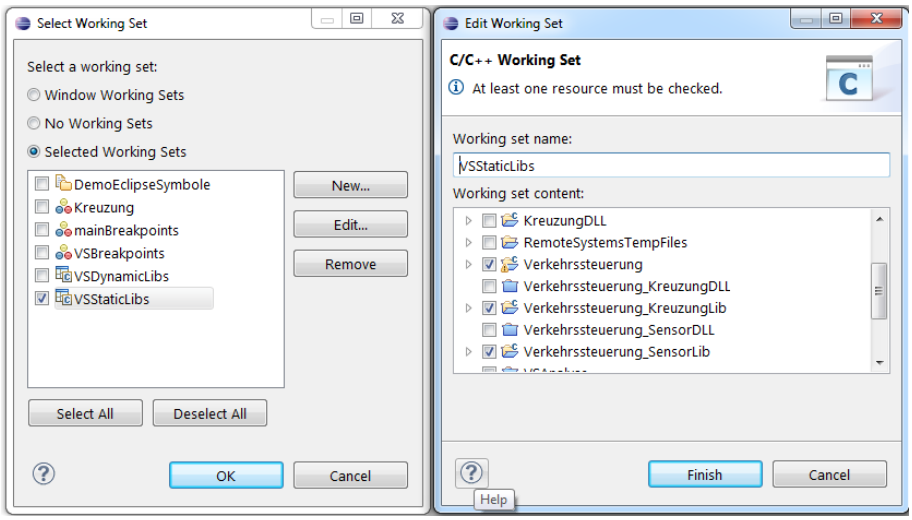

Abbildung 6: Der Dialog zur Berarbeitung eines Resource Working Sets

Werden mehrere Working Sets als Filter ausgewählt, wird die Vereinigungsmenge der Resourcen aus den ausgewählten Working Sets angezeigt. Werden Working Sets auf der Basis von *Containern* aus Diagramm 2 auf Seite 13 definiert, sind alle Children enthalten.

6 Einstellungen

Die Einstellungen der Plugins werden in Textdateien verwaltet und können meistens ex- und importiert werden[13]. Speziellere Einstellungen überschreiben allgemeine Einstellungen, d.h. die Einstellungen für ein File überschreiben die Einstellungen des Folders, dessen Einstellungen die Projekteinstellungen, usw. Mit *Restore Defaults* werden die spezifischen Einstellungen auf die Einstellungen des übergeordneten Elements gesetzt.
Eclipse <- Workspace <- Projekt <- Parentfolder <- Folder <- File

Wird z.B. für einen Source Folder die Compiler Option `-std=c++11` gesetzt und für eine bestimmtes File in diesem Folder die Option ausgeschaltet, bleibt die Option für das File ausgeschaltet, wenn für den Folder die Option ausgeschaltet wird und danach wieder eingeschaltet wird, weil für das File die spezifischen Einstellungen weiter ihre Gültigkeit behalten. Damit für die Datei die Einstellungen des Folders angewendet werden, müssen diese mit *Restore Defaults* zurückgesetzt werden.

[13] workaround fehlende ex-/imports: section 7.2 auf Seite 20

Resourcen, die von ihrem Parentfolder abweichende Einstellungen haben, werden in den Navigation Views mit bestimmten Verzierungen ausgezeichnet[14]. Die Abbildung 7 zeigt den Project Explorer mit dem Projekt DemoSymboleNavigationView. Das *Modul1* hat keine abweichenden Einstellungen, aber die darin enthaltene Resource *Implementation.cpp*, erkennbar an dem kleinen “Schraubenschlüssel” Symbol. Die Resourcen *Helper.cpp* und *Modul3* sind vom Build ausgeschlossen (*Exclude resource from Build*).

Für das gesamte *Modul2* wurden spezielle Einstellungen vorgenommen und für die Datei *Implementation2.cpp* davon abweichende.

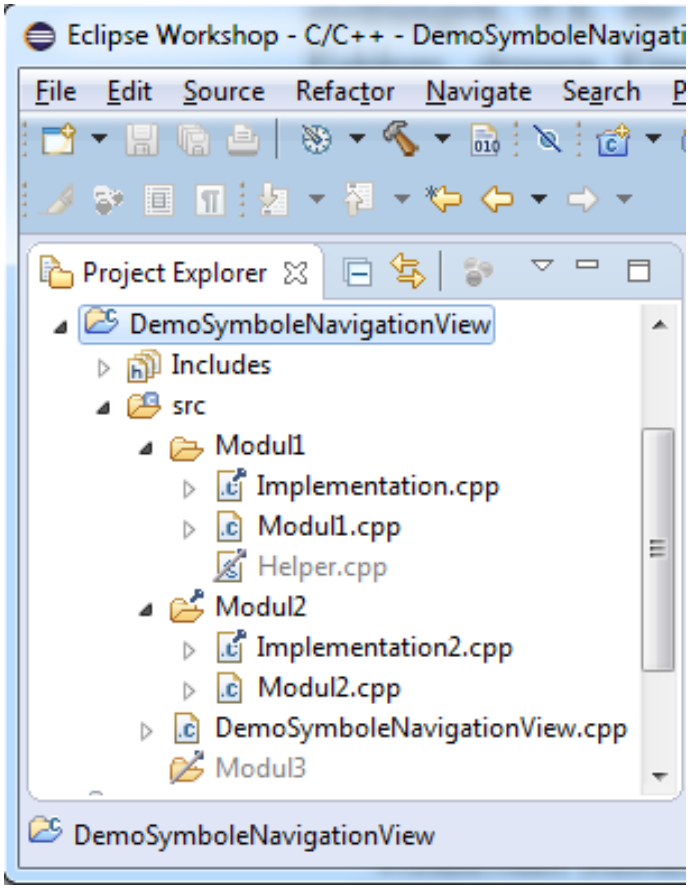

Abbildung 7: Adornments für Resourcen mit speziellen Properties

6.1 Globale und Workspace Einstellungen

Die globalen Einstellungen werden im Installationsverzeichnis von Eclipse verwaltet. Die Workspace spezifischen Einstellungen werden im Verzeichnis *.metadata* im jeweiligen Workspace verwaltet.
Der Dialog für globale und Workspace spezifische Einstellungen in Abbildung 8 auf der nächsten Seite wird über das Menü *Window.Preferences* erreicht. Links im Dialog befindet sich eine Baumansicht mit den Plugins, für die Einstellungen

[14] Ab Eclipse/CDT Luna

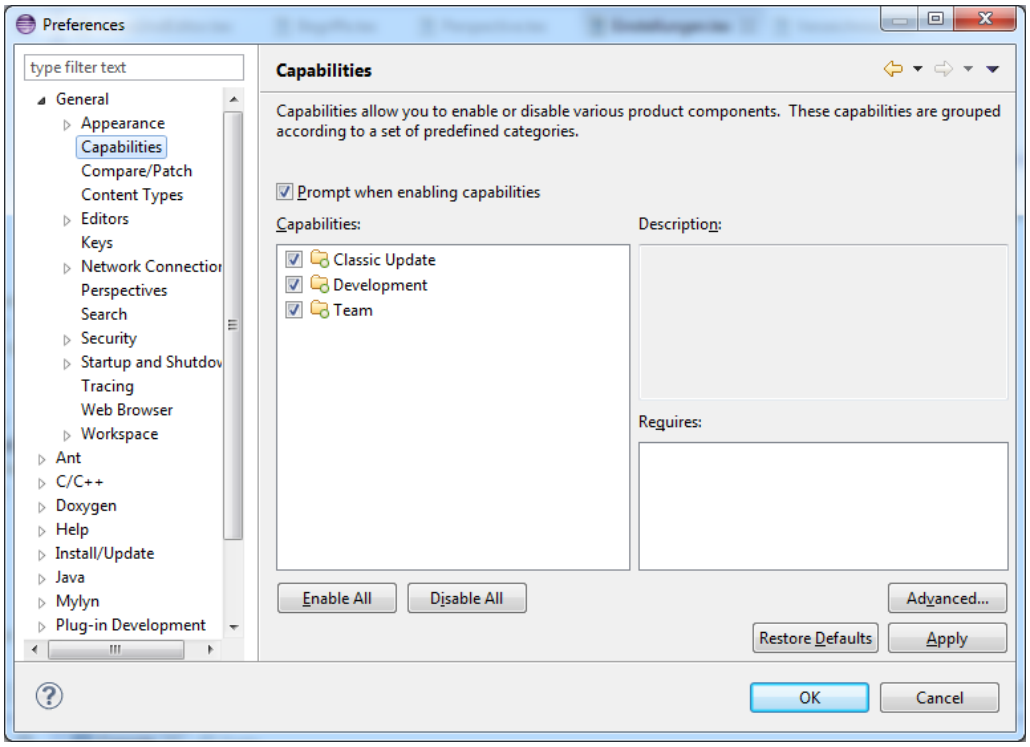

Abbildung 8: Der Preferences Dialog

vorgenommen werden können, rechts die Einstellungsmöglichkeiten für den ausgewählten Knoten. Die Art und Anzahl der Knoten variiert sehr in Abhängigkeit der installierten Plugins. Die Anordnung der Einstellmöglichkeiten variiert von Version zu Version etwas. Um bestimmte Einstellungen schneller zu finden, kann über der Baumansicht ein Begriff als Filter (*type filter text*) eingegeben werden.
Tipp: Das Wichtigste im Umgang mit den Properties ist die Kenntnis der Eclipse spezifischen Begriffe.
Im Knoten *General.Capabilities* können Plugins, die für diesen Workspace nicht relevant sind de-/ aktiviert werden. Leider gilt das nur für Plugins die sich konform verhalten und sich entsprechend in dieser Liste eintragen. In neueren Versionen gibt es diesen Knoten nicht mehr.
Welche Plugins beim Start, bzw. beim Öffnen des Workspaces geladen werden sollen, kann im Knoten *General.Startup and Shutdown* eingestellt werden.

6.2 Projekt Einstellungen

Über das Menü *Project.Properties*, den Shortcut **Alt+Enter** eines ausgewählten Projekts oder das Kontextmenu des Projekts *<ProjektName>.Properties* in einer Navigation View können die projektspezifischen Einstellungen vorgenommen werden. Der Dialog ist derselbe wie der für die Workspace Einstellungen mit eingeschränktem Inhalt.

6.3 Resource Einstellungen

Über das Kontextmenu einer Resource *<ResourceName>.Properties* in einer Navigation View oder den Shortcut **Alt+Enter** einer ausgewählten Resource können die resourcespezifischen Einstellungen vorgenommen werden. Der Dialog ist derselbe wie der für die Workspace Einstellungen mit eingeschränktem Inhalt.

6.4 Export und Import

Im Menü *File.Import* bzw. *File.Export* können verschiedene Einstellungen der Projekte und des Workspace Ex- bzw. Importiert werden.

7 Verzeichnisse

7.1 Das Eclipse Installationsverzeichnis

Das Eclipse Installationsverzeichnis sollte nicht direkt manipuliert werden. Der Inhalt wird ausschließlich über die Einstellungsdialoge von Eclipse verändert. Der Inhalt besteht u.a. aus:

- eclipse Program Launcher (exe)
- eclipse.ini mit den Einstellungen für die runtime Umgebung[15]
- Verzeichnis `configuration` mit Workspace übergreifenden Einstellungen z.B. Netzwerk
- Weitere Verzeichnise z.B. dropins, features, plugins, p2, ...

Der Program Launcher startet Eclipse mit den Einstellungen aus der Datei *eclipse.ini*. Dem Program Launcher können Parameter übergeben werden, diese überschreiben die Einstellungen aus *eclipse.ini*. Die Syntax ist: *<eclipse-args> -vmargs <java-vm-args>*. Die ersten Parameter sind die Parameter für Eclipse, nach der Option *-vmargs* kommen die Parameter für die Java Virtual Machine. Zum Beispiel kann das Verzeichnis des Workspace mit *eclipse -data <workspaceverzeichnisname>* beim Start von Eclipse angegeben werden.

Eclipse plug-ins, die nicht über den Update Manager installiert werden sollen, können in das dropins Verzeichnis kopiert werden. Beim Starten wird dieses Verzeichnis von Eclipse ausgewertet und alle enthaltenen plug-ins werden aktiviert.

7.2 Das Workspace Verzeichnis

Im Workspace Verzeichnis existiert für jedes Projekt ein Verzeichnis mit dem Namen des Projekts und das Verzeichnis *.metadata*, das die Informationen über die

[15] http://wiki.eclipse.org/Eclipse.ini

Projekte, die Einstellungen der Perspektiven, usw. des Workspace enthält, z.B. befinden sich in der Datai *.log* alle geloggten Ausgaben von Eclipse für die Fehleranalyse.

Als Workaround für fehlende Ex-/Import Möglichkeiten können verschiedene Verzeichnisse im Workspace kopiert werden:
z.B. im Verzeichnis
workspace/.metadata/.plugins/org.eclipse.ui.workbench/ ist das Layout der Workbench und die Perspectiven gespeichert.
Im Verzeichnis
workspace/.metadata/.plugins/org.eclipse.core.runtime/.settings/ sind die Settings aus *Window.Preferences* gespeichert. Die verschiedenen Plugins erweitern den Inhalt dieses Verzeichnis mit eigenen Konfigurationsinformationen.

7.3 Das Project Verzeichnis

Im Project Verzeichnis existiert die Datei *.project*, die die projektspezifischen Einstellungen enthält. In Abhängigkeit des jeweiligen Projekttyps[16] existieren weitere Dateien, z.B. existiert in einem CDT Projekt die Datei *.cproject* mit den CDT spezifischen Einstellungen, z.B. die Build Konfigurationen, oder in einem LaTeX Projekt die Datei *.texlipse*.
Im Verzeichnis von CDT Projekten befinden sich die Sourcen in speziellen Sourcefoldern. Dateien in diesen Verzeichnissen werden indiziert[17] und kompiliert. Über das Kontext Menü *Properties* kann der Dialog in Abbildung 9 des Verzeichnisses geöffnet werden und die “Sourcefolder” Eigenschaft mit *Exclude resource from build* eingestellt werden.

Für jede Build Configuration[18] wird ein Verzeichnis angelegt, das die Ergebnisse des Builds mit dieser Configuration enthält.

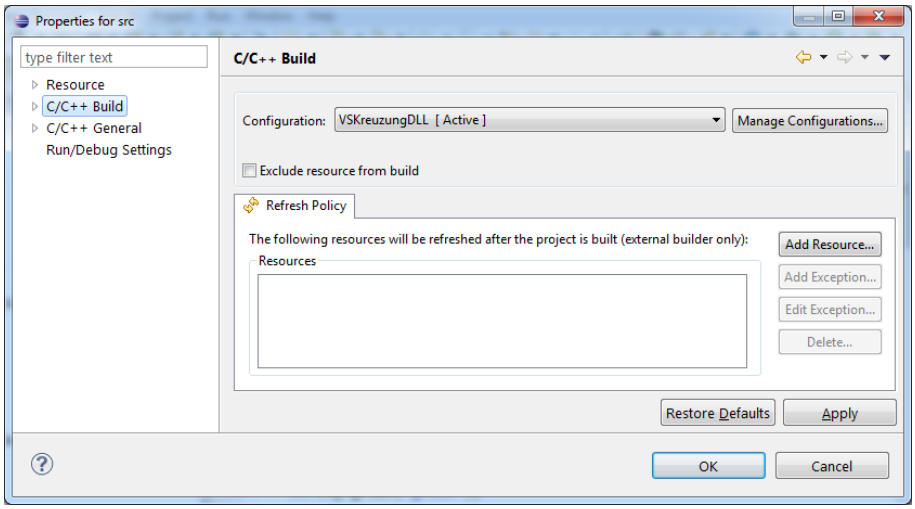

Abbildung 9: Der Properties Dialog eines Verzeichnisses

[16] Eclipse Jargon: Nature
[17] section 10.3 auf Seite 42
[18] siehe section 10.8.6 auf Seite 74

8 Bedienung

Die Bedienung basiert auf den Standards von graphischen Oberflächen, mit Menüs, Mnemonics[19], Toolbars, Kontextmenüs, Shortcuts und Drag´n Drop. Dazu kommen die Pulldown Menüs der Views und das Prinzip der Perspektiven. Sie sind zum Teil an eigene Bedürfnisse anpassbar und sollten Organisations- oder Projektweit standardisiert werden.

Erfolgt die Bedienung überwiegend mit der Tastatur, kann die Bearbeitungsgeschwindigkeit um den Faktor 10 bis 100 gegenüber der Bedienung mit der Maus gesteigert werden. Vorausgesetzt, die entsprechenden Shortcuts sind bekannt. Es lohnt sich daher, sich mit der Bedienung über die Tastatur auseinander zu setzen, auch wenn dadurch am Anfang die Bedienung erschwert ist. Die Shortcuts werden meistens in den Menüs und in der Toolbar in der Bubblehelp angezeigt.

Tipp: Tastenkombinationen für Shortcuts immer mit denselben Fingern ausführen, damit kann die Tastatur blind bedient werden.

Kombinationen:

- **Ctrl+Taste** kleiner linker Finger
- **Shift** linker Ringfinger
- **Alt+Taste** linker Daumen
- **Taste** linker Zeigefinger oder Tastenfinger der rechten Hand

Kann die Taste mit dem linken Zeigefinger nicht erreicht werden, muss die rechte Hand dazu genommen werden.
Beispiel:**Ctrl+Shift+R** kleiner Finger + Ringfinger + Zeigefinger der linken Hand öffnet den *Open Resource* Dialog[20]. Für Menschen, die die Maus mit der linken Hand bedienen, entsprechend die Finger der rechten Hand benutzen;-)

8.1 Subwindows, Views und Editoren

Mit Views und Editoren können Informationen über Resourcen angezeigt und verändert werden. Die Views und Editoren werden als Tabs in Subwindows verwaltet. Werden alle Tabs in einem Subwindow geschlossen, wird auch das Subwindow geschlossen.
Ausnahme: das Subwindow der Editoren bleibt ohne Tab geöffnet.
Über das Kontextmenü eines Tabs können alle (*Close All*), alle anderen (*Close Others*) oder der ausgewählte Tab (*Close*) geschlossen werden.
Eclipse unterstützt die Verwendung mehrerer Monitore. Die Subwindows können aus dem Hauptfenster von Eclipse herausgelöst werden und frei über den verfügbaren Platz auf den Monitoren verteilt werden, indem ein Tab aus dem Haupfenster gezogen wird (**Drag´n Drop**).

[19] Gedächtnisstütze, griechisch mnemonia, Gedächtnis (unterstrichener Buchstabe im Menütext)
[20] siehe section 9.2 auf Seite 31

Alle Views und die Editoren können innerhalb der Subwindows frei positioniert werden. Dazu wird der Tab der View per **Drag´n Drop** im selben oder einem anderen Subwindow positioniert.

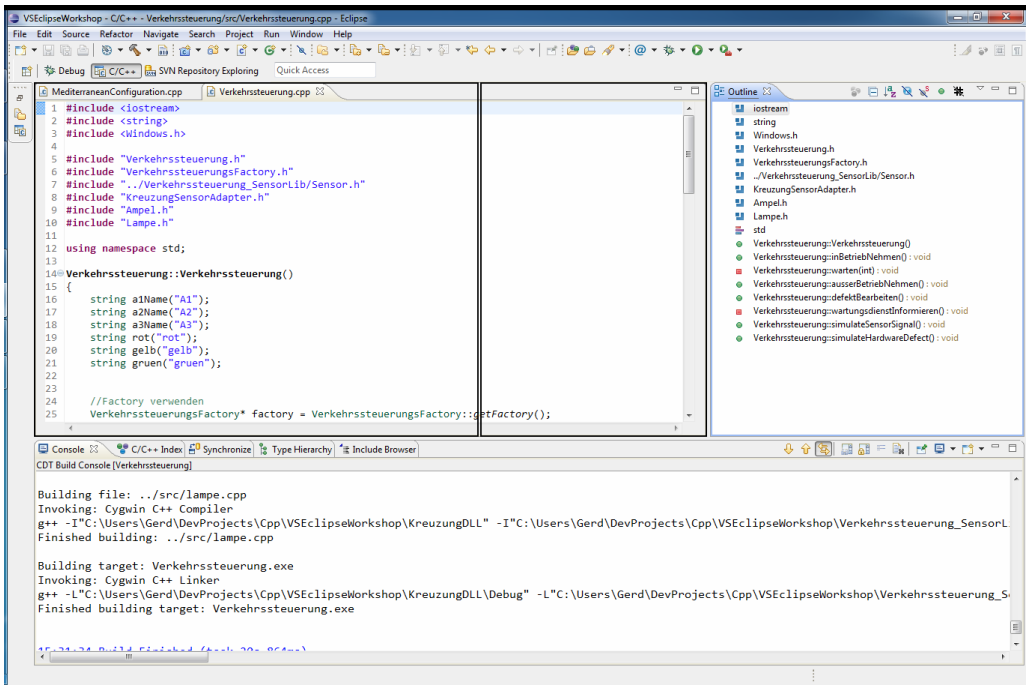

Abbildung 10: Einen Tab in ein anderes Subwindow verschieben

Die Abbildung 10 zeigt die Darstellung der neuen Position eines Tabs im Subwindow der Editoren. Die beiden schwarzen Rahmen zeigen die Aufteilung des Subwindows in zwei neue Subwindows.
Wird der Tab in die **Tab-Leiste** eines anderen Subwindows gezogen, wird die View oder der Editor in dem Subwindow als Tab verwaltet und nimmt keinen eigenen Platz ein.

Die Subwindows haben rechts oben ein Menü zum Minimieren und Maximieren des Subwindows. Der Shortcut **Ctrl+m** maximiert das Subwindow, bzw. stellt es wieder normal dar.
Minimiert erscheinen die Subwindows in einer Toolbar ähnlichen Darstellung (links am Rand unterhalb der Toolbar links neben dem Editor in Abbildung 10) die am Rand frei positionierbar sind. Die Tabs werden als Icons dargestellt. Mit einem Klick auf das Icon werden die Views zur Verwendung geöffnet und schließen sich selbständig wieder, wenn der Focus in ein anderes Fenster wechselt. Das erste Icon in diesen Toolbars stellt die ursprüngliche Darstellung wieder her.

8.1.1 Views und Resourcen

Views zeigen spezifische Informationen bzgl. ausgewählter Resourcen an.
Sie werden über das Menü *Window.Show View* oder über den Shortcut **Alt+Shift+Q, <zeichen>** geöffnet. Ist die gewünschte View nicht in der Auswahlliste, kann der Dialog in Abbildung 11 auf der nächsten Seite über *Other...* geöff-

net werden. Die Abbildung zeigt die General Views von Eclipse. In diesem Dialog finden sich alle Views der Plugins, z.B. C/C++, Subversion, usw.

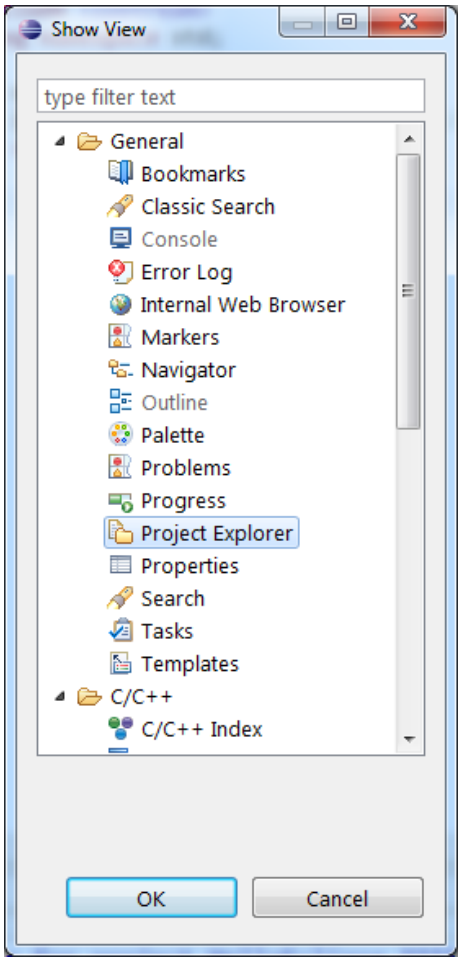

Abbildung 11: General Views von Eclipse

In manchen Views können die angezeigten Werte verändert werden. Die Veränderung wird ohne explizites Speichern beim Verlassen des Eingabefeldes sofort wirksam.

Jede View hat ein Pulldown Menü und eine Toolbar, für View spezifische Einstellungen. Einträge, die häufig benötigt werden, finden sich in der Toolbar der View wieder. Die Toolbar und Icons zur Steuerung der View befinden sich auf derselben Zeile wie die Tabs und das Menü des Subwindows, wenn das Subwindow breit genug ist, ansonsten eine Zeile tiefer.

Manche Views können mit *Pin <View-Name>* an die aktuell ausgewählte Resource gebunden werden. Die Auswirkungen sind View spezifisch[21].

Hierarchische Informationen werden als Trees, die aufgeklappt werden können, in den Views angezeigt.

[21] siehe z.B. section 9.6 auf Seite 36

8.1.2 Views und Editoren

Die Unterscheidung zwischen Editoren und Views ist durch die unterschiedliche Verwendung bedingt. Änderungen die mit Editoren durchgeführt werden, müssen gespeichert werden, damit die Resource geändert ist. Änderungen die über eine View durchgeführt werden, werden direkt angewandt[22].

Eine veränderte, nicht gespeicherte Resource, wird mit einem Sternchen vor dem Dateinamen im Tab des Editors angezeigt. Mit **Ctrl+s** kann der Inhalt der aktuellen Resource, bzw. mit **Ctrl+Shift+s**, der Inhalt aller veränderten Resourcen gespeichert werden,

Aus den meisten Views kann mit einem **klick** zu dem korrespondierenden Symbol im Editor navigiert werden und mit einem **doppel klick** der Editor für dieses Element geöffnet werden, z.B. in der Outline View auf ein `include` eines Headers.

Einige Views müssen nach Änderungen der angezeigten Resource z.B. durch andere Views oder Editoren zuerst aktualisiert werden (*Refresh View Content*), damit die Informationen korrekt angezeigt werden. In der Toolbar dieser Views befindet sich ein entsprechendes Icon wie in Abbildung 12.

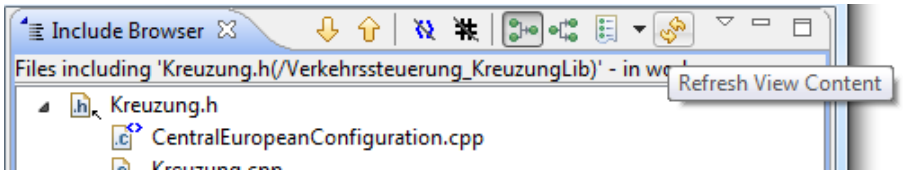

Abbildung 12: Views die aktualisiert werden müssen

Die meisten Views können über *Link with Editor* mit dem Editor verknüpft werden. Dadurch zeigt die View immer das ausgewählte Element im Editor oder das bearbeitete File z.B.:`ClassSymbols.h` wie in Abbildung 13 an.

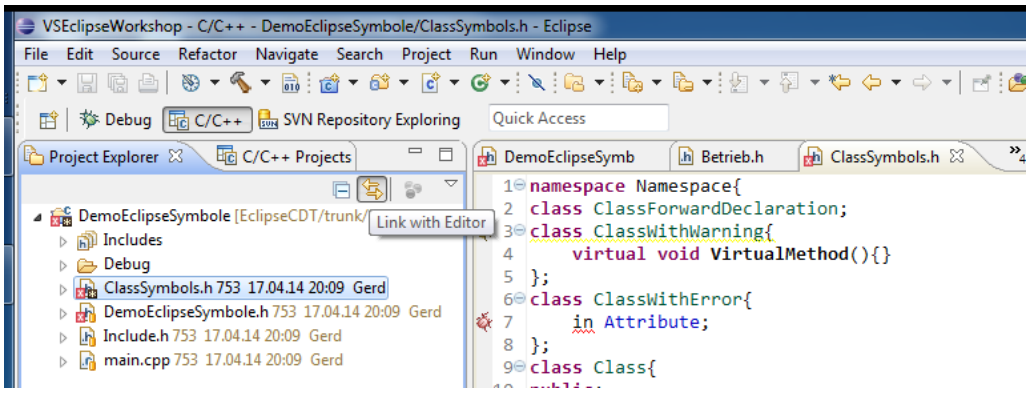

Abbildung 13: Project Explorer Link with Editor

[22][Vog13b]

8.1.3 Editoren und Resourcen

Die Editoren für die Resourcen werden mit einem Doppelklick auf der Resource in einer Navigation View oder über das Kontextmenü *open* der Resource geöffnet. Alternativ kann eine Resource über **Drag´n Drop** in das Editorfenster im Editor geöffnet werden.

Jede Resource wird in einem eigenen Editor geöffnet. Welcher Editor für welchen Resourcetyp verwendet wird, kann in
Window.Preferences.General.Editors.File Associations eingestellt werden.

Die zeichenweise Navigation im Editor erfolgt über die **Pfeiltasten rechts/links**, mit gedrückter **Ctrl**-Taste kann Wortweise navigiert werden und mit gedrückter **Shift**-Taste markiert werden. Kombinationen daraus sind möglich, z.B. wird mit **Ctrl+Shift+Pfeil nach rechts** das nächste Wort markiert.
Zeichenweises Löschen wird mit **Backspace** und **Delete** ausgeführt, mit **Ctrl** erfolgt daselbe wortweise.

Pfeiltasten nach oben/unten bewegen den Cursor um eine Zeile, in Kombination mit **Ctrl** wird die gesamte Sicht entsprechend verschoben, der Cursor bleibt an seiner Position.
Pfeiltasten nach oben/unten in Kombination mit **Alt** verschiebt die aktuelle Zeile oder den markierten Bereich und mit
Ctrl+Alt wird die Zeile oder der markierte Bereich kopiert.

Die Shortcuts sind zum Teil kombiniert:
Alt+Shift+Q, c öffnet die Console, wenn keine weitere Taste (**c**) gedrückt wird, erscheint ein Menü zur Auswahl der gewünschten Option.

Die wichtigsten Kombinationen zur Bedienung mit der Tastatur:

- **Ctrl+Shift+L** zeigt alle Shortcuts
- Markierung Copy: **Ctrl c**, Cut: **Ctrl x**,
- Paste: **Ctrl v**
- Undo: **Ctrl z**, Redo: **Ctrl y**
- **Ctrl+Q** Position der letzten Änderung
- **Alt+Pfeiltaste links** vorherige Positionen in allen Files
- **Alt+Pfeiltast rechts** nachfolgende Positionen
- Auswahl aus Menü: **Alt+Mnemonic**
  (der unterstrichene Buchstabe im Menütext)
- **Alt+Shift+Q** öffnet ein Menü für kombinierte Shortcuts für verschiedene Views
- **Alt+Shift+A** Blockmodus ein/ausschalten
- **Ctrl+m** Fullscreen toggle
- Wechsel zwischen Header und Cpp **Ctrl+Tab**

- **Ctrl+Shift+E** oder **Ctrl+F6** zeigt einen Auswahldialog der geöffneten Editoren
- **Ctrl+Click** oder **F3** navigiert zur Deklaration/Definition des Symbols an der Cursor Position
- korrespondierende Klammern: **Ctrl+Shift+P**
- Quick Outline **Ctrl+o**[23]
- Quick Type Hierarchie **Ctrl+t**[24]
- Code Assistance (vervollständigen) **Ctrl+Blank**[25], wiederholtes **Ctrl+Blank** zeigt verschiedene Alternativen und Code Templates[26] an
- **Ctrl+1** zeigt die Quick Fix Vorschläge an, wenn der Cursor auf einer fehlerhaften Codestelle steht

Am linken Rand kann ein Kontext Menü geöffnet werden, das Editor spezifische Einstellungen wie Font, Zeilennummern, Folding, usw. erlaubt.

8.1.4 Navigation zu Warnings und Errors, Annotations

Fehlermeldungen, Warnings und weitere ergänzende Informationen des Codes werden als Annotations bezeichnet. Mit **Ctrl+.** und **Ctrl+,** (Punkt und Komma) kann innerhalb einer geöffneten Datei zu der nächsten Annotation bzw. zu der vorherigen navigiert werden. Welche Annotations bei der Navigation berücksichtigt werden sollen, kann über das Dropdown Menü in der Toolbar der entsprechenden Buttons in Abbildung 14 eingestellt werden.

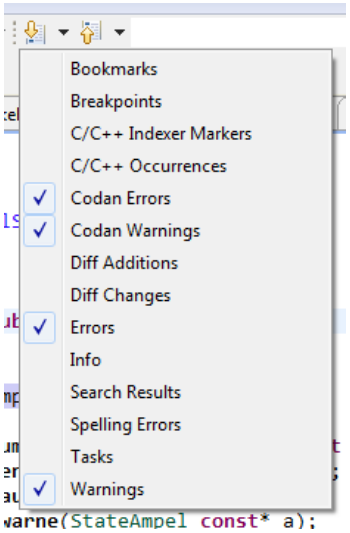

Abbildung 14: Navigation zu Annotations in der Toolbar

[23] siehe section 10.5.4 auf Seite 45
[24] siehe section 10.5.7 auf Seite 52
[25] Blank, die Leertaste
[26] siehe section 10.6.1 auf Seite 54

8.2 Dateien Vergleichen

Eclipse unterstützt den Vergleich von

- zwei ausgewählten Dateien
- den Vergleich einer Datei mit einer Vorgängerversion, die von Eclipse lokal verwaltet wird
- den Vergleich einer Datei mit einer Revision aus dem Repository einer Versionsverwaltung und
- einen 3 Wegevergleich zur Bereinigung von Konflikten

Um zwei Dateien miteinander zu vergleichen, werden diese in einer Navigation View mit **Ctrl+Click** ausgewählt. Anschließend kann über das Kontextmenü *Compare With.Each Other* ausgewählt werden. Die beiden Dateien werden nebeneinander dargestellt und die abweichenden Zeilen aufeinander abgebildet. Über die Toolbar in dem Fenster kann zwischen den Abweichungen navigiert werden. An beiden Dateien können Änderungen durchgeführt werden. Nach dem Speichern wird der Vergleich erneut durchgeführt.

Wird nur eine Datei ausgewählt, kann der Inhalt mit einer Version aus der local History von Eclipse verglichen werden oder mit einer Version aus dem Repository. Bei diesem Vergleich kann nur an der Workspace Kopie der Datei Änderungen durchgeführt werden.

Für Konflikte mit Änderungen von anderen Entwicklern steht ein 3 Wege Vergleich zur Verfügung[27].

Tipp: mit **Ctrl+m** das Subwindow maximieren um den Vergleich durchzuführen und anschließend wieder minimieren.

8.3 Suchen und Ersetzen

- Suchen/Ersetzen Dialog: **Ctrl+f**
- **Ctrl+k** sucht nach dem nächsten Auftreten eines markierten Textes, **Ctrl+Shift+K** nach dem vorherigen.
- **Ctrl+j** incremental Find **Ctrl+Shift+J** Backward
- Datei übergreifende Suche **Ctrl+h**

Der incremental Find sucht sofort nach dem eingegebenen Teilstring, der in der Statuszeile angezeigt wird.

Der Dialog zur Datei übergreifenden Suche mit dem Tab *C/C++ Search* ist in Abbildung 15 auf der nächsten Seite dargestellt.

Beachte: Gesucht wird nur in geöffneten Projekten!

[27] http://help.eclipse.org/Search:‘‘ThreewayComparison’’

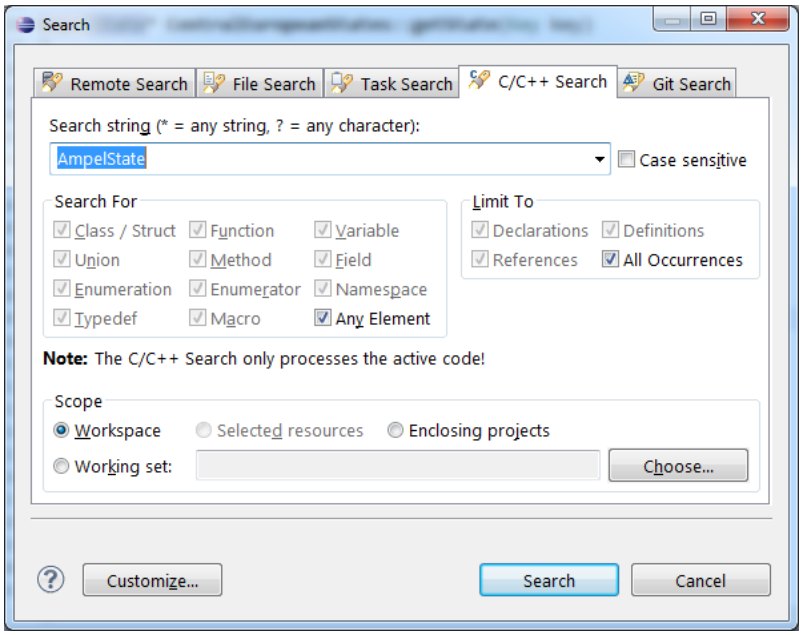

Abbildung 15: Der Dialog zur Datei übergreifenden Suche

Die Ergebnisse der Datei übergreifenden Suche werden in einer View wie in Abbildung 16 dargestellt. Im Editor werden die Textstellen markiert und die Zeilen mit einem Pfeil am linken Rand markiert.

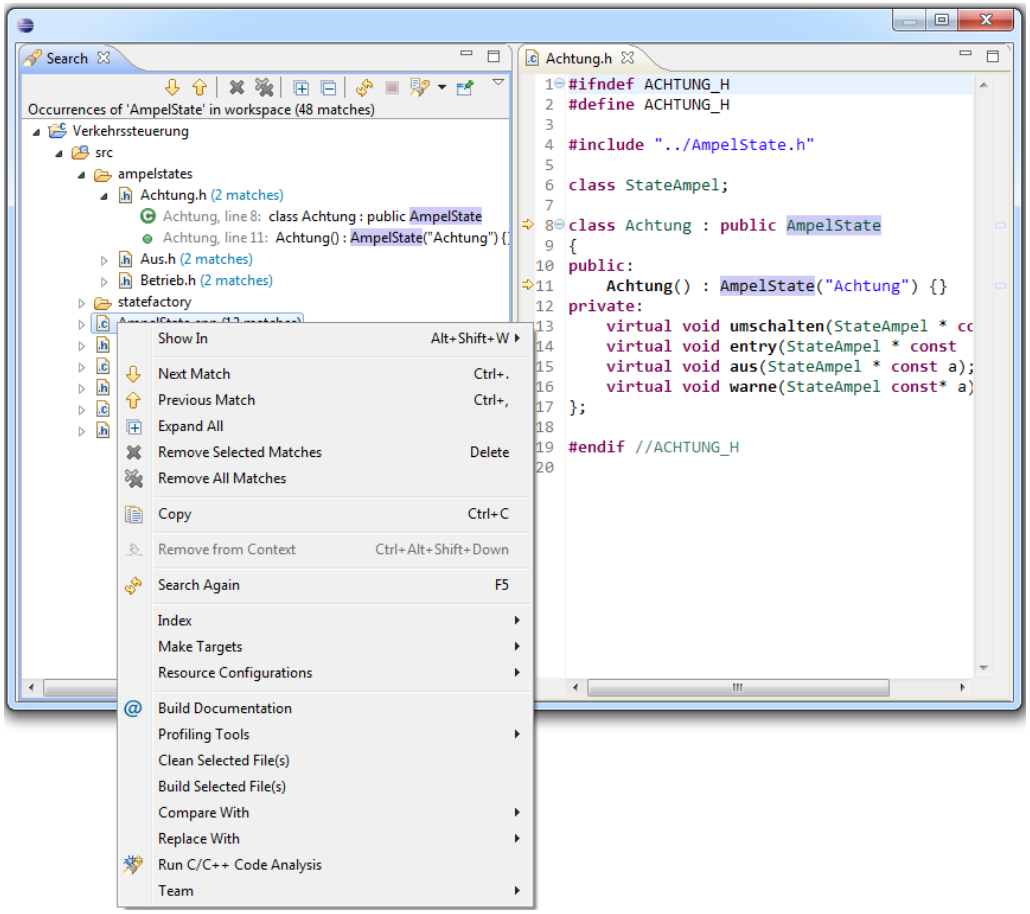

Abbildung 16: Suchergebnisse, ihr Kontextmenü und Editoren

Über die View kann direkt zu den Fundstellen im Editor navigiert werden. Nicht erwünschte Ergebnisse können aus der View mit *Delete* entfernt werden.

9 Allgemeine Views von Eclipse

9.1 Navigation Views

Die einfachste Navigation View ist über *Window.Show View.Navigator* zu erreichen. Sie zeigt die Hierarchie der Verzeichnisse im Workspace und deren vollständigen Inhalt. Der *Project Explorer* zeigt nur die projektspezifischen Inhalte. Er ist eine generelle Navigation View für Projekte verschiedener Art und ersetzt die C/C++ Projects View.

Bei Bedarf kann die View mit **F5** oder über das Kontextmenü *Refresh* aktualisiert werden. Das kann z.B. notwendig werden, wenn Dateien extern in das Verzeichnis kopiert werden.

In einer Navigation View können Resourcen mit einem **Doppelclick** oder über das Kontextmenü *open* einer Resource in einem Editor geöffnet werden.
Projekte, die von einander abhängig sind, können über das Kontextmenü eines Projekts gemeinsam geöffnet (*Open*) und andere Projekte geschlossen werden (*Close Unrelated Projects*). Beim *Open* von Projekten, die Abhängigkeiten zu anderen Projekten[28] definieren, erfolgt über einen Dialog eine Abfrage, die anderen Projekte ebenfalls zu öffnen.

Beachte: Leider werden die Abhängigkeiten, die in *<Projectname>.Properties.C/C++ General.Paths and Symbols.References* eingestellt sind, nicht berücksichtigt, sondern nur die Abhängigkeiten unter *<Projectname>.Properties.Project References* im selben Dialog.

Tipp: Die gesamte Performance von Eclipse wird besser, wenn nur wenige Projekte geöffnet sind, weil nur diese beim Indizieren (CDT) und in verschiedenen Views, z.B. der Problems View, berücksichtigt werden.

In den Navigation Views können Plugins den Pictogrammen für Files und Folder spezifische Verzierungen[29] hinzufügen. Der *Project Explorer* in Abbildung 13 auf Seite 25 zeigt das Projekt DemoEclipseSymbole mit einer roten Verzierung, des Features CDT, das einen Syntaxfehler signalisiert und einer braunen Verzierung, des Features Subclipse, das eine veränderte Resource signalisiert.

Über das Pulldown Menü einer Navigation View *Filters...* kann der Dialog in Abbildung 17 auf der nächsten Seite geöffnet werden. Die ausgewählten Elemente werden nicht angezeigt. Dieser Menüeintrag steht im Project Explorer leider nicht zur Verfügung. Working Sets[30] sind eine weitere Möglichkeit, den Inhalt der View zu Filtern.

[28] siehe Abbildung 45 auf Seite 54
[29] en Adornments
[30] siehe section 5.4 auf Seite 16

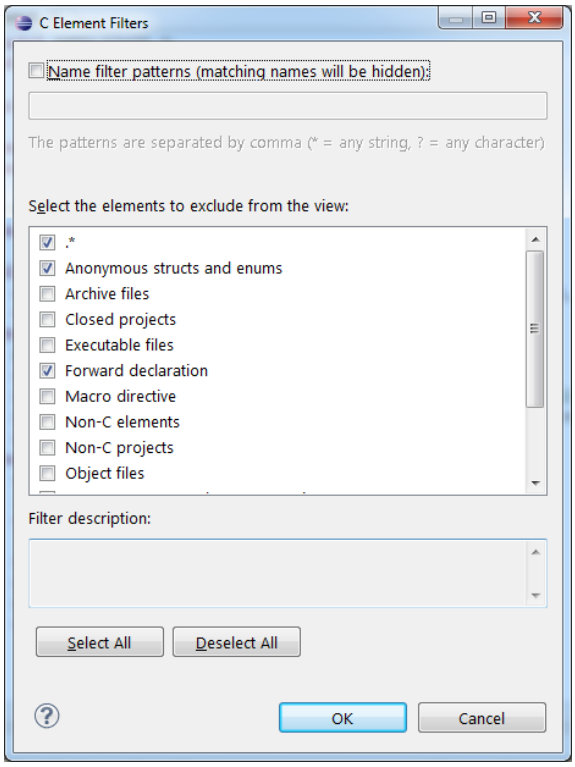

Abbildung 17: Filter Resources in einer Navigation View

9.2 Open Resource Dialog

Zum Öffnen einer Resource kann alternativ zur Navigation View der Open Resource Dialog in Abbildung 18 auf der nächsten Seite verwendet werden. Der Dialog kann über **Ctrl+Shift+R** geöffnet werden. Anstatt die Resource in einem Editor zu öffnen, kann sie auch über *Show In* einer anderen View angezeigt werden.

9.3 Tasks

Die Tasks View zeigt Aufgaben an, die im Sourcecode der geöffneten Projekte definiert sind. Sie kann über *Window.Show View.Tasks* geöffnet werden.

Eine erweiterte Aufgabenverwaltung steht mit dem plug-in **Mylyn** zur Verfügung.

Tasks können über zu definierende Schlüsselwörter, z.B. `Todo` oder `Fixme` direkt im Code als Kommentar beschrieben werden oder über das Kontextmenü am linken Rand des Editors erfasst werden.

Tasks die als Kommentare definiert sind, sind im Code sichtbar und werden bei der Versionierung mit berücksichtigt, Tasks die über das Kontextmenü erfasst wurden, werden nur als Anmerkungen am Rand des Editors angezeigt, sind nur im Eclipse Editor sichtbar und werden nur im Workspace verwaltet.

Für die Tasks, die über das Kontextmenü erfasst werden, können Prioritäten (*low, normal, high*) und der Fertigstellungsgrad über den Properties Dialog aus Abbil-

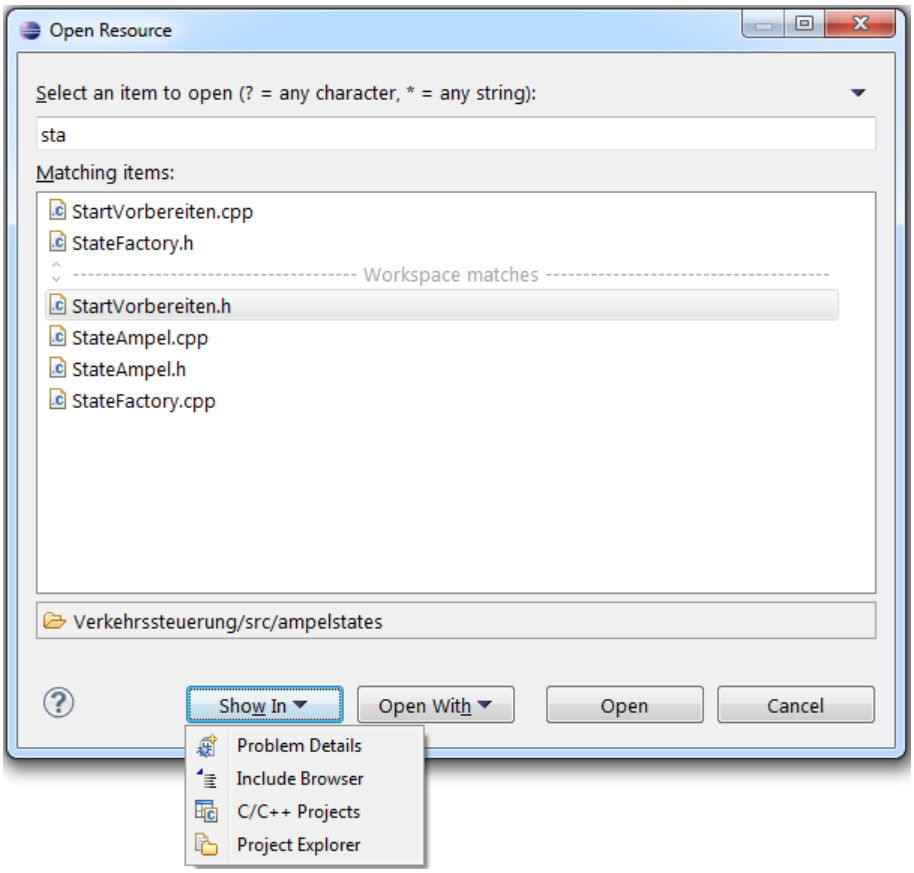

Abbildung 18: Der Open Resource Dialog

dung 21 auf Seite 34 eines Tasks festgelegt werden. Der Dialog wird über das Kontextmenü einer Task in der Task View geöffnet. *Priority:* und *Completed* können nur für Tasks bearbeitet werden, die über das Kontextmenü erfasst wurden, ansonsten werden sie gegrayed[31] dargestellt.
Die Abbildung 19 auf der nächsten Seite zeigt die Task View und die korrespondierenden Stellen im Code. Die Zeilennummern werden in der Task View angezeigt. Im Editor sind die Tasks am linken Rand durch eine Checkbox mit einem Häckchen erkennbar. Über die View kann direkt zu einem Task im Code navigiert werden.
Die Verwaltung der Tags erfolgt über *Window.Preferences* wie in Abbildung 20 auf der nächsten Seite gezeigt.
Über das Pulldown Menü *Configure Contents...* der Task View kann der komplexe Dialog aus Abbildung 22 auf Seite 34 zur Filterung des Inhalts der Task View geöffnet werden. Im selben Menü kann der Dialog zur Auswahl der verfügbaren Spalten über *Configure Columns...* geöffnet werden.

[31] als nicht verfügbar

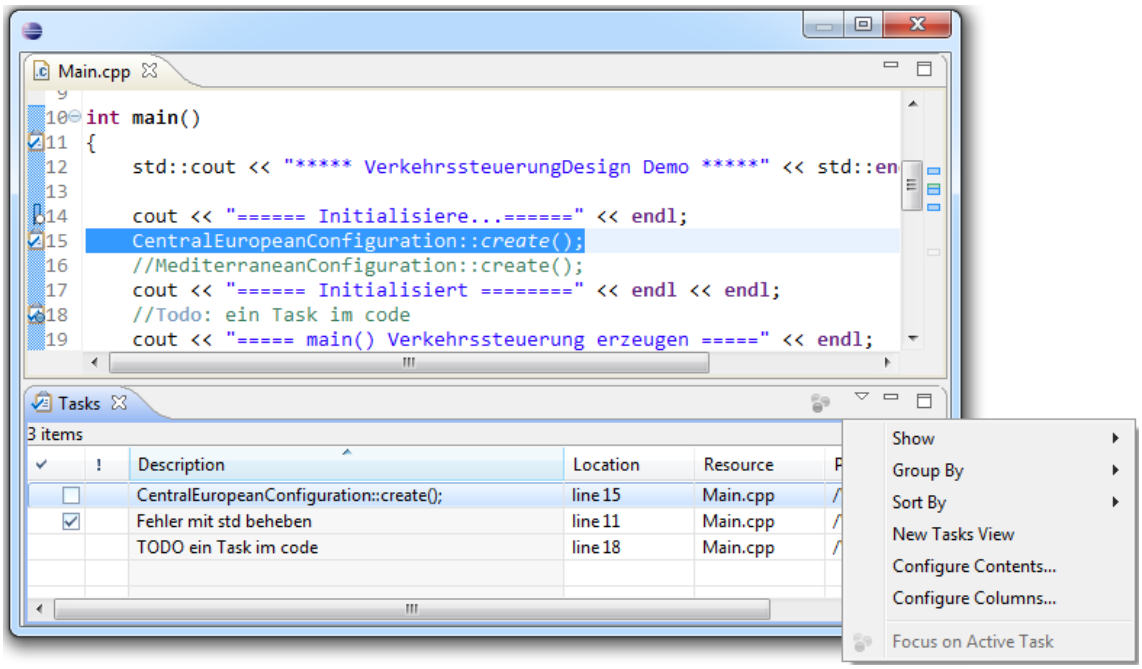

Abbildung 19: Die TaskView und Tasks im Editor

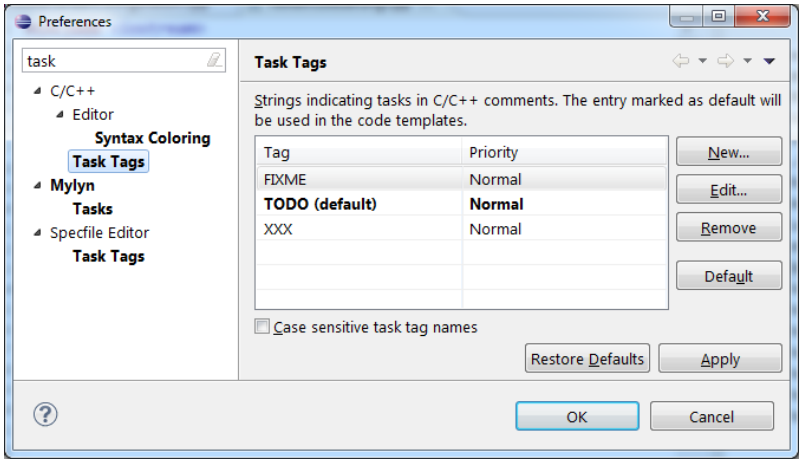

Abbildung 20: Definition von Task Tags

9.4 Console

Die Console zeigt die Ausgaben des Compilers, des ausgeführten Programms und die Ausgaben anderer Plugins, wie z.B. der Versionsverwaltung in Abbildung 23 auf Seite 35, in eigenen Ausgabefenstern. Die Console mit der Ausgabe des Compilers wird jeweils für das im Project Explorer ausgewählte Projekt angezeigt. Über die Toolbar der Console kann zwischen den verschiedenen Ausgabenfenstern umgeschaltet und neue Consolen Views geöffnet werden. Der Name der Console (*SVN* in Abbildung 23 auf Seite 35) wird links oben in der View angezeigt.

Ist eine Console mit einem Programm verknüpft, das Eingaben über den Standard Input erwartet (z.B. der GNU Debugger), können diese in der Console eingegeben werden.

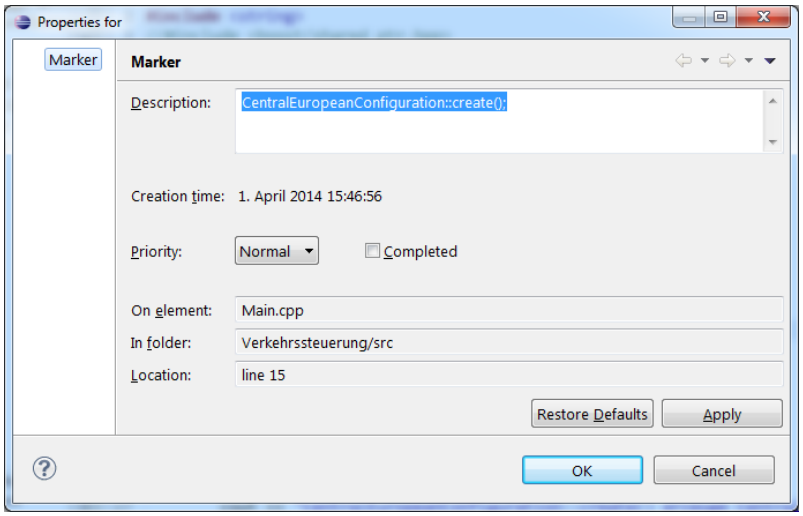

Abbildung 21: Der Properties Dialog einer Task

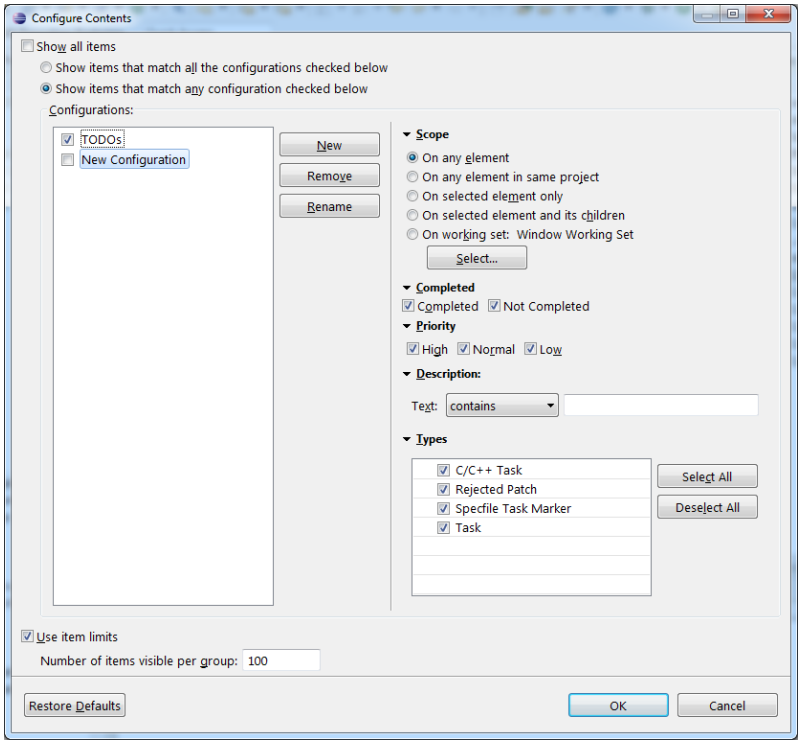

Abbildung 22: Der Filter Dialog der Task View

9.5 Problems

Die Problems View zeigt Probleme verschiedenster Art:

- Warnings, wie in Abbildung 24 auf der nächsten Seite
- Compiler Errors
- Linker Errors
- Fehler in der Build Configuration, z.B. unbekannte Optionen
- usw.

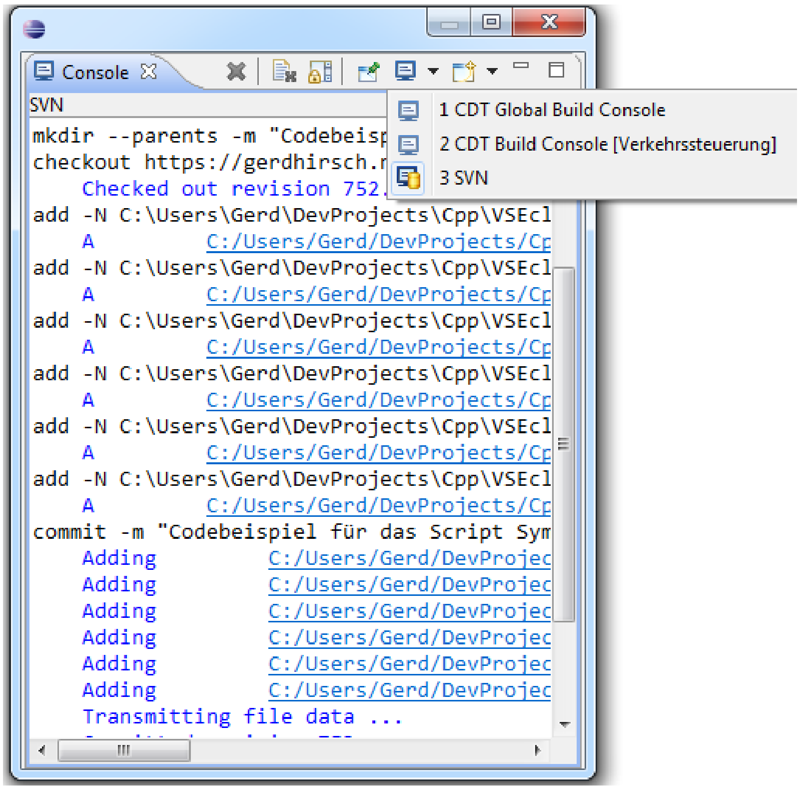

Abbildung 23: Die Console View

Aus der View kann direkt an die Stelle im Editor navigiert werden, wenn es sich um ein Problem im Sourcecode handelt. Über das Pulldown Menü können ver-

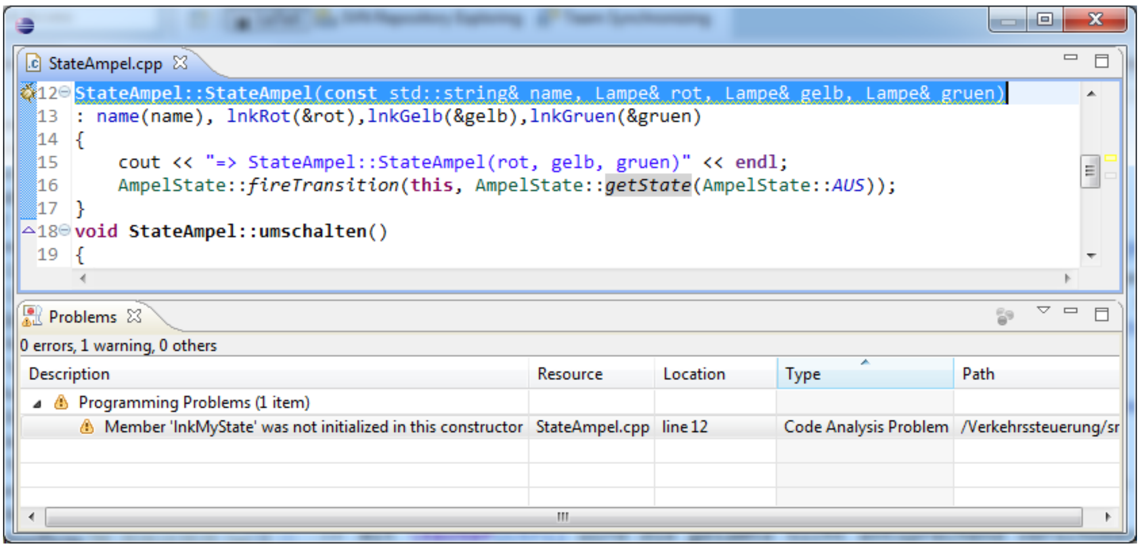

Abbildung 24: Die Problems View

schiedene Filter für die View eingestellt werden.

9.6 Properties

Die Properties View[32] zeigt die Eigenschaften eines in einer Navigation View ausgewählten Elements an.

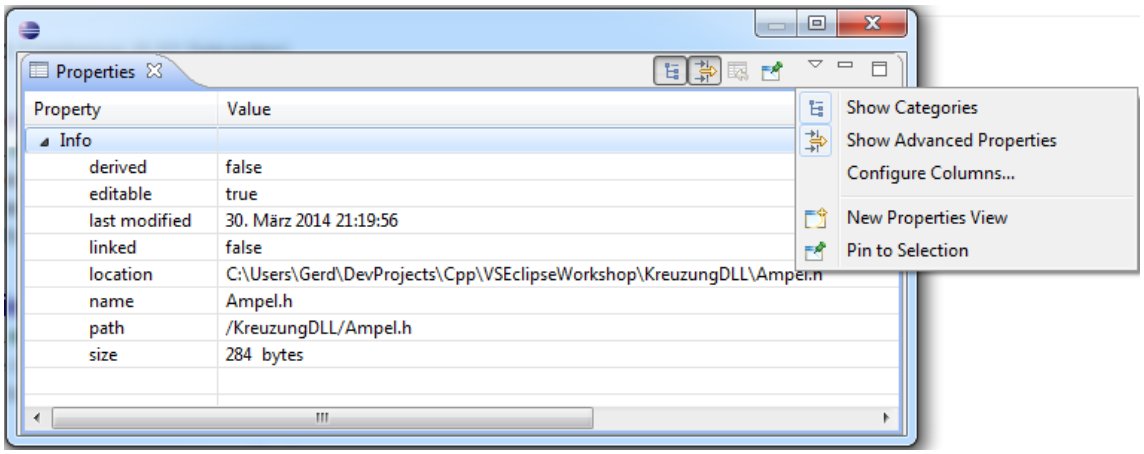

Abbildung 25: Die Properties View, Eigenschaften der Elemente

In Abbildung 25 werden die Eigenschaften der Datei `Ampel.h` angezeigt. Die Eigenschaften haben folgende Bedeutung: (so weit sie nicht selbsterklärend sind)

- *derived*: das Element ist aus anderen erzeugt
- *linked*: das Element ist eine Referenz auf eine Datei ausserhalb des Workspace

Ausnahme: Die View muss mit **F5** aktualisiert werden, nachdem eine Änderung an dem ausgewählten Element vorgenommen wurde.

Mit *Pin this property view to the current selection* kann die View fest mit dem ausgewählten Element verknüpft werden.
Über das Pulldown Menü oder über die Toolbar der View können weitere Einstellungen vorgenommen und eine weitere View erzeugt werden.

[32] http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Freference%2Fcdt_u_properties_view.htm

Teil III

CDT

10 Das C/C++ Development Toolkit (CDT)

10.1 Ein erstes Projekt, Überblick

In diesem Kapitel soll anhand eines einfachen *managed build* C++ Projekts ein erster Überblick über die Entwicklungsumgebung Eclipse/CDT gegeben und die Schritte Projekt erzeugen, übersetzen, ausführen und debuggen einmal durchgeführt werden. Die Details werden in den nachfolgenden Kapiteln beschrieben.

10.1.1 Projekt erzeugen

Eclipse starten und ein leeres Verzeichnis als Workspace auswählen. Den Begrüßungsdialog von Eclipse schließen und in die C/C++ Perspective wechseln. Im Project Explorer mit der rechten Maustaste das Kontextmenü öffnen. Die Oberfläche sollte ungefähr wie in Abbildung 26 aussehen. Den Menüpunkt *New.C++ Project* auswählen.

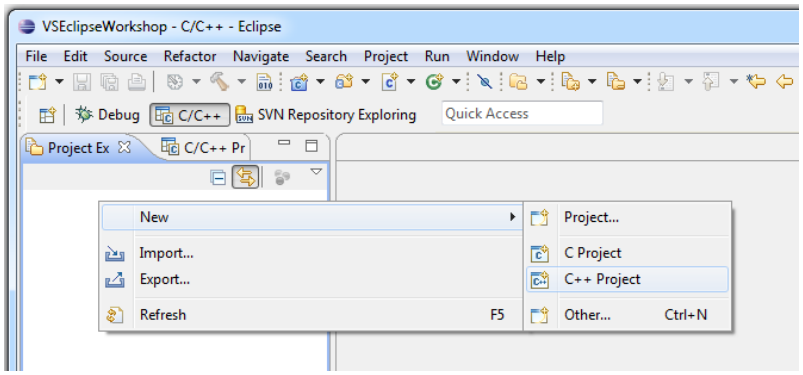

Abbildung 26: Das Kontext Menü New

Darauf öffnet sich der Dialog in Abbildung 27 auf der nächsten Seite.
In diesem Dialog die passende Toolchain auswählen, einen Projektnamen vergeben und *Project type.Executable.Hello World C++ Project* auswählen. Der Name der Toolchain ist je nach Betriebssystem und verfügbaren Toolchains verschieden. Die Toolchain für Window Systeme ist entweder MinGW GCC oder *Cygwin GCC* für die GNU Compiler Collection und auf Unix Systemen entsprechend *Linux GCC* oder ähnlich.
Mit *next >* kommt man zu dem linken Dialog in Abbildung 28 auf der nächsten Seite. Dort die entsprechenden Eingaben machen und mit *next >* zum rechten Dialog navigieren. Die vorgeschlagenen Einstellungen so belassen und mit *Finish* das Projekt erzeugen.

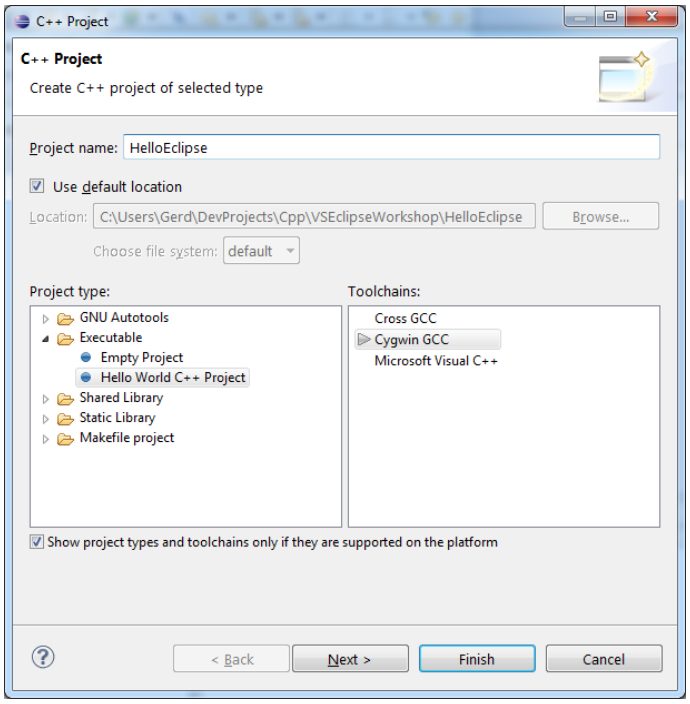

Abbildung 27: Der Dialog C++ Project 1

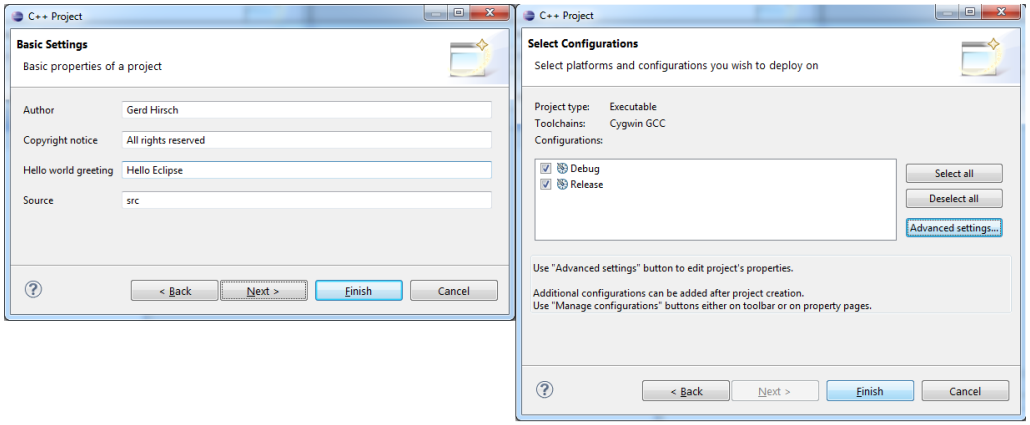

Abbildung 28: Der Dialog C++ Project 2 + 3

10.1.2 Code editieren

Der erzeugte Sourcecode in Abbildung 33 auf Seite 41 ist übersetzbar und ausführbar, eine Änderung ist für diese Einführung nicht notwendig.

10.1.3 Übersetzen/Build

Im Project Explorer das Kontextmenü des Projekts wie in Abbildung 29 auf der nächsten Seite öffnen und mit *Build Project* das Projekt übersetzen. Alternativ über die Toolbar in Abbildung 31 auf Seite 40 rechts oberhalb von *Run* das Hammer Pictogramm auswählen[33].

[33] siehe auch section 10.8.4 auf Seite 70

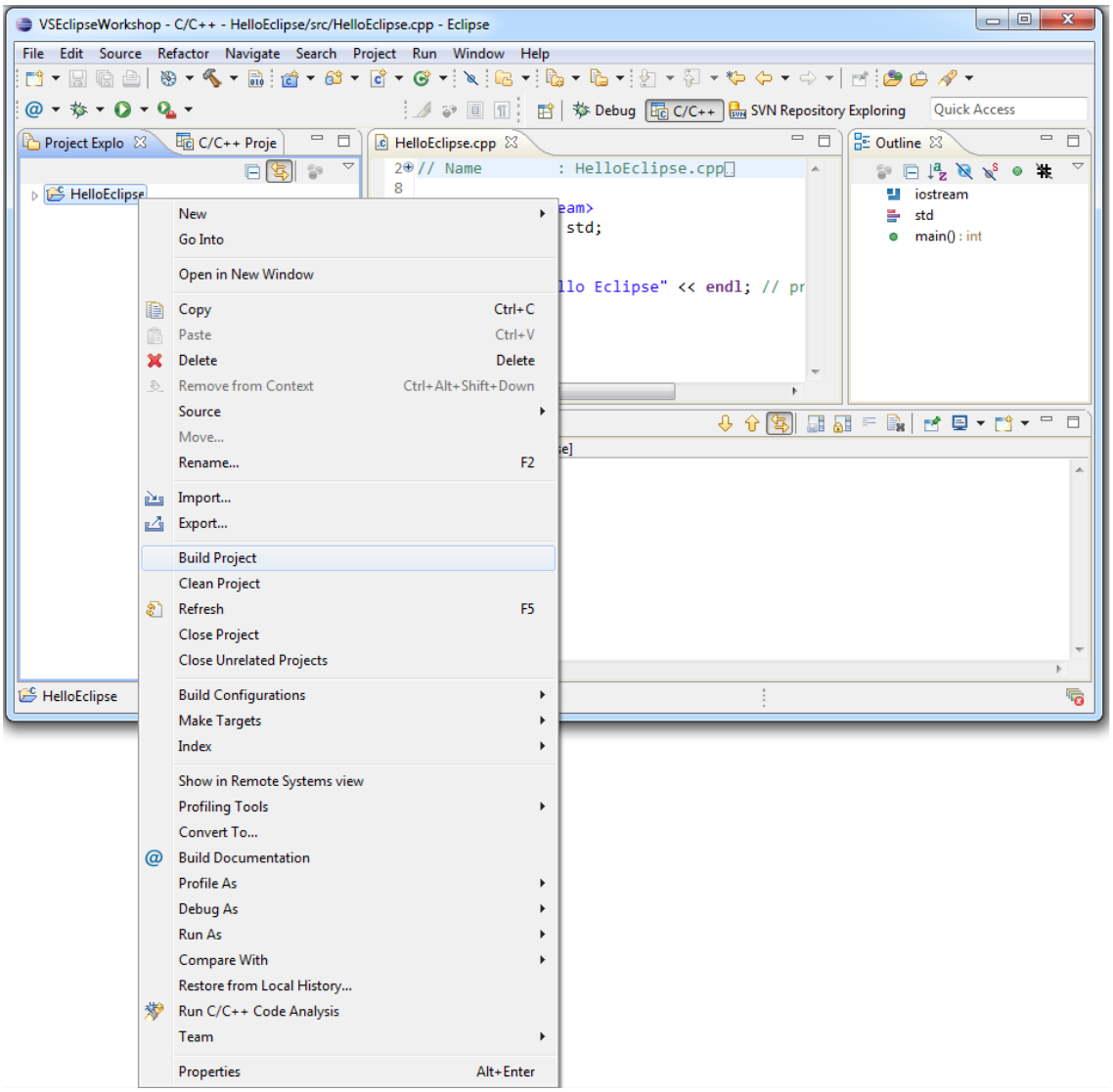

Abbildung 29: Das Kontextmenü C++ Project im Project Explorer

Die Ausgabe des Compilers in der Console ist in Abbildung 30 auf der nächsten Seite abgebildet.

10.1.4 Ausführen

Im Project Explorer das Kontextmenü des Projekts wie in Abbildung 29 öffnen und mit *Run As.Local C++ Application* das Projekt ausführen. Alternativ über die Toolbar *Run* wie in Abbildung 31 auf der nächsten Seite gezeigt.

10.1.5 Debuggen

Im Project Explorer das Kontextmenü des Projekts wie in Abbildung 29 öffnen und mit *Debug As.Local C++ Application* das Projekt im Debugger ausführen. Alternativ über die Toolbar in Abbildung 31 auf der nächsten Seite links neben

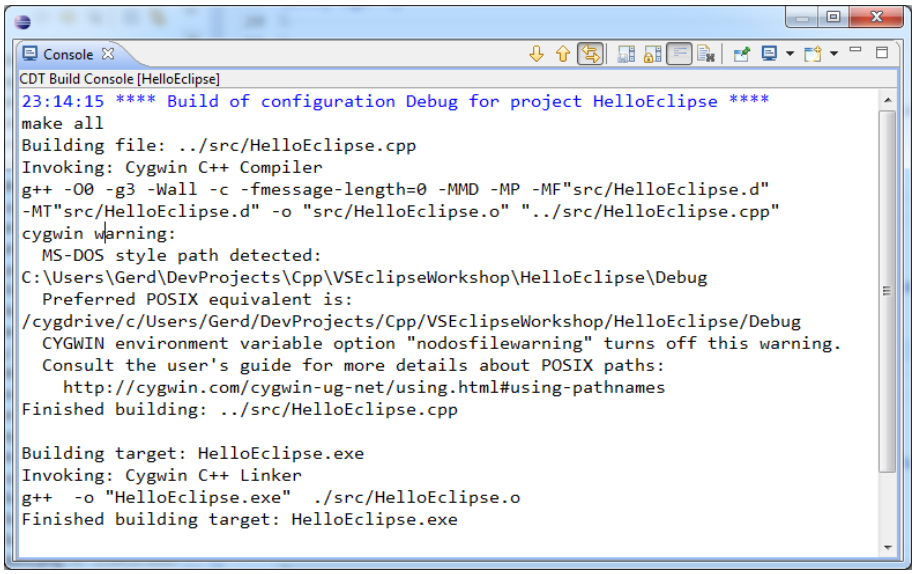

Abbildung 30: Die Compiler Ausgaben in der Console

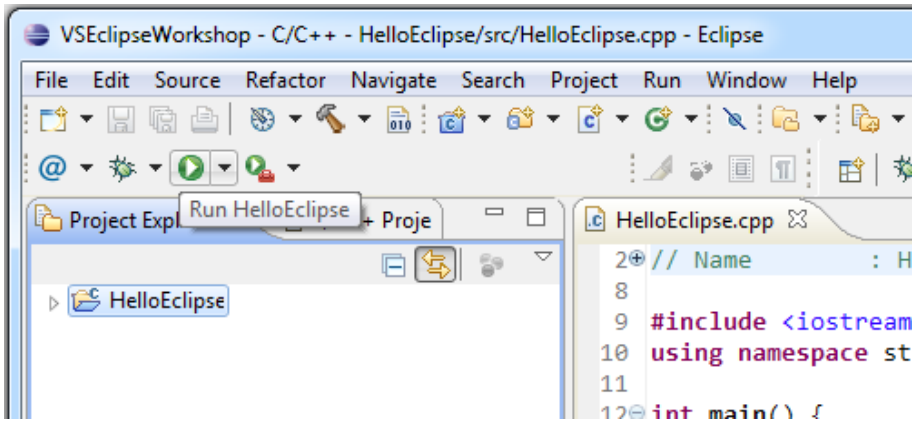

Abbildung 31: Das Programm über die Toolbar ausführen

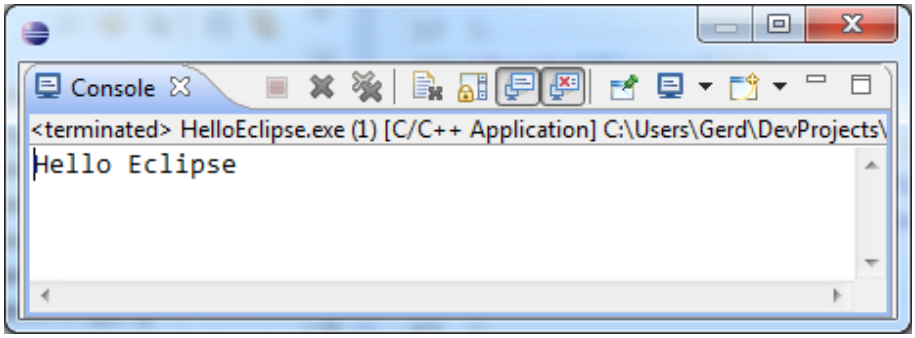

Abbildung 32: Die Ausgaben des Programms in der Console

Run das Bug Pictogramm auswählen. Den darauf folgenden Abfragedialog, zum Wechseln in die Debug Perspective die in Abbildung 33 auf der nächsten Seite abgebildet ist, bestätigen. Das Programm steht vor der ersten Anweisung von `main()`. Das wird im Editor durch die Hervorhebung der Zeile und einen kleinen blauen Pfeil am linken Rand kenntlich gemacht. Der Thread in der Debug View wird als *Suspended:Breakpoint* angezeigt. Mit dem Toolbar Icon *Resume* oder

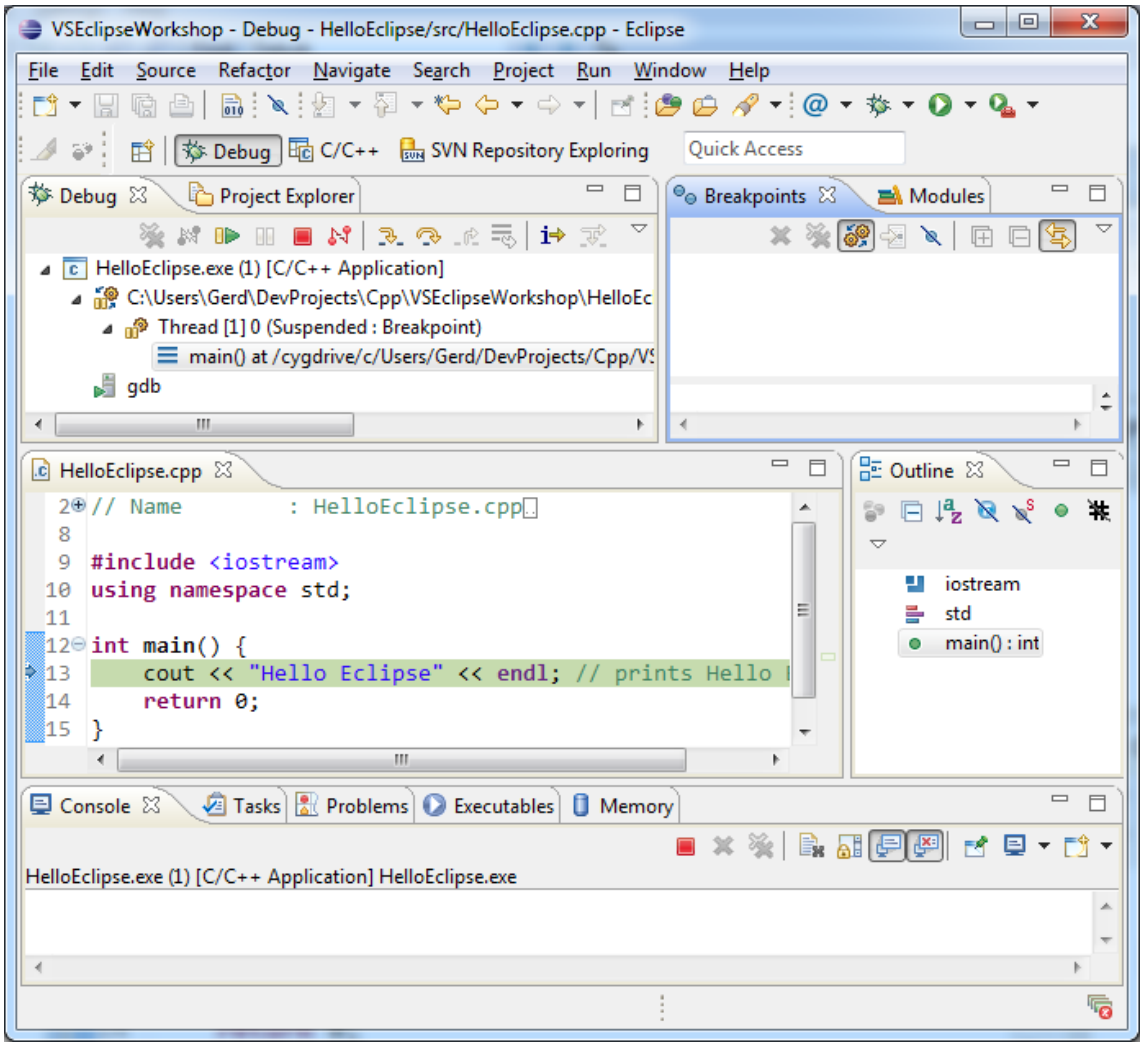

Abbildung 33: Die DebugPerspective

F8 kann der Programmablauf fortgesetzt werden. Danach wird der Thread als *<terminated, exit value: 0>* angezeigt.

10.2 Das Feature CDT

Für das CDT werden folgende Plugins benötigt:

- eclipse
- CDT C/C++ Development Tools, der Kern von CDT ohne Compiler
- CDT GNU Toolchain Build/Debug Support (Windows: Cygwin/MinGW für Posix)
- Weitere unterstützte Compiler
  - IBM XL C/C++
  - MS Visual Studio
  - Unified Parallel C
  - weitere

Weitere Plugins in dem Zusammenhang sind

- CDT SDK Erweiterungspunkte für die Plugin Entwicklung (z.B. Refactoring)
- CDT Utilities

10.3 Index View & C++ Parsen

Um die Metainformationen über das in C++ beschriebene System zu ermitteln, muss der Sourcecode analysiert werden. Diese Aufgabe übernimmt der *Scanner Configuration Builder* und erzeugt daraus einen abstract syntax tree (AST)[34]. Um schnell auf diese Informationen zugreifen und sie auswerten zu können, wird daraus eine Indexdatenbank erstellt, die ständig aktuell gehalten werden muss. Die Einstellungen für den Indexer sind in Abbildung 34 auf der nächsten Seite dargestellt. Auf dieser Basis arbeiten die vielen kleinen Helferlein der CDT, wie Outline View, Code Assistance im Editor, Type Hierarchie View, Include Hierarchie View, usw.

Bei manchen Änderungen der Projekteinstellungen wird eine Aktuallisierung des gesamten Index notwendig. Dieser kann explizit über das Project Kontextmenu *Index* neu aufgebaut werden. Das kann auch notwendig werden, wenn z.B. eine Anzeige in einer dieser Views nicht richtig funktioniert.

Mit der C/C++ Index View aus Abbildung 35 auf Seite 44 kann direkt Einsicht in den Index genommen werden. In dieser View werden alle indizierten Projekte und alle Symbole die im Index enthalten sind, angezeigt.

Die View kann über *Window.Show View.C/C++ Index* geöffnet werden. Mit einem Click auf ein Symbol in der View wird direkt an die korrespondierende Stelle im Editor navigiert.

[34] Details: https://wiki.eclipse.org/images/e/ec/McMaster_2012_invited_talk_cdt.pdf

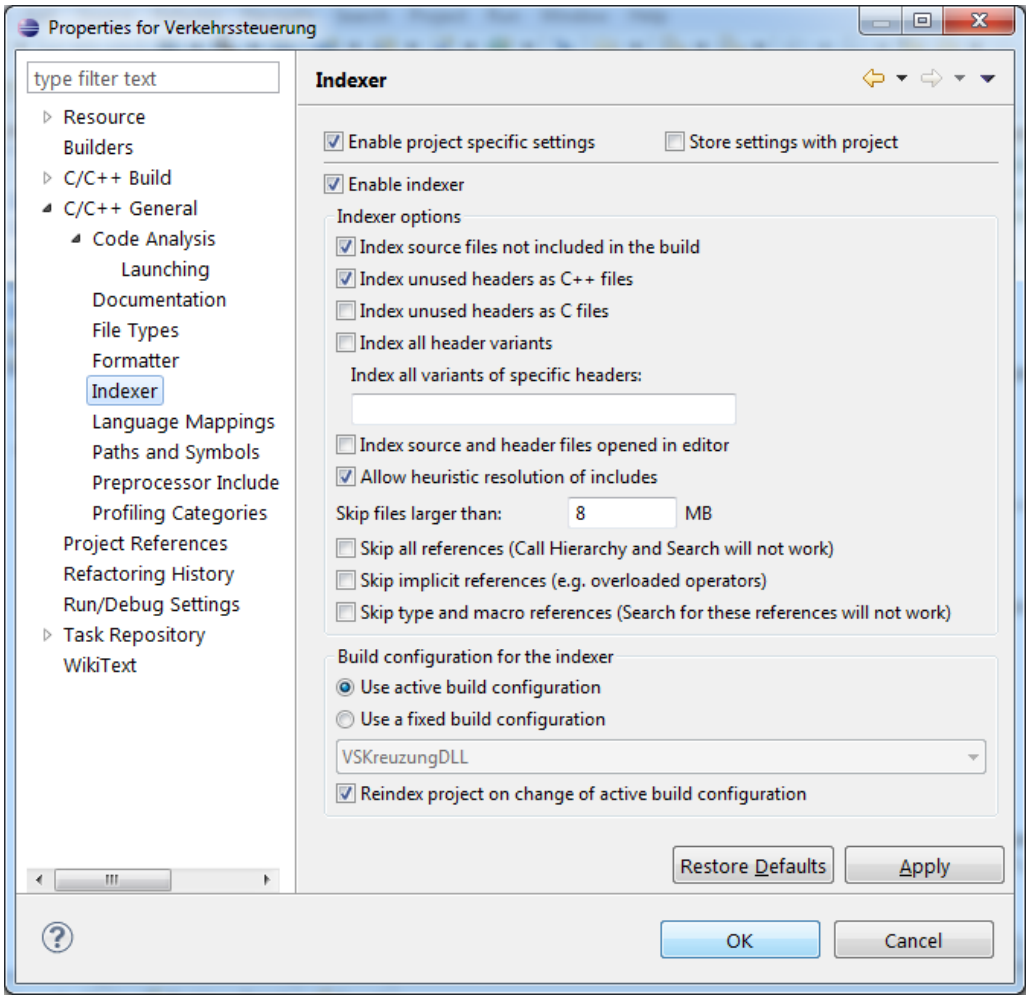

Abbildung 34: Die Properties für den Indexer

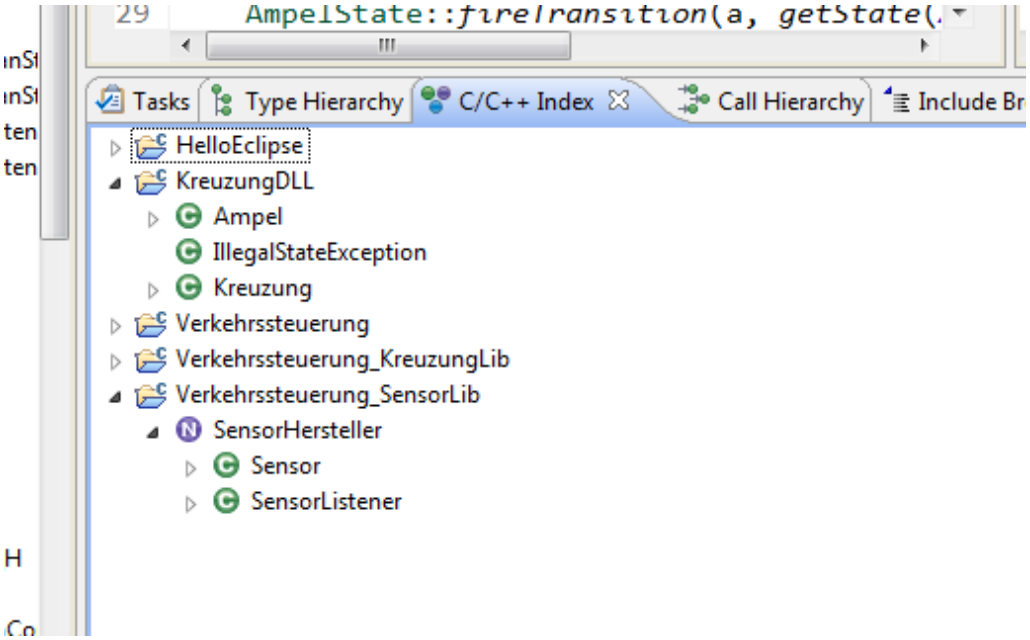

Abbildung 35: Die C++ Index View

10.4 Die Perspectives

Die wichtigsten Perspectives im Zusammenhang mit der Entwicklung von C++ Programmen sind die

- C/C++ Perspective, zur Bearbeitung von Quellcode
- Debug Perspective
- Versionsverwaltung, in Abhängigkeit des gewählten Versionsverwaltungssystems und des dazugehörigen Plugins, z.B. SVN Repository Exploring

10.5 Die wichtigsten Views der C/C++ Perspective

Die C/C++ Perspective stellt die wichtigsten Views und einen C/C++ Editor zur Verfügung.

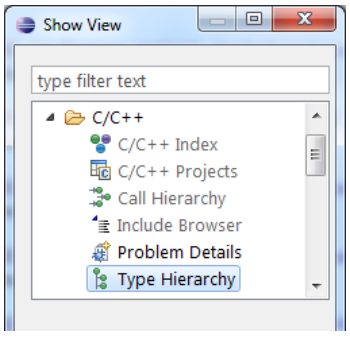

Abbildung 36: Die C++ Views

Weitere aus Abbildung 36 können über *Window.Show View* hinzugefügt werden.

10.5.1 C/C++ Editor

Die allgemeine Bedienung des Editors ist in section 8.1.3 auf Seite 26 beschrieben.

Für C/C++ stehen weitere spezifische Funktionen zur Verfügung:

- Breakpoints festlegen am linken Rand
- Markierung syntaktischer Fehler am linken Rand
- Hover über ein Symbol zeigt die Definition
- Makro Expansion **Ctrl+Shift+0** <-> **Ctrl+=**
  oder Kontextmenü *Explore Makro Expansion*
- u.v.m.

10.5.2 C/C++ Project Navigator

Diese View wird nicht länger unterstützt. Arbeiten Sie besser mit dem Project Navigator aus section 9.1 auf Seite 30.

Der C/C++ Project Navigator und der Eclipse Project Navigator stellen fast dieselbe Funktionalität zur Verfügung. Der Eclipse Project Navigator ermöglicht das Öffnen und Schließen von unrelated Projects[35], das unterstützt der C/C++ Project Navigator leider nicht. Über das Kontextmenü eines Projekts können verschiedene projektspezifische Aktivitäten gestartet werden.
Die View wird über *Winodw.Show View.C/C++ Projects* geöffnet.

10.5.3 Open Element Dialog

Mit dem in Abbildung 37 auf der nächsten Seite gezeigten Dialog kann ein nicht lokales Symbol im Workspace gesucht und angezeigt werden (die lokale Variable `verkehrssteuerung`) in `main()` wird leider nicht angezeigt). Der Dialog kann über das Menü *Navigate.Open Element...* oder mit dem Shortcut **Crtl+Shift+T** geöffnet werden[36].

10.5.4 Outline View

Die Outline View zeigt alle Symbole aus dem File, das im aktuellen Editor bearbeitet wird. Die Outline View kann über das Menü *Window.Show View.Outline* geöffnet werden.

Leider werden in der aktuellen Version bestimmte Konstrukte von C++11/14 nicht oder nicht korrekt angezeigt.

[35] siehe auch Abbildung 45 auf Seite 54
[36] siehe auch section 9.2 auf Seite 31

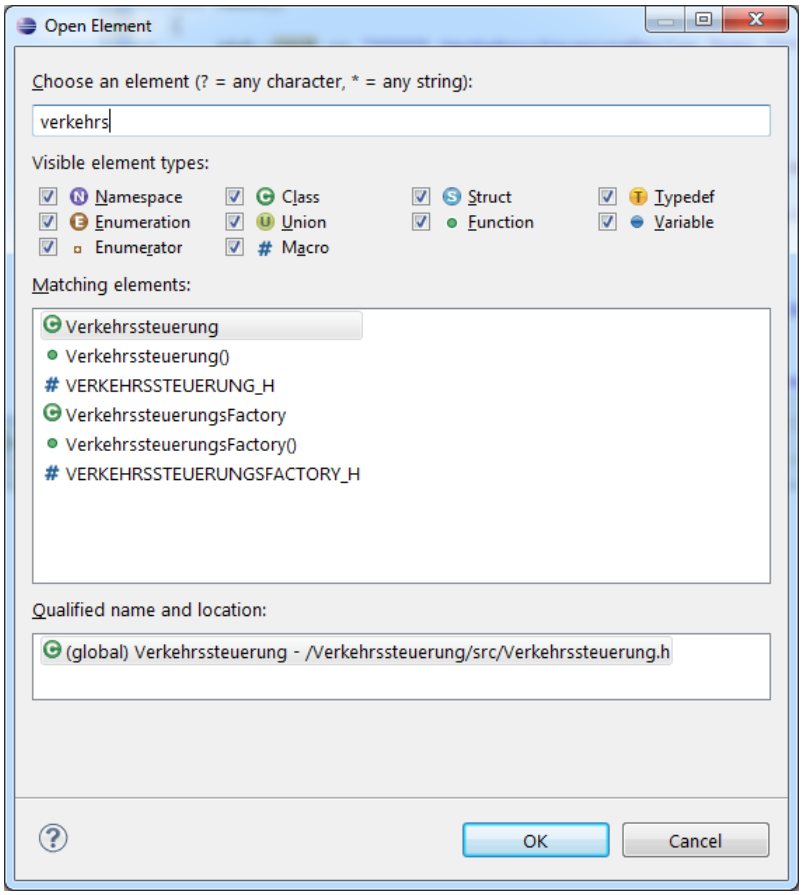

Abbildung 37: Der Open Element Dialog

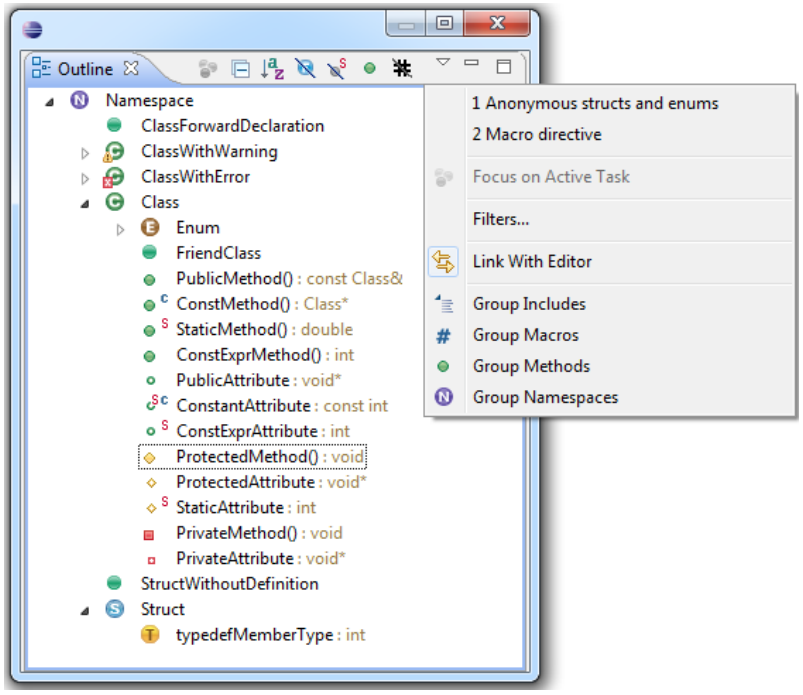

Abbildung 38: Die Outline View und die Pictogramme von CDT 1

Ein Mouseclick in der Outline View auf ein Symbol setzt den Cursor auf das korrespondierende Symbol im Editor. Aus der Outline View können Symbole via

Drag´n Drop in anderen Views angezeigt werden, z.B. Funktionen in der Call Hierarchie View. Alternativ können die Symbole mit dem Kontextmenü in anderen Views angezeigt werden, z.B. Header in der Include Hierarchie View oder Klassen/Strukturen in der Type Hierarchie View.

In Abbildung 38 auf der vorherigen Seite ist eine Outline View abgebildet, die die korrespondierenden C++ Konstrukte aus Listing 3 zeigt. Die Pictogramme vor den Symbolen werden in anderen CDT Views ebenfalls verwendet.

Über die Toolbar der View können verschiedene Filter aktiviert werden. Von Links nach Rechts:

- Focus on active Task
- Collaps all
- Alphabetisch sortieren / default: Reihenfolge im File
- Hide Fields
- Hide Static Members
- Hide Non-Public Members
- Hide Inactive Elements

Über das Pulldown Menü können weitere Filter und Einstellungen der View vorgenommen werden.

Die Formen und Farben der Pictogramme signalisieren den Typ und die Access Specifier: Rund ausgefüllt und Grün für **public** Method, Raute unausgefüllt und Gelb für **protected** Attribute, usw. Dazu kommen weitere Verzierungen, die Modifier wie **static** oder Warnings und Errors oder Inaktive Elemente kennzeichnen. Die Zugehörigkeit eines Symbols zu einem übergeordneten Element wird durch Einrückungen dargestellt.

Listing 3: Die Outline View und die C++ Konstrukte 1

```
1  namespace Namespace{
2  class ClassForwardDeclaration;
3  class ClassWithWarning{
4     virtual void VirtualMethod(){}
5  };
6  class ClassWithError{
7     in Attribute;
8  };
9  class Class{
10 public:
11    enum Enum{ EnumCONSTANT};
12    friend class FriendClass;
13
14    Class const& PublicMethod();
15    Class* ConstMethod() const;
16    static double StaticMethod();
17    constexpr int ConstExprMethod(){ return 42; }
18    void* PublicAttribute;
```

```
19    static const int ConstantAttribute;
20    static constexpr int ConstExprAttribute=42;
21 protected:
22    void ProtectedMethod();
23    void* ProtectedAttribute;
24    static int StaticAttribute;
25 private:
26    void PrivateMethod();
27    void* PrivateAttribute;
28 };
29 struct StructWithoutDefinition;
30 struct Struct{
31    typedef int typedefMemberType;
32 };
33 } // end Namespace
```

Das Warning für die Klasse `ClassWithWarning` resultiert aus einer vorhandenen `virtual` Method und einem fehlenden `virtual` Destruktor. Der Error für die Klasse `ClassWithError` resultiert aus dem Syntax Fehler `in` für **`int`**.

Die Abbildung 39 auf der nächsten Seite zeigt weitere Pictogramme. Der dazugehörige C++ Code ist in Listing 4 abgebildet. Der Header `Include.h` ist in Listing 5 auf der nächsten Seite abgebildet.

Listing 4: Die Outline View und die C++ Konstrukte 2

```
1  #define Makro
2  #ifdef Makro
3  #include "IncludeNotFound.h"
4  #include "Include.h"
5  #else
6  #include "InactiveInclude.h"
7  #define InactiveMacro
8  #endif
9
10 using namespace UsingDirective;
11 using UsingDirective::UsingDeclaration;
12
13 void globalFunctionDeklaration();
14 void globalFunctionDefinition(){}
15
16 static void localFunctionDeklaration();
17 static void localFunctionDefinition(){}
18
19 int GlobalObject;
20 const int GLOBALCONST = 42;
21
22 template<class T>
23 class Template{
24 public:
25    using usingMemberType=T; //C++11 nicht angezeigt
26    typedef T typedefMemberType;
```

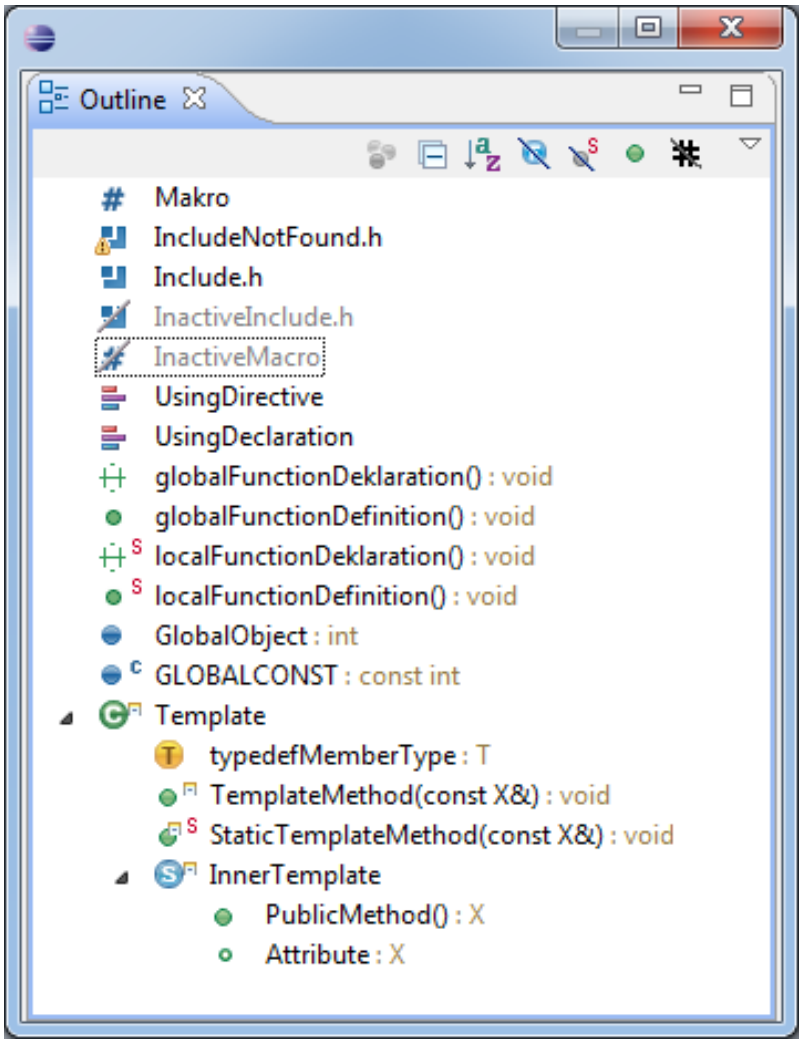

Abbildung 39: Die Outline View und die Pictogramme von CDT 2

```
27    template<class X>
28    void TemplateMethod(X const& x);
29    template<class X>
30    static void StaticTemplateMethod(X const& x);
31    template<class X>
32       struct InnerTemplate{
33          X PublicMethod();
34          X Attribute;
35       };
36 };
```

Listing 5: Die Outline View und die C++ Konstrukte 3

```
1 namespace UsingDirective{
2 void UsingDeclaration(){}
3 class FriendClass{};
4 }
```

10.5.5 Call Hierarchie

Die Call Hierarchie zeigt entweder die Funktionen, die von der ausgewählten Funktion gerufen werden oder die Funktionen, die die ausgewählte aufrufen. Die Ansicht kann über die Toolbar umgeschaltet werden.

Die Call Hierarchie View kann über

- *Window.Show View.Call Hierarchie*
- das Kontextmenü einer Funktion aus dem Editor (Shortcut: **Ctrl+Alt+H**) und aus verschiedenen Views

geöffnet werden. Eine Funktion kann via **Drag´n Drop** in die View gezogen werden. Über die View kann direkt zu der korrespondierenden Stelle im Editor navigiert werden. In Abbildung 40 wird die Funktion `externalSignal` des Sensors und die aufgerufenen Funktionen gezeigt. Die Operation `SensorListener::trigger(..)` ist pure virtual, der Aufruf dieser Operation wird in der Call Hierarchie View entsprechend angezeigt.

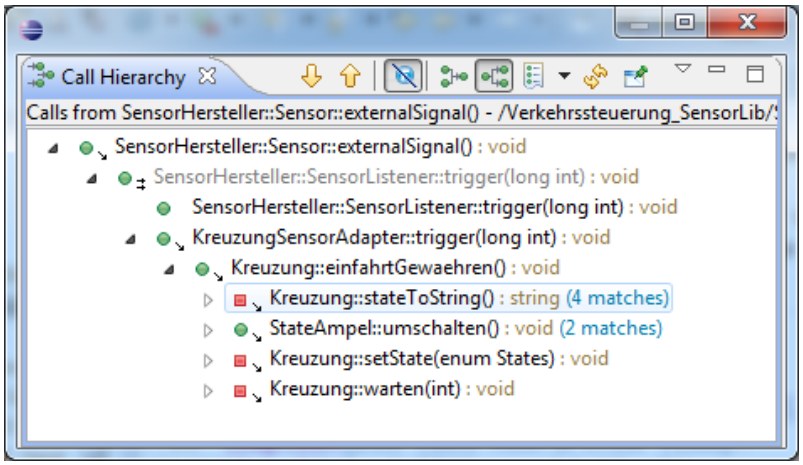

Abbildung 40: Die Call Hierarchie View aufgerufene Funktionen

Die Abbildung 41 auf der nächsten Seite zeigt dieselbe Funktion und von welchen Funktionen sie gerufen wird.

Über die Toolbar der View können verschiedene Filter aktiviert und über das Pulldown Menü Working Sets ausgewählt und weitere Einstellungen vorgenommen werden. Die View muss gegebenenfalls aktualisiert werden, damit die Informationen korrekt angezeigt werden.

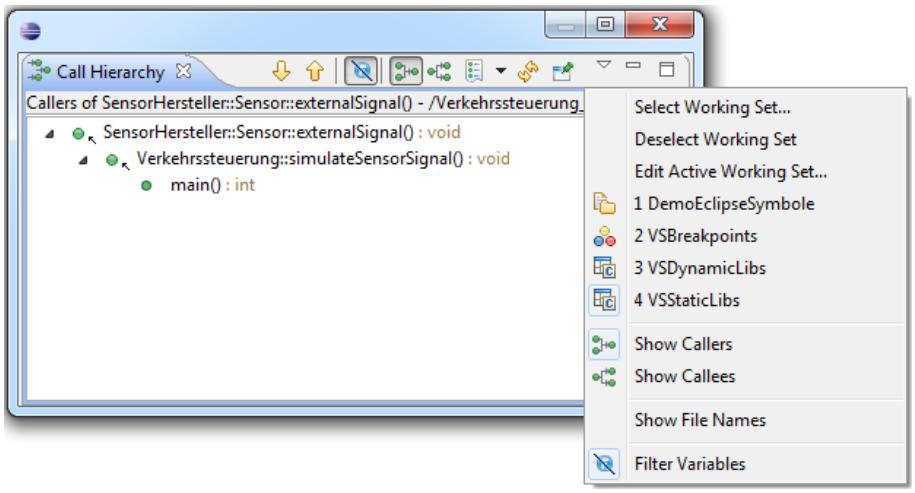

Abbildung 41: Die Call Hierarchie View aufrufende Funktionen

10.5.6 Include Browser

Die Include Browser View in Abbildung 42 zeigt die Files, die den ausgewählten Header `Kreuzung.h` includieren und in Abbildung 43 auf der nächsten Seite die Hierarchie der included Files. Die Ansicht kann über die Toolbar der View umgeschaltet werden.

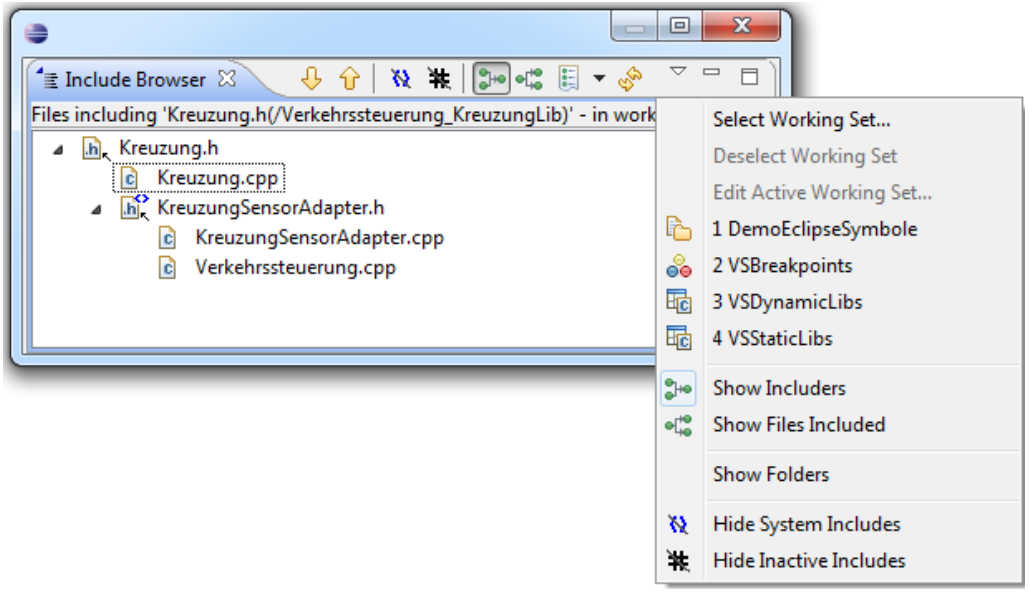

Abbildung 42: Die Include Hierarchie View Includers

Die Include Browser View kann über

- *Window.Show View*
- das Kontextmenü in der Outline View für einen Header
- das Kontextmenü im Editor *Show In.Include Browser*
- ... weitere Möglichkeiten

geöffnet werden. Bei geöffneter View kann via **Drag´n Drop** ein File in der View angezeigt werden.
Über die View kann direkt zu der korrespondierenden Stelle im Editor navigiert werden.

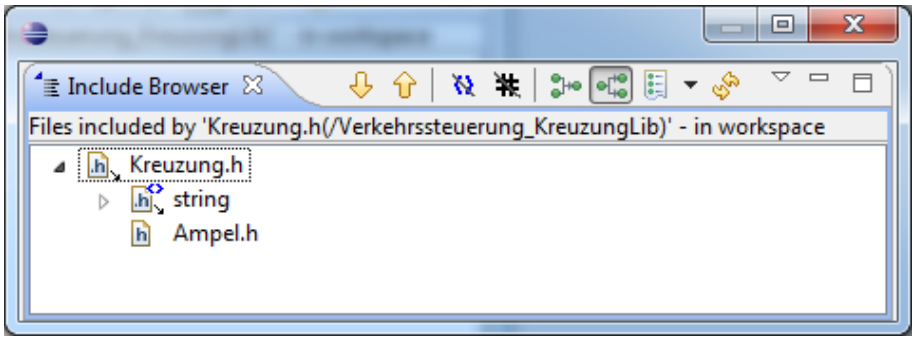

Abbildung 43: Die Include Hierarchie View Included

Über die Toolbar der View können verschiedene Filter aktiviert und über das Pulldown Menü Working Sets ausgewählt und weitere Einstellungen vorgenommen werden. Die View muss gegebenenfalls aktualisiert werden, damit die Informationen korrekt angezeigt werden. Header, die mit spitzen Klammern includiert sind (`#include <ios.h>`), werden entsprechend gekennzeichnet.

10.5.7 Type Hierarchie

Die Type Hierarchie View zeigt die Vererbungshierarchie der Typen. Die Ansicht kann über die Toolbar auf die Basistypen oder auf die Subtypen des ausgewählten Typs eingeschränkt werden.
Die Type Hierarchie kann über *Window.Show View.Type Hierarchie* oder das Kontextmenü eines Typs

- im Editor (Shortcut: **F4** Quick Type Hierarchie **Ctrl+t**)
- der Outline View und
- in der Navigation View

geöffnet werden. Bei geöffneter View kann via **Drag´n Drop** ein Typ in die View gezogen werden. Über die View kann direkt zu den Symbolen im Editor navigiert werden.

In Abbildung 44 auf der nächsten Seite wird die Type Hierarchie des Typs `Betrieb` aus dem Projekt Verkehrssteuerung mit seinem Basistyp `AmpelState` und den Subtypen `Anhalten`, usw. angezeigt. Der Typ `StartVorbereiten` hat weitere Subtypen, die nicht sichtbar sind, weil der Knoten nicht aufgeklappt ist.

Über die Toolbar der View können verschiedene Filter aktiviert und Aktionen gestartet werden. Von Links nach Rechts:

- Basistypen und abgeleitete Typen
- Nur die Basistypen

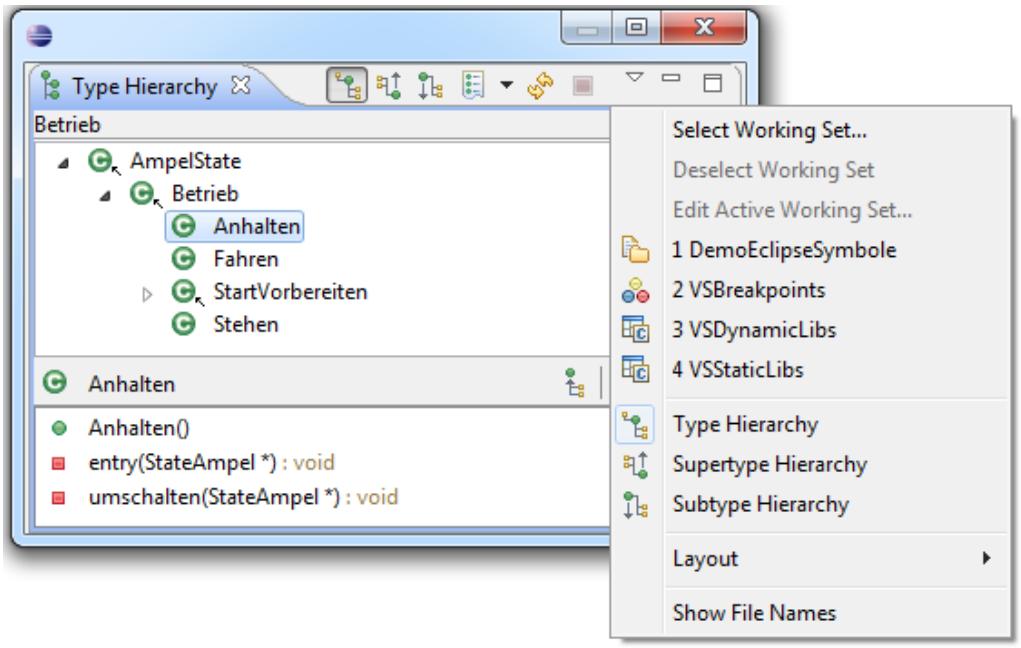

Abbildung 44: Die Type Hierarchie View

- Nur die abgeleiteten Typen
- Auswahl der zuletzt ausgewählten Typen
- Aktualisierung der View
- Abbruch der Aktualisierung

Über das Pulldown Menü der View kann das Layout angepasst und Working Sets als Filter ausgewählt werden.

Unten in der View werden die Member des in der oberen View ausgewählten Typs `Anhalten` angezeigt. Über die Toolbar dieses Fensters können ebenfalls Filter aktiviert werden. Bis auf das sichtbare Pictogramm, sind es dieselben wie in der Outline View in section 10.5.4 auf Seite 45 beschrieben. Die Anzeige der Member kann über das sichtbare Pictogramm auf die Member des ausgewählten Typs oder auf alle Member, inclusive der geerbten, eingestellt werden.
Beachte: Werden Typen aus einem anderen Projekt geerbt, muss dieses Projekt in den Projekt References wie in Abbildung 45 auf der nächsten Seite angegeben werden, damit die Typen in der View korrekt anzeigt werden.

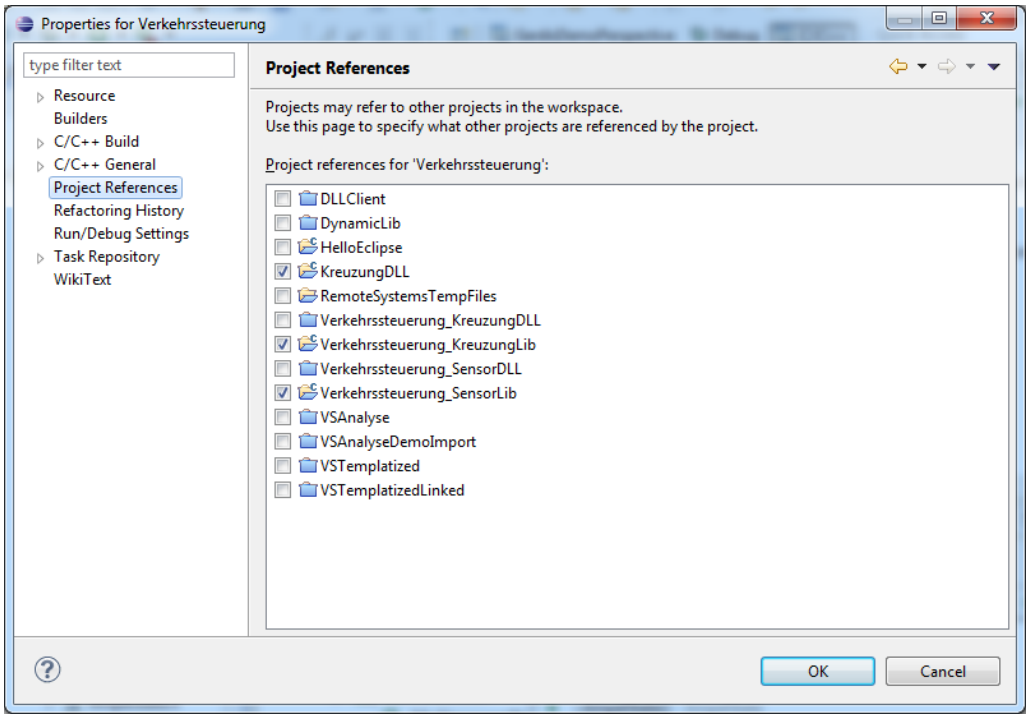

Abbildung 45: Die Project References

10.6 Code Templates und Formatierung

10.6.1 Code Templates

Code Templates werden im Editor über **Ctrl+Blank** angefordert. Eine Vorauswahl kann über den Namen der Templates erfolgen. Dazu muss im Editor der Name oder ein Teil des Namens eingegeben werden und anschließend **Ctrl+Blank** eingegeben werden.

Die Abbildung 46 auf der nächsten Seite zeigt den Auswahldialog für die vorgeschlagenen Templates im Editor. Für unbekannte Namen im Template (z.B. *var*) wird eine Benutzereingabe angefordert, Namen die Eclipse bekannt sind werden ersetzt, bzw. haben eine bestimmte Funktion. Beim Einfügen des Templates kann nach der Eingabe für die erste Variable mit **Tab** die nächsten Variable (*max*) durch einen Text ersetzt werden.
Mit Enter gelangt man an die Position der Variablen **${cursor}**, siehe Abbildung 47 auf der nächsten Seite *Preview:*. Der Name *cursor* ist eine Eclipse Variable.

Namen die eine Funktion haben sind z.B.

- ${cursor} Position des Carets nach Abschluss der Variablen Eingabe
- ${line_selection} Position der Textstelle, die vor der Anwendung des Templates markiert war
- ${enclosing_method} der Name der umgebenden Methode
- usw.

Im Editor (*Edit...* oder *New...*) eines Templates können die Variablen aus einer Liste ausgewählt werden.

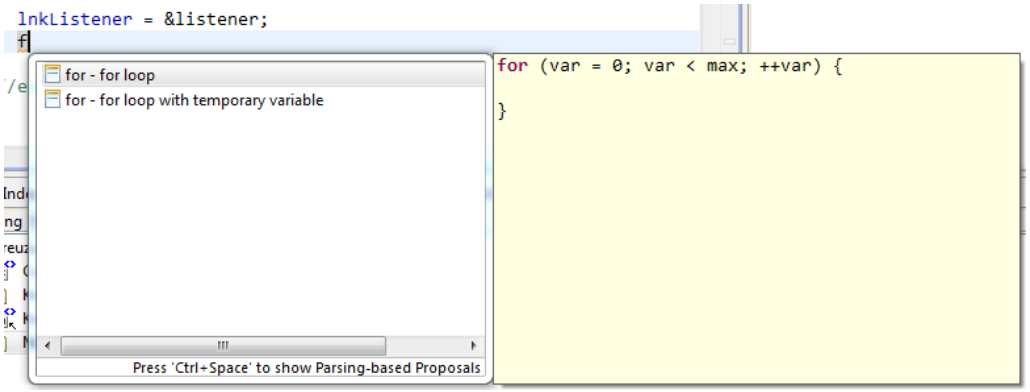

Abbildung 46: Code Assistance und Templates

Über das Menü *Window.Preferences* mit dem Suchbegriff `Temp` wie in Abbildung 47 gezeigt, können die vorkonfigurierten Code Templates bearbeitet und neue hinzugefügt werden. Der *Context* bestimmt, wann das Template angewendet werden

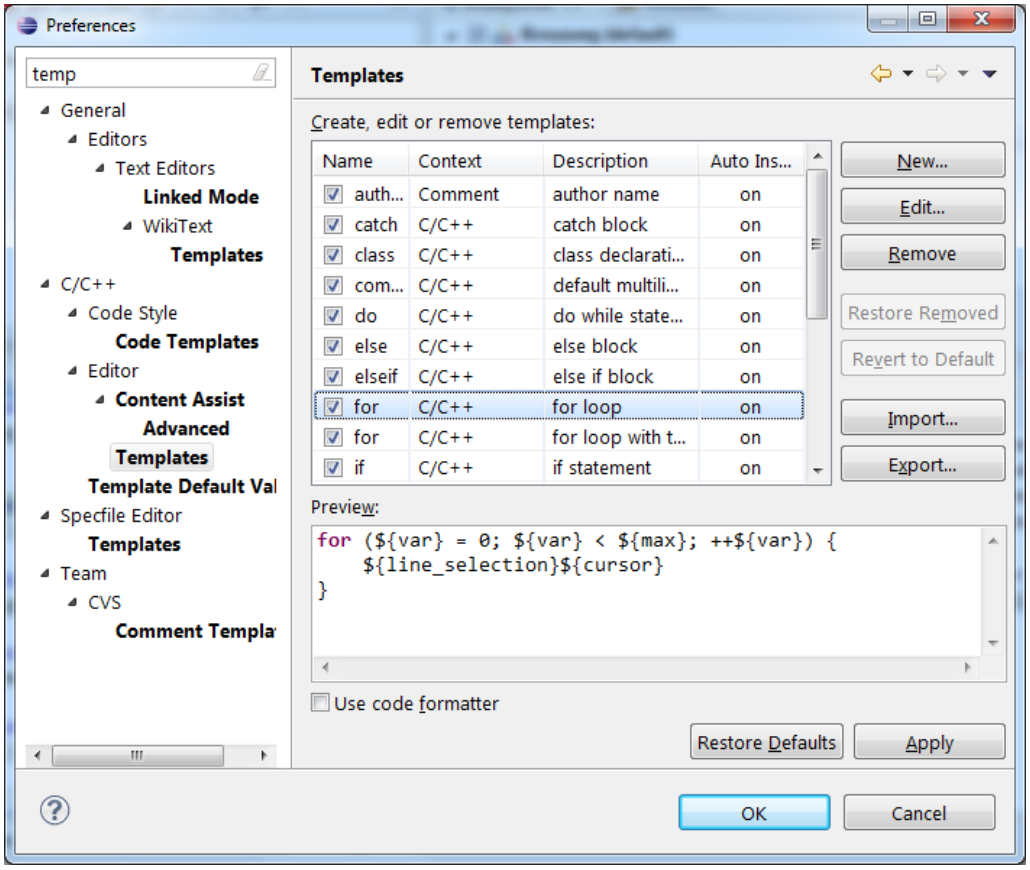

Abbildung 47: Sourcecode Templates

kann, das wird bei den Vorschlägen berücksichtigt, die Description wird in der Vorschlagsliste verwendet. *Auto Insert* bestimmt, ob das Template automatisch eingefügt wird, wenn keine anderen Ergänzungen möglich sind.

10.6.2 Formatierung

Der Sourcecode in einem Editor kann automatisch formatiert werden. Die Vorlagen dazu werden über den in Abbildung 48 abgebildeten Dialog verwaltet. Über das Kontextmenü *Source.Format* oder über **Ctrl+Shift+F** wird der Code gemäß der eingestellten Formatierung formatiert.

Tipp: Wird Code mit einer neuen Formatierung versehen, sollte der Kommentar beim Commit in ein Versionsverwaltungssystem einen entsprechenden Hinweis enthalten, da eventuell sehr viele geänderte Stellen vorhanden sind, die sich nur durch Umformatierung ergeben.

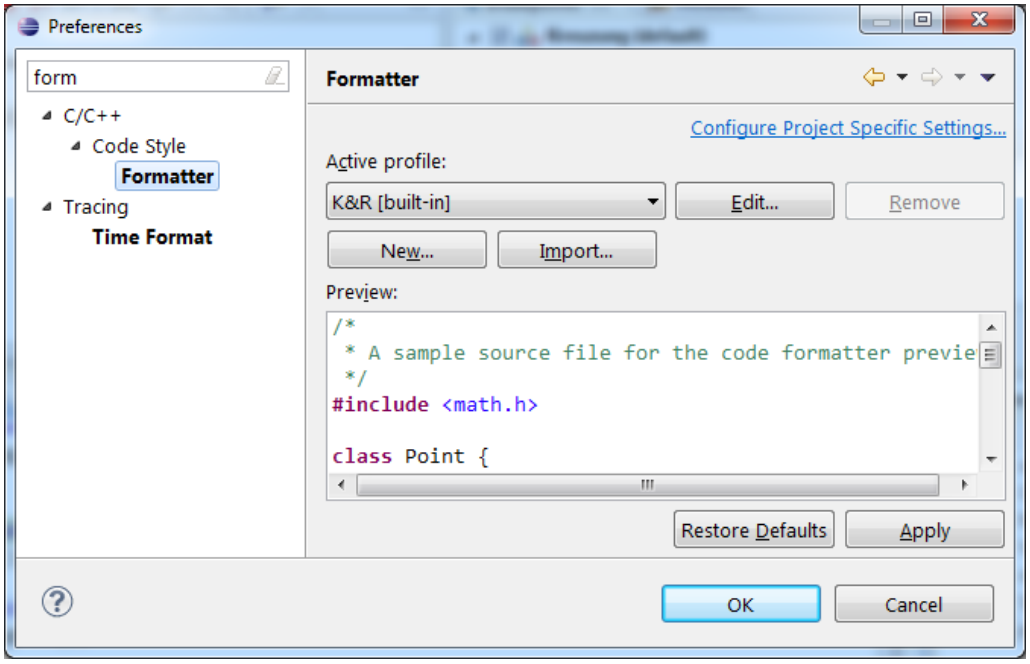

Abbildung 48: Sourcecode Formatierung

10.7 Die wichtigsten Views der Debug Perspective

Die Debug Perspective in Abbildung 49 stellt die wichtigsten Views für eine Debug Session zur Verfügung.

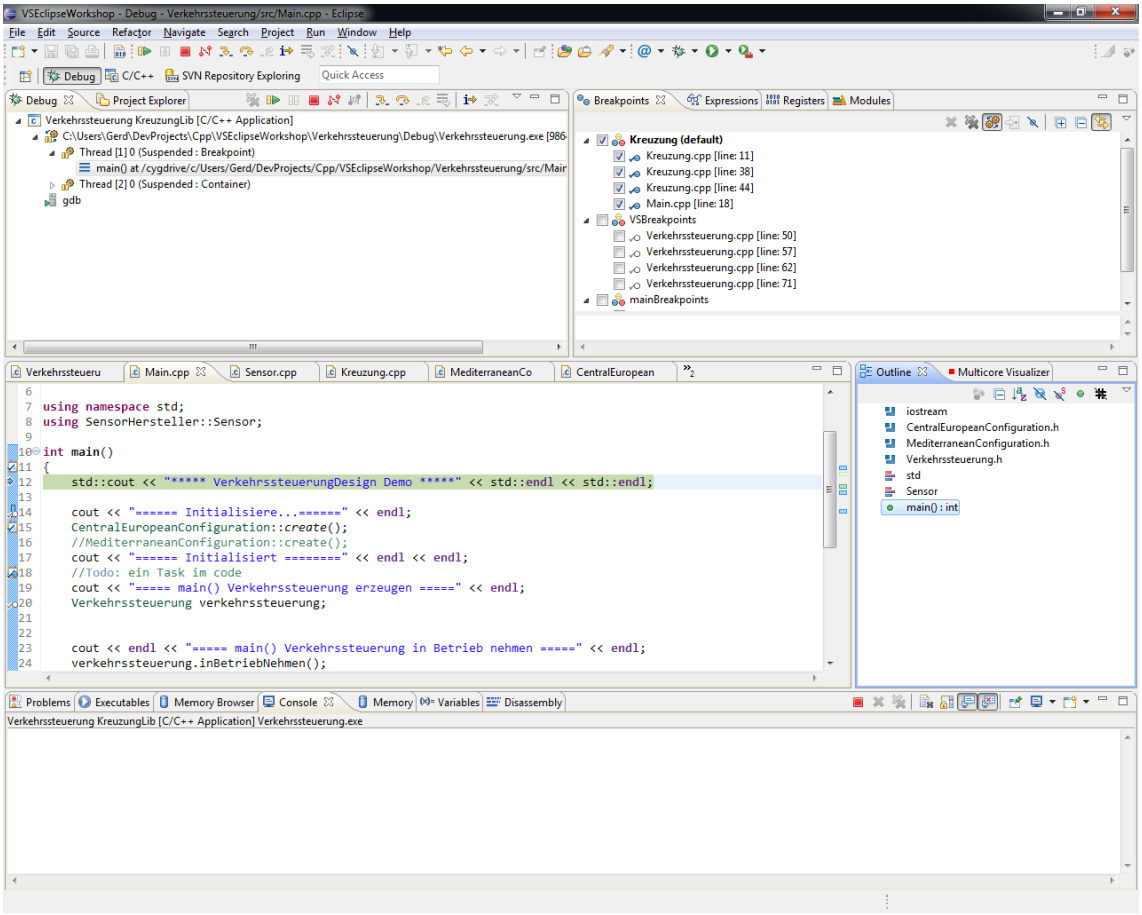

Abbildung 49: Die Debug Perspective

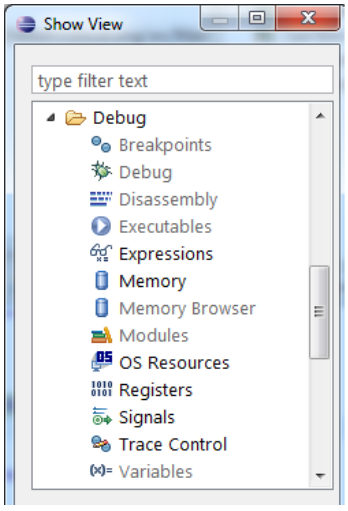

Abbildung 50: Die Debug Views

Weitere aus Abbildung 50 können über *Window.Show View* hinzugefügt werden.

Beim Einsatz von Cygwin muss beim Debuggen ein path mapping eingerichtet werden, damit der gdb die sourcen findet:
http://wiki.eclipse.org/DSDP/DD/GDBTorun:11.InthecaseofCygwin...

10.7.1 Debug

Die Debug View in Abbildung 51 zeigt die Threads der Debug Session. Über das Pulldown Menü kann die Debug Toolbar eingeschaltet werden. Hier sind die üblichen Debug Kommandos wie step into, step over, und weitere zu finden.
Von links nach rechts:

- Remove all Terminated Launches
- Resume **F8**
- Suspend
- Terminate **Ctrl+F2**
- Disconnect
- Connect to a Process
- step into **F5**
- step over **F6**
- step return **F7**
- drop to Frame
- Instruction stepping mode[37]
- Use step filters

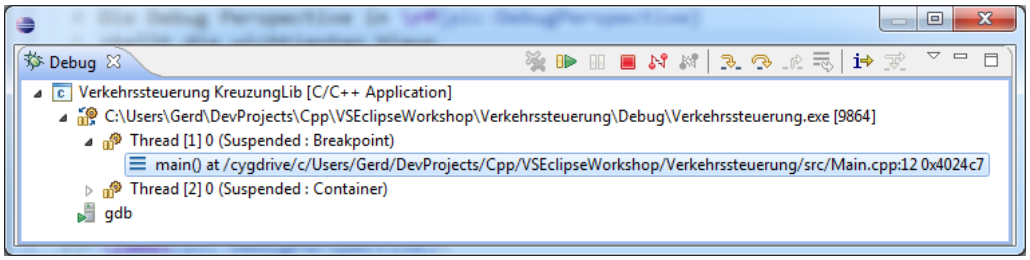

Abbildung 51: Die Debug View

Über die Console des GNU Debuggers in Abbildung 55 auf Seite 61 kann direkt mit dem Debugger kommuniziert werden.

Der Debugger verhält sich wie in Diagramm 4 auf der nächsten Seite beschrieben. Über das Kontextmenü eines *Terminated* Threads kann dieser mit *Relaunch* wieder aktiviert werden.

[37] siehe section 10.7.9 auf Seite 65

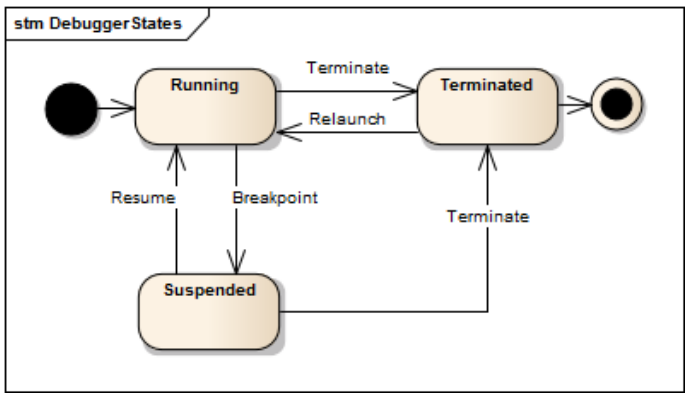

Diagramm 4: Die Zustände des Debuggers

10.7.2 Breakpoint View

Die Breakpoint View zeigt die definierten Breakpoints und ermöglicht die Definition von weiteren Breakpoints. Über Breakpoint Working Sets können Breakpoints gruppiert und als Gruppe de-/aktiviert werden.

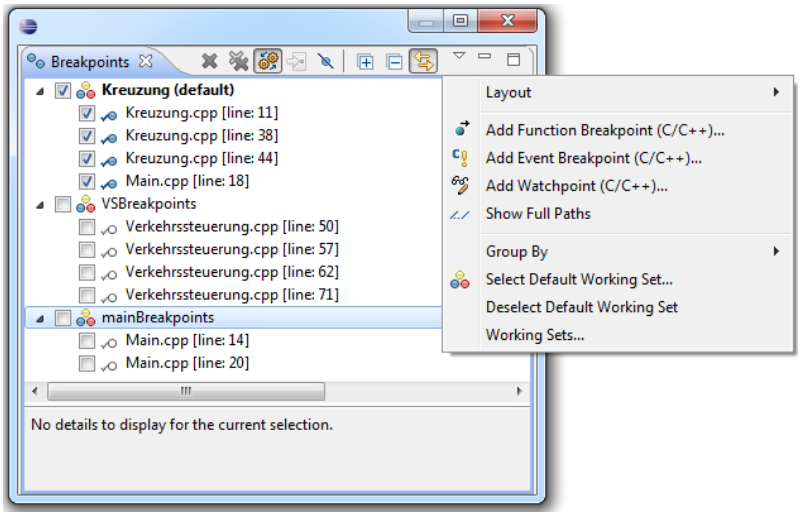

Abbildung 52: Die Breakpoint View mit Working Sets

10.7.3 Line & Address-Breakpoints

Line Breakpoints können im Editor am linken Rand durch **Doppelclick** gesetzt und gelöscht und mit dem Kontextmenü im Editor verwaltet werden. In der Disassembly View[38] können auf dieselbe Weise Address-Breakpoints verwaltet werden.

[38] section 10.7.9 auf Seite 65

10.7.4 Watchpoints

Ein Watchpoint wird häufig auch als *Data Breakpoint* bezeichnet. Watch Points können mit der graphischen Oberfläche von Eclipse nur für **globale Variablen** gesetzt werden. Der Dialog in Abbildung 53 zur Bearbeitung eines Watch Points kann über das Pulldown Menü der Breakpoint View *Add Watchpoint(C/C++)* geöffnet werden. Der Name der Variablen (hier `global`), deren Änderungen überwacht werden sollen, muss im Feld *Expression to watch:* eingetragen werden. In diesem Feld kann jede gültige Expression eingetragen werden[39]. Alternativ kann der Watchpoint über das Kontextmenü *Toggle Watchpoint* in der Outline View hinzugefügt werden oder über das Menü *Run.Toggle Watchpoint* wenn die Variable im Editor markiert ist.

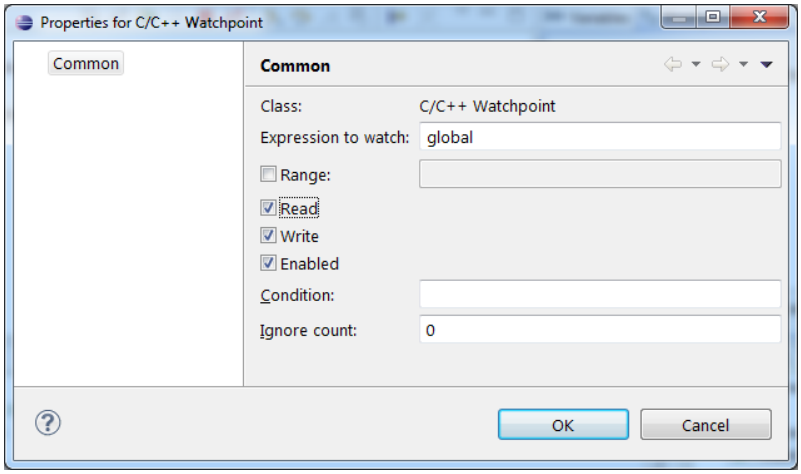

Abbildung 53: Der Watch Point Dialog Common

Über das Kontextmenü eines Breakpoints *Breakpoint Properties* wird der Dialog in Abbildung 54 auf der nächsten Seite geöffnet. Der Knoten *Common* ist derselbe wie in Abbildung 53. Nachdem der Watchpoint festgelegt ist, können im Properties Dialog verschiedene *Actions* und *Filter* festgelegt werden.

Um eine **lokale Variable** (hier `local`) als Watchpoint zu definieren, muss direkt mit dem Gnu Debugger (gdb) über die Console in Abbildung 55 auf der nächsten Seite kommuniziert werden oder in der Memory View[40] für die Adresse der Variablen ein Watchpoint gesetzt werden.
Nach der Eingabe `watch local > 1` wird der Watchpoint vom Debugger mit einer ähnlichen Ausgabe wie `Hardware watchpoint 2: local > 1` bestätigt[41]. Für andere Toolchains ist die Interaktion mit dem Debugger eventuell verschieden und kann in der Dokumentation des jeweiligen Debuggers[42] nachgelesen werden.

Der Watchpoint wird in der Breakpoint View angezeigt und kann über den Properties Dialog bearbeitet werden.

[39] http://sourceware.org/gdb/onlinedocs/gdb/Set-Watchpoints.html
[40] siehe section 10.7.7 auf Seite 64
[41] http://www.fayewilliams.com/2013/02/19/eclipse-cdt-toggle-watchpoint-greyed-out-juno/
[42] GDB QUICK REFERENCE V4: http://users.ece.utexas.edu/~adnan/gdb-refcard.pdf und http://www.yolinux.com/TUTORIALS/GDB-Commands.html

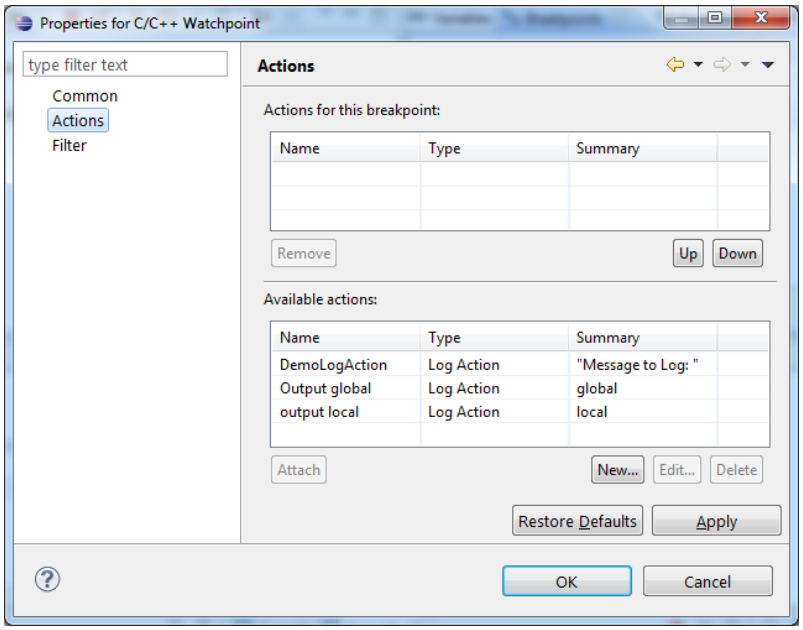

Abbildung 54: Der Watch Point Dialog Actions

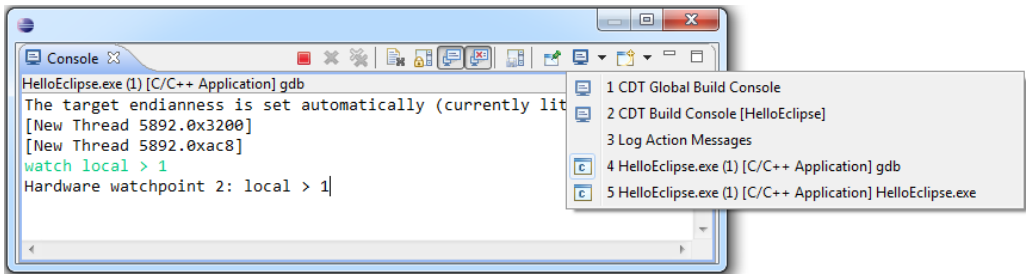

Abbildung 55: Die gnu debugger (gdb) Console

Watchpoints auf lokalen Variablen müssen bei jedem Start des Debuggers neu festgelegt werden.

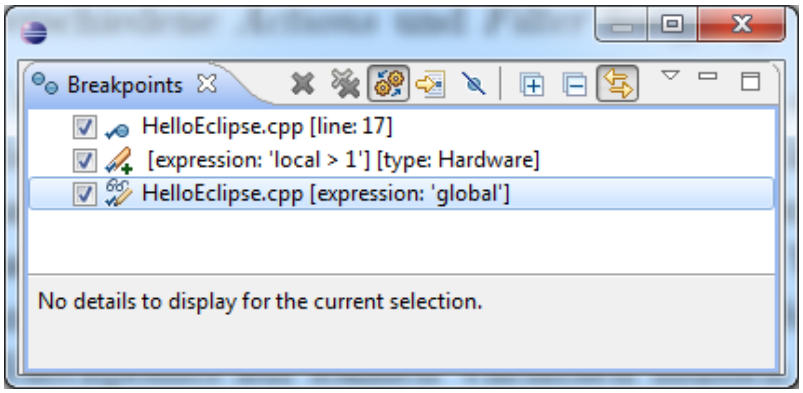

Abbildung 56: Die Breakpoint View mit einem local Watchpoint

10.7.5 Variables

Die Variables View zeigt die Variablen, die im aktuellen Scope sichtbar sind. Die Abbildung 57 zeigt das lokale Objekt `verkehrssteuerung` und die darin enthaltenen Member. Über der View ist der Editor abgebildet, in dem die aktuelle Zeile, auf der der Programcounter steht, markiert ist. Wird in der View eine Variable markiert, werden die Details dazu in der unteren View angezeigt. Die Werte können direkt in der Tabelle verändert werden.

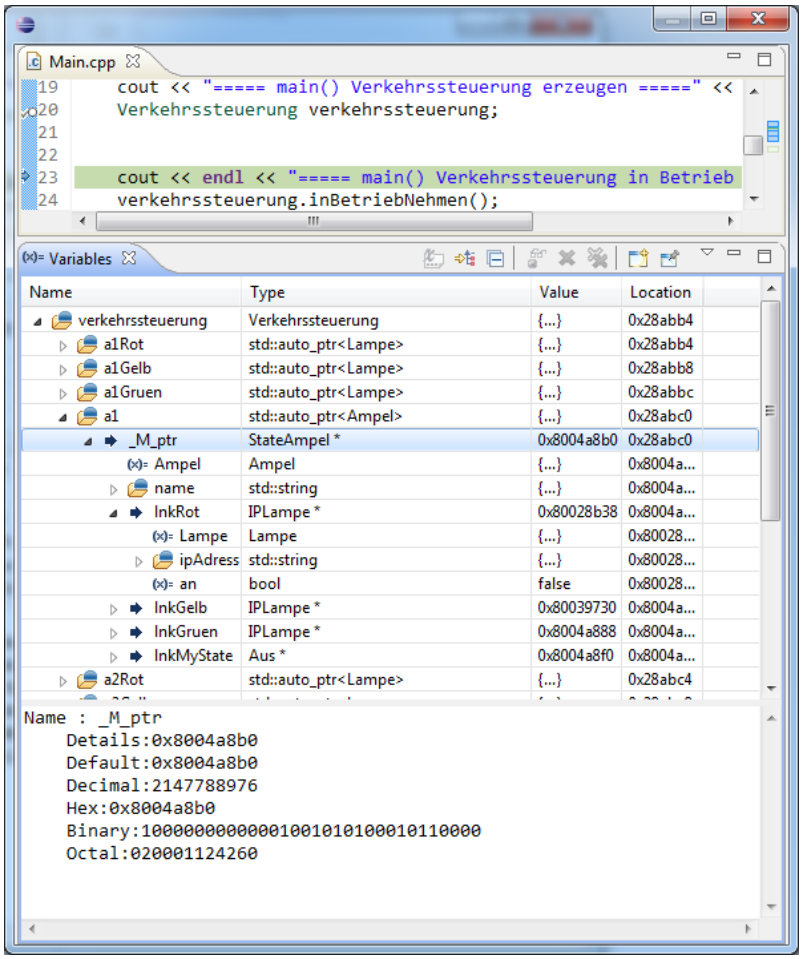

Abbildung 57: Die Variables View

10.7.6 Expressions

In der Expressions View können gültige C++ Ausdrücke auf der Basis der sichtbaren Namen evaluiert werden.

Damit ist es möglich, während einer Debug Session, eine vorhandene Funktion oder Operation einer Klasse aufzurufen (ein Funktionsaufruf ist ein Ausdruck!). Eine Methode könnte z.B. eine abstrakte Beschreibung der internen Datenstruktur eines Objekts in einer Datei ablegen, die von einem anderen Werkzeug in eine adäquate graphische Repräsentation überführt wird, mit der das Verständnis des Systemzustands erleichtert und das Debuggen unterstützt wird.

Die folgenden Abbildungen und Codefragmente zeigen ein einfaches Beispiel für das beschriebene Szenario.

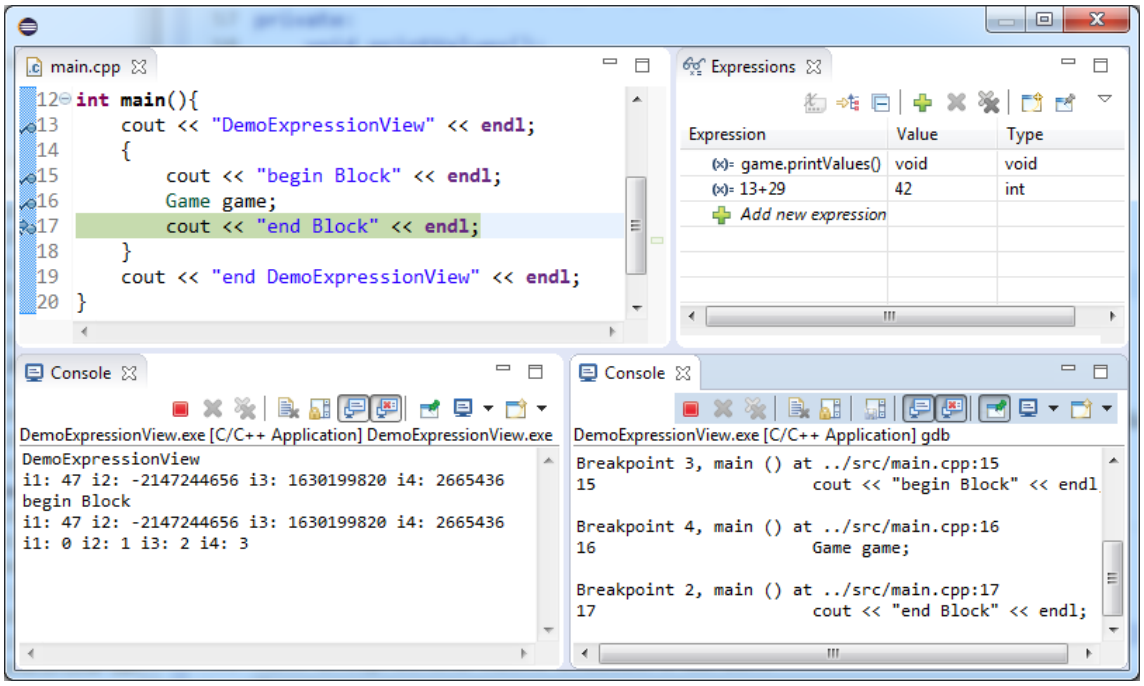

Abbildung 58: Die Expressions View

Die Abbildung 58 zeigt oben den Editor mit `main()` und den Breakpoints in Zeile 15, 16 und 17. Rechts davon die *Expressions* View mit den Expressions `game.printValues()` und `13+29`. In den Spalten neben den Expressions werden die Ergebnisse und die Ergebnistypen der Ausdrücke angezeigt.

Wenn die Expressions View geöffnet ist, werden die Expressions in der View bei jedem Stopp des Debuggers evaluiert, nachdem alle Anweisungen bis zu dem Stopp ausgeführt wurden.

Unten in der Abbildung ist links die Console der Anwendung *DemoExpressionView* mit deren Ausgaben und den Ausgaben die sich aus der Expressions View ergeben und rechts die Console des *gdb* mit den Ausgaben des Debuggers.

Solange das Objekt `game` noch nicht initialisiert ist, sind die Werte der Member entsprechend sinnlos. Erst nach dem die Zeile 16 in `main()` ausgeführt wurde, der Programcounter in Zeile 17 wie in der Abbildung steht, sind die Member initialisiert.

Listing 6: Die Klasse Game für die ExpressionsView

```
1 // Game.h
2 class Game {
3 public:
4    Game() : i1(0), i2(1), i3(2), i4(3) {}
5 private:
6    void printValues();
7    int i1, i2, i3, i4;
8 };
```

```
9  //Game.cpp
10 void Game::printValues(){
11    cout
12       << "i1: " << i1
13       << " i2: " << i2
14       << " i3: " << i3
15       << " i4: " << i4
16    << endl;
17 }
```

Das Listing 6 auf der vorherigen Seite zeigt die Klasse `Game` und die Methode `printValues()` aus der Expressions View. Methoden die `inline` implementiert sind, können nicht in der Expressions View verwendet werden.

Die Abbildung 59 zeigt die Expressions View mit dem Programcounter in Zeile 13. Das Symbol `game` ist noch nicht im Scope, die Expression `game.printValues()` kann entsprechend nicht evaluiert werden.

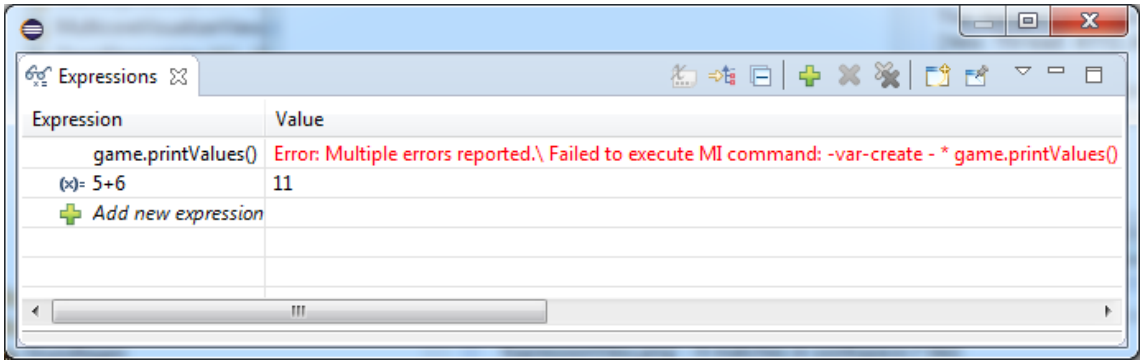

Abbildung 59: Die Expressions View mit einem Error

10.7.7 Memory

Die Memory View in Abbildung 60 auf der nächsten Seite zeigt einen Ausschnitt aus dem Hauptspeicher ab einer bestimmten Adresse. Aus der Variablen View kann eine Variable über das Kontextmenü in der Memory View angezeigt werden. Die erste Spalte zeigt die Adressen, die Spalten *0 - 3* usw. zeigen jeweils 4 Byte aus dem Hauptspeicher. Der Speicherbereich der Variablen (hier `i`) ist markiert (Spalte *8 - B*). Die Werte können direkt in der Tabelle verändert werden. Die Anzeige der Werte kann über den Tab *New Renderings...* angepasst werden.

Über das Kontextmenü eines Speicherbereichs kann ein Watchpoint für diesen Speicherbereich gesetzt werden, der in der Breakpoints View angezeigt wird: (`[expression:'0x28abe8'][units:4]`).

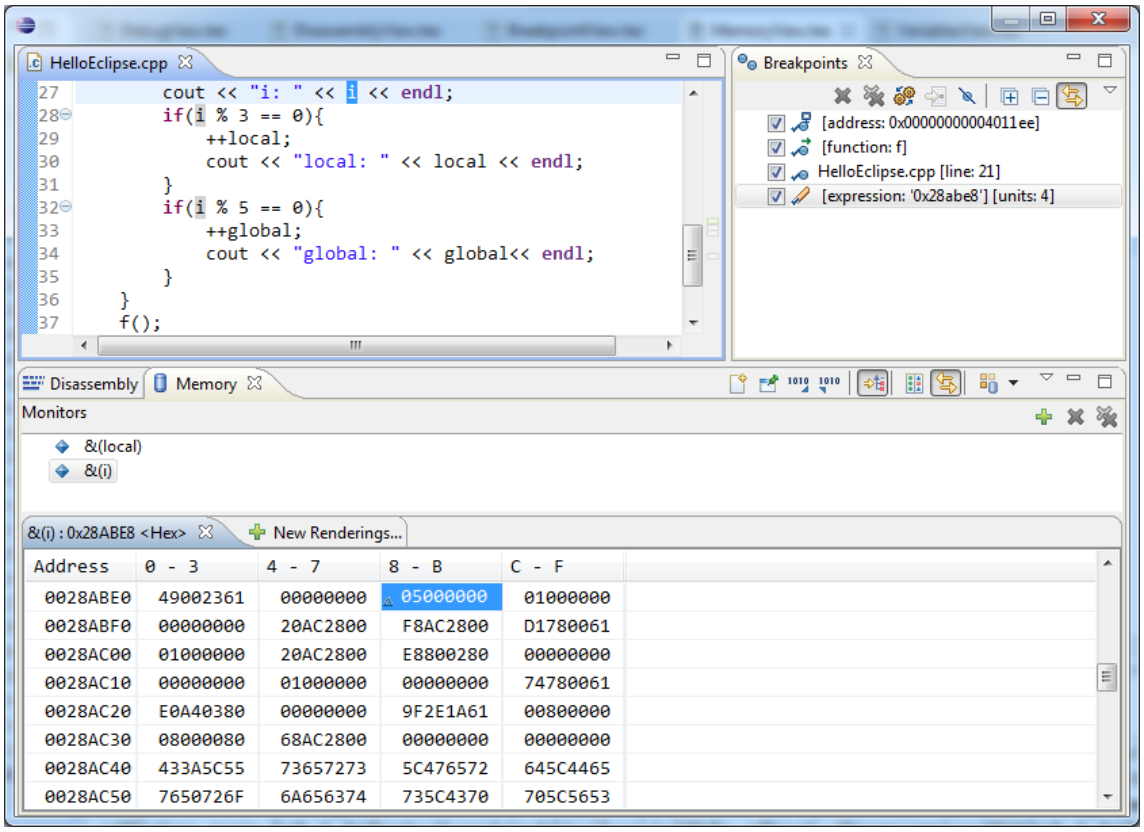

Abbildung 60: Die Monitor View

10.7.8 Registers

Die Registers View in Abbildung 61 auf der nächsten Seite zeigt die Register der CPU. Die Werte können direkt in der Tabelle verändert werden.

10.7.9 Disassembly

Die Disassembly View in Abbildung 62 auf der nächsten Seite zeigt den generierten Assembler Code. Der Editor und die View sind synchronisiert.

Im *Instruction Stepping Mode* des Debuggers, wird anstatt im Sourcecode in der Disassembly View zeilenweise ausgeführt. Der kleine Pfeil links am Rand repräsentiert den Program Counter `004011da`. Darüber befindet sich das Symbol für einen Line-Breakpoint (line: 21), der auch im Editor angezeigt wird. Darunter befindet sich ein Address-Breakpoint `004011ee`, der nur in der Breakpoints View rechts oben in Abbildung 62 auf der nächsten Seite und in der Disassembly View angezeigt wird. Address-Breakpoints werden gesetzt wie Line Breakpoints in section 10.7.3 auf Seite 59.

Die View muss gegebenenfalls aktualisiert werden.

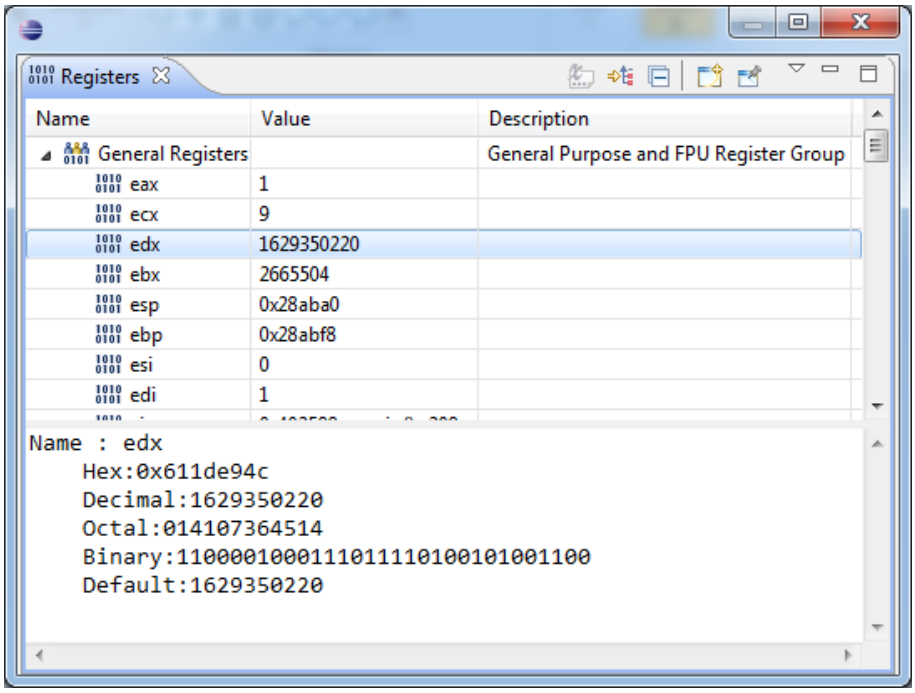

Abbildung 61: Die Registers View

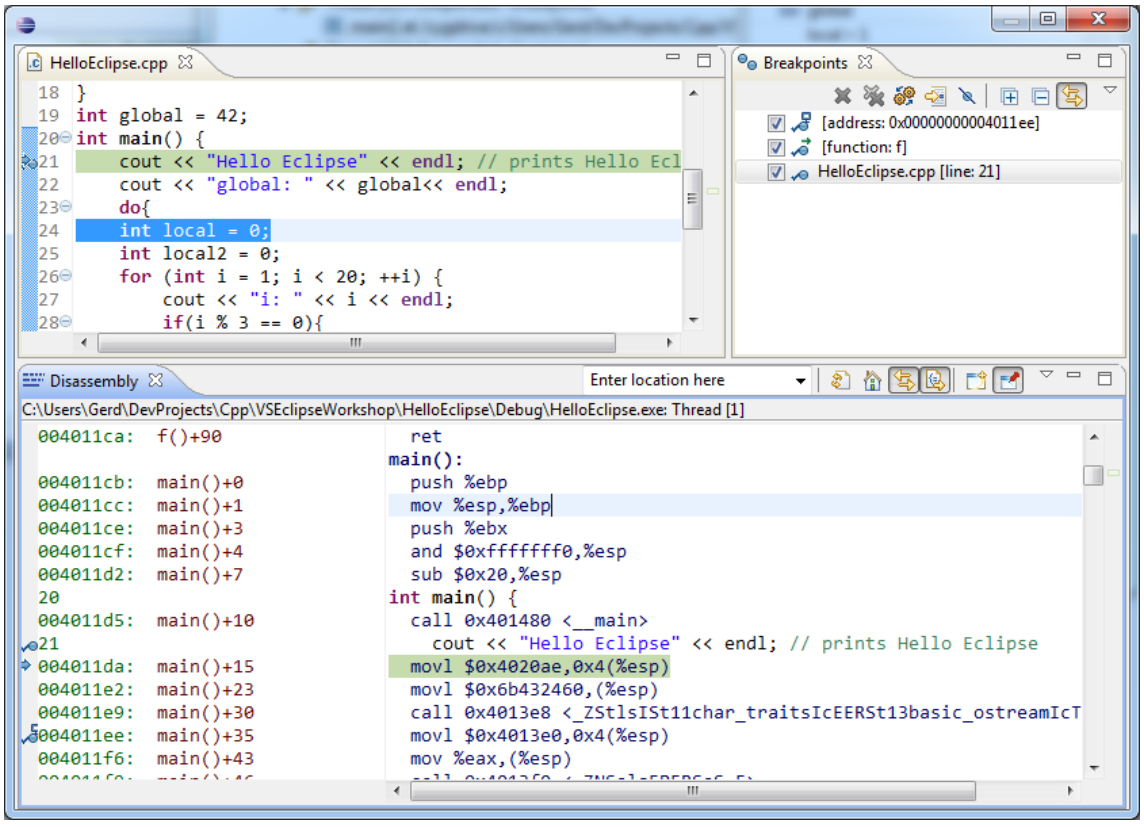

Abbildung 62: Die Disassembly View

10.8 Das Build System

Eclipse/CDT unterscheidet zwischen *managed build* und *Makefile* Projekten. Für managed build Projekte identifiziert Eclipse die Abhängigkeiten und erzeugt bei jedem Build die notwendigen Makefiles. Für Makefile Projekte müssen die Makefiles durch den Anwender gepflegt werden. Welche Art von Projekt erzeugt werden soll, kann bei der Erstellung in Abbildung 27 auf Seite 38 ausgewählt werden.

10.8.1 Toolchain

Das CDT enthält keinen eigenen Compiler. Die verschiedenen Compiler und andere Werkzeuge werden über **Toolchains** eingebunden und verwaltet. Eine Toolchain ist eine Sammlung von Tools und die Definition, welcher Input von einem Tool benötigt und welcher Output daraus erzeugt wird.

Die folgenden utitlities müssen für C/C++ installiert sein:

- Build (z.B. make oder *internal Builder*)
- Compile (z.B. g++)
- Debug (z.B. gdb)

10.8.2 Builder

Ein **Builder** steuert einen **Build Prozess**.

Jeder Builder ist mit den Projekt Typen[43] assoziiert, auf die er sinnvoll angewendet werden kann.

Für ein C++ Projekt sind mindestens der Builder zur Übersetzung (CDT Builder) und der Builder für die Erzeugung des Index (Scanner Configuration Builder[44]) konfiguriert.

Über das Kontextmenü *Properties* des Projekts kann der Dialog in Abbildung 63 auf der nächsten Seite geöffnet werden. Beim Build Process werden die Builder in der Reihenfolge in der Liste ausgeführt. Über den Button *Edit...* kann der *Configure Builder* Dialog geöffnet werden.

10.8.3 Build-Variablen

Build-Variablen und Umgebungsvariablen repräsentieren abstrakt einen Wert in Form einer Zeichenkette.

[43] Eclipse Jargon: “Nature”
[44] siehe section 10.3 auf Seite 42

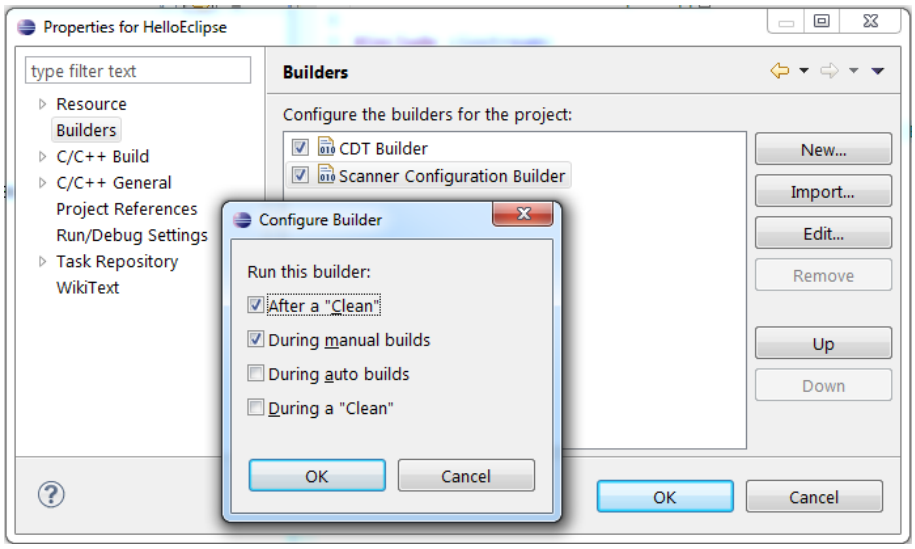

Abbildung 63: Die default Builder eines C/C++ Projekts

Sie werden durch

- das CDT
- eine Toolchain
- die Benutzer

definiert. In Abhängigkeit, wo die Variablen definiert sind, erstreckt sich ihr Scope/Gültigkeitsbereich.

Die Bereiche sind von rechts nach links ineinander eingebettet.
Workspace <- Projekt <- Toolchain <- Build Configuration
Gleichnamige Variablen in inneren Bereichen überschreiben die Variablen aus äußeren Bereichen. Das CDT fügt den Workspace weiten Variablen die System Umgebungsvariablen hinzu. Eine projektspezifische Systemvariable ist zum Beispiel *BuildArtifactFileExt* mit dem Wert **a** bei statischen Libraries und dem Wert **exe** bei ausführbaren Programmen auf Windows Systemen.

Die Abbildung 64 auf der nächsten Seite zeigt den Dialog zur Verwaltung der Build-Variablen für den gesamten Workspace. Benutzerdefinierte[45] werden fett dargestellt. Der *Type* einer Build-Variablen wird zur Auswahl des Dialogs in Abbildung 65 auf der nächsten Seite zur Bearbeitung herangezogen.

Systemvariablen können nicht gelöscht werden. Wird eine Systemvariable durch einen Benutzer redefiniert, wird sie fett dargestellt und wenn sie gelöscht wird, ist die Ursprüngliche wieder sichtbar.

Ein Ausdruck der mit einem Dollarzeichen beginnt und den Namen einer Variablen in geschwungenen Klammern enthält (**${<variablen-name>}**) repräsentiert ihren Wert. Er kann in allen Textfeldern verwendet werden die mit einem Button *Variables...* ausgestattet sind wie in Abbildung 70 auf Seite 74. Der Ausdruck wird bei Bedarf durch die Zeichenkette der Variablen ersetzt.

[45] Hier nur Beispiele mit allen *Type*s die es gibt

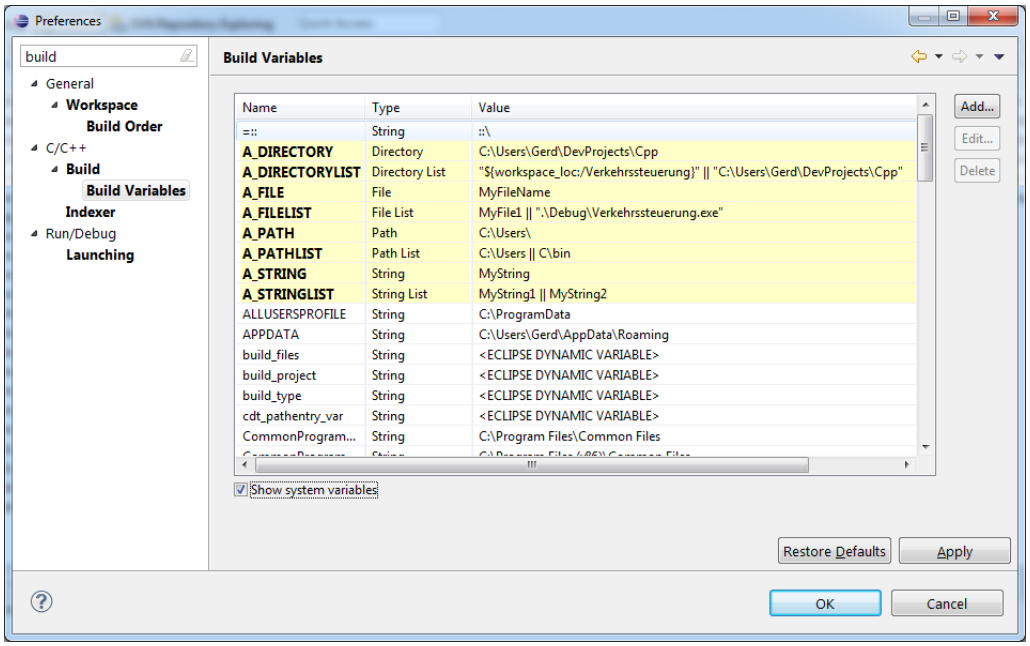

Abbildung 64: Benutzer definierte Build-Variablen

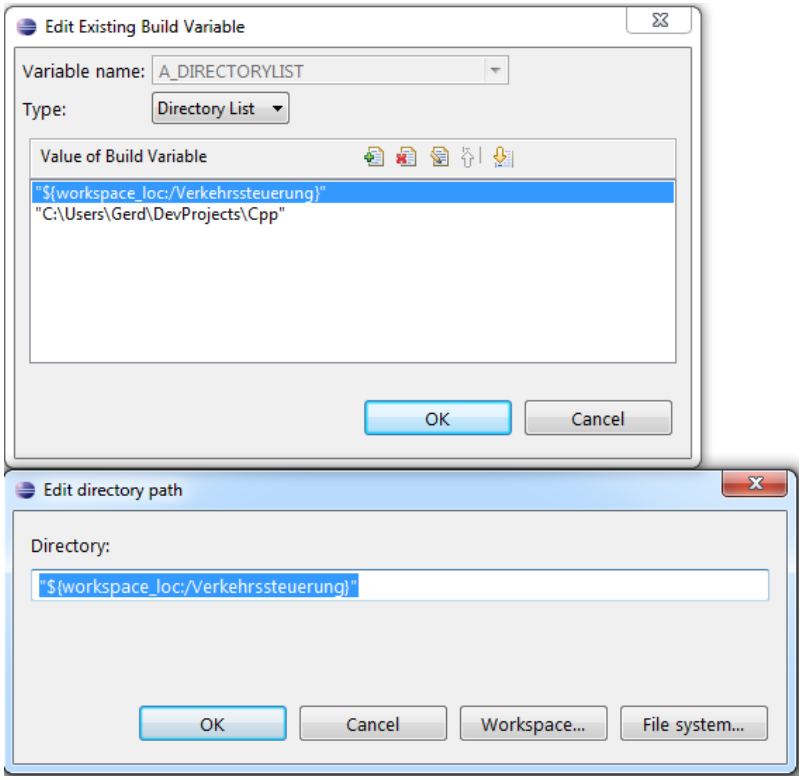

Abbildung 65: Build-Variablen bearbeiten

System Variablen mit dem *Value* **ECLIPSE DYNAMIC VARIABLE** wird der Wert in Abhängigkeit des aktuellen Kontext ihrer Verwendung zugewiesen. Einige benötigen ein zusätzliches Argument, das in der Form: **${<variablen-name>:argument}** angegeben wird.

Die Abbildung 66 auf der nächsten Seite zeigt den Tab *Environment* einer Run Configuration. Die Umgebungsvariable *PATH* wird durch die dynamische System

Variable env_var mit dem Argument PATH ermittelt und um den Pfad der DLL, von der die Anwendung abhängig ist, erweitert.
(${env_var:PATH};${workspace_loc/KreuzungDLL/Debug})

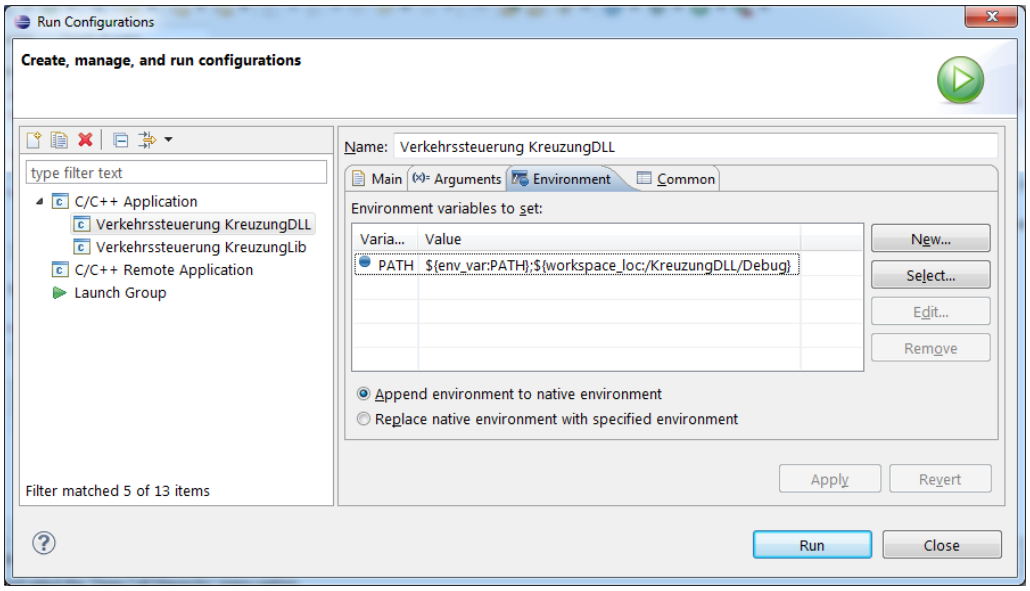

Abbildung 66: Eine Run Configuration definiert Umgebungsvariablen

10.8.4 Build Prozess

Der Build Prozess wird mit der aktiven **Build Configuration** durchgeführt, wenn er über die Toolbar[46] oder über das Kontextmenü *Build Project* des Projects oder über das Menü *Project.Build Project* gestartet wird. Alternativ kann ein Build mit mehrere Build Configurations über das Kontextmenü *Build Configurations.Build Selected..* gestartet werden.

Das Ergebnis wird in einem Ordner mit dem Namen der Build Configuration im Root Verzeichnis des Projekts abgelegt. Die Ausgaben der Tools werden in der *Build Console*[47] angezeigt.

10.8.5 Build Console und Error Parser

Die Build Console in Abbildung 67 auf der nächsten Seite zeigt die Ausgaben des Compilers.

Der Dialog in Abbildung 68 auf der nächsten Seite für die Einstellungen der Console wird über *Window.Preferences.C/C++.Build.Console* geöffnet.

Damit Eclipse die Ausgaben der Tools verarbeiten kann, um z.B. an die Stelle eines Fehlers im Editor zu navigieren (**Doppelklick auf den Fehler**) oder die Problem View zu aktualisieren, muss die Ausgabe geparst werden.

[46] siehe section 10.1.3 auf Seite 38
[47] section 10.8.5

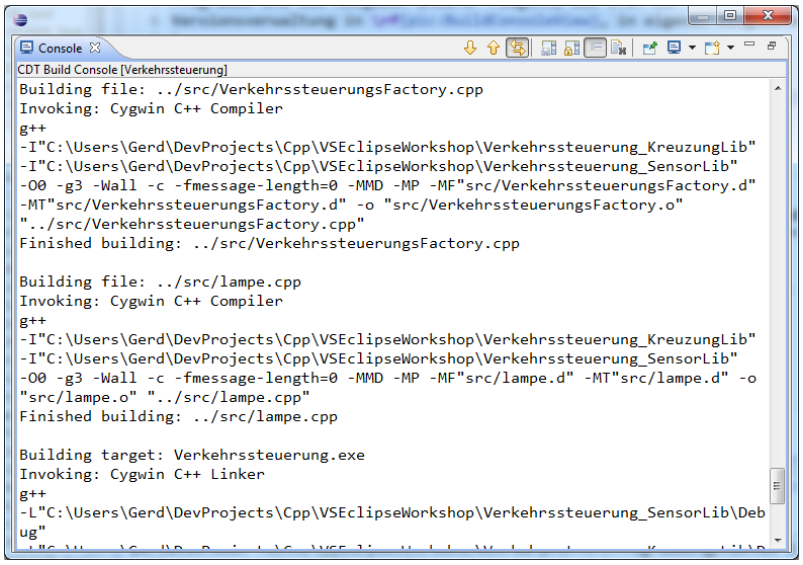

Abbildung 67: Die Build Console

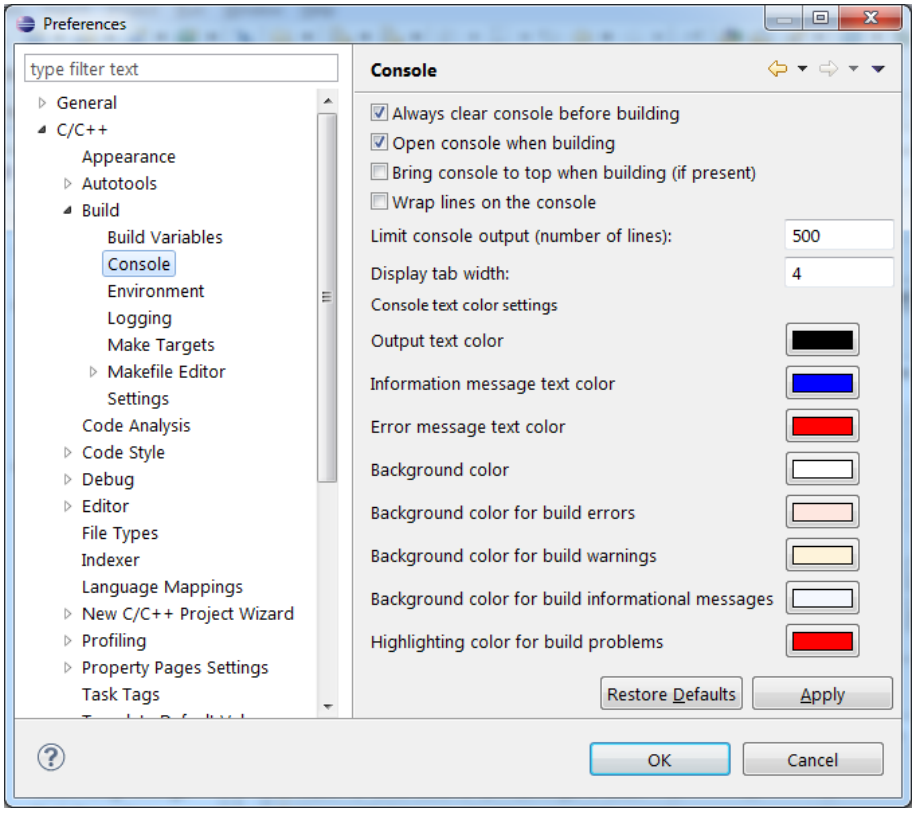

Abbildung 68: Die Voreinstellungen für die Build Console

Die Abbildung 69 auf der nächsten Seite zeigt den Dialog zur Einstellung der Workspace weiten Error Parser. Die Error Parser können mit der Checkbox in der Liste de-/aktiviert werden. Sie werden alle von oben nach unten in der Reihenfolge, wie sie der Liste dargestellt sind, auf die Ausgabe in der Console angewendet. Die Reihenfolge kann mit *Move Up/Move Down* festgelegt werden. Die Ergebnisse der Parser werden zur Verarbeitung an Eclipse weitergeleitet.

Für verschiedene Parser (z.B. GNU gcc/g++ error Parser) können in der Liste

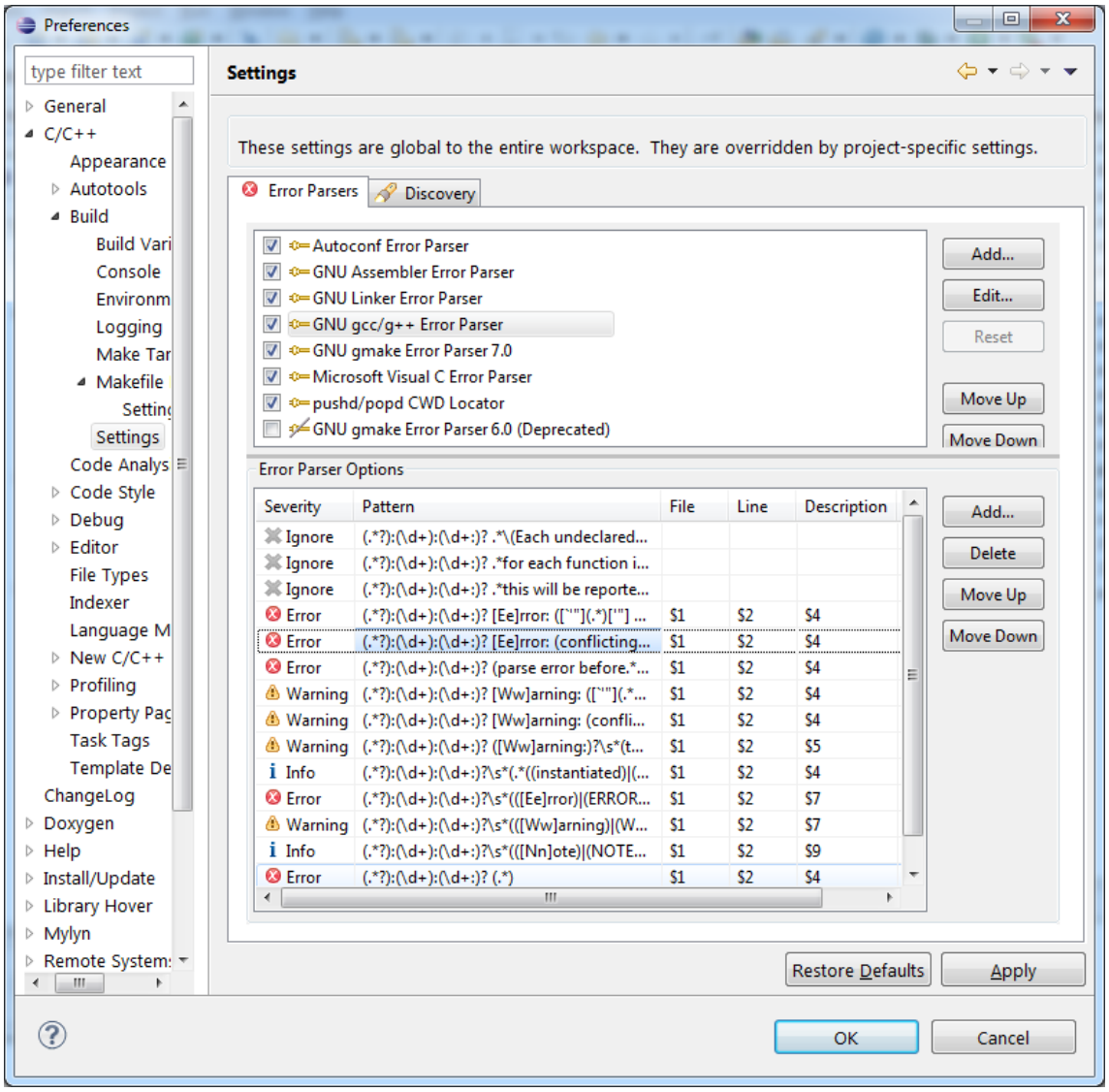

Abbildung 69: Error Parser für die Ausgabeverarbeitung

Error Parser Options die Einträge z.B. die Gewichtung (*Severity*) und das *Pattern* der Regular Expression mit einem **Doppelklick** bearbeitet werden. Die Variablen in den Spalten *File, Line, ...* entsprechen den Variablen der Pattern.

Mit *Add* können weitere *Regex Error Parser* hinzugefügt werden. In der Liste *Error Parser Options* können diesen, wieder mit *Add*, Pattern hinzugefügt und gewichtet werden. Die Erstellung eines eigenen Error Parsers für externe Programme ist in section 10.8.9 auf Seite 79 beschrieben.

Projektspezifische Einstellungen können über die Projekt Properties unter *Preferences.C/C++ Build.Settings.Error Parsers* vorgenommen werden. Der Dialog ist derselbe wie in Abbildung 69 auf der vorherigen Seite.
Beachte: Die Workspace weiten selbst erstellten Error Parser müssen für die einzelnen Projekte bei Bedarf aktiviert werden.

10.8.6 Build Configuration

Eine Build Configuration definiert

- einen Namen für die Configuration
- die Toolchain
- die Parameter für die Tools
- die Umgebungsvariablen
- den Ergebnistyp (*Shared Library, Static Library, Executable*)

Die aktive Build Configuration wird über das Kontextmenü des Projekts *Build Configurations.Set Active* und den Namen der Configuration festgelegt.

Die Einstellungen für eine oder mehrere Build Configurations werden über den Dialog in Abbildung 70 im Knoten *C/C++ Build* bzw. *C/C++ General* vorgenommen, der über das Kontextmenü *Properties* des Projects geöffnet werden kann.

Für ein Projekt können mehrere Build Configurations definiert werden. *Manage Configurations...* öffnet den Dialog zur Verwaltung von Configurations. In der ComboBox *Congiguration:* kann eine vorhandenen Build Configuration ausgewählt werden, für die Einstellungen vorgenommen werden sollen.

Stehen Mehrere zur Verfügung, können Einstellungen für alle (*All configurations*) oder für mehrere (*Multiple configurations*) gleichzeitig vorgenommen werden.

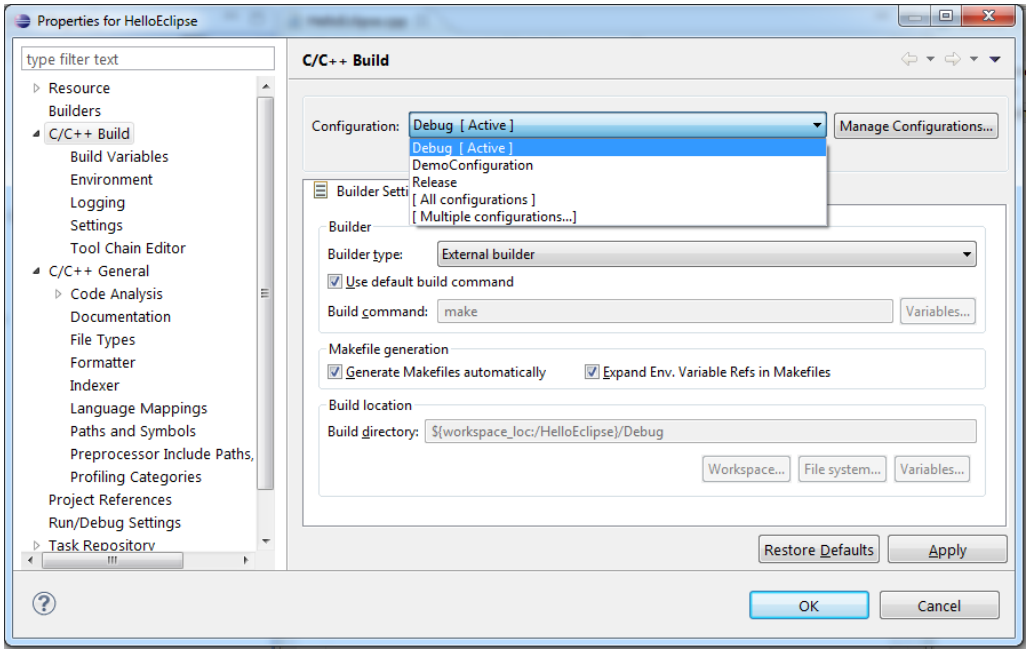

Abbildung 70: Die Einstellungen einer Build Configuration

Der *Builder type:* kann der *External builder*, der ein externes Programm, z.B. make, für den Build nutzt oder der *Internal builder* sein. Das default *Build command:* ist make.

Die Einstellungen die über den Project Properties Dialog bzgl. der Build Configurations vorgenommen werden können, entsprechen den Optionen der Werkzeuge aus der Toolchain. In Abbildung 71 ist der Knoten *C/C++ Build.Settings.Cygwin C++ Compiler* ausgewählt. Das Feld *All options:* reflektiert alle Einstellungen die in den Properties für den Compiler vorgenommen wurden.

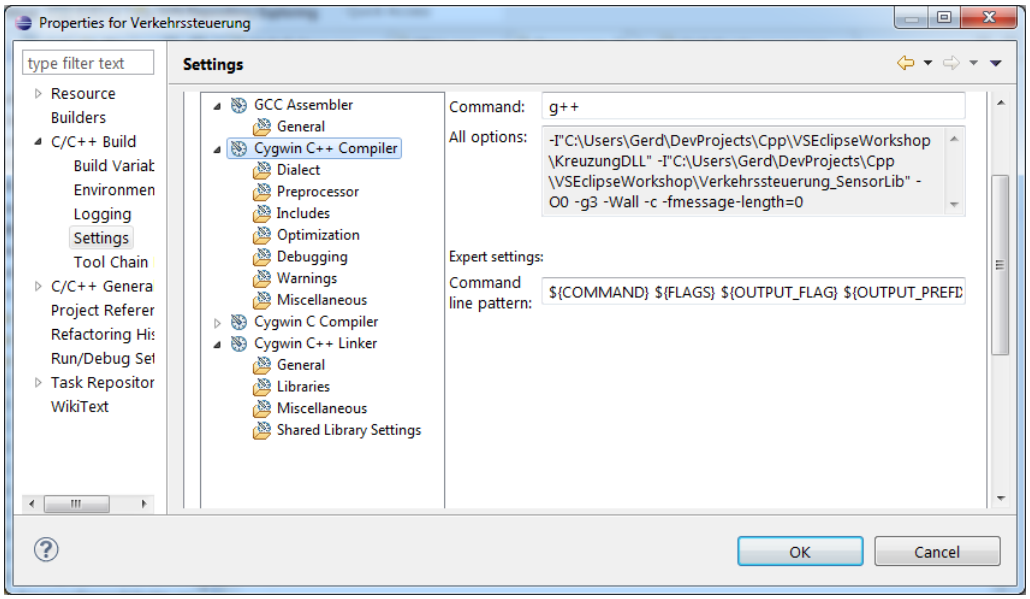

Abbildung 71: Build Configuration Settings All Options

Optionen, die keine graphische Repräsentation haben, können als Texteingabe in verschiedenen Feldern, eingestellt werden, z.B. in Abbildung 72 im Knoten *C/C++ Build.Settings.Tool Settings.Miscellaneous* im Feld *Other Flags* die Option `-std=c++11` um den Compiler anzuweisen, die neuen Sprachfeatures zu unterstützen.

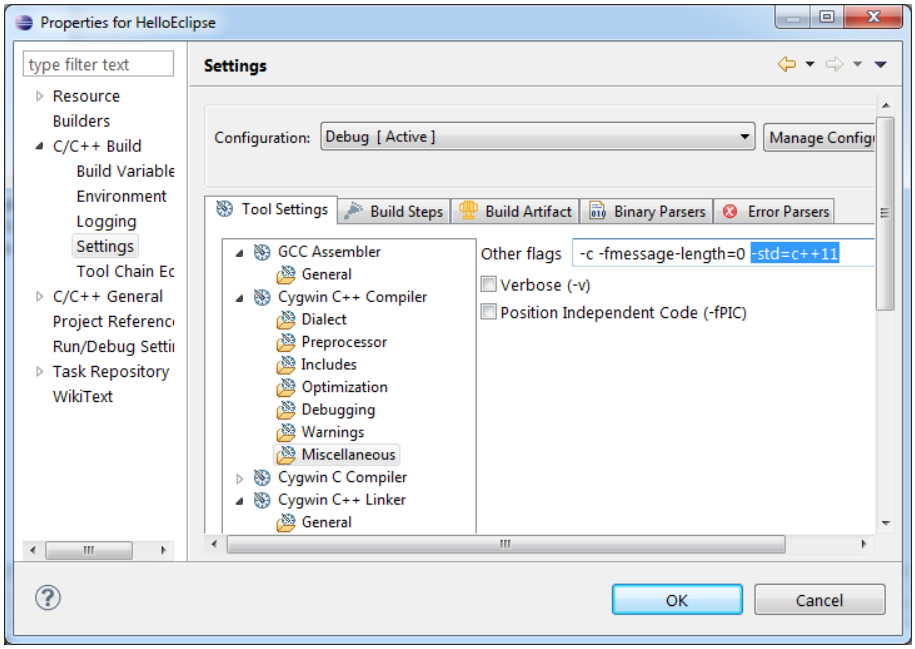

Abbildung 72: Build Configuration Settings ohne GUI

Für eine *Build Configuration* stehen im Tab *Build Artifact* drei Ergebnistypen, *Static Library*, *Shared Library* und *Executable* zur Auswahl zur Verfügung.
Tipp: Shared Libraries müssen für Unix Systeme mit dem Flag *-fPIC* übersetzt werden, auf Window Systemen ist es die default Einstellung für DLLs und führt zu einem entsprechenden Hinweis des Compilers.

10.8.7 Projekt mit externem Makefile

Projekte, deren Makefiles nicht bei jedem Build automatisch erzeugt werden sollen, können über den Dialog[48] in Abbildung 73 angelegt werden. Eclipse erzeugt

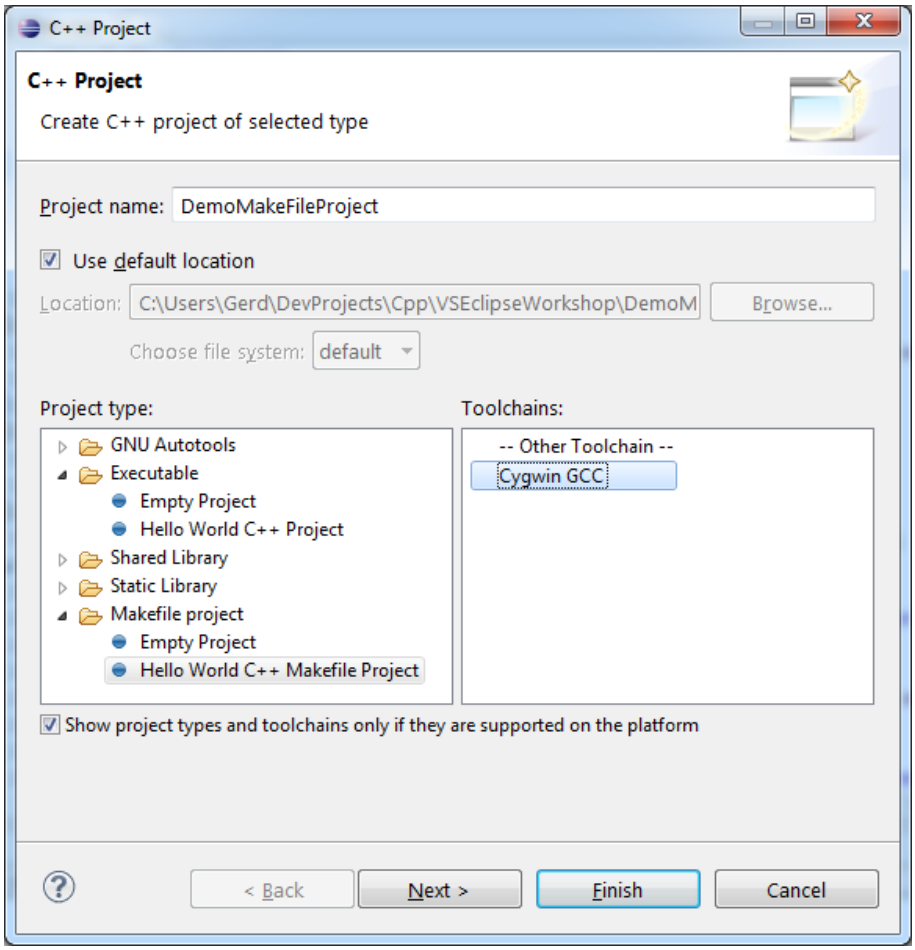

Abbildung 73: Make File Project

ein einfaches Makefile für das Projekt beim Anlegen. Änderungen und Erweiterungen der Resource Struktur und Einstellungen der Optionen für die verwendeten Tools müssen im Makefile gepflegt werden.

Die Make Targets können über die Make Target View aus section 10.8.8 auf Seite 78 verwaltet werden.

Bestehende Projekte, deren Build Prozess über ein make File verwaltet wird, können über das Kontext Menü *import.C/C++.Exisiting Code as Makefile Project* des

[48] siehe auch section 10.1 auf Seite 37

Project Explorers importiert werden.

Die Abbildung 74 zeigt den Dialog zur Erzeugung des neuen Eclipse Projekts auf der Basis der existierenden Sourcen. Die Dateien werden beim Import nicht

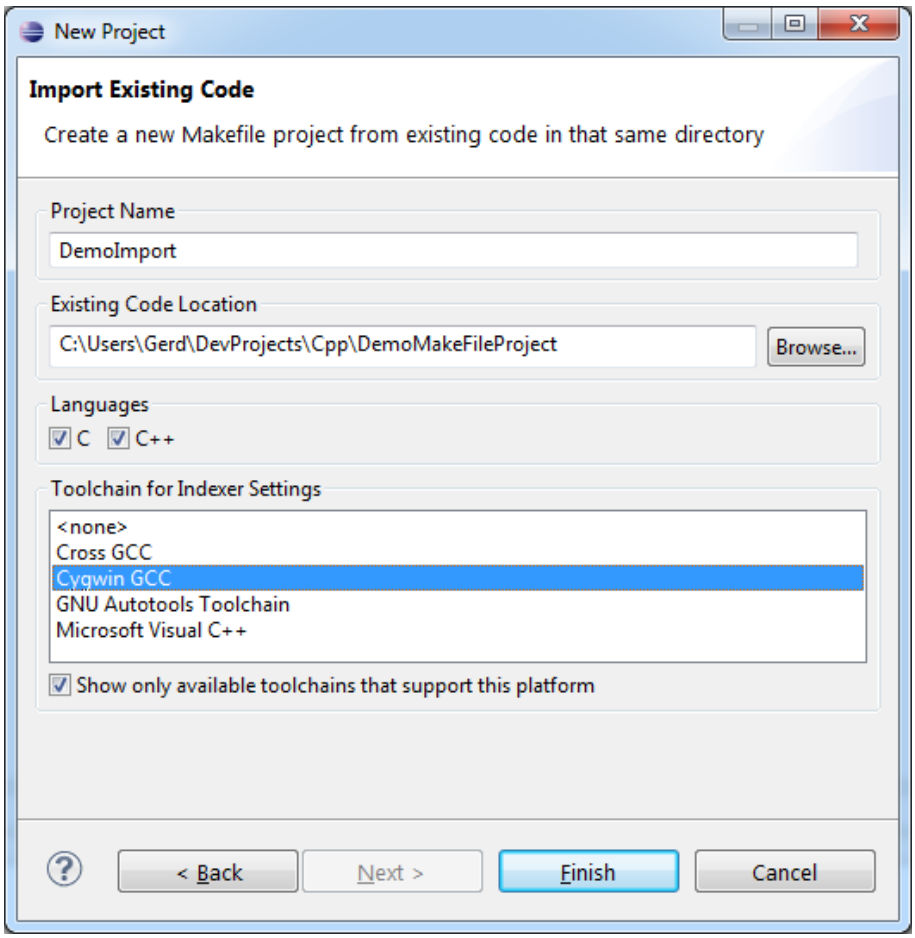

Abbildung 74: Import Existing Makefile Project

kopiert, es werden lediglich Verknüpfungen zu den Dateien hergestellt. Die Eclipse/CDT spezifischen Dateien (z.B. .cproject) werden im Projektverzeichnis erstellt. Für das Eclipse Projekt DemoImport wird kein Verzeichnis im Workspace erstellt.

Achtung: Wird das Projekt gelöscht, wird das Verzeichnis des Projekt gelöscht, wenn die Option *Delete project contents on disk* gewählt wird.

10.8.8 Make Target

Die Abbildung 75 zeigt links das Makefile, daneben die dazugehörige Outline View und die Make Target View. Die Make Target View zeigt die verschiedenen Targets, die aus den Make Files in die View übernommen oder über den Dialog in Abbildung 76 erfasst wurden.

Die Targets aus den Makefiles werden nicht automatisch in die View übernommen. Sie können über das Kontextmenü *Add make Target* eines Targets in der Outline View des Makefiles leicht in die Make Target View übernommen werden. In Abbildung 75 ist das Kontextmenü für das Target *clean* in der Outline View geöffnet.

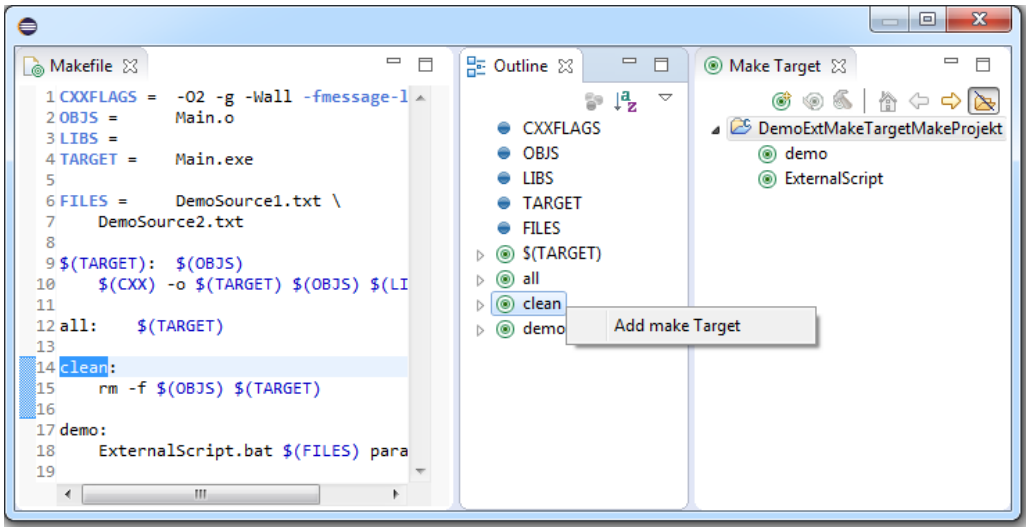

Abbildung 75: Die Make Target View

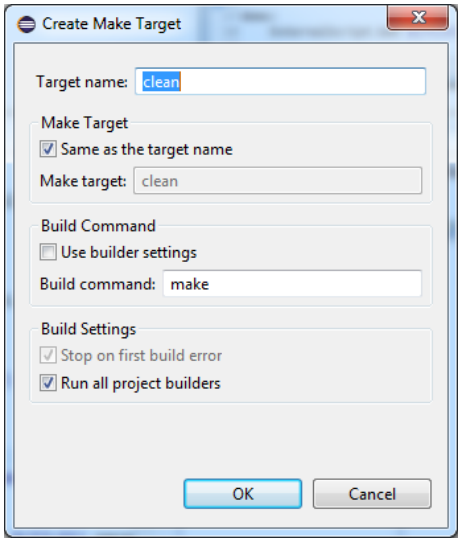

Abbildung 76: Ein neues Target aus einem Makefile

Der Dialog zur Bearbeitung eines Targets in der Make Target View ist über das Kontextmenü erreichbar. Über den Dialog können beliebige Programme und Scripte der Make Target View hinzugefügt werden. Über das Kontextmenü *Build Target*

oder einen **Doppelklick** können die Targets ausgeführt werden. Die in die View aufgenommenen Targets werden in der Datei *.cproject* gespeichert.

10.8.9 Externe Programme und Error Parser

In diesem Abschnitt soll am Beispiel eines externen Scripts dargestellt werden, wie

- externe Scripte in die Make Target View eingefügt werden
- externe Scripte über die Make Target View mit Parametern gestartet werden
- und wie deren Ausgaben durch einen selbst definierten Error Parser von Eclipse in der Console und der Problems View verarbeitet werden können

Die hier beschriebene Vorgehensweise ist nur in Projekten möglich, die **nicht den *internal Builder*** verwenden.

In Abbildung 77 ist der Bearbeitungsdialog des Targets *External Script* mit dem Namen des Scripts *Build command: ExternalScript.bat* und den Parametern (*Make target:*), die dem Script übergeben werden, zu sehen. Das Script muss in einem Verzeichnis liegen, der im Suchpfad für Ausführbares enthalten ist. Die

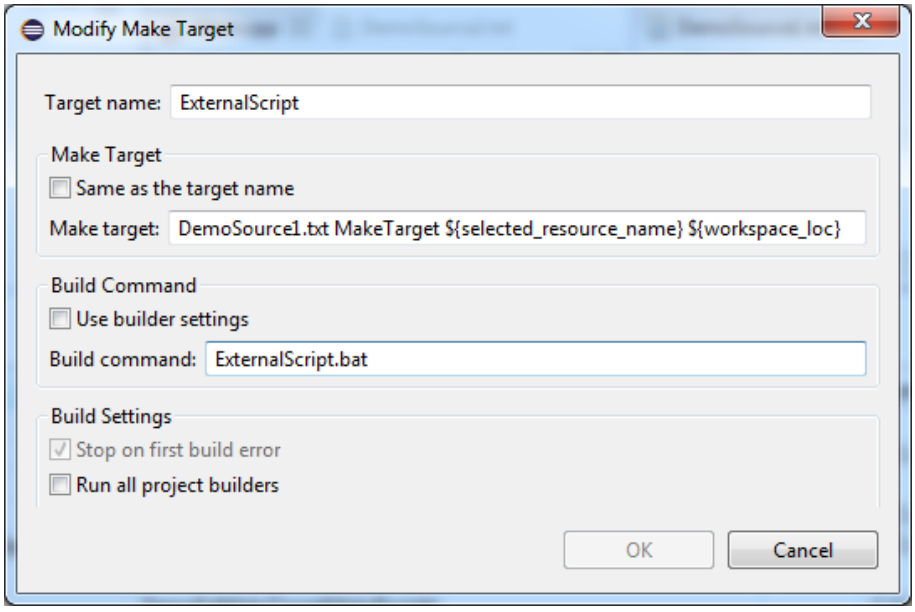

Abbildung 77: Make Target Dialog für das External Script

Parameter stehen in den Variablen %1, %2, usw zur Verfügung. Die Argumente ${workspace_loc} und ${selected_resource_name} sind sogenannte Build-Variablen[49]. Leider werden nicht alle expandiert. In der Ausgabe in Abbildung 78 auf Seite 81 ist der Pfad der sich aus ${workspace_loc} ergibt zu erkennen, die Variable ${selected_resource_name} wird aber nicht expandiert.

[49] siehe section 10.8.3 auf Seite 67

Listing 7: Das externe Script für cmd (Windows)

```
echo off
echo Demo mehrere Benutzer definierte Error Parser
echo %1.txt: 4 Beschreibung Source1
echo DemoSource2.txt: 2 %2 %3 %4
echo 10 Main.cpp Demo
```

Listing 8: Das externe Script für die Shell (Unix)

```
#!/bin/sh
echo Demo mehrere Benutzer definierte Error Parser
echo $1.txt: 4 Beschreibung Source1
echo DemoSource2.txt: 2 $2 $3 $4
echo 10 Main.cpp Demo
```

Listing 9: Ein Target mit einem Script

```
CXXFLAGS =  -O2 -g -Wall -fmessage-length=0
OBJS =      Main.o
LIBS =
TARGET = Main.exe

FILES =     DemoSource1.txt \
   DemoSource2.txt

$(TARGET):  $(OBJS)
   $(CXX) -o $(TARGET) $(OBJS) $(LIBS)

all:  $(TARGET)

clean:
   rm -f $(OBJS) $(TARGET)

demo:
   ExternalScript.bat $(FILES) param3 param4
```

Das ExternalScript.bat könnte auch von make aufgerufen werden, wenn im Makefile ein entsprechendes Target, wie in Listing 9 das Target *demo:* erstellt wird.

Die Abbildung 78 auf der nächsten Seite zeigt

- links oben den *Project Explorer* mit den Files des Projekts
- daneben die Make Target View mit dem Target ExternalScript
- die geöffneten Files *Main.cpp* und *DemoSource1.txt* mit den markierten Stellen. Durch einen **Doppelklick** in der Console oder der Problems View werden die Files geöffnet und die Zeile markiert.
- darunter die *Problems* View mit den aufgeklappten Ergebnissen des Error Parsers aus Abbildung 79 auf Seite 83
- darunter die *Console* mit der Ausgabe des Scripts aus Listing 7 das über das Target *External Script* ausgeführt wurde

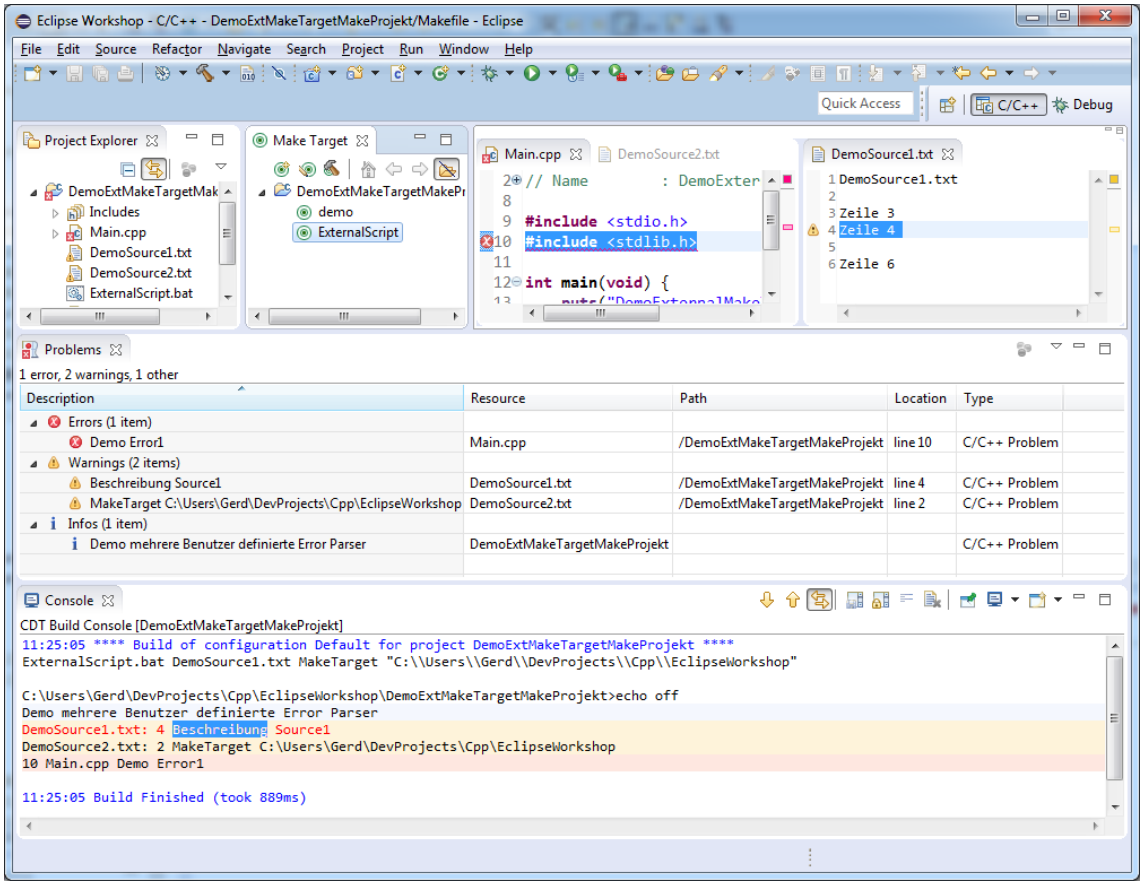

Abbildung 78: Die geparste Ausgabe und die Problems View

Die Abbildung 79 auf der nächsten Seite zeigt den Error Parser für die Verarbeitung der Ausgabe des External Scripts. Die Verwaltung von Error Parsern wird in section 10.8.5 auf Seite 70 beschrieben.

Der Error Parser ist ein regulärer Ausdruck / Regular Expression, der auf alle Ausgaben des Build Processes angewandt wird. Matched der Ausdruck, kann in der Console mit einem *Doppelklick* an die Stelle im Editor navigiert werden.

Das Script des Beispiels gibt folgendes aus:

- Demo mehrere Benutzer definierte Error Parser
- den ersten Parameter ergänzt um .txt: 4 Beschreibung Source1
- DemoSource2.txt: 2 ergänzt um die Parameter 2 bis 4
- 10 Main.cpp Demo

Die Werte für den Dateinamen, die Zeilennummer und die Beschreibung in Abbildung 79 auf der nächsten Seite stehen in den Variablen $1 bis $n in Abhängigkeit des regulären Ausdrucks zur Verfügung. Soll über die Console zu der jeweiligen Stelle navigiert werden können, muss der Dateiname und die Zeile in den korrespondierenden Spalten eingetragen werden. Soll die Zeile von anderen Error Parsern ebenfalls verarbeitet werden, muss in der letzten Spalte *Consum* ein *No* eingetragen werden.

Die erste Zeile der Ausgabe matched mit dem *Info* Error Parser
`(Demo .*)` der nur eine Variable $1 hat, in der die gesamte Zeile enthalten ist und die in der *Description* des Error Parsers verarbeitet wird. In der Problems View wird die Zeile unter dem Knoten *Infos* angezeigt.

Die beiden nächsten Zeilen matchen mit dem *Warning* Error Parser
`(.*): (\d+)(.*)` mit den Variablen $1 dem Dateinamen, $2 der Zeilennummer und $3 der Beschreibung, die in den entsprechenden Spalten des Parsers eingetragen sind. In der Problems View wird die Zeile unter dem Knoten *Warnings* angezeigt.

Die letzte Zeile matched mit dem *Error* Parser
`(\d+)(.*\...)(.*)` mit den Variablen $1 der Zeilennummer, $2 dem Dateinamen und $3 der Beschreibung, die in den entsprechenden Spalten des Parsers eingetragen sind. In der Problems View wird die Zeile unter dem Knoten *Errors* angezeigt.

10.8.10 External Tools

Externe Tools können über das Menü *Run.External Tools.External Tools Configuration* verwaltet werden. Den Tools können die Werte von Eclipse Variablen[50] übergeben werden aber leider wird die Ausgabe der Tools nicht durch die Error Parser[51] verarbeitet, wie das bei der Ausgabe eines Make Target[52] der Fall ist.

[50] section 10.8.3 auf Seite 67
[51] section 10.8.5 auf Seite 70
[52] section 10.8.9 auf Seite 79

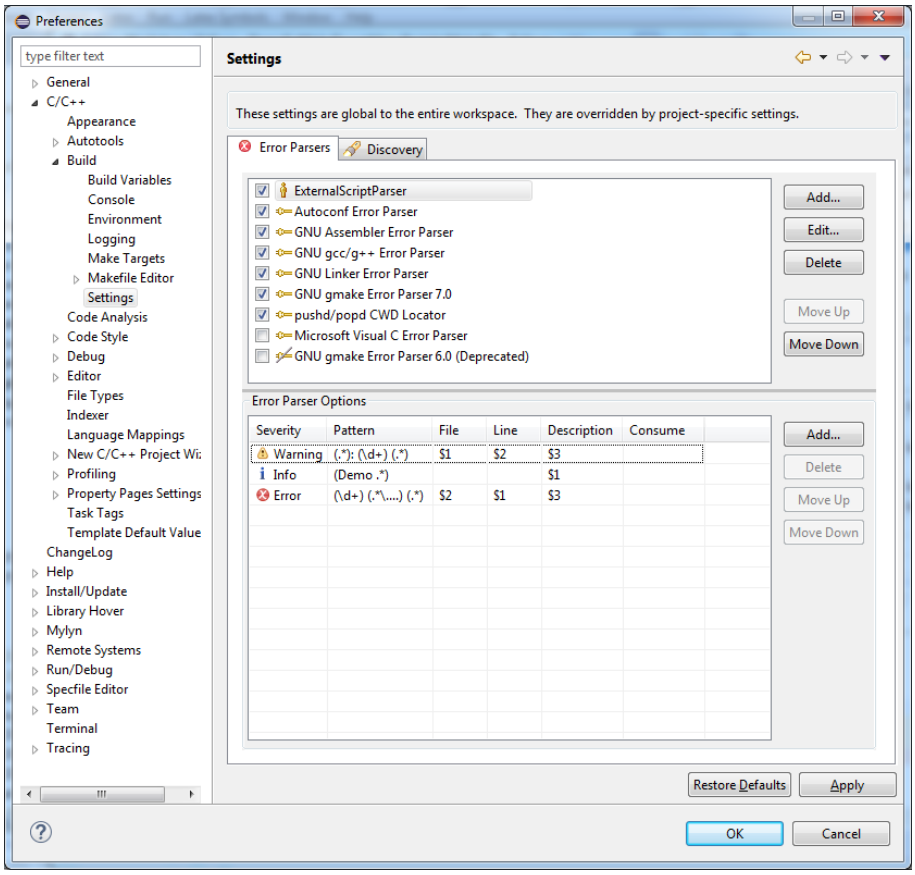

Abbildung 79: Ein Error Parser für die Ausgaben eines externen Scripts

Die Abbildung 80 auf der nächsten Seite zeigt den Bearbeitungsdialog für externe Tools mit dem Script aus section 10.8.9 auf Seite 79 als externem Tool. Dem Script wird als erstes Argument die Variable *${resource_name}* übergeben die den Namen der ausgewählten Resource enthält.

Die Abbildung 81 auf der nächsten Seite zeigt die Ausgabe des Scripts mit der ausgewählten Resource: *Makefile*.

10.8.11 Run Configuration

Eine Run Configuration (Dialog in Abbildung 82 auf Seite 85) definiert

- *Name:* einen Namen für die Configuration (`Verkehrssteuerung KreuzungDLL`)
- *Main.C/C++ Application:* das Executable und die dazugehörige Build Configuration
- *Arguments*: die Programm Argumente die `main(...)` übergeben werden
- *Environment*: die Umgebungsvariablen (siehe Abbildung 66 auf Seite 70)
- *Common*: weitere Einstellungsmöglichkeiten

Der Dialog kann über

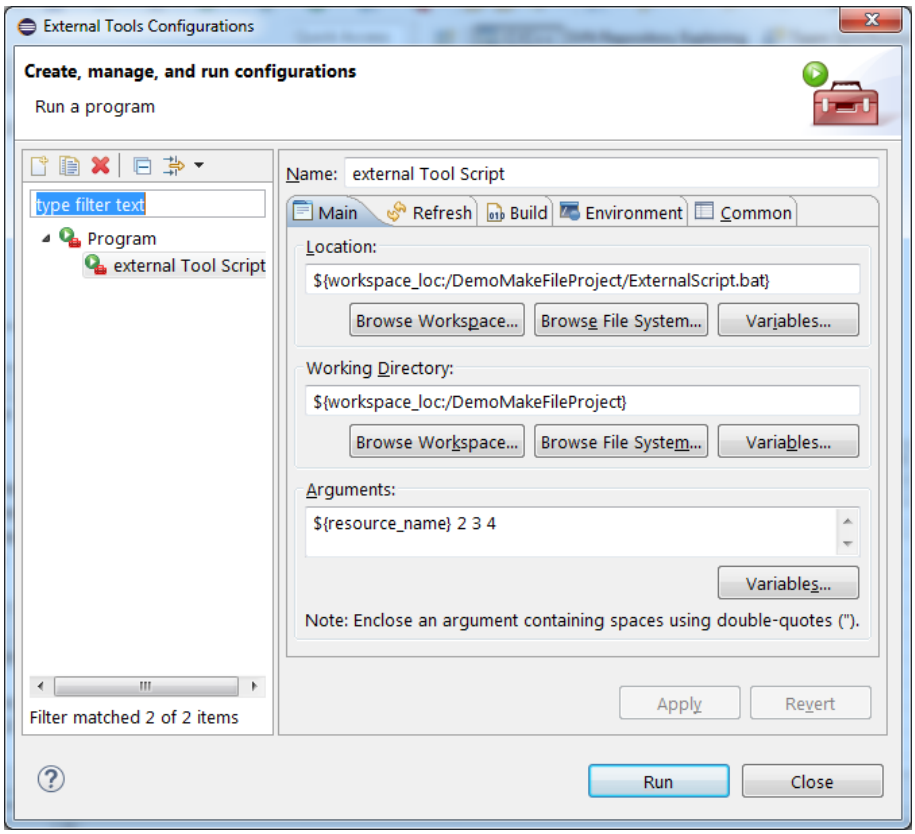

Abbildung 80: Configuration Dialog für Externe Tools mit Argumenten

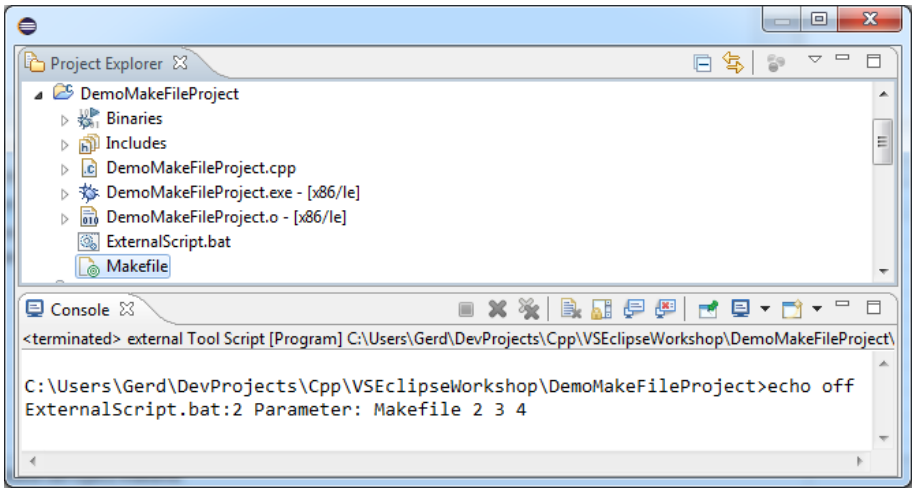

Abbildung 81: Die Ausgabe des ExternalScripts.bat

- das Menü *Run.Run Configurations...*
- das Pulldown Menü in der Toolbar *Run.Run Configurations...*
- das Kontextmenü des Projekts *Run As.Run Configurations...*

geöffnet werden. Eclipse/CDT legt automatisch eine Run Configuration beim ers-

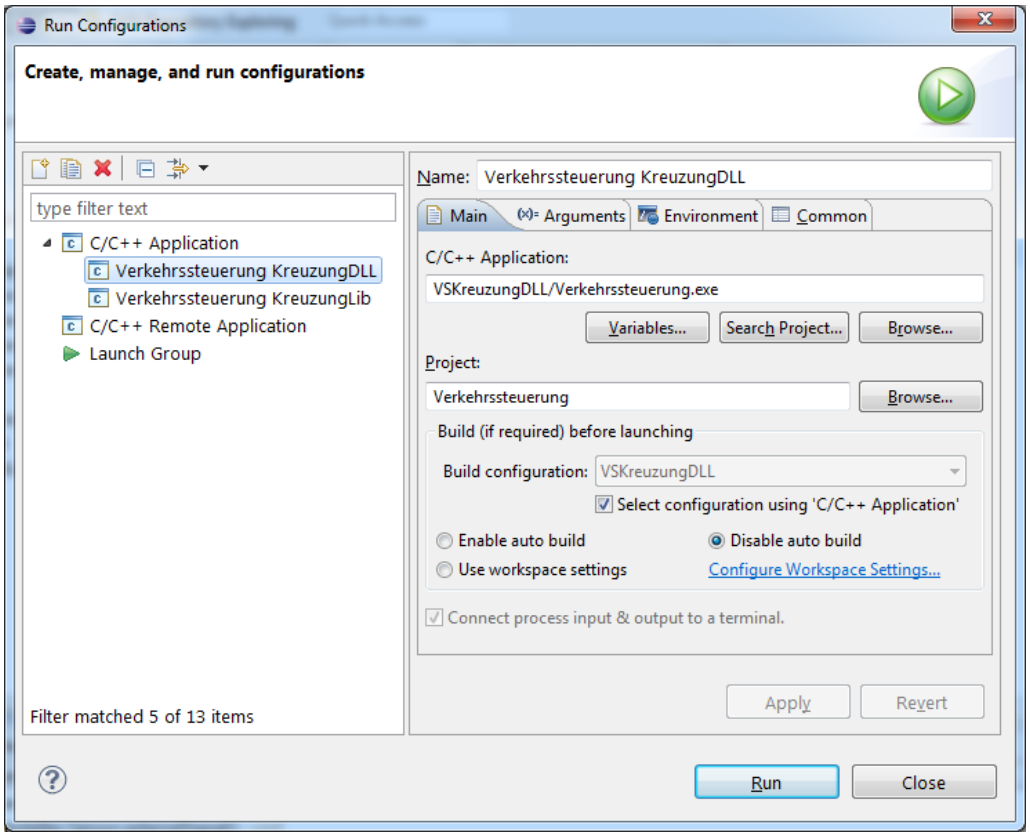

Abbildung 82: Die Einstellungen einer Run Configuration

ten Programmstart an.

Für dynamisch gelinkte Libraries muss in den Umgebungsvariablen der Pfad zu den shared Libraries gesetzt werden. Für Windows muss der Pfad zur DLL in der *PATH* Variablen enthalten sein, für Unix muss die Variable *LD_LIBRARY_PATH*[53] den Pfad zu der shared Library enthalten.

10.8.12 Debug Configuration

Eine Debug Configuration definiert dasselbe wie ein Run Configuration für eine Debug Session und weitere Debug spezifische Parameter.

[53] http://linuxtortures.blogspot.de/2012/02/shared-libraries-with-eclipse.html?showComment=1371197151984

11 Das CppUnit Framework

In diesem Kapitel wird die Installation und Verwendung des Test Frameworks CppUnit mit Eclipse beschrieben.

11.1 Installation

Ein Download der aktuelle Version steht unter
https://sourceforge.net/projects/cppunit/?source=typ_redirect zur Verfügung.

Eine gute Beschreibung der Installation für Window (MinGW/Cygwin) steht unter http://www.badprog.com/eclipse-cppunit-installation zur Verfügung.

Die Schritte zur Installation sind:

- Download der aktuellen Version des CppUnit Frameworks
- Entpacken in ein Installationsverzeichnis (`%CPPUNITDIR%`)
  z.B.: `cygdrive/d/DevProjects/cpp/cppunit/cppunit-1.12.1`
- ein Cygwin/MinGW Terminal öffenen und in das Verzeichnis wechseln,
  z.B.: `cd cygdrive/d/DevProjects/cpp/cppunit/cppunit-1.12.1`
- `./configure` ausführen
- `make` ausführen
- `make install` ausführen

Man sollte etwas Geduld mitbringen, das Ganze dauert eine ganze Weile.

11.2 Eclipse Projekt für CppUnit

Das Framework kann als statische oder dynamische Bibliothek verwendet werden. Um die Binaries zu erzeugen kann ein Eclipse Projekt mit entsprechenden *Build Configuration*s verwendet werden. Ich empfehle für dieses Projekt einen eigenen Workspace *CppUnitWorkspace* im Verzeichnis cppunit anzulegen. Das Projekt wird als leere static Library mit dem Namen *CppUnit* angelegt: *new.C++ Project.Shared Library.Empty Project*. Die *Build Configuration* für die dynamic Library kann anschließend erstellt werden.

11.2.1 Import der Sourcen

Für den Import der Sourcen des Frameworks, wird ein Verzeichnis *New.Source Folder* mit einem beliebigen Namen z.B. *cppunit* im Projekt angelegt. Über das Kontext Menü des Verzeichnisses werden die Sourcen aus dem Installationsverzeichnis `%CPPUNITDIR%/src/cppunit` von CppUnit importiert: *Import.General.File System*. Nach der Auswahl des Verzeichnisses `%CPPUNITDIR%/src/cppunit`, *Select*

All und *Finish*, werden die Sourcen importiert. Anschließend kann die statische Bibliothek über das Kontext Menü *CppUnit.Build Project* erstellt werden.

11.2.2 Die Build Configuration

Die Abbildung 83 zeigt die Einstellung zur Erzeugung des Frameworks als dynamisch Bibliothek (Dynamic Link Library DLL). In diesem Dialog kann in der Combobox *Artifact Type:* der Ergebnistyp für die jeweilige *Configuration* z.B. *Static Library* ausgewählt werden.

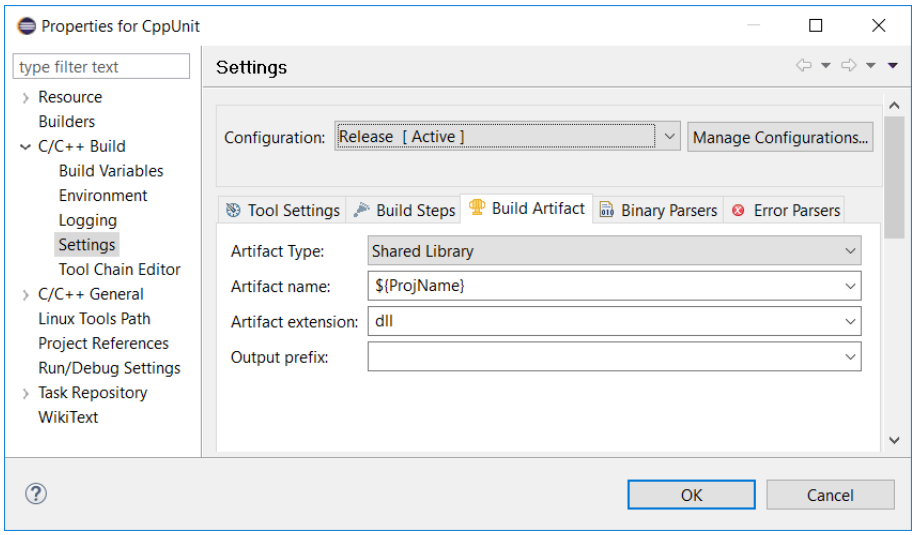

Abbildung 83: Einstellungen für die Build Ergebnisse

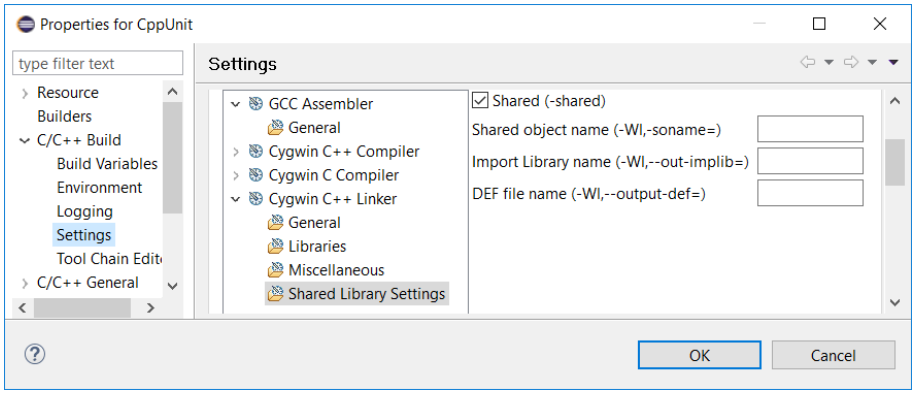

Abbildung 84: Shared Library Linker Properties

Die Abbildung 84 zeigt die notwendigen Einstellungen für den Linker. In dem Dialog muss die Checkbox *Settings.Cygwin C++ Linker.Shared Library Settings.Shared* ausgewählt sein. Diese Einstellungen gibt es für statische Libs nicht.

11.2.3 Pre- und Post- Build Steps

Um die erzeugten Bibliotheken aus allen Projekten nutzen zu können, sollten sie in einem allgemeinen Verzeichnis liegen, in dem der Linker aller Projekte sie erwartet. Mit einem *Post build steps Command* können die Erzeugnisse in ein solches Verzeichnis verschoben werden. Die Abbildung 85 zeigt eine mögliche Einstellung für das Projekt CppUnit im Knoten *C/C++.Settings.Build Steps*. Hier können Programme eingetragen werden, die vor (`Pre-build steps`) oder nach (`Post-build steps`) dem Build ausgeführt werden sollen. Die Beschreibung wird leider nur von make, dem *external Builder*, in der Console angezeigt.

Die Umgebung, in dem die Programme ausgeführt werden, ist eine Unix bash Shell. Das aktuelle Verzeichnis in dem die Programme ausgeführt werden, ist das Verzeichnis der jeweiligen *Build Configuration*, in Abbildung 85 das Verzeichnis *Release*.

Der Befehl `cp` kopiert das Ergebnis des Builds `CppUnit.dll` in ein allgemeines Verzeichnis: `cp ./CppUnit.dll /cygdrive/c/Programming/cygwin/bin`.
Der Name der statischen Bibliothek ist `libCppUnit.a`, der Eintrag in der *Build Configuration*, die die statische Bibliothek erzeugt, ist entsprechend anzupassen.

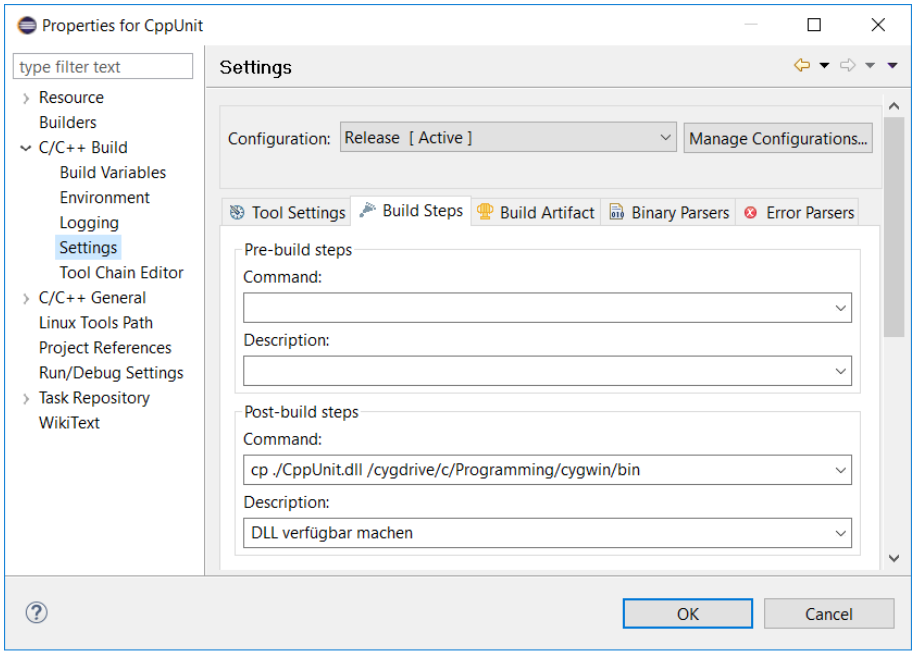

Abbildung 85: Pre- und Post- Build Steps mit external Builder

11.3 Die Anwendung des Frameworks

Unhabhängig davon, ob das Framework als DLL oder als statische Bibliothek erzeugt wird, ist die Konfiguration der Anwendung, die das Test Framework nutzt, immer gleich. In der *Build Configuration* muss dem Präprozessor das Include Verzeichnis des Frameworks mitgeteilt werden: *C/C++ Build.Settings.Tool Settings.includes* wie Abbildung 86 auf der nächsten Seite und dem Linker das Ver-

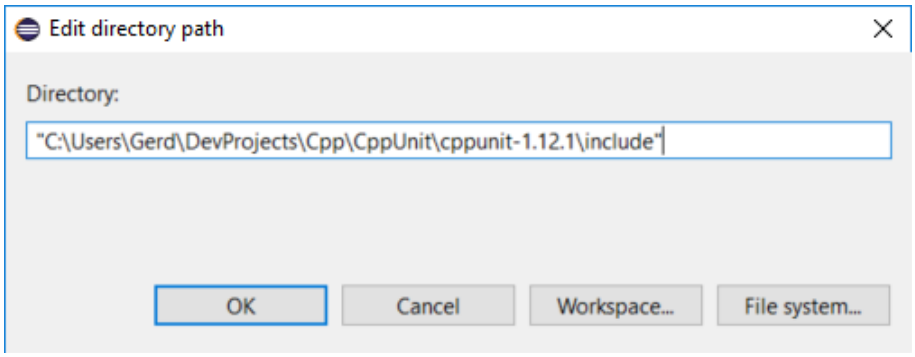

Abbildung 86: Der Include Pfad aus dem *File system...*

zeichnis und der Name der Bibliothek wie in Abbildung 87 gezeigt.

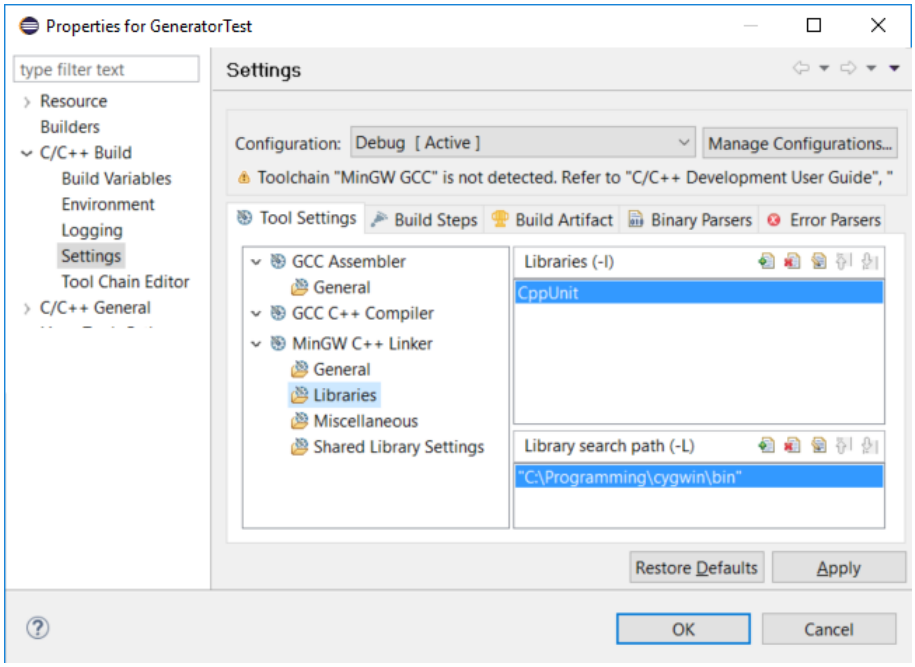

Abbildung 87: Die Linker Einstellungen der Anwendung für das CppUnit Framework

12 Das C++ Unit Test Easier (CUTE) Framework

In diesem Kapitel soll die Installation und Verwendung des Test Frameworks CUTE mit Eclipse beschrieben werden, soweit dies notwendig ist.

12.1 Allgemeines

CUTE stellt ein Plugin für Eclipse und eine eigene View für die Ergebnisse zur Verfügung. Eine umfangreiche Dokumentation findet sich auf der Website des Frameworks: http://cute-test.com/

12.2 Installation

Die Installation erfolgt über den eclipse Marketplace. *Help.Marketplace...* Die Abbildung 88 zeigt das CUTE Plugin auf dem eclipse Marketplace.

Abbildung 88: CUTE auf dem Eclipse Marketplace

12.3 Die Anwendung des Frameworks

Nach der Installation stehen in den Menüs CUTE spezifische Einträge zur Verfügung. Eine detailierte Beschreibung, wie damit gearbeitet werden kann findet sich auf der Website des Frameworks.

Teil IV

Zukünftige Erweiterungen

13 Zukünftige und erledigte Themen

13.1 Letzte Erweiterungen

- CppUnit Framework
- Dokumentation zu Eclipse/CDT
- Expressions View und Übung dazu
- Übung zu Suchen
- Make Target überarbeitet
- Externe Programme und Error Parser überarbeitet

13.2 Breakpoint Actions

Actions verwalten, zuordnen zu Breakpoints, Typen von Actions

13.3 Crosscompiling und Debugging auf dem Target

13.4 Konfigurationsmanagement mit RTC

13.5 UML Diagramm für Builder und Configuration

Zusammenhang Builder und BuildConfigs darstellen

Teil V
Übungen

14 Bedienung

14.1 Neues Projekt anlegen

Führen Sie die Schritte, Projekt erzeugen, übersetzen, ausführen und debuggen wie sie im Kapitel section 10.1 auf Seite 37 beschrieben sind, durch.

14.2 Perspective zurücksetzen

Führen Sie folgende Schritte aus:

- Schließen Sie alle Views in einer Perspective
- Öffnen Sie die Navigation View
- Setzen Sie die Perspective zurück: Kontextmenü der Perspective in der Toolbar *Reset*

Die Perspective sollte wieder in ihrem ursprünglichen Zustand sein.
Was können Sie bzgl. der Editoren beobachten?

14.3 Perspective konfigurieren

Führen Sie folgende Schritte aus:

- Öffnen Sie die *Debug* Perspective
- Fügen Sie die View *Project Explorer* hinzu
- Ziehen Sie die Variables View aus dem Mainwindow von Eclipse, so dass ein eigenes Fenster für die View erzeugt wird
- Speichern Sie die Perspective unter dem Namen “MyDebug”
- Öffnen Sie eine andere Perspective
- Schließen Sie alle Editoren
- Öffnen Sie die Perspective “MyDebug” wieder

Die Perspective sollte wieder in dem gespeicherten Zustand sein. Die Editoren werden nicht berücksichtigt.

14.4 2 Dateien Vergleichen

- Legen Sie eine Kopie von HelloWorld.cpp an (**Strg c, Strg v**)
- Führen im Orginal Änderungen durch
- Wählen Sie in einer Navigation View (z.B. Project Explorer) beide Dateien aus (erste auswählen, 2.te mit gedrückter **Strg** Taste)
- Im Kontextmenü den Eintrag *Compare With.Each Other* auswählen
- verschieben Sie die Änderungen in die andere Datei

14.5 Datei mit der local History Vergleichen

- Im KontextMenü des Orginals aus section 14.4 den Eintrag *Compare With.Local History...* auswählen

14.6 Suchen

- Wählen Sie das Projekt Sensor aus im Project Explorer aus
- Öffnen Sie den Dialog zur Datei übergreifenden Suche wie in Abbildung 15 auf Seite 29
- Geben Sie den Klassennamen AmpelState ein
- Wählen Sie *Scope.Selected resources* aus
- Aktivieren Sie die Suche mit *Search*

Das Ergebnis sollte leer sein, da in dem ausgewählten Projekt keine Klasse mit diesem Namen existiert.

Ändern Sie den Scope auf *Scope.Workspace* und führen die Suche erneut aus. Navigieren Sie zu einer beliebigen Stelle an der die Klasse gefunden wurde. Enfernen Sie den Eintrag im Suchfenster.
Auf diese Weise können Sie übergreifende Änderungen, die nicht durch das Refactoring automatisiert unterstützt werden, manuell durchführen und alle Fundstellen bearbeiten.

14.7 Open Element

Öffnen Sie das Element `class AmpelState` mit dem in section 10.5.3 auf Seite 45 beschriebenen Dialog.

14.8 Preferences nach Schlüsselworten durchsuchen

Im Dialog *Window.Preferences* nach allen Schlüsselworten suchen die bisher genannte wurden.

Tipp: Die Anfangsbuchstaben genügen schon.

- Temp
- File
- Container
- Folder
- Project
- Resource
- Workspace
- Perspective
- Window
- View
- ...

Suchen Sie die Einstellung für

- default tool chain
- die Anzahl der files der local historie.

14.9 Pin Properties

Lassen Sie sich die Property View von zwei verschiedenen Resourcen anzeigen. Öffnen sie dazu die Property View und ziehen Sie diese aus dem Hauptfenster von Eclipse, so dass die View in einem eigenständigen Subwindow angezeigt wird. Wählen Sie eine Resource aus und “pinnen” die View an die Resource. Öffnen Sie eine weitere View über das Pull down Menü in der Property View und ordnen Sie diese neben der ersten im selben Subwindow an. Wählen Sie dann eine andere Resource im Project Explorer aus.

Die Properties der beiden Resourcen werden nebeneinander angezeigt.

14.10 Expressions View

Eine Übung mit Debug Session.
Öffnen Sie das Projekt *DemoExpressionsView* und führen sie *close unrelated Projects* aus dem Kontext Menü des Projekts aus.
Starten Sie den Debugger.

Ordnen Sie die Fenster und Views ähnlich wie im Script an.
Tragen Sie in der Expressions View die Expression
`game.printValues()` aus dem Script ein.
Steppen Sie durch die Anwendung (Step Over F6).
Beobachten Sie die Veränderung der Expressions View und der Ausgabe.
Experimentieren Sie mit globalen Funktionen, mit Klassenmethoden (**`static`**) und arithmetischen Ausdrücken in der Expressions View.

14.11 Code Analyse und Erweiterung

Erweitern Sie das Beispiel Verkehrssteuerung aus section **??** auf Seite ?? um die gewünschten Kommunikationsprotokolle und die verschiedenen Verhaltensweisen der Ampel, wie sie in section **??** auf Seite ?? beschrieben sind.

Erforschen Sie dazu mit Hilfe der Views "Include Hierarchie", "Call Hierarchie", "Type Hierarchie" und allen anderen Ihnen bekannten Mittel zur Navigation im Code, das System.

Wo müssen welche Änderungen durchgeführt werden?

Teil VI

Anhang

Literatur

[Bau10] BAUER, Sebastian: *Eclipse für C/C++ Programmierer, Handbuch zu den Eclipse C/C++ Development Tools (CDT)*. 2., aktualisierte und erweiterte Auflage. dpunkt.verlag, November 2010. – ISBN 978–3–89864–715–1

[Vog13a] VOGEL, Lars: *Eclipse 4 RCP, The complete guide to Eclipse application development*. Second edition. Lars Vogel, 2013. – ISBN 978–3–94374–707–2

[Vog13b] VOGEL, Lars: *Eclipse IDE, Java programming, debugging, unit testing, task management and Git version control*. Third edition. Lars Vogel, 2013. – ISBN 978–3–94374–704–1

Abbildungsverzeichnis

Diagrammverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Weitere nützliche Quellen

Tabelle 1: Weitere nützliche Quellen

Beschreibung	Verweis
Eine Online Reference für die C++ Standard Library	http://en.cppreference.com/w/Main_Page
Fragen und Antworten zu verschiedenen Themen	http://stackoverflow.com/
UML Diagramme	http://www.uml-diagrams.org/
Eclipse	http://eclipse.org/downloads/
Eclipse Wiki	http://wiki.eclipse.org/Main_Page
CDT Dokumentation	www.eclipse.org/cdt/documentation.php
Help für Juno, Kepler, usw	help.eclipse.org/juno/index.jsp

Index