C/C++ Development Toolkit User Guide

This guide provides instructions for using the C/C++ Development Toolkit (CDT) in the workbench.

Before you begin

You must install and configure the following utilities:

- Build (such as `make`).
- Compile (such as gcc). For more information, see http://gcc.gnu.org.
- Debug (such as gdb). For more information, see http://sources.redhat.com/gdb/.

Tip: Cygwin contains these utilities for a Windows environment. For more information, see http://www.cygwin.com.

To test if cygwin was installed correctly open a command prompt and type `g++` or `make`.

The following error message means that no `make` is installed or your path is not configured correctly.

```
'g++' (or 'make') is not recognized as an internal or external
command, operable program or batch file
```

To check your path open a command prompt and type `PATH`. Make sure that the path to your build utility is defined (example `PATH=c:\cygwin\bin`).

Related reference

CDT Home

What's new in the CDT?

What's New in 2.0

Enhanced Performance

Builds can now be performed in the background.

Searches can now be performed in the background.

Automatic Project Settings Discovery

Automatically generate project defines and include paths settings from the **C/C++ > New Make Projects > Discovery Options** project settings.

C/C++ File Types

Define specific files, especially C++ headers without extensions, using the **C/C++ File Types** global preferences or project property.

Multiple Architecture Project Support

Building from multiple binary formats? Choose the appropriate formats using the **Binary Parser** project option.

Editor Hyperlink Naviagation

Enable the editor hyperlink navigation and then you can use **Ctrl+click** to jump to the declaration of an item on the C/C++ editor.

Indexer Error Markers

Enable C/C++ indexing and indexer error reporting in the C/C++ Indexer properties. This helps identify projects missing path configuration information.

Rename Refactoring Support

Use the Outline or the C/C++ Editor **Refactor** > **Rename** context menu to refactor class & type names, methods, function & member names.

Open Type

Use Open Type to open up the declaration of C/C++ classes, structures, unions, typedefs, enumerations and namespaces.

Automatic Refresh

Configure the default behavior of the automatic retrieval of shared library and register information in the C/C++ debugger.

Improved Managed Make

You can now set the compiler command for managed projects.

Improved Standard Make

Standard Make now parses response from Make command to populate paths and symbols.

Support for GNU

Now supports some of the GNU extensions to the ANSI specification.

Improved View and Browsing Features

You can now open Include files from the Outline View.

You can now perform selection searches from the C/C++ Editor

Improved Search

Search now supports external files referenced using `#include`.

Makefile Outline View

You can now browse the structure of your Makefile in Outline View.

Content Assist Enhancements

Content Assist now produces proposals from classes and structure members, local & global variables, pre-processor defines, pre-processor commands.

Content Assist now supports C++.

What's New in 1.2

Find out what's new in CDT 1.2.

C/C++ Search

You can search the projects in your workspace for references to, declarations or definitions of, particular elements. Only header files referenced by a source file in your workspace are included in a search.

Build

Managed build

You can now create a Managed build and have makefiles generated for you.

Error parser

The error parsers are now extension-points that can contributed by other plug-ins. The error parsers are used to detect errors or warnings or informational messages from compilers, linkers, etc... during a build.

New Global preferences for all Standard Make Project properties

Make builder options

Support changing/enabling default make targets for each workbench build type. New default build location setting.

Make project options

You can now make changes the Error Parser Configuration. You can change the order in which error parsers are applied or disable them entirely.

You can also specify which paths to include during a Make and customize preprocessor symbols to ensure proper indexing and search capabilities.

Notes:
There are a number of "build error parsers" (the things that turn compiler error messages into objects that we can put into the error lists). If one parser cannot figure out what the message means, then the system moves to the next one in the list.

Make target

Make targets now support Stop on error build option and ability to change the default build command.

New Standard Make projects

Old Standard Make projects will be automatically updated to support the new options. If update is declined, then selecting Update Old Make project... from the context menu of the project will update the project to a new Standard Make project.

Debug

Formatting of variables and expressions

You can now select the number system (natural, decimal, hexadecimal) used to display variables and expressions.

Variable view, detail pane

In the Variable view, a detail pane has been added to let you see the value of a selected variable. This is practical when looking at a string (char *).

Casting of variables, expressions and registers

In the Variable view, a variable can be cast to a different type or be restored to its original type. Also, a pointer can be cast to an array type.

Debug disable variable query

The value of variables are queried at every step.

This can be time-consuming on certain embedded targets. The automatic query of variables can be disabled. Manual queries are now an option.

Source location

A new source locator in the Run/Debug dialog box makes it possible to add directories to search, mapping, and the order of the search.

GDB/MI new shared library launch pane

For GDB/MI code, two new actions are added in the launch view, stop-on-solib and auto-load-symbols. Stop-on-solib will force the debugger to stop on any shared library events. Auto load will load the symbols for any shared library.

GDB/MI improvements in display of arrays

Arrays are now separated into ranges, to limit the possibility of a timeout on large arrays.

What's new for previous releases

You can keep track of previous release-specific developments in the CDT.

For more information, see **http://www.eclipse.org/cdt/ > CDT Project Management/Plans**. The **Official CDT Plans** section lists previous releases.

Getting Started

How to bring C/C++ source into Eclipse
Updating the CDT

Related reference
CDT Home

How to bring C/C++ source files into Eclipse

A common scenario that you may encounter when starting to use the CDT, is determining how to bring existing C/C++ source files into Eclipse. There are a number of ways to do this. The scenarios described below are recommended approaches.

Create a project from CVS

If your existing source tree is managed in CVS, you can use the CVS Repository perspective to "Checkout As..." any folder in the repository. The first time you "Checkout As...", the New Project wizard is launched and you need to create a C or C++ project for the folder. For more information, see Creating a project and Working with C/C++ project files.

A CVS checkout of the project into the project's location occurs. It is recommended that you eventually add and commit the CDT project files back into CVS. The CDT project files include .project, .cdtproject and .cdtbuild (for Managed Build projects) and are located at the root folder of each CDT project.

Create new projects from existing source roots

If your resource code is not managed in CVS but is available from the file system, then you need to perform two steps:

1. Identify a "root folder" of your source code tree.
2. Create a new C/C++ project using the New Project Wizard, and specify the "root folder" as a non-default location of the new project.

Typically existing projects will have their own makefiles, so you should create a new Standard Make C/C++ project. For more information see Creating a project and Working with C/C++ project files.

To help you to identify a root folder for your project, consider the following guidelines:

- all source code for the project is available on or beneath the root folder
- the build results are also produced in or beneath the root folder
- there is often a makefile in the root folder. In complex projects, the makefile in the root folder calls other makefiles in other directories to produce the build results.
- external header files and library files do not need to be in or beneath the root folder.

The resources for the project are maintained in the remote location specified, not in the workspace folder for Eclipse. However, your existing folder structure is displayed in the C/C++ Projects view. Meta data

for the project, such as the index for the project and the link to the existing source, is stored in the metadata directory in the workspace folder. For more information on the workspace folder, see **Workbench User Guide > Tasks > Upgrading Eclipse**.

Once you create a CDT project, you cannot easily move it or redefine its root folders. If you need to, you can delete the CDT project (without deleting its contents) and then recreate it specifying a different non-default location.

Import your C/C++ source file system

Another approach would be to create a C/C++ Project and then import your existing file system. For more information, see **Workbench User Guide > Tasks > Importing > Importing resources from the file system**.

This approach copies the files from your file system to an Eclipse Workbench project or folder. Your original source files remain unchanged and it is the copies of the files that will be edited, built and debugged using the CDT. When you have successfully imported your existing file system, the folder structure is displayed in the C/C++ Projects view. Again, you should identify an appropriate "root folder" to import from.

Tip:

- Importing your existing file system can consume significant disk space depending on the size of your files.
- Your files may become detached from an existing source control system that previously referenced the original file location such as a ClearCase view.

Related concepts

Overview of the CDT

CDT Projects

Related tasks

Working with C/C++ project files

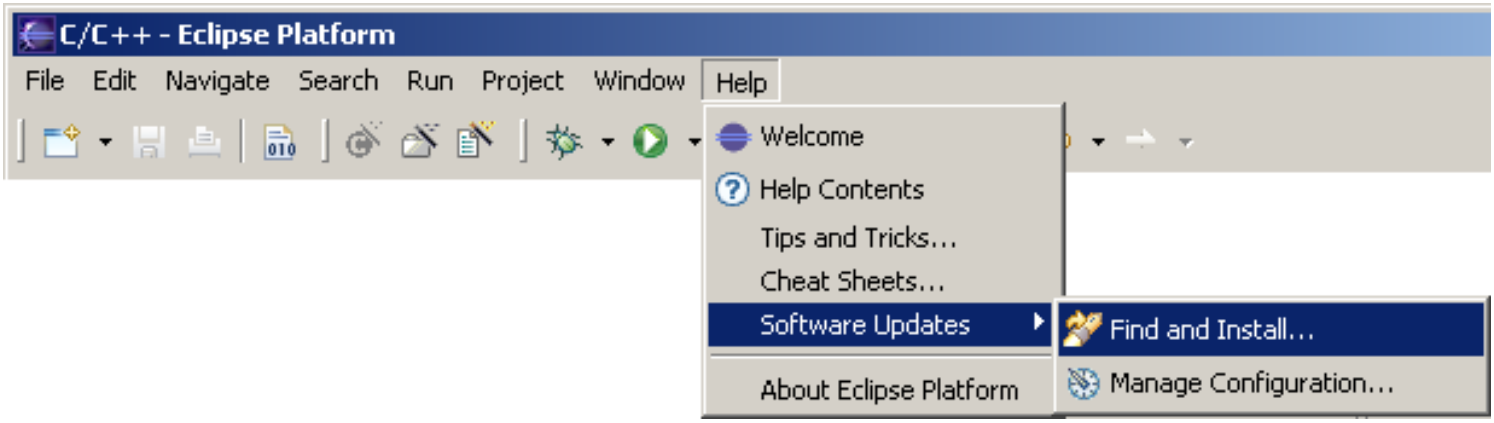
Related reference

Project properties

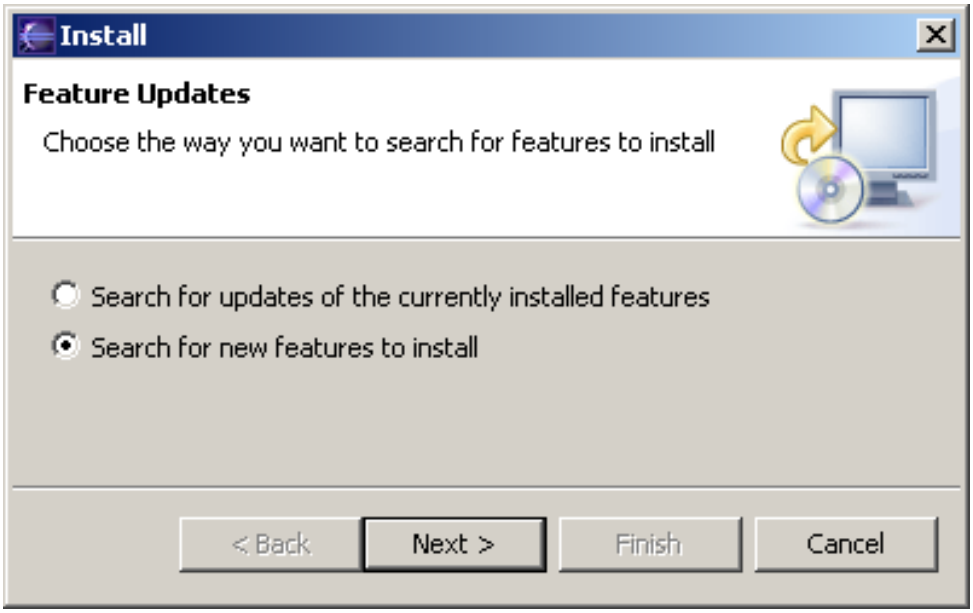
Updating the CDT

The CDT can be updated directly from the workbench using your internet connection.

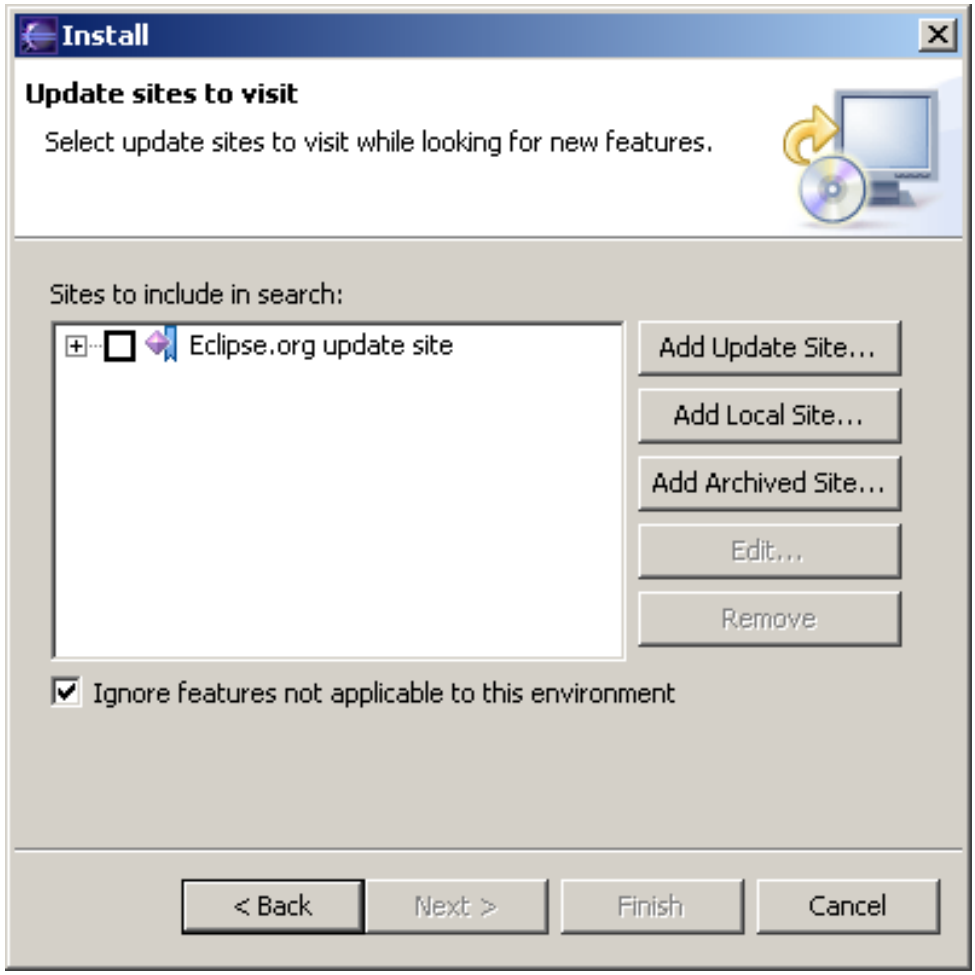
1. Click **Help > Software Updates > Find and Install**.



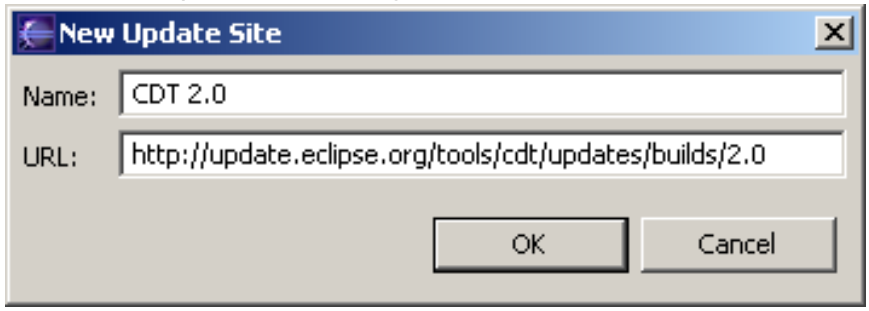
2. In the **Feature Updates** window select **Search for new features to install** and click **Next**.



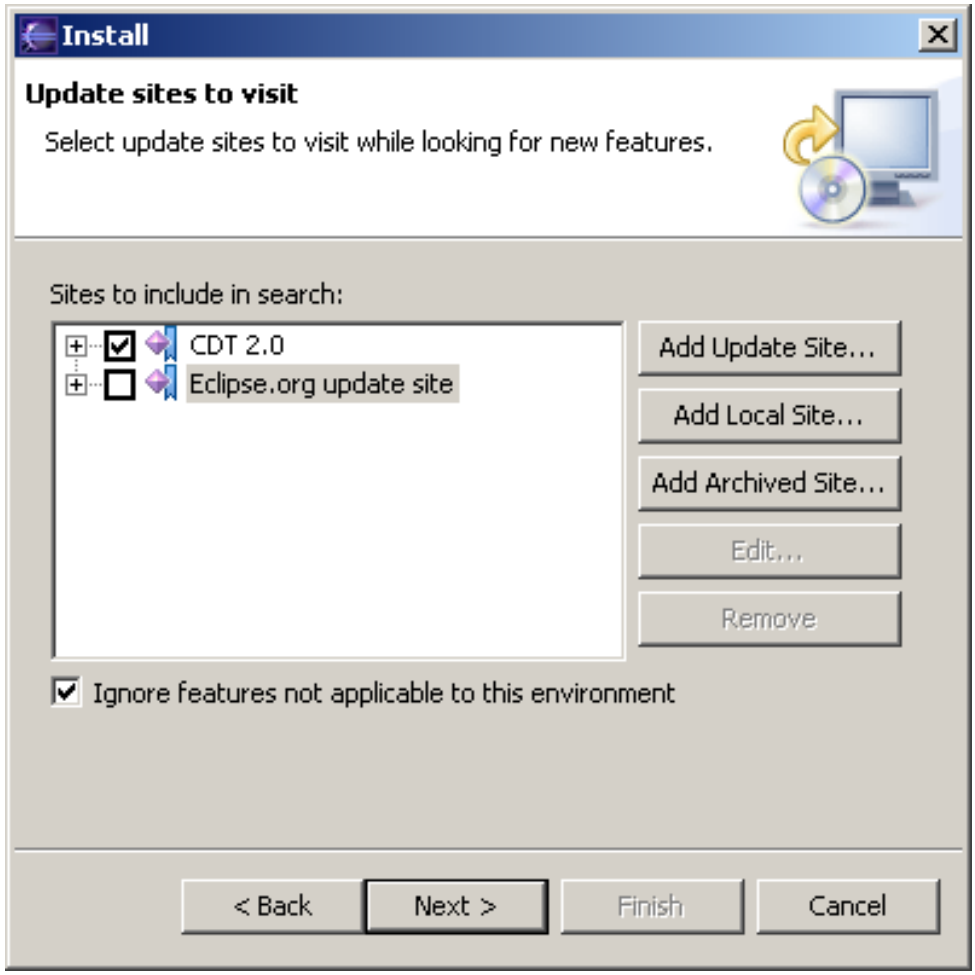
3. If you have not updated previously, you will have to enter the location of the CDT Install site. Click the **Add Update Site...** button.



4. In the New Update Site dialog box, enter a name and the URL for the update site in the spaces provided.

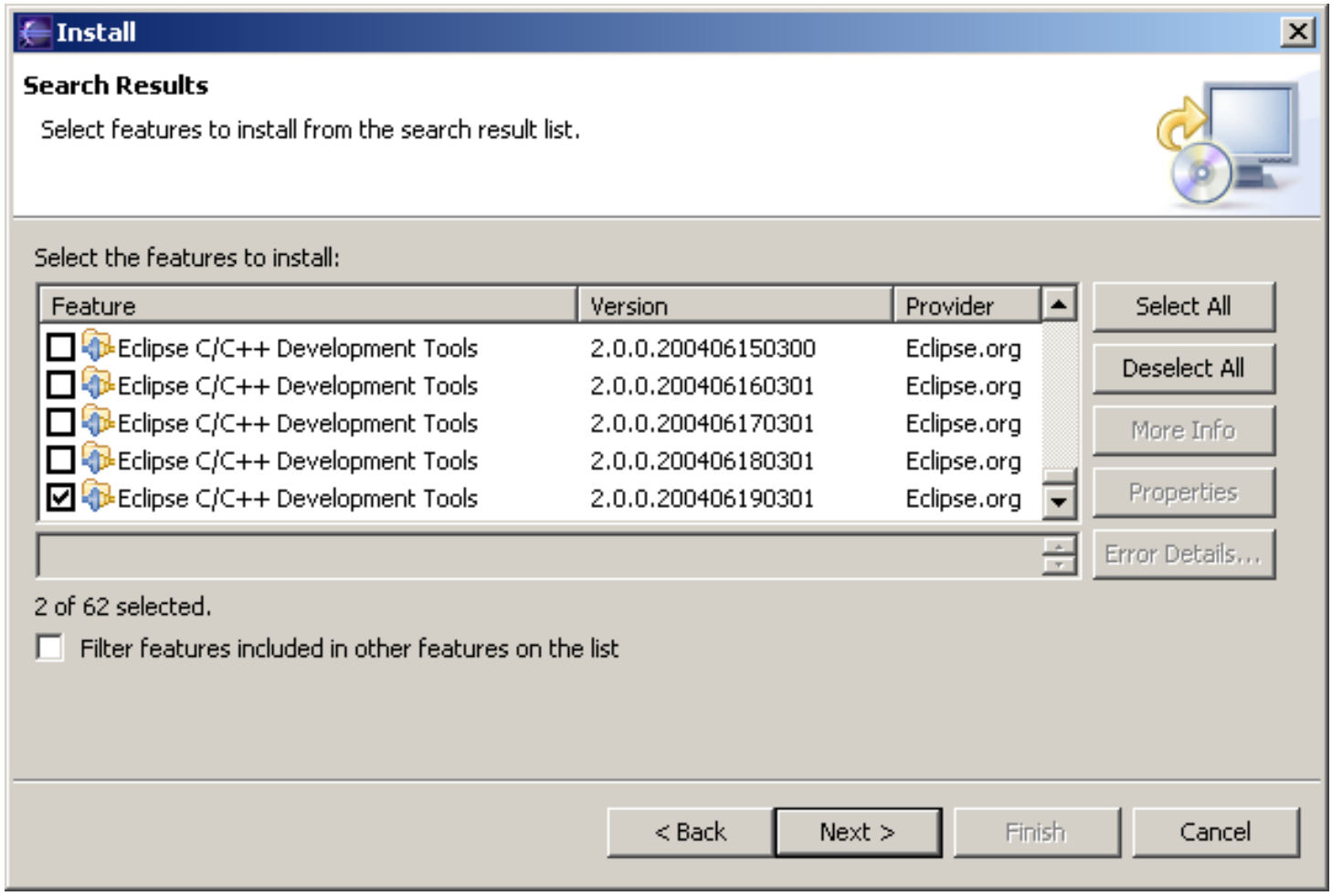


5. Select the update site you just created by clicking the appropriate checkbox and click **Next**.

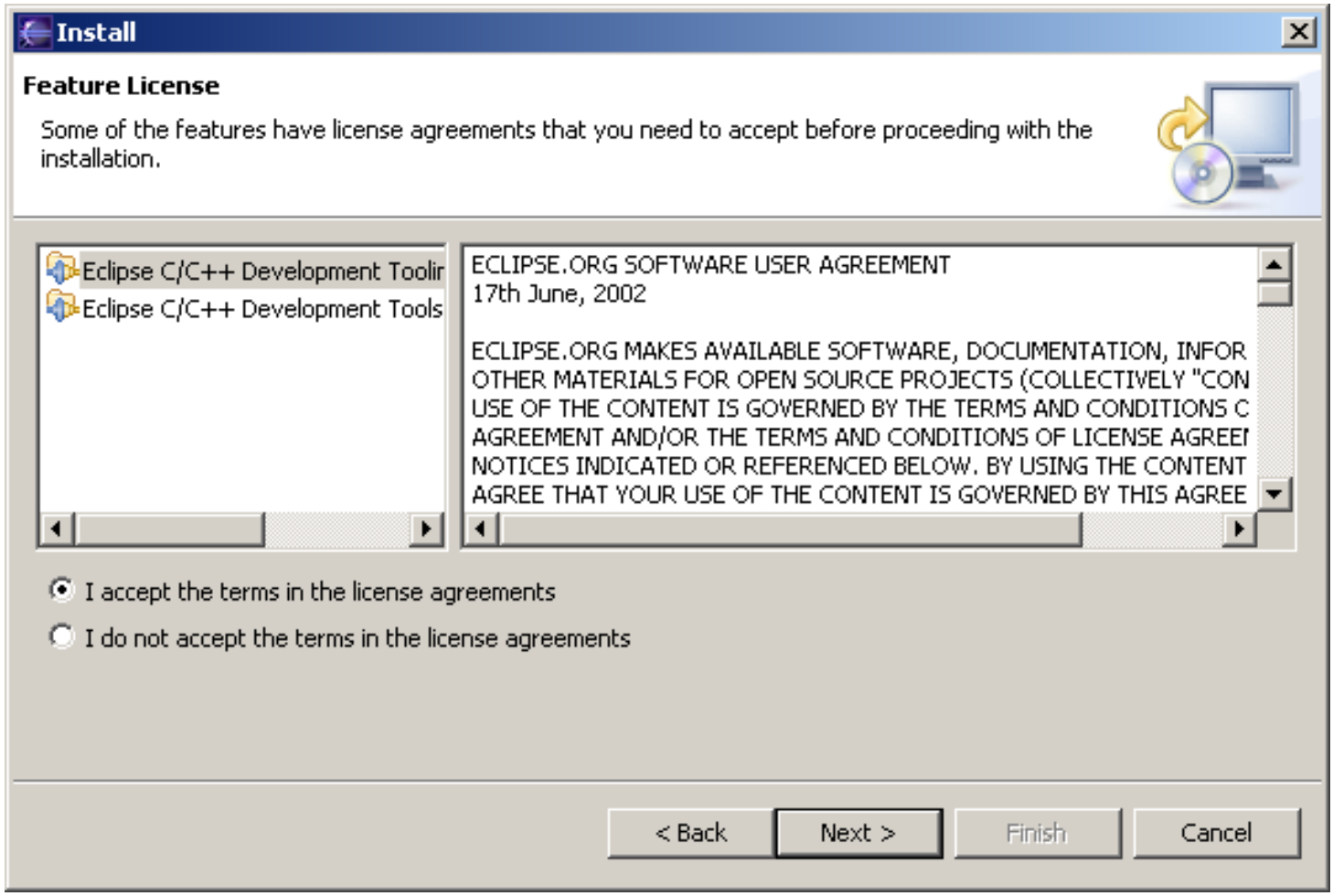


6. A dialog box will appear showing the updates available from the update site, select each of the following features, ensuring you have precisely the same version for each one:
    - Eclipse C/C+ Development Tooling SDK
    - Eclipse C/C+ Development Tools

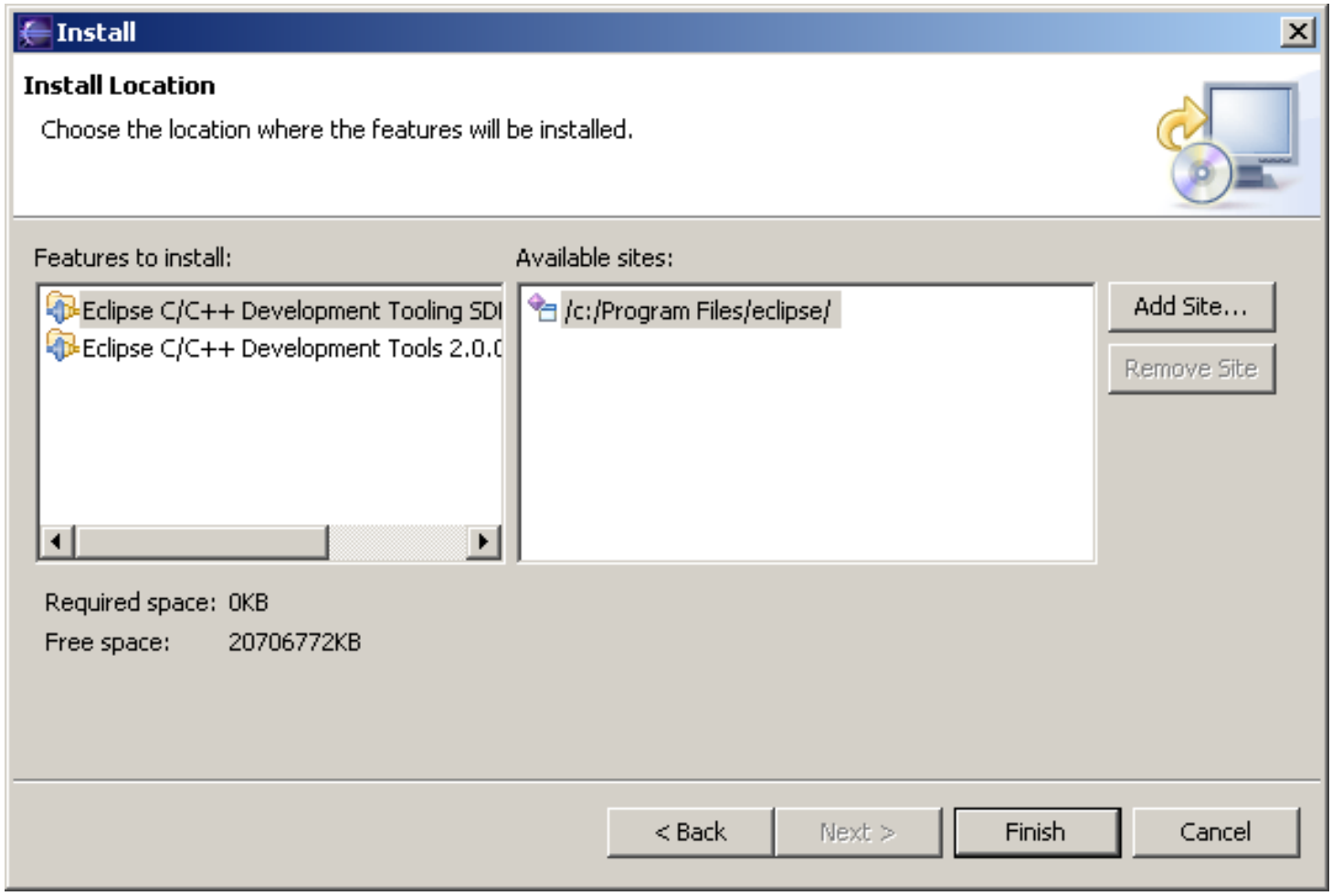
    Then click **Next**.



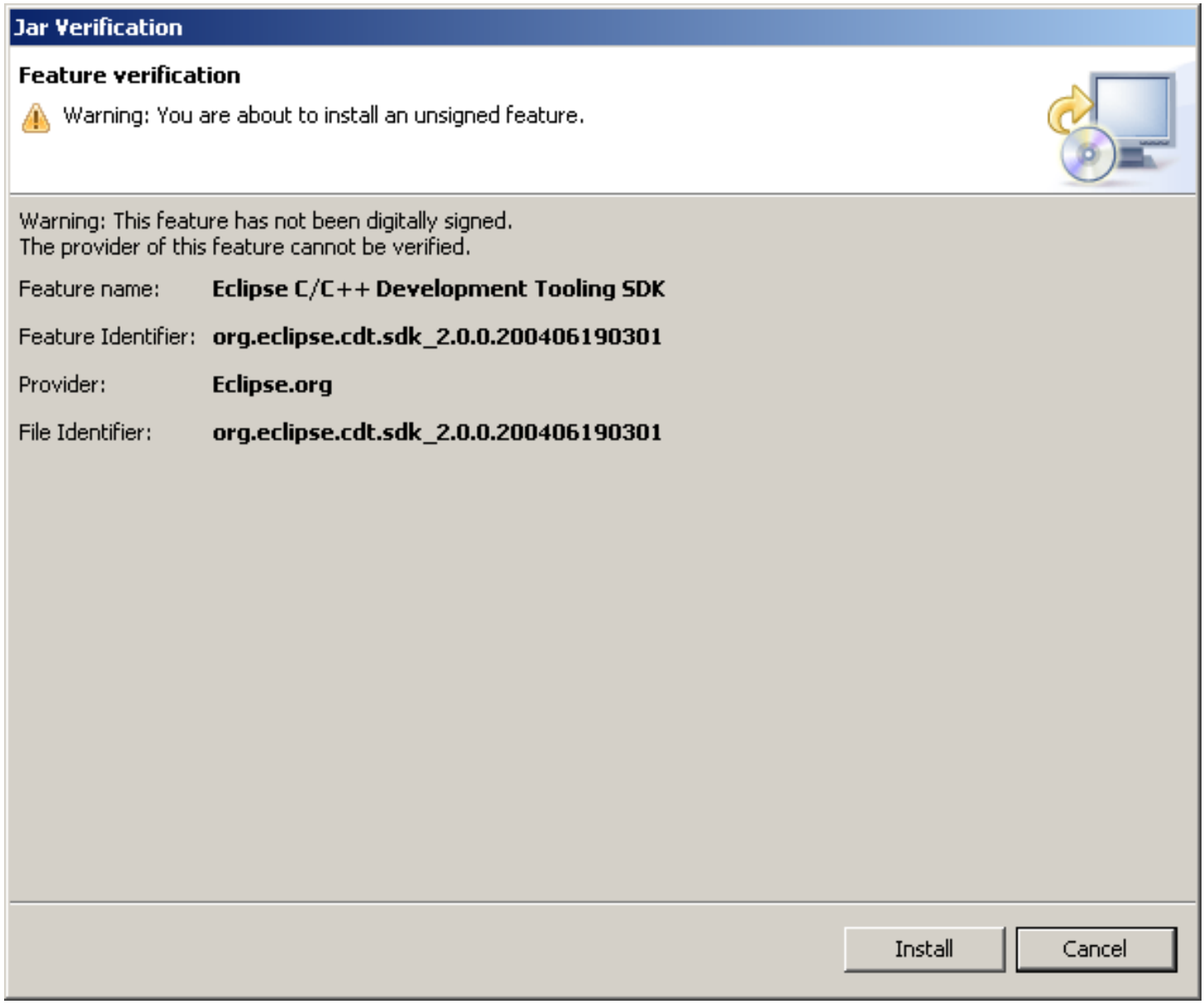
7. You should now see the Eclipse.org Software User Agreement, you must accept the agreement to install the CDT update. Do so by selecting **I accept the terms in the license agreement** and then click **Next**.



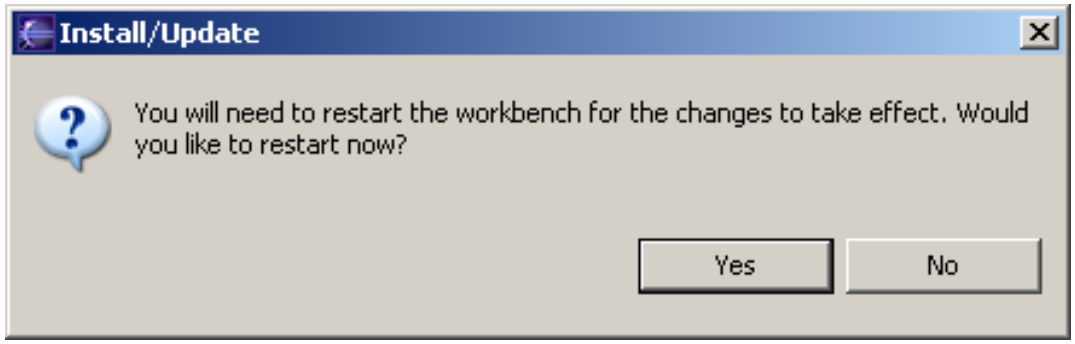
8. Now select the location you would like the updates installed, usually this is the directory where you installed Eclipse, and click **Finish**.



9. You will now start downloading the CDT components. You will have to verify that you want the features installed by clicking **Install** for each feature you selected.



10. You will now have to restart Eclipse, select **Yes** to complete the update.



Related concepts

CDT Overview

C/C++ Development perspective

Concepts

Provides background information for CDT components, tasks and objectives.

Searching External Files

CDT Overview

The C/C++ Development Toolkit (CDT) is a set of Eclipse plug-ins that provide C and C++ extensions to the Eclipse workbench. For more information about Eclipse, see **Workbench User Guide > Concepts > Workbench**.

The CDT provides a C/C++ IDE that simplifies many of the same tools that you can use from the command line. The CDT can also communicate with many external utilities and interpret their responses, for example:

- Build (such as make).
- Compile (such as gcc). For more information, see http://gcc.gnu.org.
- Debug (such as gdb). For more information, see http://sources.redhat.com/gdb/.

Note: while make, gcc and gdb are the examples used in the documentation, virtually any similar set of tools or utilities could be used.

The CDT opens as the C/C++ perspective of the Eclipse workbench. The C/C++ perspective consists of an editor and the following views:

C/C++ Projects
: Shows your C/C++ projects and files.It operates in much the same way as the **Navigator**.

Console
: Displays your program's output, as well as the output from your build and external tool chain.

Editor
: The C/C++ editor view provides specialized features for editing C/C++ related files.

Make Targets
: Enables you to select the make targets you want to build in your workspace.

Navigator
: Shows all of the file system's files under your workspace directory.

Outline
: Displays the structure of the file currently open in an editor.

Problems View
: If you encounter any errors during a build they will be displayed in the Problems view.

Properties
: Shows the attributes of the item currently selected in a view or an editor.

Search
: Shows the results of searches for files or text.

Tasks
: Lists tasks that you want to keep track of, either as a schedule of things to do or a history of

things that have been done.

For more information, see **Workbench User Guide > Concepts > Perspectives**.

CDT updates

The **Install/Update** wizard provides information about your current Eclipse installation and provides the framework to manage your updates. For more information, see **Workbench User Guide > Tasks > Updating features with the update manager**.

To view a list of the updates available for the toolsets that you installed, click **Help > Software Updates > New Updates**.

Additional information

For more information on the Eclipse CDT project, refer to http://www.eclipse.org/cdt/:

- CDT newsgroup: The place to ask questions about how to use the CDT.
- User FAQ: Provides answers to the most common questions about using the CDT.
- Developer Documentation: Provides feature and design specifications for building and extending the CDT.
- CDT Community Webpage: Showcases plug-ins and tools developed by and for the CDT Community. If you have tools or plug-ins that you would like to submit to the CDT Community Page, use the CDT Development Mailing List or the Eclipse Tools CDT newsgroup.

License

The CDT is an open source project and is licensed under the Common Public License.

Related concepts

Working with existing code

What's new

Related reference

Views

CDT projects

Before you can work in the CDT, you must create a project to store your source code, makefiles, binaries, and related files. C/C++ projects are displayed in the C/C++ Projects view.

Tip: Nested projects are not supported. Each project must be organized as a discrete entity. Project dependencies are supported by allowing a project to reference other projects that reside in your workspace. For more information, see Selecting referenced projects.

For more information about projects and where they are stored, see:

- **Workbench User Guide > Tasks > Resources**
- **Workbench User Guide > Tasks > Running Eclipse**

Project types

You can create a standard make C or C++ project or a managed make C or C++ project.

Standard make C or C++ project

You need to create a makefile in order to build your project or use an existing makefile.

Managed make C or C++ project

A managed make project generates the makefile for you automatically. In addition, the files module.dep and module.mk are created for every project sub-directory. These files are required for your managed make projects to build successfully.

Project conversion

You can convert projects from C to C++ (or from C++ to C). If, for example, your requirements change and you must convert an existing C project to C++, you can do this without recreating the project. The CDT converts your project files and resolves any source control issues.

A few notes about projects

- When you create a file within a project, a record (local history) of every change is created. For more information about local history, see **Workbench User Guide > Reference > User**

interface information > Development environment > Local history.

- Spaces in projects and filenames can cause problems with some tools, such as the make utility or the compiler.
- Be careful when you use only case to distinguish files and projects. UNIX-based operating system file names are case sensitive, but Windows filenames are not. Therefore, Hello.c and hello.c are separate files in UNIX but overwrite each other in Windows.

For more information about projects, see **Workbench User Guide > Concepts > Workbench > Resources**.

Related concepts

Project file views

How to bring C/C++ source into Eclipse

Related tasks

Working with C/C++ project files

Converting a C or C++ nature for a project

Related reference

Project properties

Views

Perspectives available to C/C++ developers

A perspective is a layout of views (development tools) in the Workbench window. Each type of perspective is a combination of views, menus, and toolbars that enable you to perform a particular task. For example, the C/C++ perspective has views that are organized to help you develop C/C++ programs; the **Debug** perspective has views that enable you to debug those programs.

Selecting / Opening Views:

- You can add views to a perspective. From the menu bar choose **Window > Show View > Other** and select a new view from the **Show View** dialog.
- To reset the current perspective to its original layout, from the menu bar choose **Window > Reset Perspective**.

The C/C++ development tools contribute the following perspectives to the workbench:

C/C++ perspective views

This perspective is tuned for working with C/C++ projects. By default it consists of an editor area and the following views:

- C/C++ Projects (the file navigator for C/C++ resources)
- Navigator (the file navigator for all Eclipse resources)
- Console
- Properties
- Tasks
- Make Targets
- Outline
- Search

Debug perspective views

This perspective is tuned for debugging your C/C++ program. By default it includes an editor area and the following views:

- Debug
- Variables
- Breakpoints
- Expressions

- Registers
- Memory
- Display (for use with JDT only)
- Outline
- Console
- Tasks

Other Perspectives

In addition to the perspectives named above and the Resource perspective (which you see when you first start Eclipse), Eclipse also has perspectives that are tuned to other types of development:

- Java
- Java Browsing
- Plug-in Development.
- CVS Repository Exploring

Related concepts

Views in the C/C++ perspective

Debug Concepts

Related tasks

Adding breakpoints

Related reference

Console view

Views in the C/C++ perspective

The C/C++ views are panels that help you perform the tasks involved in creating C/C++ programs. The **C/C++** perspective displays these panels in a single Eclipse window.

Changing Views:

- You can add views to a perspective. From the menu bar choose **Window > Show View > Other** and select a new view from the **Show View** dialog.
- To reset the current perspective to its original layout, from the menu bar choose **Window > Reset Perspective**.

The following views are commonly used in the **C/C++** perspective:

C/C++ Projects
: Displays, in a tree structure, only elements relevant to C and C++ projects.

Console
: Displays your program's output, as well as the output from your build tools.

Editor
: The C/C++ editor view provides specialized features for editing C/C++ related files.

Make Targets
: Enables you to select the make targets you want to build in your workspace.

Navigator
: Shows all of the file system's files under your workspace directory.

Outline
: Displays the structure of the file currently open in an editor.

Problems View
: If you encounter any errors during a build they will be displayed in the Problems view.

Properties
: Shows the attributes of the item currently selected in a view or an editor.

Search
: Shows the results of searches for files or text.

Tasks
: Lists tasks that want to keep track of, either as a schedule of things to do or a history of things that have been done.

Related concepts

CDT Overview

C/C++ perspectives

Related reference

Views

Coding aids

This section provides information on code entry aids.

Comments

Comments are lines in a source file that have been marked to be ignored by the compiler. Two styles of comments are supported by current C/C++ compilers:

- `/* text */`
- `// text`

Comment

You can quickly comment out one or more lines of code by inserting the leading characters `//` at the beginning of the line. To do so, select the line (or lines) of code you want to comment out and press **CTRL**+/ (slash).

Uncomment

To uncomment select the line (or lines) of code, and press **CTRL**+\ (backslash).

Tip: The characters `/* */` on lines that are already commented out, are not affected when you comment and uncomment code.

Multiline comment

You can use the Content Assist feature to insert a multi-line comment before a function. Type `com +Ctrl+Space`, and the following code is entered at the cursor location:

```
/*
 * author userid
 *
 * To change this generated comment edit the template variable
"comment":
 * Window>Preferences>C>Templates.
 */
```

To change the default comment click **Window** > **Preferences** > **C** > **Templates**. For more information see the Content Assist section.

Related concepts

Content Assist and code completion

Related tasks

Customizing the C++ editor

Commenting out code

Related reference

C/C++ editor, code templates and search preferences

Content Assist

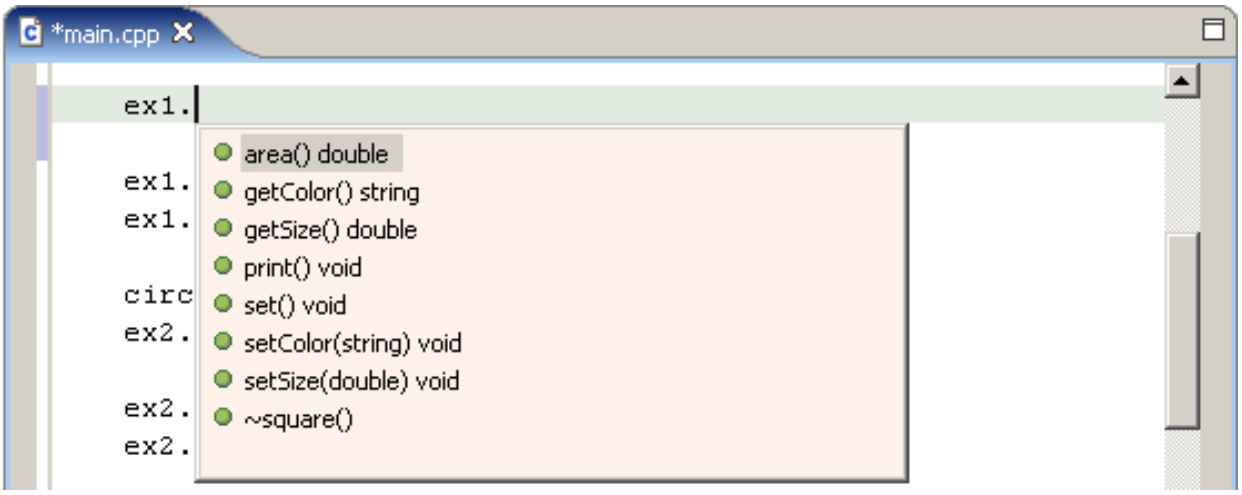
Content Assist is a set of tools built into the CDT that can reduce the number of keystrokes you must type to create your code. The Content Assist plug-in consists of several components that forecast what a developer will type, based on the current context, scope, and prefix.

Code completion

Content assist provides code completion anywhere in the document. For the current project a list is displayed of the elements that begin with the letter combination you entered, and the relevance of each proposal is determined in the following order:

- Fields
- Variables
- Methods
- Functions
- Classes
- Structs
- Unions
- Namespaces
- Enumerations

You trigger the Code completion feature when you call Content Assist (such as when you type `Ctrl+Space`), but it is auto-activated when you type "**.**", "**->**" or "**::**".



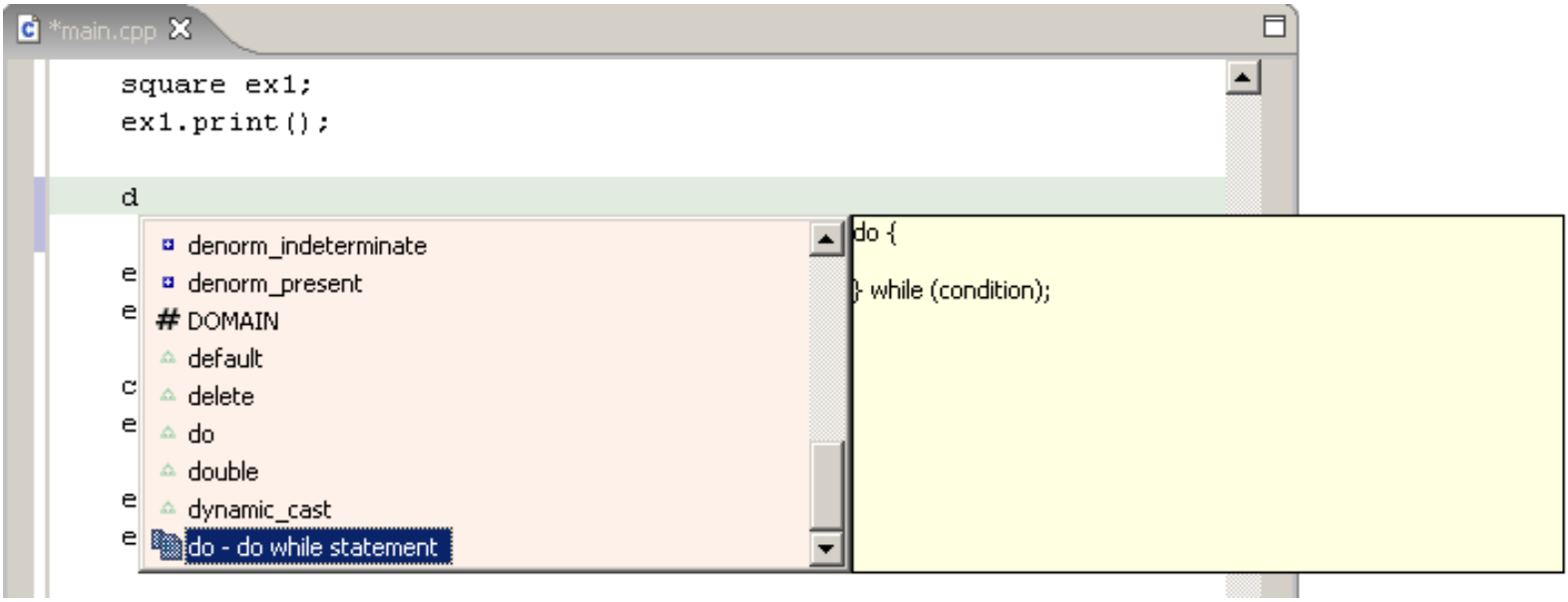
You can view the signature of each item on the list in a pop-up by pointing to it. You can then select an item in the list to insert it directly into your code.

Code templates

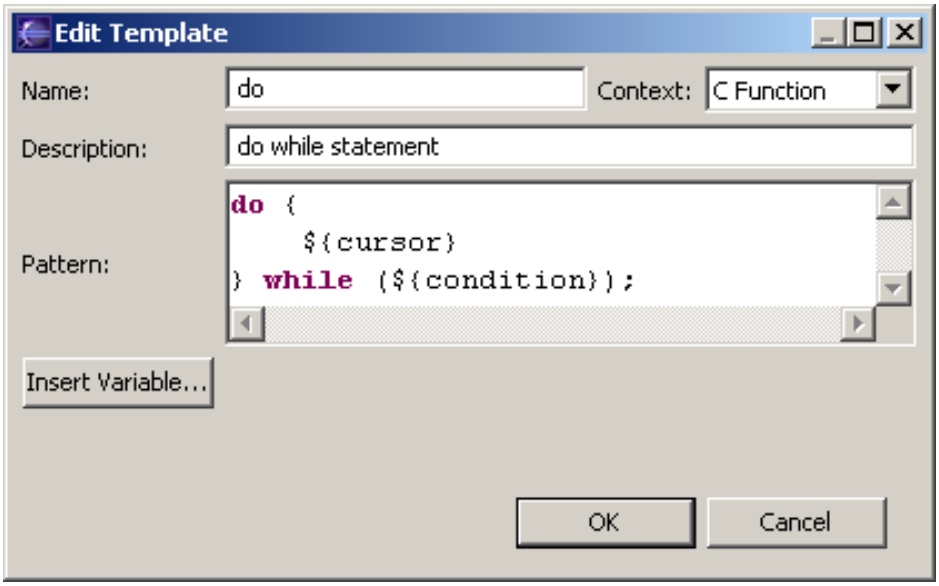
You can create and save code templates for frequently used sections of code, which will be inserted according to scope. The Content Assist feature also provides quick access to code templates.

When you enter a letter combination in the C/C++ editor, and type `CTRL+SPACE` (or right-click and click **Content Assist**), a list of code elements and code templates that start with the letter combination that you typed is displayed.

You can then select a code template from the list and it is inserted directly into your code.



For example, the code template `do while statement` contains the following code:



When you select the `do` code template from the list, you insert the following code:

```
do {
} while (condition);
```

If the completion engine finds only one proposal in your templates, that proposal is inserted at the current cursor position. For example if you create a new .cpp file and type `mai+CTRL+SPACE` the following code is inserted at the cursor location:

```
int
main(int argc, char **argv) {

}
```

No Completions

If you invoke Content Assist, but no completions are found a message will be displayed on the status to inform you that the Content Assist parser has timed out.

Related concepts

Code entry

Related tasks

Using Content Assist
Creating and editing code templates
Importing and exporting code templates

Related reference

C/C++ perspective icons

Templates

Templates are sections of code that occur frequently enough that you would like to be able to insert them with a few keystrokes. This function is known as **Content Assist**; the sections of code that are inserted are known as **templates**.

To input an existing Content Assist template into a file, such as one for an **if** statement, type the initial character ("`i`" in this case), then press **Ctrl+Space**. The templates that begin with that character appear. Double-click on a template to insert it into a file.

You can edit existing Code/Content Assist templates or create new ones. From the menu bar choose **Window > Preferences > C/C++ > Code Templates**.

Related concepts

CDT Overview

Related tasks

Creating and editing code templates

Using templates

Importing and exporting code templates

Related reference

Edit menu

Content Assist page, Preferences window

Code Templates page, Preferences window

Editing C/C++ Files

This section provides information on editing C/C++ files.

- C/C++ editor
- C++
- Makefile

C/C++ editor

The CDT provides an editor that gives you specific support for editing C/C++ code. This C/C++ editor is invoked automatically when you edit a C/C++ source file.

The C/C++ editor includes the following features:

- Syntax highlighting
- Content/code assist
- Integrated debugging features

You can customize some of the operation of the Editor view from the **Window > Preferences > C/C++ > Editor** preferences dialog.

Related concepts

CDT Overview

Related tasks

Using Content Assist

Related reference

C/C++ editor key binding actions
C/C++ editor preferences
Outline view for C/C++
Views and editors

Makefile

A makefile is a text file that is referenced by the make command that describes the building of targets, and contains information such as source-level dependencies and build-order dependencies.

The CDT can generate a makefile for you, such projects are called Managed Make projects. Some projects, known as Standard Make projects, allow you to define your own makefile.

Sample Makefile

```
# A sample Makefile
# This Makefile demonstrates and explains
# Make Macros, Macro Expansions,
# Rules, Targets, Dependencies, Commands, Goals
# Artificial Targets, Pattern Rule, Dependency Rule.

# Comments start with a # and go to the end of the line.

# Here is a simple Make Macro.
LINK_TARGET = test_me.exe

# Here is a Make Macro that uses the backslash to extend to multiple
lines.
# This allows quick modification of more object files.
OBJS =  \
 Test1.o \
 Test2.o \
 Main.o

# Here is a Make Macro defined by two Macro Expansions.
# A Macro Expansion may be treated as a textual replacement of the
Make Macro.
# Macro Expansions are introduced with $ and enclosed in
(parentheses).
REBUILDABLES = $(OBJS) $(LINK_TARGET)

# Make Macros do not need to be defined before their Macro Expansions,
# but they normally should be defined before they appear in any Rules.
# Consequently Make Macros often appear first in a Makefile.
```

```
# Here is a simple Rule (used for "cleaning" your build environment).
# It has a Target named "clean" (left of the colon ":" on the first
line),
# no Dependencies (right of the colon),
# and two Commands (indented by tabs on the lines that follow).
# The space before the colon is not required but added here for
clarity.
clean :
 rm -f $(REBUILDABLES)
 echo Clean done

# There are two standard Targets your Makefile should probably have:
# "all" and "clean", because they are often command-line Goals.
# Also, these are both typically Artificial Targets, because they
don't typically
# correspond to real files named "all" or "clean".

# The rule for "all" is used to incrementally build your system.
# It does this by expressing a dependency on the results of that
system,
# which in turn have their own rules and dependencies.
all : $(LINK_TARGET)
 echo All done

# There is no required order to the list of rules as they appear in
the Makefile.
# Make will build its own dependency tree and only execute each rule
only once
# its dependencies' rules have been executed successfully.

# Here is a Rule that uses some built-in Make Macros in its command:
# $@ expands to the rule's target, in this case "test_me.exe".
# $^ expands to the rule's dependencies, in this case the three files
# main.o, test1.o, and  test2.o.
$(LINK_TARGET) : $(OBJS)
 g++ -g -o $@ $^

# Here is a Pattern Rule, often used for compile-line.
# It says how to create a file with a .o suffix, given a file with a .
cpp suffix.
# The rule's command uses some built-in Make Macros:
# $@ for the pattern-matched target
# $lt; for the pattern-matched dependency
```

```
%.o : %.cpp
 g++ -g -o $@ -c $<

# These are Dependency Rules, which are rules without any command.
# Dependency Rules indicate that if any file to the right of the
colon changes,
# the target to the left of the colon should be considered out-of-
date.
# The commands for making an out-of-date target up-to-date may be
found elsewhere
# (in this case, by the Pattern Rule above).
# Dependency Rules are often used to capture header file dependencies.
Main.o : Main.h Test1.h Test2.h
Test1.o : Test1.h Test2.h
Test2.o : Test2.h

# Alternatively to manually capturing dependencies, several automated
# dependency generators exist.  Here is one possibility (commented
out)...
# %.dep : %.cpp
#        g++ -M $(FLAGS) $< > $@
# include $(OBJS:.o=.dep)
```

Frequently Asked Questions:

Your Console view can be very useful for debugging a build.

Q1. My Console view says `"Error launching builder"`. What does that mean?

```
Error launching builder (make -k clean all )
(Exec error:Launching failed)
```

Most probably, the build command (by default "make") is not on your path. You can put it on your path and restart Eclipse.
You can also change the build command to something that is on your path. If you are using MinGW tools to compile, you should replace the build command with "mingw32-make".

Q2. My Console view says `"No rule to make target 'X'"`.

```
make -k clean all
make: *** No rule to make target 'clean'.
make: *** No rule to make target 'all'.
```

By default, the make program looks for a file most commonly called "Makefile" or "makefile". If it cannot find such a file in the working directory, or if that file is empty or the file does not contain rules for the command line goals ("clean" and "all" in this case), it will normally fail with an error message similar to those shown.

If you already have a valid Makefile, you may need to change the working directory of your build. The default working directory for the build command is the project's root directory. You can change this by specifying an alternate Build Directory in the Make Project properties. Or, if your Makefile is named something else (eg. `buildFile.mk`), you can specify the name by setting the default Build command to `make -f buildFile.mk`.

If you do not have a valid Makefile, create a new file named Makefile in the root directory. You can then add the contents of the sample Makefile (above), and modify it as appropriate.

Q3. My Console view says `"missing separator"`.

```
make -k clean all
makefile:12: *** missing separator.  Stop.
```

The standard syntax of Makefiles dictates that every line in a build rule must be preceded by a Tab character. This Tab character is often accidentally replaced with spaces, and because both result in white-space indentation, this problem is easily overlooked. In the sample provided, the error message can be pinpointed to line 12 of the file "makefile"; to fix the problem, insert a tab at the beginning of that line.

Q4. My Console view says `"Target 'all' not remade because of errors"`.

```
make -k clean all
make: *** [clean] Error 255
rm -f Test1.o Test2.o Main.o test_me.exe
g++ -g -o Test1.o -c Test1.cpp
make: *** [Test1.o] Error 255
make: *** [Test2.o] Error 255
make: *** [Main.o] Error 255
g++ -g -o Test2.o -c Test2.cpp
g++ -g -o Main.o -c Main.cpp
make: Target 'all' not remade because of errors.
```

The likely culprit here is that g++ is not on your Path.

The Error 255 is produced by make as a result of its command shell not being able to find a command for a particular rule.
Messages from the standard error stream (the lines saying Error 255) and standard output stream (all the other lines) are merged in the Console view here.

Q5. What's with the -k flag?

The -k flag tells make to continue making other independent rules even when one rule fails. This is helpful for build large projects.

You can remove the -k flag by turning on Project Properties > C/C++ Make Project > Make Builder > Stop on first build error

Q6. My Console view looks like:

```
mingw32-make clean all
process_begin: CreateProcess((null), rm -f Test1.o Test2.o Main.o
test_me.exe, ...) failed.
make (e=2): The system cannot find the file specified.

mingw32-make: *** [clean] Error 2
rm -f Test1.o Test2.o Main.o test_me.exe
```

This means that mingw32-make was unable to find the utility "rm". Unfortunately, MinGW does not come with "rm". To correct this, replace the clean rule in your Makefile with:

```
clean :
        -del $(REBUILDABLES)
        echo Clean done
```

The leading minus sign tells make to consider the clean rule to be successful even if the del command returns failure. This may be acceptable since the del command will fail if the specified files to be deleted do not exist yet (or anymore).

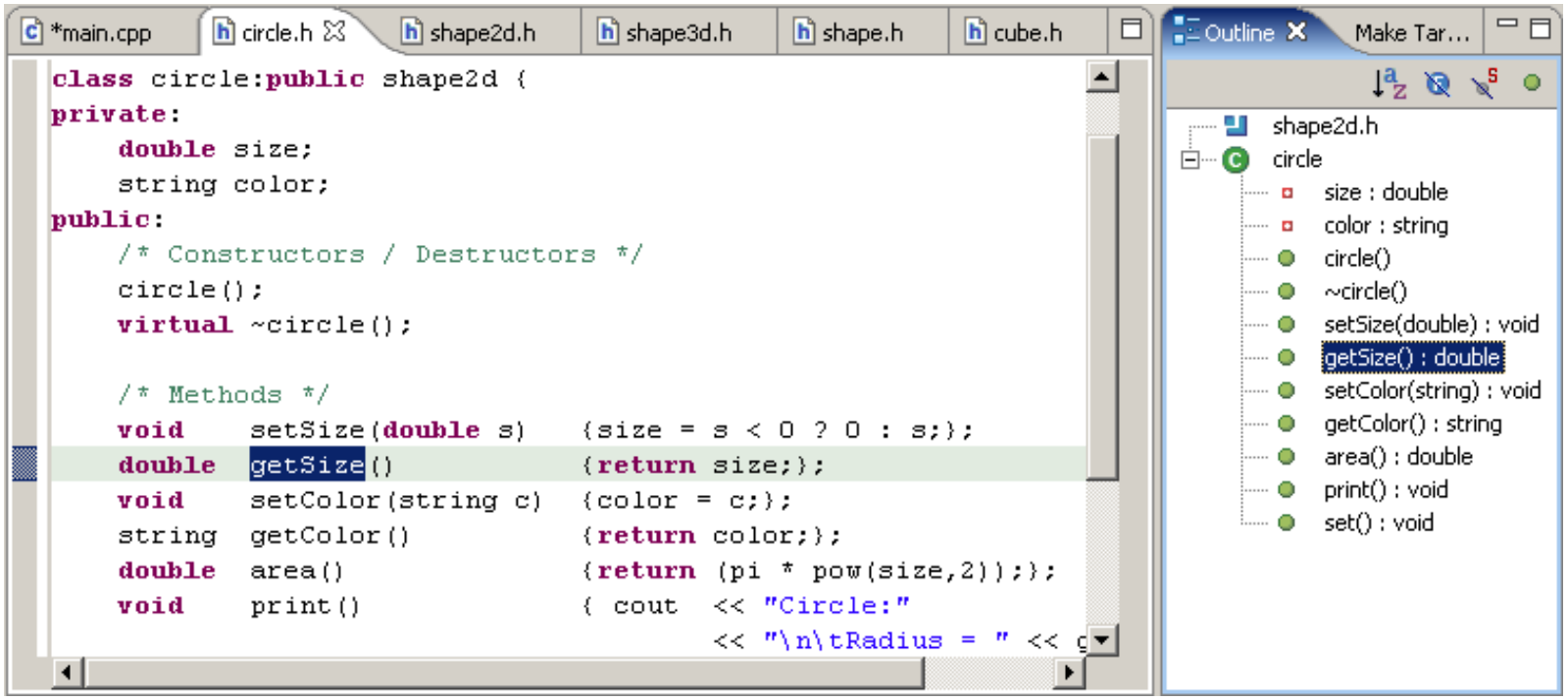
Navigation Aids

This section provides information on navigating through the C/C++ Perspective.

- Outline View
- Project File views
- Make Targets View
- C/C++ search
- Open declarations
- Open Type
- Class Browser

Outline view

The Outline view displays an outline of a structured C/C++ file that is currently open in the editor area, by listing the structural elements.



The Outline view shows the following elements in the source file in the order in which they occur:

- Class
- Namespace
- Include
- Enum
- Enumerator
- Field private
- Field protected
- Field public
- Include
- Method private
- Method protected
- Method public
- Struct
- Typedef
- Union
- Variable
- Function
- Macro Definition

You can also sort the list alphabetically. When you select an element in the Outline view, the C/C++ editor highlights both the selected item and the marker bar (left margin). For example, to move to the start of main() in the C/C++ editor, click `main()` in the Outline view.

For more information about the marker bar, see **Workbench User Guide > Reference > User interface information > Views and editors > Editor area**.

Filtering the Outline View

You can filter the Outline view by choosing to display or hide the following items:

- Fields
- Static members
- Non-public members

You can select an element in the Outline view, and perform the following actions:

- Open the C/C++ Search window box. The Search string box is populated and the element type is selected.
- Complete a text-based search, of a workspace or a specified working set for the selected element.
- Open a selected .h file in the editor.
- Rename Refactor

Icons

	Hide Fields
	Hide Static Members
	Hide Non-Public Members
	Sort

For more information about the Eclipse workbench, see **Workbench User Guide > Tasks > Upgrading Eclipse**.

For more information about Working sets, see **Workbench User Guide > Concepts > Working sets**.

Related concepts

Comments
Content Assist and code completion
C/C++ search
Open Declarations

Related tasks

Displaying C/C++ file components in the C/C++ Projects view
Searching for C/C++ elements

Related reference

Outline view

Project file views

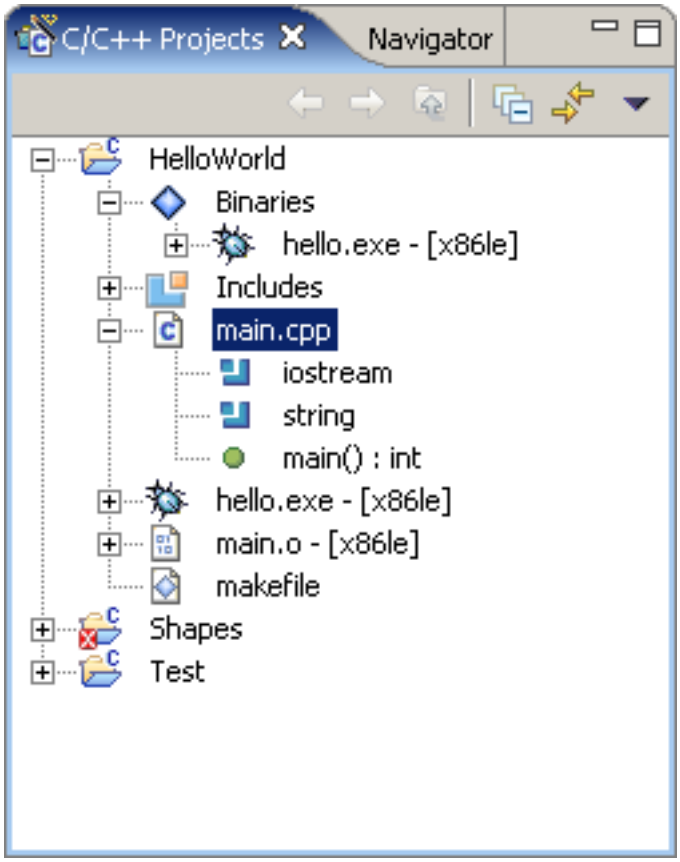
Projects files and elements are displayed in the C/C++ Projects view and in the Navigator view.

C/C++ Projects view

Displays, in a tree structure, only elements relevant to C and C++ projects. In this view you can do the following:

- Browse the elements of C/C++ source files
- Build Targets
- Create new projects, classes, files, or folders
- Import or Export files and projects
- Manage existing files (cut, paste, delete, move or rename)
- Open files in the editor view
- Open projects in a new window
- Refactor
- Restore deleted files from local history
- Search

Files that you select in the C/C++ Projects view affect the information that is displayed in other views.

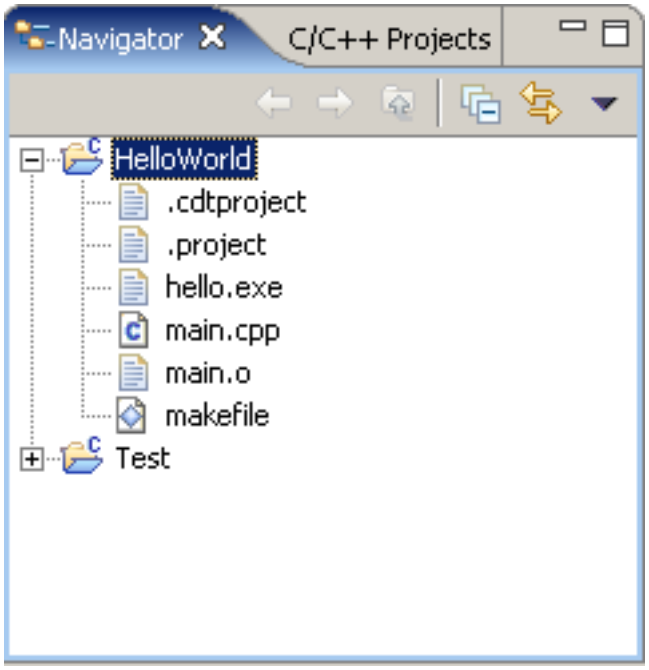


Navigator view

The Navigator view provides a hierarchical view of all the resources in the workbench, not just your C/C++ resources. From this view, you can open files for editing or select resources for operations such as exporting.

Right-click any resource in the Navigator view to open a pop-up menu from which you can perform operations such as copy, move, create new resources, compare resources, or perform team operations. For a description of what each menu item does, select an item and press F1.

By default, the Navigator view is included in the Resources perspective. To add it to the current perspective, click **Window > Show View > Navigator**.



Toolbar icons

Icon	Name	Description
	Minimize Console.	Minimizes the view.
	Maximize Console	Maximizes the view.
	Back	This command displays the hierarchy that was displayed immediately prior to the current display. For example, if you Go Into a resource, then the Back command in the resulting display returns the view to the same hierarchy from which you activated the *Go Into* command. The hover help for this button tells you where it will take you. This command is similar to the Back button in a web browser.
	Forward	This command displays the hierarchy that was displayed immediately after the current display. For example, if you've just selected the Back command, then selecting the Forward command in the resulting display returns the view to the same hierarchy from which you activated the Back command. The hover help for this button tells you where it will take you. This command is similar to the Forward button in a web browser.
	Up	This command displays the hierarchy of the parent of the current highest level resource. The hover help for this button tells you where it will take you.
	Collapse All	This command collapses the tree expansion state of all resources in the view.

	**Link with Editor**	This command toggles whether the Navigator view selection is linked to the active editor. When this option is selected, changing the active editor will automatically update the Navigator selection to the resource being edited.
	**Menu**	Click the black upside-down triangle icon to open a menu of items specific to the Navigator view. Select Working Set Opens the **Select Working Set** dialog to allow selecting a working set for the Navigator view. Deselect Working Set Deselects the current working set. Edit Active Working Set Opens the **Edit Working Set** dialog to allow changing the current working set. Sort This command sorts the resources in the Navigator view according to the selected schema: ○ **By Name**: Resources are sorted alphabetically, according to the full name of the resource (e.g., A.TXT, then B.DOC, then C.HTML, etc.) ○ **By Type**: Resources are sorted alphabetically by file type/ extension (e.g., all DOC files, then all HTML files, then all TXT files, etc.). Filters This command allows you to select filters to apply to the view so that you can show or hide various resources as needed. File types selected in the list will not be shown in the Navigator. Link with Editor See the toolbar item description above.

For information about the Navigator view toolbar and icons, see **Workbench User Guide > Concepts > Views > Navigator View**.

For information about the pop up menu in the Navigator view, see **Workbench User Guide >**

Reference > User interface information > Views and Editors > Navigator View.

For information about the Working Sets, see **Workbench User Guide > Concepts > Workbench > Working sets**.

Related concepts

CDT Projects

Working with existing code

Related tasks

Creating a project

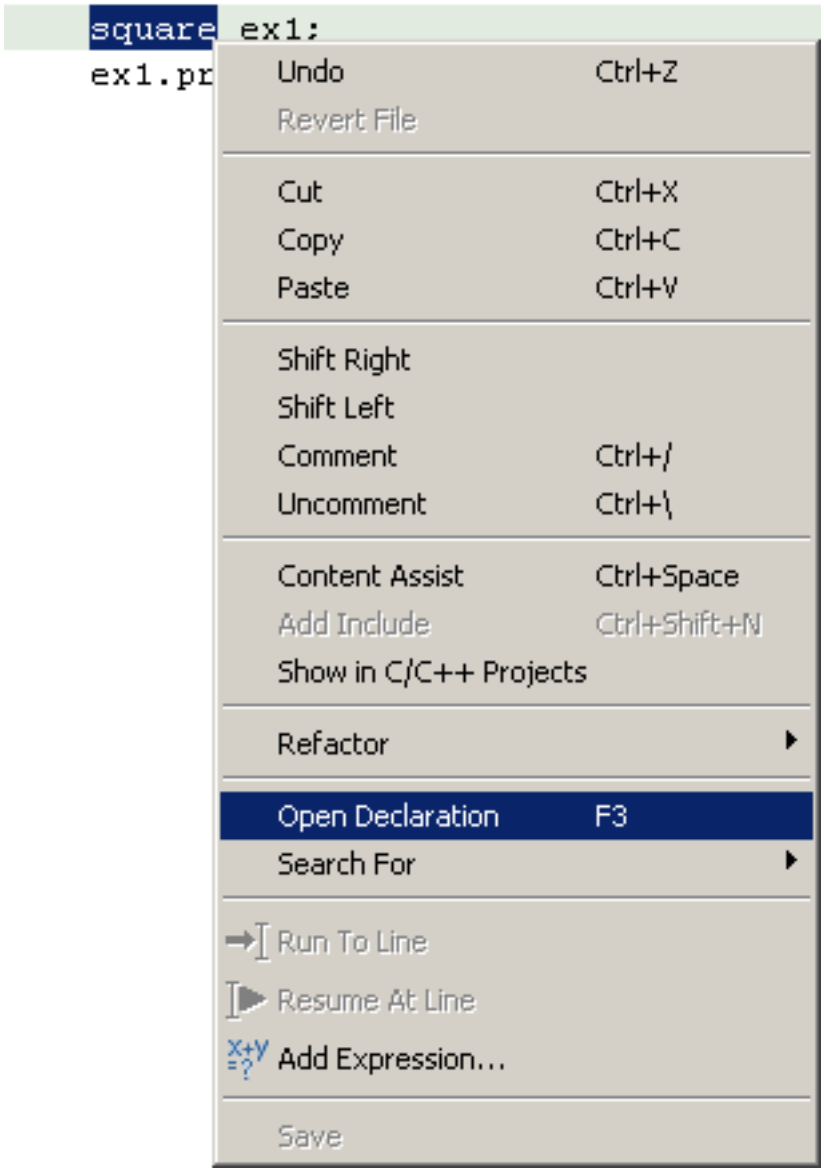
Working with C/C++ project files

Related reference

C/C++ perspective icons

Open declaration

You can select an element name in your code and quickly navigate to its declaration.



Open declaration will attempt to navigate to the exact declaration of the selected element. Open declaration requires your file to have the proper include paths set up to the declaration. If for any reason open declaration cannot find the declaration, it will display the following message in the status line:

For more information see Adding Include paths and symbols.

Related concepts

CDT Projects

C/C++ search

Related tasks

Adding Include paths and symbols

Navigate to a C or C++ element's declaration

Searching for C/C++ elements

Build

This section describes the build views and terminology.

Building C/C++ Projects
Manage Build Extensibility Document

Building C/C++ projects

The CDT relies on an external make utility, such as GNU make, to build a project. The CDT can generate makefiles automatically when you create a Managed Make C project or a Managed Make C++ project. You have the option of creating a Standard Make C project or a Standard Make C++ project and providing the makefile yourself.

Required utilities

You must install and configure the following utilities:

- Build (e.g. make).
- Compile (e.g. gcc).
- Debug (e.g. gdb).

Note: while make, gcc and gdb are the examples used in the documentation, virtually any similar set of tools or utilities could be used.

Tip: Cygwin contains these utilities (make, gcc and gdb) for a Windows environment, while running the cygwin installation ensure `gcc` and `make` are selected, they are not installed by default. For more information, see http://www.cygwin.com. Red Hat users, all you need to build your project is included in the Red Hat Linux installation. For other operating systems please refer to your installation documentation.

Build terminology

The CDT uses a number of terms to describe the scope of the build.

Build Project

This is an incremental build (make all, assuming all is defined in your makefile). Only the components affected by modified files in that particular project are built.

Rebuild Project

Builds every file in the project whether or not a file has been modified since the last build. A rebuild is a clean followed by a build.

For more information on builds, see:

- **Workbench User Guide > Concepts > Workbench > Builds**
- **Workbench User Guide > Tasks > Building resources**

Build-related information is displayed as follows:

- The Console view displays the output of the build tools.
- The Tasks view displays a list of compiler errors and warnings related to your projects.
- Makefile targets are displayed in the Make Targets view.

For more information about the Tasks view, see **Workbench User Guide > Reference > User interface information > Views and editors > Tasks view**.

Getting a makefile

You can either create a C/C++ project for which you supply the **makefile** or create a C/C++ project for which the CDT generates makefiles automatically.

To create a new project, from the menu bar choose **File > New > Project**. In the dialog that appears:

- To create a project for which you supply the **makefile**, select either **Standard Make C project** or **Standard Make C++ project**.
- To create a project for which the CDT supplies a basic **makefile**, select either **Managed Make C project** or **Managed Make C++ project**.

Setting build preferences

You can set build preferences in Eclipse:

Build order
: If certain projects must be built before others, you can set the *build order*. If your project refers to another project, the CDT must build the other project first. To set the build order, from the menu bar select **Window > Preferences > Build Order**.

  When you set the build order, the CDT does not rebuild projects that depend on a project; you must rebuild all projects to ensure all changes are propagated.

Automatic save
: You can set the CDT to perform an *automatic save* of all modified resources when you perform a manual build; from the menu bar, select **Windows > Preferences > Workbench**.By default, this feature is enabled.

Controlling the building of your project

The C/C++ compiler that a project uses is controlled by the project's **Properties** setting. To view a project's properties, right-click on the project and select **Properties**. In the dialog that appears, the **C/C++ Standard Make Project** page enables you to control a variety of settings, including:

Build Setting
: Controls whether the compiler will **Stop On Error**or **Keep Going On Error**. Choosing **Keep Going On Error** will force the compiler to attempt to build all referenced projects even if the current project has errors.

Build Command
: Controls which make is used.

Workbench Build Behavior
: Controls which makefile target will be built depending on the scope of the build.

Viewing build information

Build-related information is displayed as follows:

- The **Console** view displays the output of the make utility.
- The **Tasks** view displays a list of compiler errors and warnings related to your projects.
- Build actions display in the **Make Targets** view.

Related concepts

CDT Projects

Project file views

Related tasks

Building

Managed Build System Extensibility Document

This document describes the design of the managed build system and how to extend it.

Author : Sean Evoy

Revision Date : 10/21/2003 - Version: 0.1.0

Change History : 0.1.0 - Document Creation

Table of Contents

1 Introduction

C and C++ developers are a diverse group. The tools they use, the processes they follow, and the level of support they expect from their development environments vary widely. The CDT provides a framework for integrating those tools into Eclipse and the managed build system is part of that framework. Understanding how the managed build system works, and what steps are required to extend it is the focus of this document.

1.1 Who Needs This Information

The information in this document describes the design of the managed build system and discusses how to add new build targets and tools to it through the `ManagedBuildInfo` extension point. It is intended for someone who wants to understand how the managed build system works, or is interested in adding their own tool chain specification to it.

The CDT comes configured to generate makefiles to build executables, static libraries, and shared libraries on Cygwin, Linux, and Solaris using Gnu tools. If you are using the CDT on those platforms, have access to Gnu tools and find the predefined targets sufficient for your needs, then you do not need to modify anything. Please feel free to skip sections 2, 3, and 5 as they are primarily concerned with adding new tool chains to the build model.

If you are working with tools other than Gnu, or you wish to build for targets the CDT does not support out of the box, then you have to decide whether you will provide your own makefile and use the standard builder, or add a description of your target to the extension point and let the CDT generate the makefiles for your project.

If you choose to add your own tools to the managed build system, it is assumed that you are familiar with XML and the Eclipse extension point mechanism. Having made the standard disclaimer, it should be said that the tutorial in section 5 presents a cookbook approach to adding a new tool specification, so you can always jump right in and refer to the online help in the *Platform Plug-in Developer Guide* if you get stuck.

1.2 Managed Build System Overview

The managed build system consists of several components that interact to build a project. At the core of the managed build system is the build model. It is the central clearing house for all the build-related information that internal and external clients require. There are three internal clients; the user interface components, a makefile generator that is responsible for generating a correct makefile for a project when it is built, and the CDT parser. The external clients are the end-user, who interacts with the build model through the user interface, and tool-chain integrators who supply tool definitions to the build model. The diagram below shows the basic set of relationships between these components.

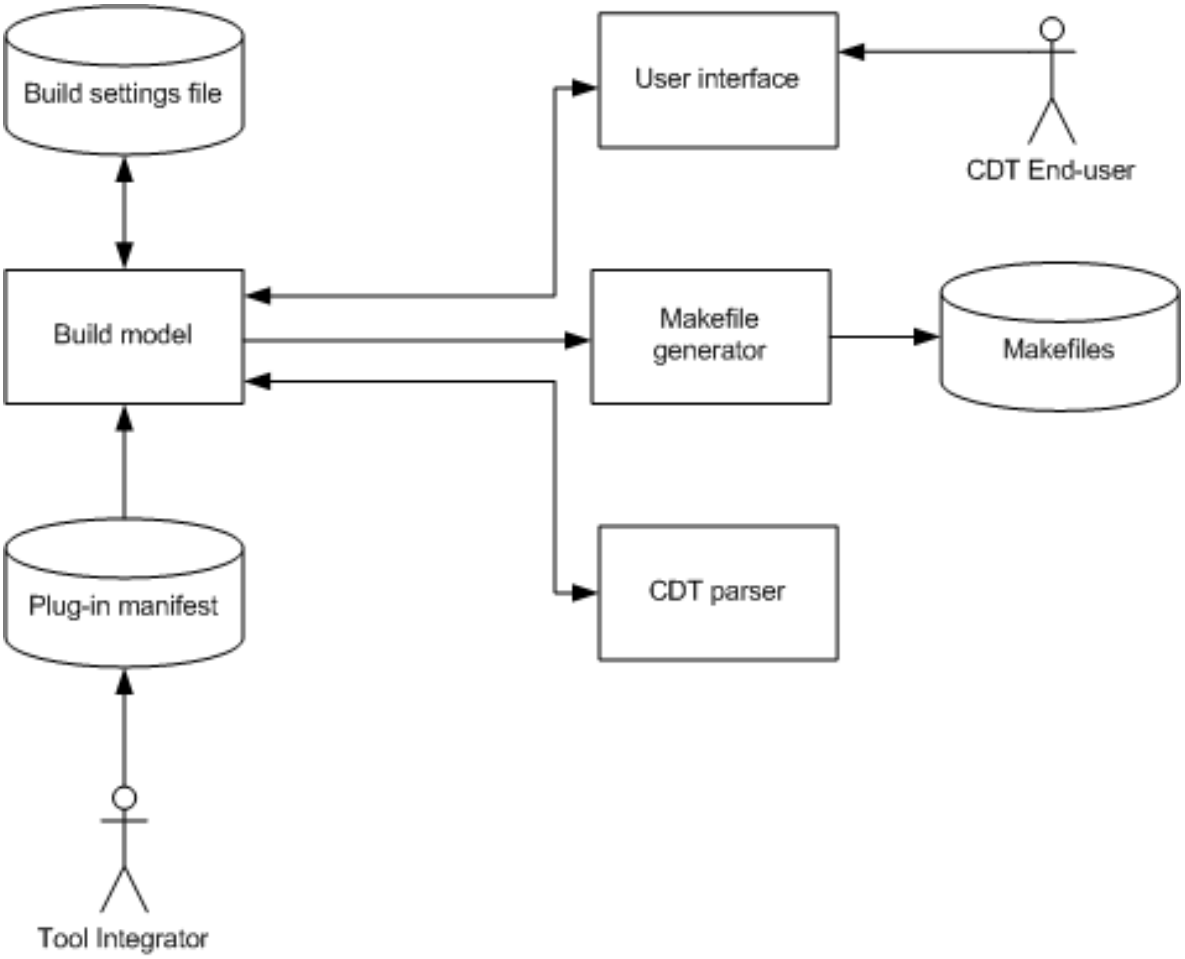


Figure 1 Managed build system Overview

1.2.1 External Users

From the perspective of the build model, there are two external users. The first is the end-user that interacts with the build model through the UI elements described in section 4. The UI includes a new project wizard that asks the build model about what tools have been defined for new projects. When the project has been created, the project property page uses the information in the build model to populate its display. The user can change the information associated with the tools for a project through the property page and the build model is responsible for saving those changes between sessions. The second external user is the tool integrator who adds information about their tool-chain to a plug-in manifest as described in the tutorial in Section 5. The tool-chain integrator is the primary audience for this document.

1.2.2 Internal Users

There are three internal clients of the information in the managed build system. The first is the makefile generator that creates a correct makefile for a project based on the tools and settings defined for the project in the build system. The second is the built-in CDT parser that relies on the build system to tell it about the include paths and defined preprocessor symbols for a given project so that it can properly parse a file. The third client is the UI component of the build system that queries the build model for the tools and options defined for a project to build its display and store the user settings.

1.2.3 Tool Definitions and Settings Storage

A key feature of the managed build system is that it is extensible. Tool integrators can use the grammar, described in Section 2, to add their own tools to the build system. The same grammar is used to save the settings that the user overrides through the UI between sessions.

1.3 The Standard Build System

There is also a standard build system supplied as part of the CDT framework that is unrelated to the managed build system. The standard system provides a small set of tools to build a user's projects. The user is expected to supply a makefile which includes enough information to build their project. The UI allows the user to switch between targets defined in the makefile, like clean or all, and for the user to enter the information the parser requires.

The decision to use the standard or managed build system is a trade-off. For users with an existing project that already has a set of working makefiles, or for users that prefer to write their own makefile, the standard system may be perfect. However, many users are uncomfortable writing makefiles, so the standard system may present a barrier to adoption for them.

2 Build Model Grammar Elements

The managed build system defines a grammar to describe tool chain information. This information is used to store invariant data, like the command line invocation for a specific compiler, for example. The build system also stores user settings between sessions, like the level of debugging information that is needed for a particular build configuration. The following section describes the format of the grammar and what the information is used for by the build model.

2.1 Model

The figure below shows a UML model of schema elements.

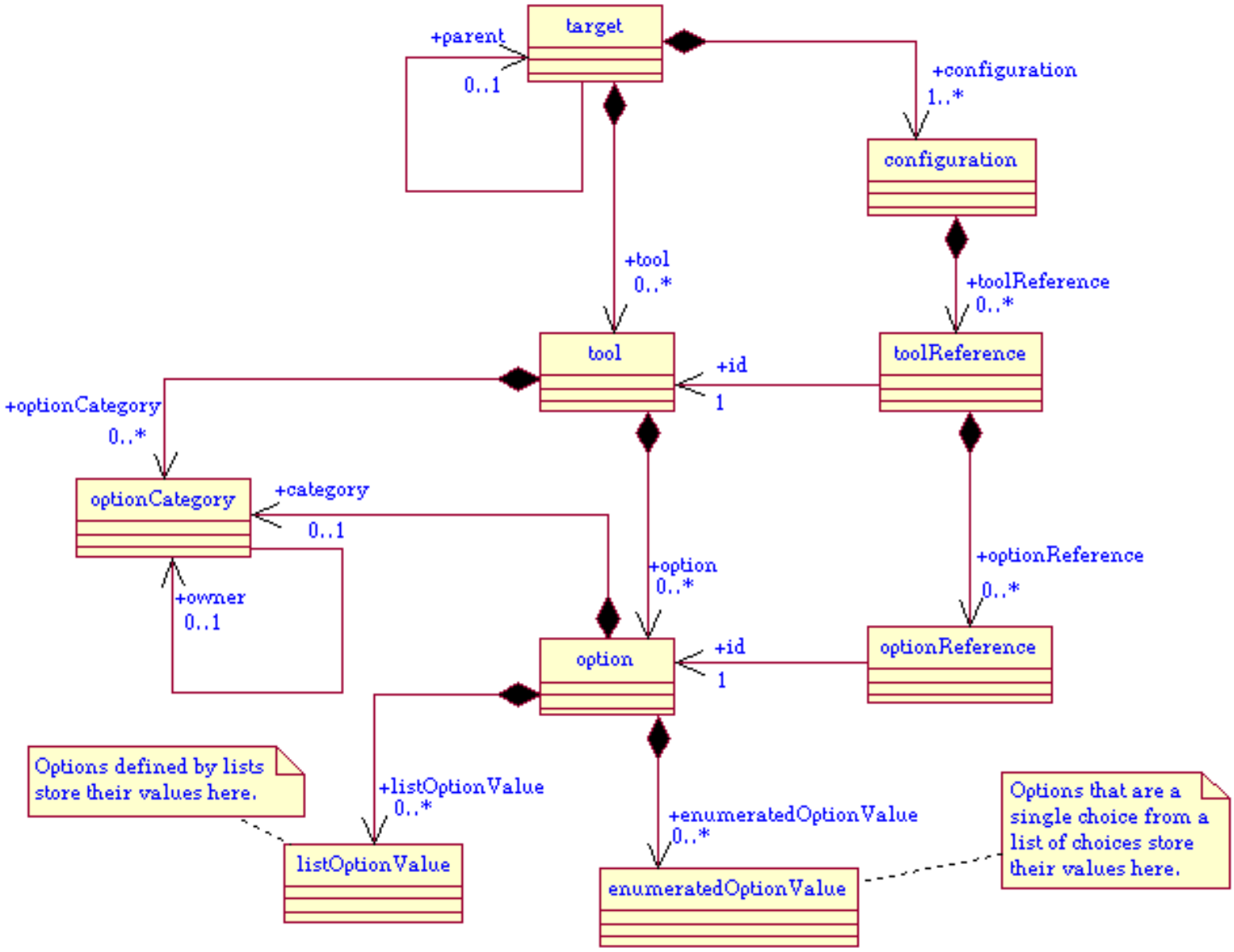


Figure 2 Managed build model elements

2.2 Target

In its current implementation, the *target* in the build model is confusing because it encompasses the responsibilities of two distinct participants in a build. The first is the host target where the build tools are located. The information that this type of target needs to managed are things like the command to start the make utility, to remove files, and to invoke build tools. The second is the physical target that the build artifact is supposed to run on, which may or may not be the same as the host target. This distinction will be clarified in the next iteration of the build model.

Targets can be arranged into hierarchies to promote the efficient sharing of tools and configurations. If you have defined a target that should not be selected by the user, but is a root for other targets, it may be declared abstract by setting the *isAbstract* attribute to `true`. Abstract targets do not appear in the UI. You must provide a unique identifier for the target in the *id* attribute. The build model uses this information to distinguish between the target definitions it finds. Children of the abstract target will have the same tools and configurations the abstract target

has. For these children to function properly, their parent attribute must contain the unique identifier of the parent target.

A concrete target must have at least one configuration defined for it. A target must also define (or inherit) a set of tool definitions that work together to produce the build goal as an output. You must also provide a meaningful *name* that will be displayed to the user in the UI and new project wizards.

The target defines the command needed to invoke the make utility in the *makeCommand* attribute. Any special flags that need to be passed to the make utility are defined in the *makeFlags* attribute. The command to remove files on the host machine is defined in the *cleanCommand* attribute.

Typically a build target will only be valid on a limited subset of operating systems. For example, it does not make much sense to allow a user to create a Solaris shared library project if they are running Eclipse and the CDT on Windows. You can specify the operating systems that the target is restricted to as a comma-separated list in the *osList* attribute. At the moment, you can specify `win32`, `linux` and `solaris` as the filters.

The CDT offers a facility for parsing binary files if it knows which output format the build artifact has been produced with. The *binaryParser* attribute must contain the id of the appropriate parser if you want build artifacts of the target to be parsed in the workspace. There are currently two defined binary parsers; `org.eclipse.cdt.core.PE` for Windows artifacts, and `org.eclipse.cdt.core.ELF` for Linux and Solaris. This information is used to set the parser when a project is created and is not something the user can change through the UI.

The target is responsible for maintaining the name of the final build goal. The user selects the name of the build target in the UI, and the build model maintains it in the *artifactName* attribute. The implementer of a tool chain should not specify this in the plug-in manifest. However, the default extension for the target can be specified using the *defaultExtension* attribute.

2.2.1 Schema

Attribute	Description	Required
artifactName	The name of the build goal defined by the target. This is set by the user through the UI and stored in the build model through this attribute.	no
binaryParser	The id of the appropriate parser for the build artifact.	yes
cleanCommand	The command to remove files on the build machine. You must define this value if the target does not have a parent, or it is not defined in the parent.	yes
defaultExtension	The extension the build goal will have, for example ‘.exe’ or ‘.so’.	no
id	A unique identifier that the model manager will use to keep track of this specific target.	yes

isAbstract	Flags the target as abstract. An abstract target can not be selected by the user in the UI, but children of the target will inherit its tools and configurations.	yes
isTest	A target can be flagged for test purposes only. It can be manipulated programmatically, in JUnit tests for example, but not selected by the user in the UI.	yes
makeCommand	The command to invoke the make utility. You must define this value if the target does not have a parent, or it is not defined in the parent.	yes
makeFlags	The default flags passed to the make utility on the command line.	yes
name	The name for the target that is displayed to the user in the UI.	yes
osList	A comma-separated list of operating systems that the target is valid for.	no
parent	The unique ID of the parent of the target.	no

2.2.2 Example

The example below shows a target definition called ‘Executable’. Tool and configuration information will be added to our definition is later sections.

```
<extension …>
  <target
    makeFlags="-k"
    isTest="false"
    cleanCommand="rm -rf"
    name="Executable"
    defaultExtension=".exe"
    isAbstract="false"
    makeCommand="make"
    binaryParser="org.eclipse.cdt.core.PE"
    osList="win32"
    id="example.target.executable">
    <configuration …>
    <tool …>
  </target>
</extension>
```

2.3 Tool

A tool represents some sort of executable component that can take a set of inputs and produce a set of outputs. A tool must have a unique *id* for the build model, and a *name* that is displayed to a user through the UI.

Certain tools logically belong to certain kinds of projects. For example, the Gnu compiler is invoked differently for C and C++ source files. You can specify a filter for a tool based on the nature of a project using the *natureFilter* attribute. When a new C project is created, a `cnature` is added to it. New C++ projects have both a `cnature` and `ccnature`. The build model interprets the filter as follows. If you specify a `cnature` filter, then the tool will only be displayed if the project has a `cnature` and does *not* have a `ccnature`. If you specify a `ccnature` filter, then the tool will be displayed if the project has a `ccnature`. The default if no filter is specified is to display the tool for all projects.

Tools can be added to the plug-in manifest as part of a target or as a stand-alone specification. Tools defined as part of a target will be available for projects that are created to build for the target or any child of the target in which the tool is defined. If you want targets o use a tool that is not specified as belonging to it, you must create a reference to the tool in the target specification. Please refer to section 2.6 for a description of how to use tool references in your plug-in manifest.

Tools can define a set of input file extensions in the *sources* attribute. This indicates that a tool will build for those and only those file types. Similarly, a tool might specify a set of file extensions that they will produce in the *outputs* attributes.

Each tool specifies a *command* that will be placed in the makefile during the makefile generation stage of building. Two optional flags control how the command is generated. If the tool requires a special output flag, such as `-o` for a compiler or linker, the implementer must specify that in the *outputFlag* attribute. If the output of the tool usually has a special prefix, like the prefix `lib` for libraries on POSIX systems, the implementer must specify this in the *outputPrefix* attribute.

One of the clients of the information in the build model is the makefile generator. It must track the dependencies between elements in the workspace, and to do that, it needs to know if a file is a header or a source file. Currently, the build model uses the list of file extensions specified in the *headerExtensions* attribute to identify a file as containing an interface.

2.3.1 Schema

Attribute	Description	Required
id	A unique identifier for the tool that will be used by the build model.	yes
name	Human-readable name for the tool to be used in the UI.	yes
sources	A comma-separated list of file extensions that the tool will produce output for.	no
outputs	The extension that the tool will produce from a given input.	no
command	The command that invokes the tool. For example, gcc for the Gnu C compiler, or g++ for the Gnu C++ compiler.	yes

outputFlag	An optional flag for tools that allow users to specify a name for the artifact of the tool. For example, the GCC compiler and linker tools typically allow the user to specify the name of the output with the '-o' flag, whereas the archiver that creates libraries does not.	no
outputPrefix	Some tools produce files with a special prefix that must be specified. For example, a librarian on POSIX systems expects the output to be lib.a so 'lib' would be the prefix.	no
dependencyCalculator	Unused in 1.2	no
headerExtensions	A comma-separated list of file extensions that are used for header files by the tool chain.	yes
natureFilter	Specify the project natures the tool should apply to.	yes

2.3.2 Example

The tool shown in the example below will appear in the UI with the label *Compiler*. It will be used to build any file in the project with a `.C`, `.cc`, or `.cpp` extension and will produce a file with an `.o` extension. When the makefile is generated, a rule will be generated with the command `g++ <…> -o <…>`.

```
<target …>
  <tool
    sources="cc,cpp,C"
    name="Compiler"
    outputFlag="-o"
    outputs="o"
    command="g++"
    headerPrefix="h,hpp,H"
    natureFilter="ccnature"
    id="executable.compiler">
       <optionCategory …> </optionCategory>
       <option id="category.flags.comp_flags" …> </option>
  </tool>
</target>
```

2.4 Option Category

A tool can have a large number of options. To help organize the user interface for these options, a hierarchical set of option categories can be defined. A unique identifier must be specified in the *id* attribute. This will be used by the build model to manage the category. The user will see the value assigned to the *name* attribute. If the category is nested inside another category, the unique identifier of the higher level category must be specified in the *owner* attribute, otherwise use the identifier of the tool the category belongs to.

2.4.1 Schema

Attribute	Description	Required
id	Used by the build model to uniquely identify the option category.	yes
name	A human-readable category name, such as 'Pre-processor Options'. This will be the name the user sees displayed in the UI.	yes
owner	Option categories can be nested inside other option categories. This is the ID of the owner of the category.	yes

2.4.2 Example

This example shows an option category that will be displayed in the UI with the label *Flags*. There are two options defined in this category, *General*, and *Optimization*.

```
<tool …>
  <optionCategory
    owner="executable.compiler"
    name="Flags"
    id="compiler.category.flags">
  </optionCategory>
  <option
    name="General"
    category="compiler.category.flags" …>
  </option>
  <option
    name="Optimization"
    category="compiler.category.flags" …>
  </option>
</tool>
```

2.5 Configuration

A target defines the information about the tools needed to build a project for a particular environment. Configurations are used to pre-define and store the settings that the user specifies for those tools.

A target must have at least one default configuration defined for it. Users can create new configurations for a project, but they must be based on the settings defined in a default configuration. For example, a user may want to create a *Profile* configuration based on the target's default *Debug* configuration.

Each configuration must have a unique identifier specified in the *id* attribute that will be used by the build model to manage the configuration. It must also have a *name* that will be displayed in the UI in the build property page and new project wizards.

2.5.1 Schema

Attribute	Description	Required
id	A unique identifier that the model manager will use to keep track of this specific configuration.	yes
name	The human-readable name that will be displayed in the UI to identify this configuration.	yes

2.5.2 Example

The example below shows a configuration named *Default* that belongs to the target *Executable*.

```
<target name="Executable" …>
  <configuration
    name="Default"
    id="example.config.default">
  </configuration>
</target>
```

2.6 Tool Reference

A tool reference is primarily intended to be used when saving user settings between sessions. When the user has overridden an option in the referenced tool, an option reference with the new setting is created and added to the tool reference.

Tool references are used by the build model for two distinct tasks; because they contain option references, tool references hold onto user settings between sessions. They can also be added to a default configuration specification if the default settings for the tool should be overridden. For example, a ‘Debug’ configuration may have optimization disabled by default, whereas a ‘Release’ configuration may default to the highest possible level.

2.6.1 Schema

Attribute	Description	Required
id	The unique identifier of the tool this is a reference for.	yes

2.6.2 Example

The example below shows how the user has overridden the compiler flags option in the compiler tool in the *Default* configuration.

```
<configuration name="Default" …>
  <toolReference id = "executable.compiler">
    <optionReference id="category.flags.comp_flags" …></optionReference>
  </toolReference>
</configuration>
```

2.7 Option

Options in the build model are used to organize and maintain the command arguments that are sent to tools during the build. Users interact with the build model through the UI to set the value of options. Options hold different kinds of values, so there are some subtle, yet important, rules for how options are to be defined. These rules are summarized in Table 1

Each option must have a unique *id* for the build model to properly manage it. A descriptive *name* that will appear in the UI must be specified. Options can be organized into categories to keep the UI more manageable. If an option category has been defined for the tool, and the option should be displayed as part of that category, then the unique identifier of the option category must be specified in the *category* attribute.

2.7.1 Option Types

Some options contain commands to turn a feature off or on, such as setting a flag to see descriptive messages from a tool. Others contain lists of values, such as a set of directories to search for files. Still others are a single selection from a pre-determined range of choices, like the level of debugging information to produce, or the type of architecture to build for. The *valueType* attribute is used to indicate to the build model what kind of option it is.

Specifying the type of value an option contains is an important design decision, since it controls how the build model treats the contents of the option's attributes, and just as importantly, how the option is displayed to the user. The basic types are `string`, `boolean`, `stringList`, and `enumerated`.

There are also four specialized cases of list options, `includePath`, `definedSymbols`, `libs`, and `userObjs` to manage the list of paths to search for header files, the defined preprocessor symbols, external libraries to link against, and object module to link in respectively.

2.7.1.1 String Options

An option of type 'string' contains a set of values the user has typed in the UI. When the UI is created, it will display the option using a simple entry widget. This option type is useful for options that cannot be easily specified using lists or enumerations, or for options that are not frequently set. For these types of options, the build model will ignore what it finds in the *command* attribute.

2.7.1.2 Boolean Options

An option of type 'boolean' is used to specify an option that is either true or false. The option will be displayed to the user as a check box. The value of the option is set true by selecting the check box, and false by deselecting it. If true, the command associated with the option will be passed to the tool when it is invoked. The default value of the option will be considered when it is displayed in the UI.

2.7.1.3 Enumerated Options

Enumerated options are displayed in the UI in a drop-down list-box. With enumerated options, the option definition takes on an organizational role; the important information is stored in the enumerated option values. Any information specified in *defaultValue* is ignored, since the contents of the enumerated value definitions are used to populate the selection widget. The option answers the command of the selected enumerated value, so any information in *command* is also ignored.

2.7.1.4 String List Options

String list options are displayed in the UI using a list control and a button bar that allows users to add, remove, and reorder list items. Elements of the list are defined and stored in list options values, as described in section 2.9. Like enumerated options, lists ignore the information in the *defaultValue* attribute, but unlike the enumerated option, they treat any pre-defined list option values as defaults. The value defined in the *command* attribute will be applied to all the values in the list.

2.7.1.4.1 Special List Options

There are four special cases of string list options; `includePaths` specify the paths to search for header files, `definedSymbols` for user-defined preprocessor defines, `libs` for libraries that must be linked into the final build goal, and `userObjs` for external object files that must be linked.

While specifying these types of options as type *stringList* will make them appear in the UI correctly, the build model will not be able to recognize them as special in any way. Since certain functions of the CDT require this information to function correctly, it is important to flag these types of options appropriately. For example, the search and indexing function may not perform correctly if the includes paths and defined symbols are not set for a project. Similarly, the makefile generator may not be able to generate dependencies correctly in the makefile if it is unaware that there are libraries and external object files that participate in the final build step.

2.7.2 Default Values

Options can contain default values that apply to the option until the user has edited them through the UI. You can specify those values using the *defaultValue* attribute. However, the type of option will determine how the build model treats the value it finds associated with the attribute. Options that define simple string values will pass the value to the tool exactly as it is defined in the attribute. For Boolean options, any value but the string `true` will be treated as false. List options treat all the defined list option values as default, and enumerated options search through the defined enumerated values for the default.

2.7.3 Option Commands

The values stored in the options are passed to build tools with unique flags, depending on the compiler and the option. For example, an option defining the paths a linker should search for libraries might contain a large number of search paths, but each path is passed to the linker with a `-L` flag. The *command* attribute is used to hold the actual flag to pass along with the option value.

The build model handles the value it finds associated with the command attribute differently depending on the type of value the option is managing based on the following heuristic. For string options, the command is ignored since

the contents of the option are treated as the command. For enumerated options, the command associated with the selected enumerated value is used, not the command defined in the option. For Boolean options, the command is used if the option value is set to true, otherwise it is ignored. For list options, the command is applied to each element of the list.

Option Value Type	Uses Default Value	Uses Command	UI Element
string	Yes	No	Entry widget
boolean	Yes	Yes if true, else no	Check box
enumerated	No.	No.	Drop-down list-box
stringList	No.	Yes.	List and button bar

2.7.4 Schema

Attribute	Description	Required
id	A unique identifier for the tool that will be used by the build model.	yes
name	Human-readable name for the tool to be used in the UI.	yes
valueType	Type of value the option contains.	yes
category	This is the id of the option category for this option. The id can be the id of the tool which is also a category.	no
defaultValue	Optionally specifies the value for the option if the user has not edited it. For options containing a Boolean value, the string ‘true’ is treated as 1, any other value as 0.	no
command	An optional value that specifies the actual command that will be passed to the tool on the command line.	no

2.7.5 Example

The example below shows the specification for the optimization level option for a compiler. Note that it is an enumerated type, so the only attributes defined for the option itself are its id for the build model, a human-readable name, the id of the category it belongs to, and the type of value the option holds.

```
<target name="Executable" …>
  <option
    name="Optimization Level"
    valueType="enumerated"
    category="compiler.cat.optimization"
    id="optimization.level">
    <enumeratedOptionValue
      name="None (-O0)"
      command="-O0"
      id="level.none">
    </enumeratedOptionValue>
    <enumeratedOptionValue
      name="Optimize (-O1)"
      command="-O1"
      id="level.optimize">
    </enumeratedOptionValue>
    <enumeratedOptionValue
      name="Optimize more (-O2)"
      isDefault="true"
      command="-O2"
      id="level.more">
    </enumeratedOptionValue>
    <enumeratedOptionValue
      name="Optimize most (-O3)"
      command="-O3"
      id="level.most">
    </enumeratedOptionValue>
  </option>
</target>
```

2.8 Option Reference

An option reference always belongs to a tool reference, and is used in two ways. First, the build model uses option references to hold onto information the user has changed through the UI and to store it between sessions. The second is to override the default option settings in a configuration.

The reference identifies the option it overrides through the *id* attribute. The *defaultValue* attribute is used to hold onto the user entry, but it is used differently depending on the *valueType* of the option. The attribute contains the strings `true` or `false` for Boolean options. String options contain the data entered by the user. For enumerated options, the attribute contains the selected enumerated list value. For list options, this attribute is not used. Instead, *listOptionValues* are used.

2.8.1 Schema

Attribute	Description	Required
id	The unique identifier of the option that this is a reference to.	yes

defaultValue	For boolean and string options, this field is used to hold the value entered by the user. For enumerated options, it is used to hold the selected enumerated option value. For list options, this attribute is not used.	no
command	unused in 1.2	no

2.8.2 Example

The example below shows how the build model saves overridden option information in the project file. In this case, the tool reference is a linker, and the option references are for linker flags and library paths.

```
<toolReference id="linker">
  <optionReference defaultValue="-shared -fPIC" id="linker.flags"/>
  <optionReference id="linker.paths">
    <listOptionValue value="/home/workspace/ProjectB"/>
  </optionReference>
</toolReference>
```

2.9 List Option Value

Some options are best described using a list of values. This build model element is used to define an individual element of a list option. Typically, these options are populated by the user, not by the person describing the option. However, if you define one or more values in your extension point, they will be displayed in the UI when the user edits the build settings for the project. If the user modifies those settings, the overridden values will be stored by the build model and displayed in the UI.

There is an exception to this, however. Certain core functions in the CDT rely on the built-in parser to function correctly. In order to return accurate values, the CDT parser must mimic (as closely as possible) the preprocessor that ships with the tool chain used by the target. Unfortunately, these tools often have a number of built-in symbols and include paths that the user is never required to set, and may be unaware even exist. In those cases, the implementer of the tool chain must set those values in the tool definition and flag them by setting the value of the *builtIn* attribute to true. Built in list option values are never shown to the user, and are only passed to clients of the build model that specifically request them.

2.9.1 Schema

Attribute	Description	Required
builtIn	An optional Boolean field that tells the build model to treat the value defined as read-only.	no
value	The contents of the list item. The build model will apply the flag defined in the option to each value in the list.	no

2.9.2 Example

The example below shows an option, *Defined Symbols*, which contains a pre-populated list of built-in values; `__I386__`, and `__i386__` respectively.

```
<option name="Defined Symbols" valueType="definedSymbols" …>
  <listOptionValue
       builtIn="true"
       value="__I386__">
  </listOptionValue >
  <listOptionValue
       builtIn="true"
       value="__i386__">
  </listOptionValue >
</option>
```

2.10 Enumerated Option Value

Some options are best described as a single selection from a list of choices. For example, users typically select the level of optimization they want the compiler to apply when creating a build artifact. The enumerate option value is used to define the elements of the list of choices.

Each element of an enumerated option has a *name* that will be shown to the user in the UI. It also has a *command* which should correspond to the command line option that gets passed to the tool by the builder if this element is selected.

A default element can be indicated by setting the value of *isDefault* to ‘true’. If the user has not overridden the selection in the UI, the default element will be displayed. If no default is specified, the first element in the list is assumed to be the default and is displayed to the user.

2.10.1 Schema

Attribute	Description	Required
id	A unique identifier for the tool that will be used by the build model.	yes
name	A descriptive name that will be displayed to the user in the UI as one of the option values to select.	yes
isDefault	Flags this enumerated value as the default to apply to the option if the user has not changed the setting.	no
command	The command that the enumerated value translates to on the command line.	yes

2.10.2 Example

The option below shows an enumerated option to flag the language dialect for the Gnu preprocessor.

```
<option name="Source Language" valueType="enumerated" …>
  <enumeratedOptionValue
       name="C"
       command="-x c"
       isDefault="true"
       id="source.language.c">
  </enumeratedOptionValue>
  <enumeratedOptionValue
       name="C++"
       command="-x c++" …>
  </enumeratedOptionValue>
</option>
```

3 UI Representation

In addition to controlling the way a project is built, the build model also defines how the user interface will appear. There are two principle ways a user interacts with the build settings model. The first is at project creation time through the New Project wizards, the second is through the build settings property page.

3.1 New Project Wizard

The new project wizard relies on the target and configuration settings from all specified tool chains to populate the list of choices it presents to the user. The figure below shows how the list of targets is populated with any target whose *isTest* and *isAbstract* attribute are set to `false`. The value of the target's *name* attribute is used to populate the drop-down list-box selection widget. Similarly, the configuration check list is populated with all the defined configurations associated with the selected target. Note that the target selection widget is labelled *Platform* in the UI. This will change in the next iteration of the build system as we further refine the concept of host target and build target.

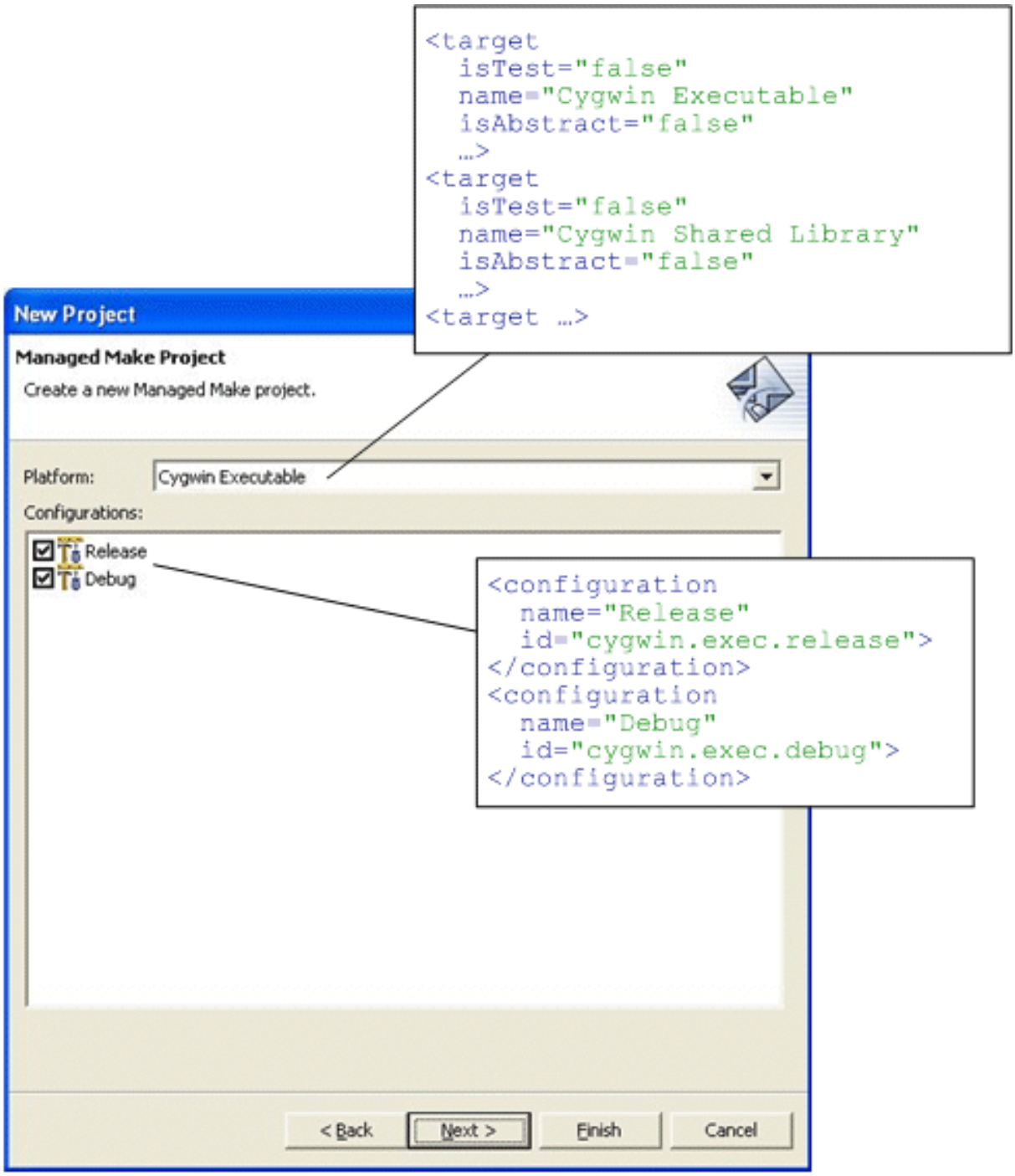


Figure 3 New project wizard

3.2 Build Property Page

The contents of the build property page for a project are created by examining the tools, option categories, and options defined for the current configuration and target. In this section we will look at how the user interface interprets the information in the build model to display options to the user.

The configuration information pane of the build property page consists of two -boxes. The first is populated with a list of all targets that apply to the project. The second contains a list of configurations that are defined for the target currently selected in the first. The figure below shows a project targeted solely at a Cygwin executable, with two configurations ‘Release’ (not shown), and ‘Debug’. Note that the build settings model is queried for the target and

configuration *name* information.

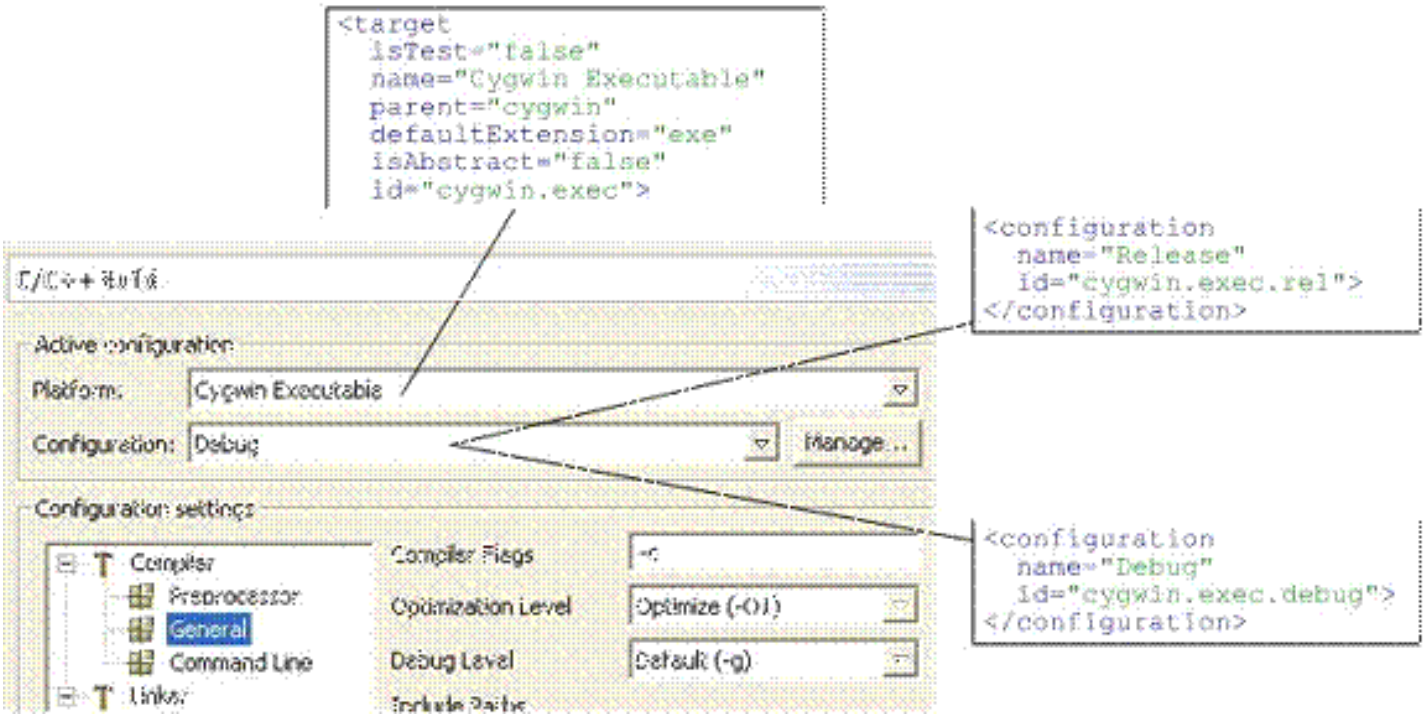


Figure 4 Configuration selection

Users change the build settings for options associated with categories and tools. The UI relies on the information in the build settings model for that information. The figure below shows how the tool list, displayed in a tree view, is populated. Tools are the root elements of the tree. Categories are displayed as leaves of the tool they belong to. In both cases, the name defined in the plug-in manifest is used as the text of the tree elements. Note that the tool uses an externalized string to identify its name to help internationalize a tool specification, but this is not necessary.

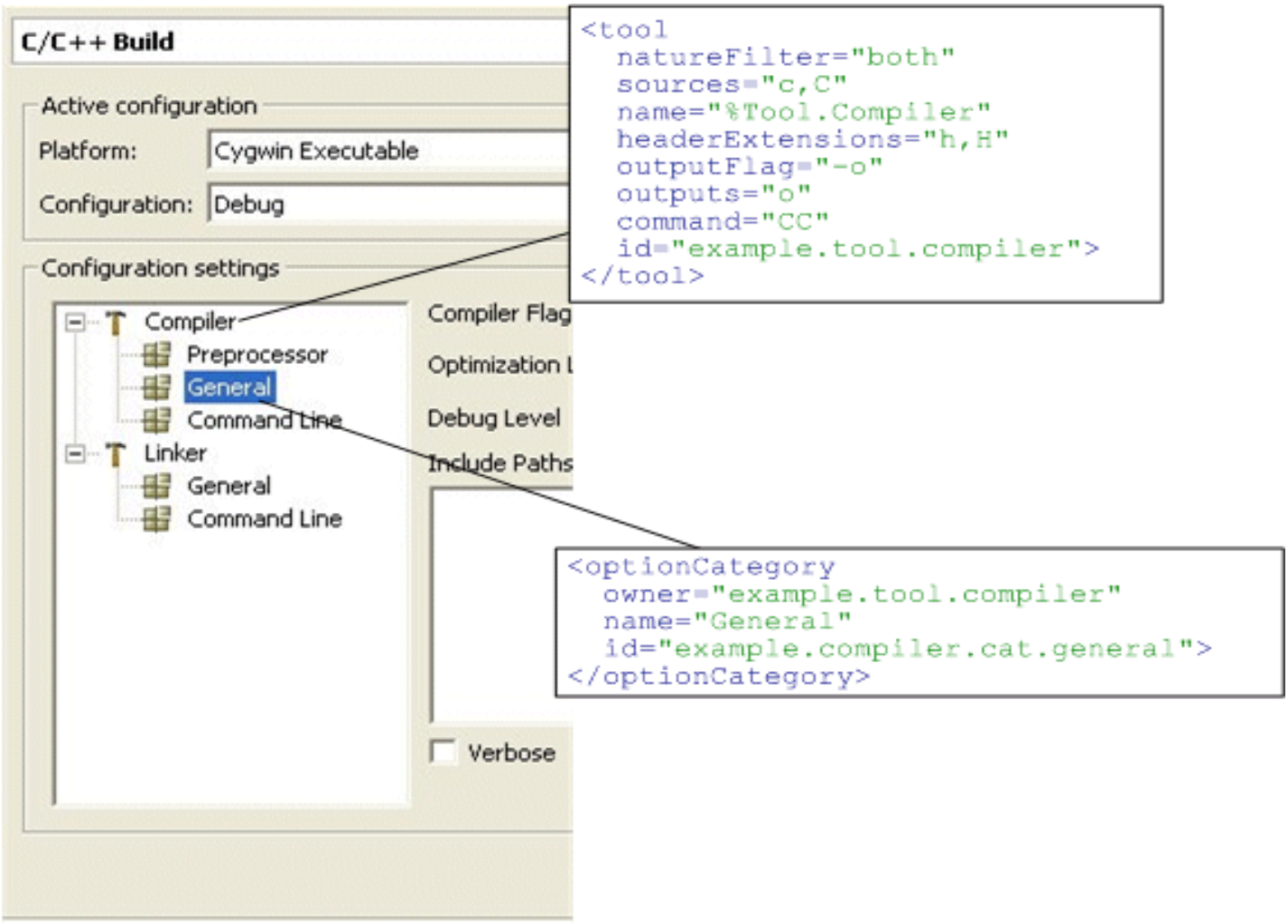


Figure 5 Tools and option category display

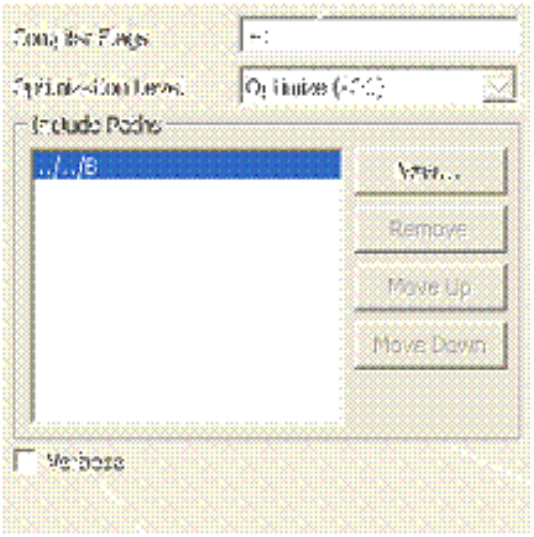
As mentioned in the discussion of the build settings model, options know what type of data they manage. Different option types require different UI widgets to properly represent that data to the user. The figure below shows what UI elements are created for each type of option.

The Compiler Flags option contains a *string* option. In this example, the option is intended to be the place the user enters all those extra flags that are not defined anywhere else in the property page. Options containing strings display their contents in a simple entry widget.

The Optimization Level option is an *enumerated* option. These types of options force the user to select a single value from a list of possible choices. Note that the *name* of the option is applied to the label in the UI, whereas the *name* of each individual *enumeratedOptionValue* element is used to populate the list.

The Include Paths option is a special case of a *stringList* option. The contents of this option are left undefined in this example, so the user sees an empty list. However, all list options are displayed in a list control with an associated button bar for adding, removing, and reordering list elements. Note that the *optionType* attribute is set to `includePath`. This notifies the build system that it must pay special attention to the values entered in this option. There are clients of this information in the CDT that will query the build system for this information, and this is currently the only way to flag these values as special.

```
<option
   defaultValue= "-c"
   name="Compiler Flags"
   category= "cygwin.compiler.category.general"
   valueType= "string"
   id="cygwin.compiler.general.ccflags" >
</option>
```




```
<option
   name="Optimization Level"
   category= "cygwin.compiler.category.general"
   valueType= "enumerated"
   id="cygwin.compiler.general.optimization.level" >
   <enumeratedOptionValue
      name="None (-O0)"
      command= "-O0"
      id="cygwin.optimization.level.none" >
   </enumeratedOptionValue>
   <enumeratedOptionValue
      name="Optimize (-O1)"
      command= "-O1"
      id="cygwin.optimization.level.optimize" >
   </enumeratedOptionValue>
   ...
</option>
```

```
<option
   name="Include Paths"
   category= "cygwin.compiler.category.general"
   command= "-I"
   valueType= "includePath"
   id="cygwin.compiler.general.include.paths" >
</option>
```

```
<option
   defaultValue= "false"
   name="Verbose"
   category= "cygwin.compiler.category.general"
   command="-v"
   valueType= "boolean"
   id="cygwin.compiler.general.verbose" >
</option>
```

Figure 6 Option display

Finally Verbose, a Boolean option, is displayed as a check-box. Since the default value for this option is defined as `false`, the check-box is left unselected when it is created.

Note that the UI actually builds itself *on the fly* based on the options descriptions in the plug-in manifest. The order of the options is the basis of the page layout. If the layout is not satisfactory, you must edit the plug-in file itself. You must then restart the workspace after editing the manifest for your changes to take effect in the UI.

4 Makefile Generator

The third key element of the managed build system is the makefile generator. The makefile generator is one of the key clients of the information stored in the build settings model. The best way to understand how the makefile generator works is to look at a real project. The figure below shows the project that we will be using for the purposes of this discussion. The source for the project is spread over the directories `source1/`, `source2/`, and `source2/source21`. Header files are located in 2 locations; `headers/`, and `source2/source21`.

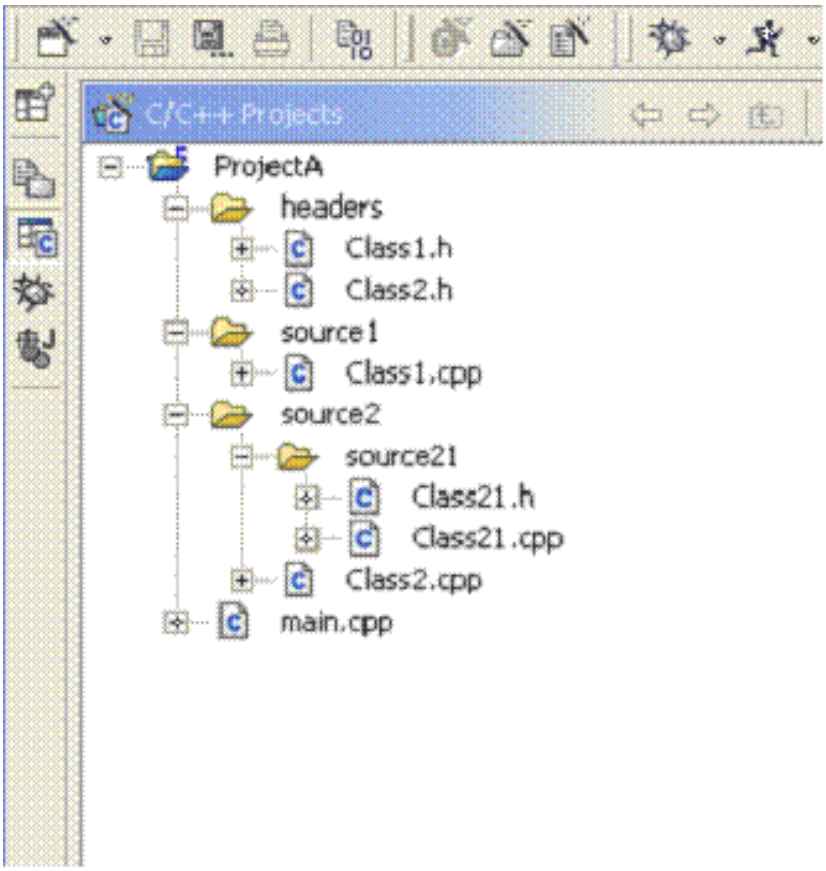


Figure 7 Example project source files

While simple, this example illustrates some of the problems projects using *make* typically face when source files are organized hierarchically. One approach to these types of problems is to generate a makefile for each subdirectory, then call make recursively, culminating in the final build step which, in theory, brings all of the build results together.

The problem with managing this type of approach lies in understanding the dependencies and handling them properly when the makefiles are generated. In order for this to happen, all the dependencies have to be properly specified and complete. As long as there are no dependencies between resources in different subdirectories, the makefiles in a recursive approach will contain a properly partitioned set of dependencies. However, in a more realistic project organization, the fragmentary makefiles will have incomplete representations of the dependencies. In order to correct for this, we would have to do some of the work that *make* gives us for free.

The approach the makefile generator takes is to use a single makefile to build the entire project. To keep the makefile manageable and readable, the makefile generator creates makefile and dependency file fragments for each subdirectory of the project that contributes source code, and uses the include capability of *make* to bring them all together.

The figure below shows the makefile, makefile fragments, and dependency fragments that are generated for the project.

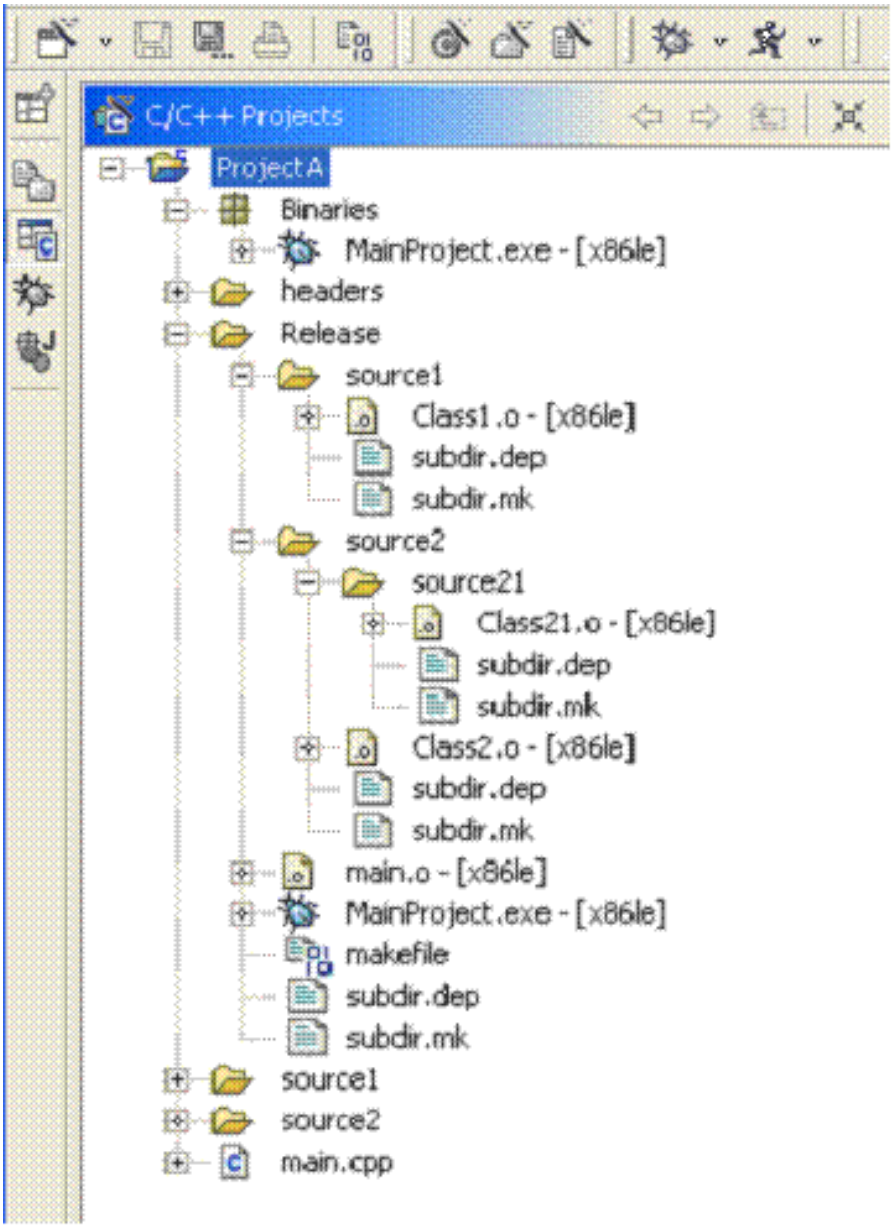


Figure 8 Generated makefiles

In the next sections, we will examine the makefiles that are generated in more detail.

4.1 Main Makefile

There is one main makefile generated for a project. Based on information for the target, the proper clean command is defined as a macro. Note that for efficiency, the contents of macros are calculated only when they are defined or modified, thus all assignment operators are generated as ':=' or '+=' with the exception of the list of objects.

The makefile defines the macros that hold the list of build sources, but they are populated in the makefile fragments. It also contains a list of subdirectories that contribute source files to the build. The makefile generator will generate fragmentary makefiles for each of the directories, so the main makefile must include each of these fragments.

```
ROOT := ..
RM := rm -rf

# Each subdirectory must contribute its source files here
C_SRCS :=
CC_SRCS :=
CXX_SRCS :=
CAPC_SRCS :=
CPP_SRCS :=

LIBS :=

USER_OBJS :=

OBJS = $(C_SRCS:$(ROOT)/%.c=%.o) $(CC_SRCS:$(ROOT)/%.cc=%.o) $(CXX_SRCS:$(ROOT)/%.cxx=%.o) \
$(CAPC_SRCS:$(ROOT)/%.C=%.o) $(CPP_SRCS:$(ROOT)/%.cpp=%.o)

# Every subdirectory with source files must be described here
SUBDIRS := \
source2/source21 \
source2 \
source1 \
. \

# Include the makefiles for each source subdirectory
-include ${patsubst %, %/subdir.mk, $(SUBDIRS)}


all: MainProject.exe

MainProject.exe: $(OBJS)
	g++ -o $@ $(OBJS) $(USER_OBJS) $(LIBS)

clean:
	-$(RM) $(OBJS) MainProject.exe

.PHONY: all clean deps

# Include automatically-generated dependency list:
-include ${patsubst %, %/subdir.dep, $(SUBDIRS)}
```

This makefile is passed as an argument to *make*, so it contains the real build target, along with *clean* and *all* targets. Finally, the makefile generator will calculate dependencies for each of the source files in the build, and generate these into a dependency fragment for each subdirectory. The main makefile includes each of the fragments as well.

4.2 Makefile Fragments

Obviously, the makefile we just looked at is incomplete. There are no rules for building actual source files, and no source files listed. However, the makefile generator places that information into makefile fragments for each subdirectory contributing source to the build. The figure below shows what the fragment for the `source1/` subdirectory looks like.

```
# Each subdirectory must contribute its source files here
C_SRCS += \
$(addprefix $(ROOT)/source1/,\
)

CC_SRCS += \
$(addprefix $(ROOT)/source1/,\
)

CXX_SRCS += \
$(addprefix $(ROOT)/source1/,\
)

CAPC_SRCS += \
$(addprefix $(ROOT)/source1/,\
)

CPP_SRCS += \
$(addprefix $(ROOT)/source1/,\
Class1.cpp \
)

# Each subdirectory must supply rules for building sources it contributes
source1/%.o: $(ROOT)/source1/%.cpp
	g++ -I. -I.. -IC:\eclipse\runtime-workspace\ProjectA\headers -O3 -gstabs -Wall -c -o $@ $<
```

The fragment contributes one file, `class1.cpp`, and a rule to build all source files with the ‘cpp’ extension. The content of the dependency and command lines is derived from the build settings model. For the dependency line, the makefile generator asks the build model if there are any tools that build files with a particular extension. If so, the tool is asked for the extension of the output. For the command line, the tool that builds for the extension supplies the actual command, while the options for the tool supply the arguments to pass to it.

4.3 Dependency Makefile Fragments

There is one final piece to the puzzle, and that is a list of dependencies for each source file in the build. Recall that *make* will rebuild any file that is out of date in its dependency graph, but it only adds the dependency to the graph if it is explicitly told to do so. Thus, it is the responsibility of the makefile generator to completely describe all dependencies for *make*. Consider the dependencies of the final build target to *Class1*, as shown in the graph above. We can see that *make* will need to rebuild `Class1.o` if `Class1.cpp, Class1.h` or `Class2.h` changes. In the makefile fragment, we have only defined a dependency between files with an ‘o’ and a ‘cpp’ extension.

The makefile generator places the remaining, explicit dependencies in a separate makefile fragment for each subdirectory. The figure below shows the fragment for the `source1/` subdirectory.

```
# Automatically-generated dependency list:
source1/Class1.o: \
C:\eclipse\runtime-workspace\ProjectA\headers\Class1.h \
C:\eclipse\runtime-workspace\ProjectA\headers\Class2.h
```

4.4 Inter-Project Dependencies

A project may reference one or more additional projects in the workspace. The makefile generator attempts to generate these dependencies in two ways. First, the makefile must have a dependency on the build goal of the referenced project in the main target, and it must include instructions for building those targets as a separate rule.

For the remainder of this discussion, let us consider the following basic scenario. Project A builds an executable, `a.`

exe. It references project B which builds a library libB.a. The main build target in the makefile for project A would be generated with the output of project B as a dependency.

```
all: deps A.exe

deps:
	cd C:/Eclipse/runtime-workspace/ProjectB/Release && $(MAKE) clean all

A.exe: $(OBJS) C:/Eclipse/runtime-workspace/ProjectB/Release/libB.a
	g++ -o $@ $(OBJS) $(USER_OBJS) $(LIBS)
```

As you can see from the generated makefile above, the rule for the target A.exe will be evaluated if the output of B has changed. This works well if the output of project B can be determined. However, that is only the case when project B is managed. Standard make projects do not know what the output of their build step is since that information is encoded in the makefile. If project A references a standard project, it will not have an explicit dependency on the output of that project.

The second element of the inter-project dependency is the rule to build the dependent project. This is generated as part of the deps target to ensure that the output of B is up-to-date when A is built. The rule to build the referenced project is simply a command to change to the appropriate build directory of the referenced project and call make again. Note that $(MAKE) will evaluate to the same invocation that was used to build the main project including the flags that were passed to it.

5 Tutorial: An Example Tool Chain

New managed build system tool chains are specified by extending the ManagedBuildInfo extension point defined in the org.eclipse.cdt.managedbuilder.core plug-in. The easiest way to do this is to create a new plug-in project and add your own definitions there.

5.1 Setting up your Environment

If you are starting with a clean environment, you will need to import the plug-ins org.eclipse.cdt.core, org.eclipse.cdt.make.ui, and org.eclipse.cdt.managedbuilder.ui (and any plug-ins they require) into your run-time workbench instance by performing the steps below. If you already have the required plug-ins in your workbench instance, proceed to the section "Creating your plug-in project".

1. From the resource perspective, use **File > Import... > External Plug-ins and Fragments**.
2. Continue clicking on the **Next >** button until you get to the screen called **Selection**. This screen will contain a list of all of the plug-ins in your host workbench instance.
3. Select org.eclipse.cdt.core, org.eclipse.cdt.make.ui, and org.eclipse.cdt.managedbuilder.ui from the list and then click the button **Add Required Plug-ins**.
4. Click on the **Finish** button. Your Navigator view should contain the selected plug-ins and all of the plug-ins they require.

5.2 Creating your Plug-in Project

You will need to create a project to add your tool chain definition. Technically the extension can be defined in any plug-in manifest, but for this tutorial we will create a new, empty plug-in project with an empty plug-in manifest file.

1. Open the **New Project...** wizard (**File > New > Project...**), choose **Plug-in Project** from the **Plug-in Development** category and click the **Next >** button.
2. On the **Plug-in Project Name** page, use `org.eclipse.cdt.example.toolchain` as the name for your project, and click the **Next >** button.
3. On the **Plug-in Project Structure Page** you will see that the wizard has set the id to `org.eclipse.cdt.example.toolchain` by default. We are going to be defining the tool chain in the plug-in manifest file without writing any code, so choose the **Create a simple project** radio button and click on the **Finish** button.
4. If asked if you would like to switch to the **Plug-in Development** perspective, answer **Yes**.

5.3 Creating the Extension

You have added the required plug-ins to your workspace instance and you have a brand new project with an empty manifest file. We are now ready to add our tool chain definition to the managed build system by extending the `ManagedBuildInfo` extension point.

1. Double click on the `org.eclipse.cdt.example.toolchain` project in the **Package Explorer** to expand it. Double click on the **plugin.xml** file to edit its contents.
2. We have to add a dependency between our project and the `org.eclipse.cdt.managedbuilder.core` plug-in where the extension point is defined. Click on the **Dependencies** tab located along the bottom of the manifest editor. Click the **Add…** button located beside the **Required Plug-Ins** list. Select `org.eclipse.cdt.managedbuilder.core` from the list and then click the **Finish** button.
3. Select the **Extensions** tab located along the bottom of the manifest editor. Click the **Add…** button located beside the **All Extensions** list. Make sure that **Generic Wizards** is selected in the left-hand list, and **Schema-based Extensions** from the right, and then click the **Next >** button.
4. You should now be on the **Extension Point Selection** page. Make sure that the **Show only extension points from the required plug-ins** check-box is selected. Select `org.eclipse.cdt.managedbuilder.core.ManagedBuildInfo` from the list of extension points. Use `org.eclipse.cdt.example.toolchain` as the **Point ID** for the extension, and `Example Tool Chain` for the **Point Name**. Click the **Finish** button.

5.4 Adding a Target

Now we will add a new target, configuration, and an example tool to the extension.

1. Right click on `org.eclipse.cdt.managedbuilder.core.ManagedBuildInfo` to access the context menu. Select **New > target** to add a target definition. A new target named `org.eclipse.cdt.example.toolchain.target[n]` should appear below the extension point. Right click on the new target to access the context menu and select **Properties** to open the properties editor for the new entry.
2. Let's give the new target a better name. Locate the **name** property in the **Properties** browser and click on the row to edit the value of the property. For now, let's use the name `Example Executable` for our target.
3. Set the value of the binary parser property based on the platform you will be using to create your example projects on. For example, if you are running this tutorial on Linux or Solaris, enter the value `org.eclipse.cdt.core.ELF`. If you are running the tutorial on Windows, enter the value `org.eclipse.cdt.core.PE`.
4. Now set the clean command for the target. For the purposes of this example, click on the **cleanCommand** property to edit it and enter `rm –f`.
5. Do the same for the make command. Locate the **makeCommand** property, click on it to edit the value, and enter `make`.
6. We want the new target to appear when we run the new project wizard on our host platform, so we have to

define the operating systems that the target should be visible on. Locate the **osList** property and click it to edit the value. Enter `win32` if you are running the tutorial on Windows, `linux` if you are running on one of the Linux distributions, or `solaris` if you are running on a version of Solaris.

5.5 Adding a Configuration

We have now added a basic target definition. We now want to define a default configuration. Normally, you would consider defining both a release and debug configuration, but we want to keep this example simple so we will restrict ourselves to a single configuration.

1. Right click on **Example Executable** in the **All Extensions** list. From the context menu select **New > configuration**. Click on the new configuration to bring up its properties in the property browser.
2. Click on the **name** property and edit the value to be `Test Configuration`.

5.6 Adding a Tool

We could now run the new project wizard and create a new managed project based on this target, but before we do that, let's define a tool for the target.

1. Right click on **Example Executable** to get the context menu and select **New > tool**. Give the tool the name `Compiler`.
2. Tools declare which file extensions they operate on and, optionally, the file extensions they produce. Our imaginary compiler only works on files with a 'c' or 'C' extension. Locate the **sources** property and set its contents to be a comma-separated list containing `c,C`. Note that there should not be any spaces between the values. Let us assume that the output of the compiler is an object module that has the extension 'o'. Set the value of the **outputs** property of the tool to `o`.
3. Let us assume that the tool should appear for both C and C++ projects, although this is not always the case. Locate the **natureFilter** property and select `both` from the list of choices.
4. The build model needs to know if there are any special file extensions that indicate a file is a 'header' file. Set the **headersExtension** property to be a comma-separated list containing `h,H`.
5. Tools often have a flag to specify the output of a tool. For the purposes of this example, set the **outputFlag** property to `-o`.
6. Finally, we want to specify the command that is needed to invoke the tool. For this example, we are not interested in actually calling a real tool, so just enter `ccc` as the value for the **command** property.

5.7 Testing the Target

We have now defined enough information to create a project for our new example target, so let's go test it out.

1. Make sure our example project is selected in the **Package Explorer**. Select **Run > Debug As > Run-time Workbench** to start a new run-time workbench instance that includes the new tool information you have created. You may be prompted to save the resource you were editing. If prompted, answer **Yes**.
2. In the new workspace, open the **C/C++ Development Perspective**.
3. Run the new project wizard. From the Selection page choose either a managed C or C++ project. Click the **Next >** button, give your project any name you wish, and click **Next >** again. Note, if the wizard does not display a next button, you have probably forgotten to specify the make and clean commands. You will have to add this information to the tool chain definition and restart your debugging session.
4. You should now be at the **Select a Target** page. Your new target will appear as a choice in the **Platform**

selection widget. Select it and note that the list of available configurations now contains the single configuration we defined for the target. Click **Finish**.

5. Right click on your new project in the **Navigator** or **C/C++ Project** view to access the context menu, and select **Properties** to open the property browser for the project. Select **C/C++ Build** from the choices and note that the tool we defined appears in the list.

At this point, you have no doubt noticed that the property page does not have any way to edit the settings for the tool. That is because we have not defined any options yet. It is time to edit the tool chain definition again.

5.8 Adding Tool Options

Users expect to be able to change the settings for their build tools through the property page for a project. What they see is controlled by the way options are defined in the tool chain specification. We will create an option category, and then add two example options to it.

1. Switch back to the **Plug-in Development** perspective. Right click on the `Compiler` entry in the extension description to bring up the context menu. Select **New > optionCategory** to add the category. Set the **name** of the category to `General`.
2. You must specify the id of the tool the category belongs to in the **owner** property. The simplest way to do this is to copy the **id** from the compiler and paste it into the **owner** property of the category. Click on the `Compiler` entry to open its properties. Right click on the **id** property as though you were going to edit it. Instead of typing, hit **ctrl-c**. Switch back to the option category, right click the **owner** property and hit **ctrl-v**.
3. Set the unique id of the category to anything you want, for example `example.toolchain.cat.comp`.
4. Right click on the tool, not the category, to bring up its context menu and select **New > option** to add our first option. Name the option `Include paths` and set the **valueType** property to `includePath` from the list of choices. Please refer to section 2.7 for a description of value types and options. In the **command** property, enter `-I`. In the **category** property, put the unique id of the category that you entered in step 3.
5. Add another option to the compiler. Set its name to `Other flags`. Set its **valueType** to `string`, its **category** to be the id you created in step 3, and its **defaultValue** to `-c`.

At this point, you can test how your options appear in the UI by debugging your run-time workbench. You should see something like this.

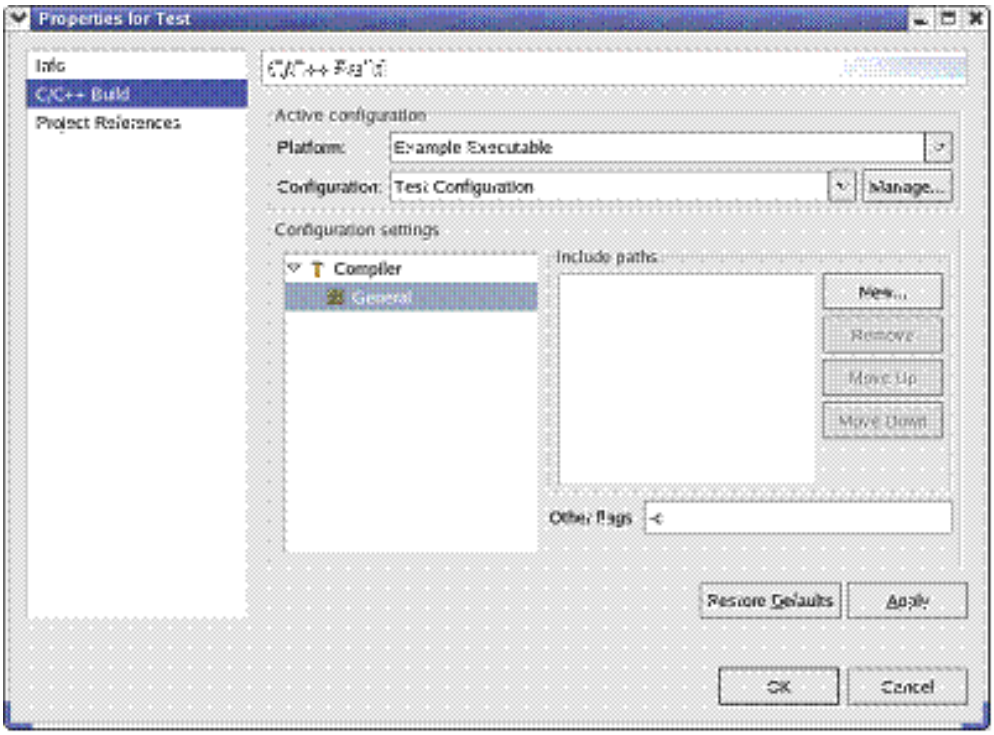


Figure 9 Property page with tool and category

5.9 Taking the Next Step

The purpose of the tutorial you just followed was to become familiar with the steps involved in creating a simple tool chain, and to get a feeling for how the choices you make in the specification of options affect the UI. In this section, we will discuss some additional points that you need to consider before specifying your own tool chain.

5.9.1 Adding More Tools

Unless you just happen to have a compiler on your system that is invoked with ‘ccc’, the example tool we created is not going to build anything. Further, the tool we defined transforms source files into object files. Another tool, like a linker, would be needed to transform those object files into a final build goal. For most targets, defining a compiler and “something else” is usually sufficient, but you may have to define additional tools if your tool chain requires intermediate build steps to function properly.

5.9.2 Defined Symbols and Header File Search Paths

There are elements of the CDT core that require build information to function properly. Things like the indexing service, search, or code-assist will only function correctly if the built-in parser can retrieve information about the paths to search for include files and the preprocessor symbols defined for the project. The build model only promises to store the type and value of an option, it does not know anything about the contents. However, you can flag certain options as special so the build model will know to pay special attention to them. As the implementer of the tool chain, you should make sure your specification has options of type “includePaths” and “definedSymbols”. The build model will pay special attention to these options and answer them to the appropriate clients in the CDT core without any further intervention on your part.

5.9.3 Built-in Symbols and Search Paths

Every compiler relies on having a correct set of preprocessor symbols and header file search paths to perform proper builds. Even compiler from the same vendor may use different symbols and search paths when building for different operating systems. Some of these values may be defined by the user, but others will be built into the tools themselves so the user will be unaware of them. Unfortunately, the CDT parser needs to know about the entire set to properly parse source files.

There are two approaches you can take, but both involve pre-populating the include path and defined symbol options with list option values containing the correct information. If you add a value to the include path or symbol option, it will be displayed to the user by default. This may be the right approach to take if you believe that users will change these values frequently. However, it will clutter the UI with values and since they are editable, users may delete them accidentally.

The alternative is to flag the list option value as a built-in value. In this case, the user will not be able to edit the values through the UI. This has the advantage of keeping the UI cleaner, but the only way for the user to edit these values if something changes is to directly edit the plug-in manifest where the extension is specified.

The CDT team is currently developing a mechanism to specify this information in an extensible way. In the current release however, we are relying on the implementers of a tool chain to supply the default symbols and paths in their specification. Please refer to section 2.9 for more details on specifying list option values.

5.9.4 User-Specified Libraries and Object Modules

Similarly, a user may want to specify external libraries to link against in the final build step. The build model needs to be told to pay special attention to an option containing libraries so that when the makefile generator requests them, it can answer a valid list. Flag the option value type as “libs” for external libraries or “userObjs” for object modules.

5.9.5 Target Hierarchies

One area of the build model that the tutorial does not touch on is the concept of abstract targets discussed in section 2.2.1. It would be quite time consuming, not to mention error prone, if you had to redefine common tools or properties each time you wanted to create a new target. Instead, the build model allows you to organize targets into hierarchies that promote the sharing of common property settings and tools between related targets. When you create a parent target though, you may not want that target to be selected by the user in the new project wizard. In that case, make the target abstract and it will no longer appear as an option for users. Flagging a target as abstract is a UI design decision; you can declare a non-abstract target as the parent of another target. You just have to be sure that you want the user to be able to create a new project based on the parent as well as the child.

5.9.6 Publishing your Plug-in

The subject of packaging Eclipse plug-ins is well covered in the *Platform Plug-in Developer Guide*. It is strongly recommended that you review this information carefully if you plan on deploying products based on Eclipse. However, making your tool-chain specification available to other users of Eclipse is not difficult. You must supply the plugin manifest we created inside the Eclipse platform's plug-in directory. The plug-in directory is named **plugins** and is typically located underneath the main directory where you installed the Eclipse platform.

1. From the **Plug-in Development Perspective**, select the **plugin.xml** file for your plug-in in the package explorer. Open the **File > Export...** wizard. On the **Select** page, chose **File system** from the export

destination list. Click the **Next >** button.

2. Make sure that `org.eclipse.cdt.example.toolchain` is selected in the left-hand list and that only **plugin.xml** is selected in the right. To select an export destination, click the **Browse...** button beside the entry widget labelled **To directory**. Browse to the **plugins** subdirectory of your Eclipse installation. Click the Finish button.
3. Restart Eclipse, switch to the **C/C++ Development Perspective** and run the new project wizard to create a new project based on your tool-chain specification.

Debug

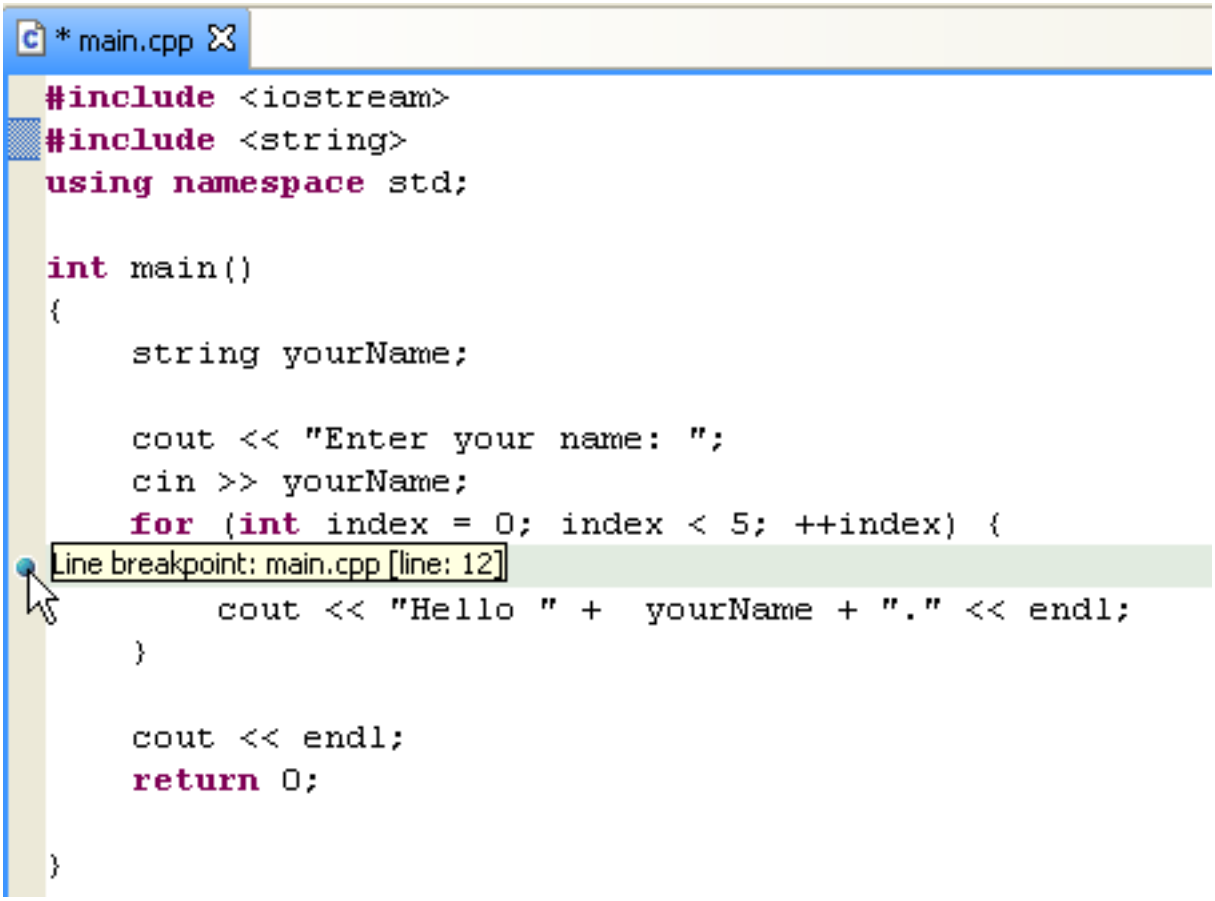
This section describes CDT debug concepts.

Breakpoints

A breakpoint suspends the execution of a thread at the location where the breakpoint is set. To set a breakpoint, right-click in the frame on the left side of an editor beside the line where you want the breakpoint, then choose **Add Breakpoint**.

Once set, a breakpoint can be enabled and disabled by right-clicking on its icon or by right-clicking on its description in the **Breakpoints** view.

- When a breakpoint is enabled, it causes a thread to suspend whenever it is hit. Enabled breakpoints are indicated with a blue circle. Enabled breakpoints that are successfully installed are indicated with a checkmark overlay.
- When a breakpoint is disabled, it will not cause threads to suspend. Disabled breakpoints are indicated with a white circle.



Debugging breakpoints are displayed in the marker bar in the editor area and in the **Breakpoints** view.

Note: Execution will also suspend if **Stop at main() on startup** is enabled on the **Launch Configuration** dialog. To access the **Launch Configuration** dialog, from the menu bar choose **Run** > **Debug**.

Related reference

Run menu

Breakpoints view

Debug overview

The debugger lets you see what's going on "inside" a program while it executes.

In order to debug your application, you must use executables compiled for debugging. These executables contain additional debug information that lets the debugger make direct associations between the source code and the binaries generated from that original source.

The CDT debugger uses GDB as the underlying debug engine. It translates each user interface action into a sequence of GDB commands and processes the output from GDB to display the current state of the program being debugged.

Tip: Editing the source after compiling causes the line numbering to be out of step because the debug information is tied directly to the source. Similarly, debugging optimized binaries can also cause unexpected jumps in the execution trace.

Related concepts

Overview of the CDT

Debug information

Related tasks

Debugging

Related reference

Run and Debug dialog box

Debug information

The **Debug** perspective lets you manage the debugging or running of a program in the Workbench. You can control the execution of your program by setting breakpoints, suspending launched programs, stepping through your code, and examining the contents of variables.

The **Debug** perspective displays the following information:

- The stack frame for the suspended threads for each target that you are debugging
- Each thread in your program represented as a node in the tree
- The process for each program that you are running

The **Debug** perspective also drives the **C/C++ Editor**. As you step through your program, the **C/C++ Editor** highlights the location of the execution pointer.

Variables

You can view information about the variables in a selected stack frame in the Variables view. When execution stops, the changed values are by default highlighted in red. Like the other debug-related views, the Variables view does not refresh as you run your executable. A refresh occurs when execution stops.

Expressions

An expression is a snippet of code that can be evaluated to produce a result. The context for an expression depends on the particular debug model. Some expressions may need to be evaluated at a specific location in the program so that the variables can be referenced. You can view information about expressions in the Expressions view.

Registers

You can view information about the registers in a selected stack frame. Values that have changed are highlighted in the Registers view when your program stops.

Memory

You can inspect and change your process memory.

Shared libraries

You can view information about the shared libraries loaded in the current debug session.

Signals

You can view the signals defined on the selected debug target and how the debugger handles each one.

Related concepts

Overview of the CDT

Debug overview

Related tasks

Debugging

Related reference

Run and Debug dialog box

Debug views

C/C++ search

You can conduct a fully or partially qualified name search. Further qualifying a search increases the accuracy and relevance of search results. The sections below provide guidance on how to control the scope of your search through the use of search delimiters, correct syntax, and wildcards.

You can search for:

- language constructs within:
  - projects in your workspace
  - selected resources from various views
  - working sets
- a working set for references to particular elements
- declarations of particular elements
- definitions of particular elements
- references of particular elements

For information on working sets, see **Workbench User Guide > Concepts > Workbench > Working sets**

What you can search for

The table below lists the element types that you can search for and special considerations to note when searching for a given element type. You can search for some or all of the element types matching a search string that you specify. If you choose to search for matching elements, all types, macros, and typdefs are included in the search.

Element	Note
Class/Struct	Searches for classes and structs. You can further qualify the search by specifying "class" or "struct" in front of the name that you are searching for. Specifying "class" or "struct" also allows you to search for anonymous classes and structures.

Function	Searches for global functions or functions in a namespace (functions that are not members of a class, struct, or union). You can specify parameters to further qualify your search. When specifying a parameter list, everything between the parentheses should be valid C/C++ syntax. Do not specify the return type of the function.
Variable	Searches for variables that are not members of a class, struct, or union.
Union	Searches for unions. Anonymous unions can be searched for by specifying "union" as the search pattern.
Method	Searches for methods that are members of a class, struct, or union. Searching for methods also finds constructors and destructors. See above note for functions.
Field	Searches for fields that are members of a class, struct, or union.
Enumeration	Searches for enumerations.
Enumerator	Searches for enumerators.
Namespace	Searches for namespaces.

How you can limit your search

You can limit your search to one or all of the following:

- Declarations
- References
- Definitions (for functions, methods, variables and fields)

You can control the scope of the search by specifying which of the following is to be searched:

- Workspace
- Working Set
- Selected Resources

Wildcard characters

You can use wildcard characters to further refine your search.

Use this wildcard character	To search for this
*	Any string **Tip:** Use the character * to search for operators that begin with *. See syntax examples in the table below.
?	A single character
: :	Nested elements

Tip: Do not use wild cards between the brackets of a function or method pattern. For example, the search string `f( * )` is an invalid search that results in a search for any function `f` because the asterisk is interpreted as a pointer rather than a wild card.

Syntax examples

The table below provides syntax examples and an explanation for each example to help you conduct an effective search.

Syntax	Searches for this
`::*::*::A`	A nested element two levels deep
`::*::*::A?`	Any two-letter name that begins with A and is two levels deep
`::A`	Searches for A not nested in anything
`*()`	Any function taking no parameters
`*( A * )`	Any function taking 1 parameter that is a pointer to type A

`f( int * )`	Will search for function f taking 1 parameter that is an int *
`f( const char [ ], A & )`	Will search for a function f, taking 2 parameters; one is a const char array, the other is a reference to type A
`operator \*`	Finds only operator *
`operator \*=`	Finds only operator *=
`operator *`	Finds all operators
`class`	Searches for anonymous classes
`struct`	Searches for anonymous structs
`union`	Searches for anonymous unions

Search results

Search results are displayed in the Search view. You can sort your search by Name, Parent Name and Path. You can also repeat your last search.

Search Concepts

Declarations

According to the ANSI C++ Spec, a declaration is a statement that “introduces a name into a translation unit or re-declares a name that has been previously introduced by a previous declaration.

All C/C++ search elements can be searched for declarations.

Definitions

Most declarations are also definitions; in other words, they also define the entity for they declare the name for. However there are some elements that can have separate definitions from their declarations.

For C/C++ search the following elements can be searched for definitions:

- Functions/Methods – the definition is where the code implementation resides
- Variable:
    1. Extern – the definition is where the variable is initialized
    2. Non extern - the definition of a variable is where it is declared
- Field:
    1. Static fields - the definition of a static field is where it gets initialized
    2. Non static fields - the definition corresponds to the fields declaration
- Namespace – the definition of a namespace is the same as its declaration

References

By selecting references, C/C++ search will return all of the places the selected element is used.

All Occurrences

Selecting ‘All Occurrences’ in the Limit To section will result in a search for declarations, definitions (if applicable) and references for whatever element or elements have been selected.

Any Element

Selecting ‘Any Element’ in the Search For section will result in a search for all of the listed elements plus macros and typedefs.

For more information, see:

- **Workbench User Guide > Concepts > Views > Search view**
- **Workbench User Guide > Tasks > Navigating and finding resources**

Related concepts

C/C++ Indexer

CDT Projects

Open Declarations

Related tasks

Searching for C/C++ elements

Navigating to C/C++ declarations

Related reference

C/C++ search page, Search dialog box

C/C++ perspective icons

C/C++ Indexer

The C/C++ indexer uses the parser to create a database of your source and header files that provides the basis for C/C++ search, navigation features and parts of content assist.

The indexer runs on a background thread and reacts to resource change events such as:

- C/C++ project creation/deletion
- Source files creation/deletion
- File imports
- Source file content changes

It is possible to customize the behavior of the indexer through the use of source folders or even turn it off completely. This customizable behavior is available on a per-project basis (i.e. it is possible to have different indexer settings for each project in your workspace).

Related concepts

C/C++ search
C/C++ Indexer Problem Reporting
C/C++ Indexer Opening or Closing a project
C/C++ Indexer Progress Bar

Related tasks

Selection Searching for C/C++ elements
Enable/Disable the C/C++ Indexer
C/C++ Indexer Problem Reporting
C/C++ Indexer - Indexer Timeout
Setting Source Folders

Related reference

Search, C/C++ Preferences window
C/C++ search page, Search dialog box
C/C++ Project Properties, Managed, Indexer
C/C++ Project Properties, Standard, Indexer

C/C++ Indexer Problem Reporting

C/C++ Index Problem reporting places a problem marker on the editor and adds an item to the error list for each preprocessor or semantic problem reported by the parser. Note that the markers will only show up the next time the file is indexed.

Note: This feature is not recommended for large projects.

Preprocessor Problems

In order for search and search related features to work properly, it is imperative that include paths are set up properly so that the parser can find source files and index them. If you suspect that your search results are lacking, you can turn on the preprocessor problem markers. These will place a marker on the line that has the preprocessor problem.

This includes:

- Pound error
- Inclusion not found
- Definition not found
- Invalid macro definition
- Invalid directive
- Conditional evaluation error

Semantic Problems

The problem markers can also indicate semantic errors in your code.

The errors flagged include:

- Name not found
- Invalid overload
- Invalid using
- Ambiguous lookup
- Invalid type
- Circular inheritance
- Invalid template

Related concepts

C/C++ search

C/C++ Indexer

C/C++ Indexer Opening or Closing a project

C/C++ Indexer Progress Bar

Related tasks

Selection Searching for C/C++ elements

Enable/Disable the C/C++ Indexer

C/C++ Indexer Problem Reporting

C/C++ Indexer - Indexer Timeout

Setting Source Folders

Related reference

Search, C/C++ Preferences window

C/C++ search page, Search dialog box

C/C++ Project Properties, Managed, Indexer

C/C++ Project Properties, Standard, Indexer

C/C++ Indexer Opening or Closing a project

The user opening a previously closed project results in the entire project being re-indexed.

Closing a project results in the index being deleted. Search features will not reperot any results for closed projects.

Related concepts

C/C++ search
C/C++ Indexer
C/C++ Indexer Problem Reporting
C/C++ Indexer Progress Bar

Related tasks

Selection Searching for C/C++ elements
Enable/Disable the C/C++ Indexer
C/C++ Indexer Problem Reporting
C/C++ Indexer – Indexer Timeout
Setting Source Folders

Related reference

Search, C/C++ Preferences window
C/C++ search page, Search dialog box
C/C++ Project Properties, Managed, Indexer
C/C++ Project Properties, Standard, Indexer

C/C++ Indexer Progress Bar

The indexer progress bar shows the progress status of the indexing jobs in the progress views.

The indexing jobs can be temporarily paused by pressing the stop button on the progress bar. This will cause the indexer to wait until the next time the user runs a search job or makes a change to an indexed element (by such actions as modifying an existing source file, deleting a file, creating a new file, moving file and so on). The indexer at this point will resume with the previously postponed indexing job before moving on to the new one.

If you wish to cease indexing all together, you can cancel an indexing job and disable the indexer through the properties.

Related concepts

C/C++ search
C/C++ Indexer
C/C++ Indexer Problem Reporting
C/C++ Indexer Opening or Closing a project

Related tasks

Selection Searching for C/C++ elements
Enable/Disable the C/C++ Indexer
C/C++ Indexer Problem Reporting
C/C++ Indexer - Indexer Timeout
Setting Source Folders

Related reference

Search, C/C++ Preferences window
C/C++ search page, Search dialog box
C/C++ Project Properties, Managed, Indexer
C/C++ Project Properties, Standard, Indexer

Searching External Files

C/C++ search, by default, will only search your workspace. If you wish to search external files that are included by files in your workspace but don’t reside in your workspace, you must enable external search markers.

When a match in an external file is now found, it will be linked into your project and you will be able to open the match from the search pane as usual.

When a project is closed, or the workbench is shutdown the links are removed.

Related concepts

C/C++ search

Related tasks

Searching External Files

Related reference

Search, C/C++ Preferences window

C/C++ search page, Search dialog box

Tasks

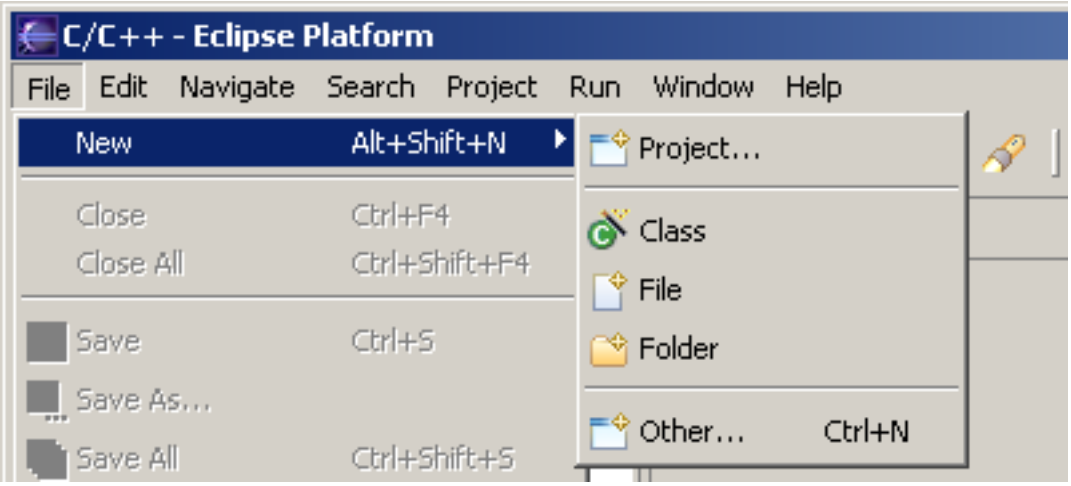
Task topics provide step-by-step procedural instructions to help you perform required tasks.

Creating a project

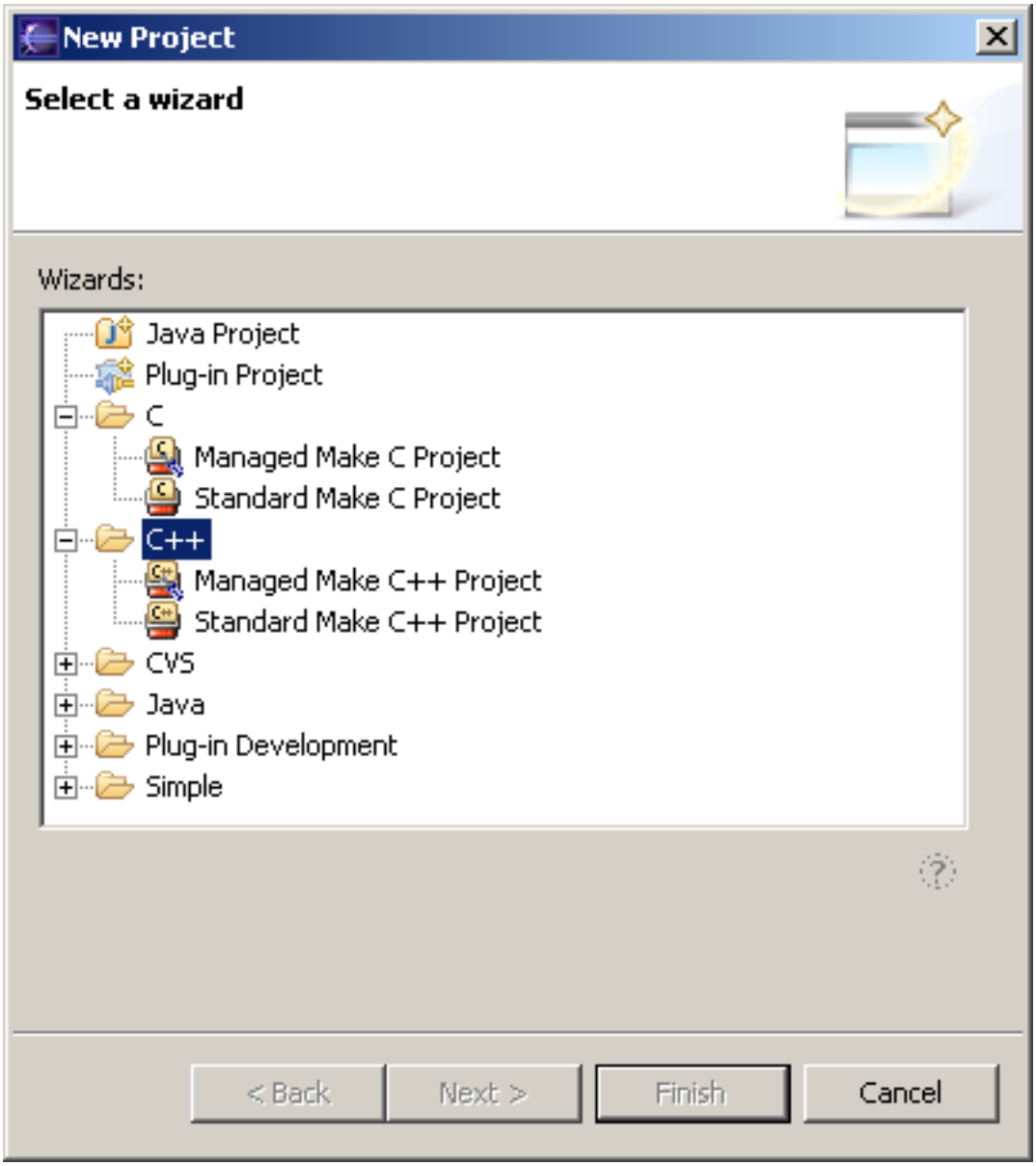
You can create a standard make or managed make C or C++ project.

To create a project:

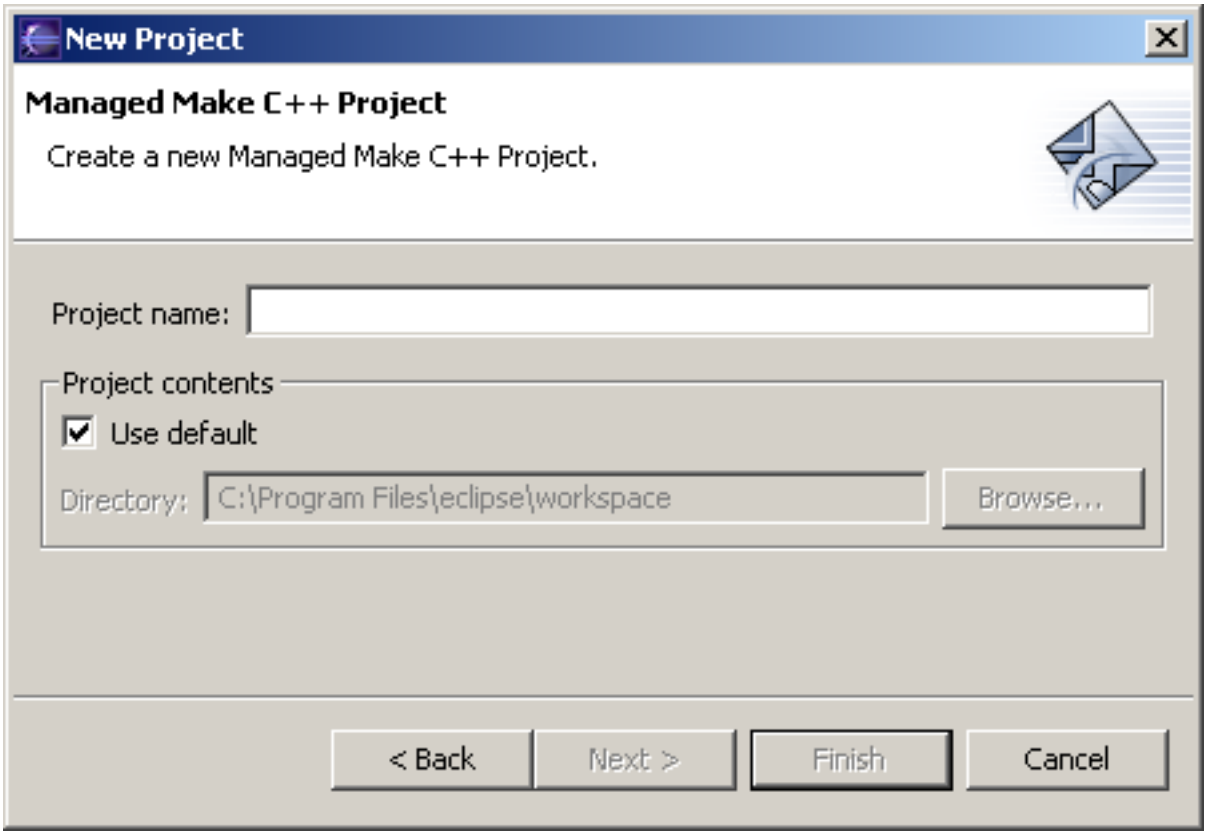
1. Click **File > New > Project**.



2. In the New Project wizard, click **C** or **C++**.
3. Choose either a **Standard Make C++ Project** or a **Managed Make C++ Project**.



4. Click **Next**.
5. In the **Name** box, type a name.
6. To specify a different directory in which to save your project, clear the **Use Default Location** check box, and enter the path in the **Location** box.
7. For managed make projects, click **Next** to select a deployment platform. For more information, see Selecting a deployment platform.
8. To create your project, click **Finish**.



9. If a message box prompts you to switch perspectives, click **Yes**.
10. Define your project properties. For more information, see Defining project properties.

Related concepts

CDT Projects

Project file views

Related tasks

Working with C/C++ project files

Related reference

Project properties

Views

Working with C/C++ project files

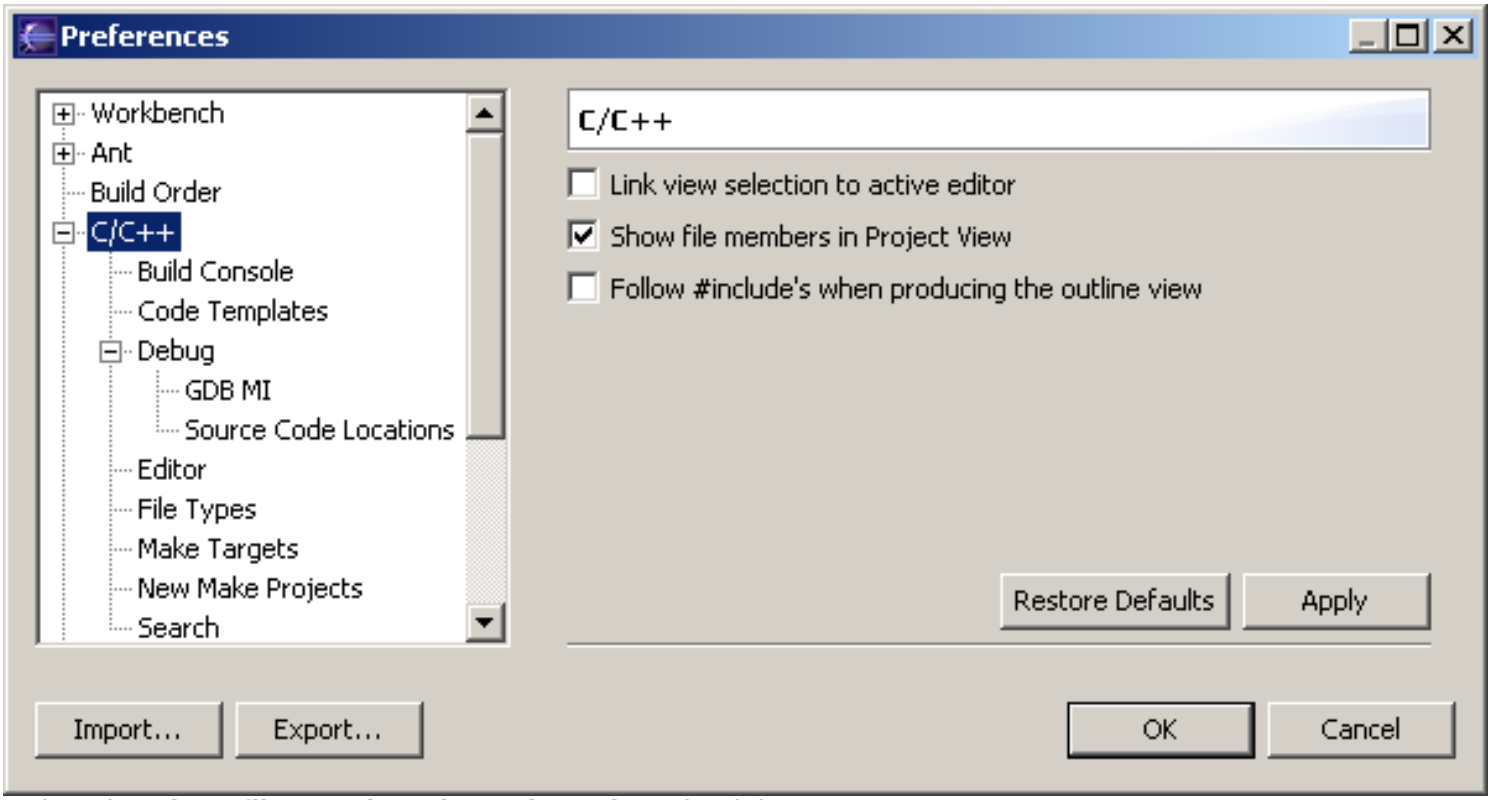
This section explains how to create and manage project files.

Displaying C/C++ file components in the C/C++ Projects view

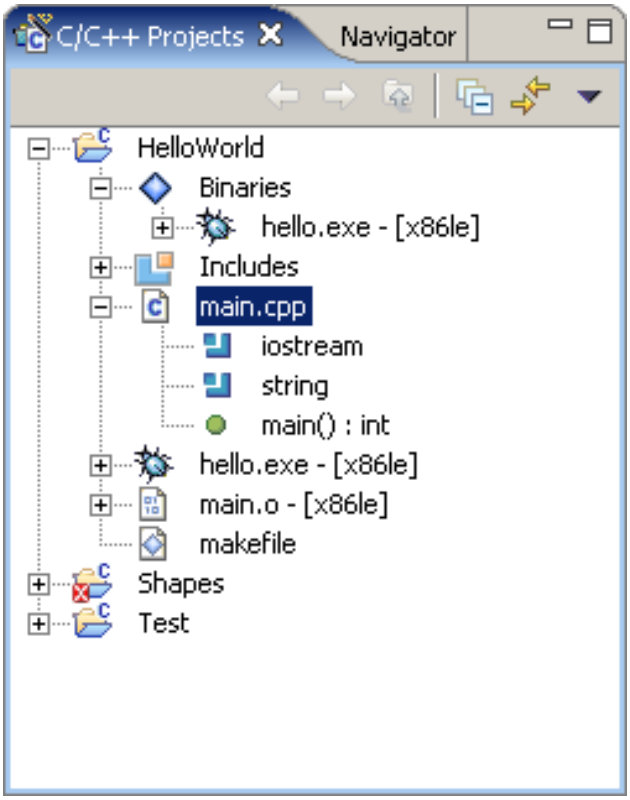
File components are displayed in the C/C++ Projects view and in the Outline view. You can display or hide all file components in the C/C++ Projects view.

To display file components

1. Click **Window > Preferences**.
2. In the Preferences dialog box, select **C/C++** from the list.



3. Select the **Show file members in Project View** check box.
4. Click **OK**.
5. In the C/C++ Projects view, double-click a file component.
   The component is highlighted in the C/C++ editor.



The C/C++ Projects view can also be filtered to show certain types of file components. For more information, see Hiding files by type in the C/C++ Projects view.

Related concepts

CDT Projects
Project file views

Related tasks

Hiding files by type in the C/C++ Projects view
Searching for C/C++ elements
Navigate to C/C++ declarations

Related reference

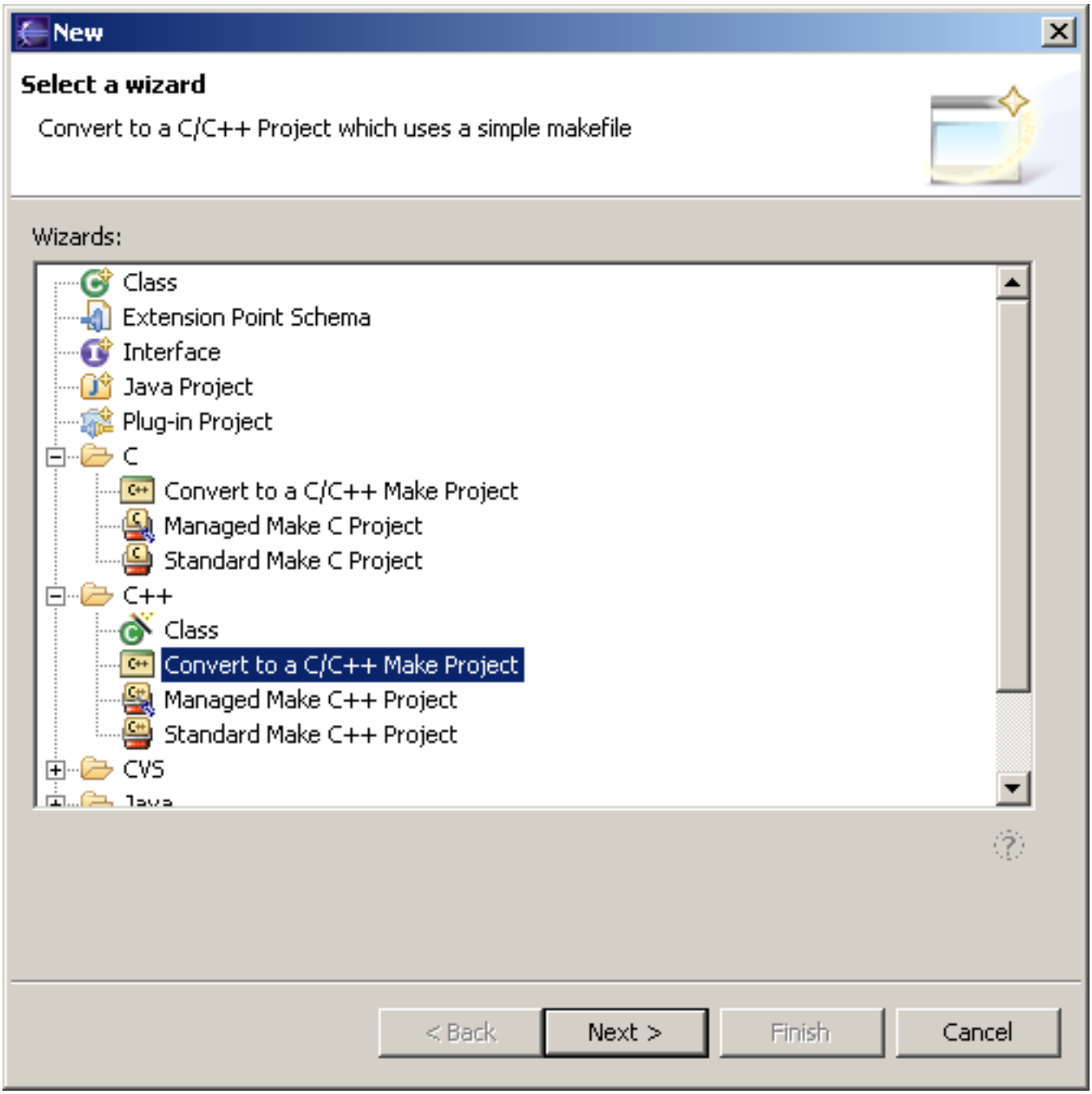
C/C++ editor, code templates and search preferences

Converting a C or C++ nature for a project

You can assign a C nature to a C++ file or vice versa.

To assign a C or C++ nature to a project

1. Click **File > New > Other**.



2. Click **C** or **C++**.
3. Click **Convert to C/C++ Make Project**.
4. Click **Next**.
5. In the Candidates for conversion list, select the projects to convert.
6. In the Convert to C or C++ box, click **C Project** or **C++ Project**.
7. Click **Finish**.

Related concepts

CDT Projects

Project file views

Related tasks

Writing code

Related reference

Project properties

Creating a C/C++ file

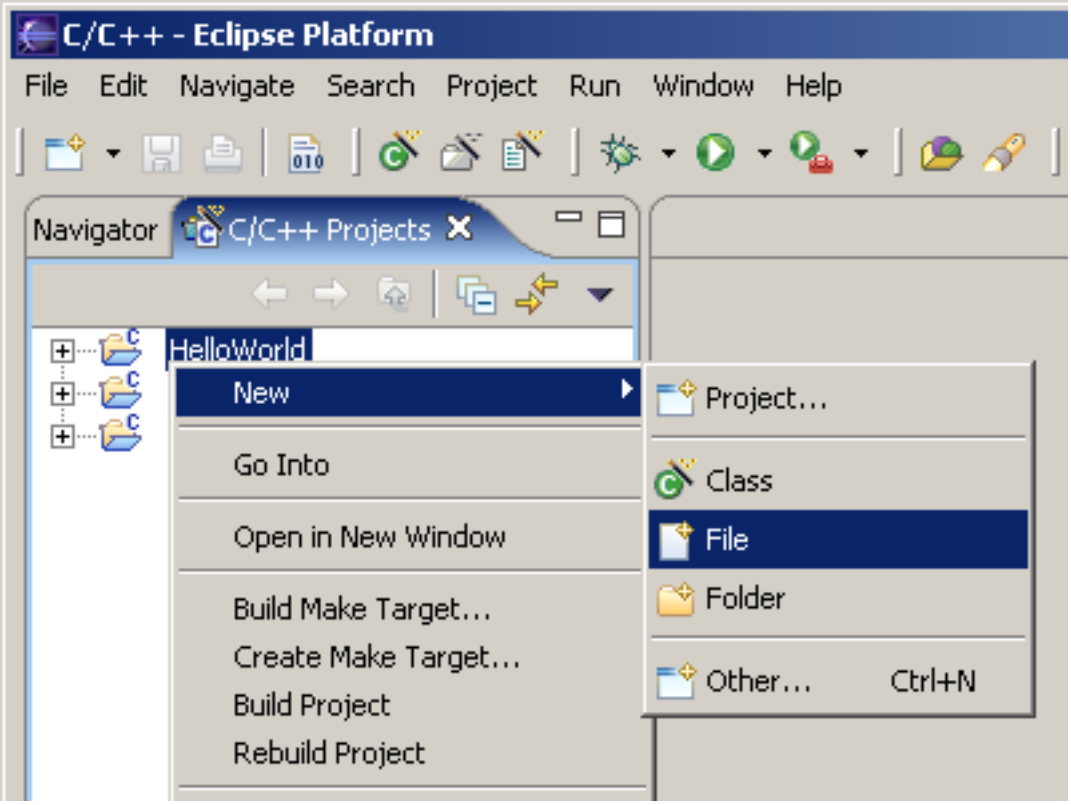
Files are edited in the C/C++ editor that is by default, located in the editor area to the right of the C/C++ Projects view.

The marker bar on the left margin of the C/C++ editor, displays icons for errors, warnings, bookmarks, breakpoints and tasks.

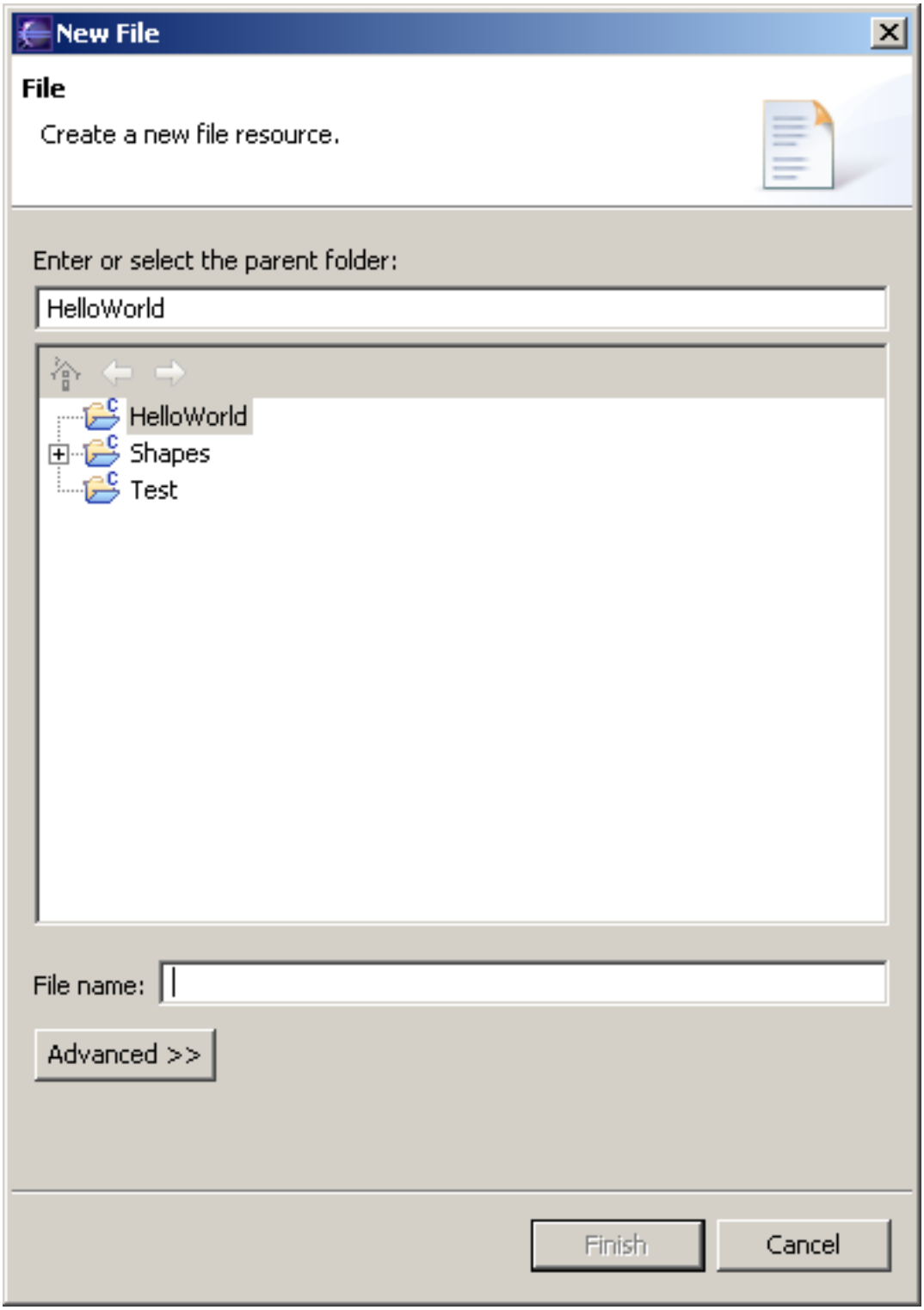
For more information on the marker bar, see **Workbench User Guide > Reference > User interface information > Views and editors > Editor area**.

To create a C++ file:

1. In the **C++ Projects** view, right-click a project, and select **New > File**.

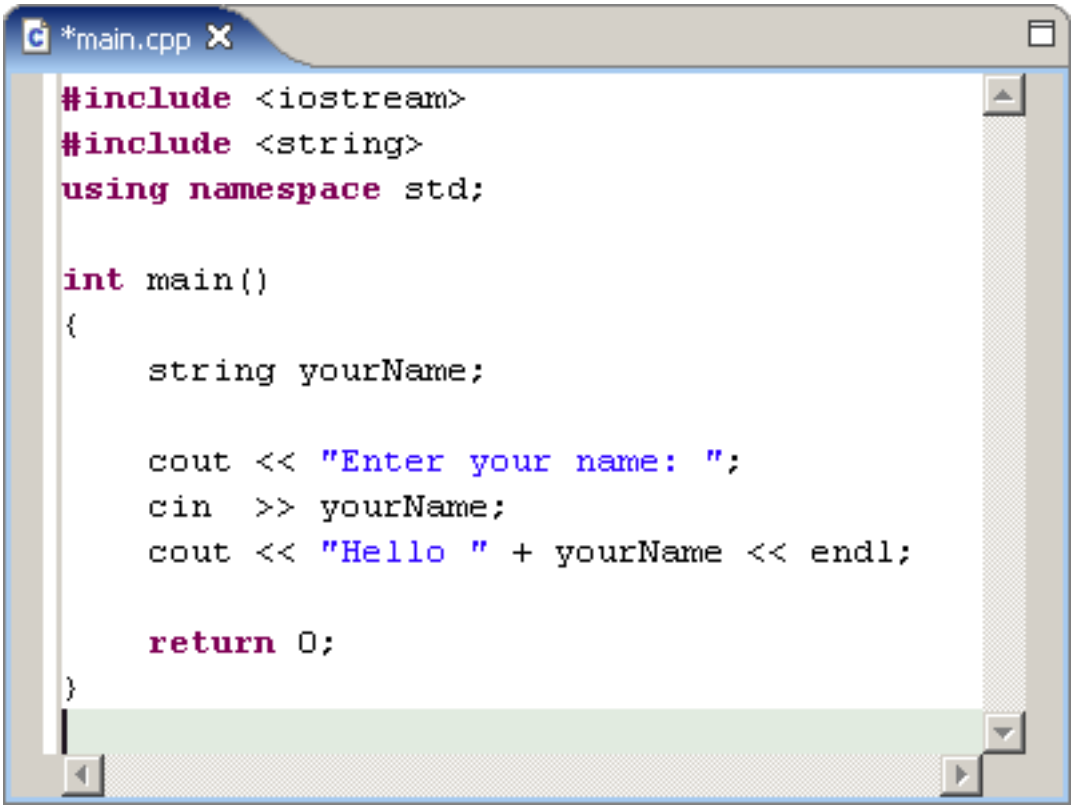


2. In the list of projects, verify that the correct project is selected.
3. In the **File name** box, type a name followed by the appropriate extension.
4. Click **Finish**.



The file will open in the C/C++ editor.

5. Enter your code in the editor view..



6. Type **CTRL+S** to save the file.

Related concepts

CDT Projects

Project file views

Related tasks

Displaying C/C++ file components in the C/C++ Projects view

Hiding files by type in the C/C++ Projects view

Related reference

Project properties

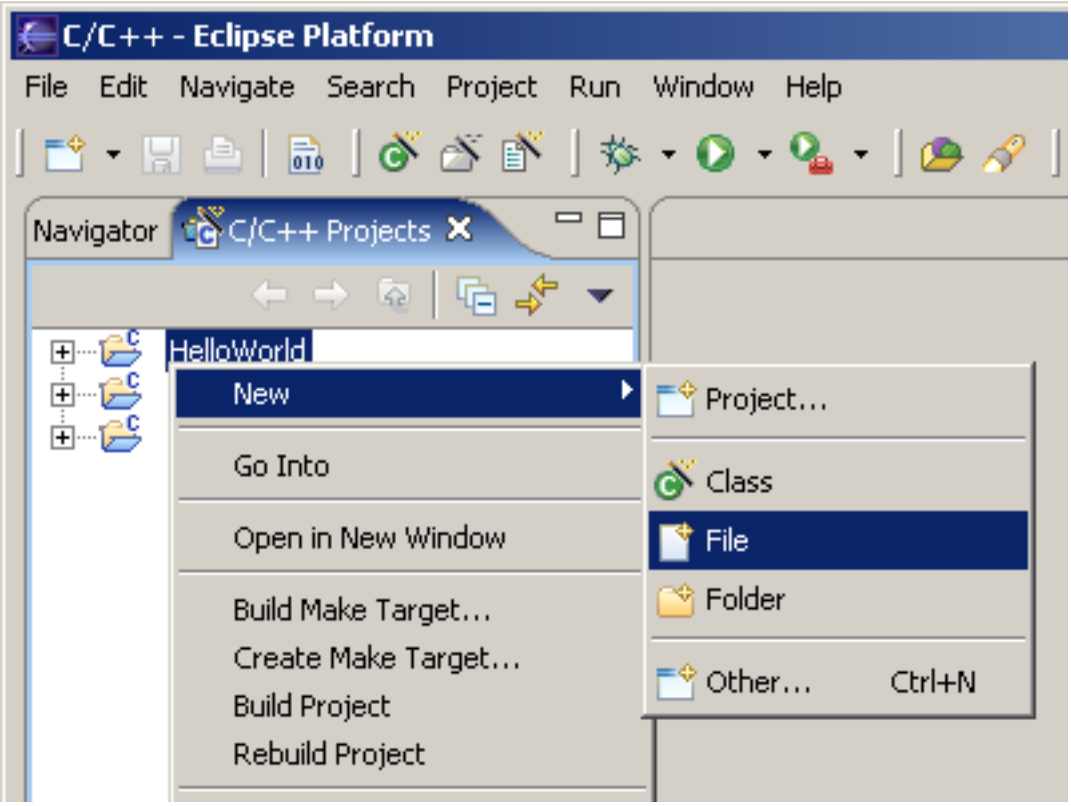
Creating a makefile

If you have created a Standard Make C/C++ Project, you need to provide a makefile.

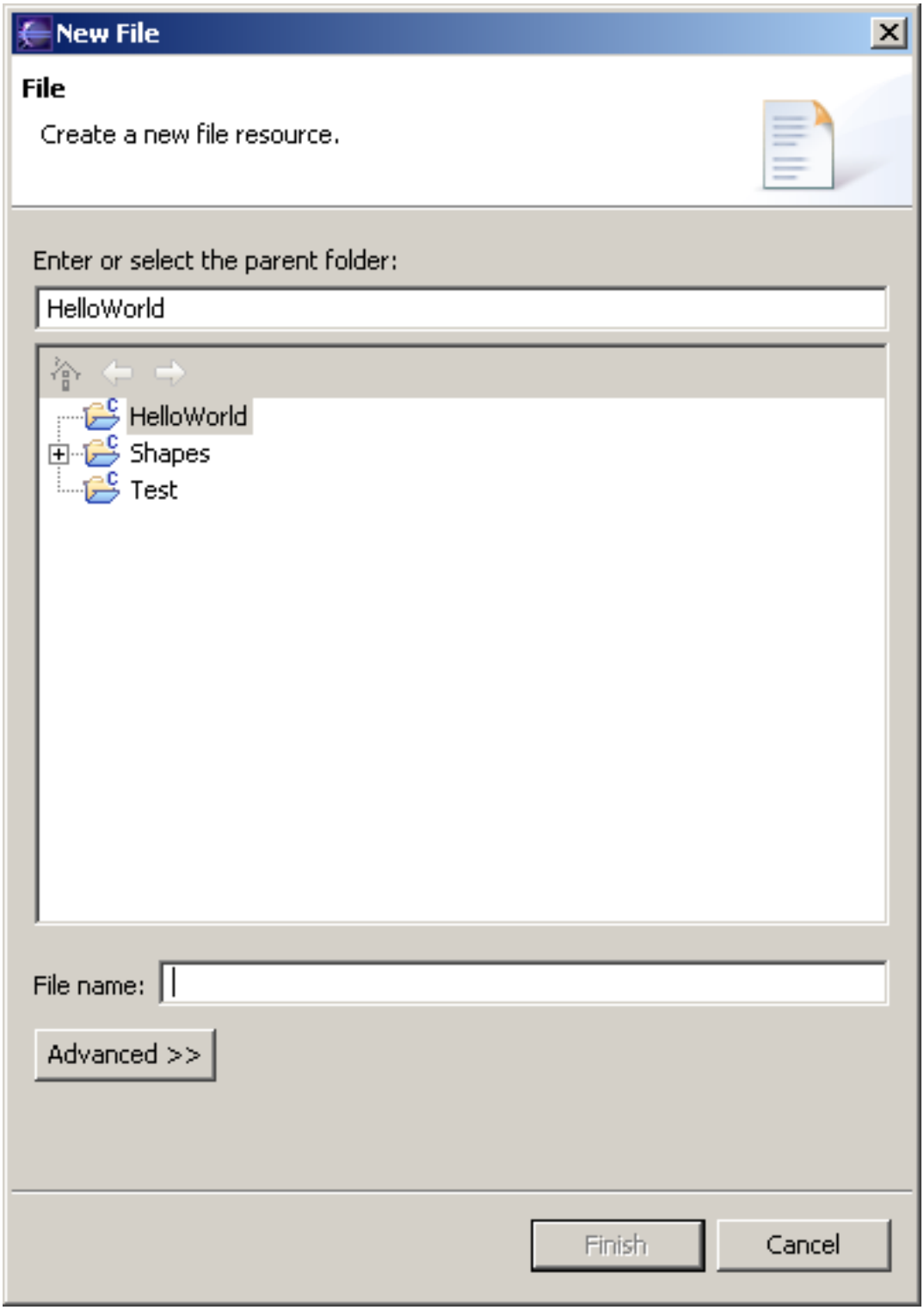
When you build a project, output from make is displayed in the Console view. Makefile actions are displayed in the Make Targets view.

To create a makefile:

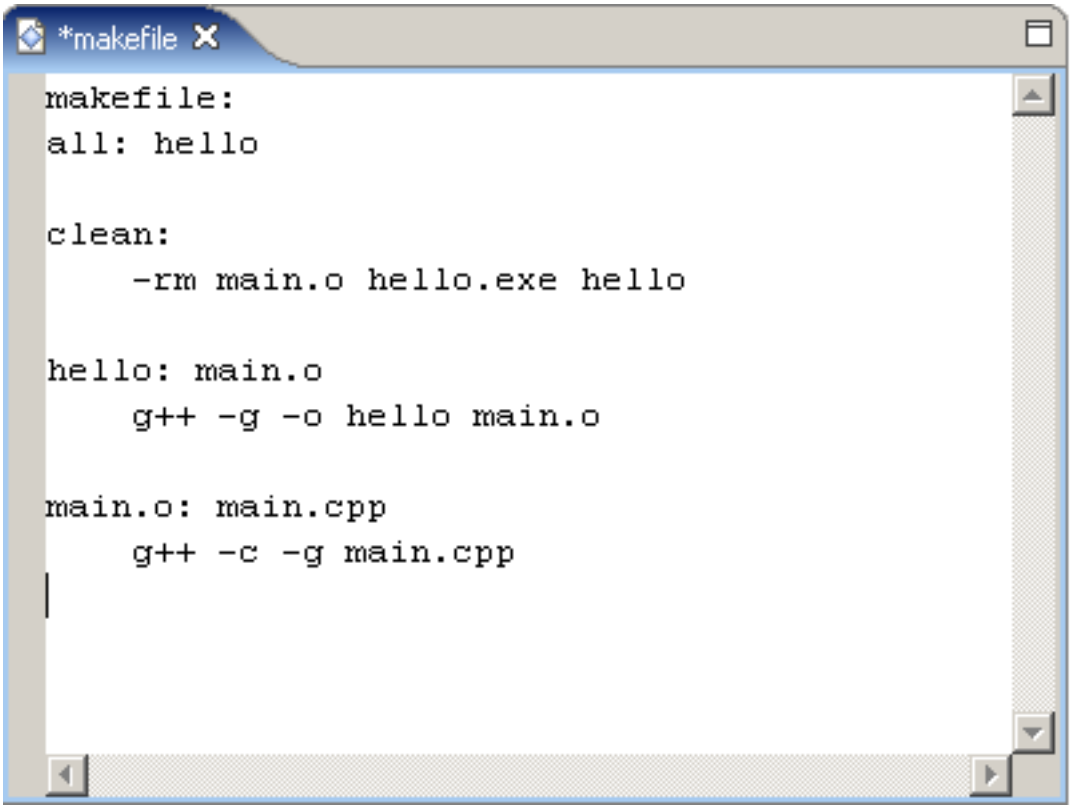
1. In the **C++ Projects** view, right-click a project, and select **New > File**.



2. In the **File name** box, type **makefile**.
3. In the list of projects, verify that the correct project is selected.
4. Click **Finish**.



5. The C/C++ editor opens. Type makefile instructions in the C/C++ editor.



6. Click **File > Save**.

Related concepts

Makefile

Working with C/C++ project files

Related tasks

Displaying C/C++ file components in the C/C++ Projects view

Hiding files by type in the C/C++ Projects view

Related reference

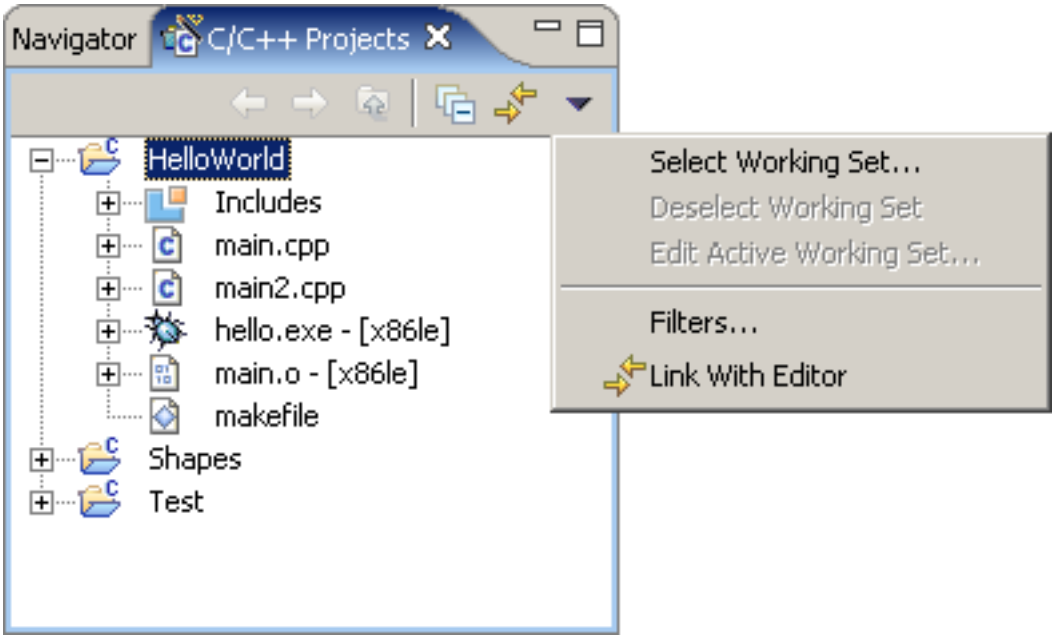
Views

Hiding files by type in the C/C++ Projects view

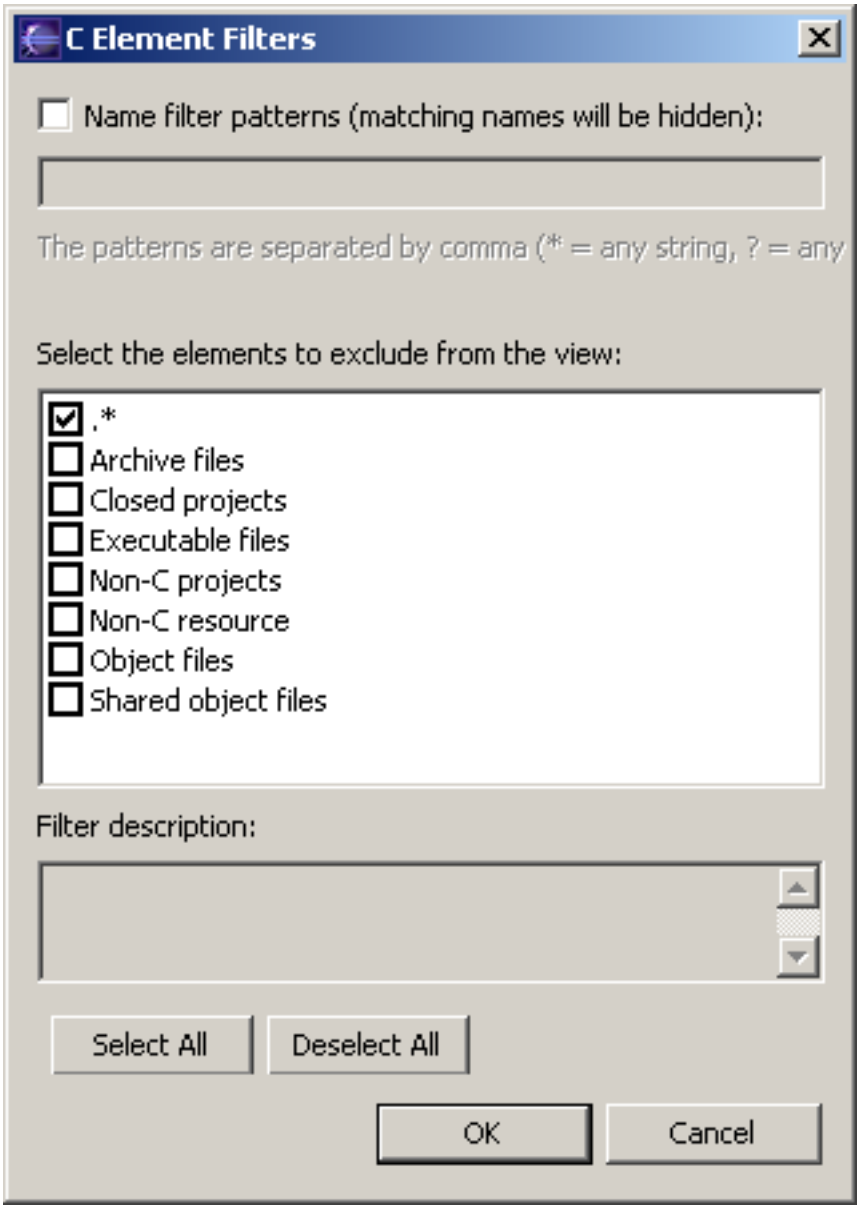
You can hide files by type that you do not want to see in the C/C++ Projects view.

To hide files by type:

1. In the C/C++ Projects view, click the **Menu** icon .



2. Click **Filters**.



3. Select the file types that you want to hide.
4. Click **OK**.

   The C/C++ Projects view refreshes automatically.

Related concepts

CDT Projects

Project file views

Related tasks

Displaying C/C++ file components in the C/C++ Projects view

Hiding files by type in the C/C++ Projects view

Related reference

Views

Converting CDT 1.x Projects

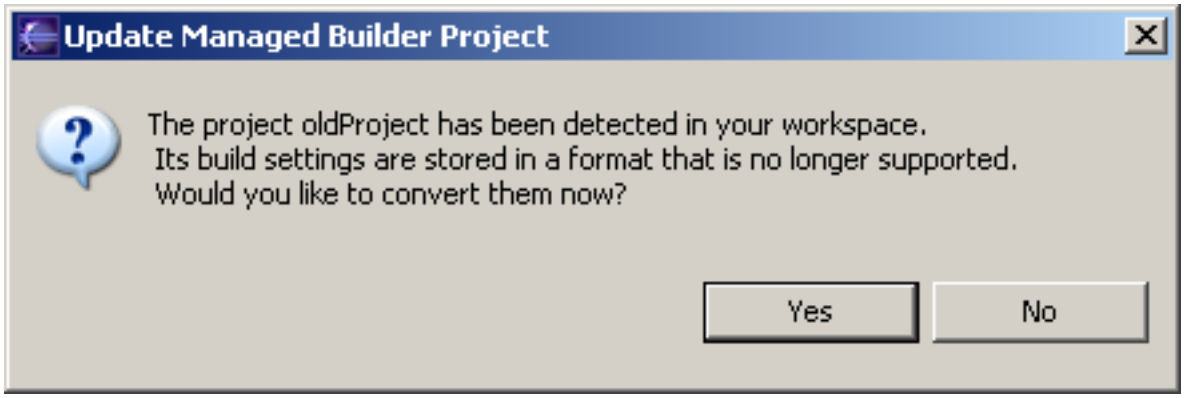
There are two ways to update CDT 1.x projects to CDT 2.0, after they have been imported, either restart eclipse, or update them using the **Convert to C/C++ Make Project Wizard**.

Note: the project must first be imported into your workspace using **File > Import > Existing Project into Workspace** before you can update the project.

For more information on importing projects see **Workbench > Tasks > Importing > Importing Existing Projects**.

Restart Eclipse

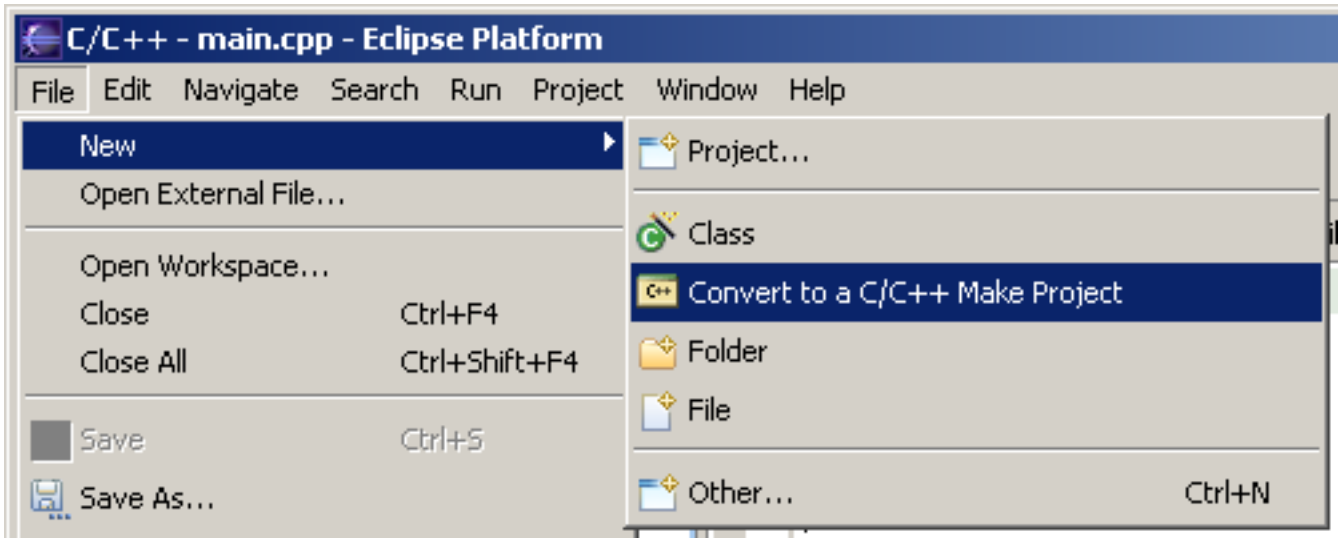
When Eclipse starts, the CDT 1.x project will be detected and you will be prompted to update the project.



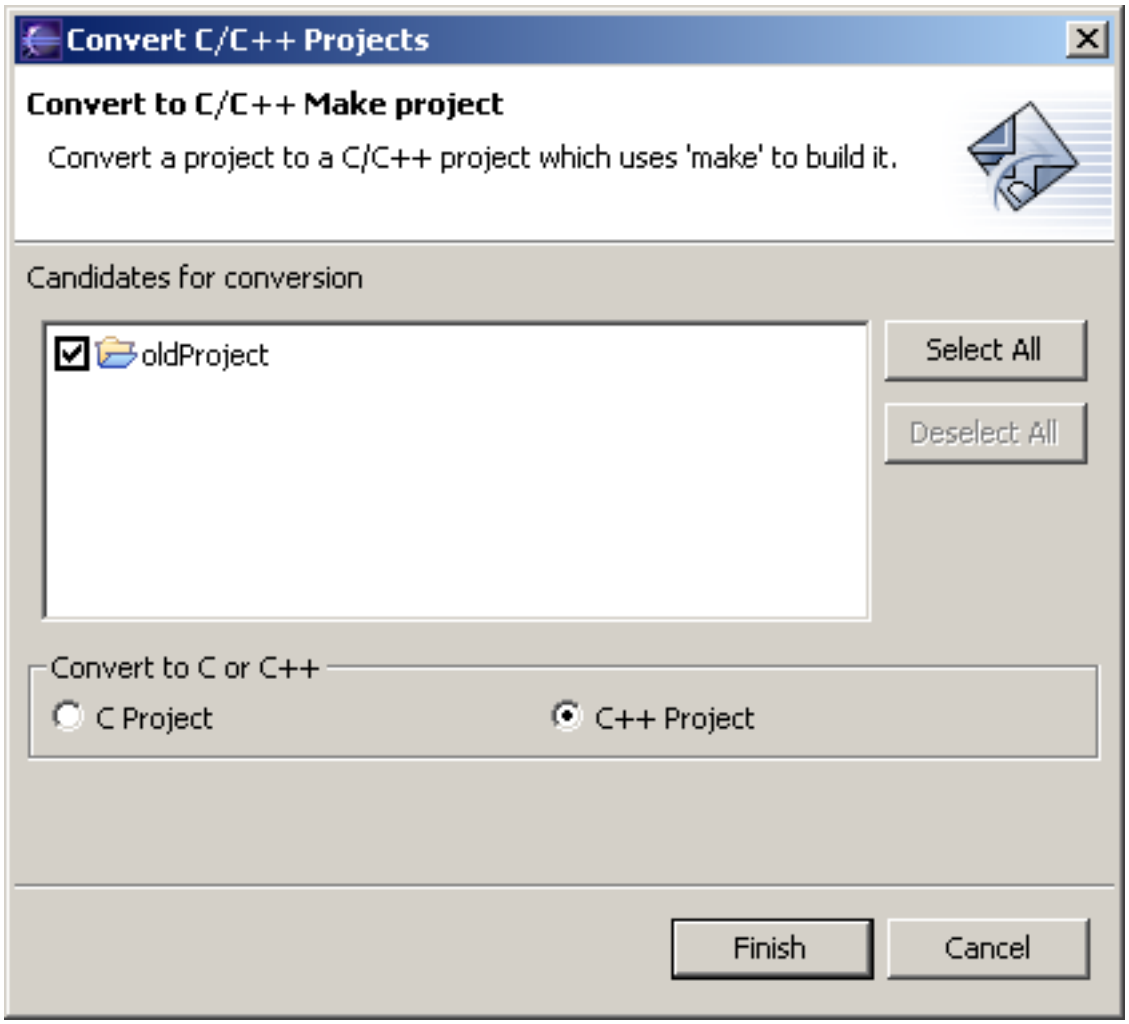
Click **Yes** and the project will be updated.

Convert to C/C++ Make Project Wizard

Select **File > New > Convert to a C/C++ Make Project**. If that selection is not available, you can find the instructions for adding it here.



From the **Convert to C/C++ Make Project Wizard** select the project you want to convert and click **Finish**.



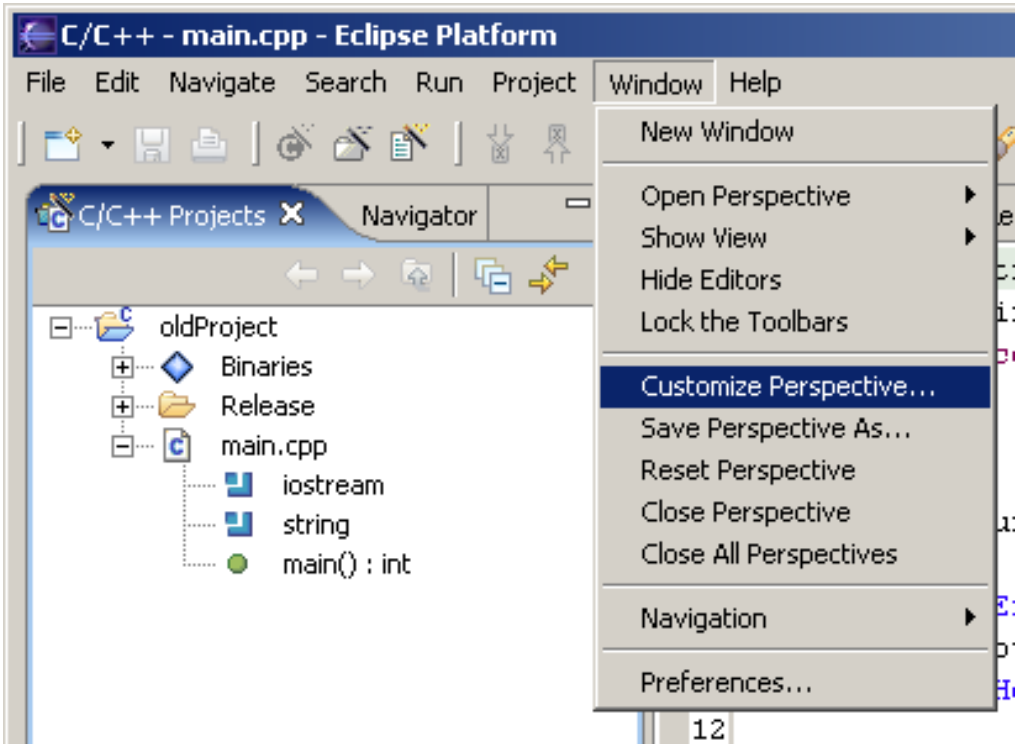
Note: You may need to manually enable Path Discovery for CDT 1.x Standard Make projects, depending on how your CDT project was configured. See Set Discovery Options for details.

Related tasks

[Adding Convert to a C/C++ Make Project to the New menu](#)
[Set Discovery Options](#)

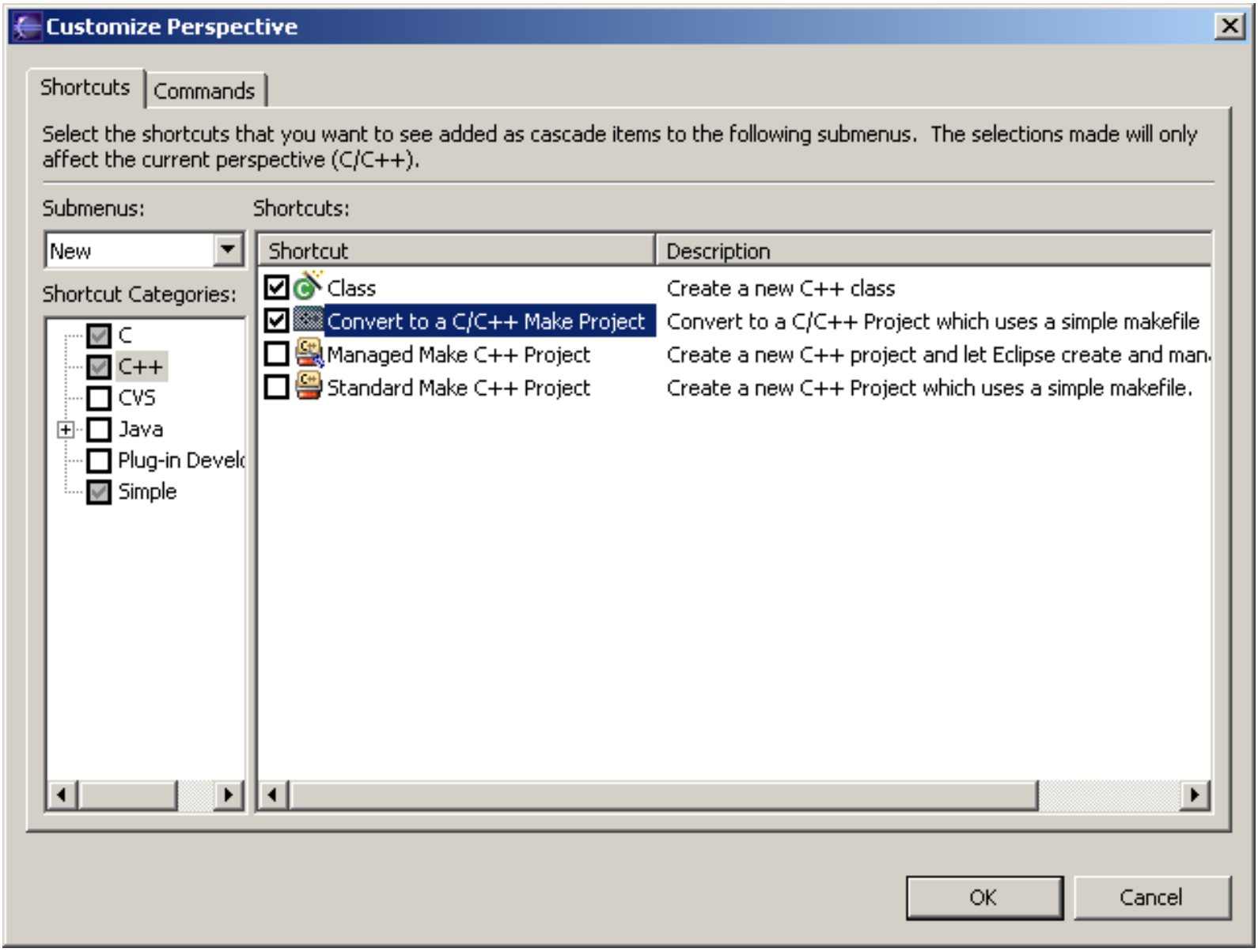
Adding Convert to a C/C++ Make Project to the New menu

If **Convert to a C/C++ Make Project** is not available in your menubar, you can add it by clicking **Window > Customize Perspective**

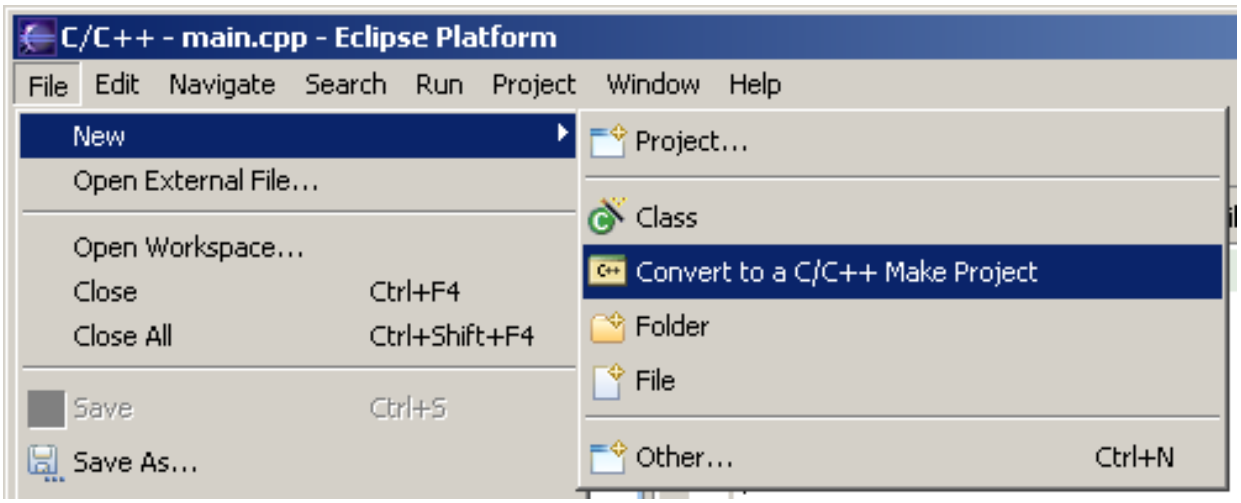


From the **Customize Perspecive Wizard** select **Convert to a C/C++ Make Project** and click **OK**.

Note: Ensure **New** is selected in the **Submenus:** list.



The **File > New > Convert to a C/C++ Make Project** option will now be available.

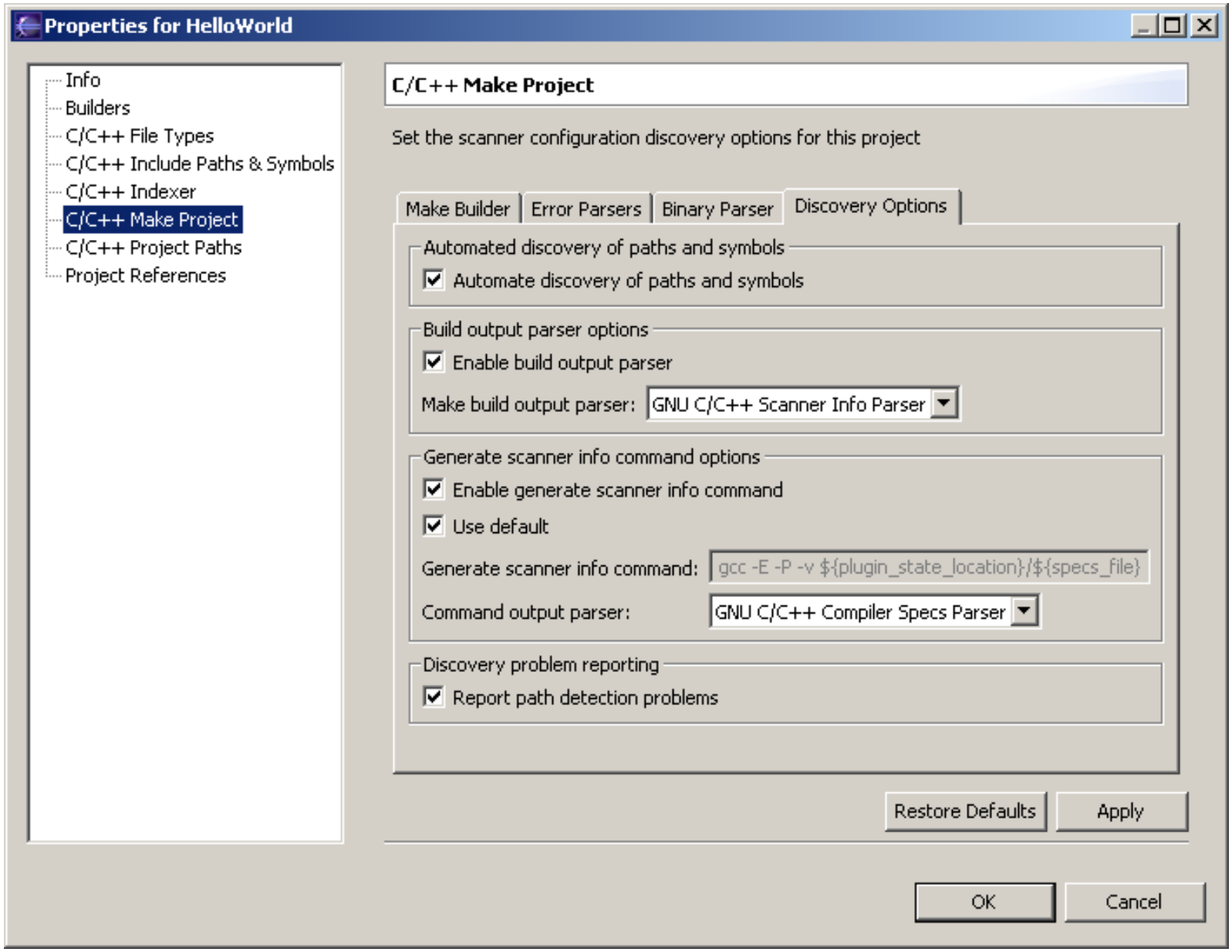


Related tasks
Converting CDT 1.x Projects

Set Discovery Options

For most standard make projects you will want to parse the output of the build to populate your paths and symbols tables.

To do so right click on your project and select **Properties > C/C++ Make Project > Discovery Options** and select the **Automate discovery of paths and symbols** checkbox.



Related reference
Converting CDT 1.x Projects
C/C++ Project Properties, Standard, Discovery Options

Writing code

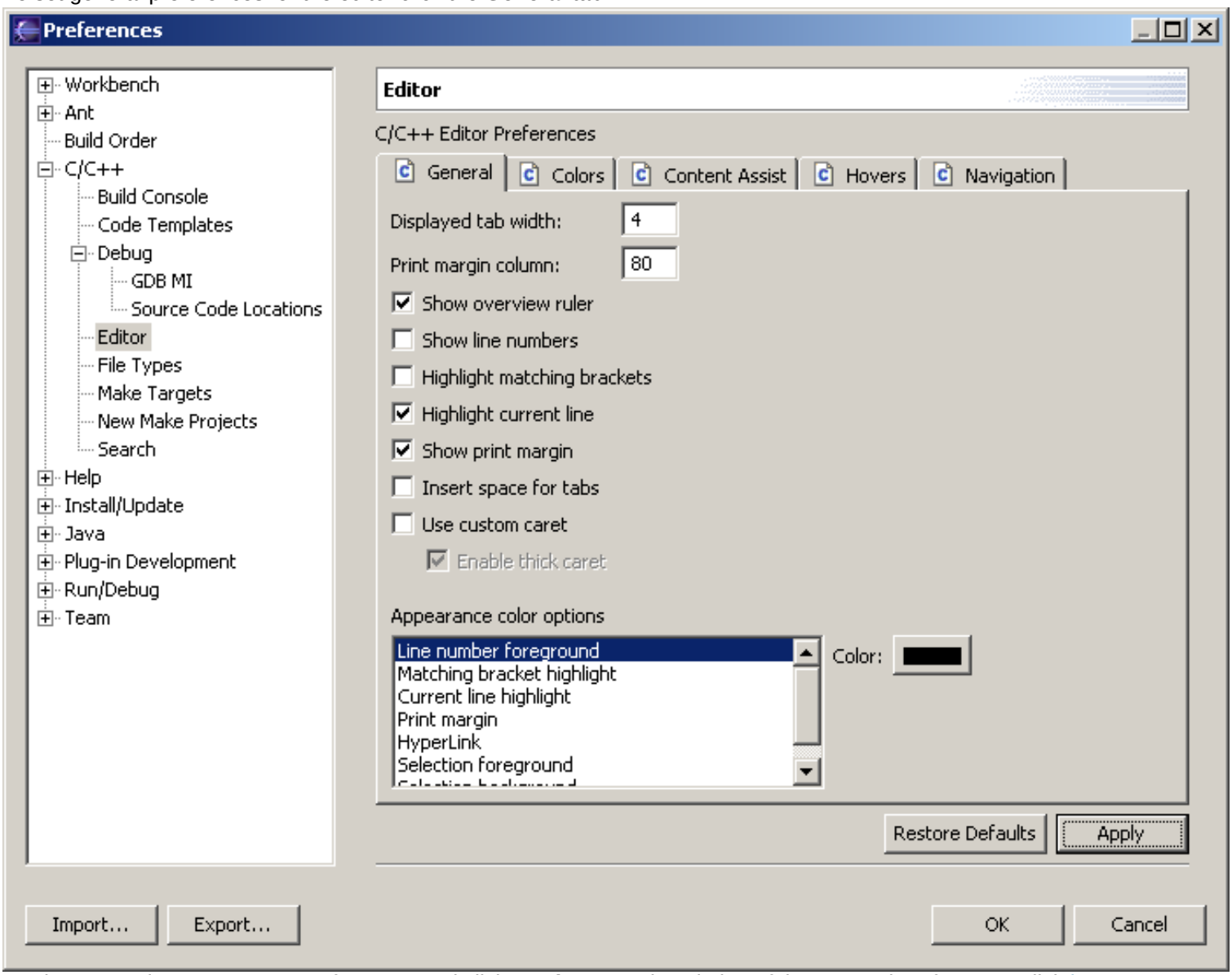
This sections explains how to work with the C/C++ editor.

Customizing the C/C++ editor

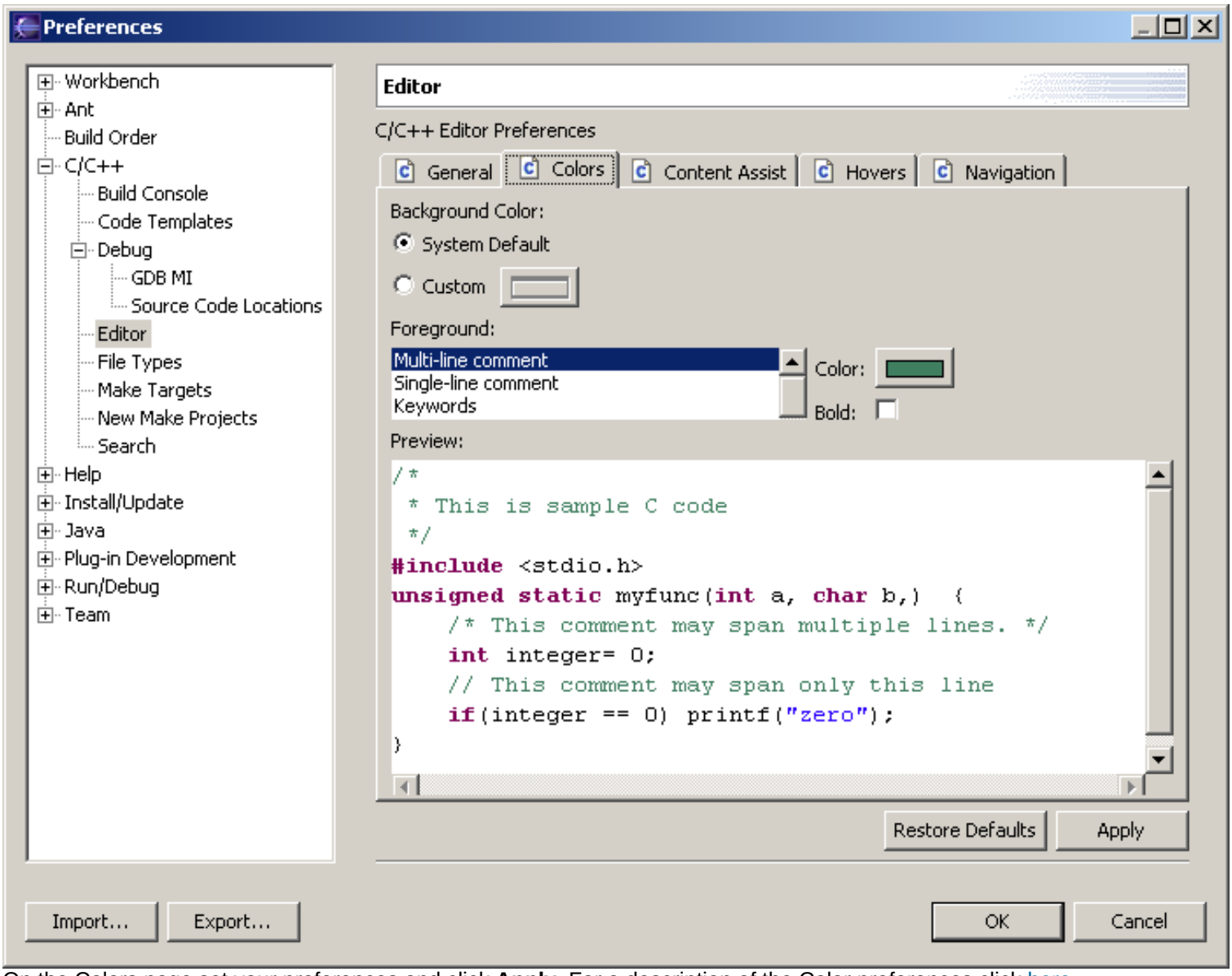
You can change many of the C/C++ editor preferences.

To customize the C/C++ editor preferences:

1. Click **Window > Preferences**.
2. Expand **C/C++**, and click **C/C++ Editor**.
3. To set general preferences for the editor click the **General** tab.



4. On the General Page set your preferences and click **Apply**. For a description of the General preferences click here.
5. To customize the color of the text in the editable area of the C/C++ editor, click the **Colors** tab.



6. On the Colors page set your preferences and click **Apply**. For a description of the Color preferences click here.
7. To customize Content Assist preferences, click the **Content Assist** tab. For more information, see Using Content Assist
8. Click **OK**.

Related concepts

Coding aids

Related tasks

Customizing the C/C++ editor

Related reference

C/C++ editor preferences

Commenting out code

You can comment out one or more lines of code. The leading characters `//` are added to the beginning of each line.

Tip: The characters `/* */` on lines that already are already commented out are not affected when you comment and uncomment code as described above.

To comment out code:

1. In the C/C++ editor, select the line(s) of code that you want to comment out. If no lines are selected comments will be added (or removed) at the current cursor positition.
2. Right-click and do one of the following:
    - To comment out the selected code, select **Comment**.
    - To remove the leading `//` characters from the selected line(s) of code, select **Uncomment**.

Tip: Instead of using the context menu (right click) you can quickly comment out by pressing CTRL+/ or remove comments by pressing CTRL+\.

Related concepts

Code entry

Related tasks

Customizing the C/C++ editor

Working with Content Assist

Related reference

C/C++ editor, code templates and search preferences

Working with Content Assist

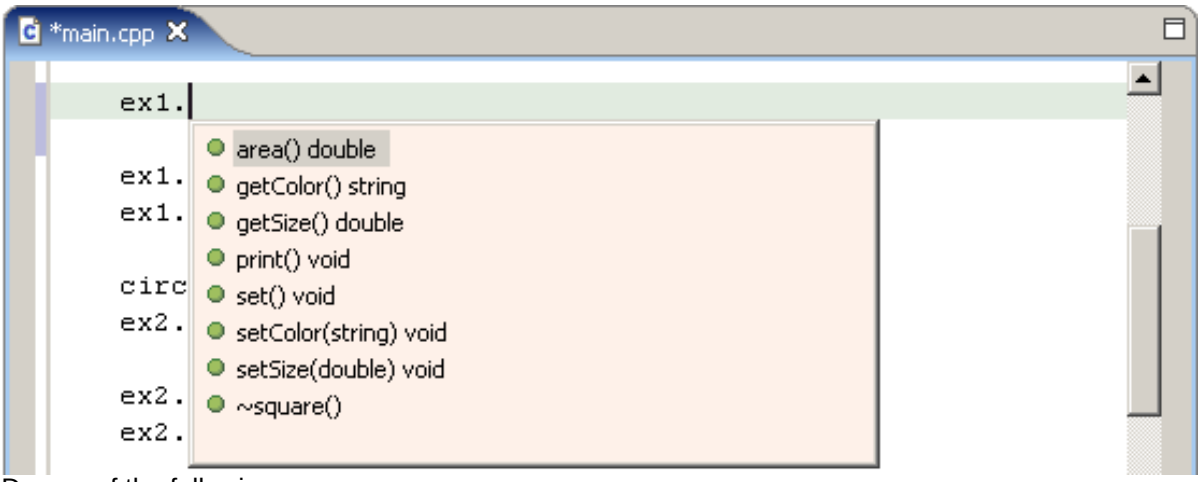
This section provides information on code entry aids.

Using Content Assist

Use Content Assist to insert C/C++ elements of your project, and code templates into your code. You can insert a code template into your source code rather than retyping commonly-used snippets of code.

To insert a code template or element:

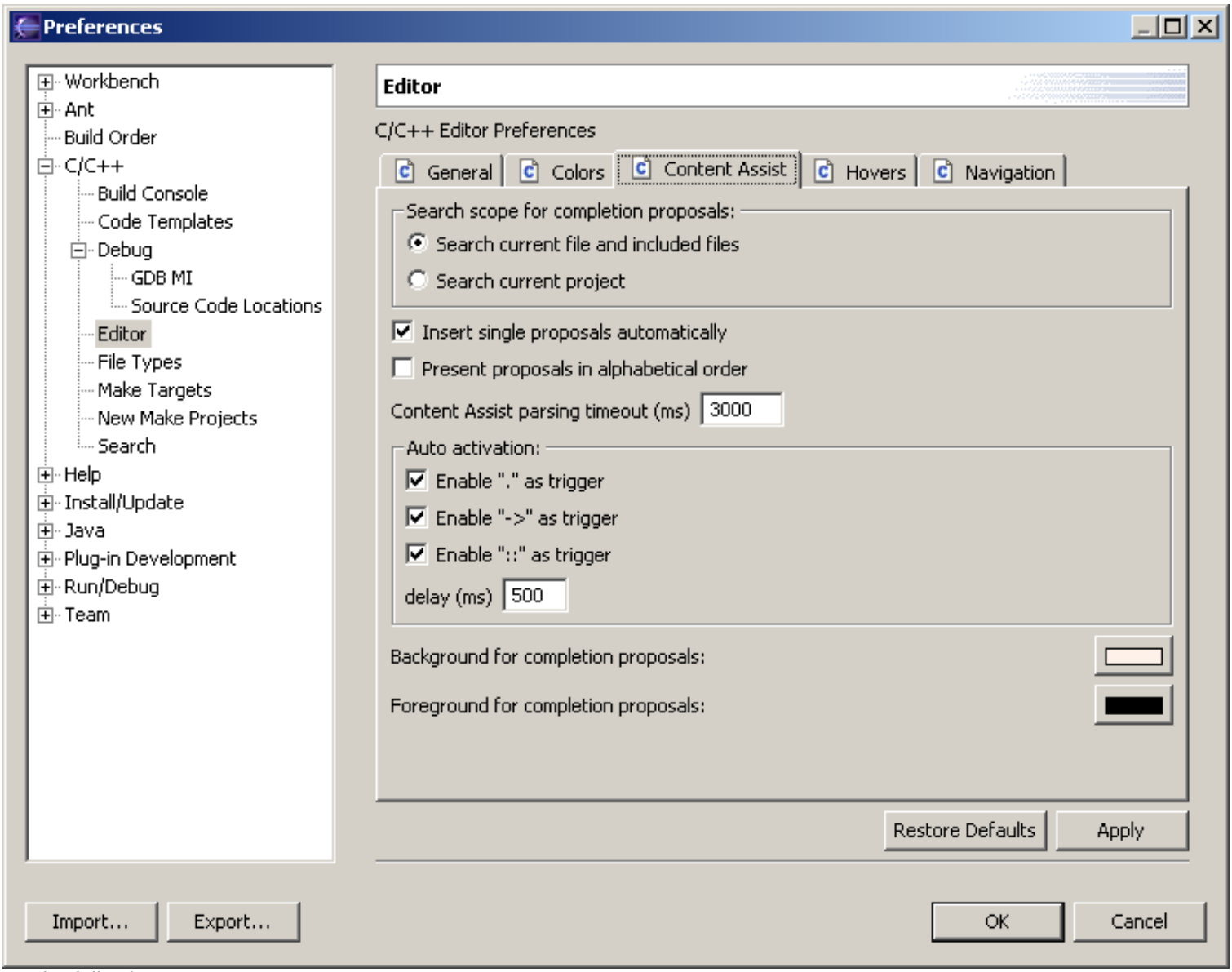
1. In the C/C++ editor, type at least the first letter of a code template or element then **Ctrl+Space**.
   A list displays the code templates followed by the elements that start with the letter combination you typed.



2. Do one of the following:
   - Continue typing. The list shortens. When there is only one item in the list, it is automatically inserted.
   - Double-click an item in the list to insert it into your code.
   - Press **Esc** to close the Content Assist Window.

To set Content Assist preferences:

1. Click **Window > Preferences**.
2. Expand **C/C++**, and click **C/C++ Editor**.
3. Click the **Content Assist** tab.



4. Do the following:
   - To change the scope from which Content Assist retrieves its proposals, select either **Search current file and included files** or **Search current project** by selecting the appropriate the radio button.
   - To insert an element when you open Content Assist and it is the only item in the list, select the **Insert single proposals automatically** check box.
   - To automatically insert a proposal, if it is the only one found when Content Assist is invoked, select the **Enable auto activation** checkbox.
   - To display proposals in alphabetical order, rather than by relevance, select the **Present proposals in alphabetical order** checkbox.
   - To change the amount of time Content Assist is permitted to parse proposals enter the value for **Content Assist parsing timeout** in the text box area.
   - Enable Auto activation of content assist for ".", "->" or "::" triggers by selecting the appropriate checkboxes.
   - To change the delay before Content Assist is automatically invoked for the triggers (shown above), enter the new delay in the **Auto activation delay** text box area.
   - To change the background color of the Content Assist dialog box, click the color palette button.
   - To change the foreground color of the Content Assist dialog box, click the color palette button.
5. Click **OK**.

Related concepts

Content Assist

Related tasks

Creating and editing code templates

Importing and exporting code templates

Related reference

Content Assist page, Preferences window

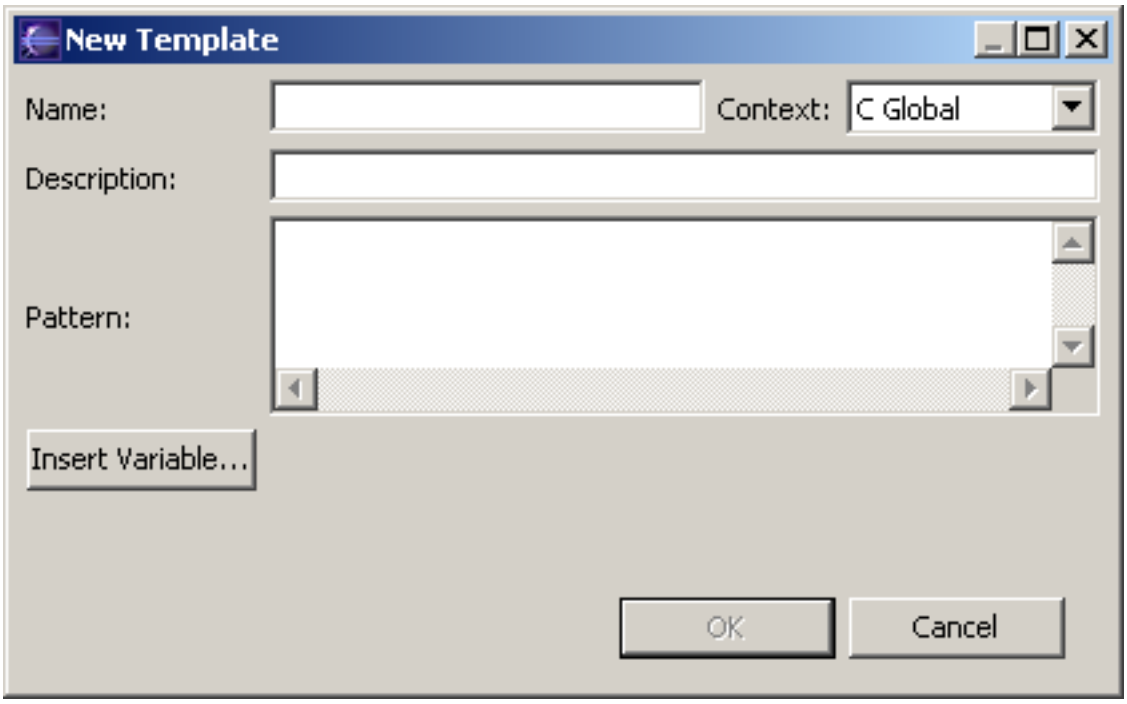
Code Templates page

Creating and editing code templates

Content Assist uses code templates enable you to use commonly used code snippets quickly.

To create a code template:

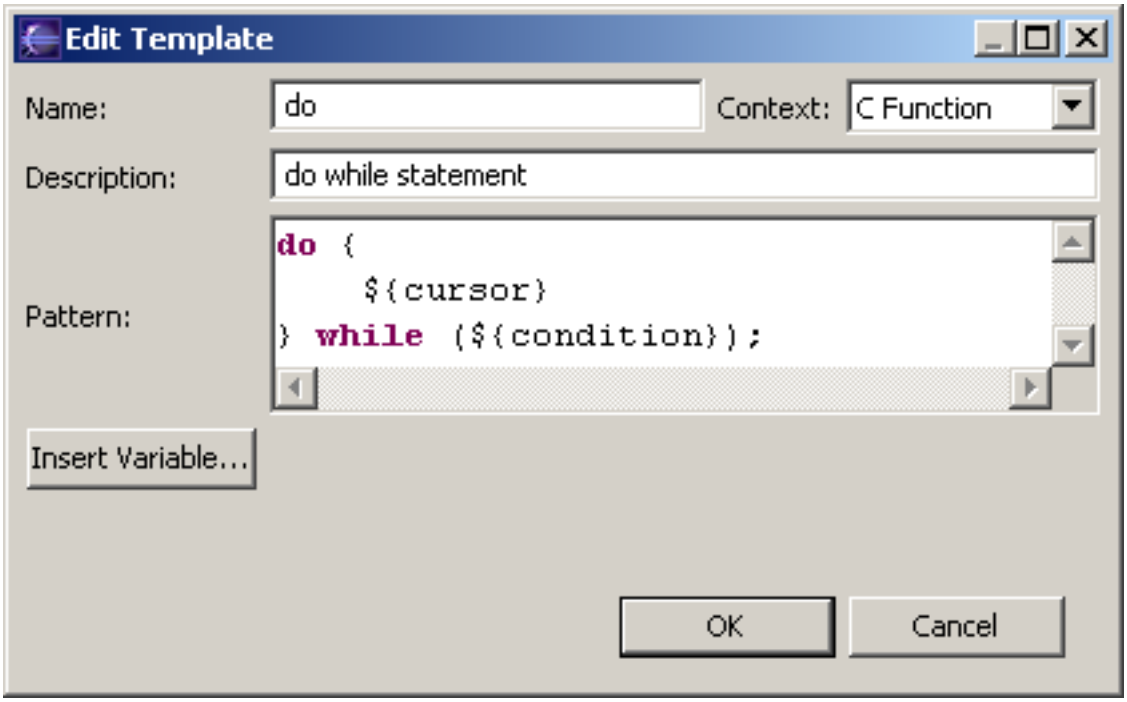
1. Click **Window > Preferences**.
2. Expand **C/C++**, and click **Code Templates**.
3. Click **New**.



4. Do the following: (if required)
   - In the **Name** field, enter a template name.
   - Select **Structure**, **Global** or **Function** for either **C** or **C++** from the Context drop down list.
   - In the **Description** field, enter a description for your new code template.
   - In the **Pattern** field, enter the code for your template.
   - Click **Insert Variable** to add a variable from the list to the code you have entered in the Pattern box.
5. Click **OK**.
   The new code template is now displayed in the list.

To edit a code template:

1. Click **Window > Preferences**.
2. Expand **C/C++**, and click **Code Templates**.
3. Click **Edit**. The New Template dialog box opens.



4. Do the following:
   - In the **Name** field, change the template name to create a new template based on the current template.
   - Select **Structure**, **Global** or **Function** for either **C** or **C++** from the Context drop down list to select where the template will appear.
   - In the **Description** field, change the description of the code template to reflect your changes.
   - In the **Pattern** field, edit the code.
   - Click **Insert Variable** to add a variable from the list to the code you have edited in the Pattern box.
5. Click **OK**.

Related concepts

Content Assist

Related tasks

Using Content Assist

Importing and exporting code templates

Related reference

Code Templates page, Preferences window

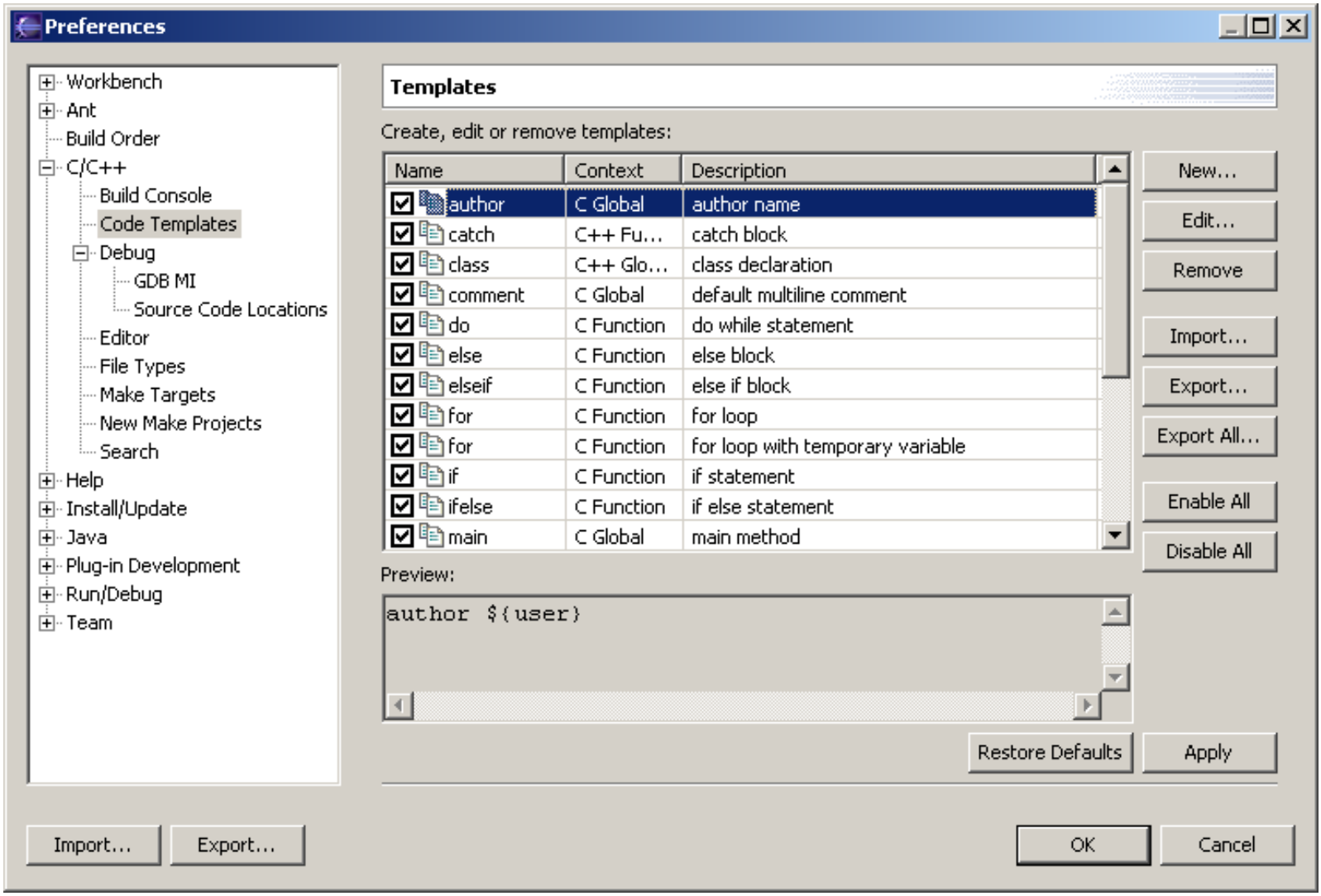
Code Templates page

Importing and exporting code templates

You can import and export code templates.

Note: A code template must be an .xml file formatted as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<templates>
<template name="author" description="author name" context="C" enabled="true">author ${user}</template>
</templates>
```



To import a code template

1. Click **Window > Preferences**.
2. Expand **C/C++**, and click **Code Templates**.
3. Click **Import**.
4. Select the template file that you want to import.
5. Click **OK**.
   The code template list is updated to include the template that you imported.

To export a code template

1. Click **Window > Preferences**.
2. Expand **C/C++**, and click **Code Templates**.

3. Select the templates that you want to export and click the **Export...** or **Export All...** button. The Exporting x dialog opens where x is the number of code templates that you are exporting.
4. In **File name** box, type the path where you want your code templates file to be saved.
5. Click **Save**.
   The templates.xml file containing the code templates you exported is saved in your file system.

Related concepts

Content Assist

Related tasks

Using Content Assist

Creating and editing code templates

Related reference

Code Templates page, Preferences window

Code Templates page

Shifting lines of code to the right or left

You can shift lines of code to the left or right in the C/C++ editor. You can change the tab width in the C/C++ editor preferences window. For more information, see Customizing the C/C++ editor.

To shift lines of code to the right or left:

1. In the C/C++ editor, select the full length of the lines that you want to shift.
2. Do one of the following:
    - To move the text to the right, press **Tab**.
    - To move the text to the right, click **Edit > Shift Right**.
    - To move the text to the left, press **Shift+Tab**.
    - To move the text to the left, click **Edit > Shift Left**.

Related concepts

Code entry

Related tasks

Customizing the C/C++ editor

Related reference

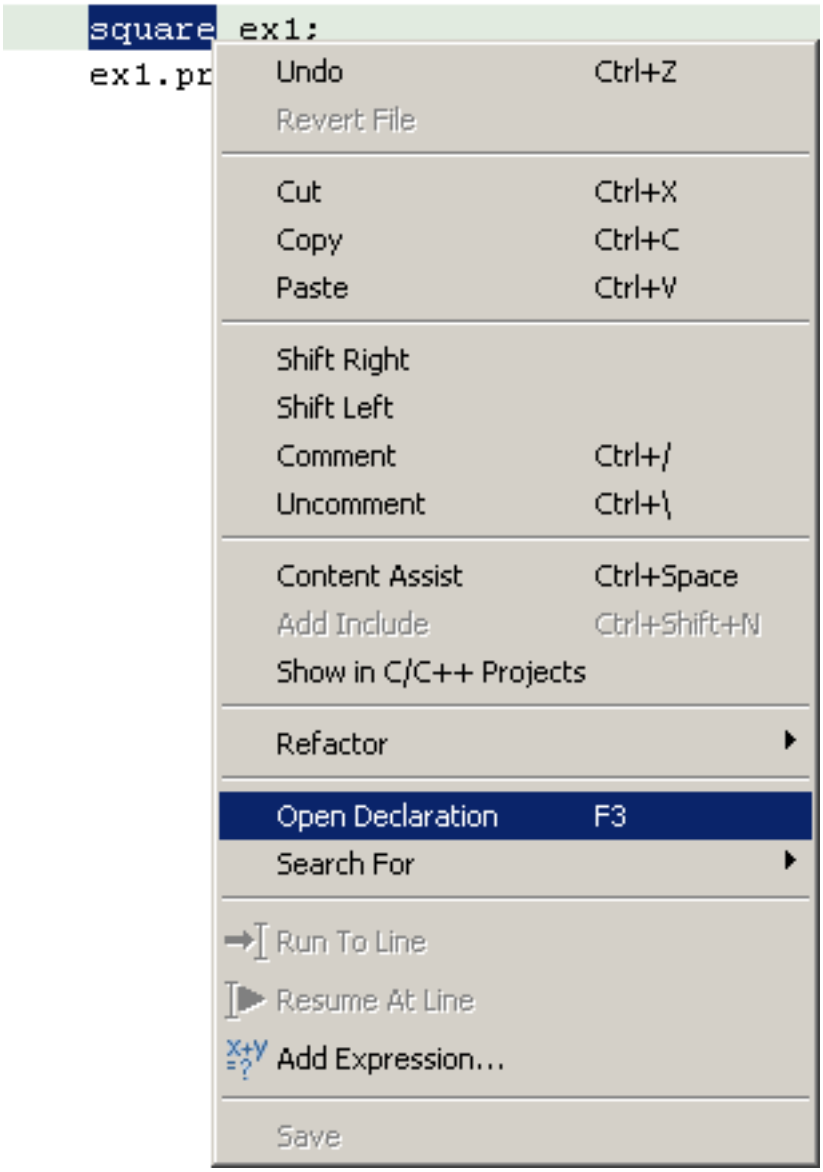
C/C++ editor, code templates and search preferences

Navigating to C/C++ declaration

The Open Declaration feature lets you navigate to the declaration that matches a selected element in the C/C++ editor. It is recommended that you look for element declarations on successfully compiled programs.

To navigate to C/C++ declaration:

1. In the C/C++ editor, select an object.
2. Right-click the selected element, select **Open Declaration**.



For more information, see:

- **Workbench User Guide > Tasks> Navigating and finding resources**

Related concepts

Open Declarations

CDT Projects

C/C++ search

Related tasks

Searching for C/C++ elements

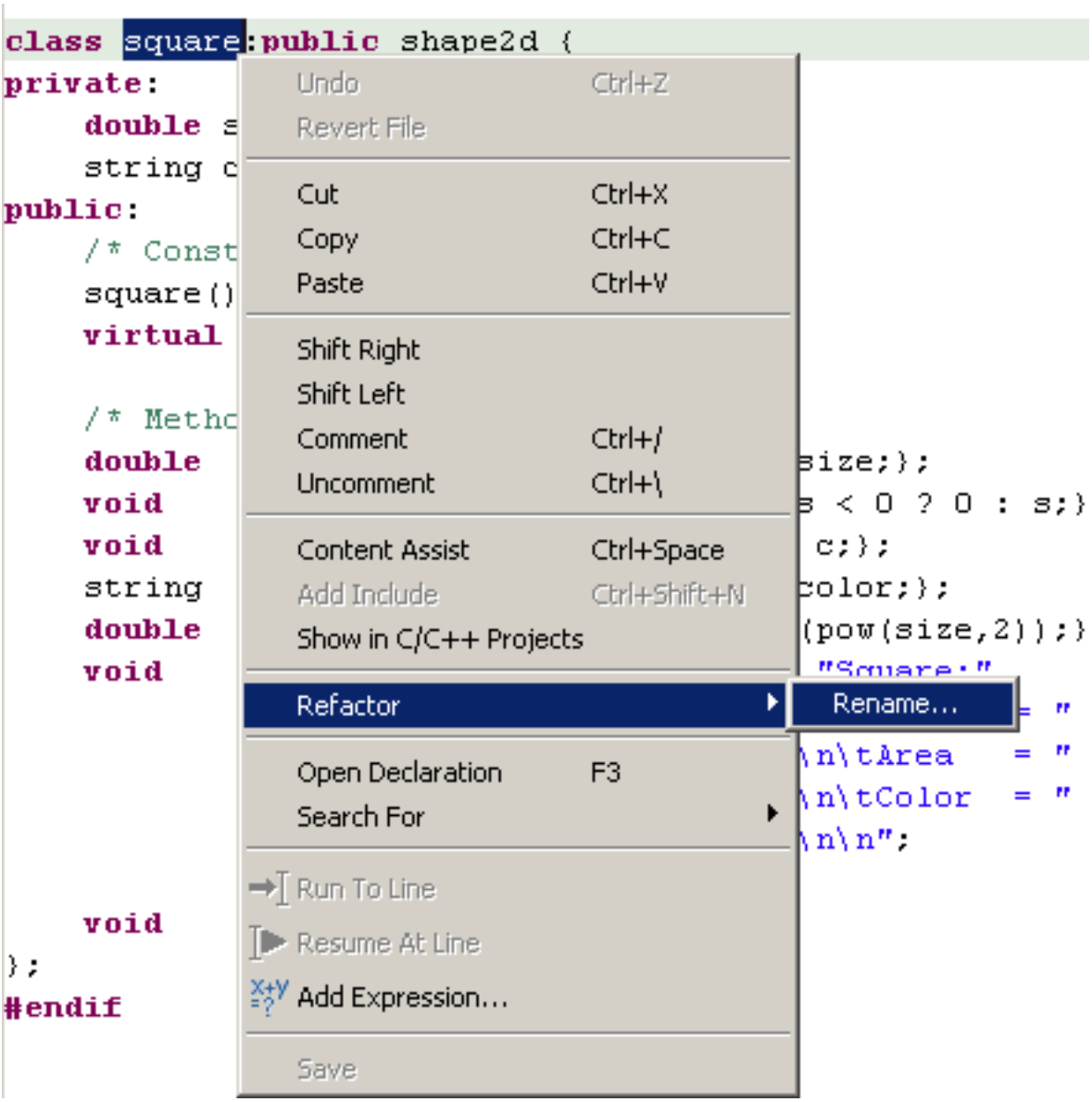
Related reference

C/C++ search page, Search dialog box

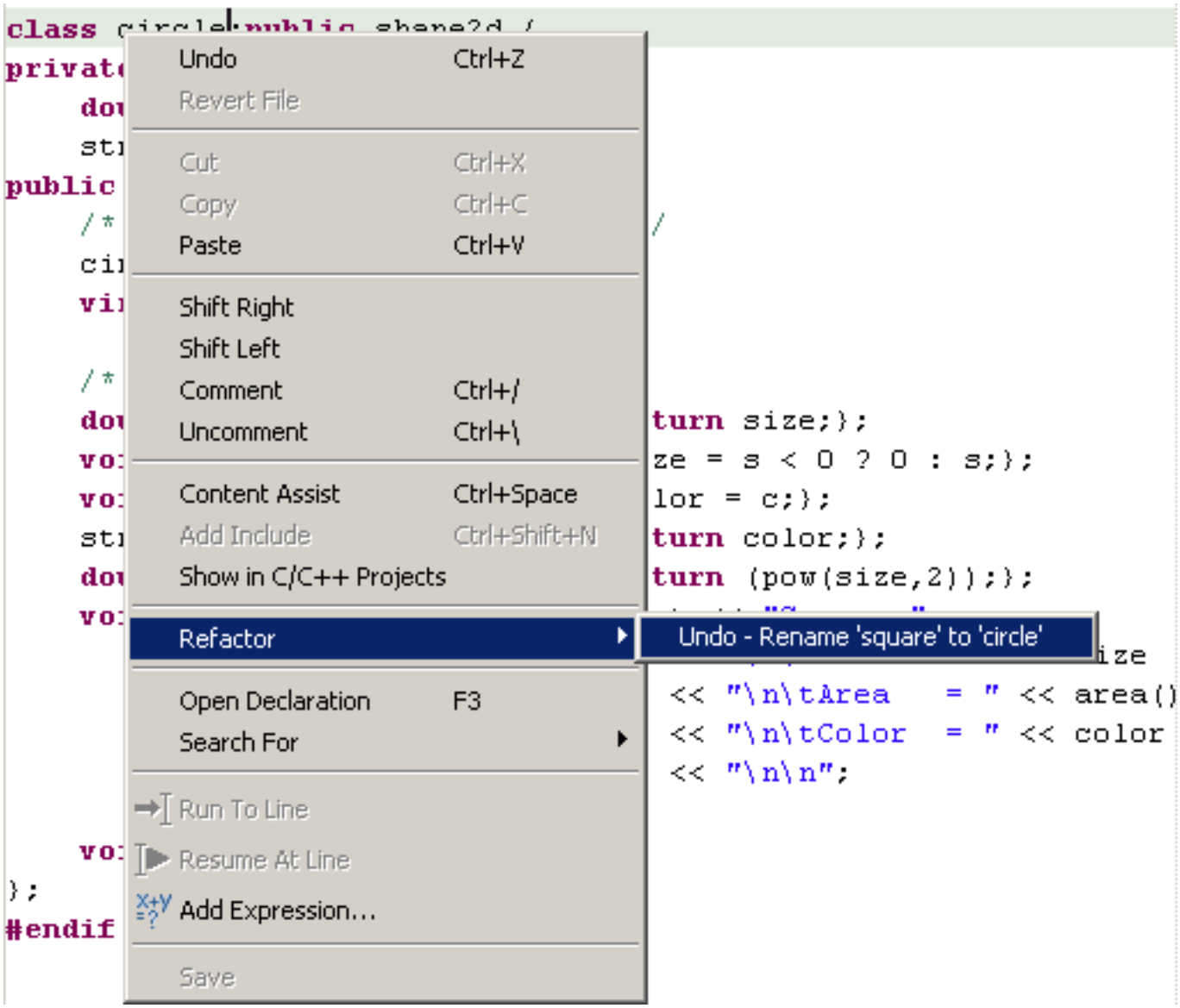
Refactoring

Use the C/C++ Projects, Outline, or the Editor view **Refactor > Rename** context menu to refactor class & type names, methods, function & member names.

To refactor an object select the object, right click and select **Refactor > Rename...**



The refactoring engine will rename all instances of the object in all referenced files. You can Undo refactoring by right clicking a second time and selecting **Refactor > Undo**



Related concepts

Open Declarations
CDT Projects
C/C++ search

Related tasks

Searching for C/C++ elements

Related reference

C/C++ search page, Search dialog box

Building projects

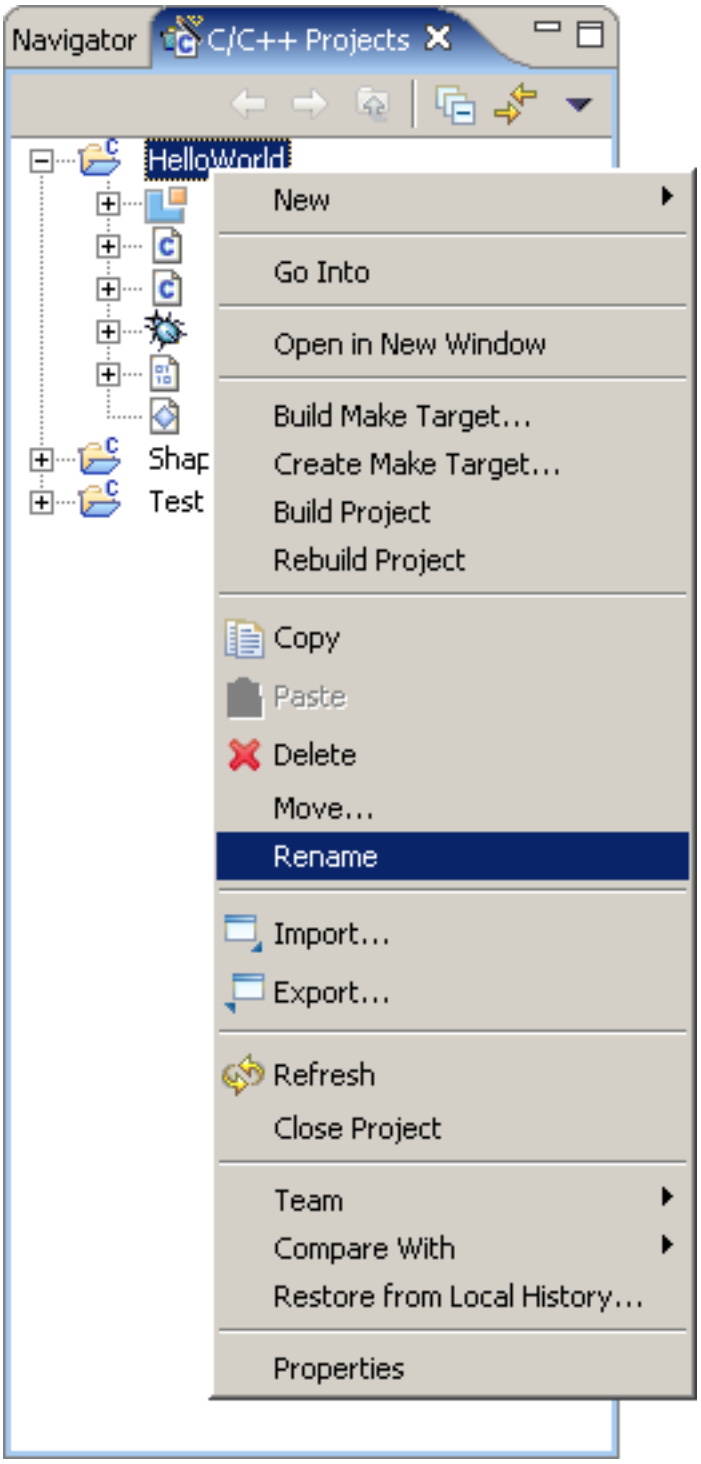
This sections explains how to build your project and manage compile errors.

Renaming a project

You can rename a project, and have all references changed using the refactoring engine.

To rename a project:

1. In the C/C++ Projects view, right-click a project, and select **Rename**.



2. Type a new name.
3. Press **Enter**.

Note: Renaming a project causes it to be re-indexed. This can take a significant amount of time for very large projects.

Related concepts

CDT Projects

Project file views

Related tasks

Working with C/C++ project files

Related reference

Project properties

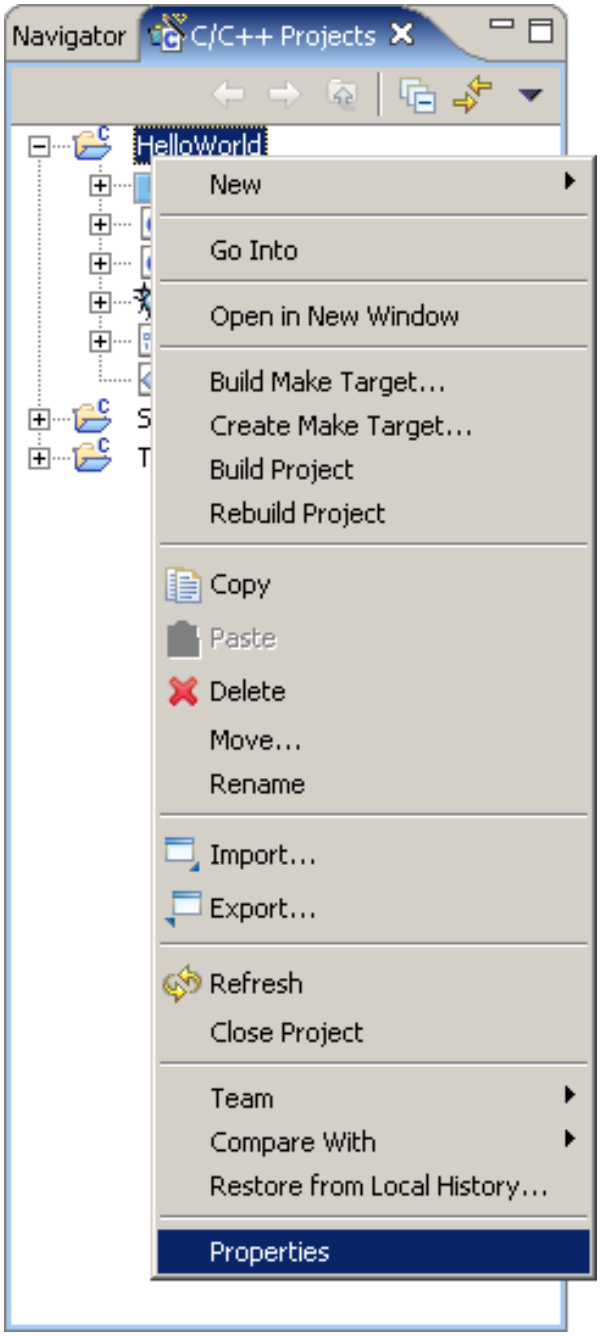
Selecting referenced projects

The Referenced C/C++ Projects list on the Projects References page for a given project, displays every project in your workspace in the build order you specify. For more information, see Setting build order.

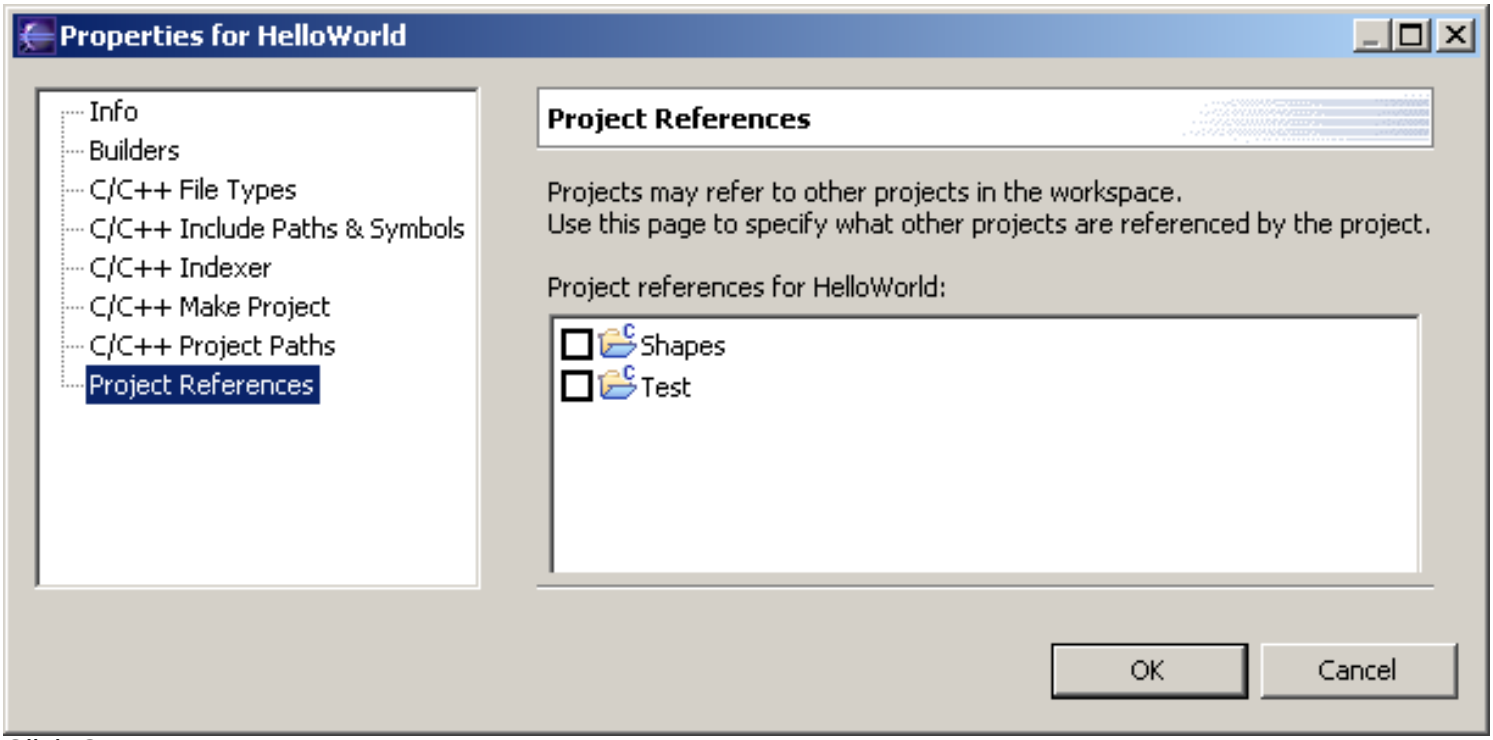
Projects selected in the list are built before the current project according to their dependencies. The least used projects are built first.

To select referenced projects:

1. In the C/C++ Projects view, right-click a project, and select **Properties**.



2. On the left, select **Project References** from the list.
3. In the Project references... list, select referenced projects..



4. Click **OK**.

Related concepts

CDT Projects

Project file views

Related tasks

Working with C/C++ project files

Related reference

Project properties

Defining build settings

The **Make Builder** page lets you:

- Configure how the CDT handles make errors.
- Change the default build command.
- Map the target passed to make when you select build or rebuild.

You can define the properties on a per project basis in the New Project wizard, in the C/C++ Projects view or in the Navigator view. You can also define project properties in the Preferences window for future standard make projects.

Before you begin

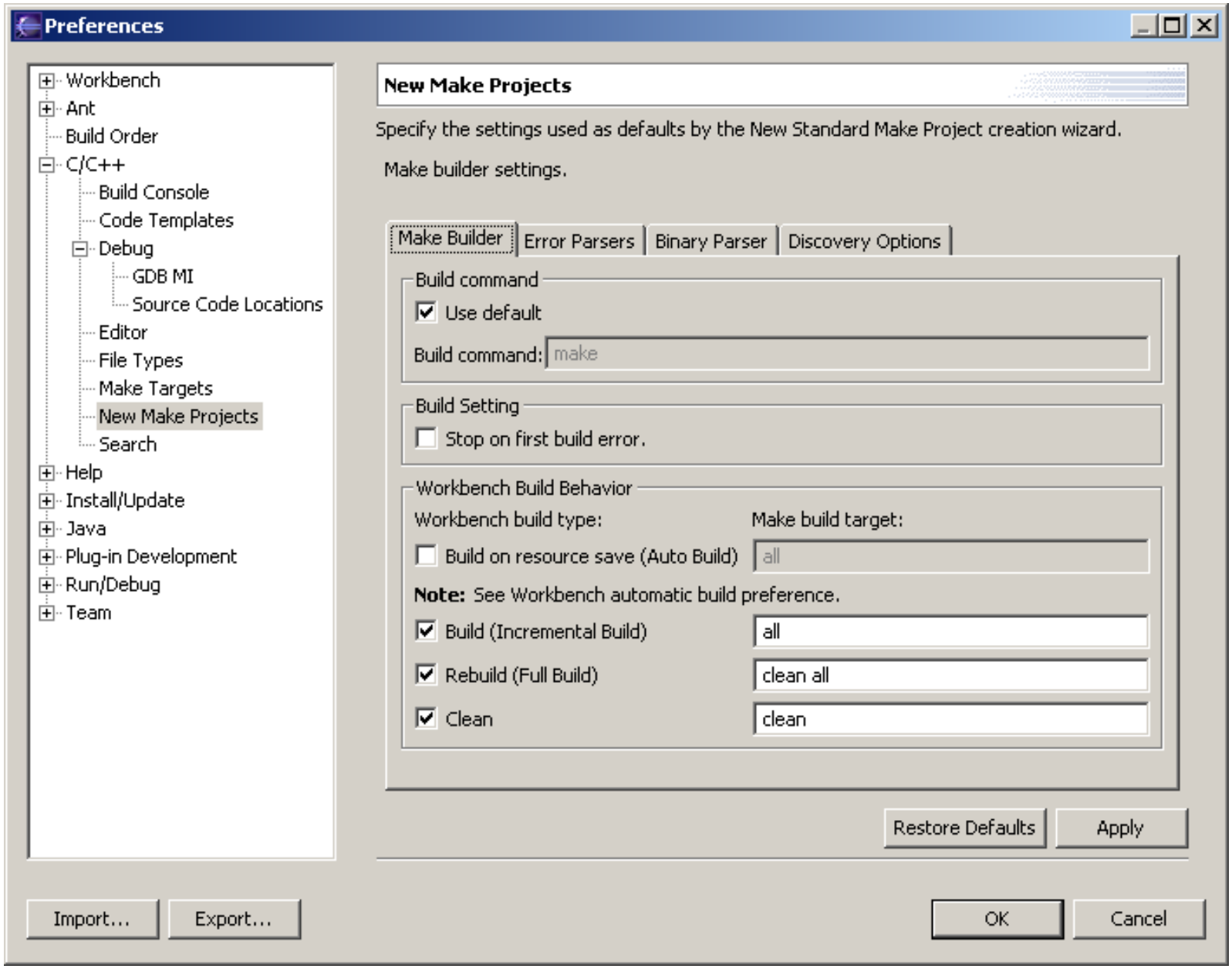
In order to be able to turn on or to turn off the feature that automatically performs an incremental build of your project every time a resource is saved for individual projects, you need to enable the Workbench **Build Automatically** preference. You can then disable this feature or change the associated make target for individual projects.

To enable build on resource save:

1. Click **Window > Preferences**.
2. To automatically perform an incremental build of your project every time a resource is saved, select **Workbench** from the list.
3. Select the **Perform build automatically on resource modification** check box.
   Note: this is the default setting.

   You can now enable or disable this feature on a per project basis.

To define build settings:



1. Do one of the following:
   - Click **Window > Preferences**. Expand **C/C++**, click **New Make Projects**.
   - In the C/C++ Projects view, right-click a standard make project, and select **Properties**. Select **C/C++ Make Project** from the list.
2. Click the **Make Builder** tab.
3. Do one of the following:
   - To stop the build when an error is encountered, select **Stop on first build error**.
4. Select one of the following build command settings:
   - To use the default make command, select the **Use Default** check box.
   - To use a build utility other than the default make command **Build Command** box, clear the **Use Default** check box .
5. In the Workbench Build Behavior box, do the following:
   - To build your project when resources are saved and change the default make build target, select the **Build on resource save (Auto Build)** check box. Enter a new build target in the **Make build target** box.
   - To change the build default make build target, select the **Build (Incremental Build)** check box. Enter a new build target in the **Make build target** box.
   - To change the rebuild default make build target, select the **Rebuild (Full Build)** check box. Enter a new build

target in the **Make build target** box.

6. Click the **Finish**.

Related concepts

CDT Projects

Project file views

Related tasks

Working with C/C++ project files

Related reference

Make Builder page, C/C++ Properties window

Filtering errors

The Error Parsers page of the Properties window lists a set of filters that detect error patterns in the build output log.

If an error or warning is detected, an icon appears on the file in the Tasks view. If the file is not found, the icon appears on the project.

You can define the properties on a per project basis in the New Project wizard, in the C/C++ Projects view or in the Navigator view. You can also define project properties in the Preferences window for future standard make projects.

To select error parsers:

1. Do one of the following:
    - To set properties for future Standard Make projects, click **Window > Preferences** . Expand**C/C++**, and click **New Make Projects**.
    - In the C/C++ Projects view, right-click a standard make project, and select **Properties**. Select **C/C++ Make Project** from the list.
2. Click the **Error Parsers** tab.
3. In the **Error parsers** list, select error parsers.
4. Click **OK**.

Related concepts

CDT Projects

Project file views

Related reference

Error Parsers, C/C++ Properties window

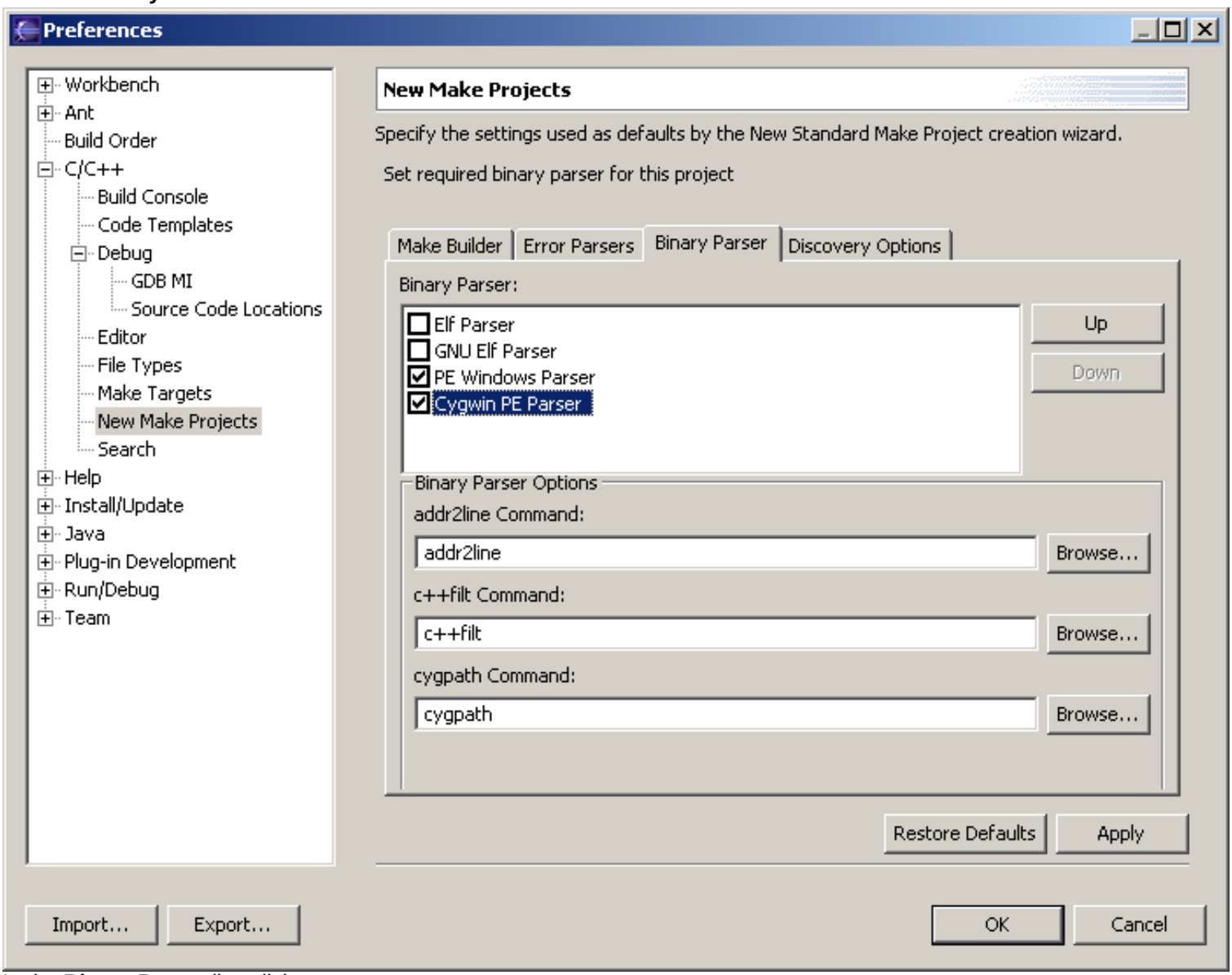
Selecting a binary parser

Selecting the correct binary parser is important to ensure the accuracy of the C/C++ Projects view and the ability to successfully run and debug your programs. Windows users should select the PE Windows Parser. UNIX and Linux users should select the Elf Parser. When you select the correct parser for your development environment and build your project, you can view the components of the .o file in the C/C++ Projects view and view the contents of the .o file in the C/C++ editor. You can also easily browse for the executable when defining run/debug configurations.

You can define the properties on a per project basis from the New Project wizard, in the C/C++ Projects view or in the Navigator view. You can also define project properties in the Preferences window for future standard make projects.

To select a binary parser:

1. To set properties for future Standard Make projects, click **Window > Preferences** . Expand **C/C++,** click **New Make Projects**.
2. Click the **Binary Parser** tab.



3. In the **Binary Parser** list, click:
   - **Elf Parser**, if you are a Solaris, UNIX, or Linux user.
   - **PE Windows Parser**, if you are a Windows user.
   - **Cygwin PE Parser**, if you are using Cygwin.
4. Click **OK**.

Related concepts
CDT Projects
Project file views

Related tasks
Working with C/C++ project files

Related reference
Binary Parser, C/C++ Properties window

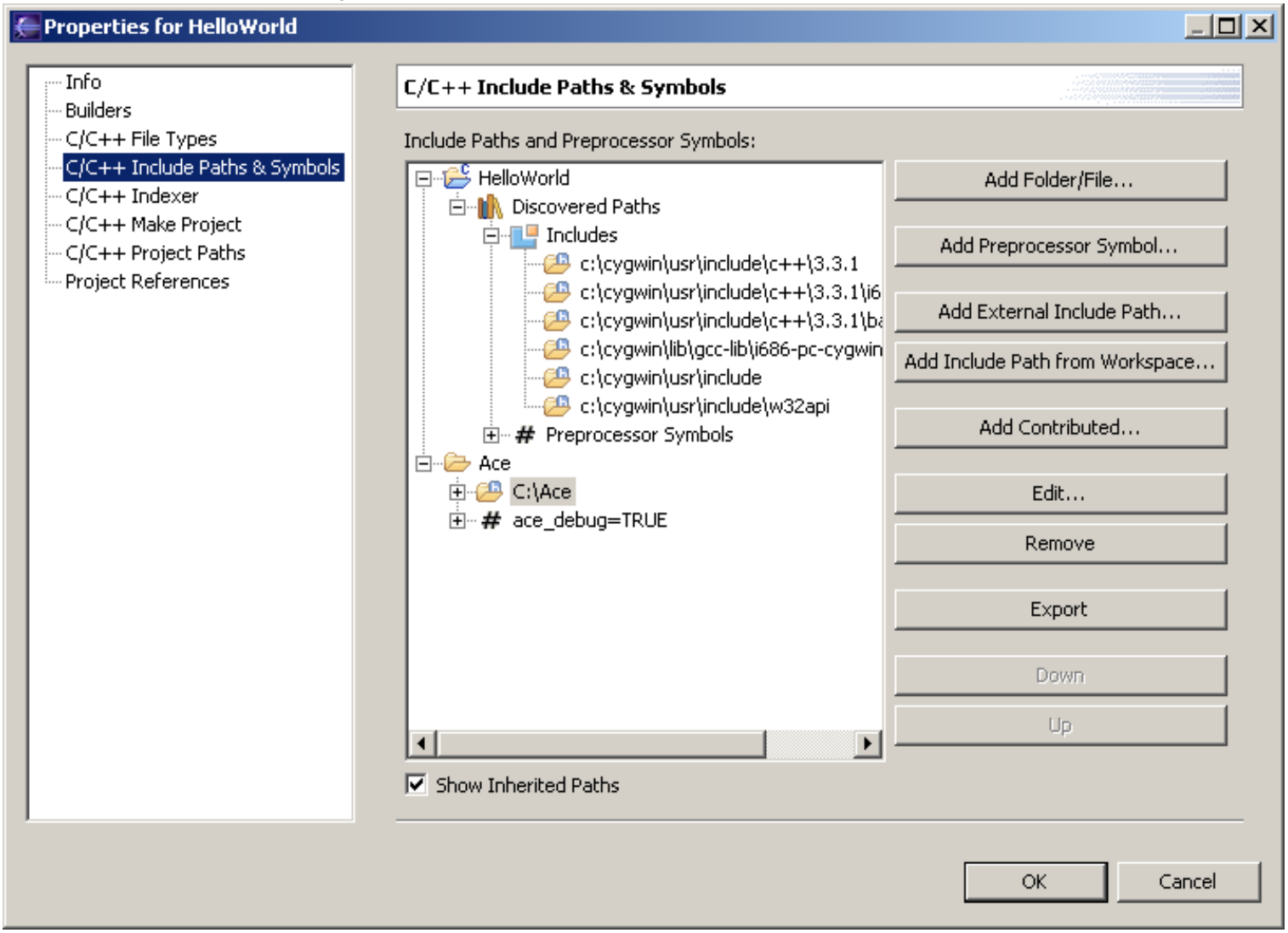
Adding Include paths and symbols

For Standard Make projects you can define include paths and preprocessor symbols for the parser. This enables the parser to understand the contents of the C/C++ source code so that you may more effectively use the search and code completion features.

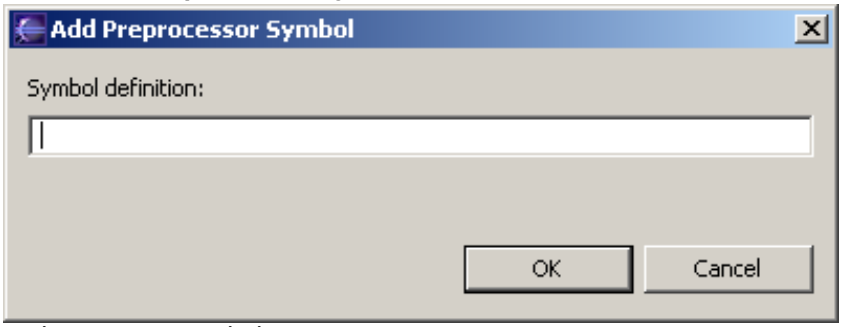
If Autodiscovery is enabled once a build has completed any discovered paths and symbols will be displayed in the Discoverd Paths section. You can also define the properties on a per project basis in the C/C++ Projects or Navigator views.

To add include paths and symbols:

1. To set properties for your project right click your standard make project and select **Properties**.
2. Click **C/C++ Include Paths and Symbols**.

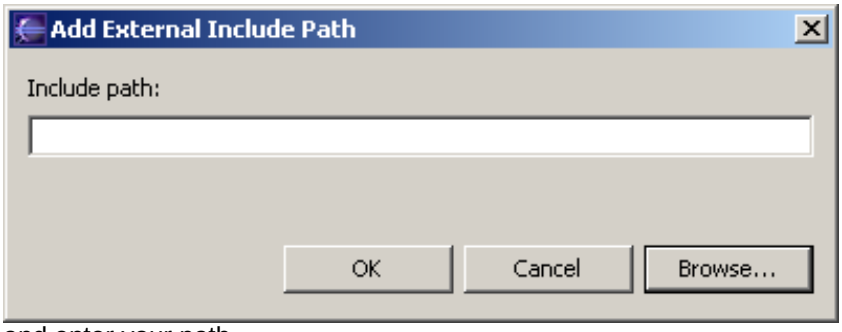


3. Select **Add Preprocessor Symbol...**



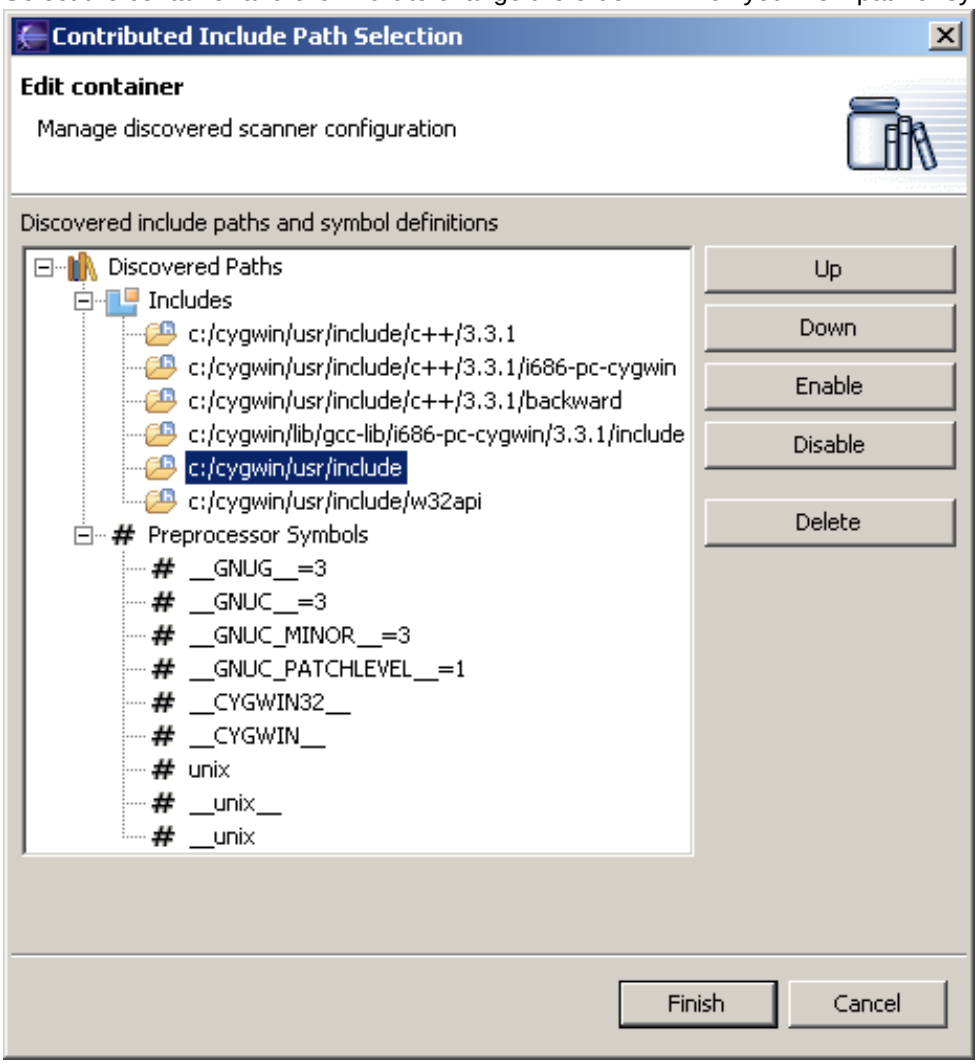
and enter your symbol.

4. Select **Add External Include Path...**



and enter your path.

5. Select the container and click **Edit** to change the order in which your new path or symbol is used.



6. Select the new object and click **Up** or **Down** to move it higher or lower in the order, or you can disable it by clicking **Disable**.
7. Click **Finish** to close the Edit Container window.
8. Click **OK** to close the Preferences window.

Related concepts

CDT Projects

Project file views

Related tasks

Working with C/C++ project files

Selecting a deployment platform

For managed make projects, you need to select the platform on which you plan to deploy your program. You also need to select a release or debug configuration for your project.

To select a deployment platform:

1. In the C/C++ Projects view, right-click a project, and select **Properties**.
2. Select **C/C++ Build** from the list.
3. To create a configuration with optimization flags enabled and debug symbols disabled, in the Configurations list, select **Release**.
4. To create a configuration with optimization flags disabled and debug symbols enabled, in the Configurations list, select **Debug**.

Related concepts

CDT Projects

Project file views

Related tasks

Working with C/C++ project files

Related reference

Target platform, C/C++ Properties window

Setting build order

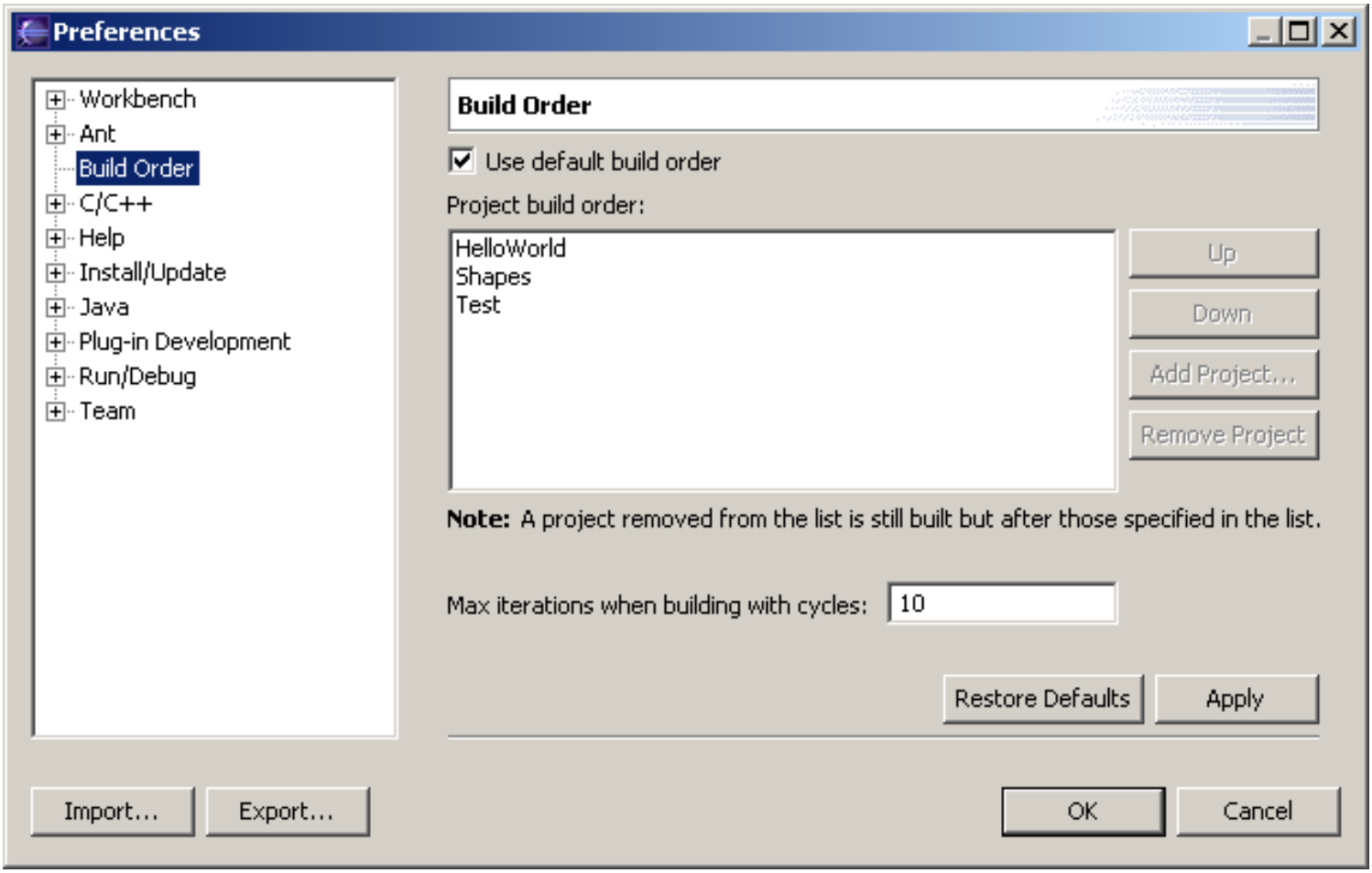
You can specify the order in which projects are built. Referenced projects are built first.

When you set the build order, the CDT does not rebuild projects that depend on a project. You must rebuild all projects to ensure changes are propagated.

For more information on build order, see **Workbench User Guide > Reference > Preference > Build Order**.

To set the project build order:

1. Click **Window > Preferences**.
2. Select **Build Order** from the list.



3. Clear the **Use default build order** checkbox.
4. Select a project in the list.
5. Do one of the following:
   - Click **Up** to move the project up the list.
   - Click **Down** to move the project down the list.
6. To add projects to the build path, click **Add Project**.
7. Select the projects to add to the build order list.
8. Click **OK**.
9. To remove a project from the Project build order list, click **Remove Project**.

   When building or rebuilding all projects, the projects that have been removed from the build order are built last.

10. Click **Apply**.

Related concepts

Build overview

Related tasks

Defining build settings

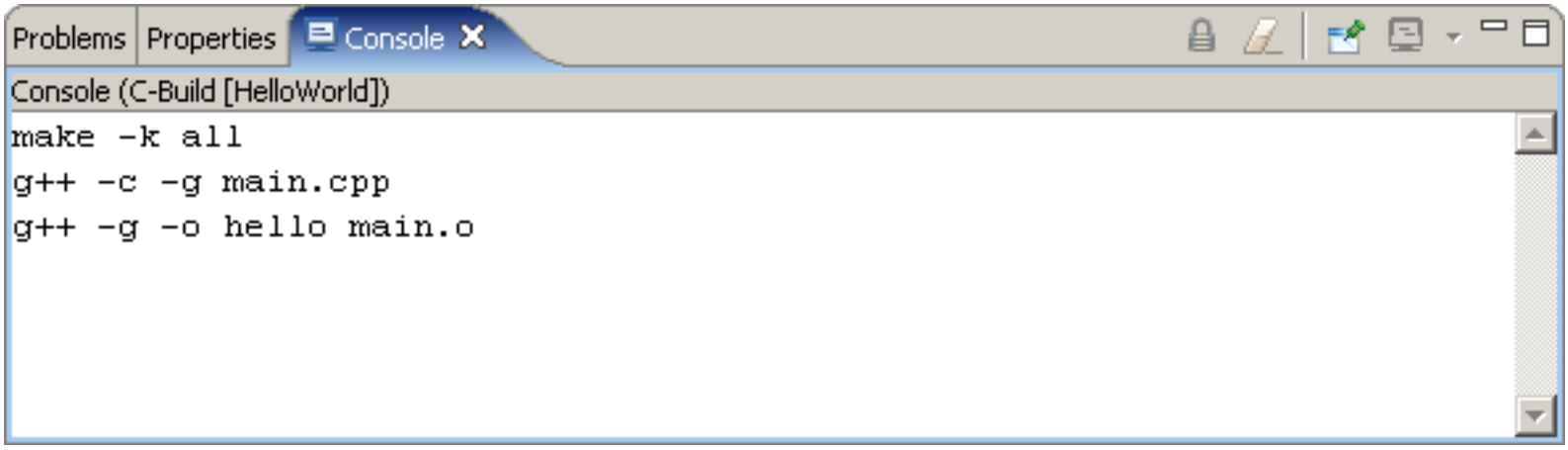
Building

Related reference

Make Builder page, C/C++ Properties window

Building Manually

Manual builds let you choose the scope of a build, as well as options for building, or rebuilding projects. You can view the output of the make utility in the console.



Incremental Builds
To incrementally build all open projects, you can select **Project > Build All** or type **CTRL+B**.

Build Individual Projects
To build individual projects click **Project > Build Project**.

Rebuild Projects
To rebuild a project right click on the project and select **Rebuild Project**.
Note: This will rebuild projects that this project references as well, but will not rebuild projects that references this one.

Build Automatically
This performs a Build All whenever any project file is saved, such as your `makefile`.
Tip: For C/C++ projects this feature should be turned off, if there is a checkmark beside **Build Automatically** it is on, to turn it off select **Build Automatically**.

Tip: If you get the error message:

```
Exec error:Launching failed
```

The error message means that the CDT cannot locate the build command, (usually `make`). Either your path is not configured correctly or you do not have `make` installed on your system.

Tip: The menubar item **Project > Build Working Set** submenu for C/C++ projects simply creates a link to the build all target as defined in your makefile, and is no different from an **Incremental Build**.

Related concepts
Build overview

Related tasks

Defining Build Settings

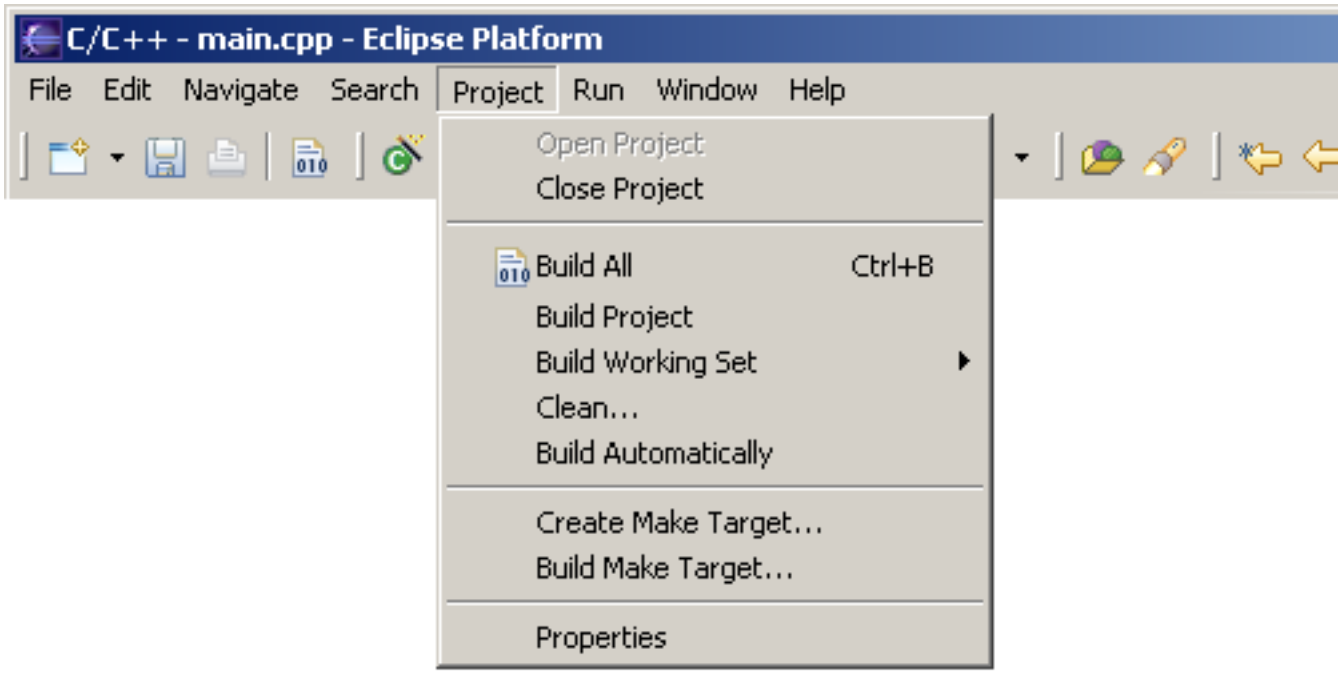
Building

Related reference

Make Builder page, C/C++ Properties window

Removing Build Automatically

The Eclipse workbench is configured to build projects automatically. However for C/C++ development you should turn this feature off, otherwise your entire project will be rebuilt whenever, for example, you save a change to your makefile or source files. Click **Project > Build Automatically** and ensure there is no checkmark beside the **Build Automatically** menu item.



Related concepts

Build overview

Related tasks

Building Manually

Defining Build Settings

Building

Related reference

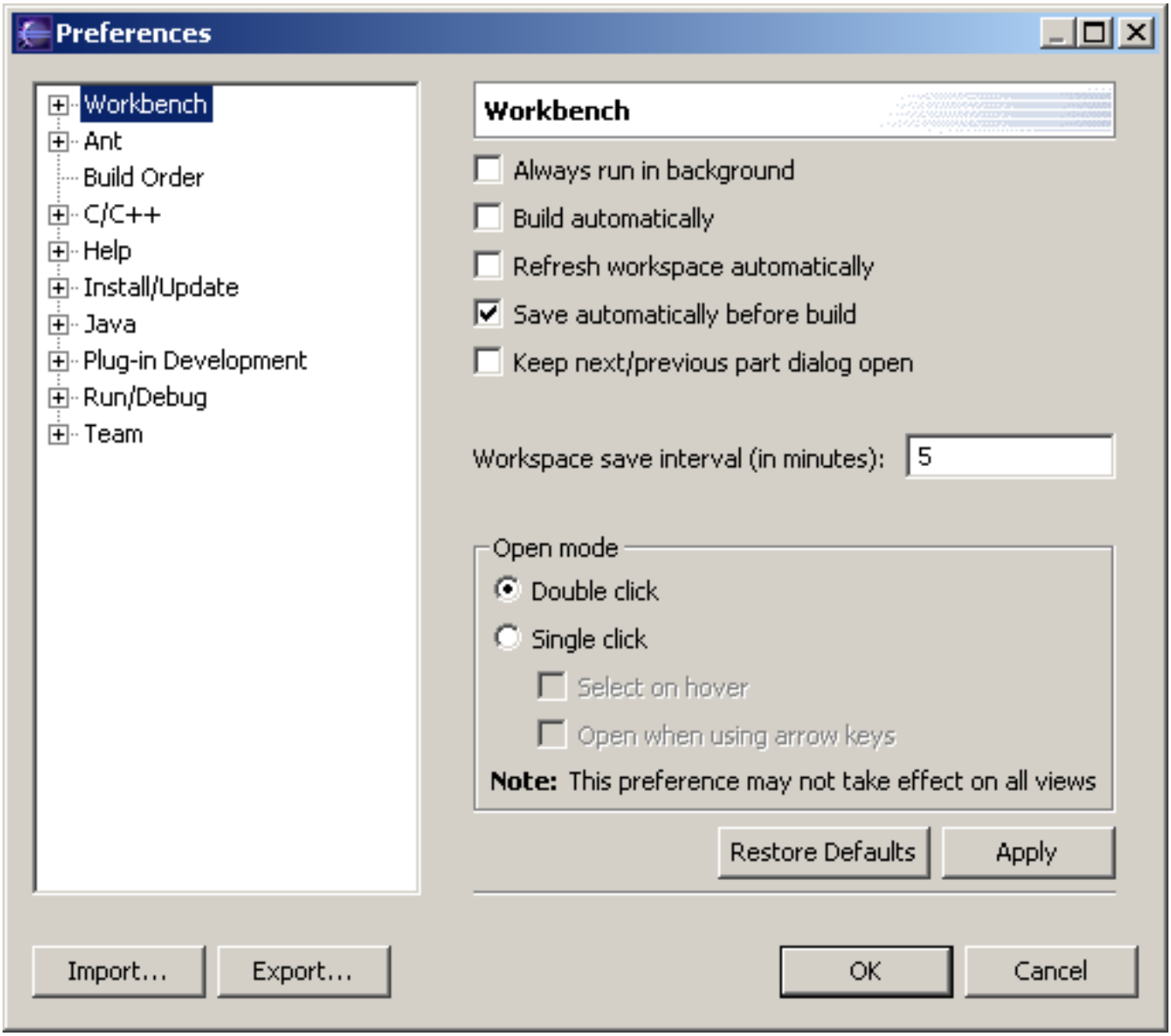
Make Builder page, C/C++ Properties window

Autosaving on a build

You can let the CDT save modified files whenever you perform a manual build. The files are saved before the build is performed so that the latest version of your files is built. You can view the output of the make utility in the Console view.

To save resources before manual builds:

1. Click **Window > Preferences**.
2. Select **Workbench** from the list.



3. On the Workbench page, select the **Save automatically before build** check box.
4. Click **OK**.

The CDT will now save your resources when you build a project.

Related concepts

Build overview

Related tasks

Defining build settings

Building

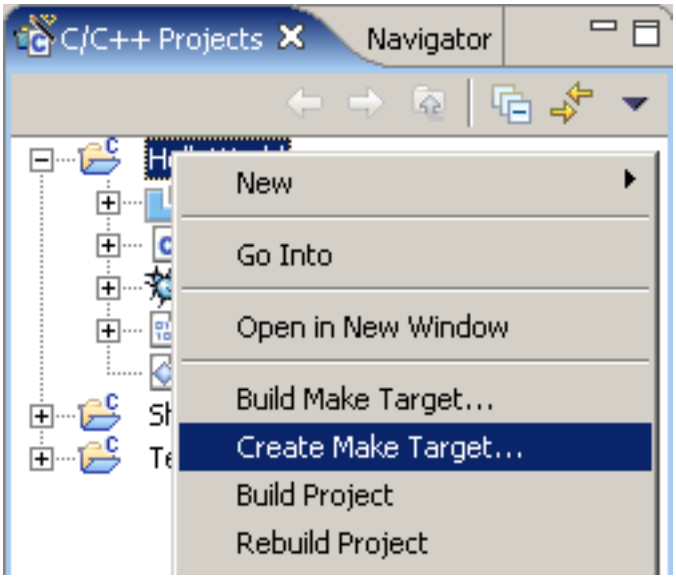
Related reference

Make Builder page, C/C++ Properties window

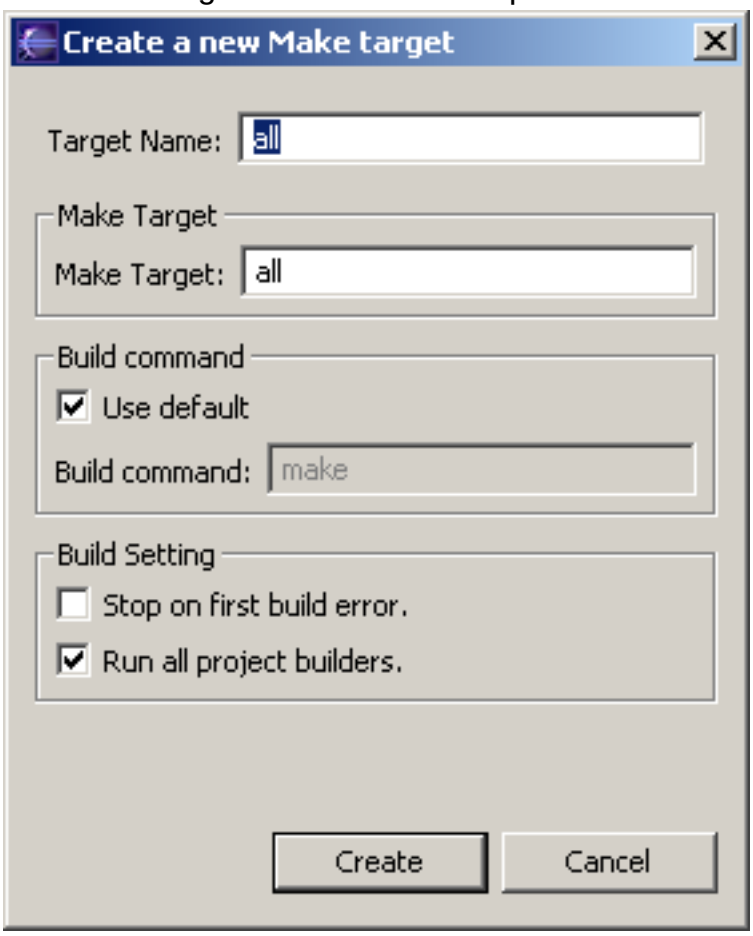
Creating a Make Target

To create a make target:

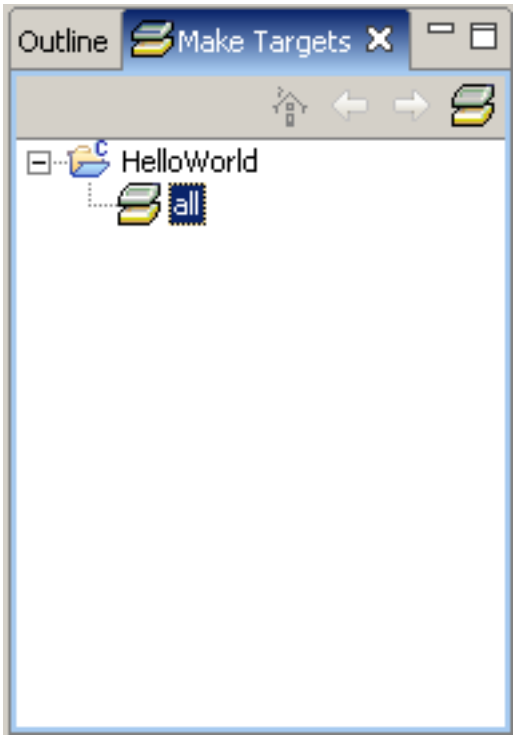
1. Right click on the project and select **Create Make Target...**



2. Enter the Target name and build options and click **Create**.



3. The make target will appear in the Make Targets view.



Related concepts

Build overview

Related tasks

Defining Build Settings

Building

Related reference

Create a Make Target

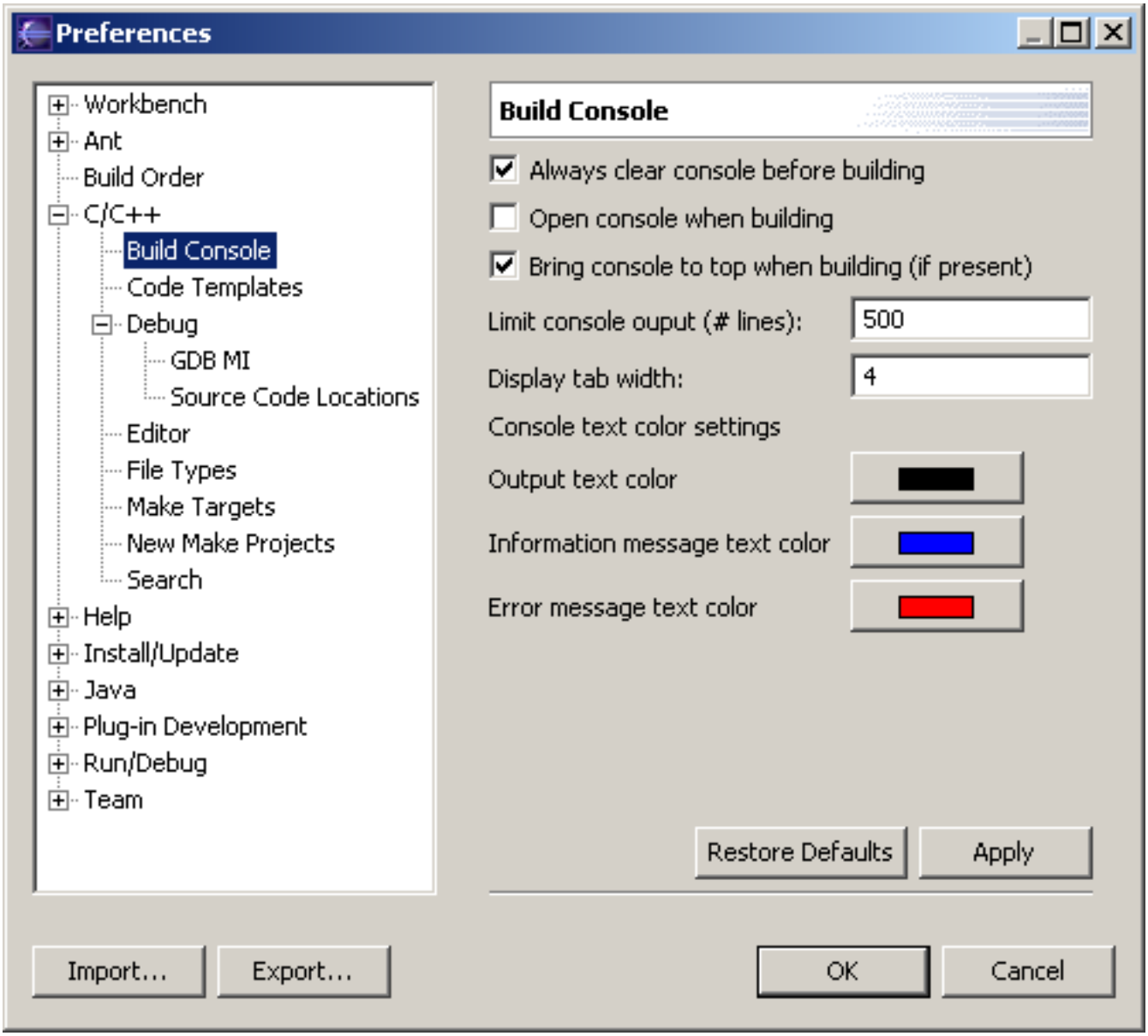
Make Target View

Customizing the Console view

The Console view displays the output of the utilities invoked when building a project or the programs output when running/debugging..

To set Console view preferences

1. Click **Window > Preferences**.
2. Expand **C/C++**, and click **Build Console**.



3. To display information on the latest build only, select the **Always clear console before building** check box.
4. To open the Console view when a project is built, select the **Open console when building** check box.
5. To display the console if it is already open when a project is built, select the **Bring the console to top when building (if present)** check box.
6. To change the number of lines the console stores in its history, enter a new value in the text field

next to **Limit console output (# lines)**.

7. To change the number of spaces displayed by a tab, enter the number in the txt field next to **Display tab width**.
8. Click **OK**.

Related concepts

Build overview

Related tasks

Defining Build Settings

Building

Related reference

Views

Viewing and managing compile errors

This section describes how to view and manage compile errors. Compile errors are displayed in the console view.

Jumping to errors

The CDT will parse the output from the make and compiler/linker. If the CDT can determine the location of an error, it is added to the Console view.

To jump to the source of an error:

- In the Console view, double-click the **Error** icon  or the **Warning** icon  .
  The file opens in the C/C++ editor and the cursor moves to the line with the error.

To jump to errors sequentially:

- Click **Jump to next** or **Jump to previous**.

Related concepts

Build overview

Related tasks

Defining Build Settings
Filtering the Tasks view
Setting reminders

Related reference

Run and Debug dialog box

Filtering the Problems view

Depending on the complexity and stage of your program, an overwhelming number of errors can be generated. You can customize Problems view to only view certain types of errors.

To filter errors:

1. In Problems view, click the **Filters** icon  .
2. To view all errors and warnings, select all checkboxes in the **Show items of type** list, and click **On any resource**.
3. Click **OK**.

Related concepts

Build overview

Related tasks

Defining Build Settings
Jumping to errors
Setting reminders

Related reference

Run and Debug dialog box

Setting reminders

The Tasks view lets you create your own tasks. In addition to having the Tasks view automatically list build errors, you can set personal reminders for tasks, such as unfinished functions that you write or error handling routines that you want to verify.

To set a reminder:

1. In Tasks view, right-click the **Tasks** pane, and select **New Task**.
2. In the **Description** box, type a new task.
3. Click **OK**.

For more information on the Tasks view, see:

- **Workbench User Guide > Concepts> Views > Tasks view**
- **Workbench User Guide > Reference > User interface information > Views and editors > Tasks view**

Related concepts

Build overview

Related tasks

Defining Build Settings

Jumping to errors

Filtering the Tasks view

Related reference

Run and Debug dialog box

Running and debugging projects

This section explains how to run a C or C++ application using an existing run configuration and how to create a new run configuration.

Creating or editing a run/debug configuration

You can run an application by right-clicking the file and clicking Open With > System Editor or you can create a run configure a run environment with which to run your application.

This section explains how to create a run or debug configuration.

The Run and Debug dialog boxes each contain the following tabs:

- Main
- Arguments
- Environment
- Debugger
- Source
- Common

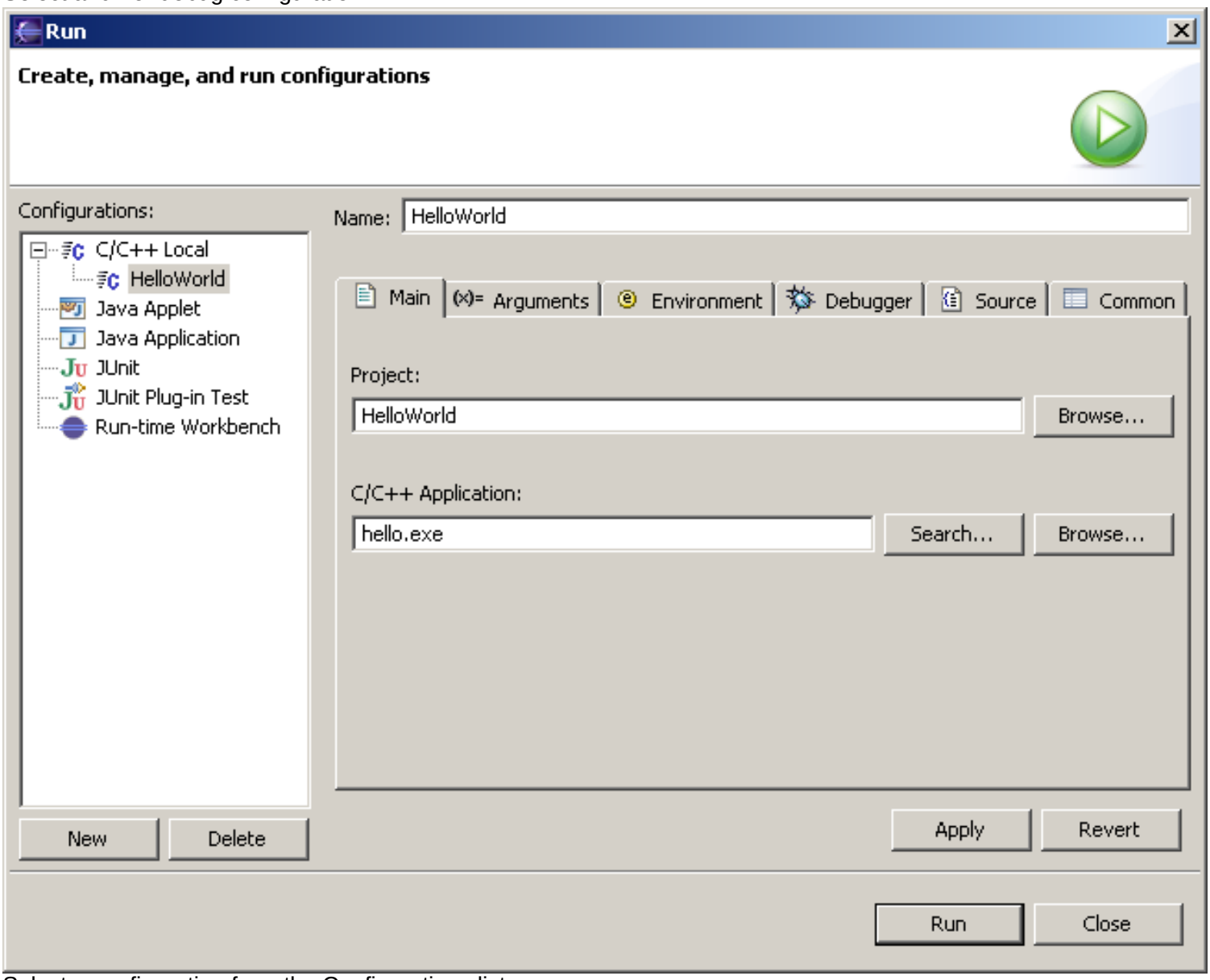
Selecting a run or debug configuration
Creating a run or debug configuration
Selecting an application to run or debug
Specifying execution arguments
Setting environment variables
Defining debug settings
Specifying the location of source files
Specifying the location of the run configuration

Selecting a run or debug configuration

You can select an existing run configuration to use to run your program.

To select a run configuration:

1. In the C/C++ Projects view, select a project.
2. Click **Run > Run** or **Run > Debug**.
3. In the **Configurations** box, expand **C/C++ Local**.
4. Select a run or debug configuration.



5. Select a configuration from the Configurations list.
6. Click **Run** or **Debug**.

Related concepts

CDT Projects

Project file views

Related tasks

Creating or editing a run configuration

Related reference

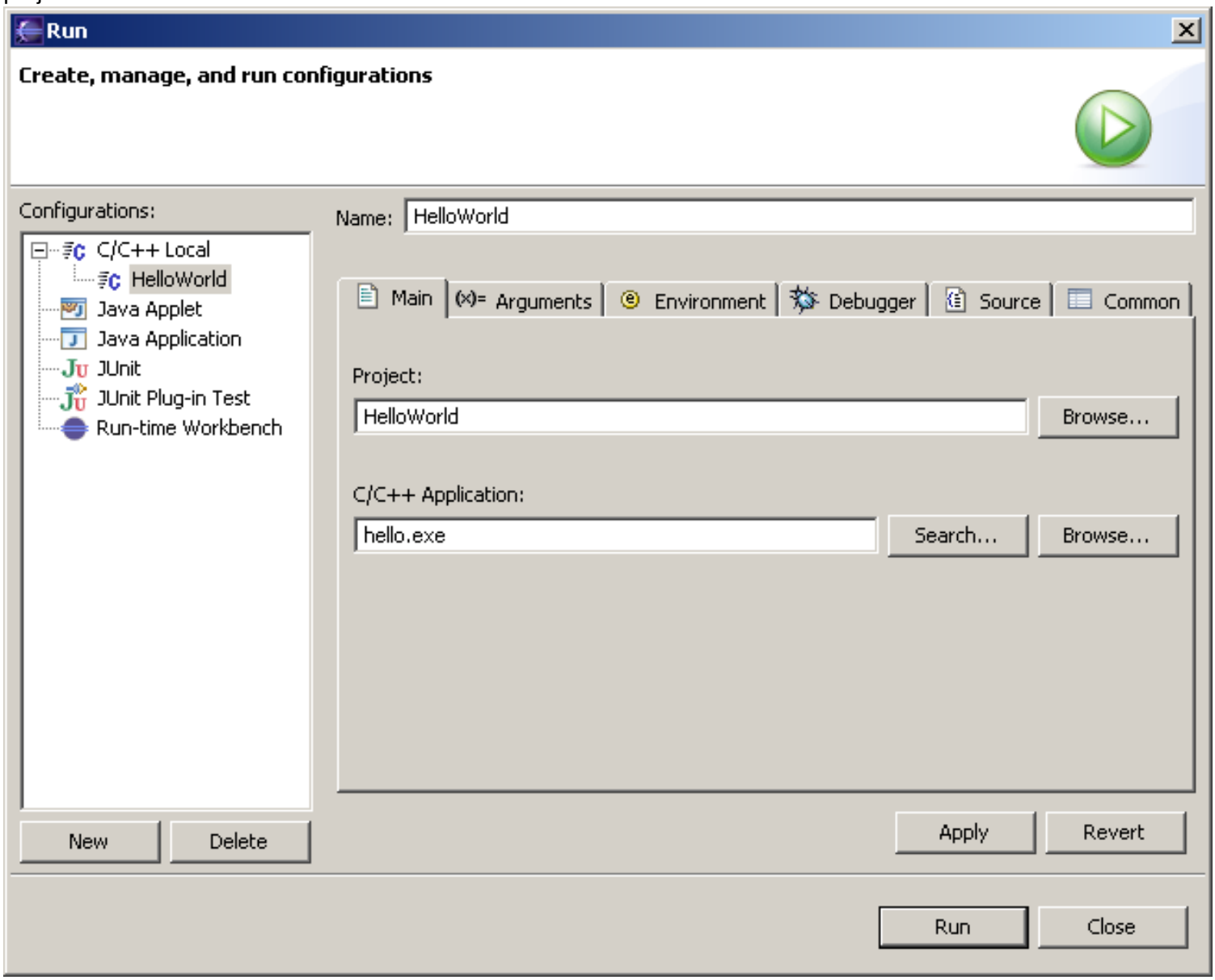
Run and Debug dialog box

Creating a run or debug configuration

You can create customized run configuration which you can save for reuse.

To create a run configuration:

1. In the C/C++ Projects view, select a project.
2. Click **Run > Run** or **Run > Debug**.
3. In the **Configurations** box, expand **C/C++ Local**.
4. Click **New**. The name of the new project is displayed in the Configurations box. The default name is the name of the project.



5. To change the default name of the new run/debug configuration, see Selecting an application to run or debug.

Related concepts

CDT Projects

Project file views

Related tasks

Creating or editing a run configuration

Related reference

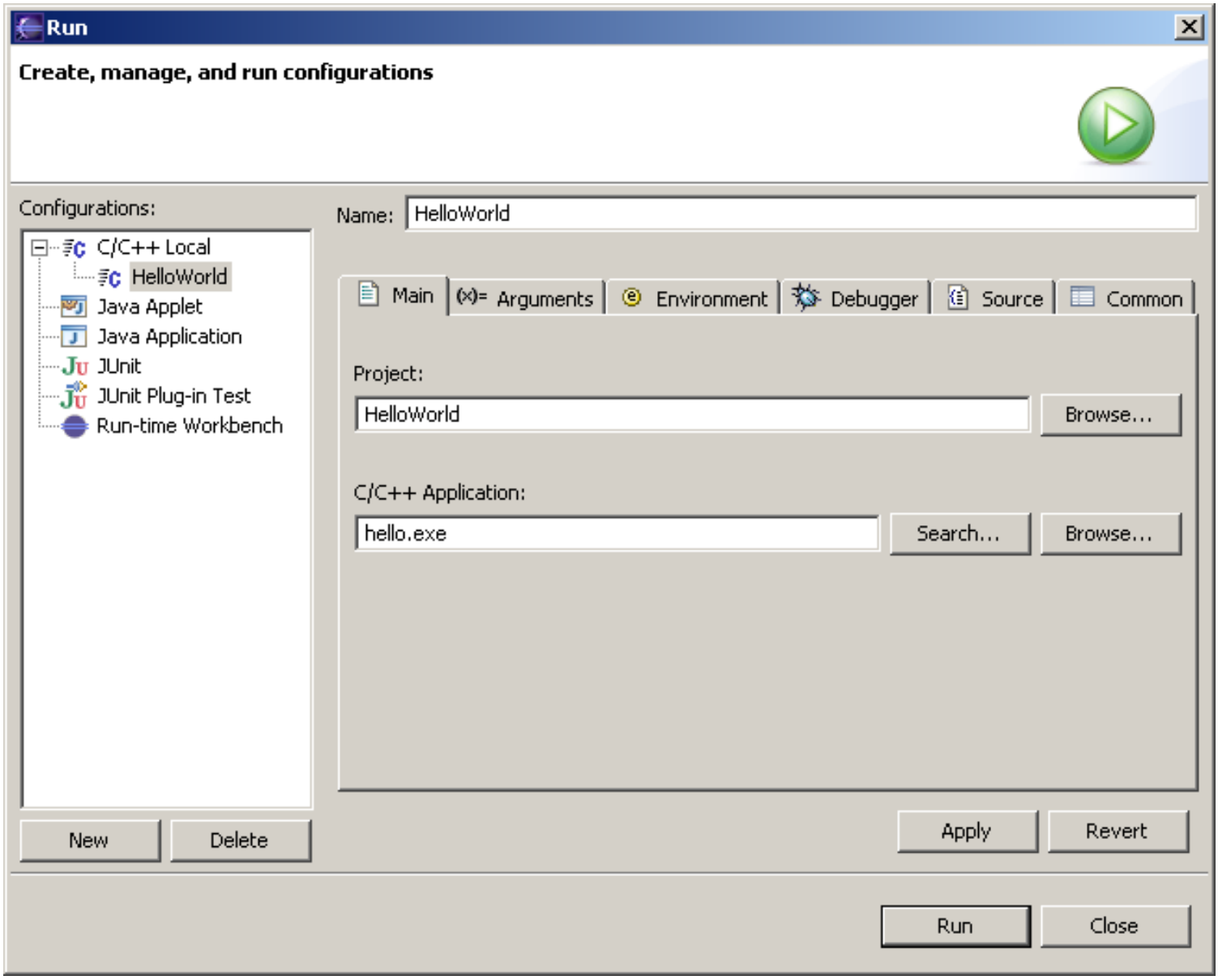
Run and Debug dialog box

Selecting an application to run or debug

You need to specify the project or program that you want to run or debug for this run configuration.

To select an application to run:

1. In the C/C++ Projects view, select a project.
2. Click **Run > Run** or **Run > Debug**.
3. In the **Configurations** box, expand **C/C++ Local**.
4. Select a run or debug configuration.
5. Click the **Main** tab.



6. Do the following:
   - In the **Name** box, type a descriptive name for this new a descriptive name for this launch configuration.
   - In the **Project** box, type the name of the project containing the application that you want to run.
   - In the **C/C++ Application** box, type the name of the executable that you want to run.

7. Click **Run** or do the following, as required:
   - To specify the execution arguments that an application uses, and to specify the working directory for a run configuration, see Specifying execution arguments.
   - To set the environment variables and values to use when an application runs, see Setting environment

variables

- To select a debugger to use when debugging an application, see Selecting a debugger
- To specify the location of source files used when debugging a C or C++ application, see Specifying the location of source files
- To specify where the run configuration is stored, how you access it and the perspective to open when running an application, see Specifying the location of the run configuration

Related concepts

CDT projects

Project file views

Related tasks

Creating or editing a run configuration

Related reference

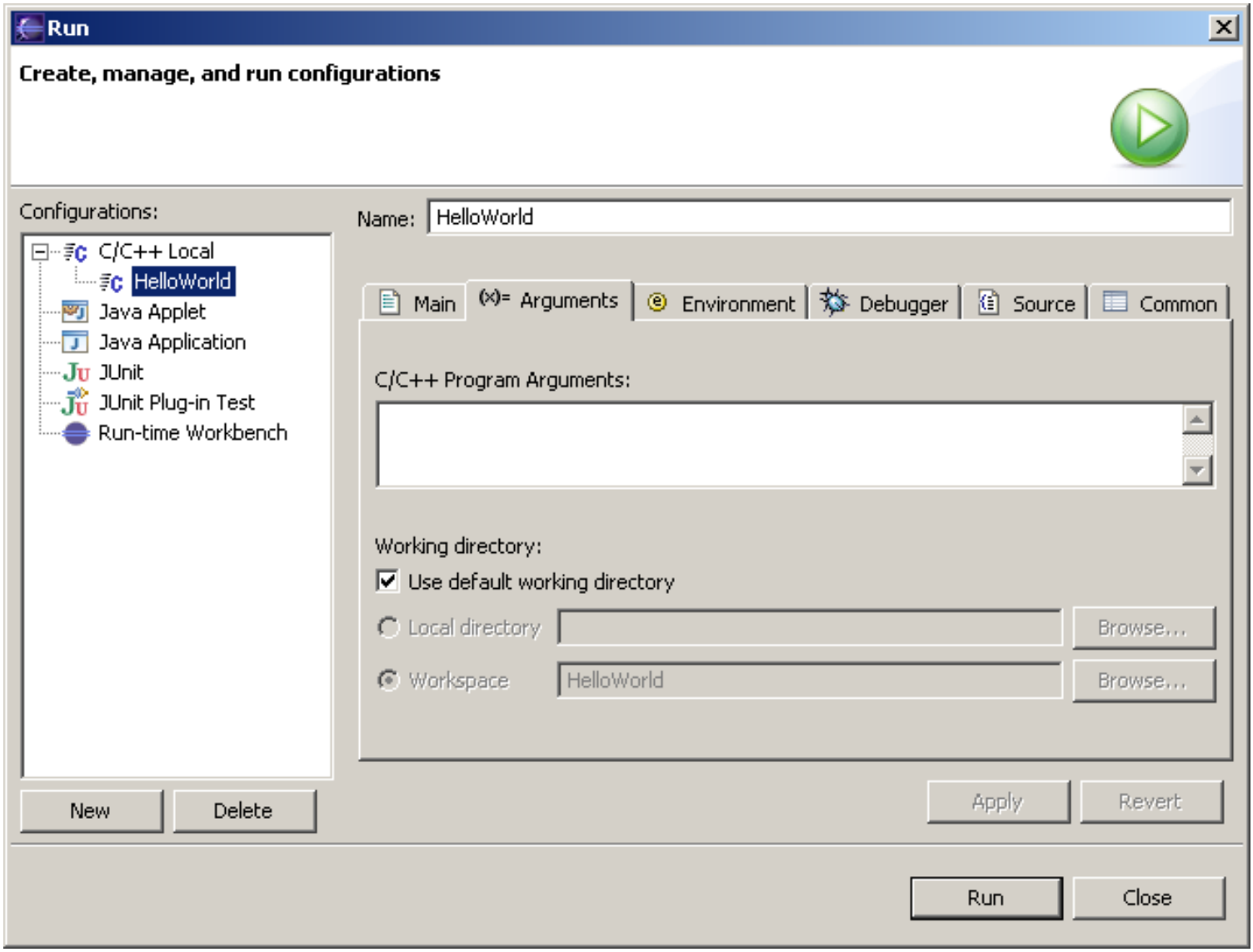
Run and Debug dialog box

Specifying execution arguments

You can specify the execution arguments that an application uses and the working directory for a run configuration.

To specify execution arguments:

1. In the C/C++ Projects view, select a project.
2. Click **Run > Run** or **Run > Debug**.
3. In the **Configurations** box, expand **C/C++ Local**.
4. Select a run or debug configuration.
5. Click the **Arguments** tab.



6. In the **C/C++ Program Arguments** box, type the arguments that you want to pass to the command line.
7. To specify a local directory or a different project in your workspace, clear the **Use default working directory** check box.
8. Click **Run** or do the following, as required:
   - To set the environment variables and values to use when an application runs, see Setting environment variables
   - To select a debugger to use when debugging an application, see Creating a run configuration
   - To specify the location of source files used when debugging a C or C++ application, see Specifying the location of source files
   - To specify where the run configuration is stored, how you access it and the perspective to open when running an application, see Specifying the location of the run configuration

Related concepts

CDT projects

Project file views

Related tasks

Creating or editing a run configuration

Related reference

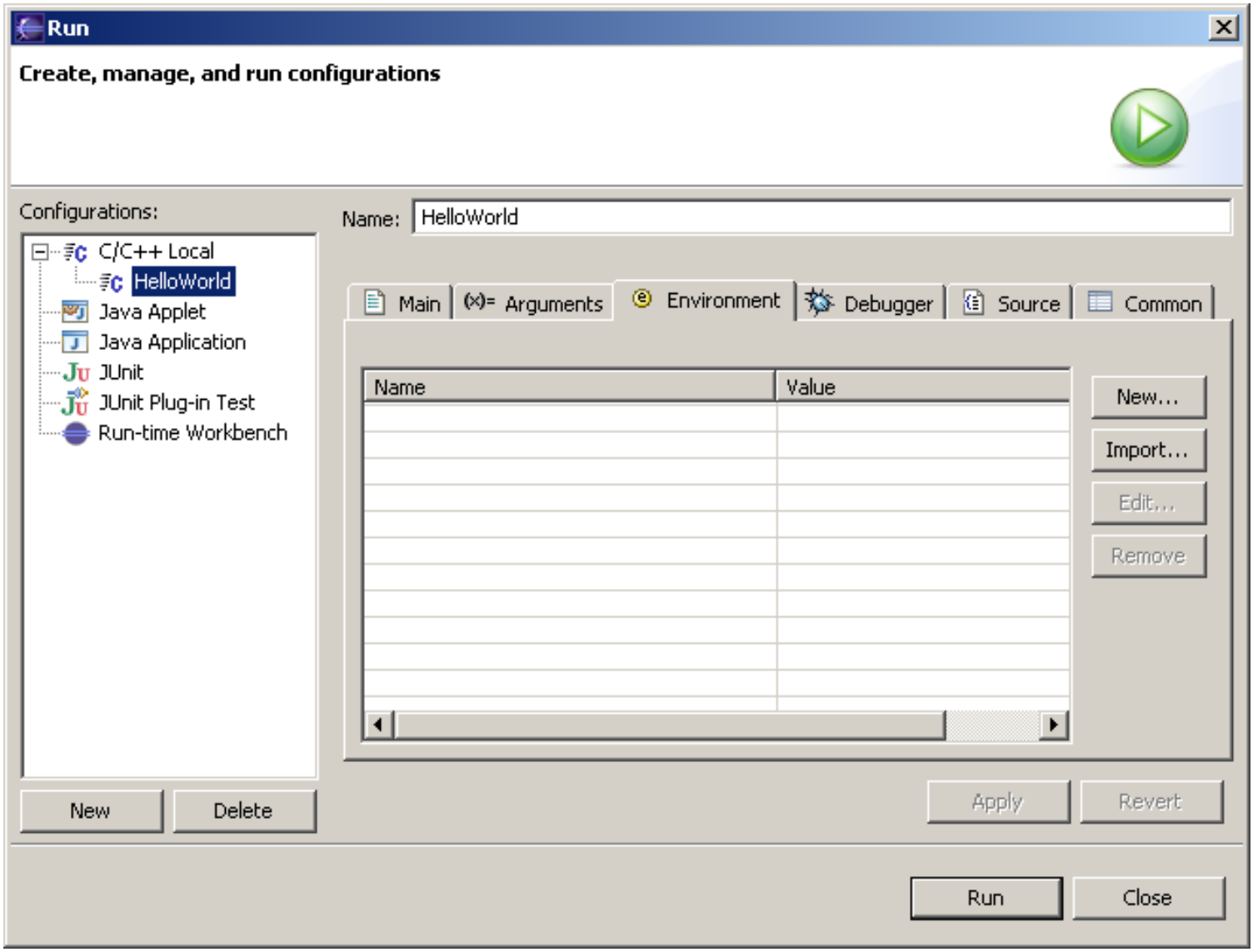
Run and Debug dialog box

Setting environment variables

You can set the environment variables and values to use when an application runs.

To set environment variables:

1. In the C/C++ Projects view, select a project.
2. Click **Run > Run** or **Run > Debug**.
3. In the **Configurations** box, expand **C/C++ Local**.
4. Select a run or debug configuration.
5. Click the **Environment** tab..



6. Do one of the following:
   - To create a new environment variable, click **New**.
   - To import an environment variable, click **Import**.
   - To edit an existing environment variable, select an item from the list and click **Edit**.
   - To remove an existing environment variable, select an item from the list and click **Remove**.
7. Type a name in the **Name** box.
8. Type a value in the **Value** box.
9. Click **Run** or do the following, as required:
   - To specify the execution arguments that an application uses and to specify the working directory for a run configuration, see Specifying execution arguments.

- To select a debugger to use when debugging an application, see Selecting a debugger
- To specify the location of source files used when debugging a C or C++ application, see Specifying the location of source files
- To specify where the run configuration is stored, how you access it, and the perspective to open when running an application, see Specifying the location of the run configuration

Related concepts

CDT Projects

Project file views

Related tasks

Creating or editing a run configuration

Related reference

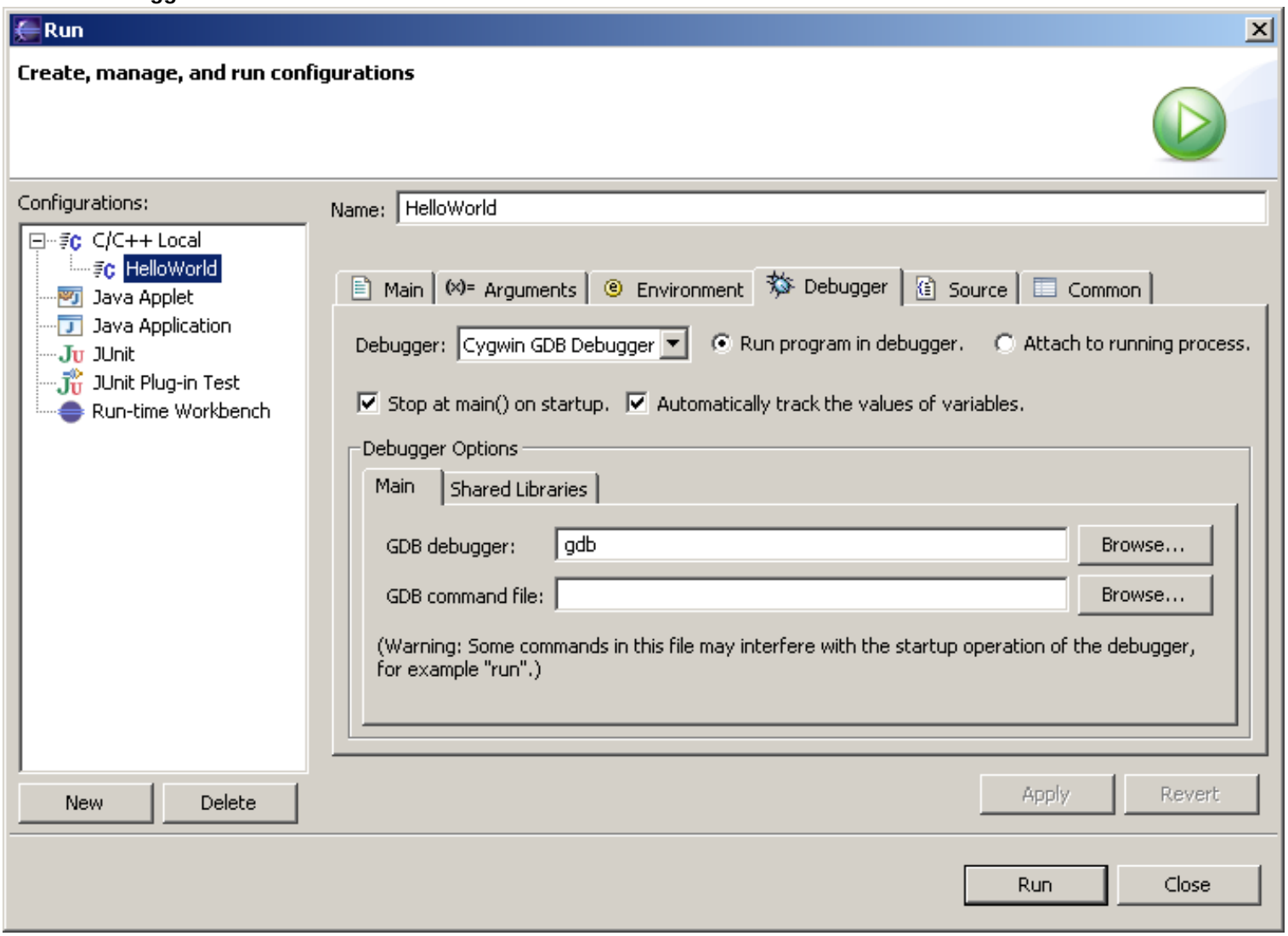
Run and Debug dialog box

Defining debug settings

Select a debugger to use when debugging an application.

To select a debugger:

1. In the C/C++ Projects view, select a project.
2. Click **Run > Run** or **Run > Debug**.
3. In the **Configurations** box, expand **C/C++ Local**.
4. Select a run or debug configuration.
5. Click the **Debugger** tab..



6. Select a debugger from the **Debugger** list.
7. To be prompted to select a process from a list at run-time, select **Attach to running process**.
8. To let your program run until you interrupt it manually, or until it hits a breakpoint, clear the **Stop at main() on startup** check box .
9. Specify debug options in the **Debugger Options** box.
10. Click **Run** or do the following, as required:
    - To specify the execution arguments that an application uses and the working directory for a run configuration, see Specifying execution arguments.
    - To set the environment variables and the values to use when an application runs, see Setting environment variables
    - To specify the location of source files used when debugging a C or C++ application, see Specifying the location of source files
    - To specify where the run configuration is stored, how you access it, and the perspective to open when running an application, see Specifying the location of the run configuration

Related concepts

CDT Projects
Project file views
Debug overview
Debug information

Related tasks

Creating or editing a run configuration

Related reference

Run and Debug dialog box

Specifying the location of source files

You can specify the location of source files used when debugging a C or C++ application. By default, this information is taken from the build path of your project.

To specify the location of source files:

1. In the C/C++ Projects view, select a project.
2. Click **Run > Run** or **Run > Debug**.
3. In the **Configurations** box, expand **C/C++ Local**.
4. Select a run or debug configuration.
5. Click the **Source** tab.
   The Generic Source Locations list shows the location of the project selected in the C/C++ Projects view and any referenced projects.
6. To add an existing source locations:
   - Click **Add** to be prompted to select a process from a list at run-time.
   - In the **Add Source Location** dialog box, select a location type.
   - Click **Next**.
   - Do one of the following:
     - Select an existing project in your workspace. Click **Finish**.
     - Specify a location in your file system. Click **Finish**.
7. You can change the order of source locations are used by selecting a location and clicking the **Up** or **Down** buttons.
8. You can remove a source location by selecting the location and clicking the **Remove** button.
9. To search for duplications in your source locations select the **Search for duplicate source files** checkbox.
10. Click **Run** or do the following, as required:
    - To specify the execution arguments that an application uses and the working directory for a run configuration, see Specifying execution arguments.
    - To set the environment variables and values to use when an application runs, see Setting environment variables.
    - To select a debugger to use when debugging an application, see Selecting a debugger.
    - To specify where the run configuration is stored, how you access it, and the perspective to open when running an application, see Specifying the location of the run configuration.

Related concepts

CDT Projects

Project file views

Related tasks

Creating or editing a run configuration

Related reference

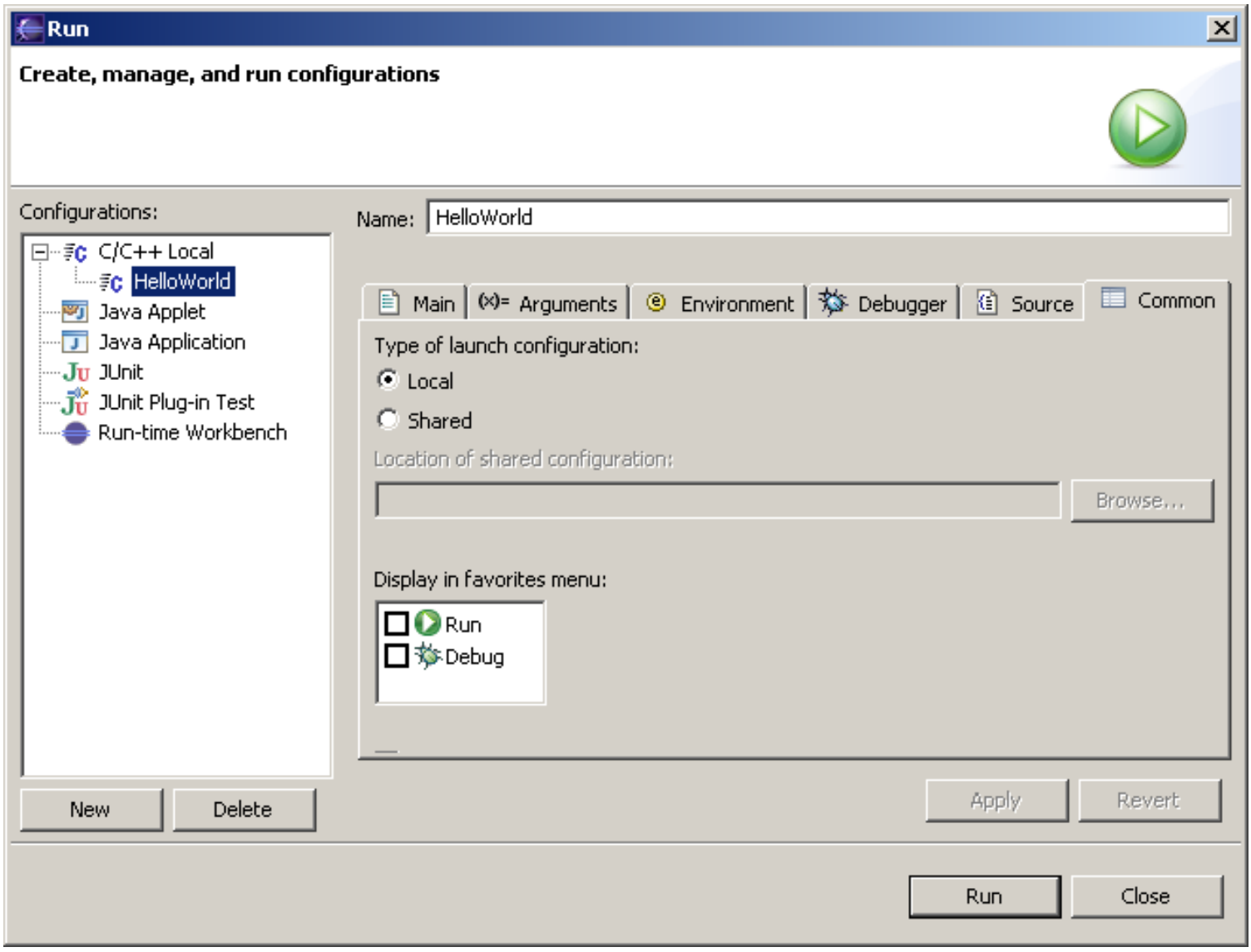
Run and Debug dialog box

Specifying the location of the run configuration

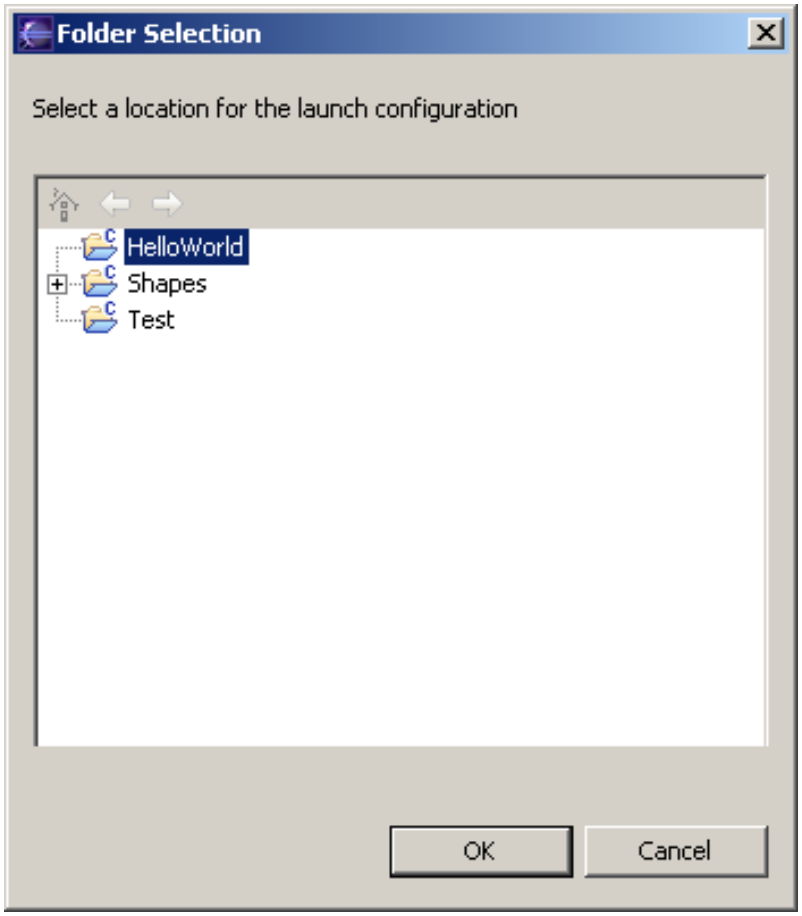
When you create a run configuration, it is saved with the extension `.launch` in `org.eclipse.debug.core`. You can specify an alternate location in which to store your run configuration. You can also specify how you access it and what perspective to open when running an application.

To specify the location of a run configuration:

1. In the C/C++ Projects view, select a project.
2. Click **Run > Run** or **Run > Debug**.
3. In the **Configurations** box, expand **C/C++ Local**.
4. Click the **Common** tab.



5. To save `.launch` to a project in your workspace, and to be able to commit it to CVS, click **Shared**.
6. In the **Folder Selection** window, select a project, and click **OK**.



7. To specify which perspective opens when you run, select a perspective from the **Run mode** list.
8. To specify which perspective opens when you run, select a perspective from the **Debug mode** list.
9. Click **Run**, or do the following, as required:
    - To set the environment variables and values to use when an application runs, see Setting environment variables.
    - To select a debugger to use when debugging an application, see Defining debug settings.
    - To specify the location of source files used when debugging a C or C++ application, see Specifying the location of source files.
    - To specify where the run configuration is stored, how you access it, and the perspective to open when running an application, see Selecting a run/debug configuration.

Related concepts

CDT Projects

Project file views

Related tasks

Creating or editing a run configuration

Related reference

Run and Debug dialog box

Debugging

This section explains how to debug your project.

Debugging a program

You must create a debug launch configuration the first time you debug your program.

To create a debug configuration:

1. In C/C++ Projects view, select a project.
2. Click **Run > Debug**.
3. In the Debug dialog box, select a debug configuration type from the **Configurations** list.
4. Click **New**.
5. In the **Name** box, type a descriptive name for this debug configuration.
6. In the **Project** box, type the name of the project containing the application you want to debug.
7. In the **C/C++ Application** box, type the name of the executable that you want to run.
8. Click the **Debugger** tab.
9. Select **Run program in debugger**.
10. Select the **Stop at main() on startup** checkbox.
11. Click **Debug**.

    The debug perspective is opened and the application window opens on top. The C/C++ editor window is repositioned in the perspective.

For more information:

- To specify the execution arguments an application uses and the working directory for a run configuration, see Specifying execution arguments.
- To set the environment variables and values to use when an application runs, see Setting environment variables
- To select a debugger to use when debugging an application, see Selecting a debugger
- To specify the location of source files used when debugging a C or C++ application, see Specifying the location of source files
- To specify where the run configuration is stored, how you access it and what perspective to open when running an application, see Specifying the location of the run configuration

To use a debug configuration:

You can reuse a previously created debug launch configuration to debug your program.

1. Click **Run > Debug**.
2. In the Debug dialog box, select a debug configuration from the Configurations list.
3. Click **Debug**.

Related concepts
Debug overview

Debug information

Related tasks

Debugging

Related reference

Run and Debug dialog box

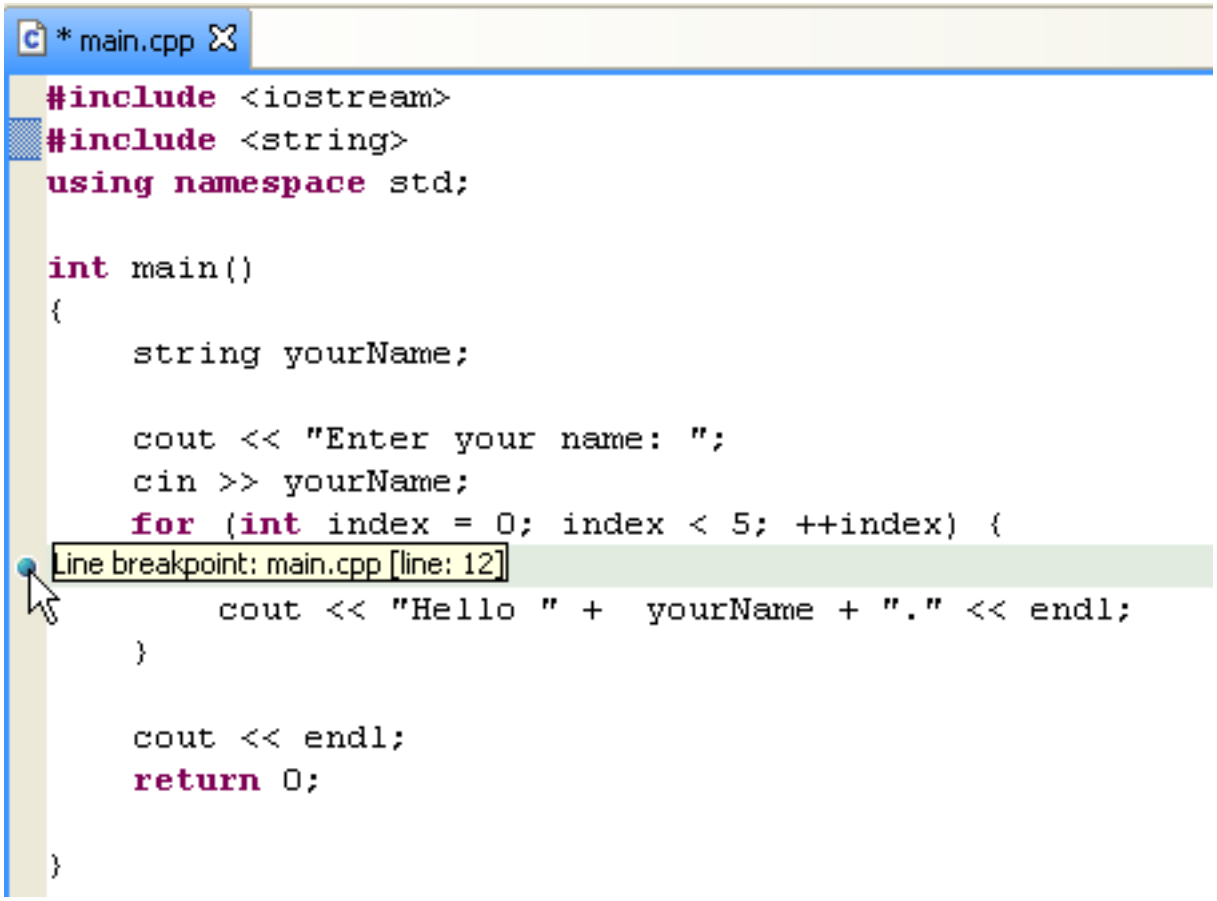
Working with breakpoints and watchpoints

This section explains how to work with breakpoints and watchpoints.

Adding breakpoints

A breakpoint is set on an executable line of a program. If the breakpoint is enabled when you debug, the execution suspends before that line of code executes.

To add a breakpoint point, double click the marker bar located in the left margin of the **C/C++ Editor** beside the line of code where you want to add a breakpoint. A dot is displayed in the marker bar and in the **Breakpoints** view, along with the name of the associated file.



For more information on marker bar icons, see **Workbench User Guide > Reference > User interface information > Icons and buttons > Editor area marker bar**.

Related concepts

C/C++ Development perspective

Related tasks

Working with breakpoints and watchpoints

Related reference

Run and Debug dialog box

Adding watchpoints

A watchpoint is a special breakpoint that stops the execution of an application whenever the value of a given expression changes, without specifying where this may happen. Unlike breakpoints which are line-specific watchpoints are associated with files. They take effect whenever a specified condition is true regardless of when or where it occurred.

To add a watchpoint:

1. Click **Run > Add C/C++ Watchpoint**.
    1. If **Add C/C++ Watchpoint** is not listed on the Run menu, select **Window > Customize Perspective**.
    2. In the Customize Perspective dialog box, expand **Other** in the **Available Items** list.
    3. Select the **C/C++ Debug** check box. Click **OK**.
2. In the **Add C/C++ Watchpoint** dialog box, type an expression in the **Expression to watch** box. The expression may be anything that can be evaluated inside an `if` statement.
3. Do any of the following:
    - To stop execution when the watch expression is read, select the **Read** check box.
    - To stop execution when the watch expression is written to, select the **Write** check box.
4. In the C/C++ editor, open the file that you added the watchpoint to.
5. Click **OK**.
6. The watchpoint appears in the **Breakpoints** view list.

Related concepts

C/C++ Development perspective

Related tasks

Working with breakpoints and watchpoints

Related reference

Run and Debug dialog box

Removing breakpoints and watchpoints

When you remove a breakpoint or watchpoint, the corresponding icon is removed from the marker bar where it was inserted and the Breakpoints view.

To remove breakpoints or watchpoints:

1. In the Breakpoints view, do one of the following:
    - Select the breakpoints and watchpoints you want to remove.
    - Click **Edit > Select All**.
    - Right-click, click **Select All**.
2. In the Breakpoints view, right-click and select **Remove** or **Remove All**.

For more information on marker bar icons, see **Workbench User Guide > Reference > User interface information > Icons and buttons > Editor area marker bar**.

Related concepts

C/C++ Development perspective

Related tasks

Working with breakpoints and watchpoints

Related reference

Run and Debug dialog box

Enabling and disabling breakpoints and watchpoints

You can temporarily disable a breakpoint or watchpoint without losing the information it contains.

To enable or disable breakpoints or watchpoints:

1. In the Breakpoints view, do one of the following:
    - Select the breakpoints and watchpoints that you want to remove.
    - Click **Edit > Select All**.
    - Right-click, and select **Select All**.
2. In the Breakpoints view, right-click the highlighted breakpoints and watchpoints and click **Disable** or **Enable**.

Related concepts

C/C++ Development perspective

Related tasks

Working with breakpoints and watchpoints

Related reference

Run and Debug dialog box

Controlling debug execution

The debug execution controls are superceded by breakpoints. For example, if you attempt to step over a function and the program hits a breakpoint, it pauses, regardless of whether the function is completed. You can control your debug execution in various ways, but they all rely on a core set of debug controls.

To control a debug execution:

1. In the **Debug** view, select a thread.
2. To complete the debug session, click:
    - **Run > Resume**
    - **Run > Suspend**
    - **Run > Terminate**
    - **Run > Disconnect**
    - **Run > Remove All Terminated Launches**
    - **Run > Restart**

Related concepts

C/C++ Development perspective

Related tasks

Debugging

Related reference

Debug launch controls

Debug view

Stepping into assembler functions

Disassembly mode lets you can examine your program as it steps into functions that you do not have source code for [such as printf()]. When the instruction pointer enters a function for which it does not have the source, the function is displayed in the Assembly editor.

When disassembly mode is disabled, the debugger steps over functions for which you do not have the source.

To step into assembler functions during debugging:

- In the Debug view, right-click, and select **Disassembly Mode**.

  As you step Into assembler functions, the execution trace is displayed in the Assembly Editor.

Related concepts
C/C++ Development perspective

Related tasks
Debugging

Related reference
Debug views

Working with variables

During a debug session, you can display variable types, and change or disable variable values.

To display variable type names:

- In Variables view, click the **Show Type Names** toggle button.

To change a variable value while debugging:

During a debug, you can change the value of a variable to test how your program handles a particular value or to speed through a loop.

1. In Variables view, right-click a variable, and select **Change Variable Value**.
2. Type a value.

To disable a variable value while debugging:

You can disable a variable so that the debugger does not read the variable's value from the target. This is useful if the target is very sensitive or the variable is specified as volatile.

- In Variables view, right-click a variable, and select **Disable**.

Related concepts

C/C++ Development perspective

Related tasks

Debugging

Related reference

Debug views

Adding expressions

You can add and view expressions in the Expressions view. The Expressions view is part of the Debug perspective.

To add an expression:

1. Click **Run > Add Expression**.
2. Type the expression that you want to evaluate. For example, (x-5)*3).
3. Click **OK**.
   The expression and its value appear in the Expressions view. When the execution of a program is suspended, all expressions are reevaluated and changed values are highlighted.

Related concepts

C/C++ Development perspective

Related tasks

Debugging

Related reference

Debug views

Working with registers

You can modify registers in the Registers view.

To modify Registers:

1. In the Registers view, right-click a register and click **Change Register Value**.
2. Type a new value.
3. Press **Enter**.

   The Register value is highlighted in red whether or not it was changed.

To change the number system displayed:

You can change the number system used to display register values.

1. In the Registers view, right-click a register, and select **Format**.
2. Type a new value.
3. Do one of the following:
   - Click **Natural**.
   - Click **Decimal**.
   - Click **Hexadecimal**.

To modify Registers view preferences:

1. Click **Window > Preferences**.
2. Expand **Debug**, and click **Registers View**.
3. Make the required changes, and click **OK**.

Related concepts

C/C++ Development perspective

Related tasks

Debugging

Related reference

Debug views

Working with memory

You can inspect and change process memory.

The Memory view supports the same addressing as the C language. You can address memory using expressions such as:

- `0x0847d3c`
- `(&y)+1024`
- `*ptr`

You can configure your output to display hexadecimal or decimal. You can also set the number of display columns and the memory unit size. You can configure each memory tab independently.

You can customize the Memory view to colors and fonts displayed. You can also customize some of its behavior. The customizations affect the entire **Memory** view.

To change process memory:

Warning: Changing process memory can cause a program to crash.

1. In the Debug view, select a process. Selecting a thread automatically selects the associated process.
2. In the Memory view, click a memory tab.
3. Do one of the following:
    - In the **Address** box, type an address and press **Enter**.
    - In the memory view, type a new value for memory. The **Memory** view works in "typeover" mode. To jump from byte to byte use the arrow keys:

To change the appearance of the Memory view:

1. Do one of the following:
    - In the Memory view, click one of the tabs.
    - Click **Window > Preferences**.
2. In list of memory addresses, right-click, and select:
    - **Format > Hexadecimal, Signed Decimal or Unsigned Decimal**.
    - **Memory Unit Size > 1, 2, 4, or 8 bytes**
    - **Number of Columns > 1, 2, 4, 8, or 16** columns.

Related concepts

C/C++ Development perspective

Related tasks

Debugging

Related reference

Debug views

Searching for C/C++ elements

It is recommended that you perform searches on successfully compiled programs to ensure the accuracy of search results. It is important to familiarize yourself with the correct search syntax to use to complete an effective search. It is also important to ensure that include paths and symbols are correctly defined. For more information, see Including paths and symbols.

See C/C++ search, for more information on:

- What you can search for
- How to limit your search
- How to use wildcard characters in your search
- Syntax examples

Performing a C/C++ Search can be done in a number of different ways but regardless of the manner chosen the same information must be provided to C/C++ Search.

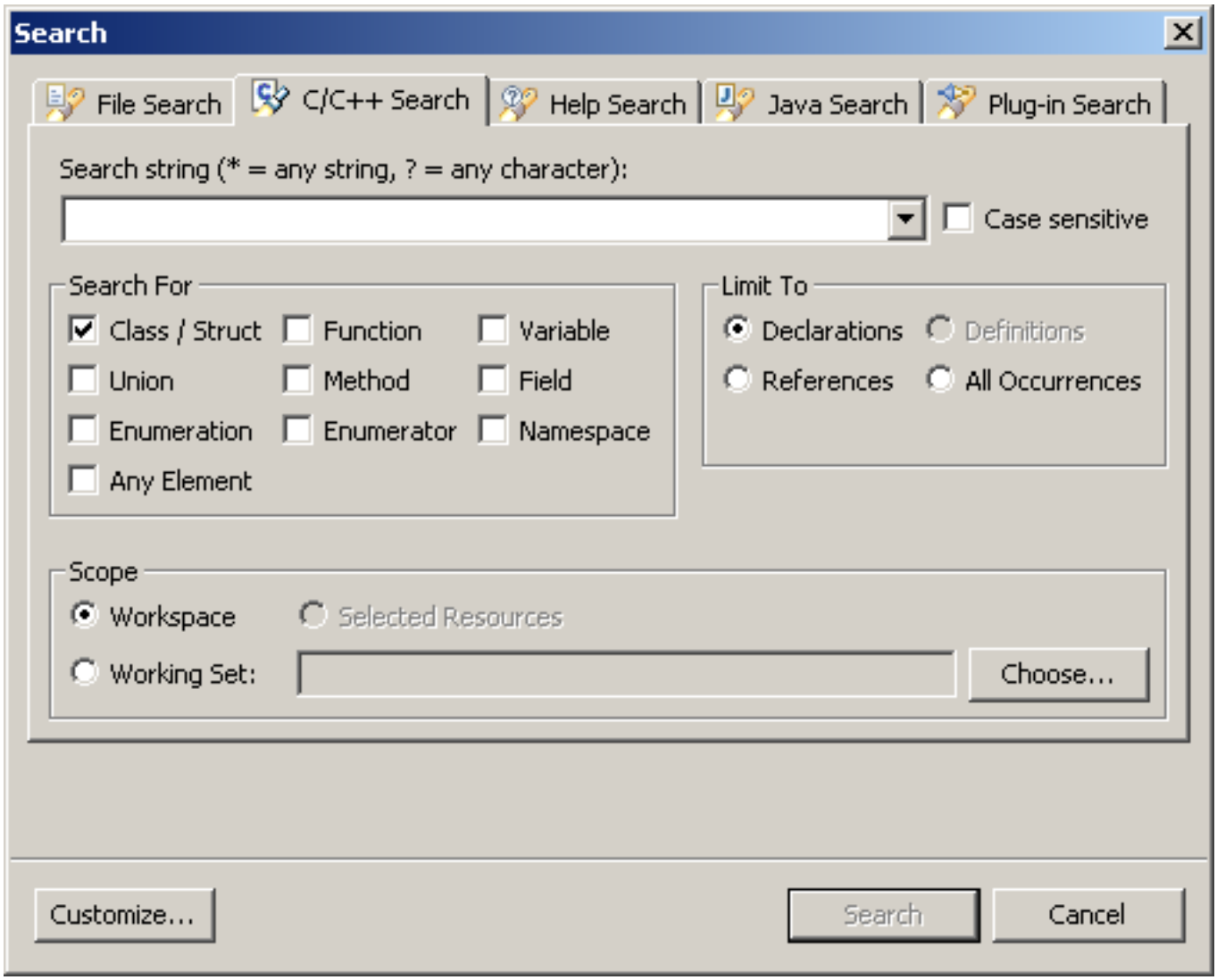
The info required to do a search is:

Search string	this is the name of the element you are looking for. See C/C++ search for more information on how to specify wildcard searches and how to further refine your searches by using fully qualified names
Search For	this is the element type that you wish to search for. You can also select to search on 'Any Element' which will perform the search on a combination of all elements
Limit To	this allows you to limit your search to declarations, definitions or references. You can also select 'All Occurrences' which will search for declarations, definitions and references that match the element
Scope	this allows the user to limit the scope of the search. The three available scopes are: **Workspace** — this searches all of the open projects in the workspace **Selected Resources** — this option becomes enabled whenever something is selected in one of the following views in the C/C++ perspective: • C/C++ Projects • Navigator • Search • Outline The scope will be limited to whatever element is selected in the view. **Working Set** — working sets can be selected and created

There are 3 main ways for initiating a C/C++ search:

- Using the **C/C++ Search** dialog
- Selecting an element in the **Editor** view
- Selecting an element in the **C/C++ Projects** view or Selecting an element from the **Outline** view

Using the C/C++ Search dialog



1. Enter the search string in the **Search String** field (optional mark it case sensitive).
   Note: that previous search queries (from the same work session) are remembered and can be accessed via the drop down list.
2. Select the **Search For** element.
3. Select the **Limit To**.
4. Select the **Scope**.
5. Press **Search**.

Results are displayed in the **Search** view.

Selecting an element in the Editor view

1. Select the desired element in the editor.
2. Right click and select **All Declarations** or **All References** and the scope you wish to search.

Results are displayed in the **Search** view.

Selecting an element in C/C++ Projects or Outline View

1. Select the desired element in the tree.
2. Right click and select **All Declarations** or **All References** and the scope you wish to search.

Results are displayed in the **Search** view.

For more information, see:

- **Workbench User Guide > Concepts > Views > Search view**
- **Workbench User Guide > Concepts > Workbench > Working Set**
- **Workbench User Guide > Tasks > Navigating and finding resources**

Related concepts

C/C++ search
C/C++ Indexer
CDT Projects
Open Declarations

Related tasks

Selection Searching for C/C++ elements
Navigate to C/C++ declarations

Related reference

C/C++ search page, Search dialog box

Selection Searching for C/C++ elements

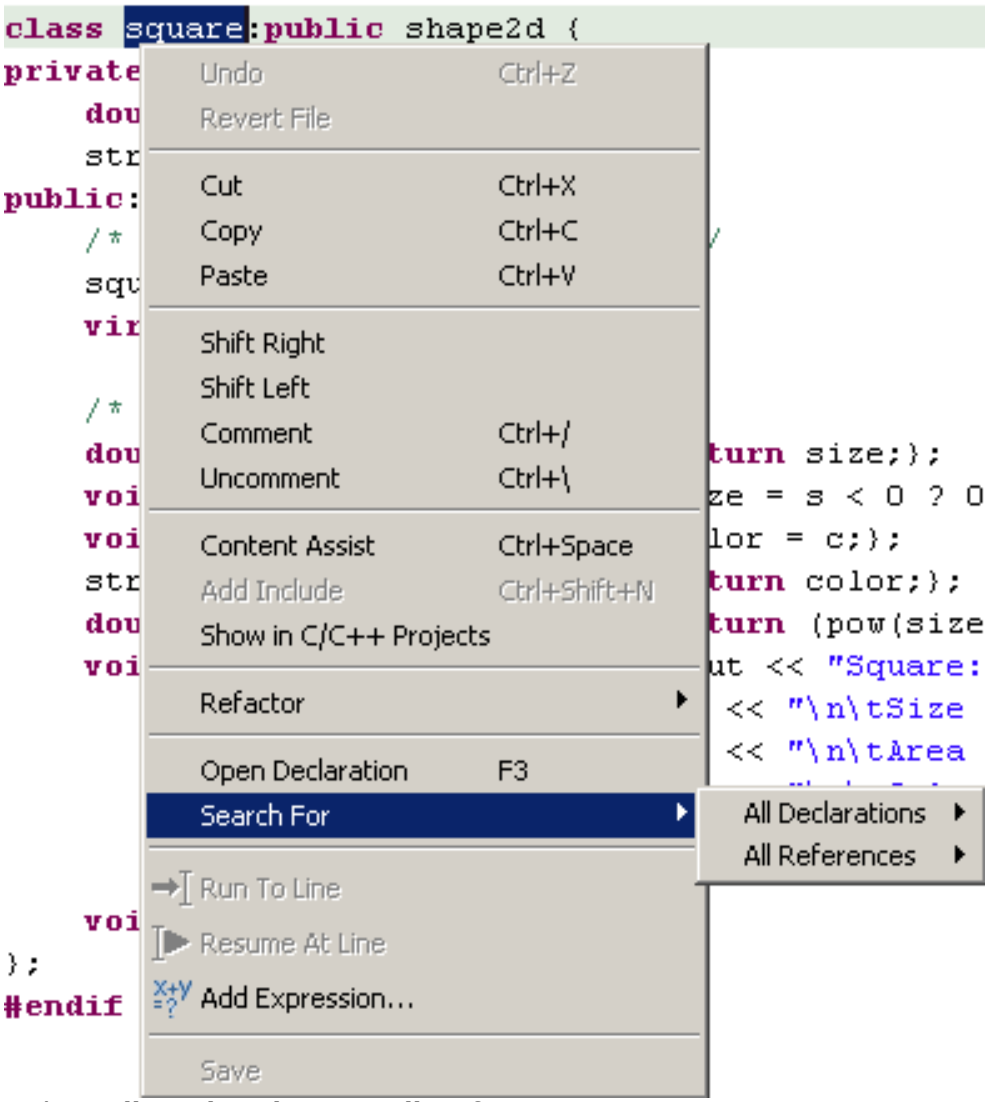
It is recommended that you perform searches on successfully compiled programs to ensure the accuracy of search results. It is important to familiarize yourself with the correct search syntax to use to complete an effective search. It is also important to ensure that include paths and symbols are correctly defined. For more information, see Including paths and symbols.

See C/C++ search, for more information on:

- What you can search for
- How to limit your search
- How to use wildcard characters in your search
- Syntax examples

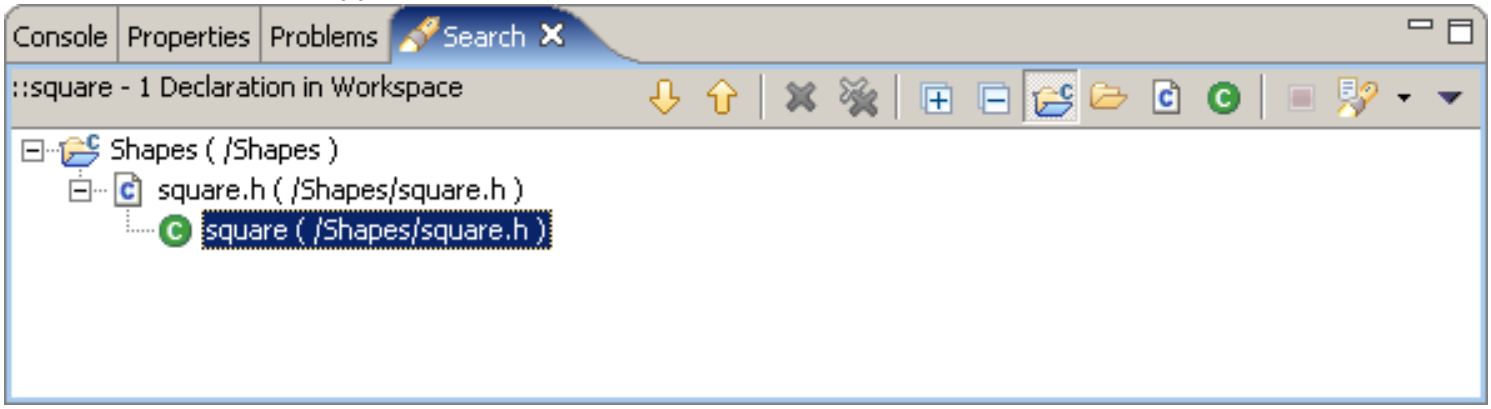
To search for an element in your project:

1. Highlight the element you want to search.
2. Right click and select **Search For** from the context menu.



3. Select **All Declarations** or **All References**.
4. Select **Workspace** or **Working Set...**.

5. The search results will appear in the Search View



For more information, see:

- **Workbench User Guide > Concepts > Views > Search view**
- **Workbench User Guide > Tasks > Navigating and finding resources**

Related concepts

C/C++ search
CDT Projects
Open Declarations

Related tasks

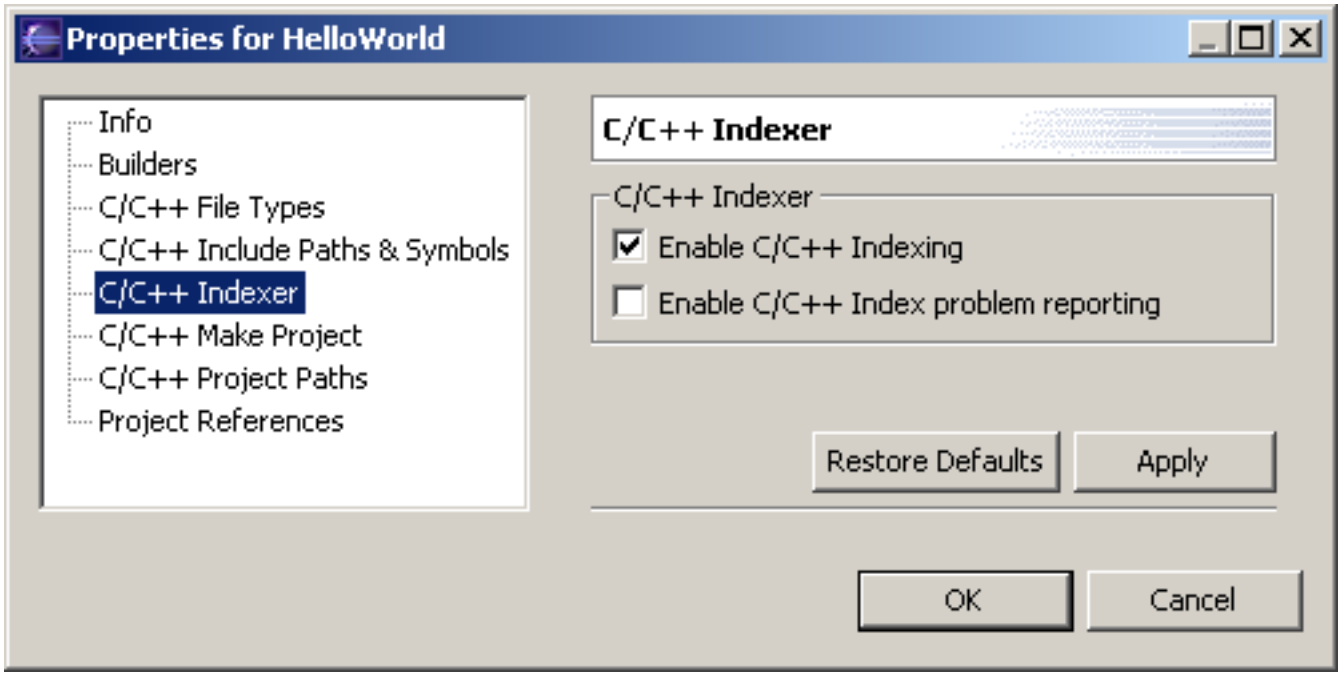
Searching for C/C++ elements
Navigate to C/C++ declarations

Related reference

C/C++ search page, Search dialog box

Enable/Disable the C/C++ Indexer

The C/C++ indexer is on by default. You can enable/disable indexing on any project from the C/C++ perspective.



The indexer can be enabled or disabled at any time by:

1. Right clicking on the project
2. Selecting **Properties**
3. Navigate to the **C/C++ Indexer** page
4. Select/Deselect the **Enable C/C++ Indexing** checkbox.

If you enable the index on a project that had the index disabled, the indexer will reindex all of the project's source folders.

If you disable the index on a project, it will no longer react to any resource change events. If the indexer is indexing at the time you disable the index, it will throw away the rest of the index jobs at that point.

If some projects in your workspace have the indexer disabled, then search will display a warning message in the status bar. If all projects have the index disabled then Search will not allow you to continue until at least one project has the index enabled.

You can also disable the indexer when you first create a new project by clicking on the C/C++ Indexer tab (which is present in both Standard and Managed project wizards) and deselecting the **Enable C/C++ Indexing** checkbox.

Related concepts

C/C++ search
C/C++ Indexer
C/C++ Indexer Problem Reporting
C/C++ Indexer Opening or Closing a project
C/C++ Indexer Progress Bar

Related tasks

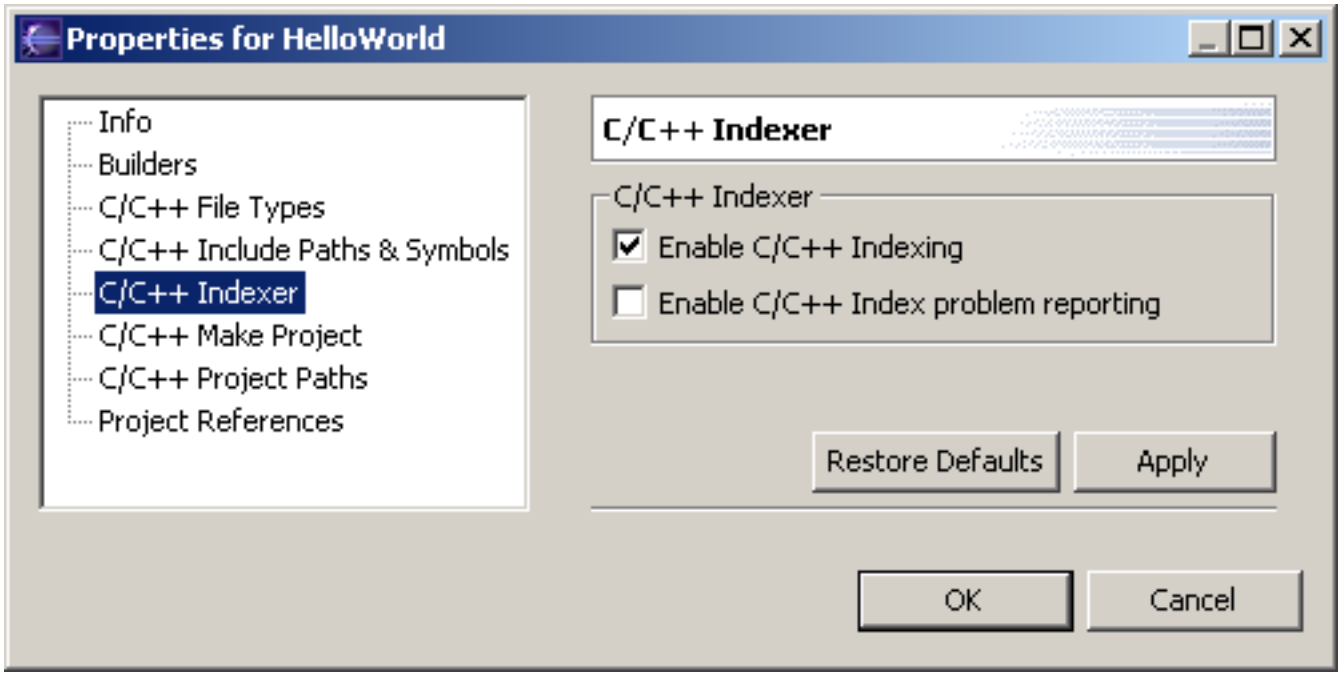
Selection Searching for C/C++ elements
C/C++ Indexer Problem Reporting
C/C++ Indexer - Indexer Timeout
Setting Source Folders

Related reference

Search, C/C++ Preferences window
C/C++ search page, Search dialog box
C/C++ Project Properties, Managed, Indexer
C/C++ Project Properties, Standard, Indexer

C/C++ Indexer Problem Reporting

C/C++ Index Problem reporting places a problem marker on the editor and adds an item to the error list for each preprocessor or semantic problem reported by the parser. Note that the markers will only show up the next time the file is indexed.



To enable C/C++ Index Problem reporting:

1. Right click on the project and select **Properties > C/C++ Indexer**
2. Select the **Enable C/C++ Index problem reporting** checkbox
3. Click **OK**

Related concepts

C/C++ search
C/C++ Indexer
C/C++ Indexer Problem Reporting
C/C++ Indexer Opening or Closing a project
C/C++ Indexer Progress Bar

Related tasks

Selection Searching for C/C++ elements
Enable/Disable the C/C++ Indexer
C/C++ Indexer - Indexer Timeout
Setting Source Folders

Related reference

Search, C/C++ Preferences window

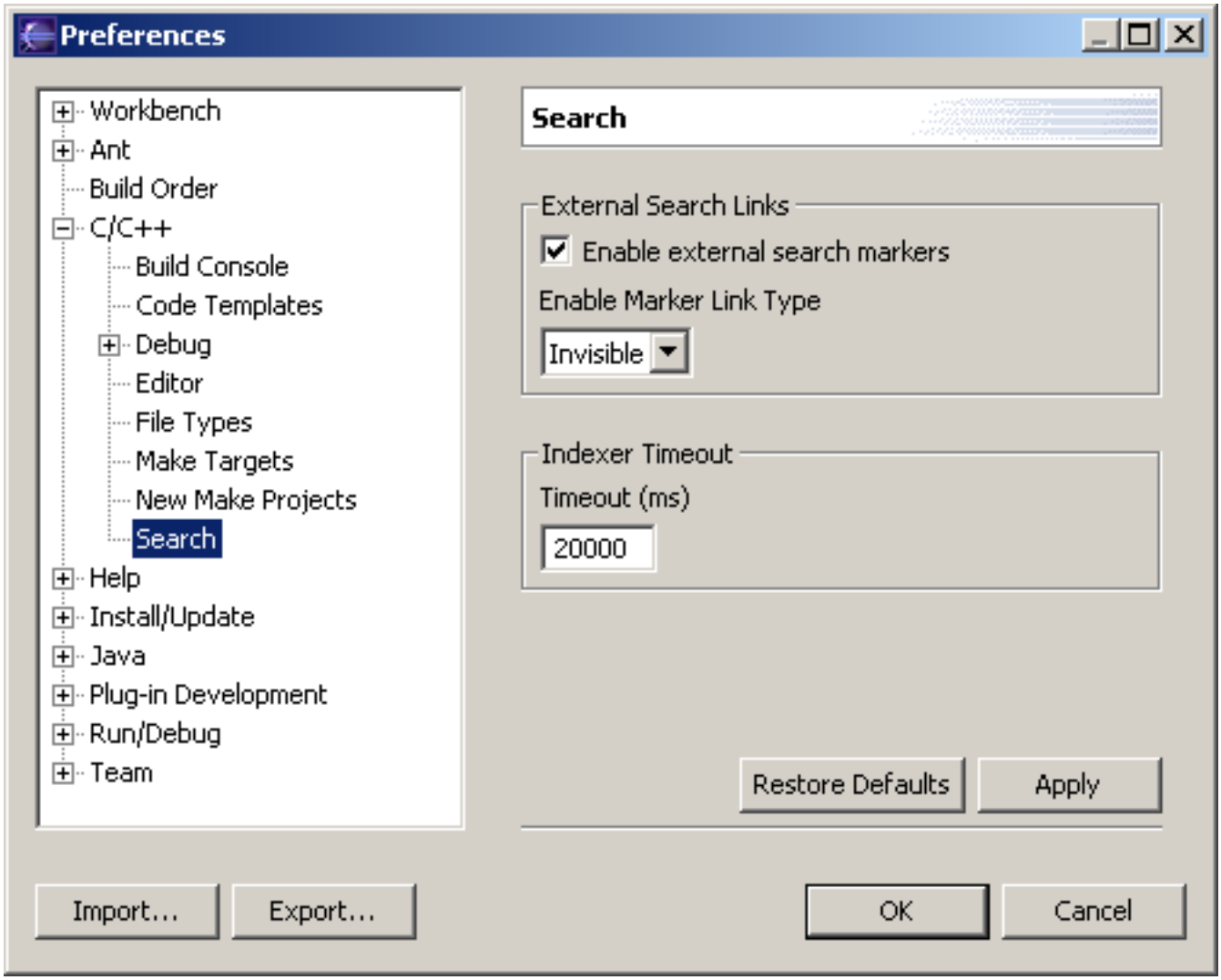
C/C++ search page, Search dialog box

C/C++ Project Properties, Managed, Indexer

C/C++ Project Properties, Standard, Indexer

C/C++ Indexer – Indexer Timeout

If the indexer gets stuck on a particular file while indexing there is a timeout watchdog, which will terminate the indexing attempt after a certain period.



This can be set as follows:

1. Click **Windows > Preferences > C/C++ > Search**
2. Enter the timeout value in milliseconds into the text field.
3. Click **OK**

Related concepts

C/C++ search
C/C++ Indexer
C/C++ Indexer Problem Reporting
C/C++ Indexer Opening or Closing a project
C/C++ Indexer Progress Bar

Related tasks

Selection Searching for C/C++ elements
Enable/Disable the C/C++ Indexer
C/C++ Indexer Problem Reporting
Setting Source Folders

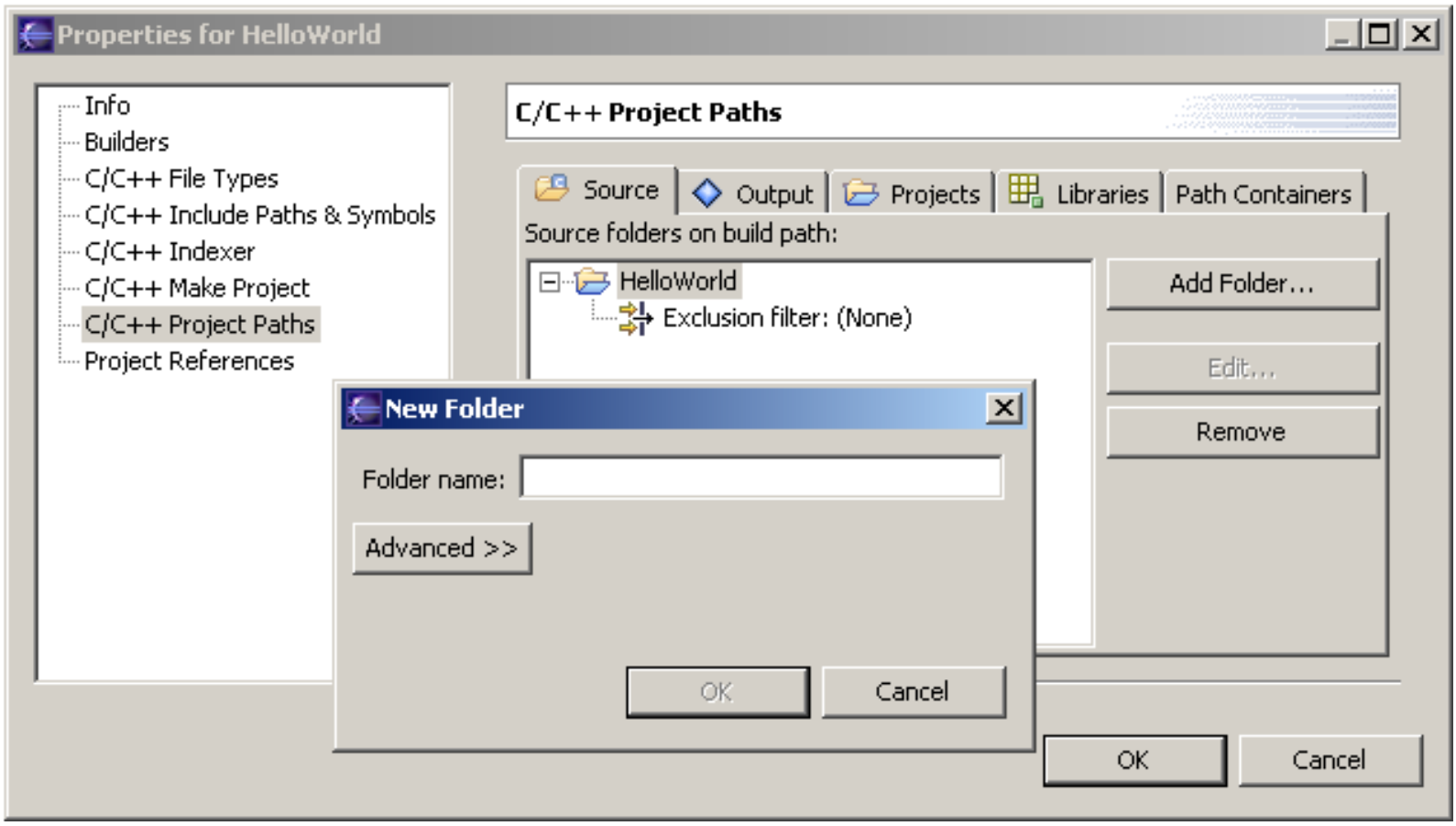
Related reference

Search, C/C++ Preferences window
C/C++ search page, Search dialog box
C/C++ Project Properties, Managed, Indexer
C/C++ Project Properties, Standard, Indexer

Setting Source Folders

Note that source folders can only be currently used with Standard Make projects. Managed Make projects treat the entire project as a source folder.

Source folders are a way to cut down on a project's indexing scope. You can mark the folders that are part of your day to day work or part of a subsystem that you work on. All files inside source folders will be indexed and are, thus, searchable. Note that any files pulled in by a file inside a source folder will also be indexed.



To setup source folders:

1. Right click on the project and select **Properties > C/C++ Project Paths**
2. Click on the **Source** tab
   By default the entire project is a source folder, which means everything will be indexed. This is reasonable for smaller projects but definitely not recommended for large projects.
3. Click **Add folder** to add the folders. A dialog will explain exclusion filters have been added to nesting folders. You will see that the folder you added will be excluded from the project folder (in order to avoid including a folder twice). Repeat until all the folders have been added.
   Note: Don't forget to remove the project folder (which appears by default) otherwise everything will be indexed.
4. Click **OK**. Your view in C/C++ projects will now change. You should see your source folders designated with a "C" and all other source and header files icons that are in a non-source folder will change to a "hollow" C or H.

Related concepts

C/C++ search
C/C++ Indexer
C/C++ Indexer Problem Reporting
C/C++ Indexer Opening or Closing a project
C/C++ Indexer Progress Bar

Related tasks

Selection Searching for C/C++ elements
Enable/Disable the C/C++ Indexer
C/C++ Indexer Problem Reporting
C/C++ Indexer - Indexer Timeout

Related reference

Search, C/C++ Preferences window
C/C++ search page, Search dialog box
C/C++ Project Properties, Managed, Indexer
C/C++ Project Properties, Standard, Indexer

Reference

This section describes the Views, Windows and Dialog Boxes available from the C/C++ perspective.

- C/C++ Editor preferences
  - General preferences
  - Color preferences
  - Content Assist preferences
  - Hover preferences
  - Navigation preferences
- File Types preferences
- Make Targets preferences
- New Make Projects properties preferences
  - Make Builder preferences
  - Error Parser preferences
  - Binary Parser preferences
  - Discovery Options preferences
- Search preferences

C/C++ Project Properties

- Managed Make Projects
  - Info
  - Builders
  - Build
  - File Types
  - Indexer
  - Error Parser
  - Binary Parser
  - Project References
- Standard Make Projects
  - Info
  - Builders
  - File Types
  - Include Paths and Symbols
  - Indexer
  - Make Project
    - Make Builder
    - Error Parser
    - Binary Parser
    - Discovery Options
  - Project Paths
    - Source
    - Output
    - Projects
    - Libraries
    - Path Containers
  - Project References

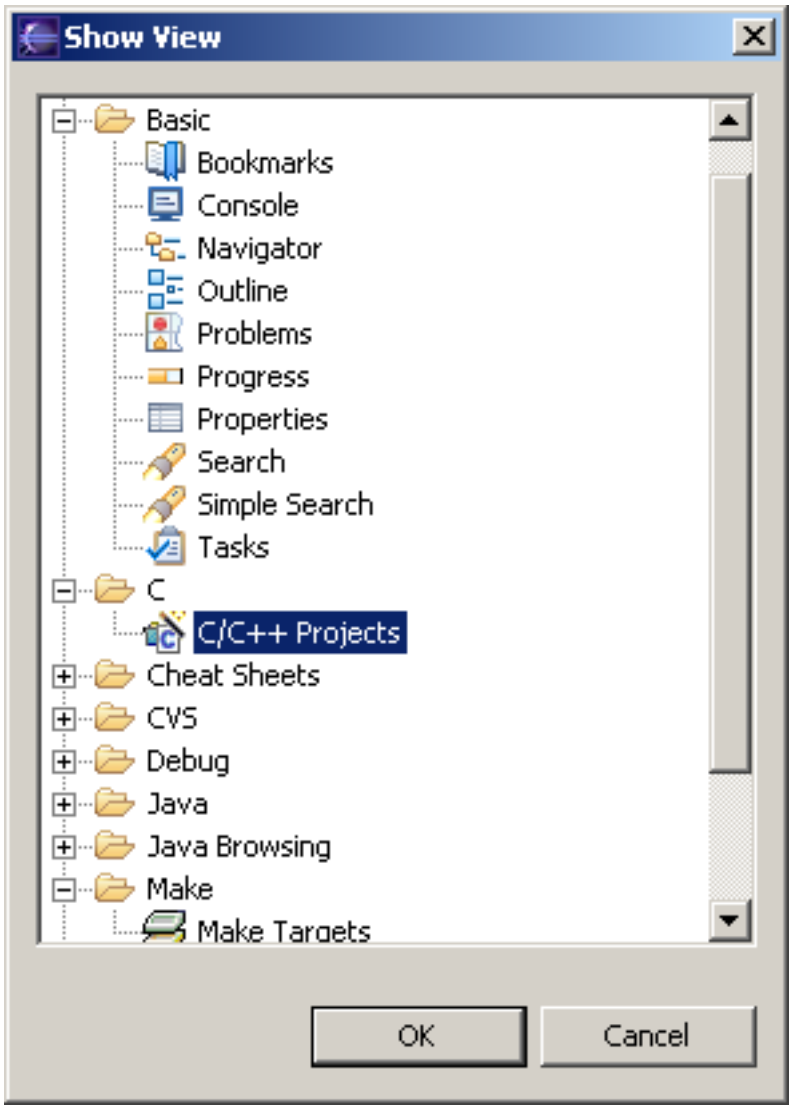
C/C++ New Project Wizard

C/C++ Views and Editors

This section describes views and editors of the C/C++ perspective.

Selecting Views and Editors

To see a list of all views, from the menu bar choose **Window > Show View > Others**.



The following views comprise the C/C++ Projects View:

Basic views

Console
: Displays the application's output.

Navigator
: Displays the file system under the ***launchdir*/workspace** directory.

Outline
: Displays the functions and header files in your source files. Open a source file in an editor to view its outline.

Problems
: Displays problems.

Properties

Displays the name, path, size, permissions, and last modified date of the resource highlighted in the **Navigator** view.

Search

Displays the results of file or text searches.

C views

C/C++ Projects

Displays your current projects. This is similar to the **Navigator** view, except that:

- Only projects and their contents are displayed
- You are able to view the "building blocks" of the files, such as the include files and the functions.

Make views

Make Targets

Lists your projects. To build a project, double-click on it..

Editor view

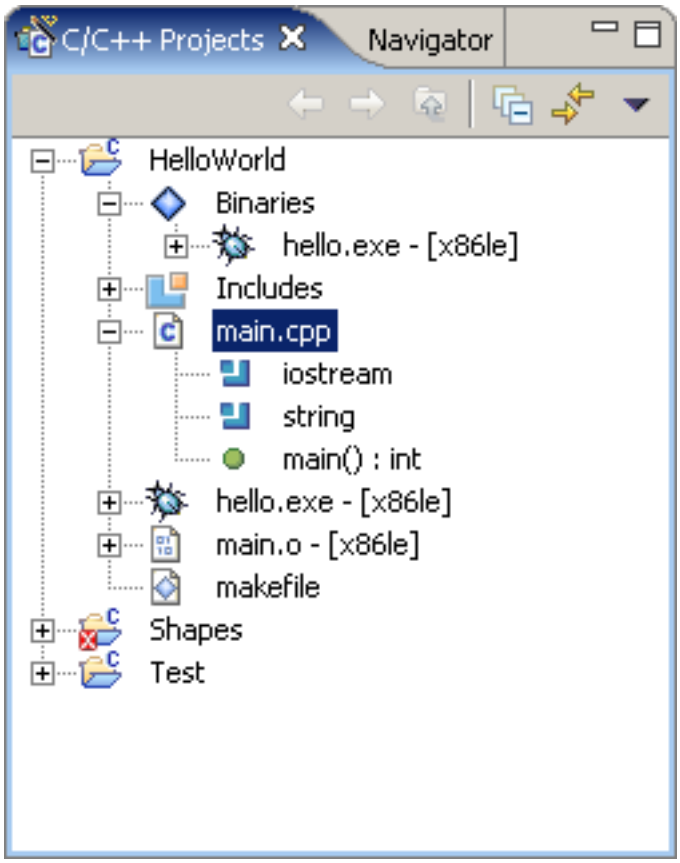
The **Editor** view is not listed under **Window > Show View** or **Window > Show View > Others**, it is opened whenever an editable file is opened from the C/C++ Projects or Navigator views.

C/C++ Projects view

The C/C++ Projects view displays, in a tree structure, only elements relevant to C and C++ project files. In this view you can do the following:

- Browse the elements of C/C++ source files
- Open files in the editor view
- Open projects in a new window
- Create new projects, classes, files, or folders
- Manage existing files (cut, paste, delete, move or rename)
- Restore deleted files from local history
- Import or Export files and projects

Files that you select in the C/C++ Projects view affect the information that is displayed in other views.



Toolbar

Icon	Name	Description
	MinimizeConsole	Minimizes the Console view.
	Maximize Console	Maximizes the Console view.

	**Back**	This command displays the hierarchy that was displayed immediately prior to the current display. For example, if you Go Into a resource, then the Back command in the resulting display returns the view to the same hierarchy from which you activated the *Go Into* command. The hover help for this button tells you where it will take you. This command is similar to the Back button in a web browser.
	**Forward**	This command displays the hierarchy that was displayed immediately after the current display. For example, if you've just selected the Back command, then selecting the Forward command in the resulting display returns the view to the same hierarchy from which you activated the Back command. The hover help for this button tells you where it will take you. This command is similar to the Forward button in a web browser.
	**Up**	This command displays the hierarchy of the parent of the current highest level resource. The hover help for this button tells you where it will take you.
	**Collapse All**	This command collapses the tree expansion state of all resources in the view.
	**Link with Editor**	This command toggles whether the Navigator view selection is linked to the active editor. When this option is selected, changing the active editor will automatically update the Navigator selection to the resource being edited.
	Menu	Click the black upside-down triangle icon to open a menu of items specific to the Navigator view. **Select Working Set** Opens the **Select Working Set** dialog to allow selecting a working set for the Navigator view. **Deselect Working Set** Deselects the current working set. **Edit Active Working Set** Opens the **Edit Working Set** dialog to allow changing the current working set. **Sort** This command sorts the resources in the Navigator view according to the selected schema: ○ **By Name**: Resources are sorted alphabetically, according to the full name of the resource (e.g., A.TXT, then B.DOC, then C. HTML, etc.) ○ **By Type**: Resources are sorted alphabetically by file type/ extension (e.g., all DOC files, then all HTML files, then all TXT files, etc.). **Filters** This command allows you to select filters to apply to the view so that

		you can show or hide various resources as needed. File types selected in the list will not be shown in the Navigator. **Link with Editor** See the toolbar item description above.

C/C++ Projects view icons

The table below lists the icons displayed in the C/C++ Projects view.

Icon	Description
	C or C++ file
	Class
	Code template
	Macro Definition
	Enum
	Enumerator
	Variable
	Field private
	Field protected
	Field public
	Include
	Makefile
	Method private
	Method protected
	Method public
	Namespace
	Struct

	Type definition
	Union
	Function

Related concepts

Project file views

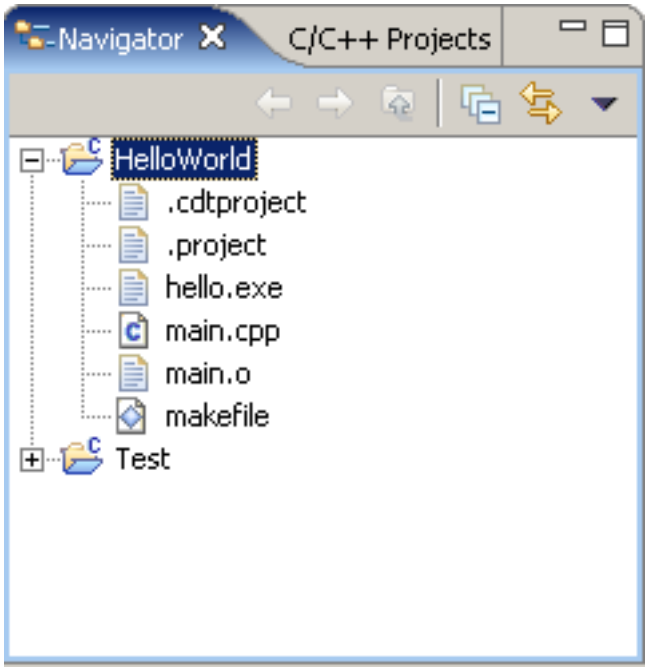
Related tasks

Displaying C/C++ file components in the C/C++ Projects view

Hiding files by type in the C/C++ Projects view

Navigator view

This view provides a hierarchical view of the resources in the Workbench.



Toolbar

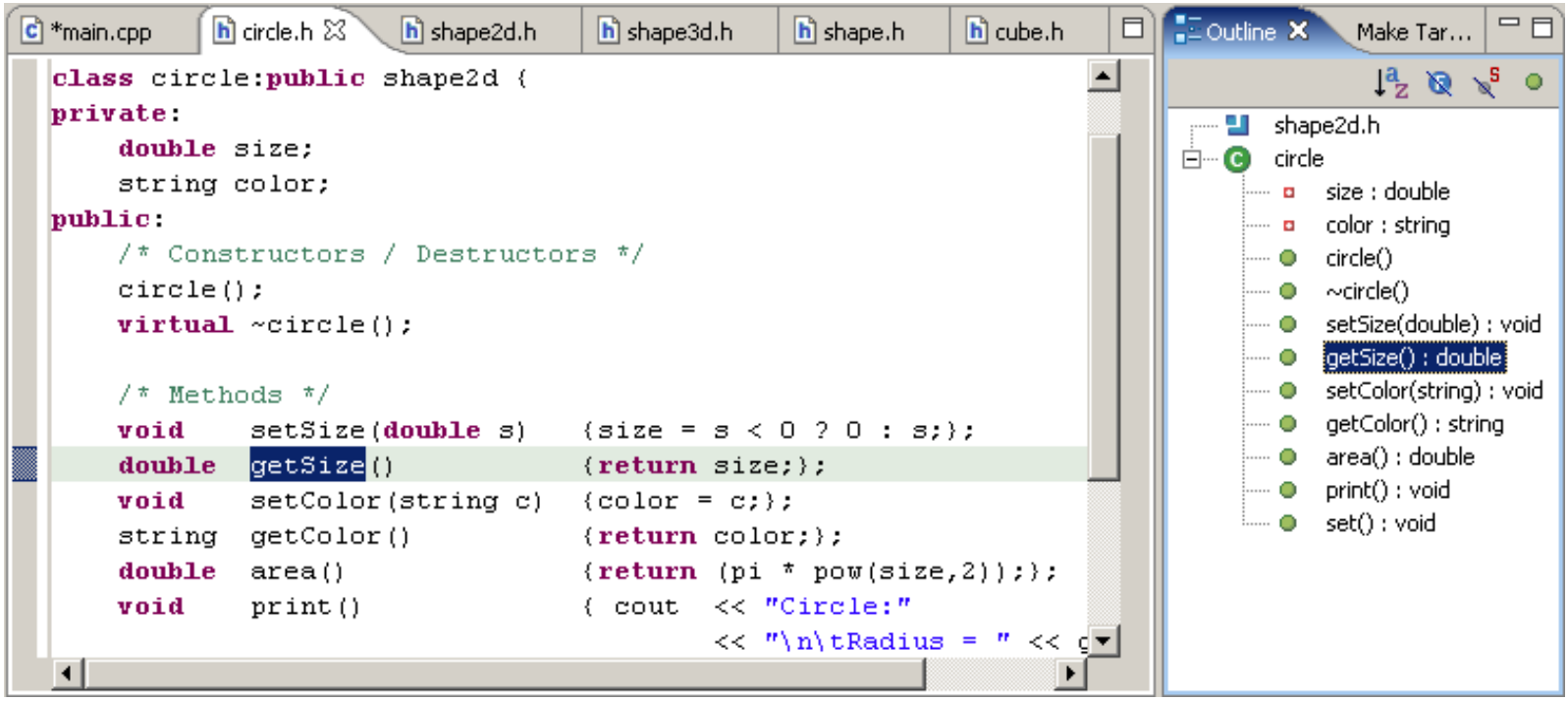
Icon	Name	Description
	MinimizeConsole	Minimizes the Console view.
	Maximize Console	Maximizes the Console view.
	Back	This command displays the hierarchy that was displayed immediately prior to the current display. For example, if you Go Into a resource, then the Back command in the resulting display returns the view to the same hierarchy from which you activated the *Go Into* command. The hover help for this button tells you where it will take you. This command is similar to the Back button in a web browser.
	Forward	This command displays the hierarchy that was displayed immediately after the current display. For example, if you've just selected the Back command, then selecting the Forward command in the resulting display returns the view to the same hierarchy from which you activated the Back command. The hover help for this button tells you where it will take you. This command is similar to the Forward button in a web browser.
	Up	This command displays the hierarchy of the parent of the current highest level resource. The hover help for this button tells you where it will take you.

	**Collapse All**	This command collapses the tree expansion state of all resources in the view.
	**Link with Editor**	This command toggles whether the Navigator view selection is linked to the active editor. When this option is selected, changing the active editor will automatically update the Navigator selection to the resource being edited.
	**Menu**	Click the black upside-down triangle icon to open a menu of items specific to the Navigator view. **Select Working Set** Opens the **Select Working Set** dialog to allow selecting a working set for the Navigator view. **Deselect Working Set** Deselects the current working set. **Edit Active Working Set** Opens the **Edit Working Set** dialog to allow changing the current working set. **Sort** This command sorts the resources in the Navigator view according to the selected schema: ○ **By Name**: Resources are sorted alphabetically, according to the full name of the resource (e.g., A.TXT, then B.DOC, then C. HTML, etc.) ○ **By Type**: Resources are sorted alphabetically by file type/ extension (e.g., all DOC files, then all HTML files, then all TXT files, etc.). **Filters** This command allows you to select filters to apply to the view so that you can show or hide various resources as needed. File types selected in the list will not be shown in the Navigator. **Link with Editor** See the toolbar item description above.

In addition to these menu items, the Navigator view menu shows a list of recently used working sets that have been selected in the view.

Outline view

The Outline view displays an outline of a structured C/C++ file that is currently open in the editor area, by listing the structural elements.



Outline view toolbar icons

The table below lists the icons displayed in the Outline view toolbar.

Icon	Description
	Hide Fields
	Hide Static Members
	Hide Non-Public Members
	Sort items alphabetically

Outline view icons

The table below lists the icons displayed in the Outline view.

Icon	Description
	Class
	Namespace
	Macro Definition
	Enum
	Enumerator

	Variable
	Field private
	Field protected
	Field public
	Include
	Method private
	Method protected
	Method public
	Struct
	Type definition
	Union
	Function

Related concepts

Outline view

Related tasks

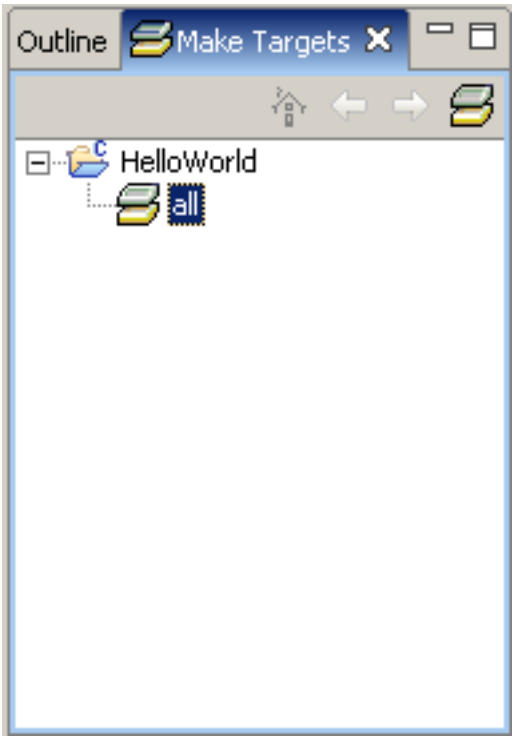
Displaying C/C++ file components in the C/C++ Projects view

Related reference

Views

Make Targets view

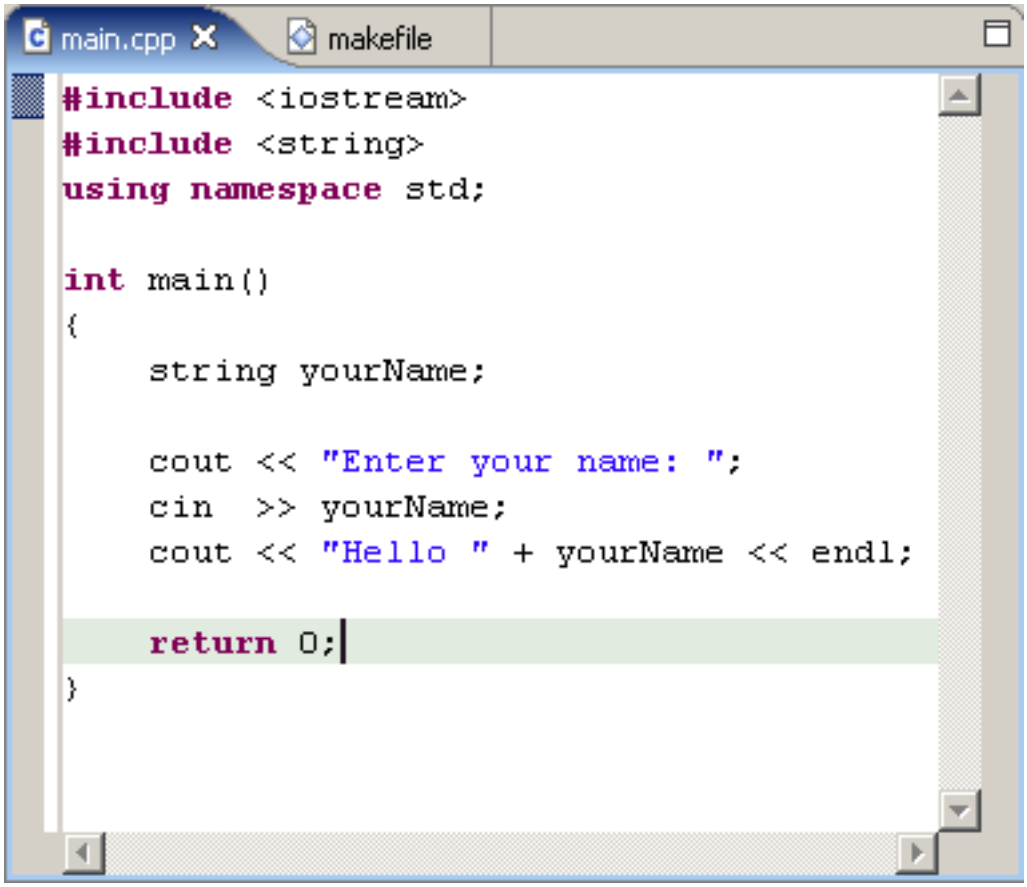
Enables you to select the make targets you want to build in your workspace.



Icon	Command	Description
	Home	Move to the top level.
	Back	Navigates back to a previous level.
	Forward	Navigates forward to the next level.
	Build Target	Builds currently selected target.

Editor view

The C/C++ editor provides specialized features for editing C/C++ related files.



Associated with the editor is a C/C++-specific Outline view, which shows the structure of the active C, C++ or makefile. It is updated as you edit these files.

The editor includes the following features:

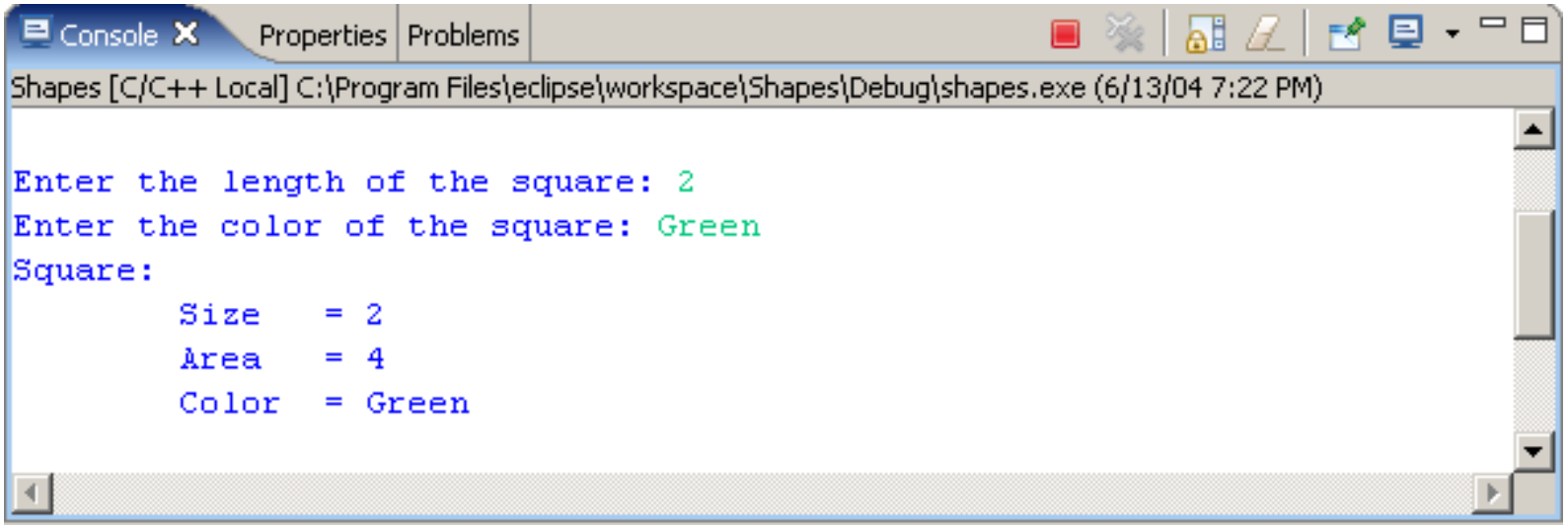
- Syntax highlighting
- Content/code assist
- Integrated debugging features

The most common way to invoke the C/C++ editor is to open a file from the Navigator or the C/C++ Project views using pop-up menus or by clicking the file (single or double-click depending on the user preferences).

The C/C++ editor does not contain a toolbar itself, but relies on the use of the main toolbar, edit menu, search menu and key binding actions.

Console view

This view shows the output of the execution of your program and enables you to enter input for the program.



The console shows three different kinds of text, each in a different color:

- Standard output
- Standard error
- Standard input

You can choose the different colors for these kinds of text on the preferences pages (**Window > Preferences > Debug > Console**).

Console View Context Menu

When you right-click in the **Console** view (or when you press **Shift+F10** when the focus is on the **Console** view), you see the following options:

Edit options: Cut, Copy, Paste, Select All
: These options perform the standard edit operations. Which options are available depends on where the focus is in the **Console** view. For example, you cannot paste text into the program output, but you can paste text to the bottom of the file.

Find/Replace
: Opens a **Find/Replace** dialog that operates only on the text in the **Console** view.

Go to Line
: Opens a dialog that moves the focus to the line you specify. The dialog also indicates the total number of lines in the console buffer.

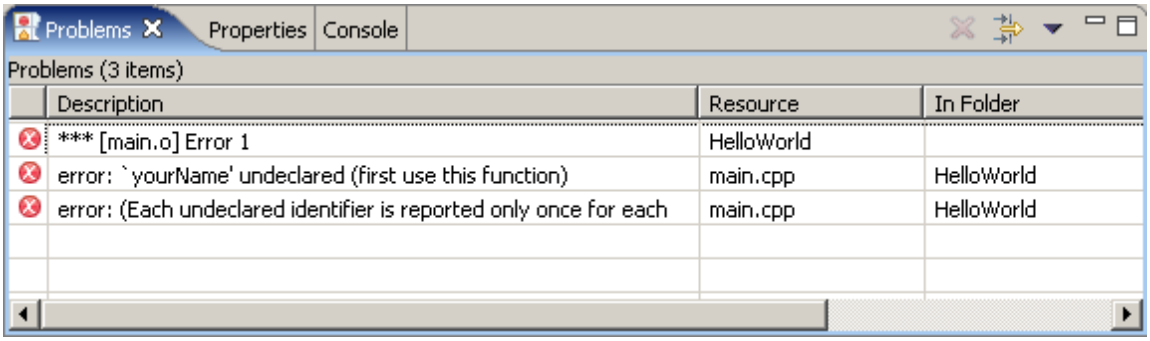
Terminate
: Terminates the process that is currently associated with the console.

Console View Toolbar

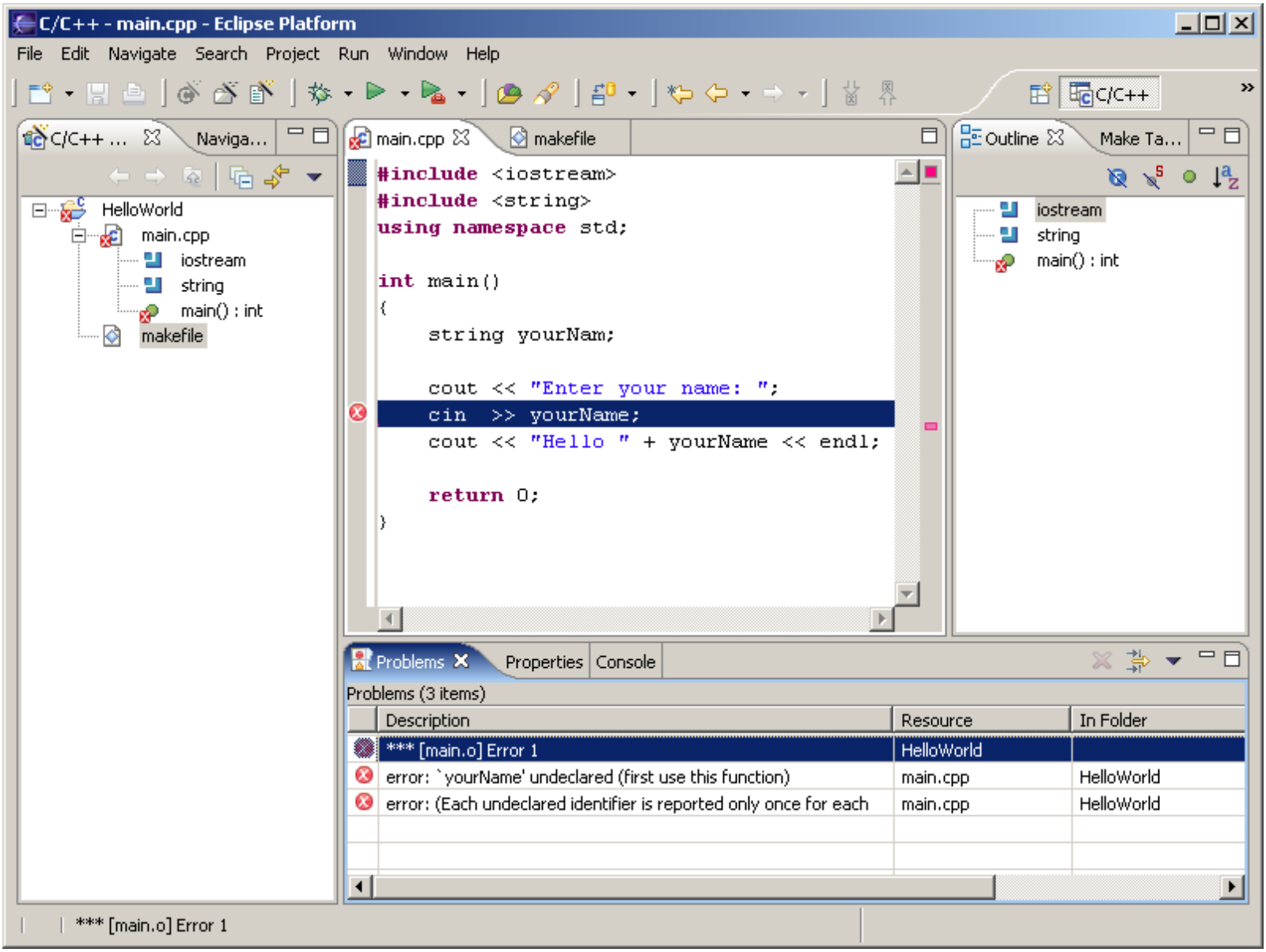
Icon	Command	Description
	Terminate	Terminates the process that is currently associated with the console.
	Remove all Terminated Launches	Removes all terminated launches that are associated with the console.
	Scroll Lock	Toggles the Scroll Lock.
	Clear Console	Clears the console.
	Pin Console	Forces the Console view to remain on top of other views in the window area.
	Display Selected Console	If multiple consoles are open, you can select the one to display from a list.
	MinimizeConsole	Minimizes the Console view.
	Maximize Console	Maximizes the Console view.

Problems view

If you encounter any errors during a build they will be displayed in the **Problems** view.



Errors are passed up from your C++ compiler. The **Problems** view lists the error, filename and folder. If you select an error the associated file will open in the **C/C++ Editor** view and the cursor will display the line where the error was encountered.



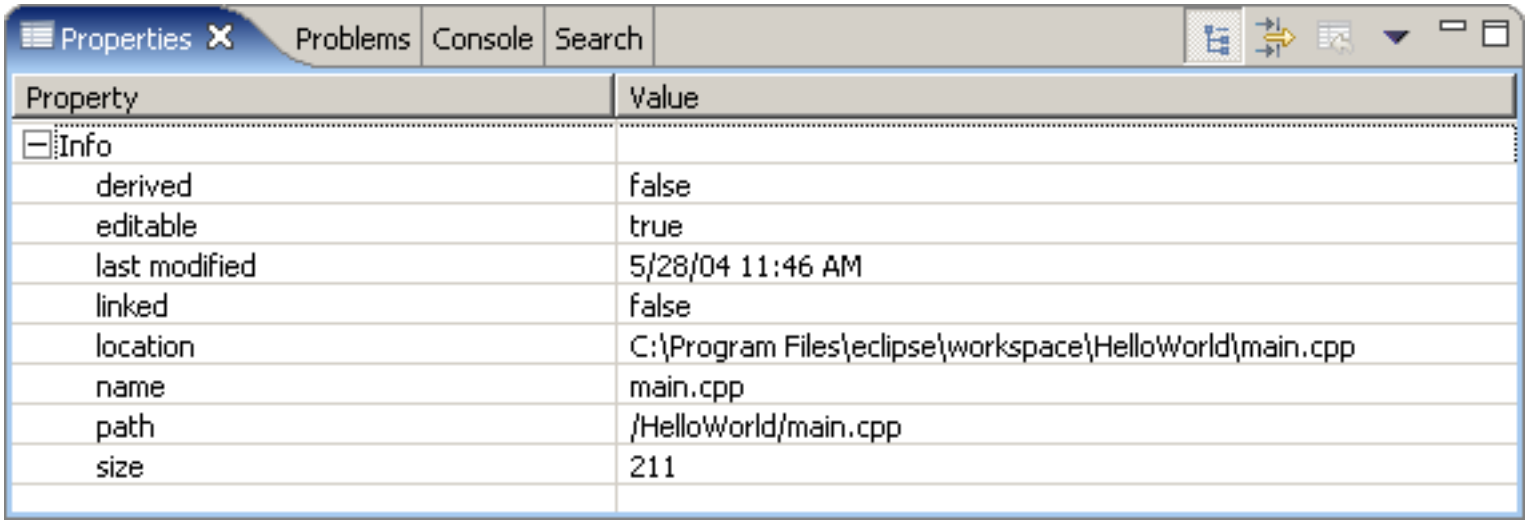
Problems view Toolbar

Icon	Command	Description

	Delete	Deletes the selected error from the Problems view.
	Filters...	Lauches the Filter dialog box to filter the errors in the problems view.
	Menu	Select the Sort or Filters help navigate through errors in the Problems view.
	Minimize	Minimizes the Problems view.
	Maximize	Maximizes the Problems view.

Properties view

The properties view displays property names and values for a selected item such as a resource.



Toolbar buttons allow you to toggle to display properties by category or to filter advanced properties. Another toolbar button allows you to restore the selected property to its default value.

To see more detailed information about a resource than the Properties view gives you, right-click the resource name in the Navigator view and select Properties from the pop-up menu.

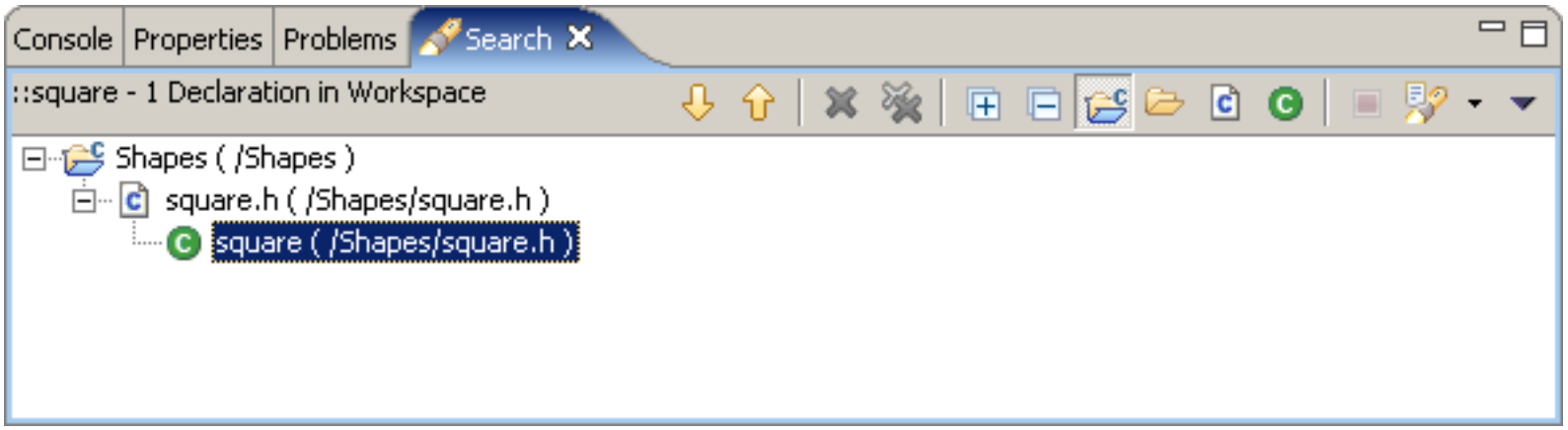
Toolbar

Icon	Name	Description
	Show Categories	This command, when selected, lists the properties in sorted categories.
	Show Advanced Properties	This command, when selected, shows advanced properties. By default, advanced properties are filtered.
	Restore Default	This command returns any modified properties to their default values.

	**Menu**	Click the black upside-down triangle icon to open a menu of items specific to the Properties view. **Show Categories** See the toolbar item description above. **Show Advanced Properties** See the toolbar item description above.
	MinimizeConsole	Minimizes the Console view.
	**Maximize Console**	Maximizes the Console view.

Search view

Any matches are reported in the **Search** view.



When you have completed a search and have results in the **Search** view, you can put the focus on that view and get more options on the **Search** menu.

A C/C++ search can also be conducted via the context menu of selected resources and elements in the following views:

- C/C++ Projects
- Make Targets
- Navigator
- Outline view
- Search result view

The search context menu is also available in the C/C++ editor. The search is only performed if the currently selected text can be resolved to a C/C++ element.

The type of the selected C/C++ element defines which search context menus are available. The C/C++ editor does not constrain the list of available C/C++ searches based on the selection.

Search view Toolbar

Icon	Command	Description
	Next	Navigates to the next search result.
	Previous	Navigates to the previous search result.
	Remove the Selected Matches	Removes user selected matches from the search console.
	Remove All Matches	Clears the search console.

	Terminate	Terminates the current search.
	Show Previous Searches	Shows the list of previously run searches which can be reselected.
	Menu	Lists two selectable view layouts for search results: Flat and Heirarchical.
	Minimize Console	Minimizes the Console view.
	Maximize Console	Maximizes the Console view.

Related concepts

Coding aids

C/C++ search

Related tasks

Searching for C/C++ elements

Customizing the C/C++ editor

Related reference

C/C++ editor preferences

Search action

Search dialog

Debug views

This section describes debug views.

Registers view

The Registers view of the Debug perspective lists information about the registers in a selected stack frame. Values that have changed are highlighted in the Registers view when your program stops. The options described below are available when you right-click a register value.

Change Register Value
Specifies a register value.

Format
Displays register values, in Natural, Decimal, or Hexadecimal number systems.

Show Type Names
Displays the type, (such as **int**) beside each register value.

Auto-Refresh
Updates the registers list each time execution stops.

Refresh
Updates the registers list.

Display as Array
Displays a selected register as an array of a specified length and index. This option is only applicable to pointers.

Cast To Type
Casts a register value to a different type.

Restore to Original Type
Restores a register value to its original type.

Related concepts
Debug information

Related tasks
Working with registers

Related reference
Views

Memory view

The Memory view of the Debug perspective lets you inspect and change your process memory. The view consists of four tabs that let you inspect multiple sections of memory. The options described below are available when you right-click a memory value.

Auto-Refresh
Updates the memory address list each time execution stops.

Refresh
Updates the memory address list.

Clear
Clears the selected memory address.

Format
Specifies the number system in which to display memory values (Hexadecimal, Signed Decimal, or Unsigned Decimal).

Memory Unit Size
Specifies the memory address size (1 byte, 2 bytes, 4 bytes, 8 bytes).

Number of Columns
Specifies the numbers of columns displayed in the Registers view (1 column, 2 columns, 4 columns, 8 columns, 16 columns).

Show ASCII
Displays the selected value as ASCII.

Detail panel
Displays the raw output from GDB for the selected variable.

Related concepts

Debug information

Related tasks

Working with memory

Related reference

Views

Memory view preferences

You can change the appearance of the memory view.

Text Color
Changes the color of the text.
Background Color
Changes the background color.
Address Color
Changes the color of the memory address text.
Changed Value color
Changes the color of values.
Font
Changes the font.
Padding Character
Specifies the character to use for padded values.
Auto-refresh by default
Updates the view automatically.
Show ASCII by default
Displays values in ASCII.

Related concepts
Debug information

Related tasks
Working with memory

Related reference
Views

Shared Libraries view

The Shared Libraries view of the Debug perspective lets you view information about the shared libraries loaded in the current debug session.

Load Symbols
Loads the symbols of the selected library. This option does not affect the libraries with loaded symbols.

Load Symbols For All
Loads the symbols of the libraries used in the current session.

Auto-Refresh
Updates the shared library information each time execution stops.

Refresh
Updates the shared library information as required.

Show Full Paths
Displays the full path of libraries.

Related concepts

Debug information

Related tasks

Debugging

Related reference

Views

Signals view

The Signals view of the Debug perspective lets you view the signals defined on the selected debug target and how the debugger handles each one.

Name
Displays the name of the signal.

Pass
Where "yes" is displayed, the debugger lets your program see the signal. Your program can handle the signal, or else it may terminate if the signal is fatal and not handled.

Suspend
Where "yes" is displayed, the debugger suspends your program when this signal is handled.

Description
Displays a description of the signal.

Related concepts

Debug information

Related tasks

Debugging

Related reference

Views

Debug view

The **Debug** view shows the target information in a tree hierarchy shown below with a sample of the possible icons:

Session item	Description	Icons
Launch instance	Launch configuration name and launch type	
Debugger instance	Debugger name and state	
Thread instance	Thread number and state	
Stack frame instance	Stack frame number, function, file name, and file line number	

The number beside the thread label is a reference counter, not a thread identification number (TID).

The CDT displays stack frames as child elements. It displays the reason for the suspension beside the target, (such as end of stepping range, breakpoint hit, and signal received). When a program exits, the exit code is displayed.

In addition to controlling the individual stepping of your programs, you can also control the debug session. You can perform actions such as terminating the session and stopping the program by using the debug launch controls available from Debug view.

Action	Icon	Description
Terminate		Ends the selected process
Disconnect		Detaches the debugger from the selected process (useful for debugging attached processes)
Remove All Terminated		Clears all the killed processes in Debug view
Terminate and Remove		Ends the selected process and remove it from Debug view
Relaunch		Restarts the process
Terminate All		Ends all active processes in Debug view

Related concepts

Debug overview

Debug information

Related tasks

Debugging

Related reference

Run and Debug dialog box

Debug preferences

The Debug view of the Debug perspective displays information about the variables in the currently selected stack frame.

Show full paths
: Displays the full path of resources

Default variable format
: Specifies the number system in which to display variables (Natural, Hexadecimal or Decimal).

Default expression format
: Specifies the number system in which to display expressions (Natural, Hexadecimal or Decimal).

Default register format
: Specifies the number system in which to display registers (Natural, Hexadecimal or Decimal).

Automatically switch to disassembly mode
: Automatically examines your program in disassembly mode as it steps into functions for which you do not have source code, such as printf().

Related concepts

Debug information

Related tasks

Debugging

Related reference

Views

C/C++ Icons

The table below lists the icons displayed in the C/C++ perspective.

Icon	Description
	C or C++ file
	Class
	Code template
	Macro Definition
	Enum
	Enumerator
	Variable
	Field private
	Field protected
	Field public
	Include
	Makefile
	Method private
	Method protected
	Method public
	Namespace
	Struct
	Typedef
	Union
	Function

Related concepts

Project file views
Outline view

Related tasks

Displaying C/C++ file components in the C/C++ Projects view

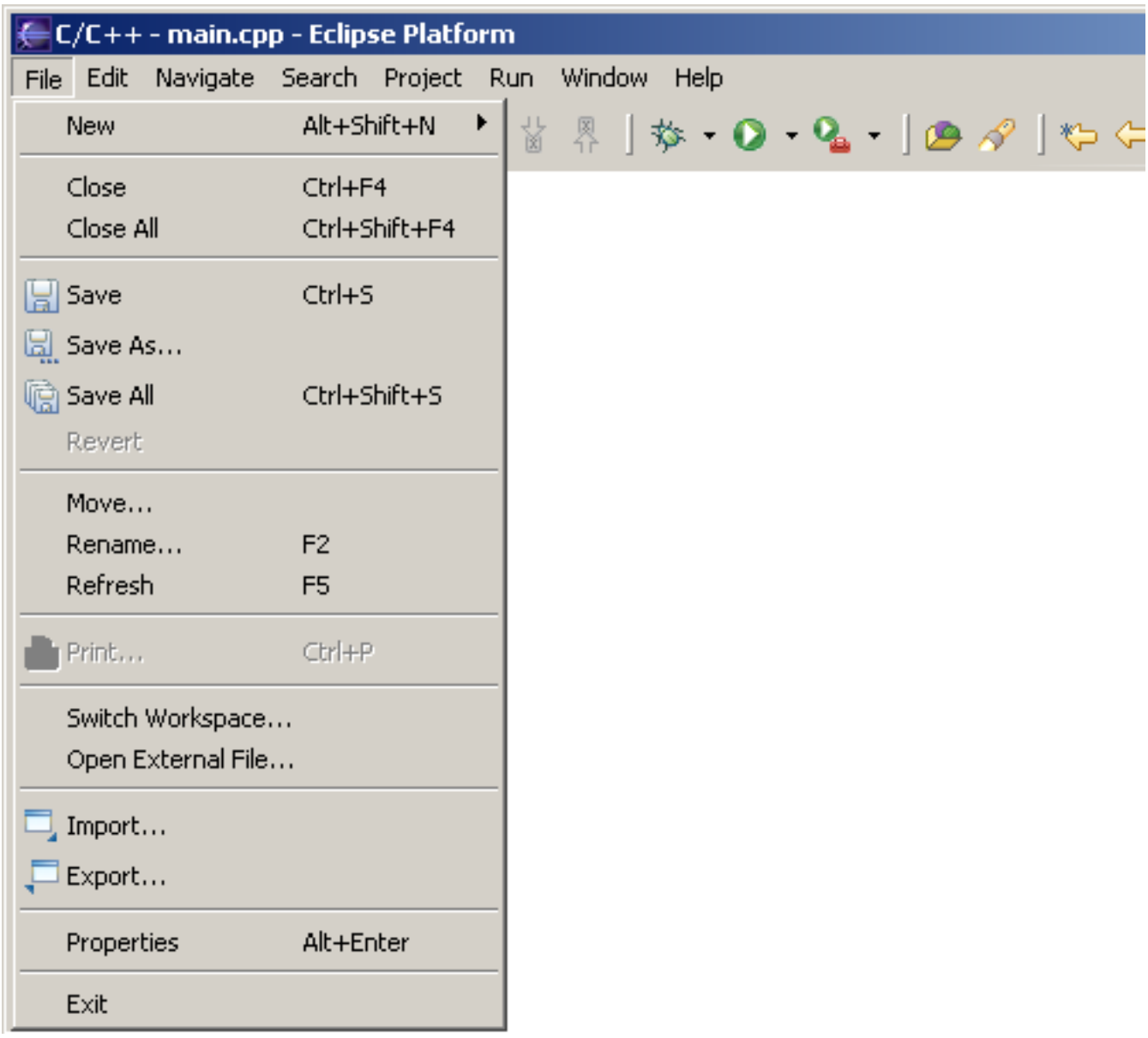
Related reference

Views

C/C++ Menubar

This section describes the menubar options available from the C/C++ perspective.

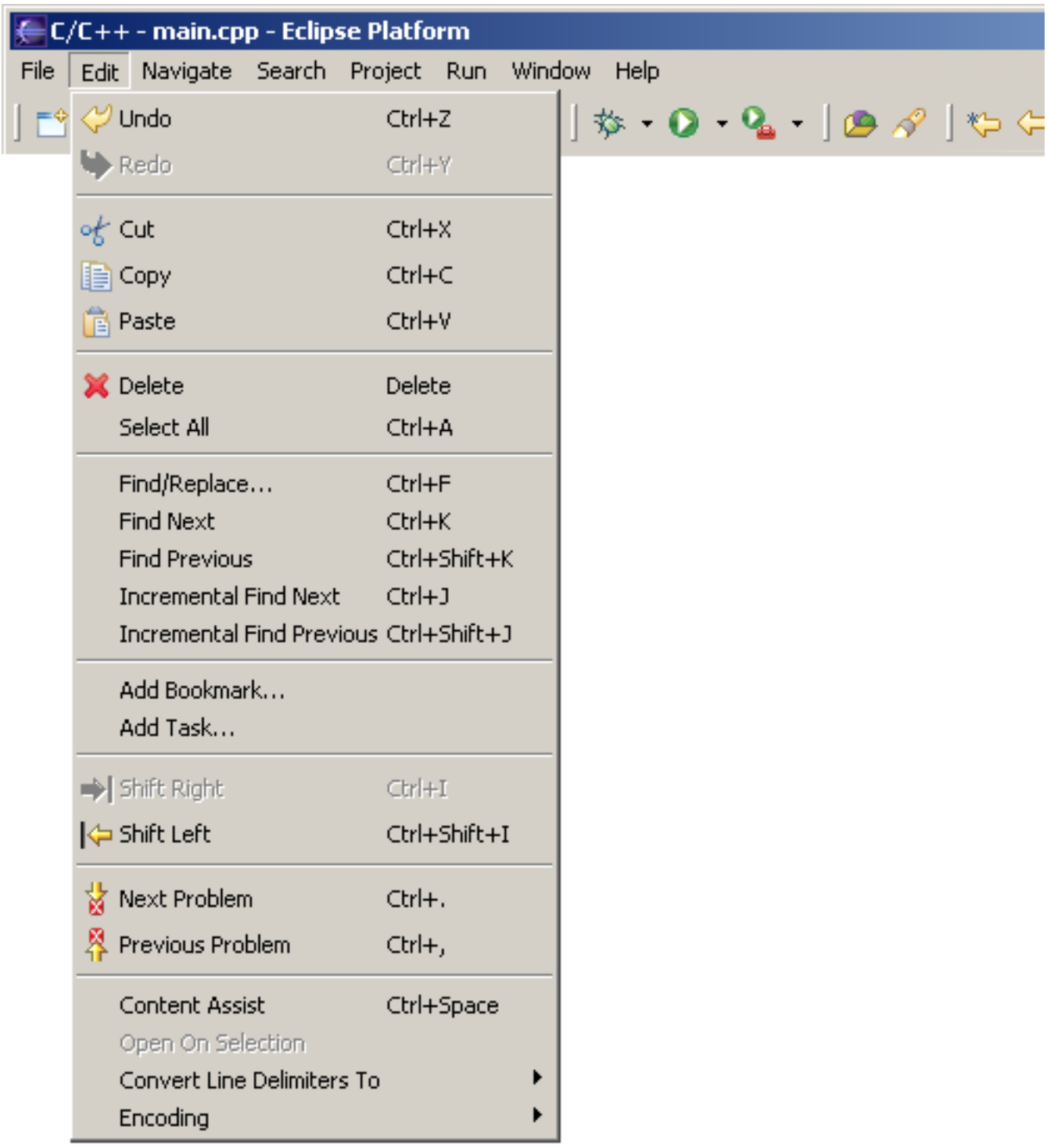
File Menu actions



Name	Function	Keyboard Shortcut
New	Create a new project, folder, or file.	Alt+Shift+N
Close	Close the current editor. If the editor contains unsaved data, a save request dialog is shown.	Ctrl+F4
Close All	Close all editors. If editors contains unsaved data, a save request dialog will be shown.	Ctrl+Shift+F4
Save	Save the content of the current editor. Disabled if the editor does not contain unsaved changes.	Ctrl+S
Save As	Save the content of the current editor under a new name.	

Save All	Save the content of the current editor. Disabled if no editor contains unsaved changes.	Ctrl+Shift+S
Revert	Revert the content of the current editor back to the content of the saved file. Disabled if the editor does not contain unsaved changes.	
Move	Move a resource.	
Rename	Renames a resource.	F2
Refresh	Refreshes the content of the selected element with the local file system. When launched from no specific selection, this command refreshes all projects.	F5
Print	Prints the content of the current editor. Enabled when an editor has the focus.	Ctrl+P
Switch workspace...	Relaunches Eclipse with a new workspace.	
Open External File...	Opens a file in the editor view.	
Import	Opens the **Import** dialog and shows all import wizards.	
Export	Opens the **Export** dialog and shows all export wizards.	
Properties	Opens the property pages of the select elements.	Alt+Enter
Exit	Exit Eclipse	

Edit Menu actions



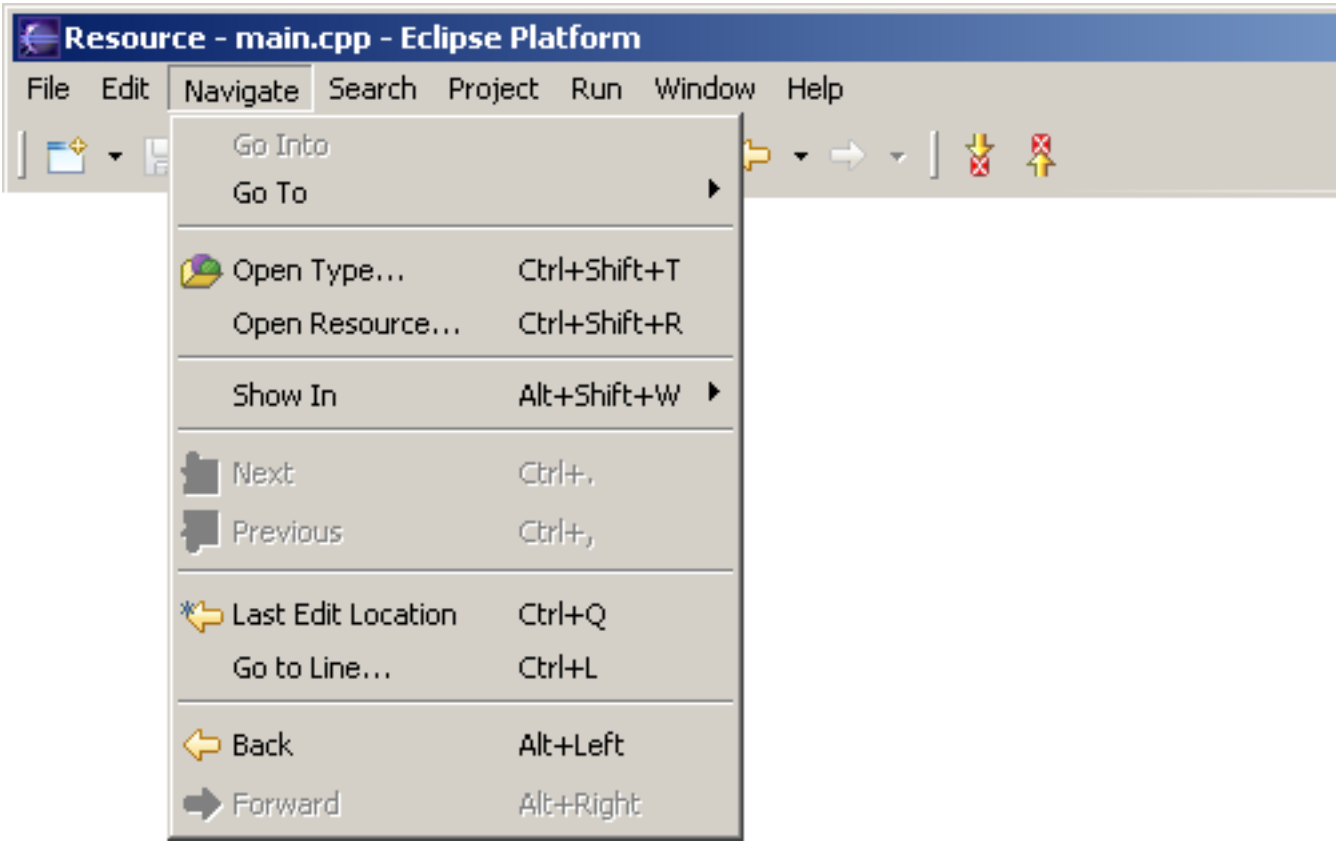
Name	Function	Keyboard Shortcut
Undo	Revert the last change in the editor	Ctrl+Z
Redo	Revert an undone change	Ctrl+Y

Cut	Copies the currently selected text or element to the clipboard and removes the element. On elements, the remove is not performed before the clipboard is pasted.	Ctrl+X
Copy	Copies the currently selected text or elements to the clipboard	Ctrl+C
Paste	Paste the current content as text to the editor, or as a sibling or child element to the a currently selected element.	Ctrl+V
Delete	Delete the current text or element selection.	Delete
Select All	Select all the editor content..	Ctrl+A
Find / Replace	Open the Find / Replace dialog. Editor only.	Ctrl+F
Find Next	Finds the next occurrence of the currently selected text. Editor only.	Ctrl+K
Find Previous	Finds the previous occurrence of the currently selected text. Editor only.	Ctrl+Shift+K
Incremental Find Next	Starts the incremental find mode. After invocation, enter the search text as instructed in the status bar. Editor only.	Ctrl+J
Incremental Find Previous	Starts the incremental find mode. After invocation, enter the search text as instructed in the status bar. Editor only.	Ctrl+Shift+J
Add Bookmark...	Add a bookmark to the current text selection or selected element.	
Add Task...	Add a user defined task to the current text selection or selected element.	Alt+Enter
Shift Right	Shifts text Right. Editor only	Ctrl+I
Shift Left	Shifts text Left. Editor only	Ctrl+Shift+I
Next Problem	Moves to the next problem encountered.	Ctrl+.
Previous Problem	Navigates to the previous problem encountered.	Ctrl+,
Content Assist	Opens a context assist dialog at the current cursor position to bring up Java code assist proposals and templates. See the Templates preference page for available templates **Window > Preferences > C/C++ > Code Templates** and go to the Editor preference page **Window > Preferences > C/C++ > C/C++ Editor > Content Assist** for configuring the behaviour of content assist.	Ctrl+Space

Convert Line Delimiters to	Toggles line delimeters to • CRLF (Windows) • LF (UNIX, MacOS X) • CR (Classic MacOS)	
Encoding	Toggles the encoding of the currently shown text content.	

Note: Other **Edit** options are used with the JDT. Refer to the *Java Development User Guide* for details.

Navigate Menu actions



Name	Function	Keyboard Shortcut
Go Into	Sets the view input to the currently selected element.	
Go To	• **Back**: This command displays the hierarchy that was displayed immediately prior to the current display. For example, if you Go Into a resource, then the Back command in the resulting display returns the view to the same hierarchy from which you activated the Go Into command. This command is similar to the Back button in an HTML browser. • **Forward**: This command displays the hierarchy that was displayed immediately after the current display. For example, if you've just selected the Back command, then selecting the Forward command in the resulting display returns the view to the same hierarchy from which you activated the Back command. This command is similar to the Forward button in an HTML browser. • **Up one level**: This command displays the hierarchy of the parent of the current highest-	

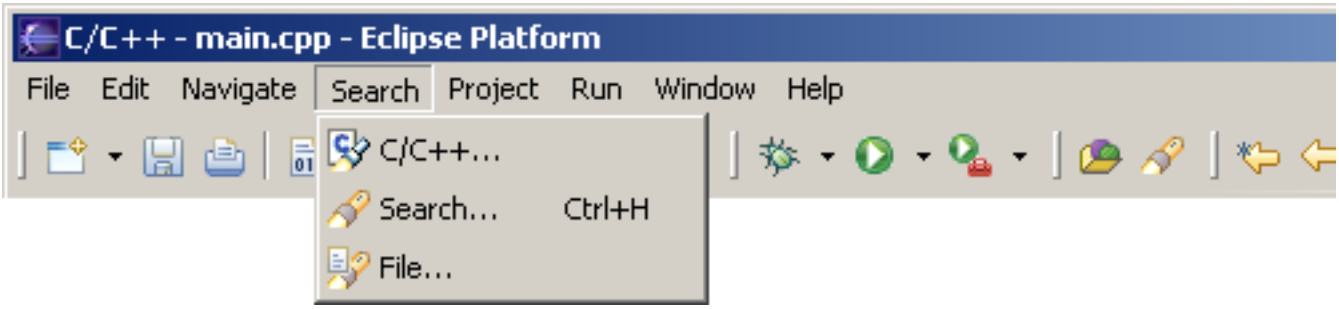
	level resource. • **Resource**: This command allows you to navigate quickly to a resource. For more information see the links to related tasks below.	
Open Type	Brings up the Open Type selection dialog to open a type in the editor. The Open Type selection dialog shows all types existing in the workspace.	Ctrl+Shift+T
Open Resource	This command displays a dialog that lets you select any resource in the workspace to open it in an editor.	Ctrl+Shift+R
Show In	This sub-menu is used to find and select the currently selected resource in another view. If an editor is active, these commands are used to select the resource currently being edited in another view.	Ctrl+Shift+W
Next	The "next" definition is based on where the focus is. For example, during a search this entry becomes **Next Match**.	Ctrl+.
Previous	The "previous" definition is based on where the focus is. For example, during a search this entry becomes **Previous Match**.	Ctrl+,
Go to Last Edit Location	Moves the cursor to the line that contains the last edit. Editor only.	Ctrl+Q
Go to Line	Open a dialog where you can specify the line number to which to move the cursor. Editor only.	Ctrl+L
Back	Moves the focus to the previous file. Editor only.	Ctrl+Q
Forward	Returns the focus from the previous file. Editor only.	Ctrl+Q

Note: Other **Navigate** options are used with the JDT. Refer to the *Java Development User Guide* for details.

Search Menu actions

Search menu commands open the search dialog. There are specialized tabs on the general **Search** dialog to help you search for:

- Files, or for text in files
- Elements in C/C++ files
- Text in the online help
- Plug-ins.



Name	Function	Keyboard Shortcut
C/C++...	Opens the search dialog on the C/C++ search page	
Search...	Opens the search dialog for your current editor	Ctrl + H
File...	Opens the search dialog on the File search page	

Related concepts

Coding aids

C/C++ search

Related tasks

Searching for C/C++ elements

Customizing the C/C++ editor

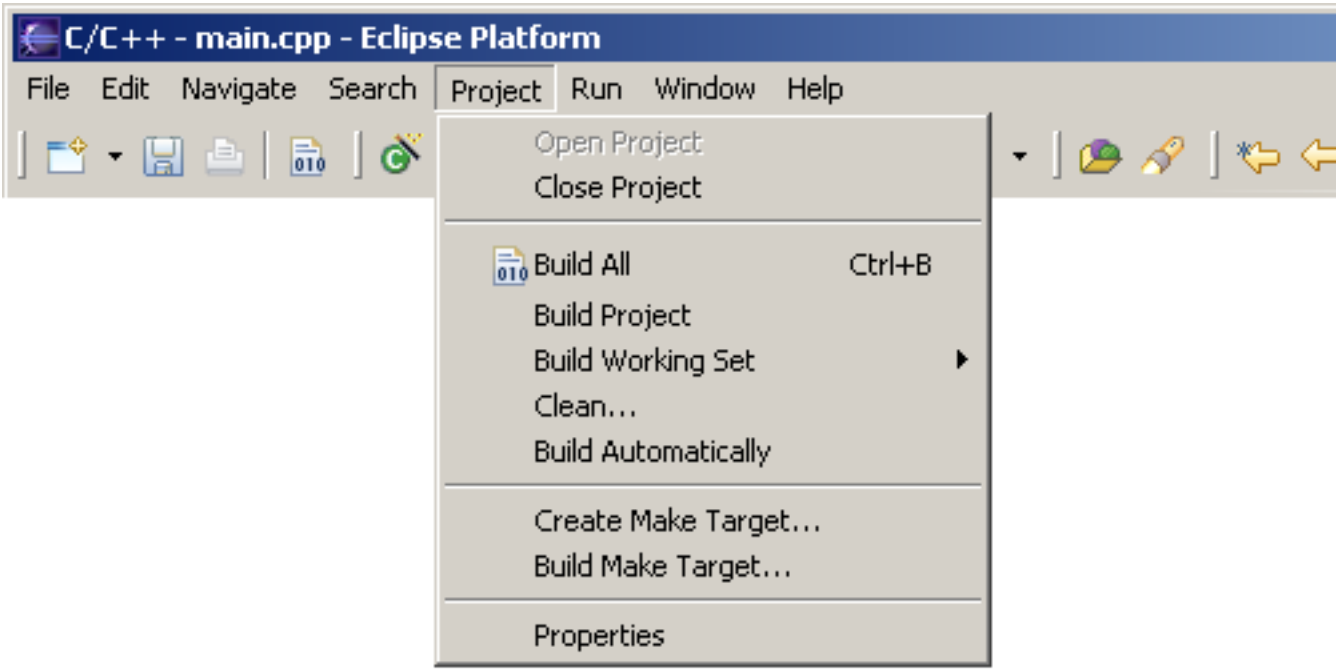
Related reference

C/C++ editor preferences

Search dialog

Search view

Project Menu actions



Name	Function	Keyboard Shortcut
Open Project	Shows a dialog that can be used to select a closed project and open it.	
Close Project	Closes the currently selected projects.	
Build All	Builds all projects in the workspace. This is a full build; all files are built.	Ctrl+B
Build Project	Builds the currently selected project. This is a full build; all files in the project are built.	
Clean	Invokes the make clean defined in the makefile	
Build Automatically	When checked, the CDT will perform a build whenever a file in a project is saved. You should turn this feature off for very large projects.	
Create Make Target	Creates a target in the **Make Targets** view. Standard Make only	
Build Make Target	Builds a specific make target defined in your makefile such as **make clean** or **make install**. Standard Make only.	
Properties	Displays the **Properties** dialog. From that dialog you can display the properties of resources in **Info**, **External Tools Builders**, **C/C++ Build** (managed only) **File Types**, **Indexer options**, **C/C++ Make Project** (standard only), **C/C++ Project Paths** (standard only), **Include Paths and Symbols** (standard only), and **Project References**.	

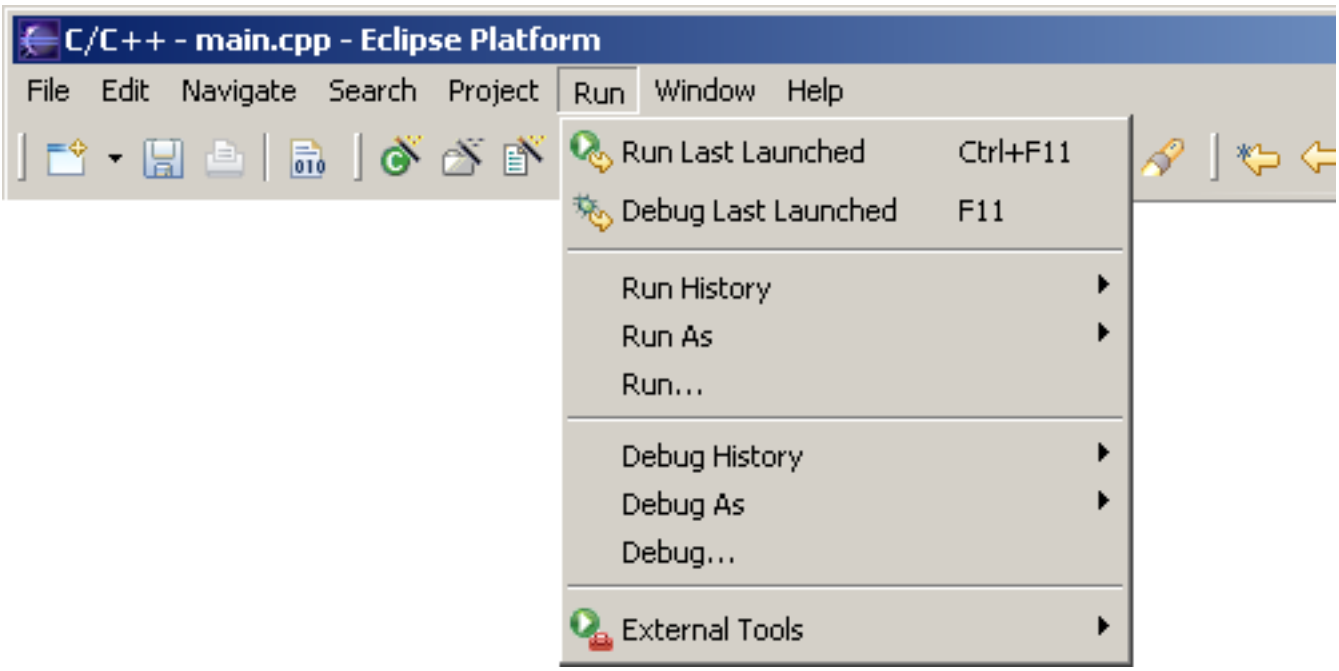
Related concepts

C/C++ compiler

Related tasks

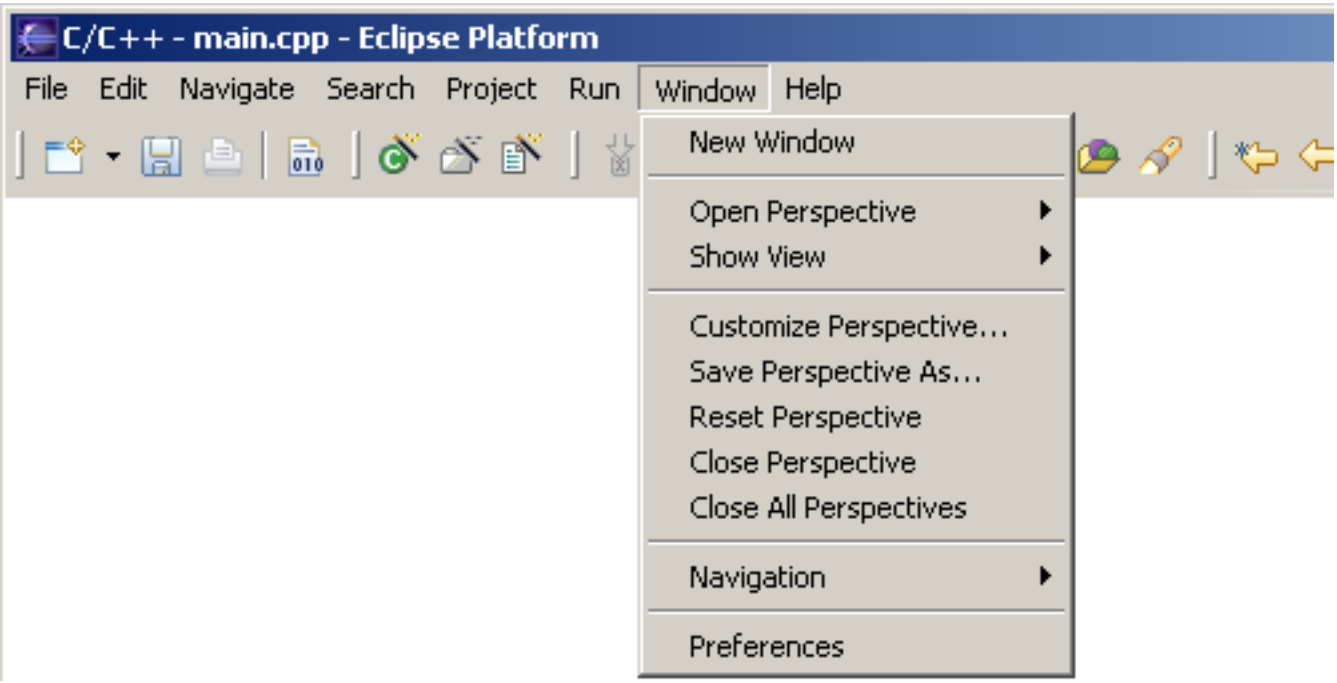
Building a program

Run Menu actions



Name	Function	Keyboard Shortcut
Run Last Launched	This command allows you to quickly repeat the most recent launch in run mode.	Ctrl+F11
Debug Last Launched	This command allows you to quickly repeat the most recent launch in debug mode.	F11
Run History	Presents a sub menu of the recent history of launch configurations launched in run mode.	
Run As	Presents a sub menu of registered run launch shortcuts. Launch shortcuts provide support for workbench or active editor selection sensitive launching.	
Run...	This command realizes the launch configuration dialog to manage run mode launch configurations.	
Debug History	Presents a sub menu of the recent history of launch configurations launched in debug mode.	
Debug As	Presents a sub menu of registered debug launch shortcuts. Launch shortcuts provide support for workbench or active editor selection sensitive launching.	
Debug...	This command realizes the launch configuration dialog to manage debug mode launch configurations.	
External Tools	Presents a sub menu of links to external run configuration dialogs to manage run mode launch configurations.	

Window Menu actions



Name	Function	Keyboard Shortcut
New Window	Window menu commands:	
Open Perspective	This command opens a new perspective in this Workbench window. This preference can be changed in the **Window > Preferences > Workbench > Perspectives** page. All of the perspectives that are open within the Workbench window are shown on the shortcut bar. The perspectives you will likely want to open are listed first.This list is dependent on the current perspective. From the **Other...** submenu you can open any perspective.	
Show View	This command displays the selected view in the current perspective. You can configure how views are opened in the **Window > Preferences > Workbench > Perspectives page**. Views you are likely to want to open are listed first.This list is dependent on the current perspective. From the **Other...** submenu you can open any view. The views are sorted into categories in the Show View dialog.	

Customize Perspective	Each perspective includes a predefined set of actions that are accessible from the menu bar and Workbench toolbar. Related actions are grouped into action sets. This command allows you to customize the current perspective by showing or hiding various action sets. The first three (**File > New**, **Window > Open Perspective**, **Window > Show View**) control which actions appear as top level items in their respective menus. The last category (**Other**) controls which action sets are visible in the perspective.	
Save Perspective As	This command allows you to save the current perspective, creating your own custom perspective. You can open more perspectives of this type using the **Window > Open Perspective > Other** menu item once you have saved a perspective.	
Reset Perspective	This command changes the layout of the current perspective to its original configuration.	
Close Perspective	This command closes the active perspective.	
Close All Perspectives	This command closes all open perspectives in the Workbench window.	
Navigation	This submenu contains shortcut keys for navigating between the views, perspectives, and editors in the Workbench window. • **Show System Menu**: Shows the menu that is used for resizing, closing or pinning the current view or editor. • **Show View Menu**: Shows the drop down menu that is available in the toolbar of the active view. • **Maximize Active View or Editor**: Maximizes the current view or editor to fill the workbench. • **Activate Editor**: Makes the current editor active. • **Next Editor**: Activates the next open editor in the list of most recently used editors. • **Previous Editor**: Activates the previous open editor in the list of most recently used editors. • **Next View**: Activates the next open view in the list of most recently used views. • **Previous View**: Activates the previous open view in the list of most recently used views. • **Next Perspective**: Activates the next open perspective in the list of most recently used perspectives. • **Previous Perspective**: Activates the previous open perspective in the list of most recently used perspectives.	

Preferences	This command allows you to indicate your preferences for using the Workbench. There are a wide variety of preferences for configuring the appearance of the Workbench and its views, and for customizing the behavior of all tools that are installed in the Workbench. See the C/C++ Page Preference Window section for more details on the CDT preferences.	

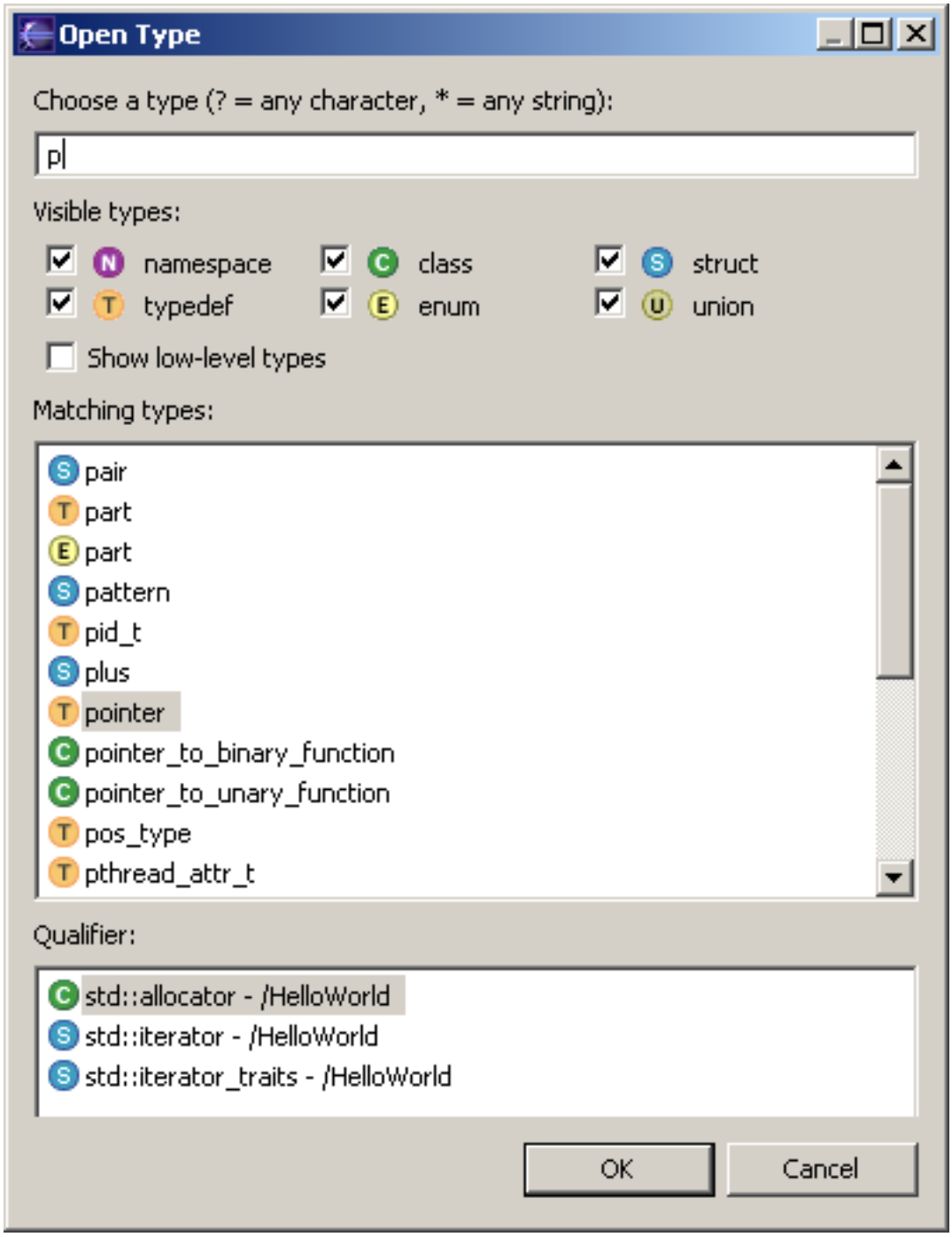
C/C++ Toolbar

C/C++ Toolbar icons

Icon	Command	Description
	Create New	Create a new project, folder, or file.
	Save	Save the content of the current editor. Disabled if the editor does not contain unsaved changes.
	Print	Prints the content of the current editor.
	Create C++ Class	Creates a C++ class within the current project.
	Create Folder	Creates a folder within the current project.
	Create File	Creates a file within the current project.
	Debug	Launches the Debug dialog box.
	Run	Launches the Run dialog box
	External Tools	Launches the External Tools dialog box
	Open Type	Brings up the Open Type selection dialog to open a type in the editor. The Open Type selection dialog shows all types existing in the workspace.
	Search	Launches the C/C++ Search dialog box
	Synchronize	Synchronizes selected project to CVS repository.
	Go to Last Edit Location	Returns editor view to the last line edited, if the file that was last edited was closed it will be re-opened.
	Back	Navigates back through open files.
	Forward	Navigates forward through open files.
	Go to Next Problem	Navigates to the next problem marked during the last build attempt.
	Go to Previous Problem	Navigates to the previous problem marked during the last build attempt.

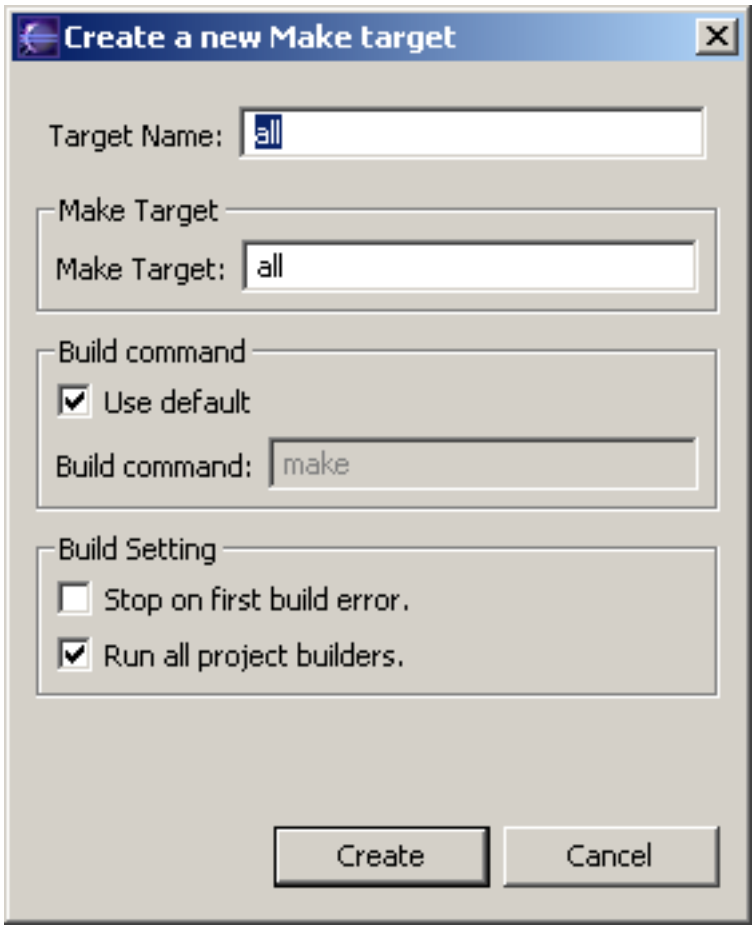
C/C++ Open Type

Use Open Type to open up the declaration of C/C++ classes, structures, unions, typedefs, enumerations and namespaces.



Create a Make Target

You can define build settings when you create a Make Target.



Target Name
: Name of the Make Target.

Make Target
: The reference to the make section in your makefile.

Use default
: Select this checkbox to use the default make command. Clear the check box to specify a new make command.

Build command
: If you clear the **Use default** checkbox type a new make command in this field.

Stop on first build error
: Stops the build when an error occurs.

Run all project builders
: Runs additional project builders such as discovery.

Related concepts

Build overview

Makefile

Related tasks

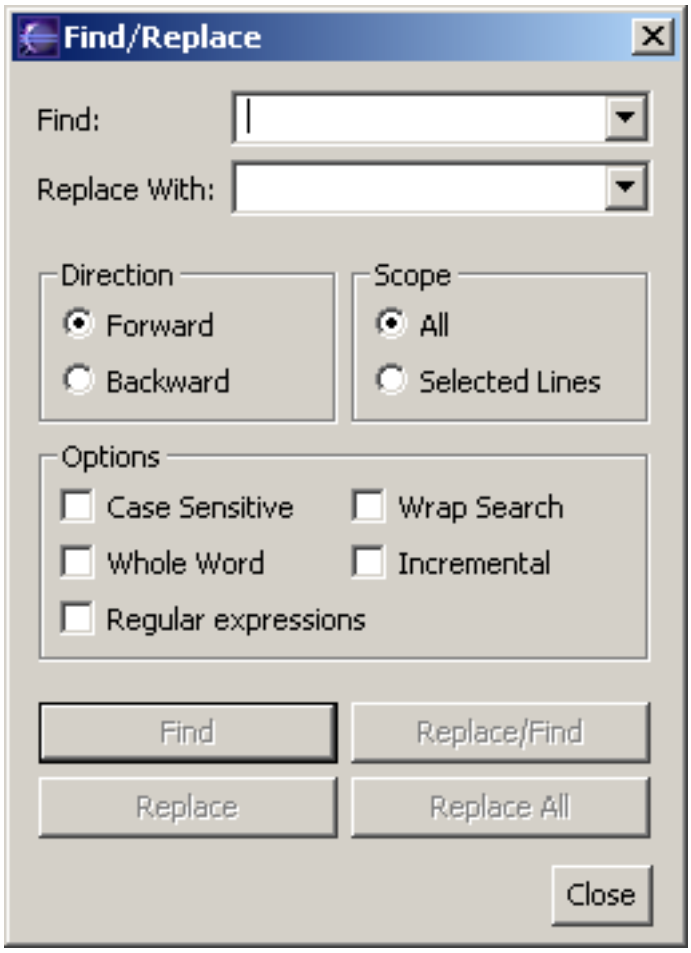
Creating a makefile

Creating a Make Target

Defining build settings

C/C++ Find/Replace

Ctrl+F (or **Edit > Find/Replace**) displays the **Find/Replace** dialog. Here you can specify text to search for and optionally text with which to replace it.



You can specify:

- The direction (forward or back from the current cursor location)
- The scope (**All** for the whole file or **Selected Lines** to search only within the highlighted area)
- Whether the search is **Case Sensitive** or **Whole Word**. You can also specify whether the search wraps at the end of the file.

If you close the **Find/Replace** dialog with text in the **Find** field, you can use **Ctrl+K** (or **Edit > Find Next**) or **Ctrl+Shift+K** (or **Edit > Find Previous**) to go to the next occurrence of that text. The directions for "Next" and "Previous" are not affected by the **Direction** setting in the **Find/Replace** dialog.

Note: Wildcards are not currently supported for searches.

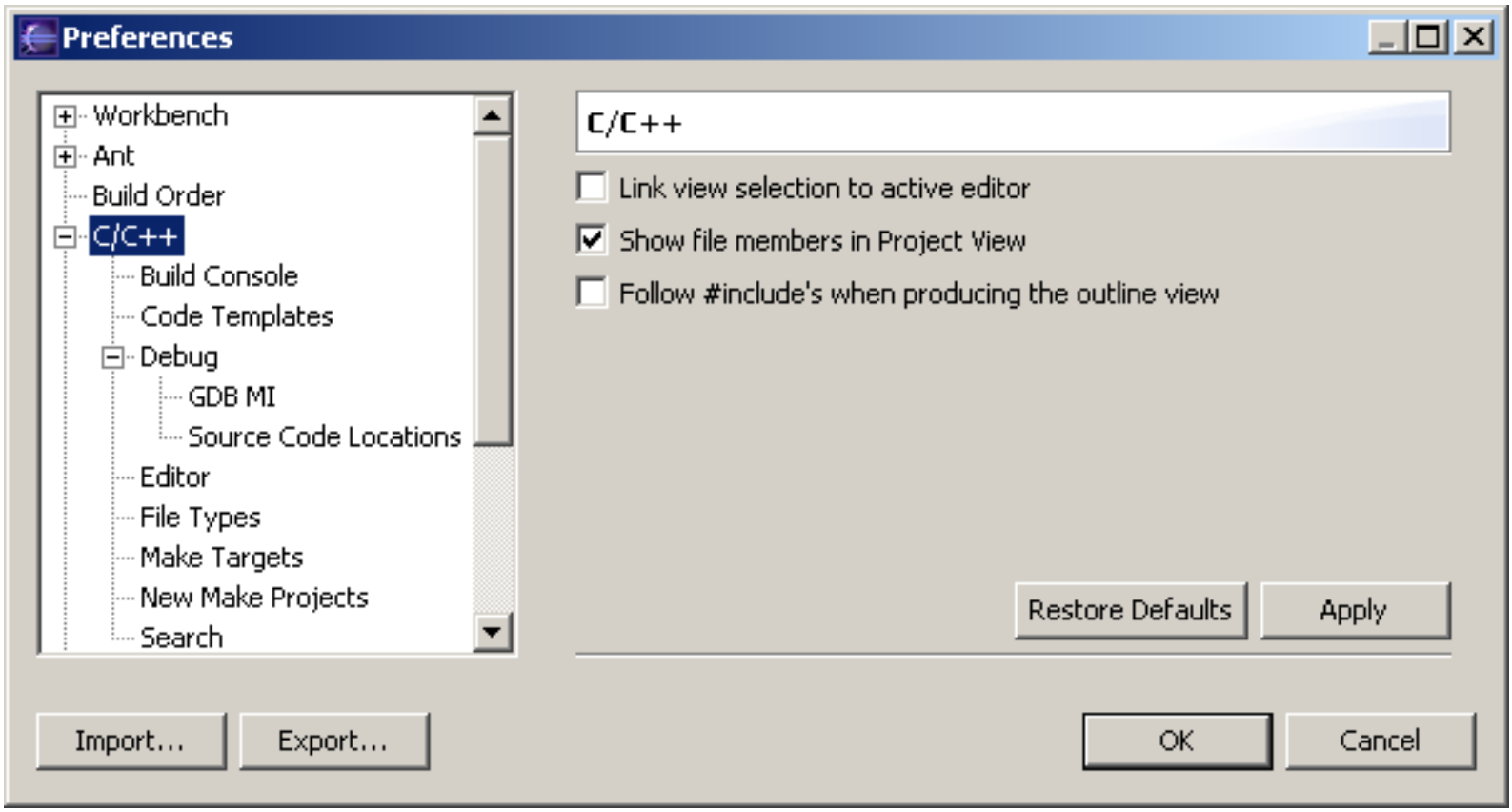
Incremental Find

You can also choose **Incremental Find** from the **Find/Replace** dialog. With this option selected, each letter you type in the **Find** field causes the editor focus to move to the first complete occurrence of the text you are typing. You can also use incremental find by pressing **Ctrl+J** (**Edit > Incremental Find**). In this case, the text you type appears in the Status Line at the bottom of the Eclipse window.

Note: The settings in the **Find/Replace** dialog do not affect the operation of incremental find in the Status Line.

C/C++ preferences

The C/C++ Preference dialog box allows you to make changes to your C/C++ environment.



Link view selection to active editor
Select this checkbox to open an item selected in the Projects view, in the Editor view.
Show file members in Project View
Select this checkbox to browse your C and C++ file members in the Projects view.
Follow #include's when producing the outline view.
Select this checkbox to follow all defined #includes when you produce the Outline view.
Note: This is not recommended for large projects or large files.

C/C++ Submenu Items

Build Console
Preferences for customize the appearance of the Build Console view.
Editor
Preferences for customizing the C/C++ editor.
Code Templates
Manipulate any of the common code templates that are predefined within the CDT.
Debug
Preferences for customizing the Debugger.
GDB MI

Preferences for customizing the GDB MI.

Source Code Locations

Modify, add or remove source code locations

Editor

Set preferences for the C/C++ editor

File Types

Define which file extensions are linked to specific languages

Make Targets

Set preferences for make target build settings

New Make Projects

Set preferences for customizing Make Builder settings

Search

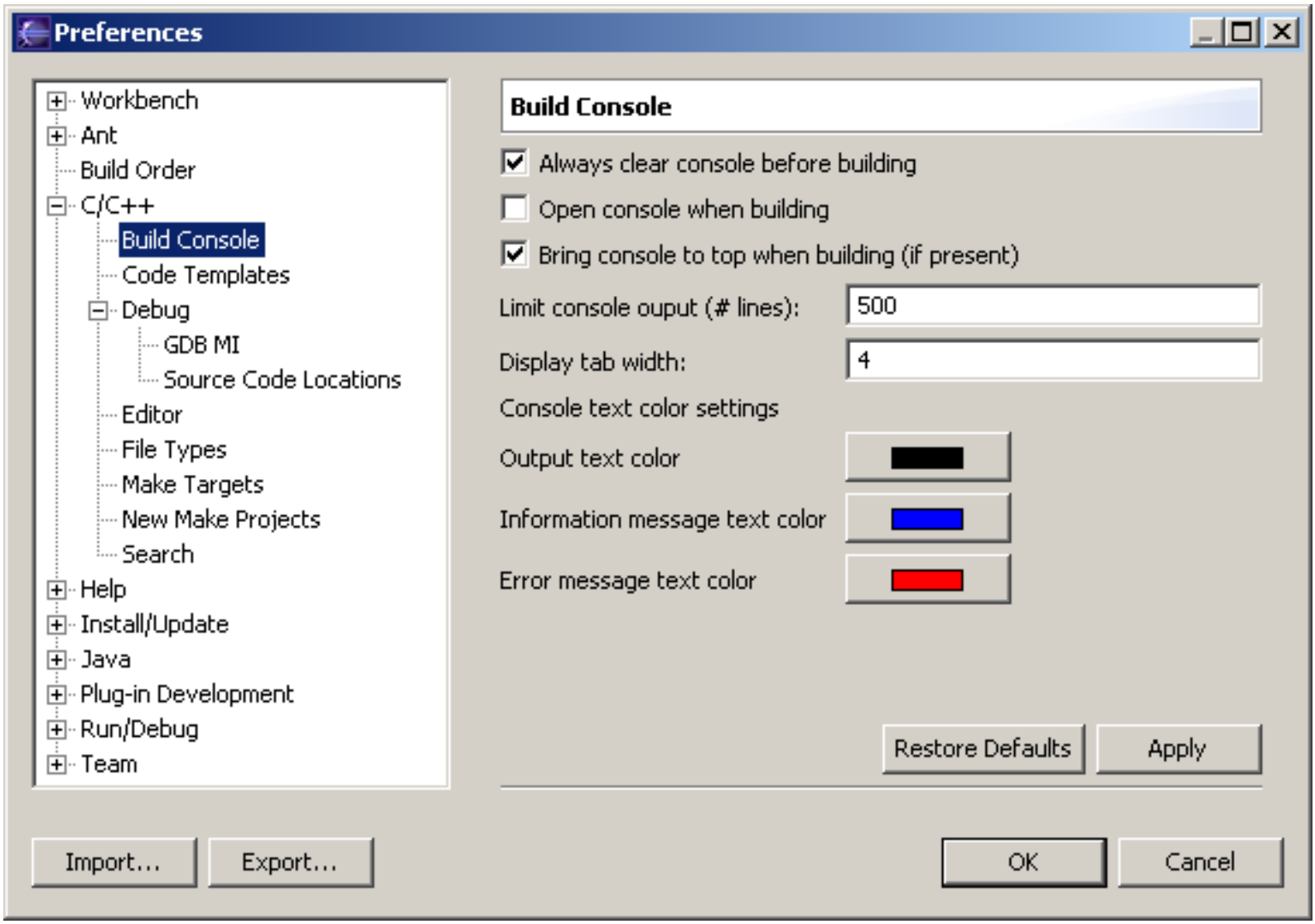
Set preferences for customizing the Search configuration

Related reference

Views

Build Console preferences

You can customize the appearance of the Build Console view.



Always clear console before building
Select this checkbox to clear the console whenever you perform a build.

Open console when building
Select this checkbox to open the console build when you perform a build.

Bring console to top when building (if present)
Select this checkbox to bring the build console to the front when you perform a build.

Limit console output (#lines)
You specify the maximum number of lines that appear in the build console view.

Display tab width
You can specify the number of spaces for a tab.

Output text color
You can customize the color of text in the build console.

Information message text color
You can customize the color of information text messages in the build console.

Error message text color

You can customize the color of error text messages in the build console.

Related concepts

Build overview

Related tasks

Defining Build Settings

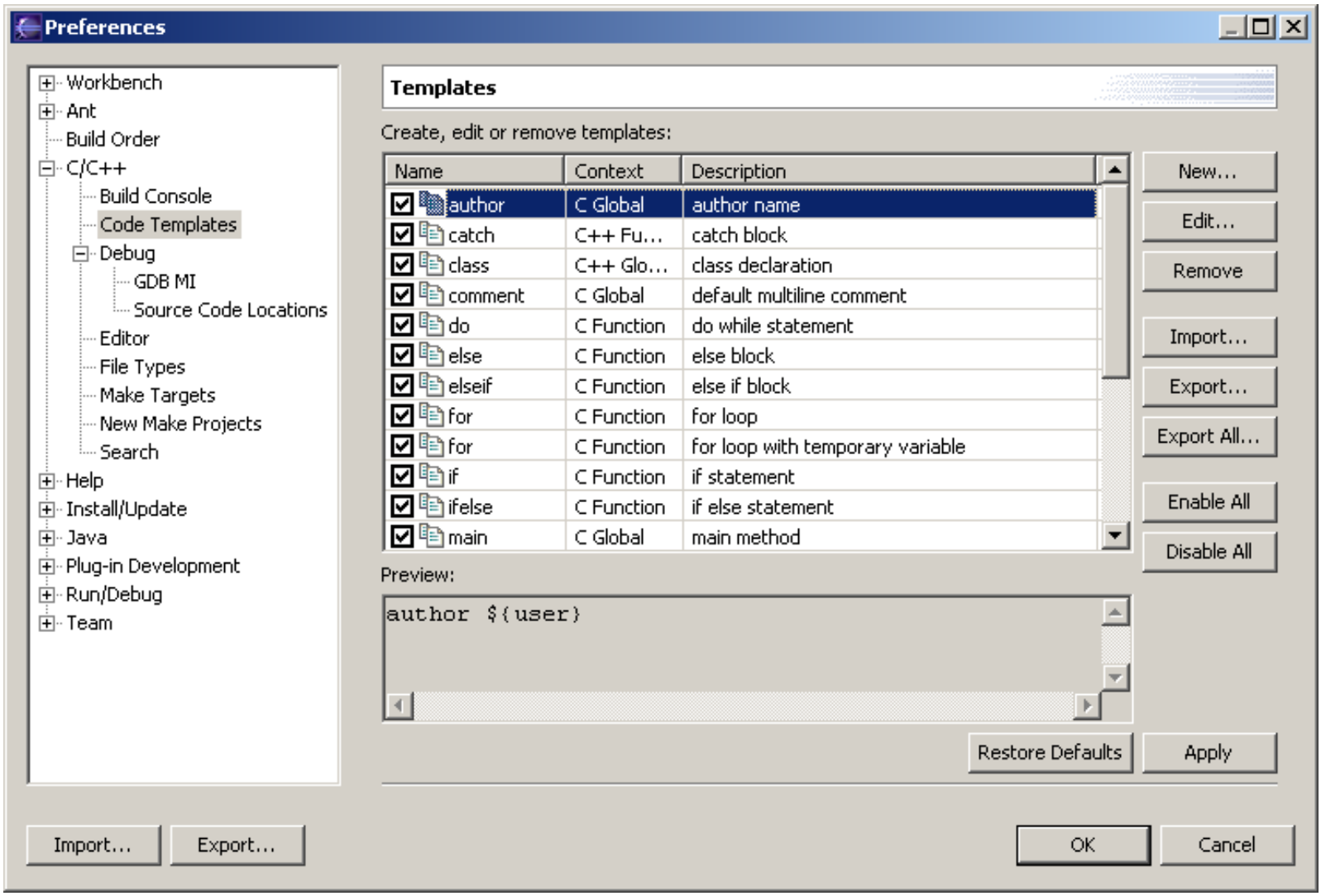
Building

Related reference

Views

Code Templates preferences

You can manipulate any of the common code templates that are predefined within the CDT. To modify, delete, export, import, or create your own templates click **Window > Preferences > C/C++ > Code Templates**.



New
Creates a new code template.

Edit
Edits the code template that is selected in the list.

Remove
Removes the selected code templates from the list.

Import
Imports a code template.

Export
Exports the selected code templates.

Export All
Exports all templates in the list.

Enable All
Makes all templates available when you invoke the Content Assist feature.

Disable All
Makes all templates unavailable when you invoke the Content Assist feature.

Related concepts

Coding aids

Related tasks

Customizing the C/C++ editor

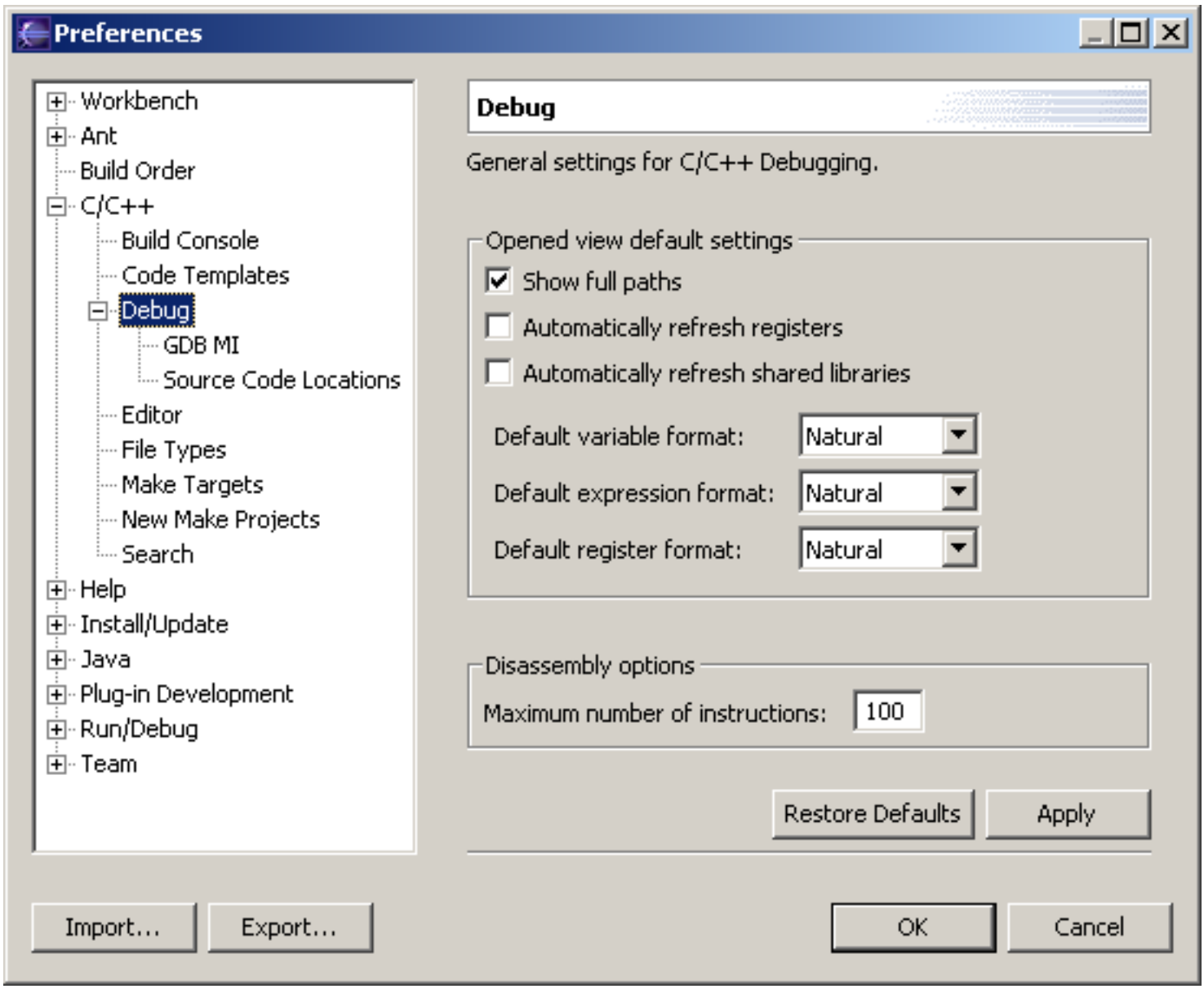
Working with Content Assist

Related reference

C/C++ editor preferences

Debug page, Preferences window

You can manipulate any of the common predefined debug settings by clicking **Window > Preferences > C/C++ > Debug**.



Show Full Paths
: Select this checkbox to show the full path to files and directories.

Automatically refresh registers
: Select this checkbox to refresh registers before debugging.

Automatically refresh shared libraries
: Select this checkbox to refresh links to shared libraries before debugging.

Default variable format:
: Select the default variable format from either Natural, Decimal or Hexidecimal.

Default expression format:
: Select the default expression format from either Natural, Decimal or Hexidecimal.

Default register format:

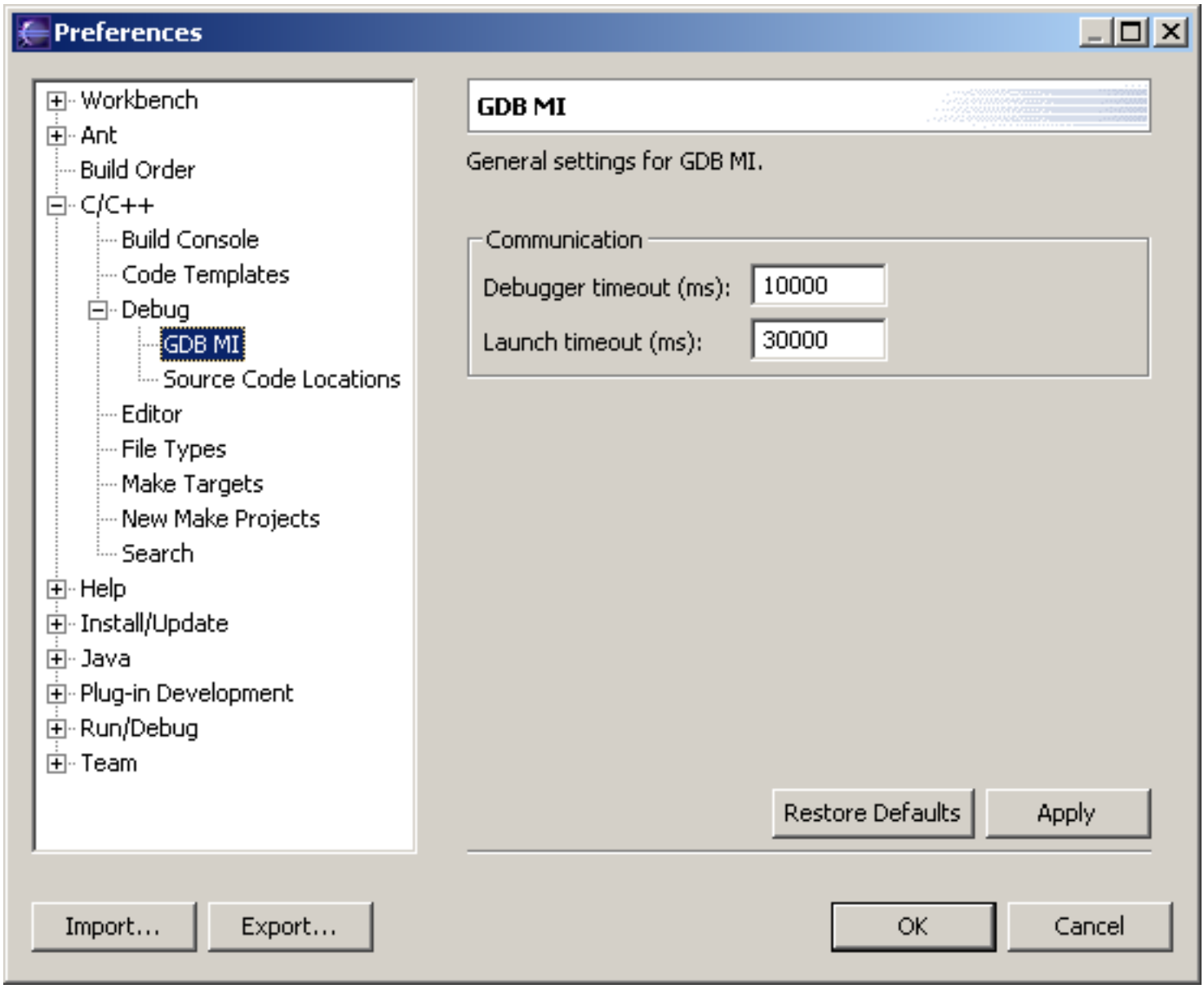
Select the default register format from either Natural, Decimal or Hexidecimal.

Maximum number of instructions:

Enter the maximum number of disassembly instruction in the field provided.

Debug GDB MI page, Preferences window

You can manipulate certain GDB timeout settings **Window > Preferences > C/C++ > Debug > GDB MI**.



Debugger timeout (ms)

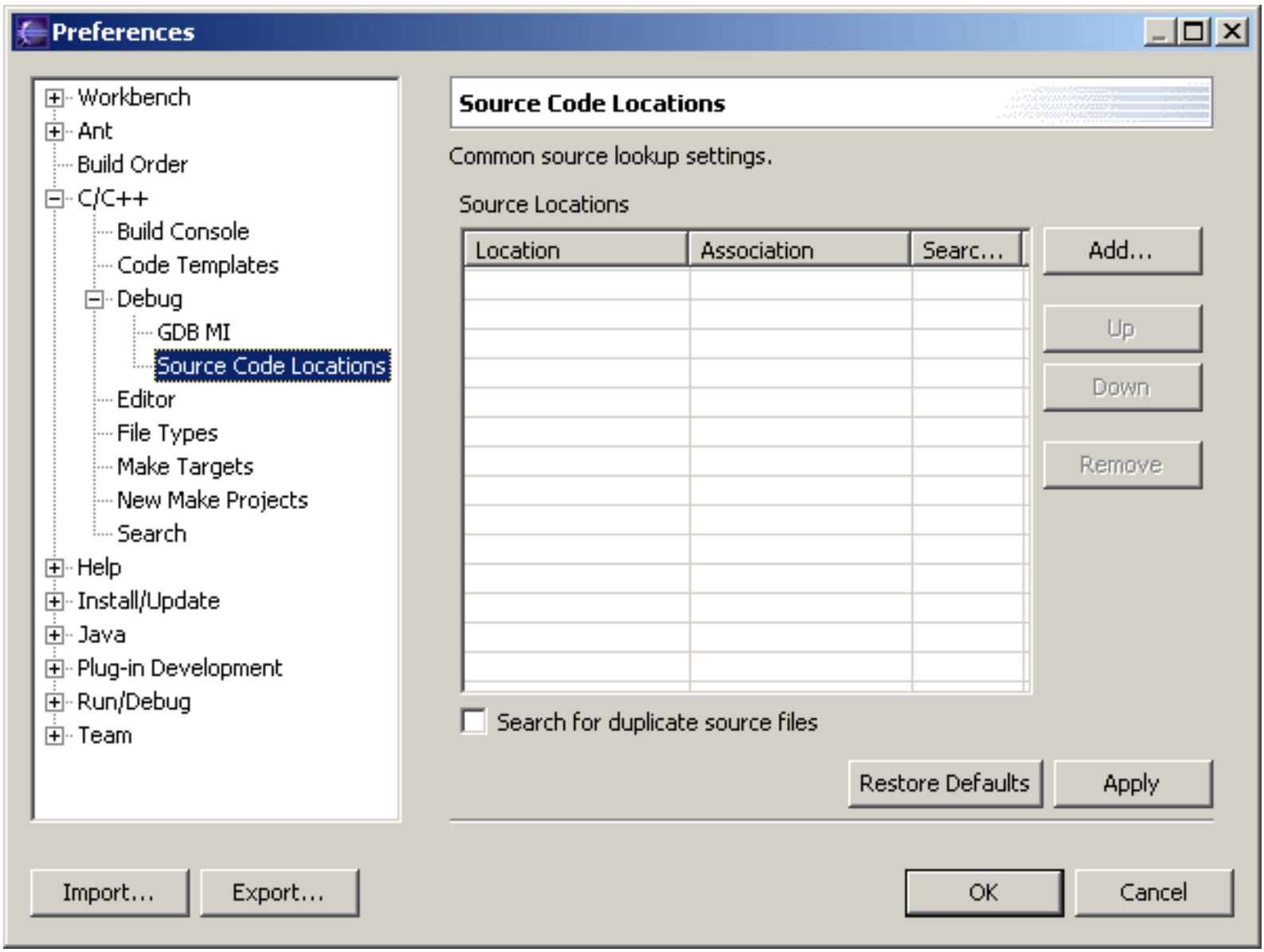
Sets the timout value for the debugger.

Launch timeout (ms)

Sets the Launch timeout for a debug session.

Source Code Locations page, Preferences window

You can add or remove source code locations by clicking **Window > Preferences > C/C++ > Debug > Source Code Locations**.



Add
: Insert a new source code location.

Up
: Move the currently selected source code location higher in the list.

Down
: Move the currently selected source code location lower in the list.

Remove
: Remove the currently selected souce code location.

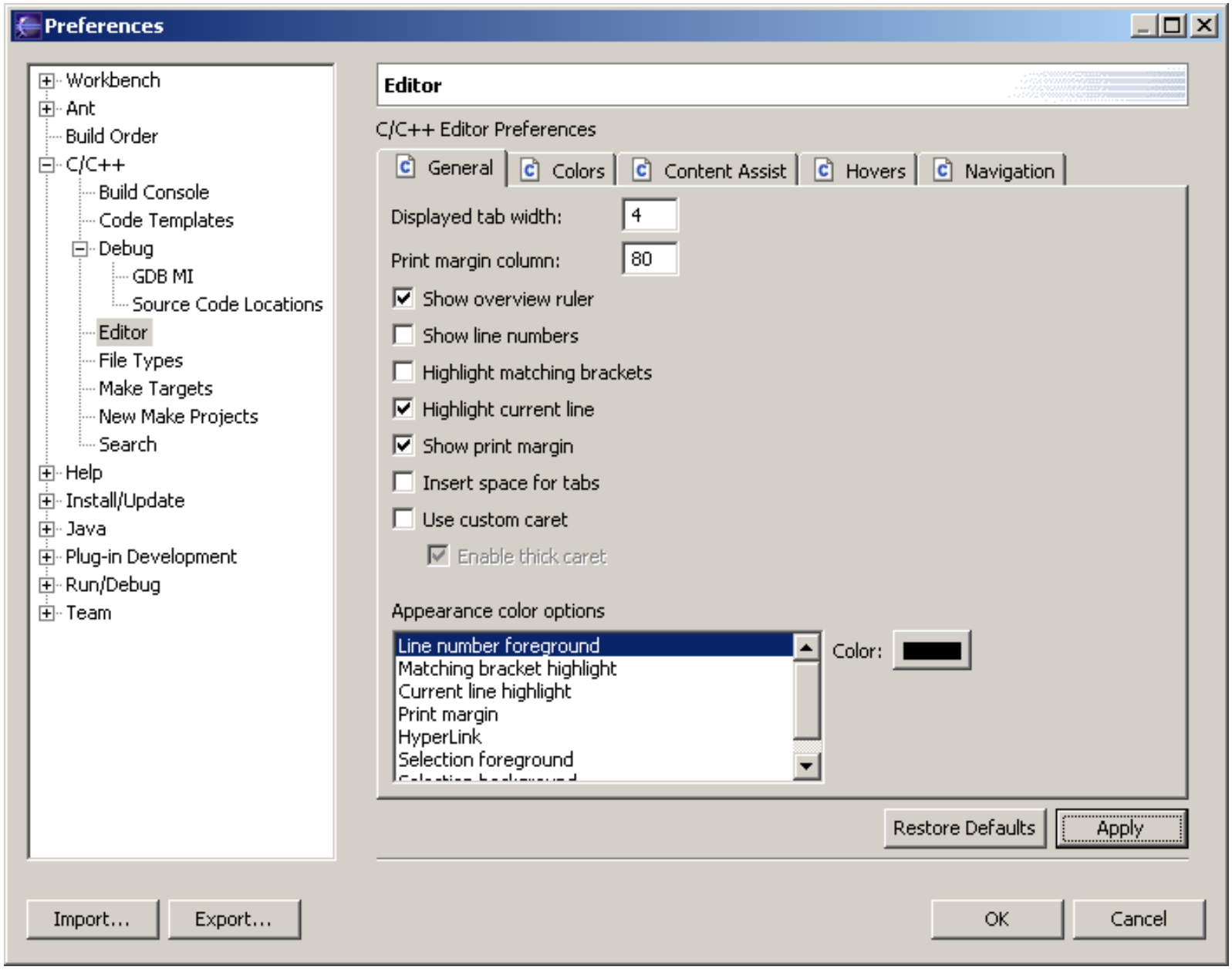
Search for duplicate source files
: Searches the source locations for duplicate entries.

C/C++ editor preferences

This section describes how to set preferences for the C/C++ editor.

General preferences

You can customize the appearance of the C/C++ editor on the General page of the Preferences window.



Displayed tab width
Specifies the width of the tab as a number of characters. For example a value of **4**, means that the tab width is 4-characters wide.

Print margin column
Specifies the width of the print margin as a number of characters. For example a value of **80** means that the print area is 80-characters wide.

Show overview ruler
Displays the vertical ruler in the editor view.

Show line numbers
Displays line numbers in the left margin.

Highlight matching brackets
When the cursor is beside a bracket, the matching bracket is highlighted.

Highlight current line
Highlights the line that was last selected.

Show print margin
Displays the print margin in the editor.
Insert space for tabs
Inserts a space instead of a tab character when you press `Tab`.
Use custom caret
Select a custom caret (vertical bar icon showing cursor position).
Enable thick caret
Displays a thicker caret (vertical bar icon showing cursor position).
Appearance color options
Lists the items for which you can specify a color.
Color
Changes the color of the item that is selected in the list.

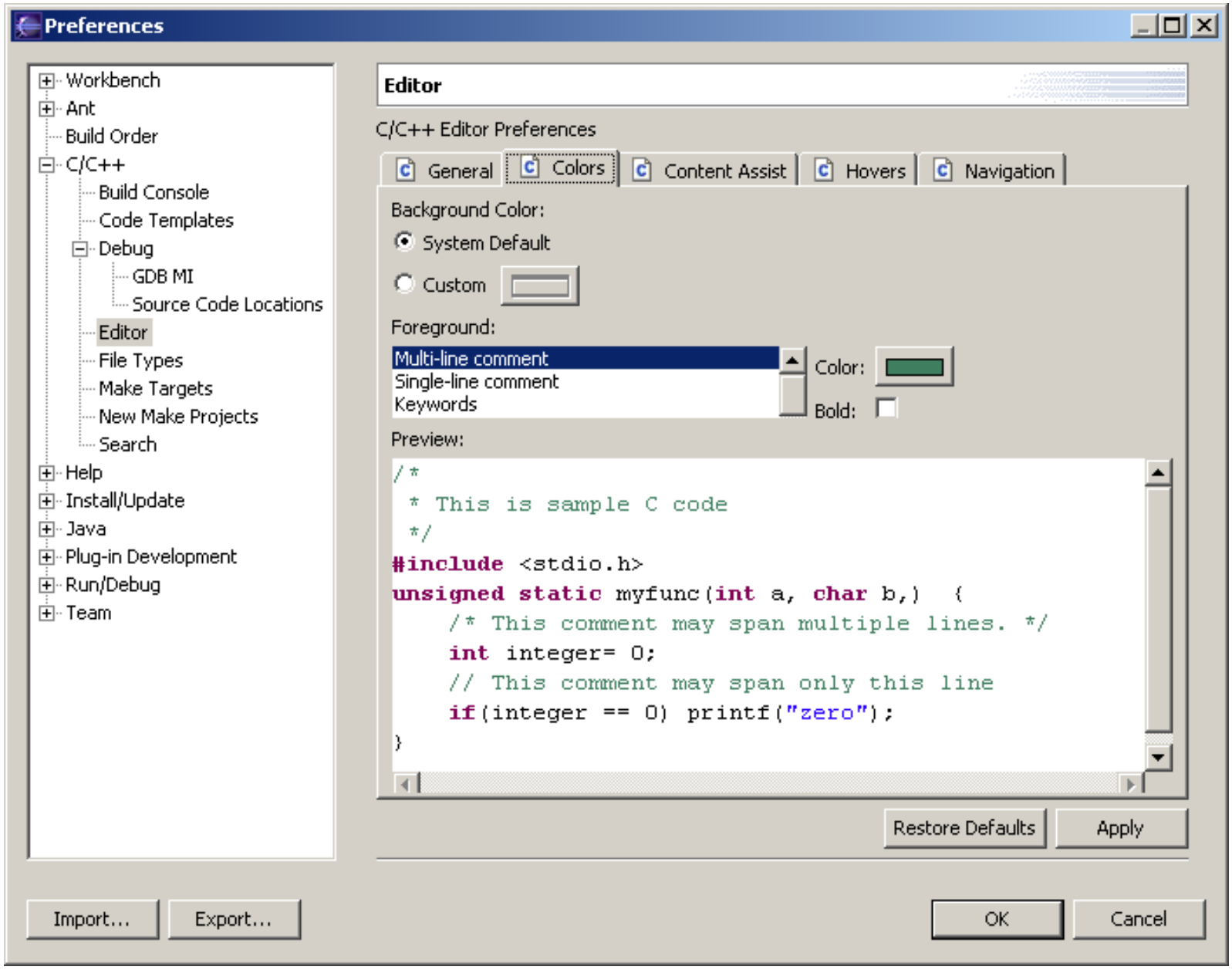
Related concepts
Coding aids

Related tasks
Customizing the C/C++ editor

Related reference
C/C++ editor preferences

Color preferences

You can customize the appearance of the C/C++ editor on the Colors page of the Preferences window.



System Default
Uses the system default for the background color.

Custom
Changes the background color.

Foreground
Lists items, such as comments, keywords, and strings, for which you can specify a color.

Color
Specifies the color in which to display the selected item.

Bold
Makes the selected item bold.

Related concepts

Coding aids

Related tasks

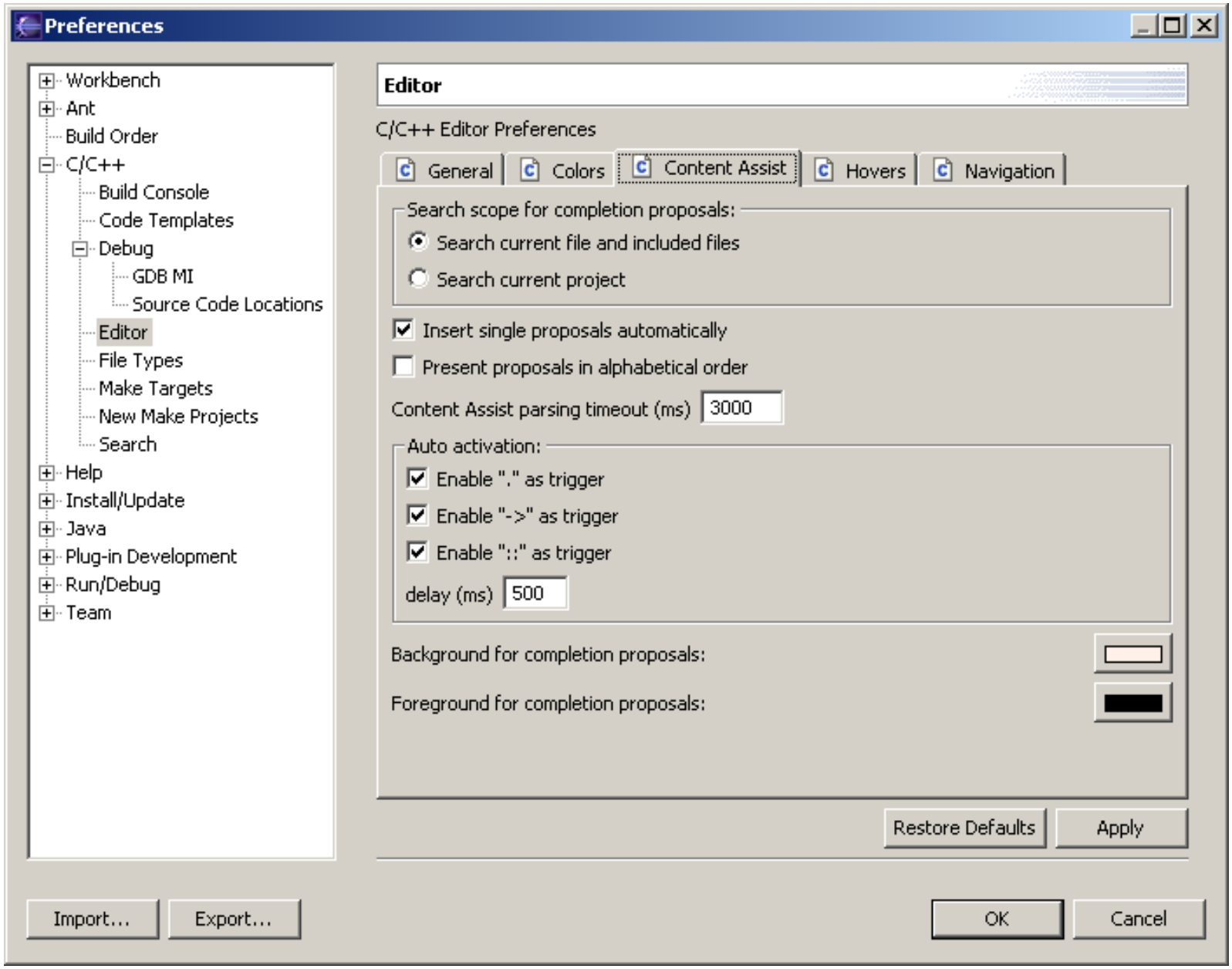
Customizing the C/C++ editor

Related reference

C/C++ editor preferences

Content Assist preferences

You can customize the Content Assist feature on the Contents Assist page of the C/C++ Editor Preferences window. To change the Content Assist preferences click **Window > Preferences > C/C++ > C++ Editor > Content Assist.**



Search scope for completion proposals

You can configure the Content Assist feature to select proposals from items contained only in the current file and included files, or from the current project.

Insert single proposals automatically

Inserts an element into your code when the Content Assist feature finds only one proposal.

Present proposals in alphabetical order

Proposals, by default, appear in order of relevance determined by context, scope and prefix. Alternatively, you can configure the Content Assist feature to order its proposals alphabetically.

Content Assist parsing timeout (ms)

For very large projects, the Content Assist feature might slow the response of the plug-in as the Content Assist feature parses the project for proposals. This setting stops the Content Assist parse when it reaches the specified threshold.

Auto activation

Certain predefined triggers force the Content Assist feature to activate. You can disable the predefined triggers with these checkboxes.

Auto activation delay

Specifies the number of milliseconds before Content Assist is activated, in Autoactivation mode.

Background for completion proposals

Specifies the background color of the Content Assist dialog box.

Foreground for completion proposals

Specifies the foreground color of the Content Assist dialog box.

Related concepts

Coding aids

Related tasks

Customizing the C/C++ editor

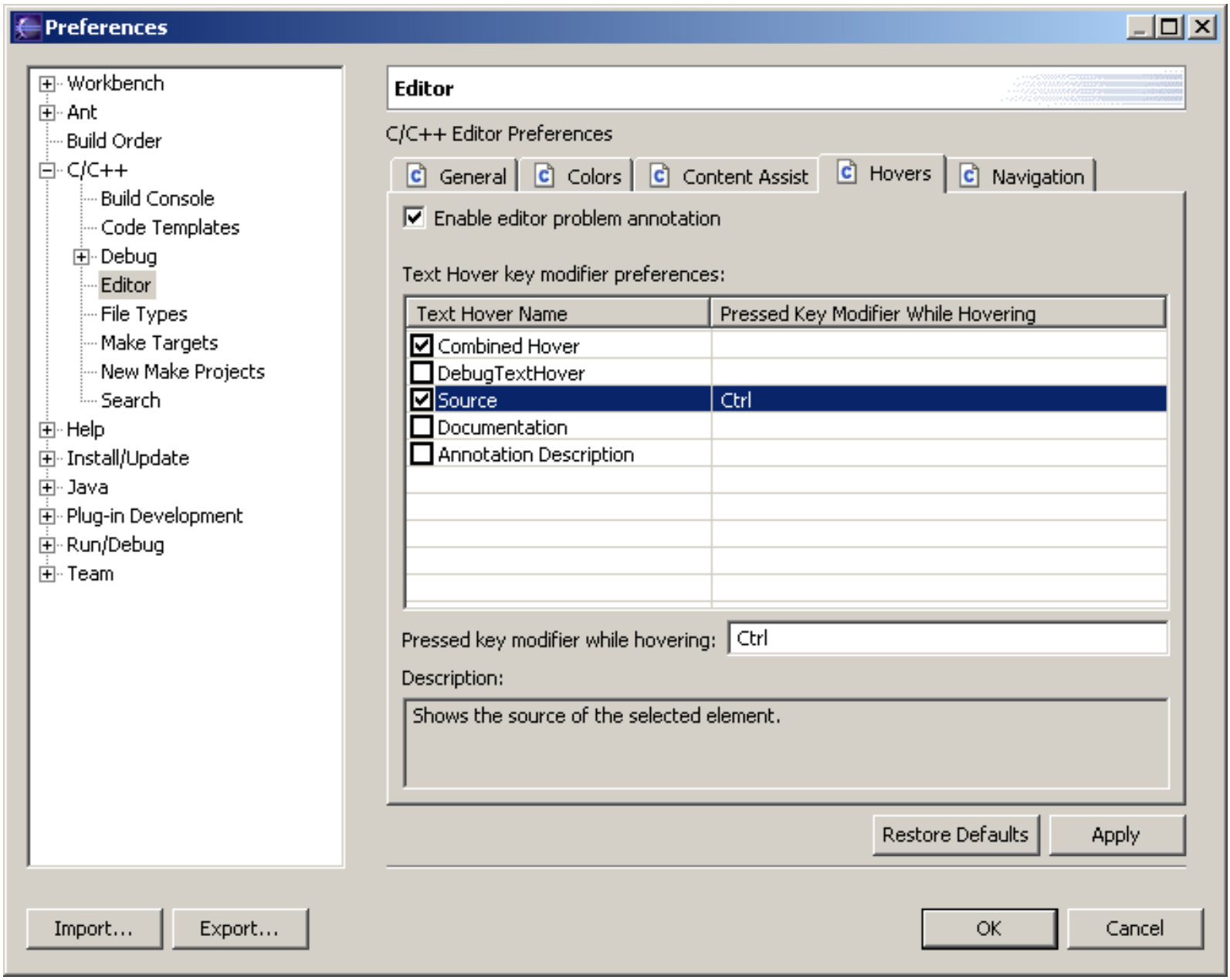
Working with Content Assist

Related reference

C/C++ editor preferences

Hover preferences

You can customize the appearance of the C/C++ editor hover behavior on the Hovers page of the Preferences window.



Enable editor problem annotation

When selected problems found will be highlighted in the editor.

Text Hover key modifier preferences:

You can select hot-keys to enable alternate hover behavior such as a mouse over while pressing the <Ctrl> key will link to the element's source declaration.

Related concepts

Coding aids

Related tasks

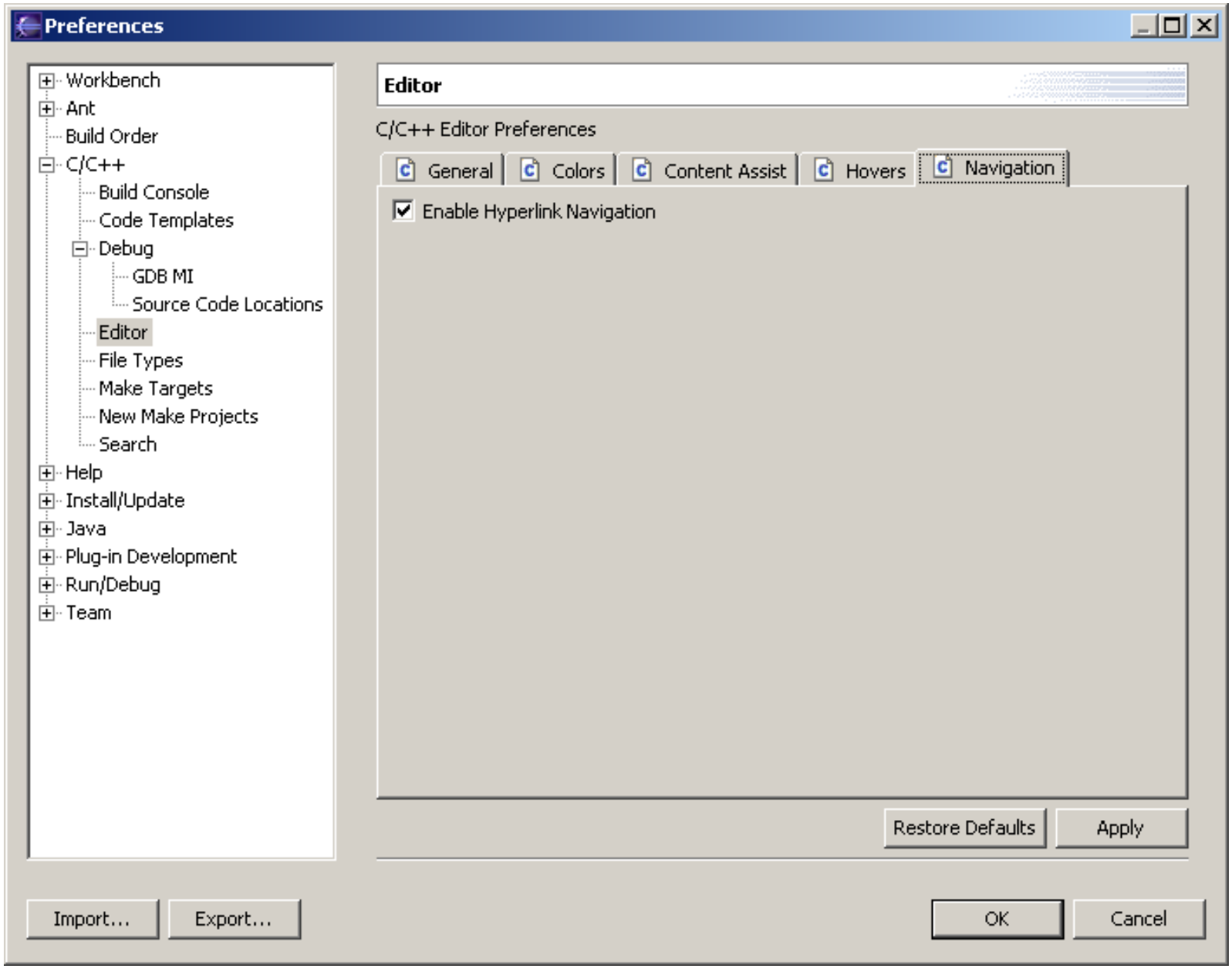
Customizing the C/C++ editor

Related reference

C/C++ editor preferences

Navigation preferences

Enable the editor hyperlink navigation and then you can use **Ctrl+click** to jump to the declaration of an item on the C/C++ editor



Enable Hyperlink Navigation

Select this checkbox to support hyperlink style navigation for Open Declaration.

Related concepts

Coding aids

Related tasks

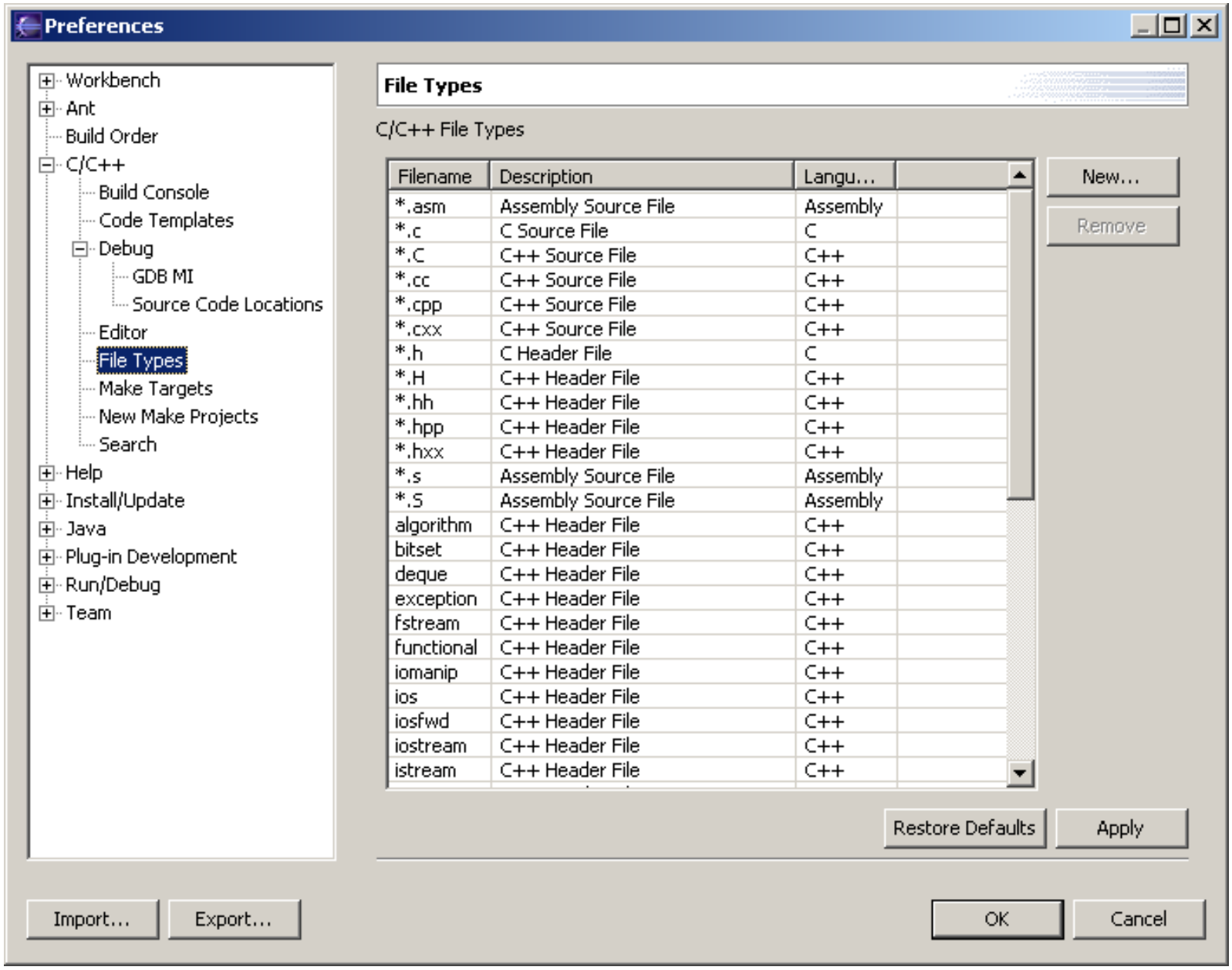
Customizing the C/C++ editor

Related reference

C/C++ editor preferences

C/C++ File Types page, Preferences window

You can define which file extensions are linked to specific languages.



New
Add a new File Type definition.

Remove
Remove the currently selected File Type definition.

Related concepts

Coding aids

Related tasks

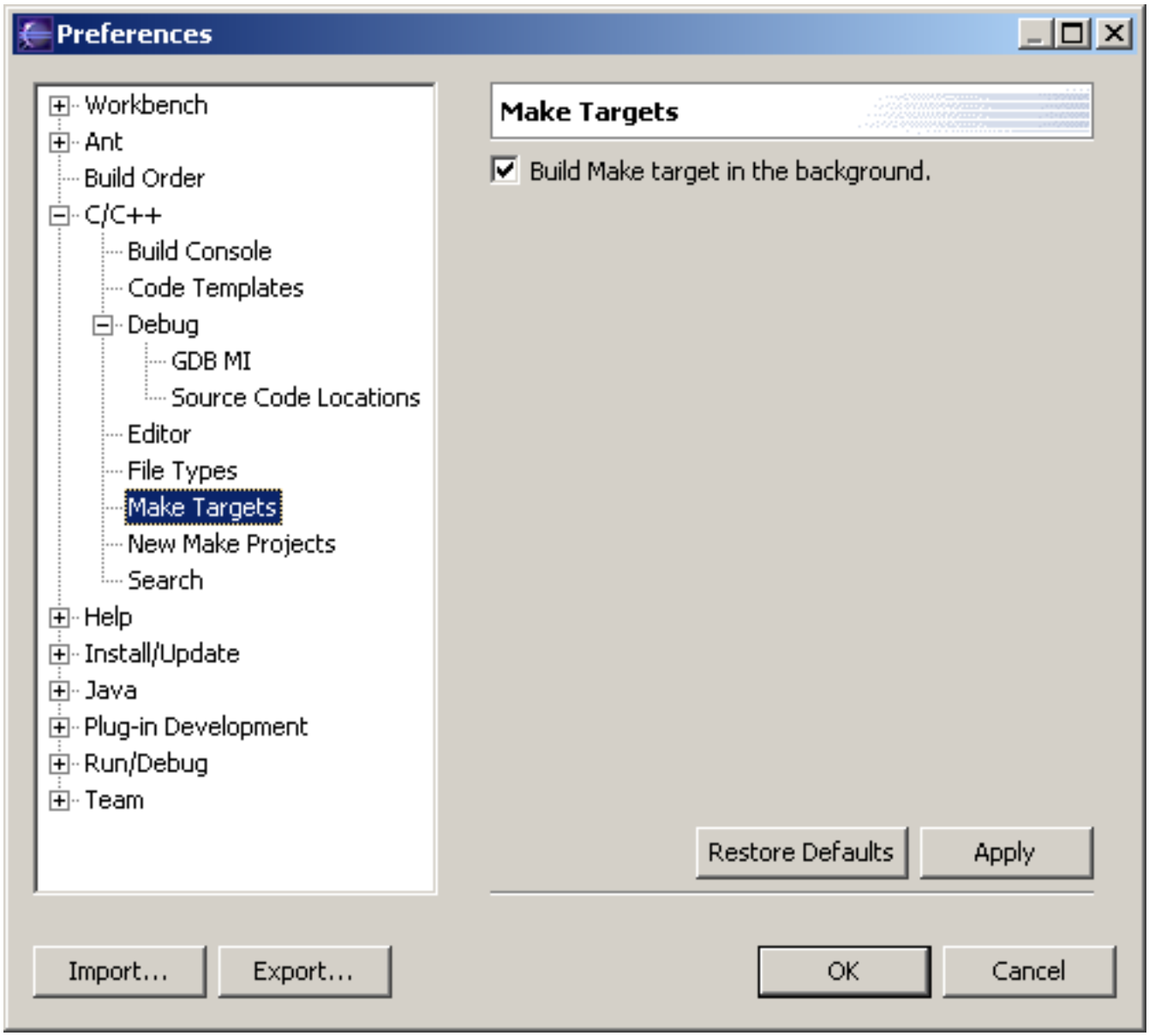
Customizing the C/C++ editor

Related reference

C/C++ editor preferences

Make Targets page, Preferences window

You can force the building of make targets in the background.



Build Make target in the background

Select this checkbox to perform builds in the background.

Related concepts

Build overview

Related tasks

Defining Build Settings

Building

Related reference

Views

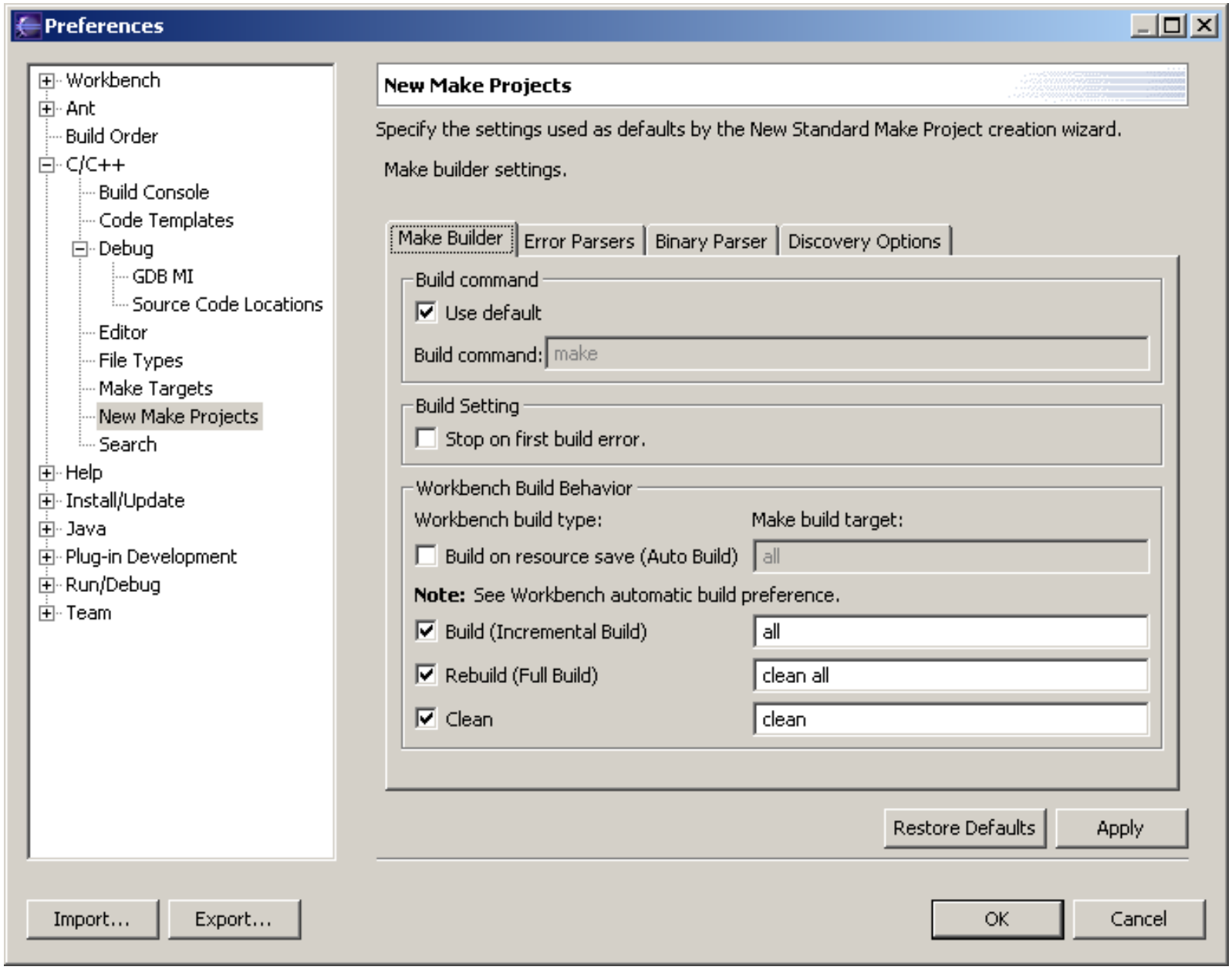
New Make Projects properties

In this section, learn about the C/C++ New Make Projects properties pages.

- Make Builder preferences
- Error Parser preferences
- Binary Parser preferences
- Discovery Options preferences

Make Builder page, C/C++ Preferences window

You can define build settings on the Make Builder page of the C/C++ Preferences window.



Use default
Select this checkbox to use the default make command. Clear the check box to specify a new make command.

Build command
If you clear the **Use default** checkbox type a new make command in this field.

Stop on first build error
Stops the build when an error occurs.

Workbench Build Behavior
These settings are what the standard builder will call by default when told to build, rebuild, clean, etc. You can change these so that new projects will use different targets if the defaults are not appropriate.

Build on resource save (Auto Build)
This defines what the standard builder will call when a file is saved, it is not recommended to enable Auto Build for C/C ++ projects.

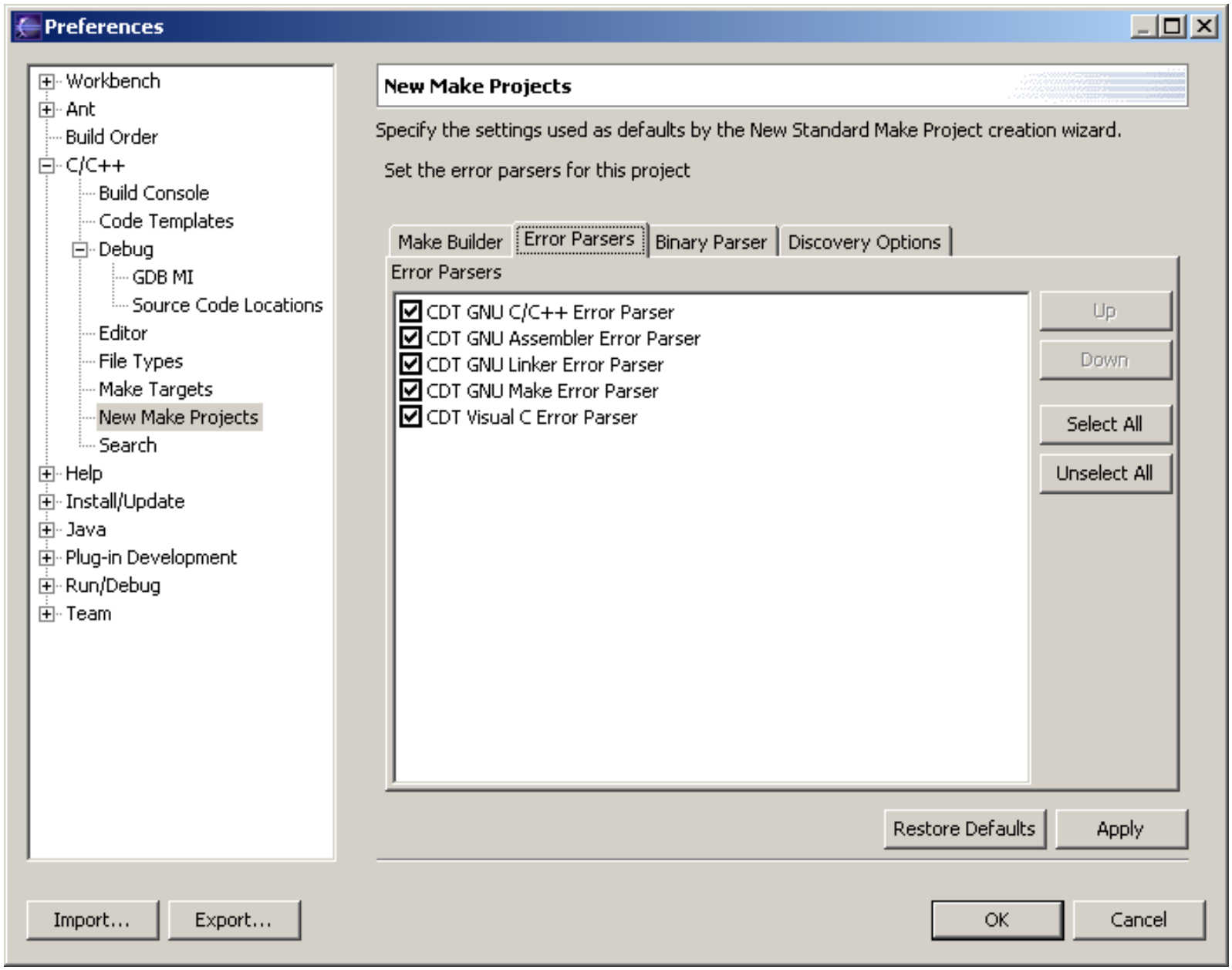
Related concepts
Build overview

Related tasks
Defining build settings

Related reference
Project properties

Error Parsers, C/C++ Preferences window

You can view a list of the filters that detect error patterns in the build output log, on the Error Parsers page of the Preferences window.



Error Parsers
Lists the various error parsers which can be enabled or disabled.

Up
Moves the selected error parser higher in the list.

Down
Moves the selected error parser lower in the list.

Select All
Selects all error parsers.

Unselect All
Clears all error parsers.

Related concepts

Build overview

Related tasks

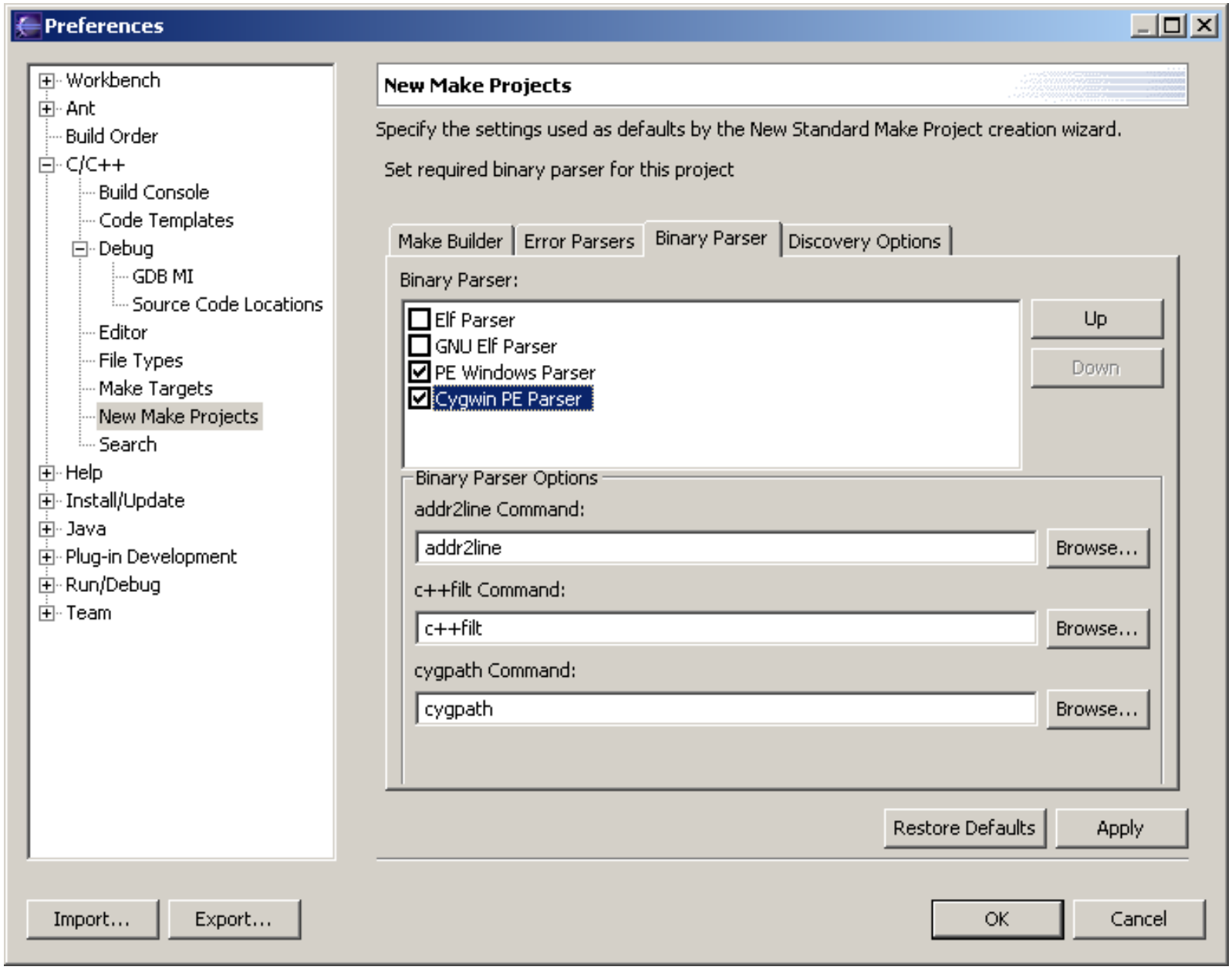
Filtering errors

Related reference

Project properties

Binary Parser page, C/C++ Preferences window

You can view a list of binary parsers on the Binary Parser page of the C/C++ Preferences window.



Binary Parser

Select binary parsers from the list, and changed the order in which they are used.

Binary Parser Options

If a binary parser has parser options you can define them in this section.

Related concepts

Build overview

Related tasks

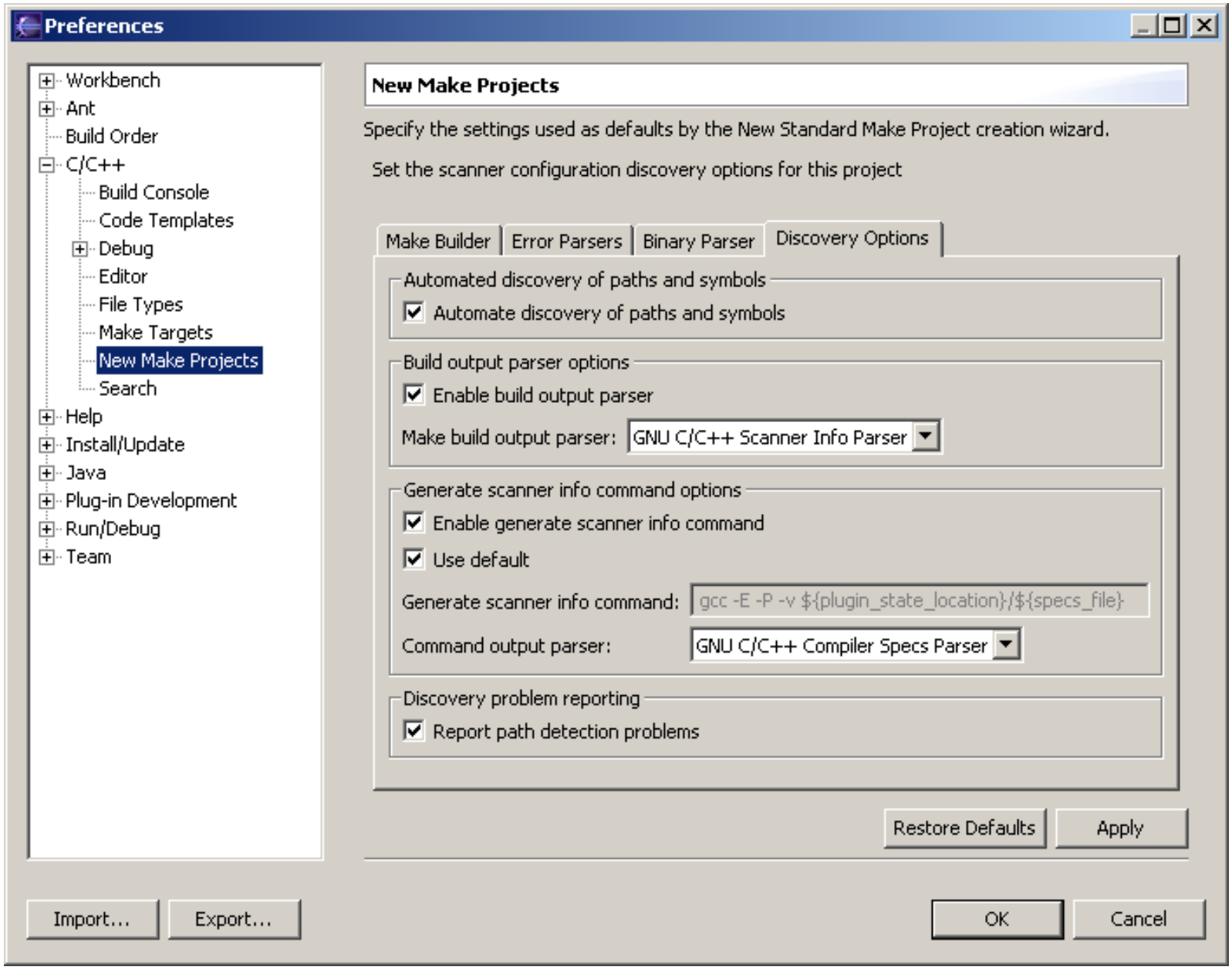
Selecting a binary parser

Related reference

Project properties

Discovery Options page, C/C++ Preferences window

You can define the discovery options on the Discovery Options page of a C/C++ Preferences window.



Automate discovery of paths and symbols
Select this checkbox to scan the build output for paths and symbols.

Build output parser options
This section allows you to select the make build output parser.

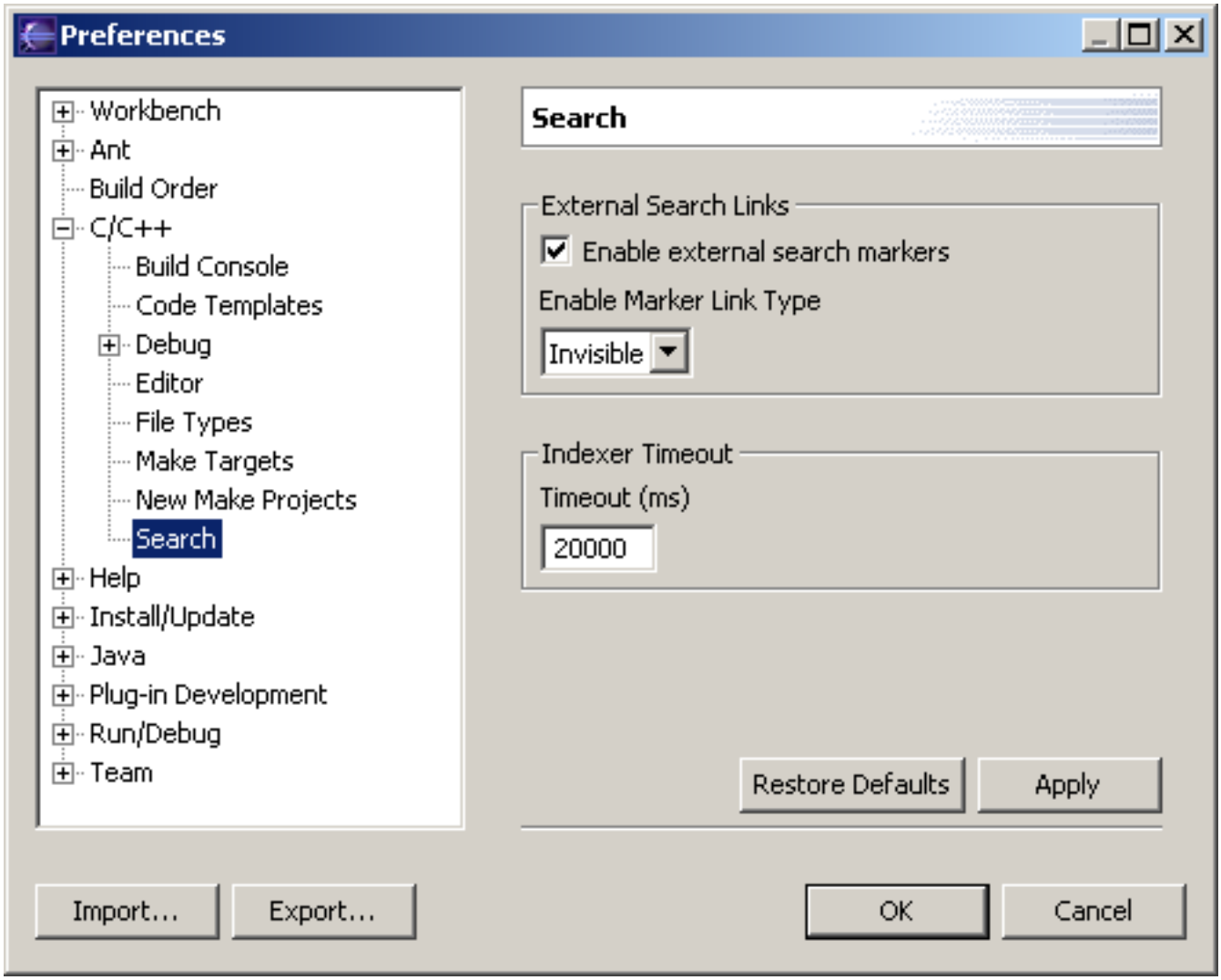
Generate scanner info command options
Select to invoke secondary paths and symbols provider (such as GNU specs).

Restore Defaults
Returns any changes back to their default setting.

Apply
Applies any changes.

Search, C/C++ Preferences window

You can configure various options for the search configuration on the Search page of the C/C++ Properties window.



Enable external search markers
Select this checkbox to enable searches in external files.

Enable Marker Link Type
Select visible or invisible markers.

Indexer Timeout:
Enter the timeout delay (in milliseconds) in the field provided.

Related concepts

Coding aids

C/C++ search

Related tasks

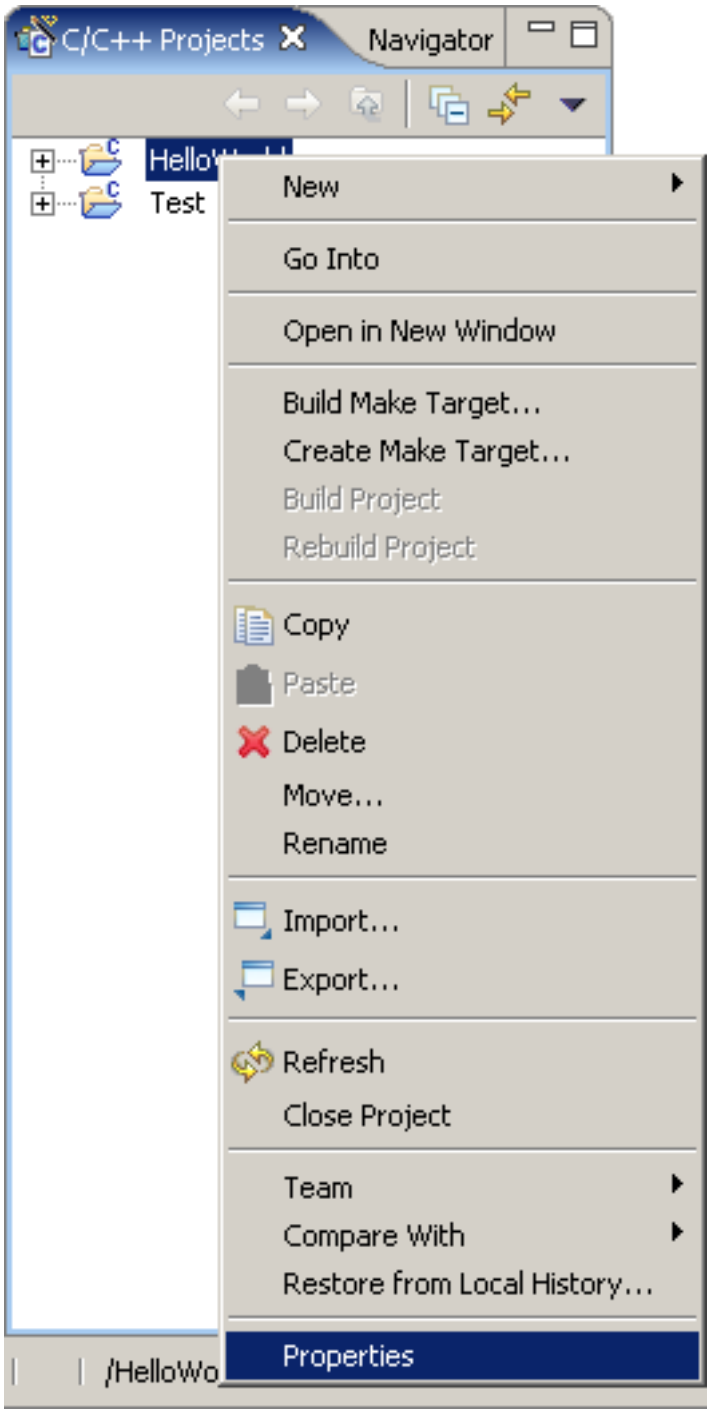
Searching for C/C++ elements
Customizing the C/C++ editor

Related reference

C/C++ search page, Search window
Search action
Search view

Project Properties

This section describes C/C++ Project Properties. To select project properties, right click a project and select **Properties**.



Related concepts

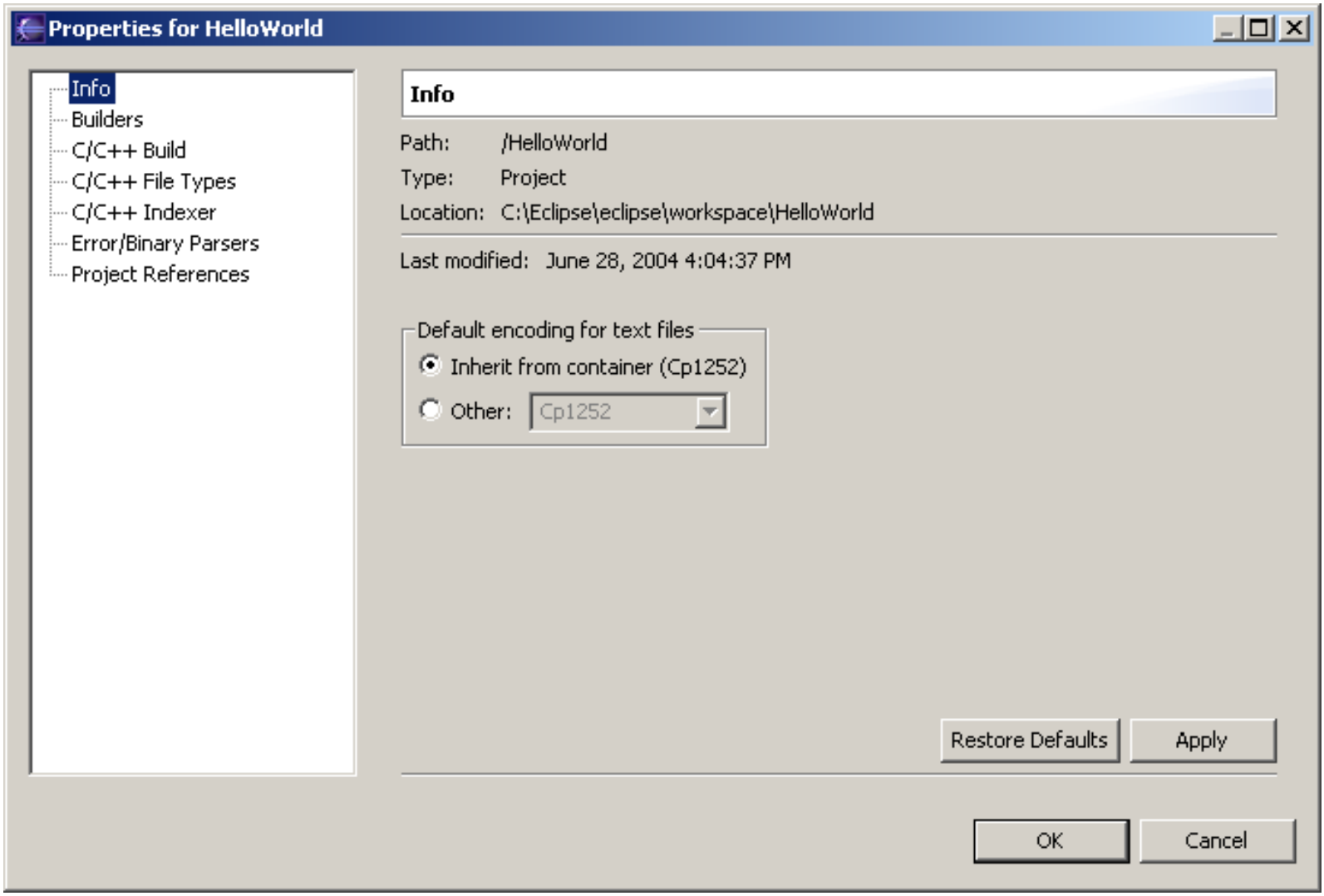
CDT projects

Related reference

C/C++ Project Properties, Managed Make Project

This section describes properties for a Managed make project.

C/C++ Project Properties, Managed, Info



Info
Shows project information.

Default encoding for text files
You can specify an alternate text encoding.

Restore Defaults
Returns any changes back to their default setting.

Apply
Applies any changes.

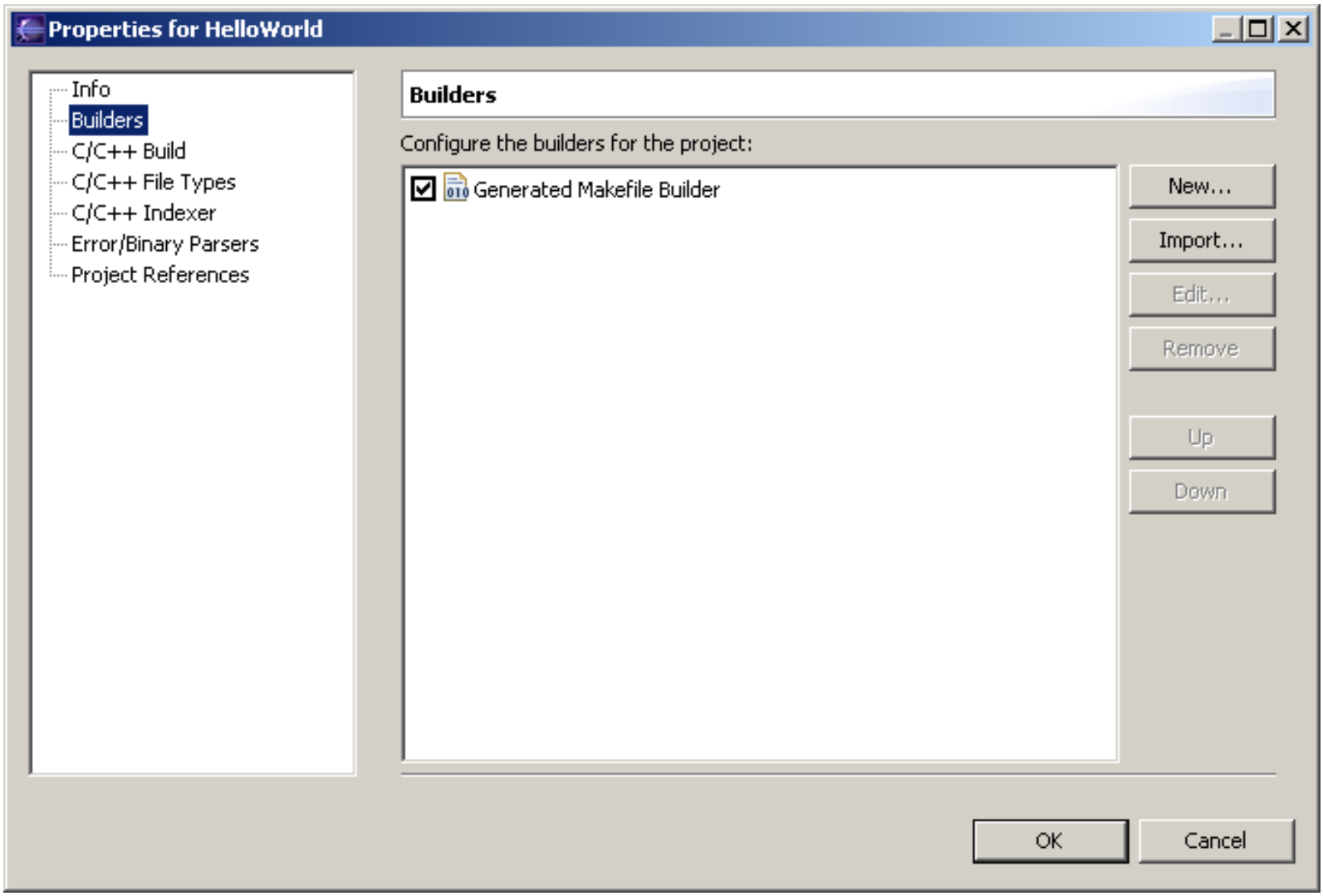
Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options

C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Managed, Builders

You can select which Builders to enable for this project and in which order they are used.



Builders
Select which Builders to enable.

New..
Add a new builder.

Import..
Import a builder.

Edit..
Edit a builder.

Remove
Remove a builder.

Up
Move the currently selected builder higher in the list.

Down
Move the currently selected builder lower in the list.

Related reference

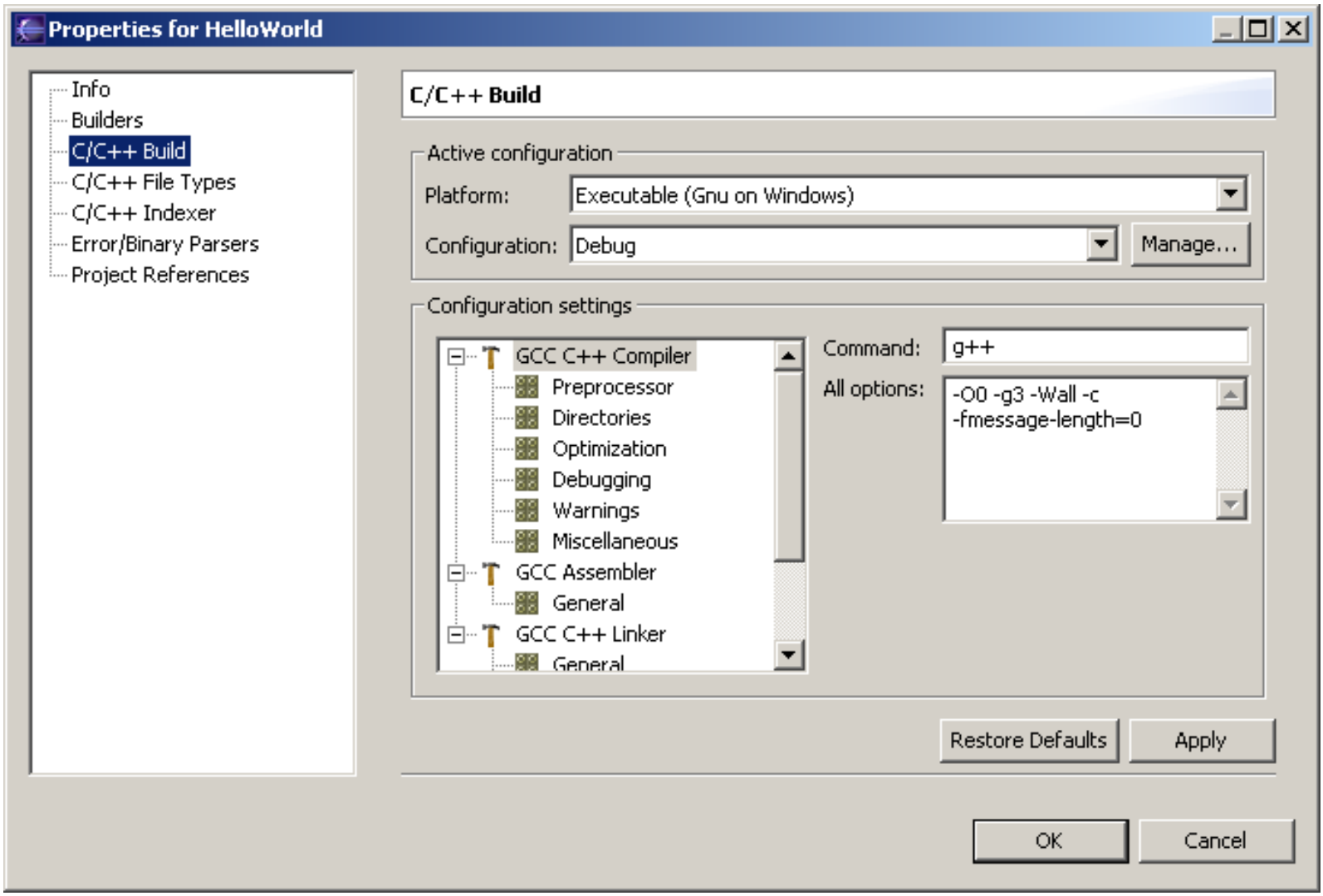
C++ Project Properties, Standard, Info

C++ Project Properties, Standard, Builders

C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Managed, Build

Customizes your build configuration, compiler and compile options.



Platform:
Select the platform from the list provided.

Configurations:
Select the build configuration from the list provided.

Manage...
You can change the make command, make flags, add and remove configurations, and change the name of the build goal.

Configuration Settings
Edit individual options.

Restore Defaults
Returns any changes back to their default setting.

Apply
Applies any changes.

Related reference

C++ Project Properties, Standard, Info

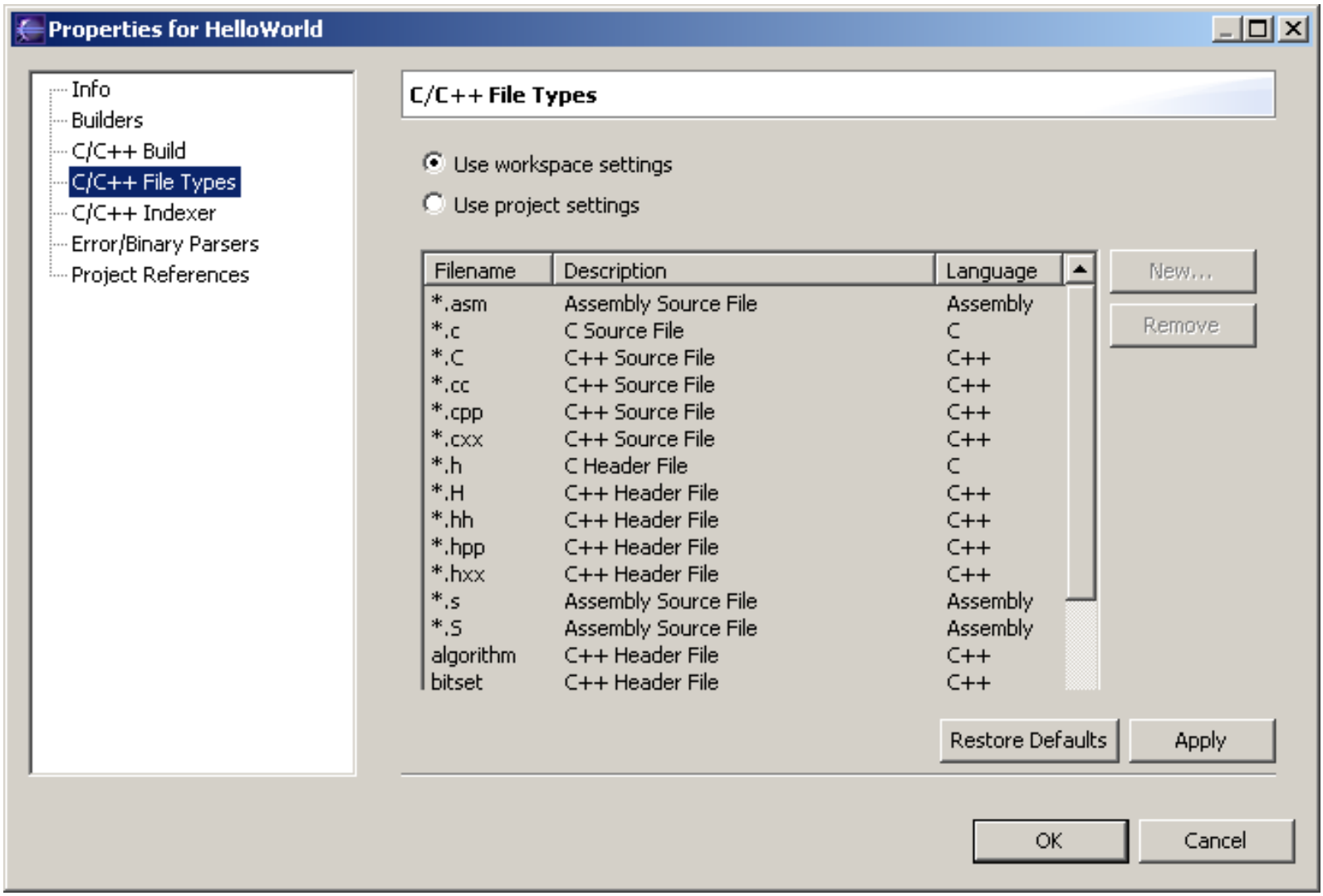
C++ Project Properties, Standard, Builders

C++ Project Properties, Standard, File Types

C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Managed, File Types

You can view a list of file types on the File Types page of a C/C++ project's properties window.



Use workspace settings
Select this to use the Managed workspace settings.

Use Project Settings
Select this option to use project settings, or add or remove specific file types.

New...
Add a new file type.

Remove
Remove a listed file type.

Restore Defaults
Returns any changes back to their default setting.

Apply
Applies any changes.

Related reference

C++ Project Properties, Standard, Info

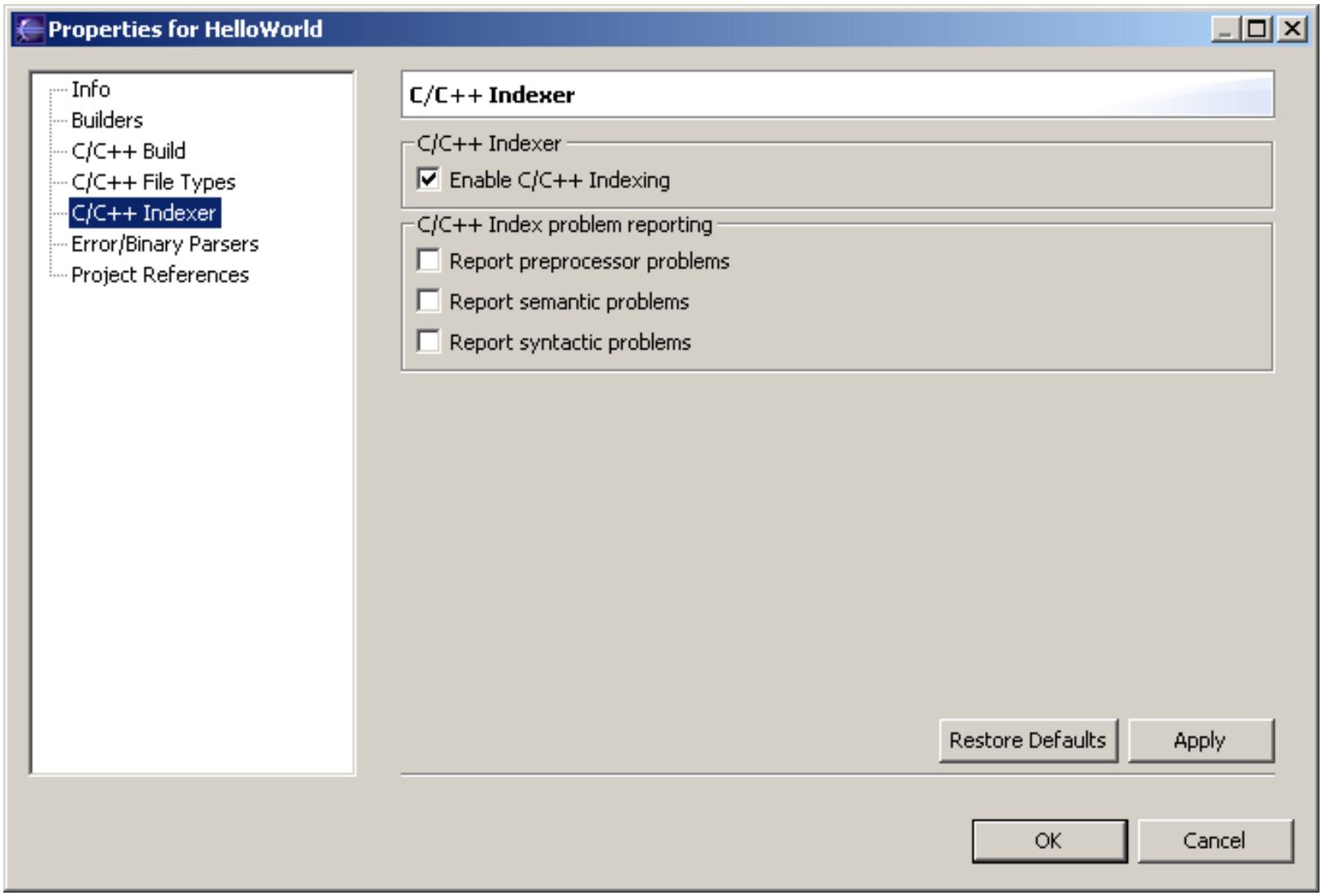
C++ Project Properties, Standard, Builders

C++ Project Properties, Standard, File Types

C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Managed, Indexer

You can select which C/C++ Indexers to use for your project. The indexer is necessary for search and related features, like content assist.



Enable C/C++ Indexing
When selected C++ Indexing is enabled for this project.

Enable C++ Index problem reporting
When selected, any index related errors detected will be reported.

Restore Defaults
Returns any changes back to their default setting.

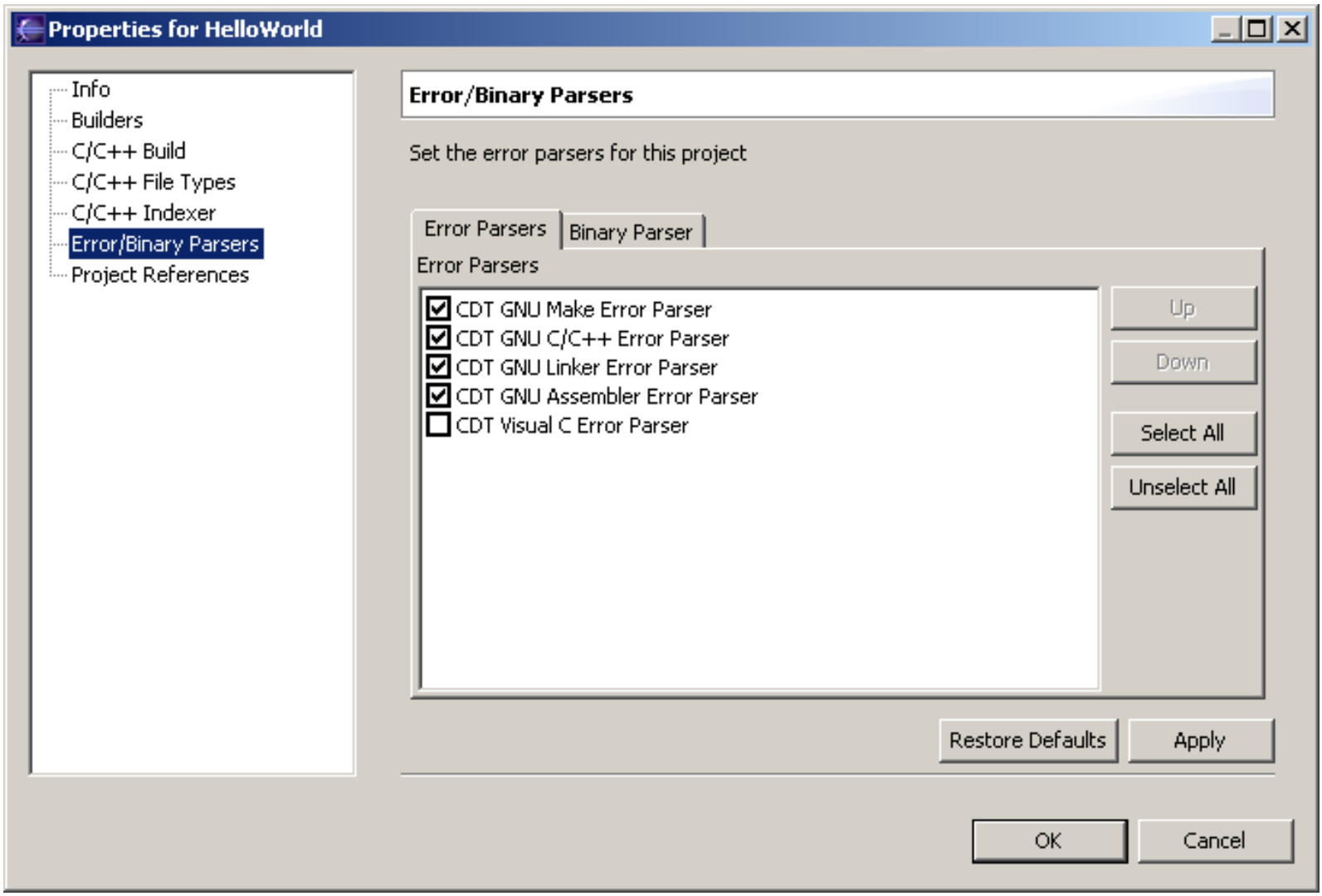
Apply
Applies any changes.

Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder

C/C++ Project Properties, Managed, Error Parser

You can view a list of the filters that detect error patterns in the build output log, on the Error Parsers page of a C/C++ project's preferences window.



Error Parsers
Lists the various error parsers which can be enabled or disabled.

Up
Moves the selected error parser higher in the list.

Down
Moves the selected error parser lower in the list.

Select All
Selects all error parsers.

Unselect All
Clears all error parsers.

Related concepts

Build overview

Related tasks

Filtering errors

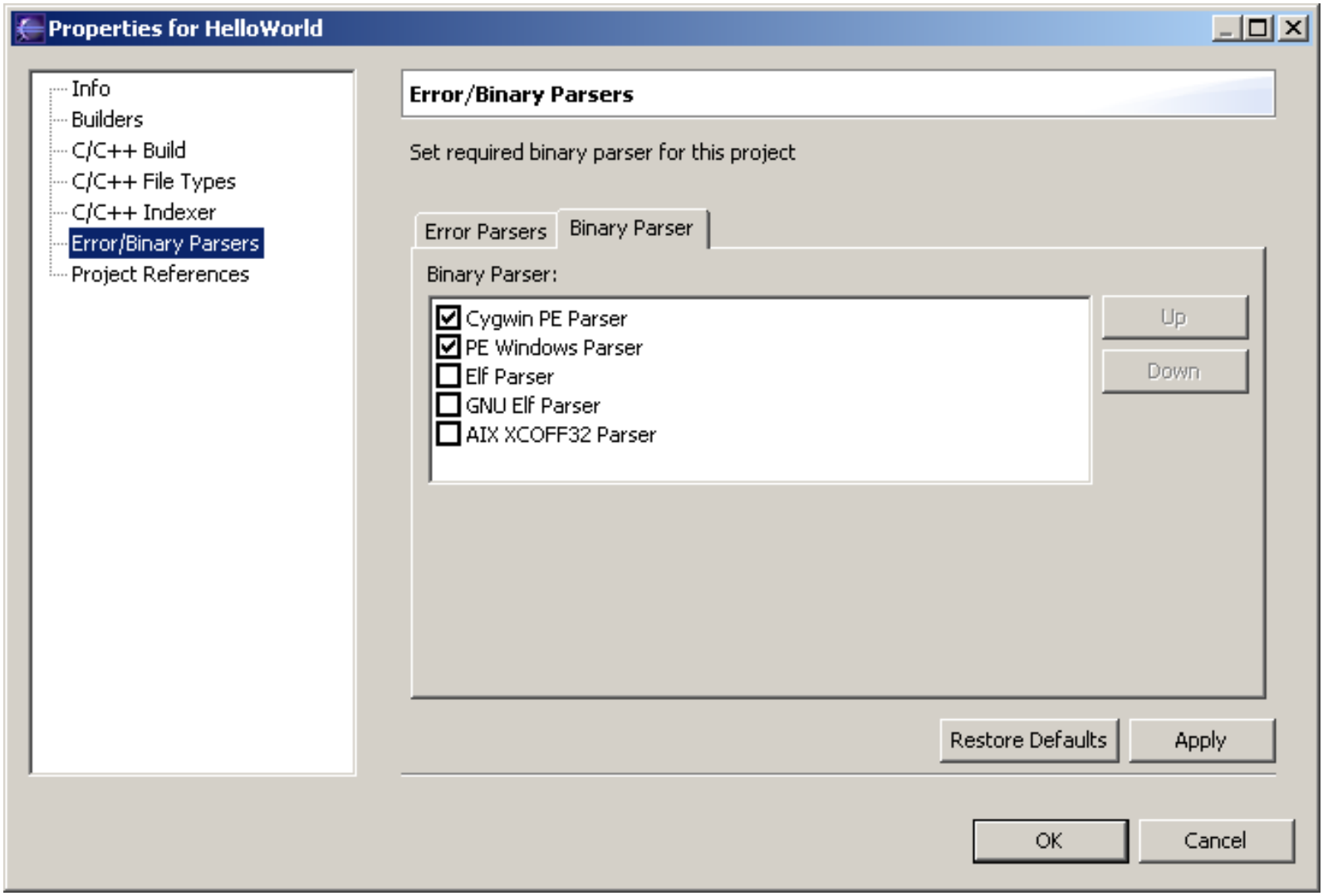
Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Managed, Binary Parser

You can select the Binary Parsers you require for the project.

To ensure the accuracy of the C/C++ Projects view and the ability to successfully run and debug your programs. After you select the correct parser for your development environment and build your project, you can view the symbols of the .o file in the C/C++ Projects view.



Binary Parser
Select a binary parser from the list.

Binary Parser Options
If a binary parser has parser options you can define them in this section.

Restore Defaults
Returns any changes back to their default setting.

Apply
Applies any changes.

Related concepts

Build overview

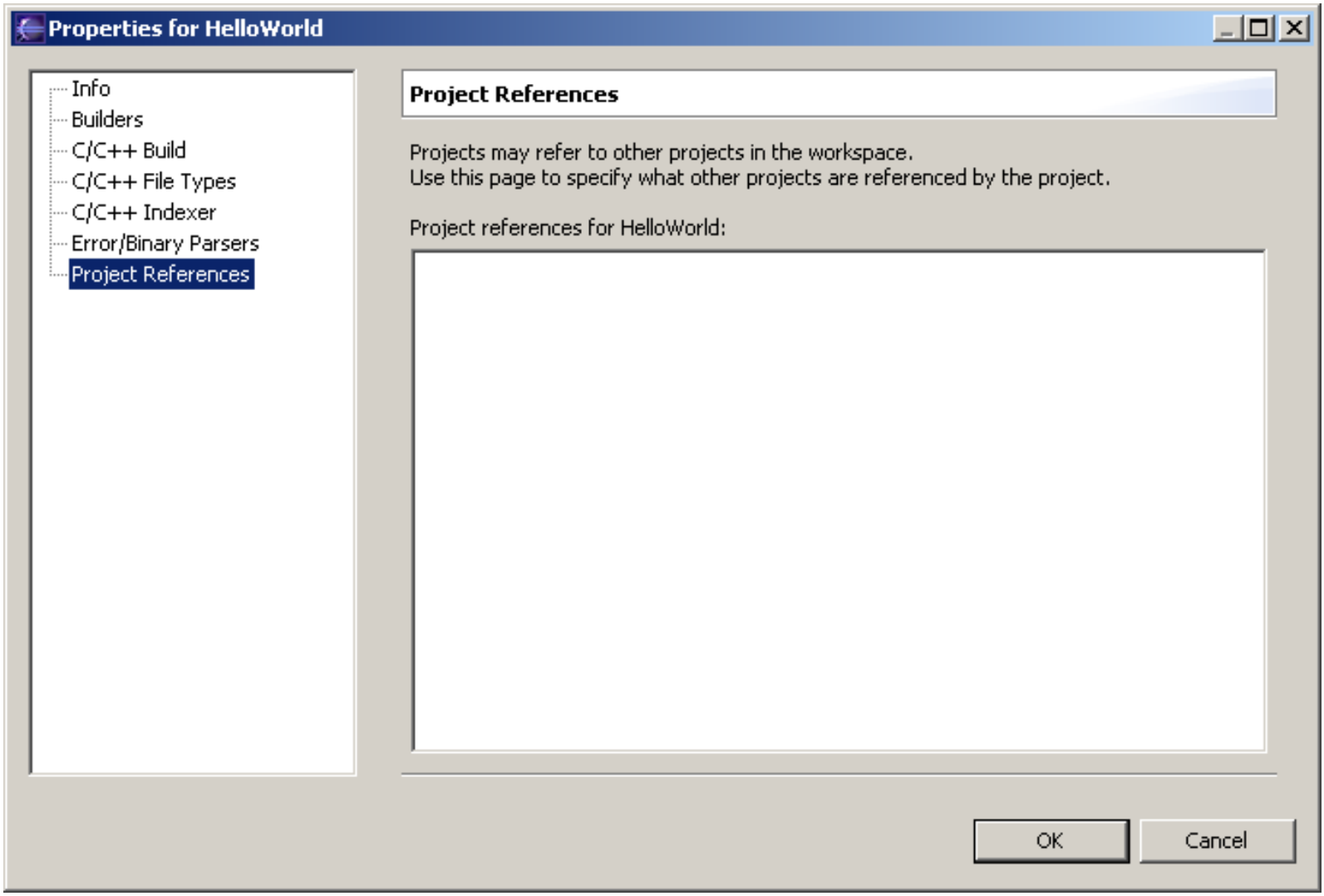
Related tasks

Selecting a binary parser

Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Managed, Project References



Project references for <project>:

Select the projects required to build this project.

Related reference

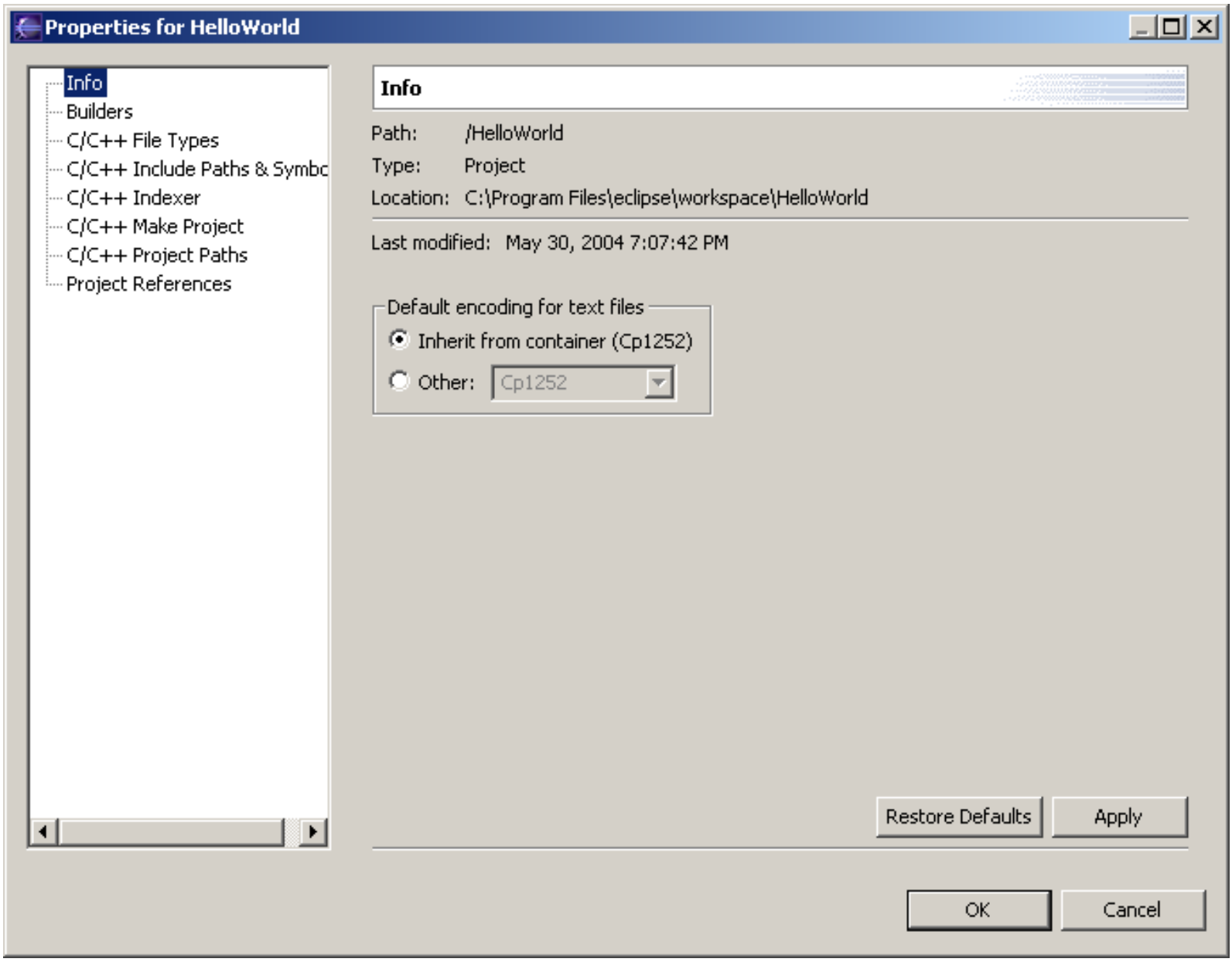
C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers

C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard Make Project

This section describes properties for a Standard make project.

C/C++ Project Properties, Standard, Info



Info
Shows project information.

Default encoding for text files
You can specify an alternate text encoding.

Restore Defaults
Returns any changes back to their default setting.

Apply
Applies any changes.

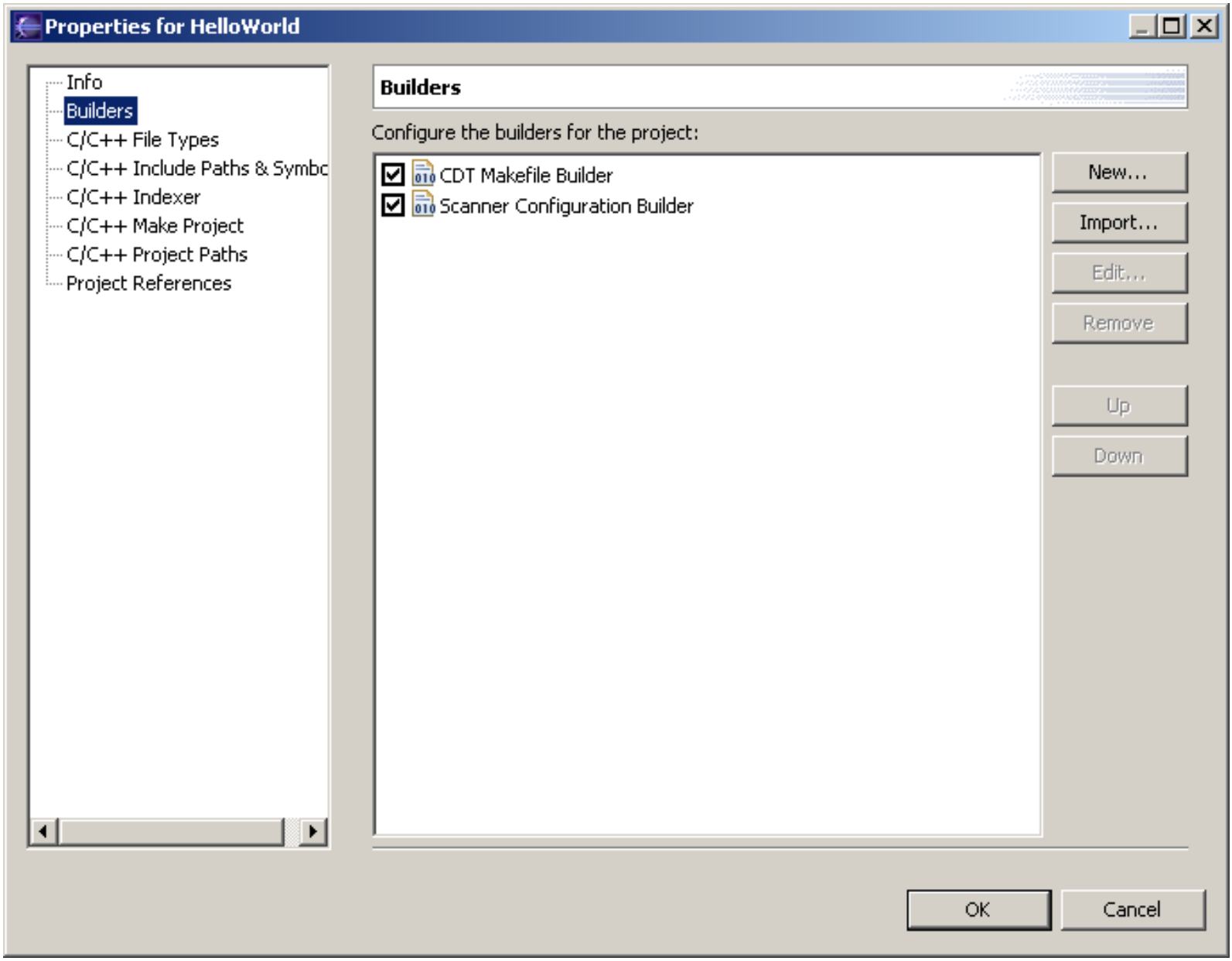
Related reference

C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder

C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard, Builders

You can select which Builders to enable for this project and in which order they are used.



Builders
Select which Builders to enable.

New..
Add a new builder.

Import..
Import a builder.

Edit..
Edit a builder.

Remove
Remove a builder.

Up
Move the currently selected builder higher in the list.

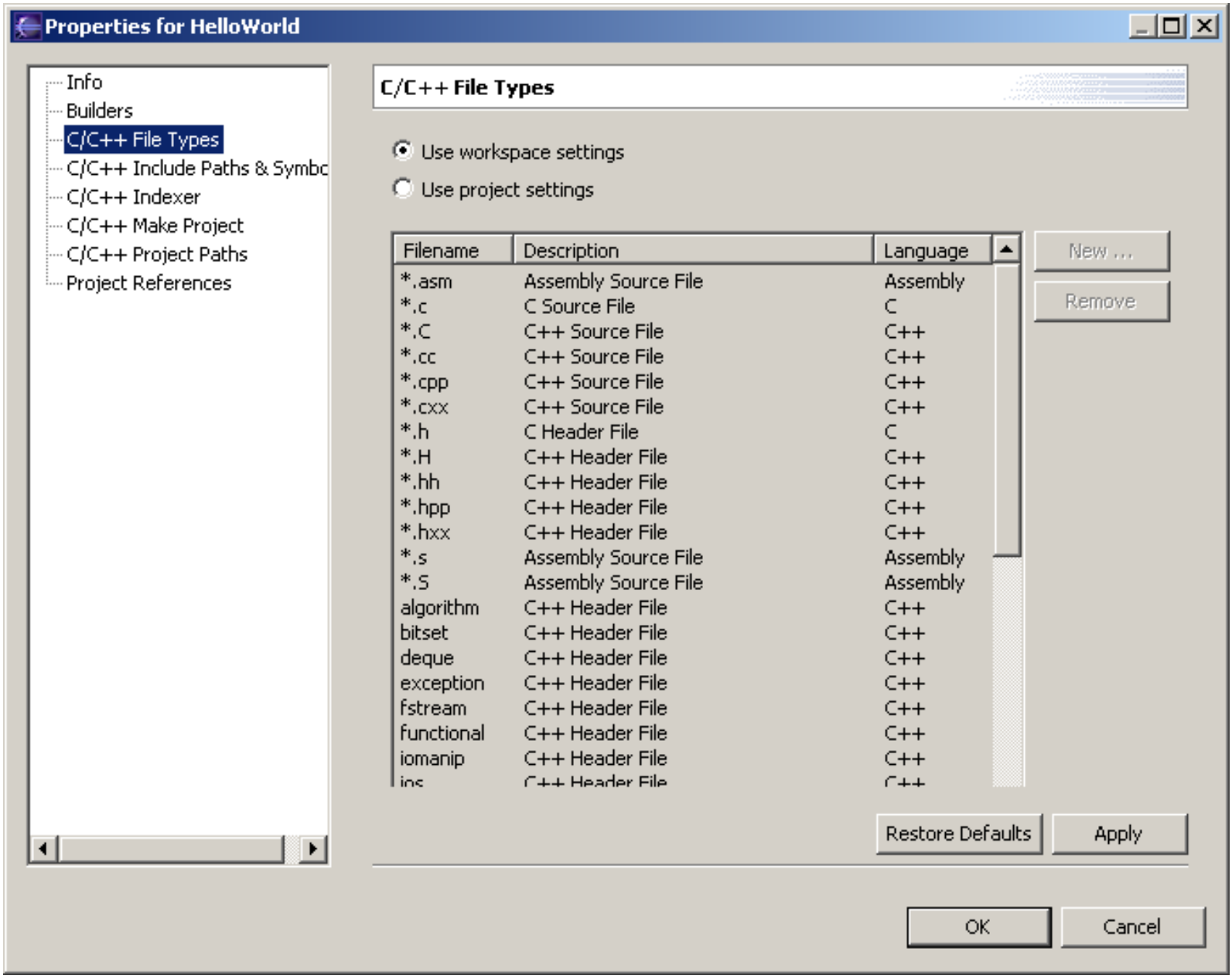
Down
Move the currently selected builder lower in the list.

Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard, File Types

You can view a list of file types on the File Types page of a C/C++ project's properties window.



Use workspace settings
Select this to use the Standard workspace settings.

Use Project Settings
Select this option to use project settings, or add or remove specific file types.

New...
Add a new file type.

Remove
Remove a listed file type.

Restore Defaults
Returns any changes back to their default setting.

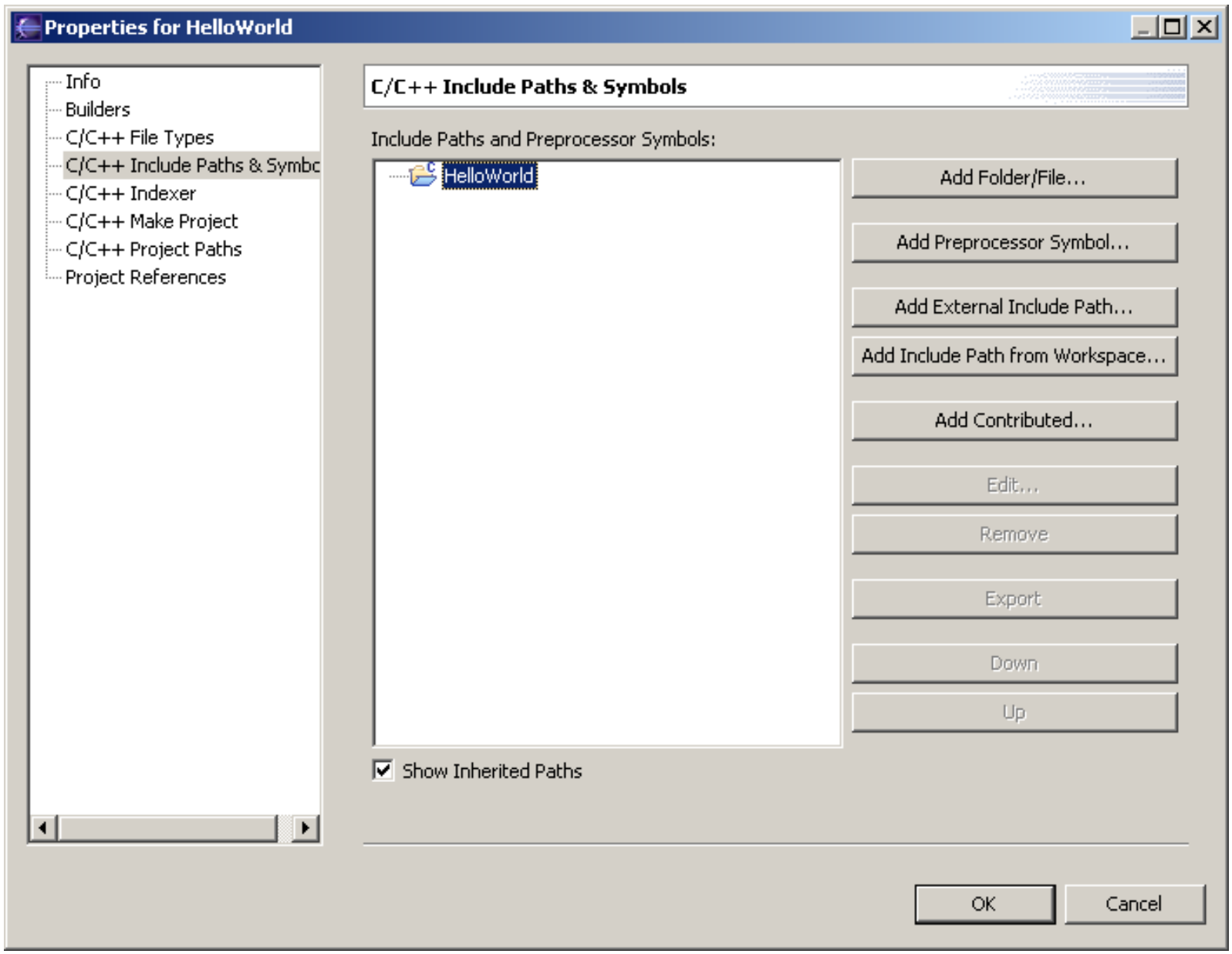
Apply
Applies any changes.

Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard, Include Paths and Symbols

You can modify the list of included paths and preprocessor symbols and change the order in which they are referenced.



Add Folder/File...
Add a new file or folder to the existing project.
Show Inherited Paths
Select this to show all paths, including those that were inherited.
Add Preprocessor Symbol...
Add a new preprocessor symbol.
Add External Include Paths...
Add a new include path.
Add Include Path from Workspace...
Add an include path from another project in the workspace.
Add Contributed...
Add a contributed path or symbol.
Edit
Edit the currently selected path or symbol.
Remove

Remove the currently selected path or symbol.

Export

Export the currently selected path or symbol to a text file.

Down

Move the currently selected path or symbol down in the list.

Up

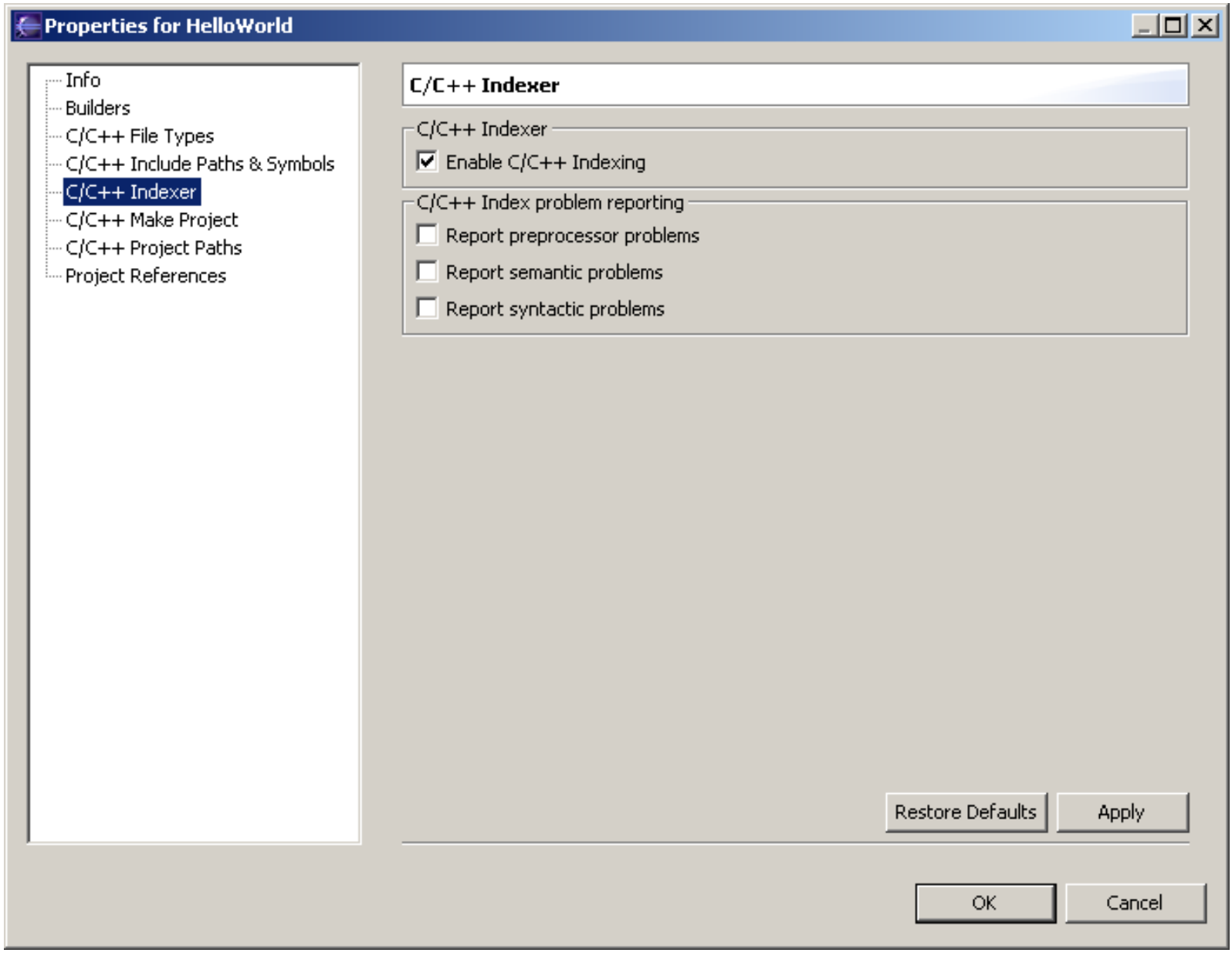
Move the currently selected path or symbol up in the list.

Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard, Indexer

You can select which C/C++ Indexers to use for your project. The indexer is necessary for search and related features, like content assist.



Enable C/C++ Indexing
When selected C++ Indexing is enabled for this project.

Enable C++ Index problem reporting
When selected, any index related errors detected will be reported.

Restore Defaults
Returns any changes back to their default setting.

Apply
Applies any changes.

Related reference

C++ Project Properties, Standard, Info

C++ Project Properties, Standard, Builders

C++ Project Properties, Standard, File Types

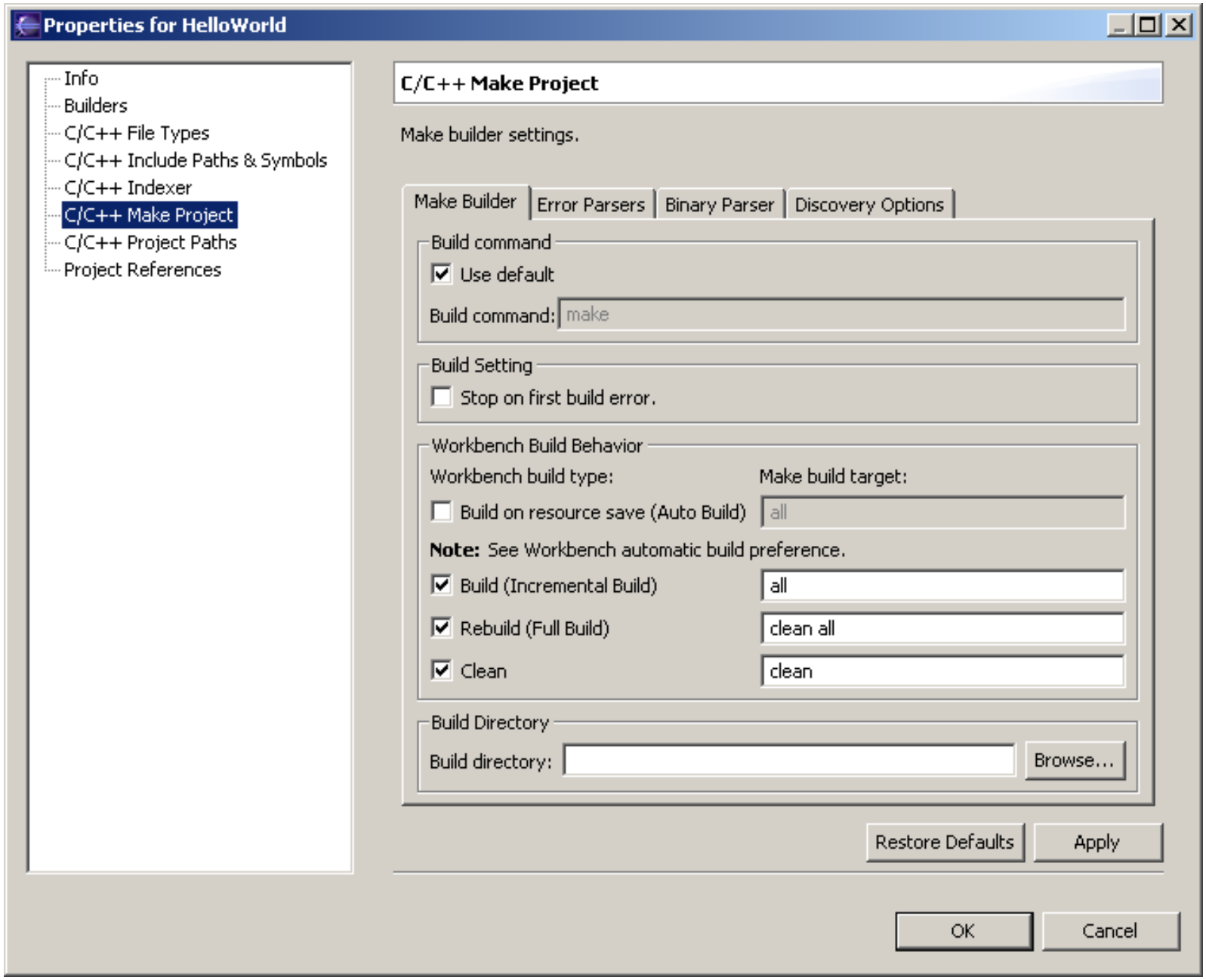
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard, Make Project

This section describes C/C++ Make Project Properties for a Standard make project.

C/C++ Project Properties, Standard, Make Builder

You can define build settings on the Make Builder page of a C/C++ project's preferences window.



Use default
Select this checkbox to use the default make command. Clear the check box to specify a new make command.

Build command
If you clear the **Use default** checkbox type a new make command in this field.

Stop on first build error
Stops the build when an error occurs.

Workbench Build Behavior
These settings are what the standard builder will call by default when told to build, rebuild, clean, etc. You can change these so that new projects will use different targets if the defaults are not appropriate.

Build on resource save (Auto Build)
This defines what the standard builder will call when a file is saved, it is not recommended to enable Auto Build for C/C ++ projects.

Related concepts

Build overview

Related tasks

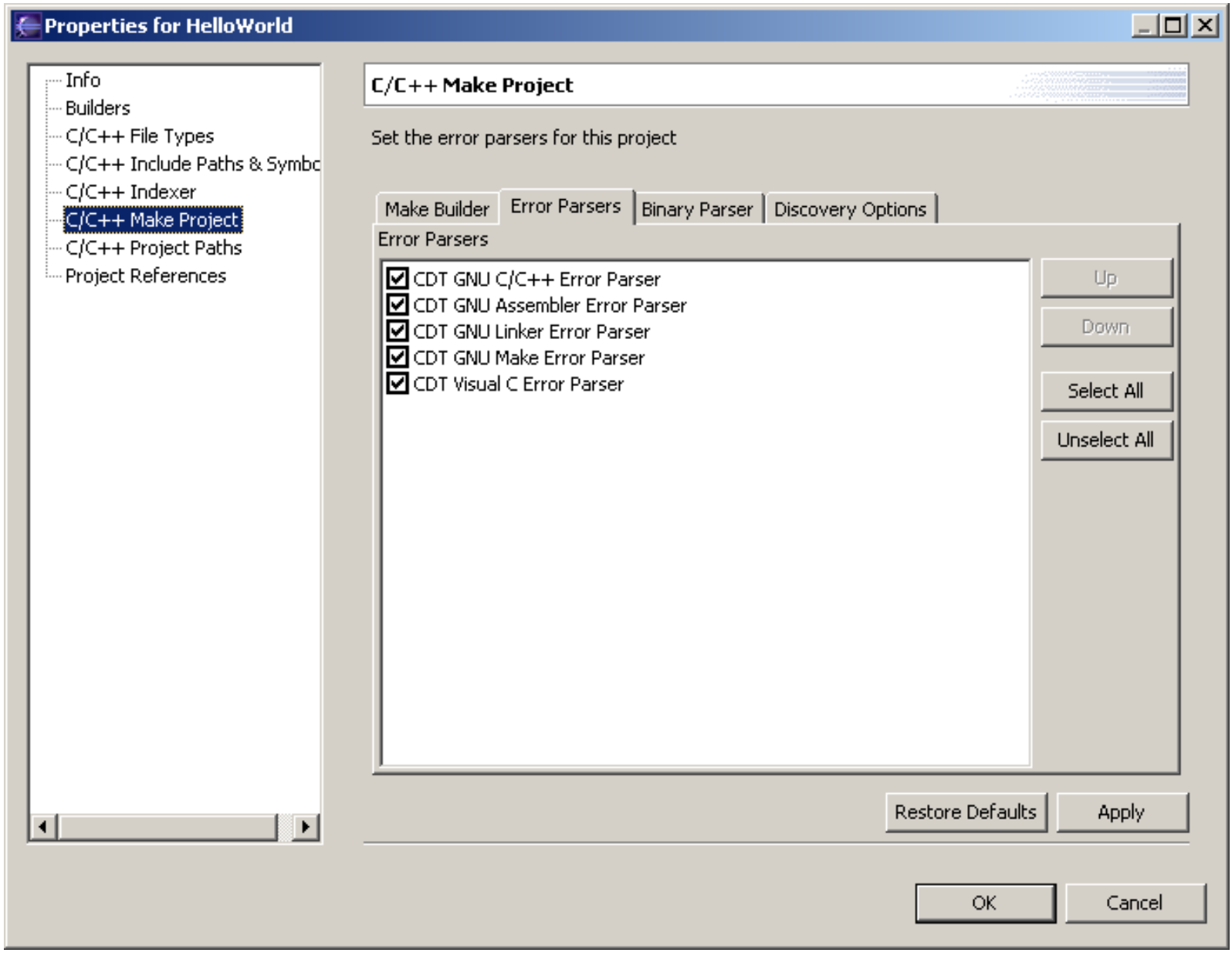
Defining build settings

Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard, Error Parser

You can view a list of the filters that detect error patterns in the build output log, on the Error Parsers page of a C/C++ project's preferences window.



Error Parsers
Lists the various error parsers which can be enabled or disabled.

Up
Moves the selected error parser higher in the list.

Down
Moves the selected error parser lower in the list.

Select All
Selects all error parsers.

Unselect All
Clears all error parsers.

Related concepts

Build overview

Related tasks

Filtering errors

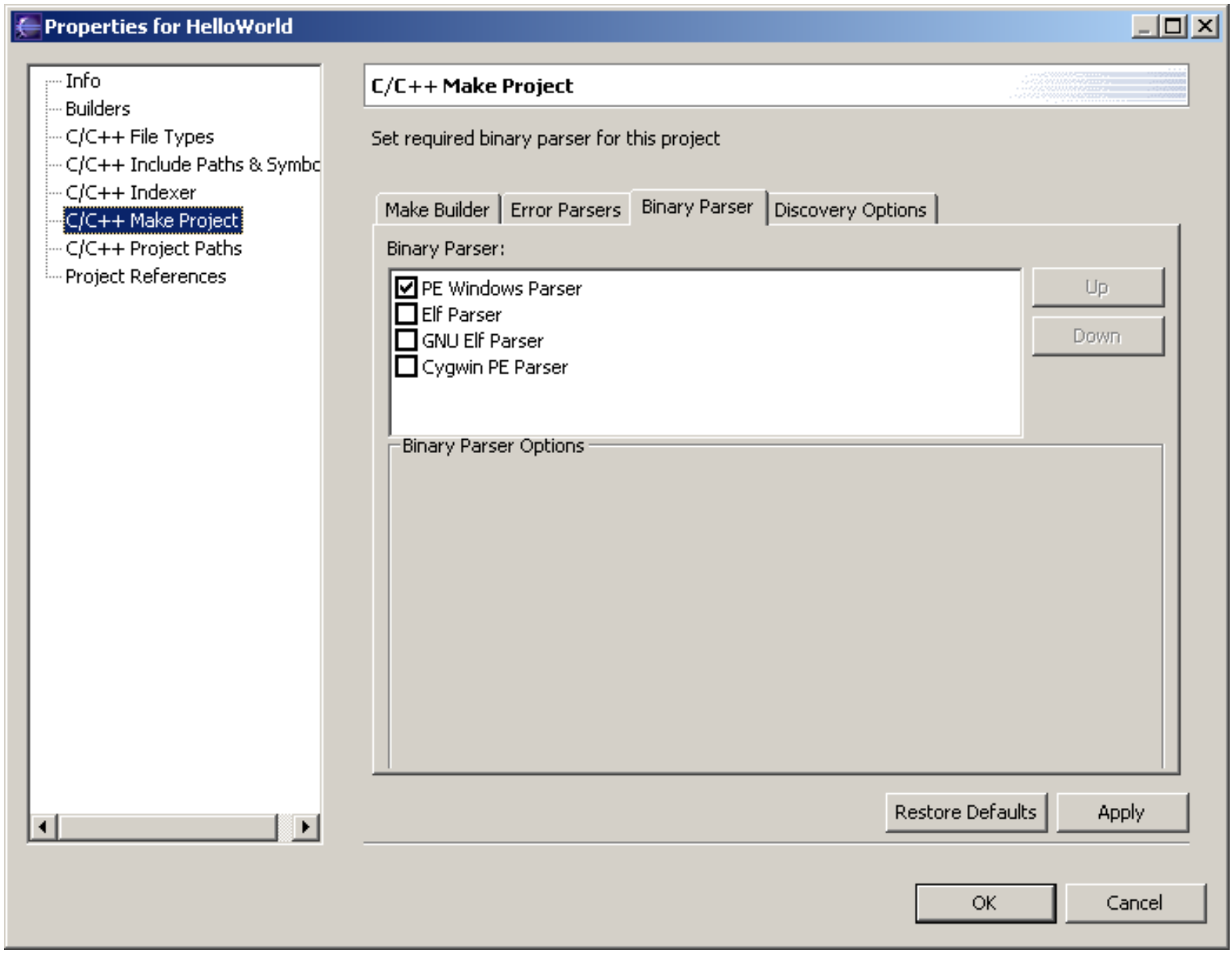
Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard, Binary Parser

You can select the Binary Parsers you require for the project.

To ensure the accuracy of the C/C++ Projects view and the ability to successfully run and debug your programs. After you select the correct parser for your development environment and build your project, you can view the symbols of the .o file in the C/C++ Projects view.



Binary Parser
Select a binary parser from the list.

Binary Parser Options
If a binary parser has parser options you can define them in this section.

Restore Defaults
Returns any changes back to their default setting.

Apply
Applies any changes.

Related concepts

Build overview

Related tasks

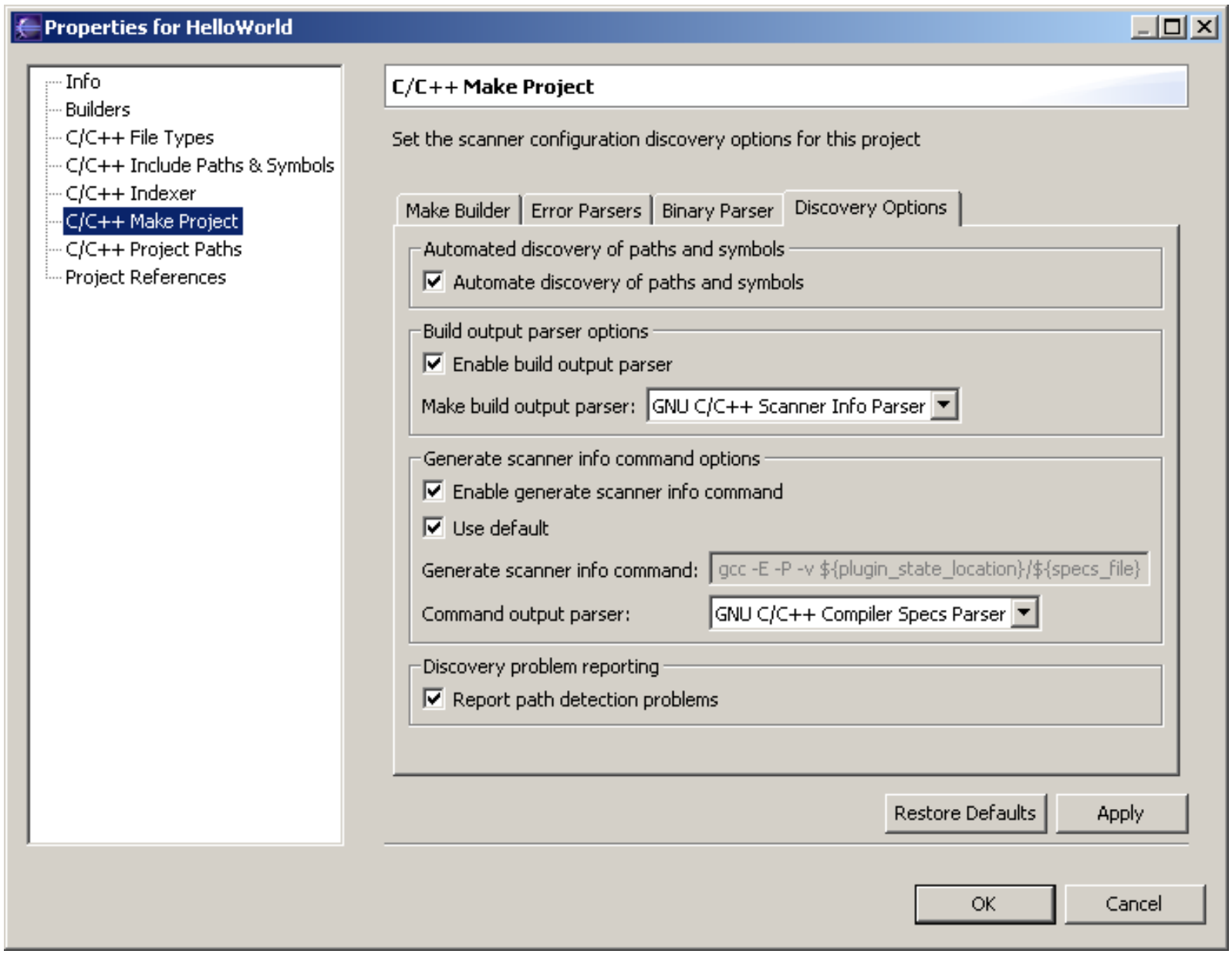
Selecting a binary parser

Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard, Discovery Options

You can define the discovery options on the Discovery Options page of a C/C++ project's properties window.



Automate scanner configuration discovery
Select this checkbox to configure the automatic discovery of paths and symbols.

Build output parser options
Monitors the output of the build to automatically keep the list of include paths and symbols up to date with the makefile.

Generate scanner info command options
This section allows you to select the scanner info settings.

Restore Defaults
Returns any changes back to their default setting.

Apply
Applies any changes.

Related reference

C++ Project Properties, Standard, Info

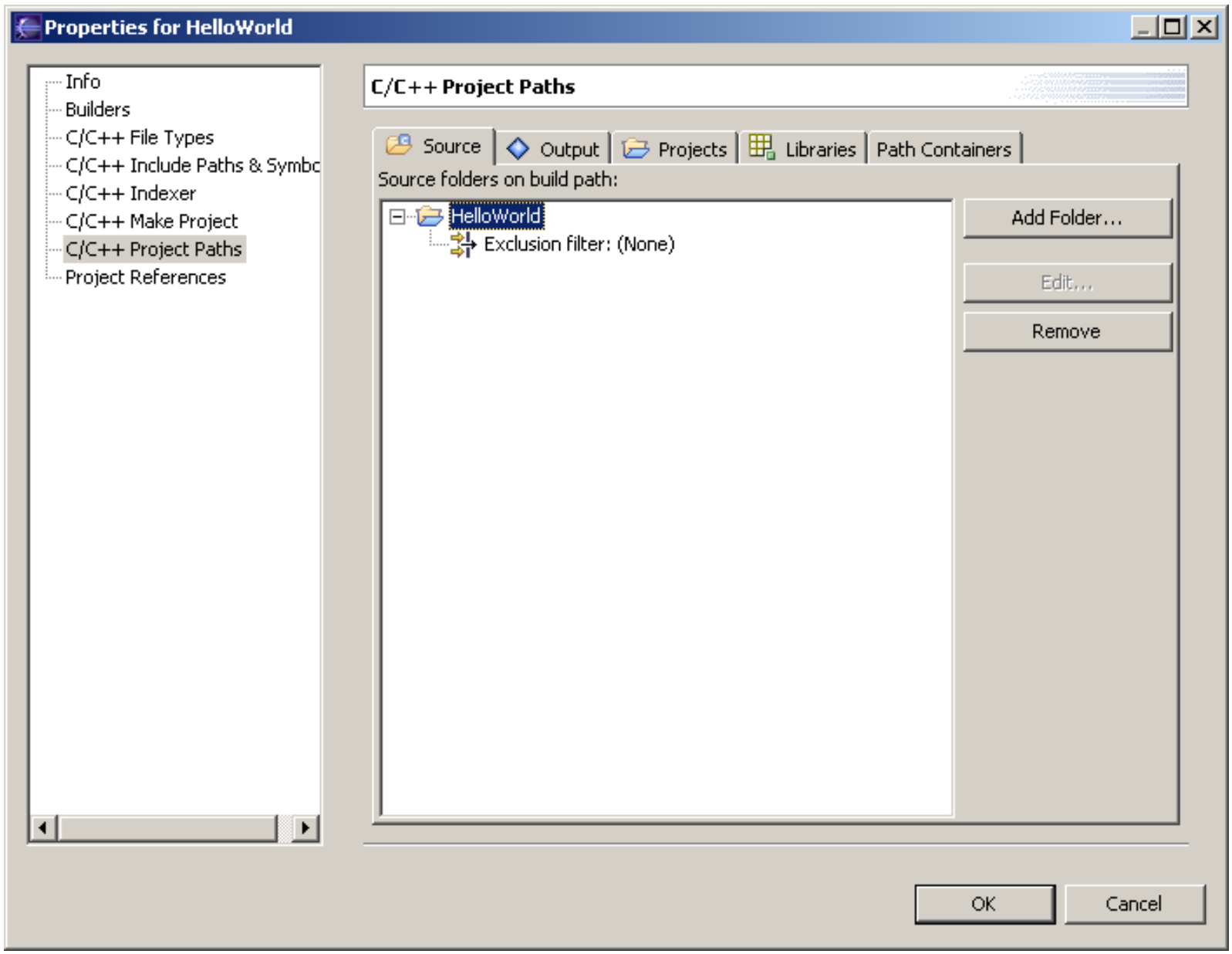
C++ Project Properties, Standard, Builders

C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard, Project Paths

This section describes C/C++ Project Path Properties of a Standard make project.

C/C++ Project Properties, Standard, Project Paths, Source



Add Folder
Add a source folder.

Edit..
Edit a source folder.

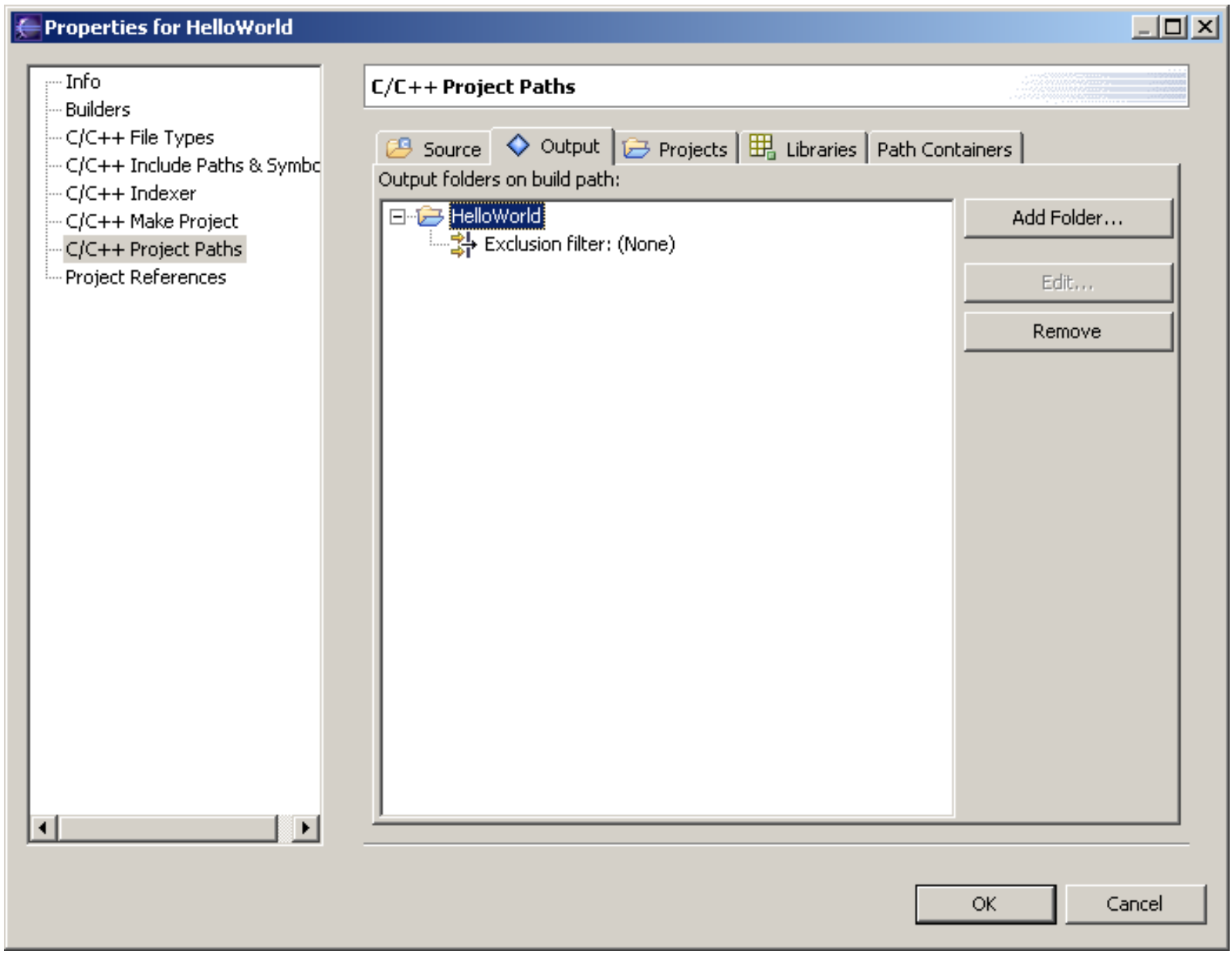
Remove
Remove a source folder.

Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser

C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard, Project Paths, Output



Add Folder
Add an output folder.

Edit..
Edit an output folder.

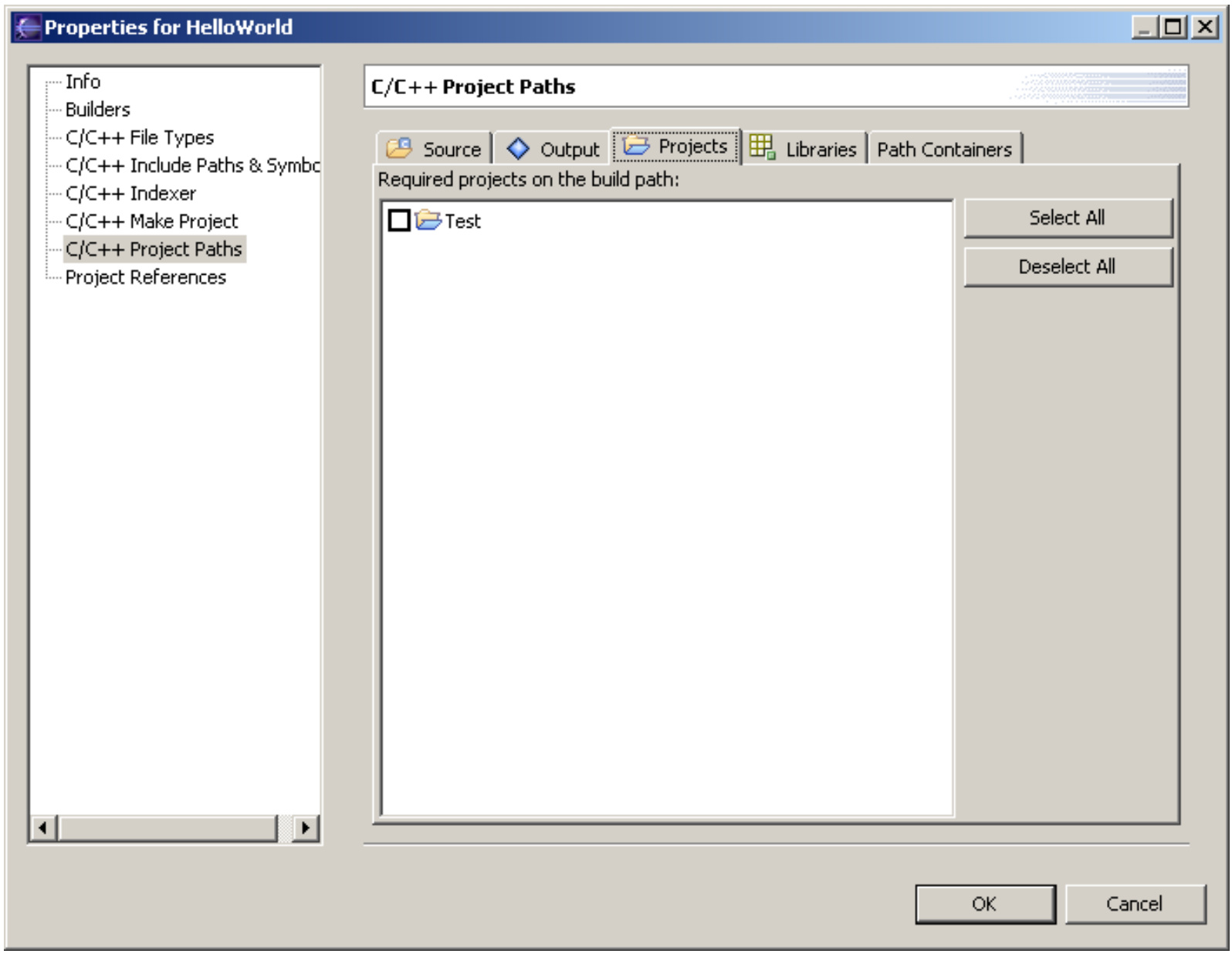
Remove
Remove an output folder.

Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser

C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

C/C++ Project Properties, Standard, Project Paths, Projects



Required projects on the build path
Selecting a project will include any exported paths and symbols from the selected project.

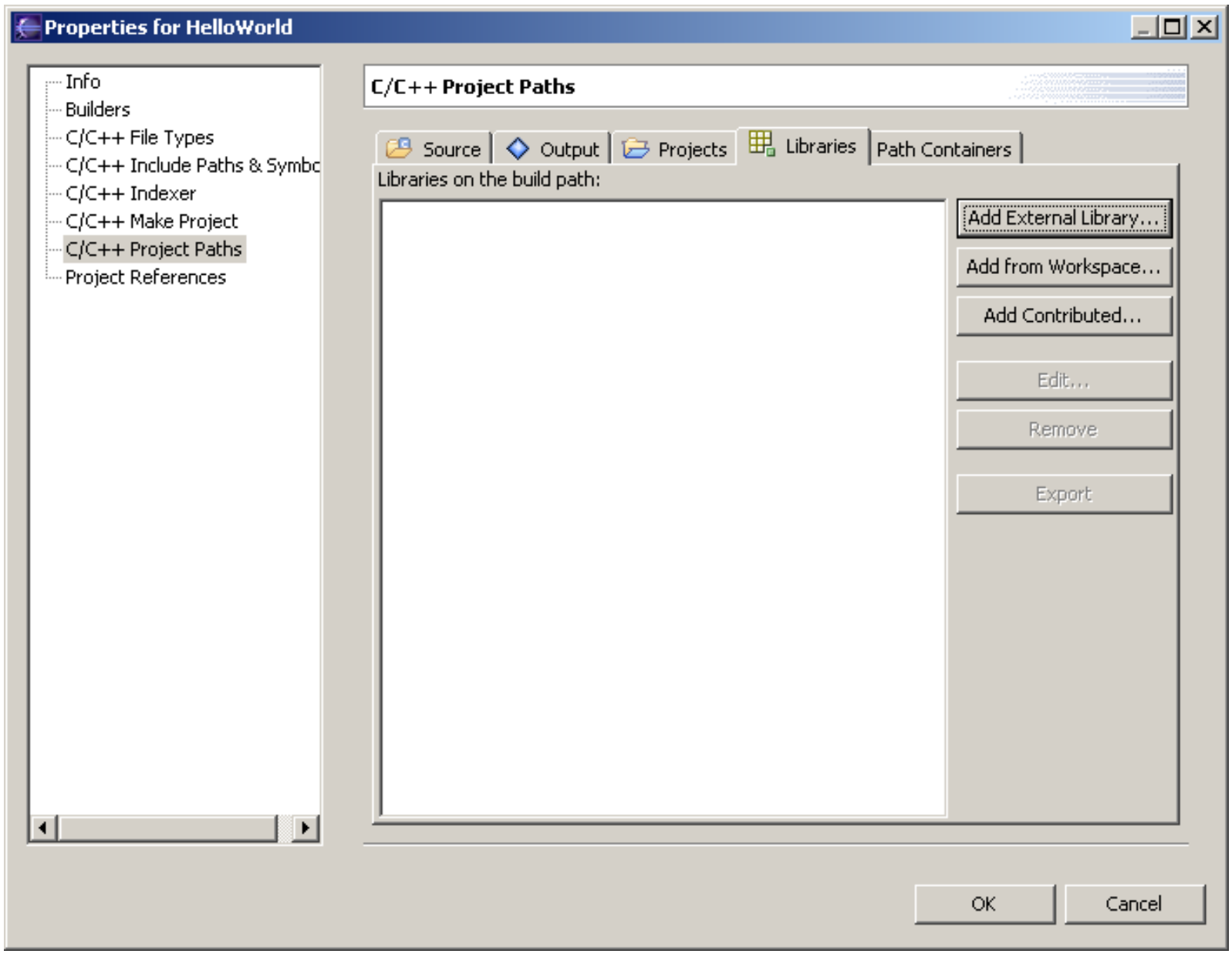
Select All
Click to select all projects listed in the **Required projects on the build path** window.

Deselect All
Click to deselect all projects listed in the **Required projects on the build path** window.

Related reference

[C++ Project Properties, Standard, Info]
[C++ Project Properties, Standard, Builders]
[C++ Project Properties, Standard, File Types]
[C++ Project Properties, Standard, Include Paths and Symbols]
[C++ Project Properties, Standard, Indexer]
[C++ Project Properties, Standard, Make Builder]
[C++ Project Properties, Standard, Error Parser]

C/C++ Project Properties, Standard, Project Paths, Libraries

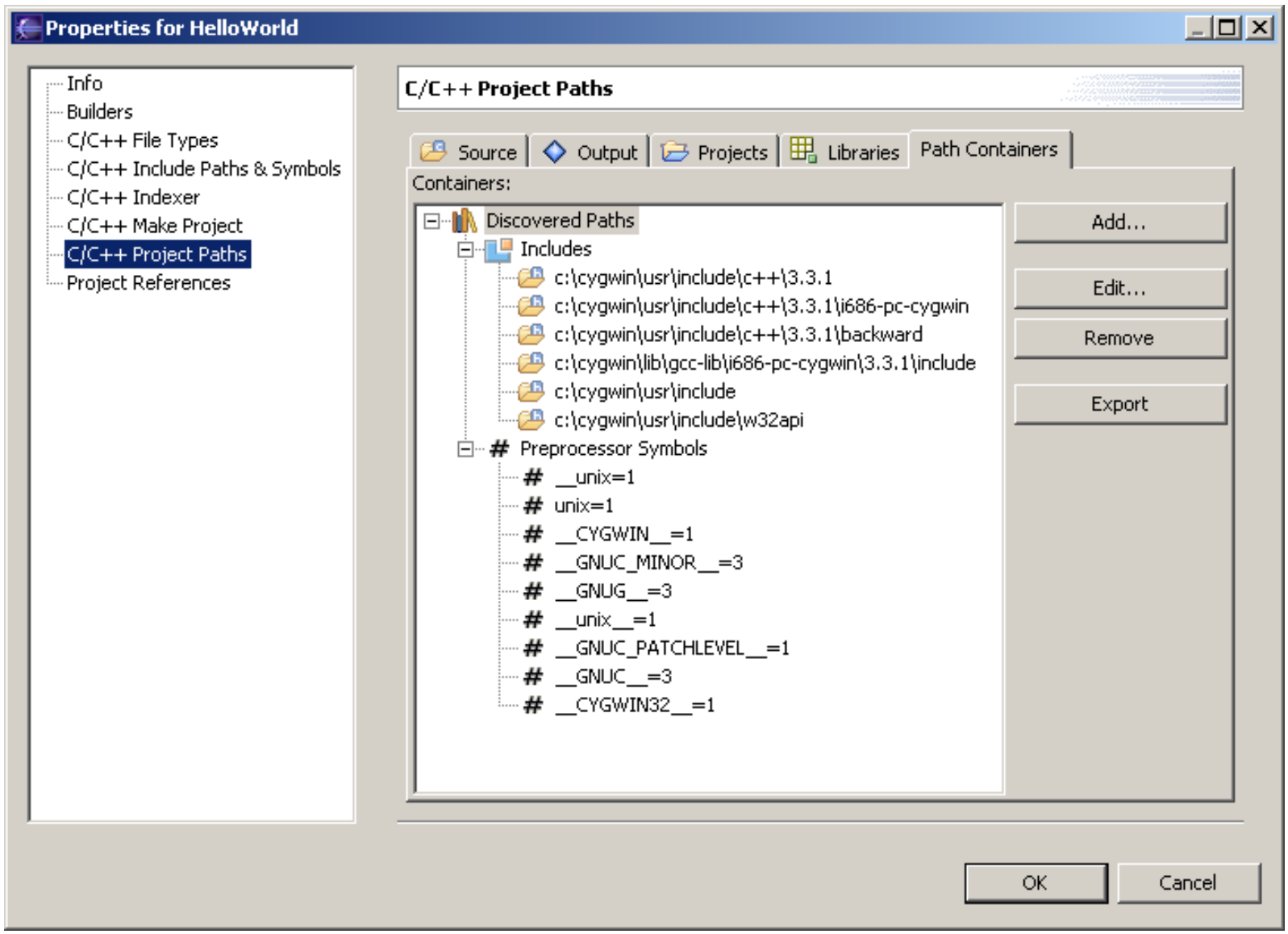


Add External Library...
Add and external library.
Add from Workspace...
Add a library from a project in your workspace.
Add Contributed...
Add a contributed library.
Edit...
Edit the currently selected library.
Remove
Remove the currently selected library.
Export
Export the currently selected library to a text file.

Related reference
C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders

C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output
C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

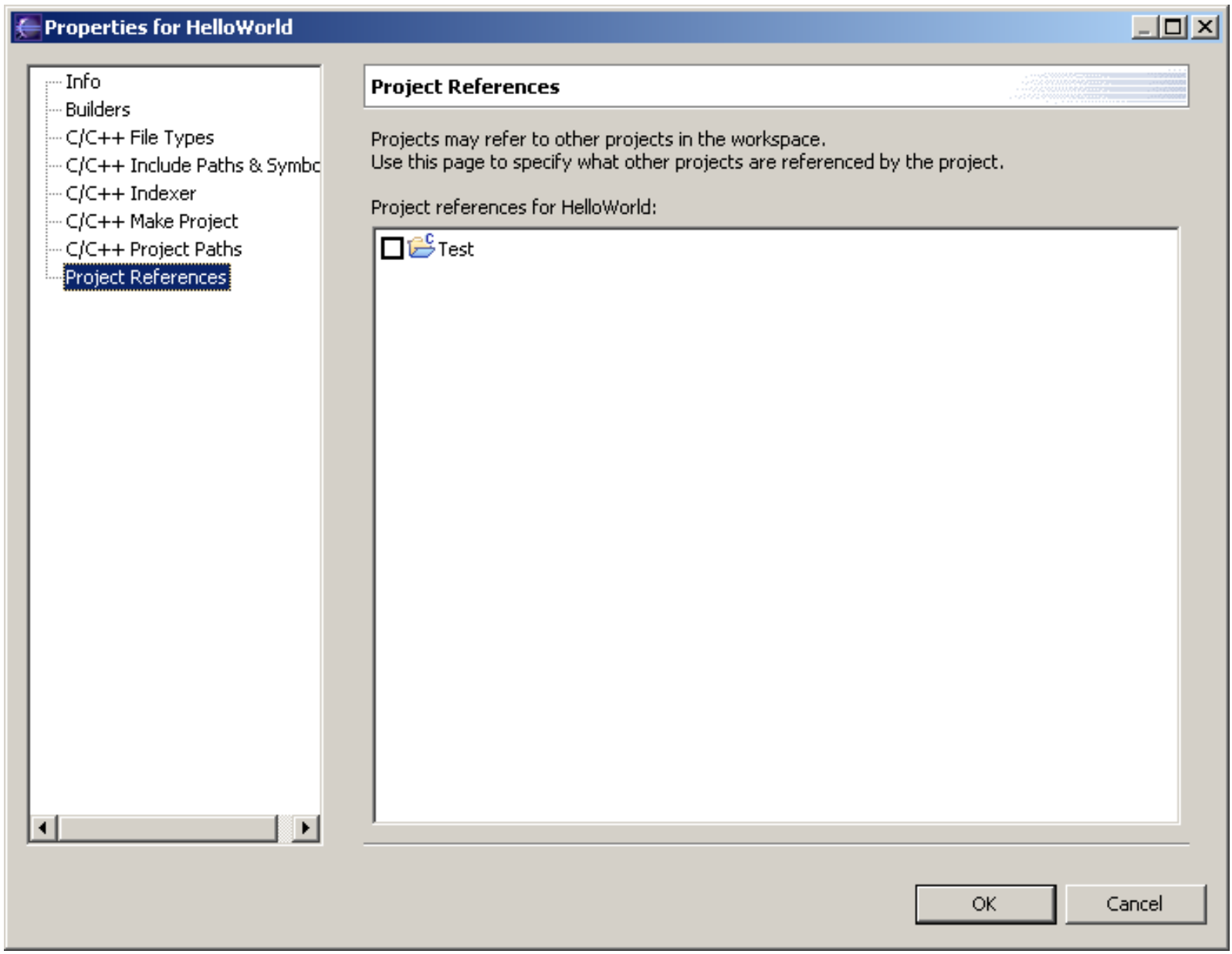
C/C++ Project Properties, Standard, Project Paths, Path Containers



Add....
Add a new path container.
Edit...
Edit the currently selected path container.
Remove
Remove the currently selected path container.
Export
Export the currently selected path container to a text file.

Related reference
C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser

C/C++ Project, Standard, Project References



Project references for <project>:

Select the projects required to build this project.

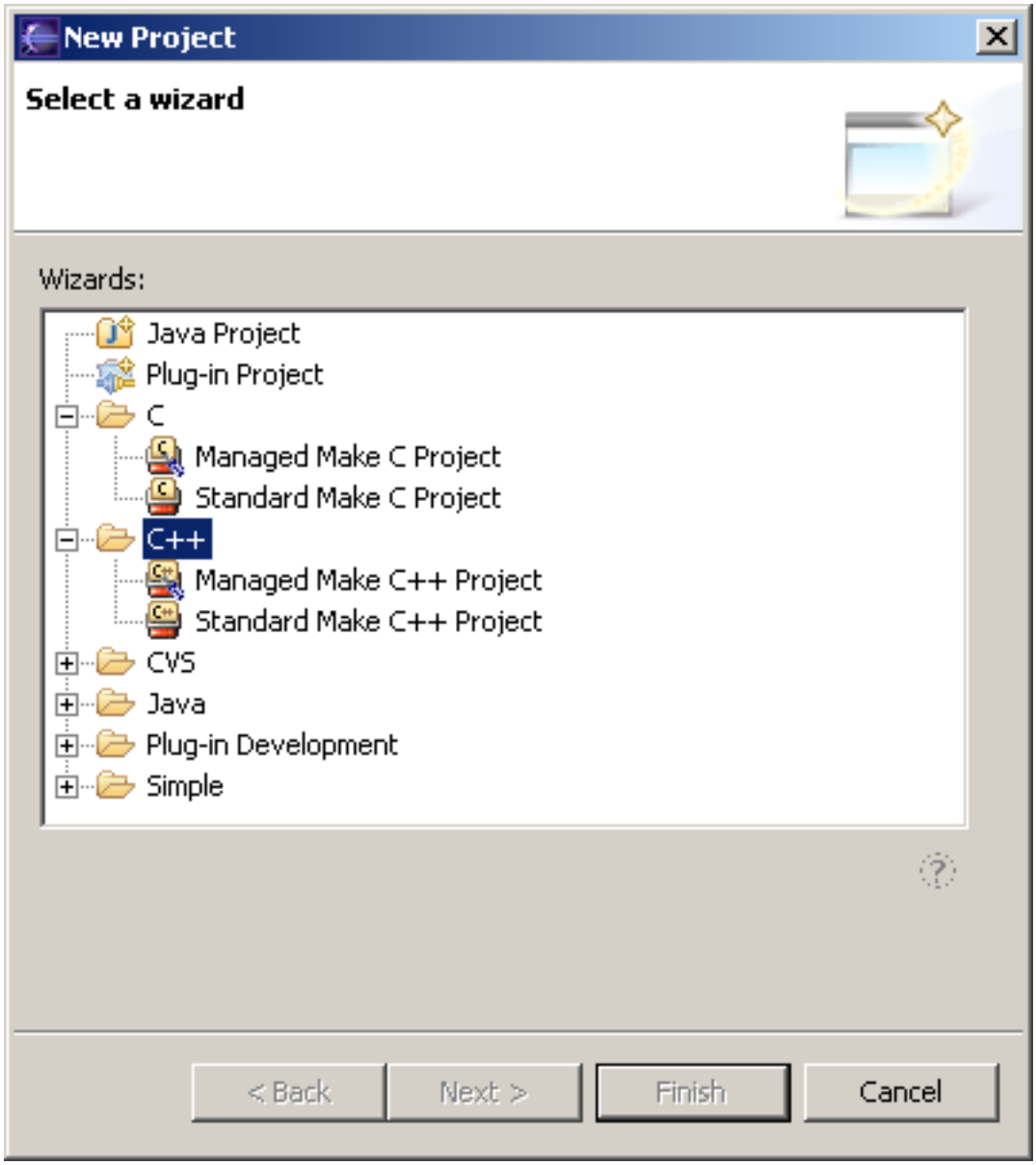
Related reference

C++ Project Properties, Standard, Info
C++ Project Properties, Standard, Builders
C++ Project Properties, Standard, File Types
C++ Project Properties, Standard, Include Paths and Symbols
C++ Project Properties, Standard, Indexer
C++ Project Properties, Standard, Make Builder
C++ Project Properties, Standard, Error Parser
C++ Project Properties, Standard, Binary Parser
C++ Project Properties, Standard, Discovery Options
C++ Project Properties, Standard, Source
C++ Project Properties, Standard, Output

C++ Project Properties, Standard, Projects
C++ Project Properties, Standard, Libraries
C++ Project Properties, Standard, Path Containers
C++ Project Properties, Standard, Project References
C++ Project Properties, Managed, Info
C++ Project Properties, Managed, Builders
C++ Project Properties, Managed, Build
C++ Project Properties, Managed, File Types
C++ Project Properties, Managed, Indexer
C++ Project Properties, Managed, Error Parser
C++ Project Properties, Managed, Project References

New Project Wizard

The **New Project** wizard helps you create a new C or C++ project in the workbench. To access the wizard, from the menu bar select **File > New > Project**. The **New Project** wizard appears:



With the **New Project** wizard you can choose to:

- Create a **Managed Make C Project**
- Create a **Standard Make C Project**
- Create a **Managed Make C++ Project**
- Create a **Standard Make C++ Project**

Related concepts
CDT projects

Related tasks

CDT Managed Make Tutorial

CDT Standard Make Tutorial

Related reference

Managed Make, Name

Managed Make, Select a Target

Managed Make, Referenced Projects

Managed Make, Error Parsers

Managed Make, C/C++ Indexer

Standard Make, Name

Standard Make, Referenced Projects

Standard Make, Make Builder

Standard Make, Error Parsers

Standard Make, Binary Parser

Standard Make, Discovery Options

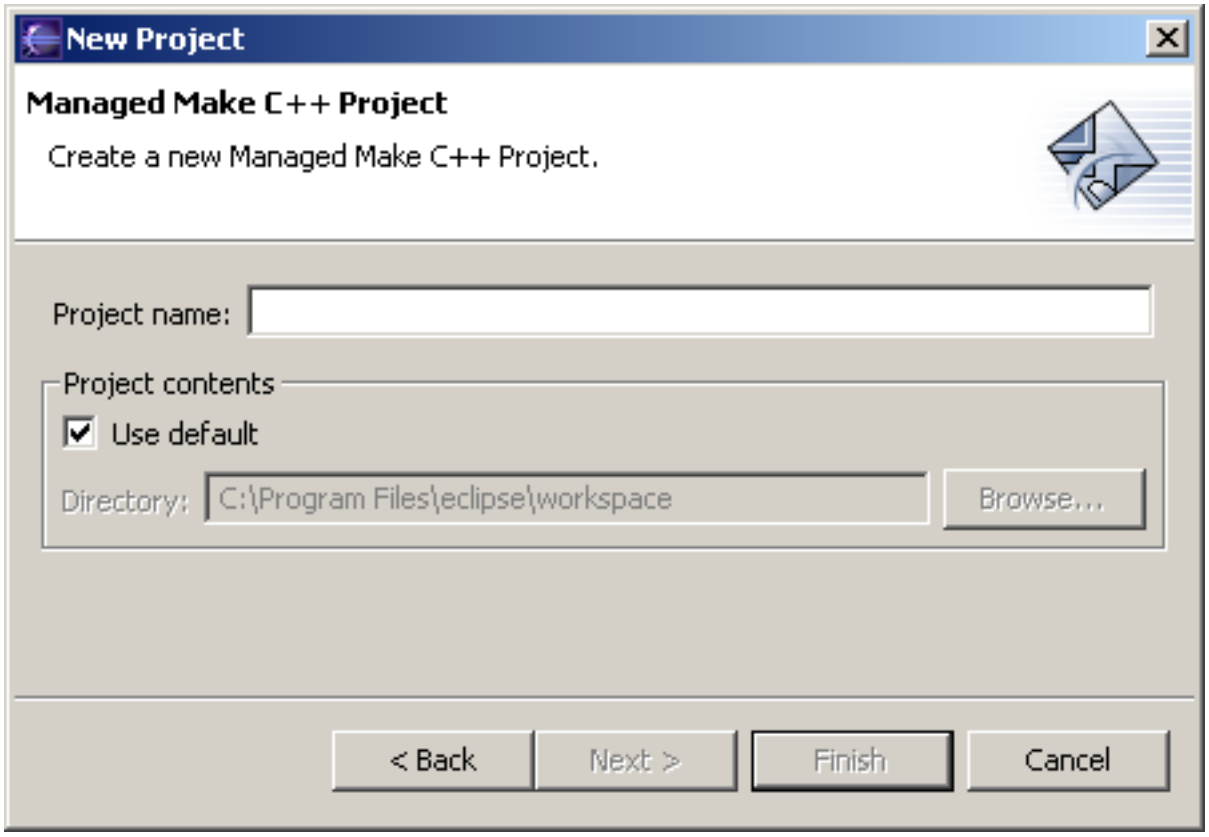
Standard Make, C/C++ Indexer

C/C++ New Project Wizard, Managed Make Project

This section describes properties for creating a Managed make project in the **C/C++ New Project Wizard**.

New Project Wizard - Managed Make, Name

Select a name for the project. You can also enter a new path for your project by deselecting the **Use Default Location** checkbox and entering the new path in the **Location** text box.



Name	Function
Name	Specifies the name of the project.
Use Default Location	When selected the new project will be created in the default workspace location.
Location	If **Use Default Location** is not selected, enter the location where the project is to be created.

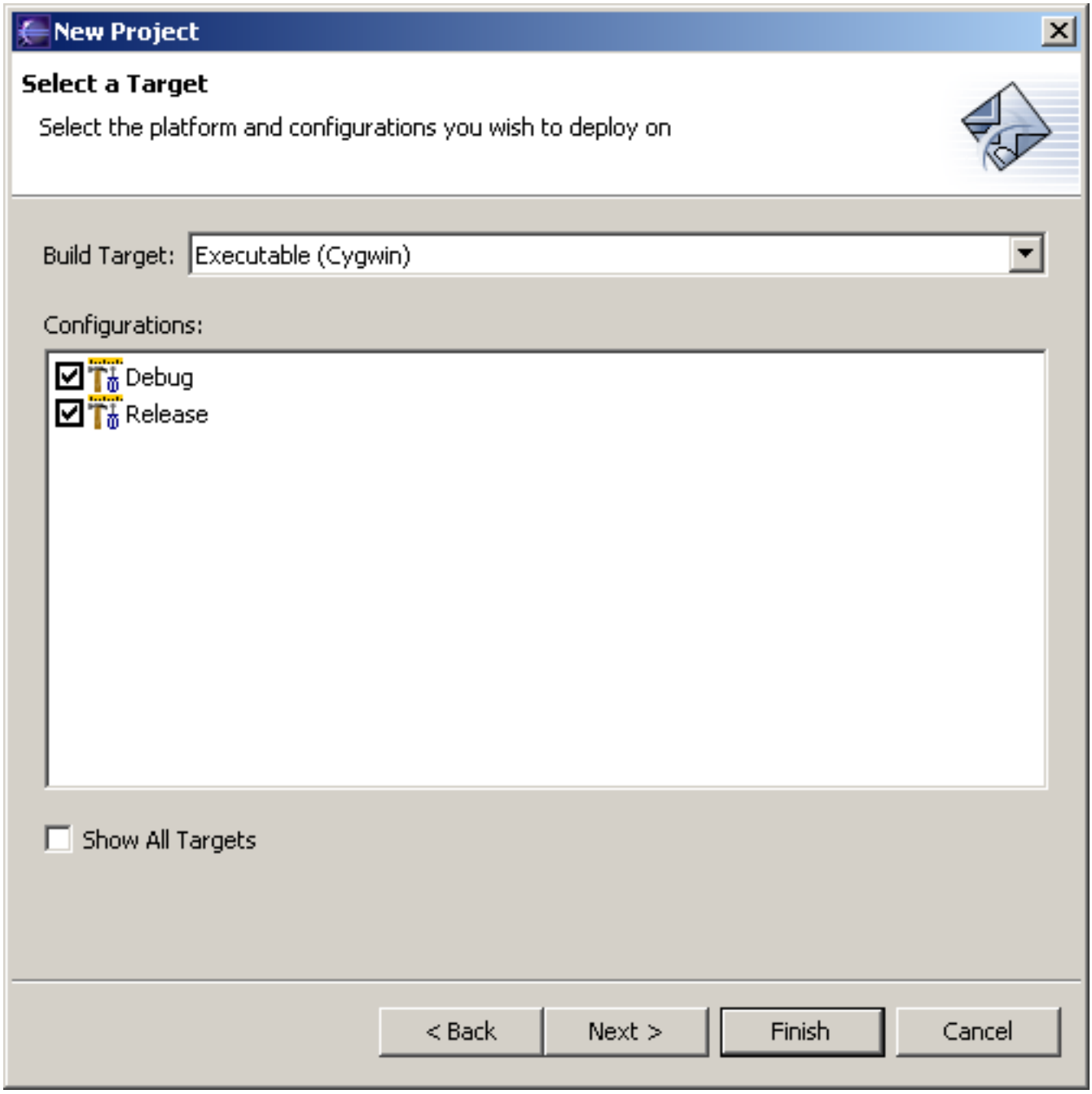
Related concepts
CDT projects

Related tasks
CDT Managed Make Tutorial
CDT Standard Make Tutorial

Related reference
New Project Wizard
Managed Make, Select a Target

New Project Wizard - Managed Make, Select a Target

You can enter the build targets and build configurations from this page of the wizard.



Name	Function
Build Target	You can select a build target from the drop down list.
Configurations	Specifies which build configurations will be supported for your project.
Show All Targets	If selected, all known targets will appear in the **Configurations** window, by default, the list is filtered so that only targets for the host OS are shown.

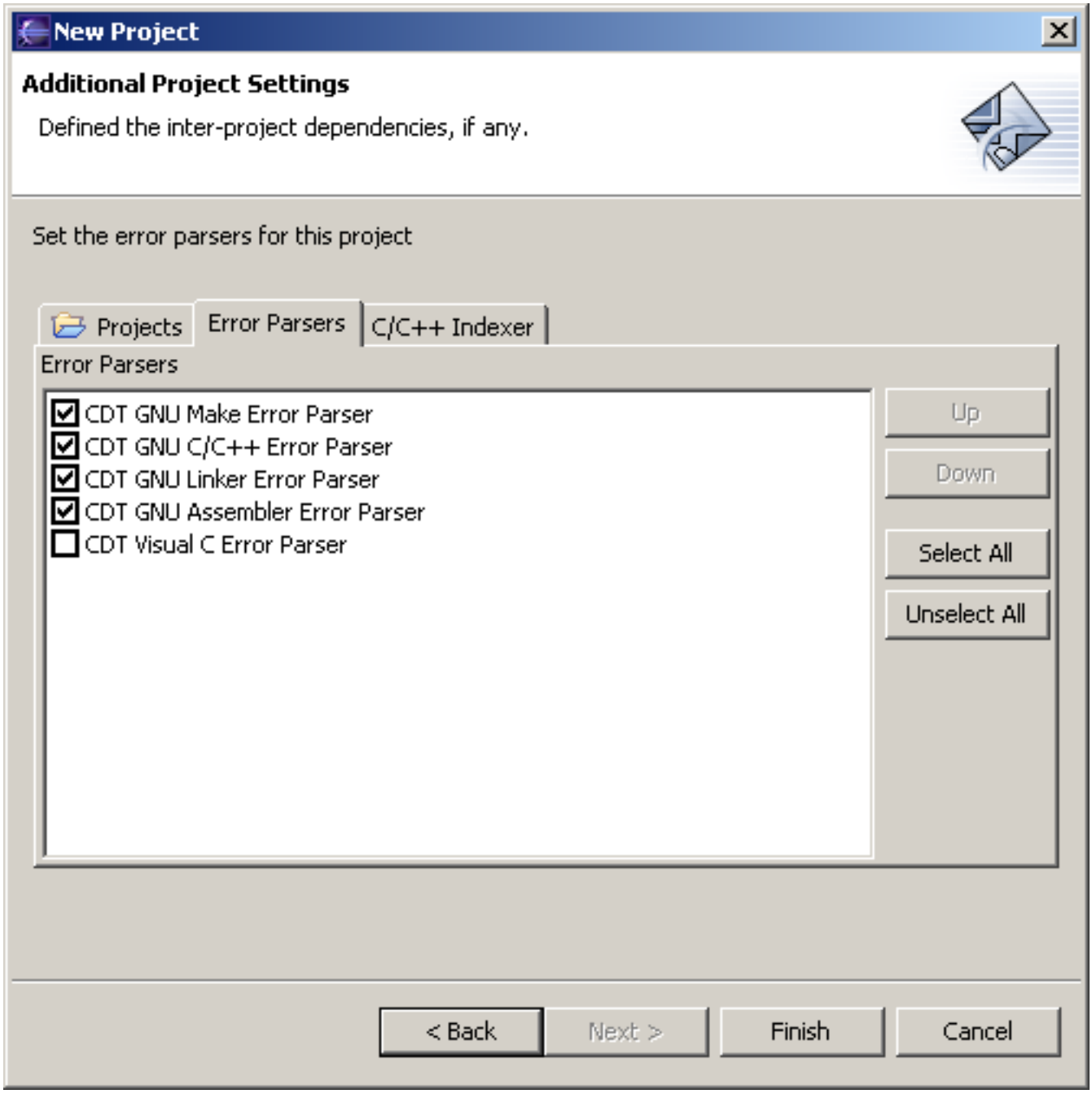
Related concepts
CDT projects

Related tasks
CDT Managed Make Tutorial
CDT Standard Make Tutorial

Related reference
New Project Wizard
Managed Make, Referenced Projects
Managed Make, Error Parsers
Managed Make, C/C++ Indexer
Standard Make, Name
Standard Make, Referenced Projects
Standard Make, Make Builder
Standard Make, Error Parsers
Standard Make, Binary Parser
Standard Make, Discovery Options
Standard Make, C/C++ Indexer

New Project Wizard - Managed Make, Error Parsers

You can select which error parsers to use and in which order they are used for your project.



Name	Function
Error Parsers	You can select which Error Parsers to enable from this window.
Up	Moves the currently selected Error Parser higher in the ordered list.
Down	Moves the currently selected Error Parser lower in the ordered list.
Select All	Selects all Error Parsers.
Unselect All	Unselects all Error Parsers.

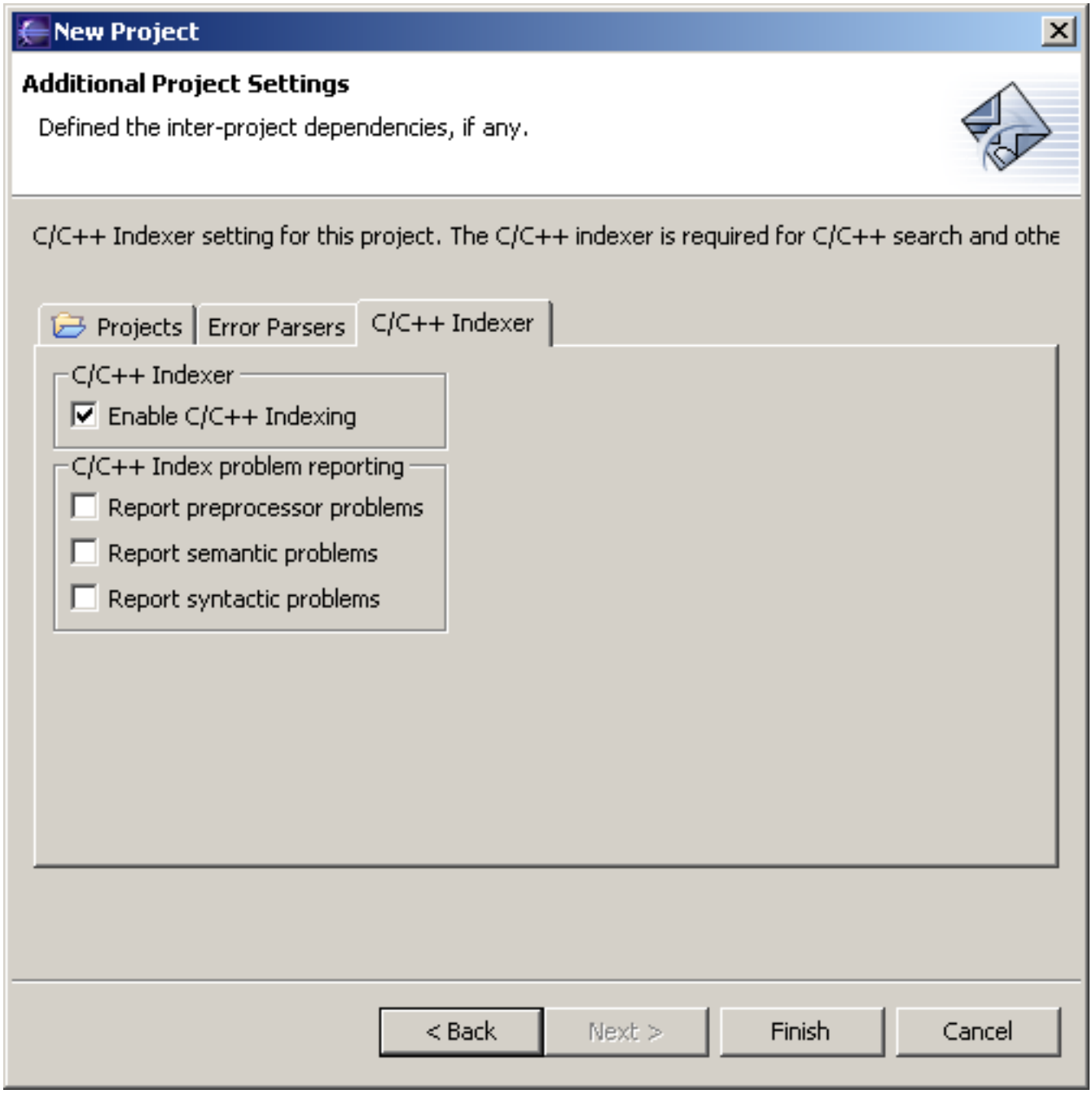
Related concepts
CDT projects

Related tasks
CDT Managed Make Tutorial
CDT Standard Make Tutorial

Related reference
New Project Wizard
Managed Make, Select a Target
Managed Make, Referenced Projects
Managed Make, C/C++ Indexer
Standard Make, Name
Standard Make, Referenced Projects
Standard Make, Make Builder
Standard Make, Error Parsers
Standard Make, Binary Parser
Standard Make, Discovery Options
Standard Make, C/C++ Indexer

New Project Wizard - Managed Make, C/C++ Indexer

You can select which C/C++ Indexers to use for your project from this page of the wizard. The indexer is necessary for search and related features, like content assist.



Name	Function
Enable C++ Indexing	When selected C++ Indexing is enabled for this project.
Enable C++ Index problem reporting	When selected, any index related errors detected will be reported.

Related concepts

CDT projects

Related tasks

CDT Managed Make Tutorial
CDT Standard Make Tutorial

Related reference

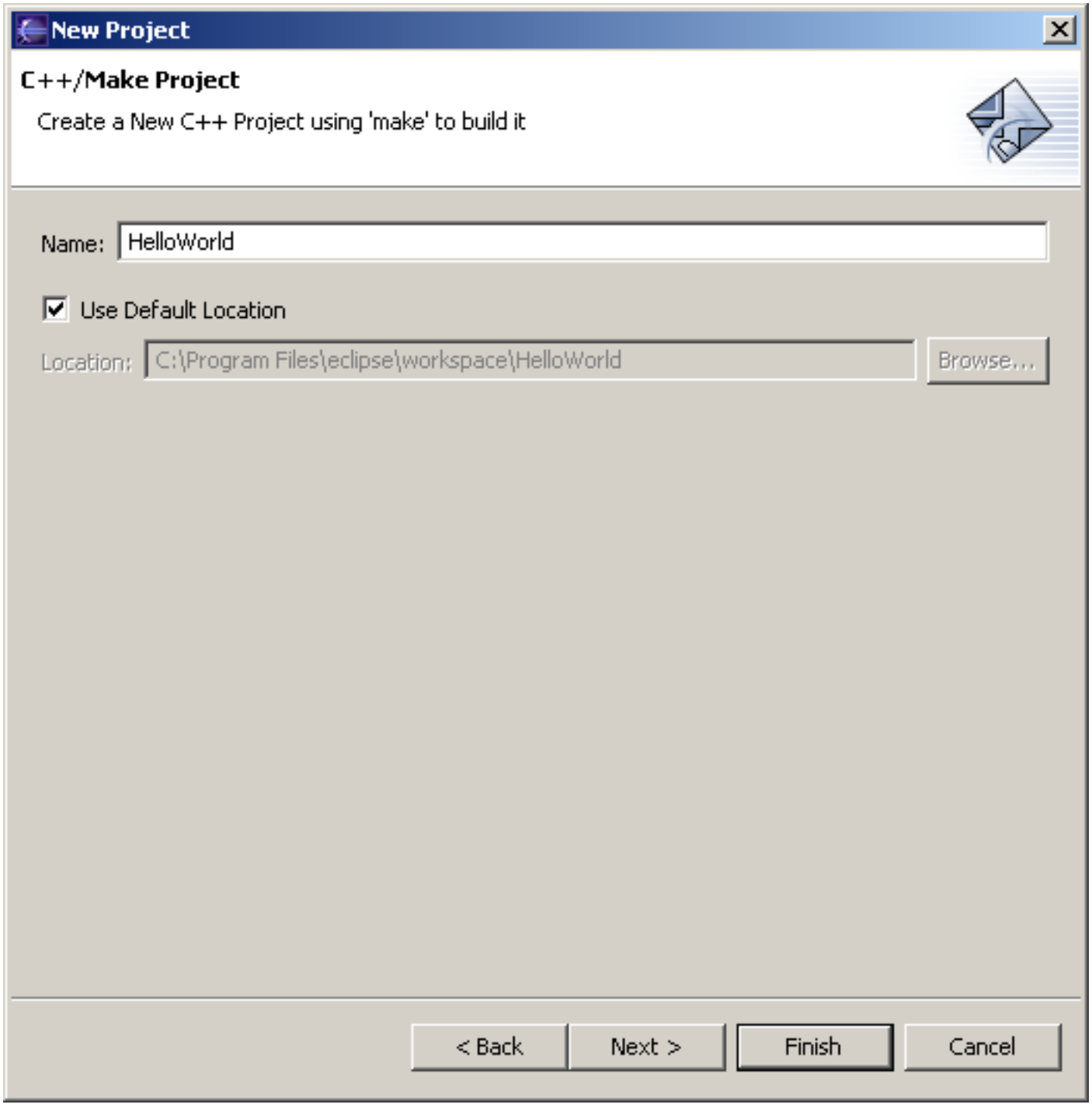
New Project Wizard
Managed Make, Select a Target
Managed Make, Referenced Projects
Managed Make, Error Parsers
Standard Make, Name
Standard Make, Referenced Projects
Standard Make, Make Builder
Standard Make, Error Parsers
Standard Make, Binary Parser
Standard Make, Discovery Options
Standard Make, C/C++ Indexer

C/C++ New Project Wizard, Standard Make Project

This section describes properties for creating a Standard make project in the **C/C++ New Project Wizard**.

New Project Wizard - Standard Make, Name

Select a name for the project. You can also enter a new path for your project by deselecting the **Use Default Location** checkbox and entering the new path in the **Location** text box.



Name	Function
Name	Specifies the name of the project.
Use Default Location	When selected the new project will be created in the default workspace location.

Location	If **Use Default Location** is not selected, enter the location where the project is to be created.

Related concepts

CDT projects

Related tasks

CDT Managed Make Tutorial

CDT Standard Make Tutorial

Related reference

New Project Wizard

Managed Make, Select a Target

Managed Make, Referenced Projects

Managed Make, Error Parsers

Managed Make, C/C++ Indexer

Standard Make, Referenced Projects

Standard Make, Make Builder

Standard Make, Error Parsers

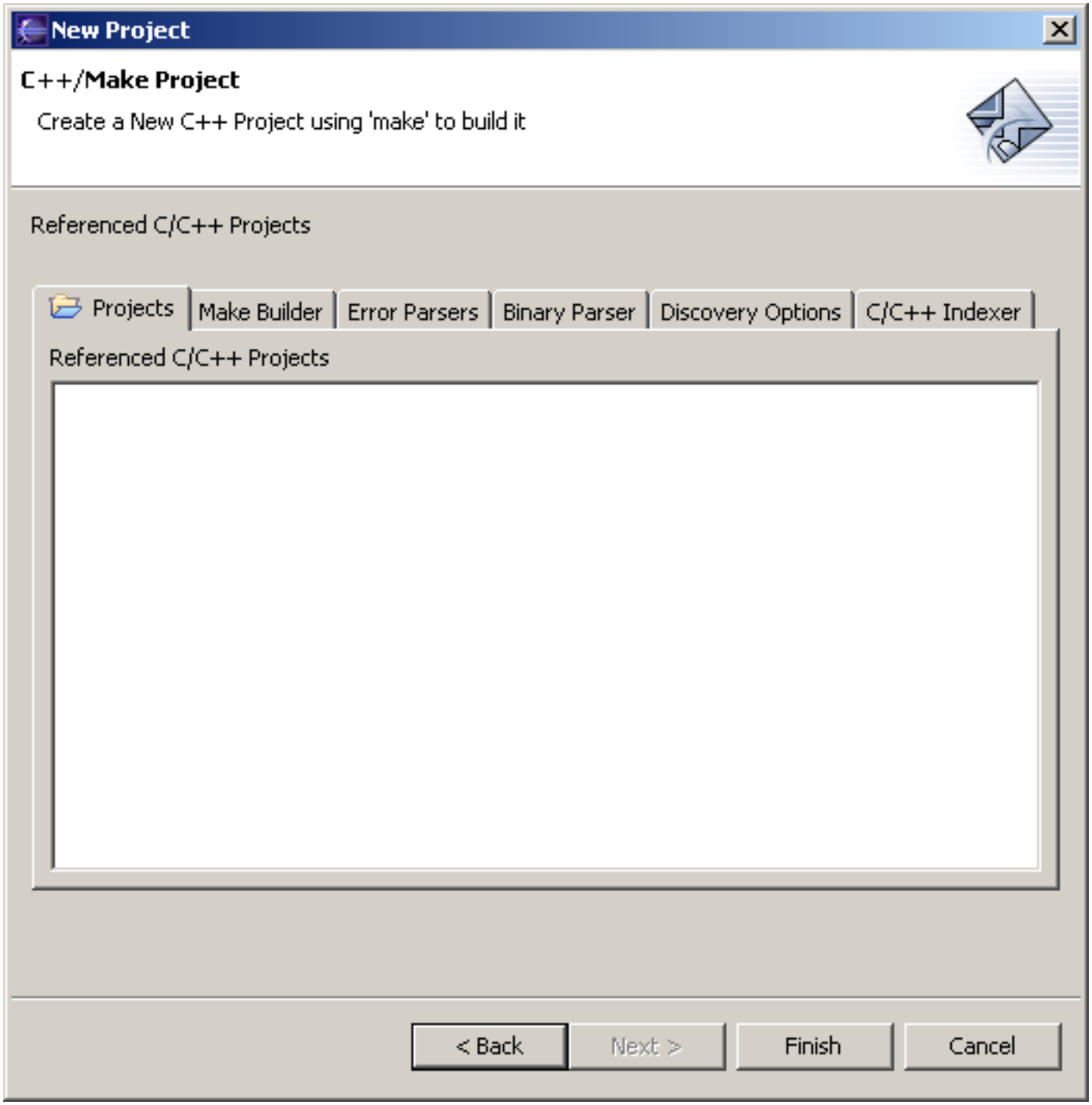
Standard Make, Binary Parser

Standard Make, Discovery Options

Standard Make, C/C++ Indexer

New Project Wizard - Standard Make, Referenced Projects

Select project references from your workspace.



Name	Function
Select Project References	If you have any other projects in your workspace, you can select them as references for this new C or C++ project.

Related concepts

CDT projects

Related tasks

CDT Managed Make Tutorial

CDT Standard Make Tutorial

Related reference

New Project Wizard

Managed Make, Select a Target

Managed Make, Referenced Projects

Managed Make, Error Parsers

Managed Make, C/C++ Indexer

Standard Make, Name

Standard Make, Make Builder

Standard Make, Error Parsers

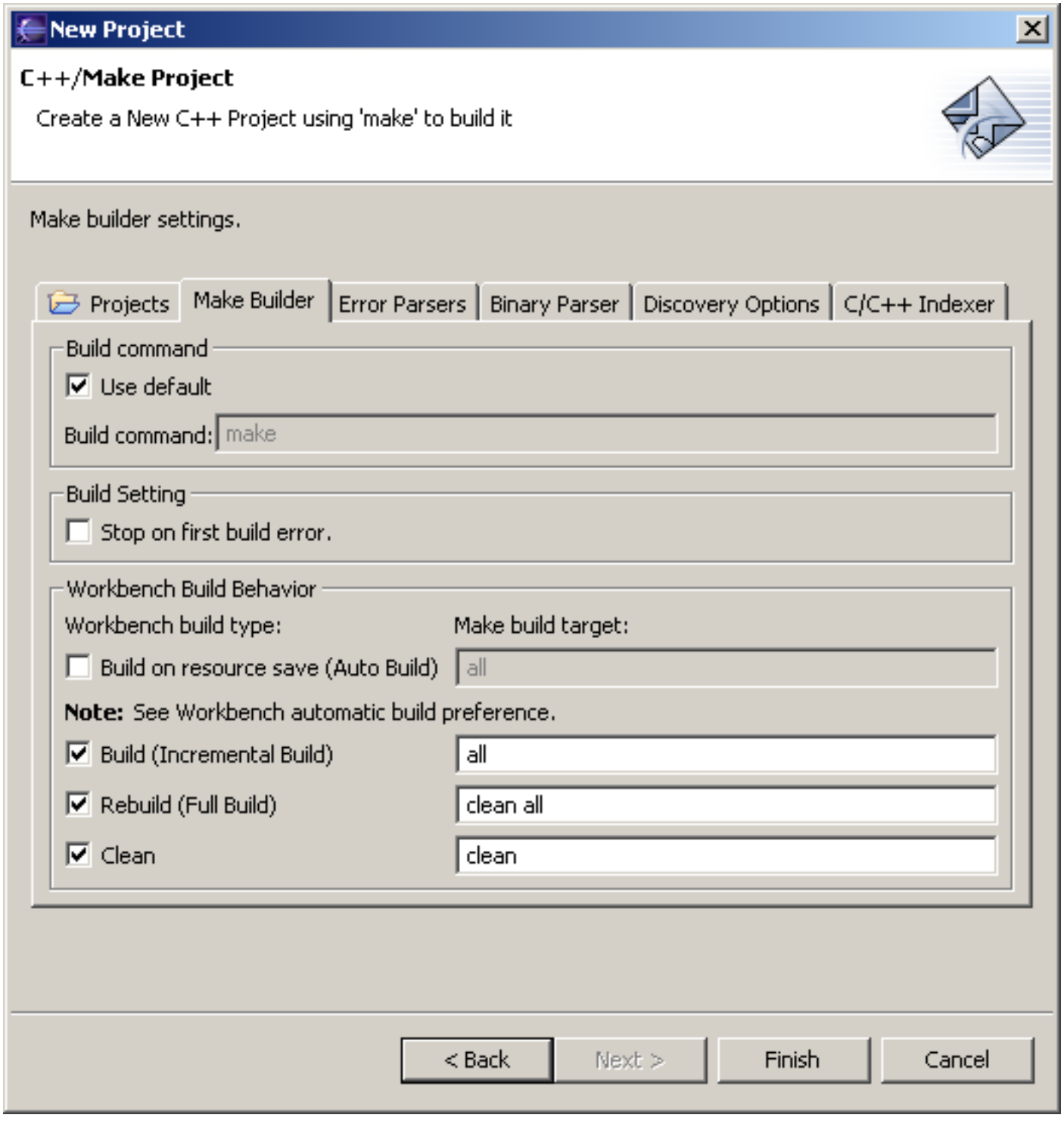
Standard Make, Binary Parser

Standard Make, Discovery Options

Standard Make, C/C++ Indexer

New Project Wizard - Standard Make, Make Builder

You can enter build configuration settings for your project from this page of the wizard.



Name	Function
Build Command	Specifies which **make** command to use when a build is performed.

Build Setting	Controls whether the compiler will **Stop On Error** or to keep building. If you do not select **Stop On Error** this will force the compiler to attempt to build all referenced projects even if the current project has errors.
Workbench Build Behavior	You can specify the behavior of builds predefined by the C/C++ toolkit by linking them to specific build targets in your makefile. The predefined builds are **Auto Build**, **Incremental Build**, **Rebuild** and, **Clean**

Related concepts

CDT projects

Related tasks

CDT Managed Make Tutorial

CDT Standard Make Tutorial

Related reference

New Project Wizard

Managed Make, Select a Target

Managed Make, Referenced Projects

Managed Make, Error Parsers

Managed Make, C/C++ Indexer

Standard Make, Name

Standard Make, Referenced Projects

Standard Make, Error Parsers

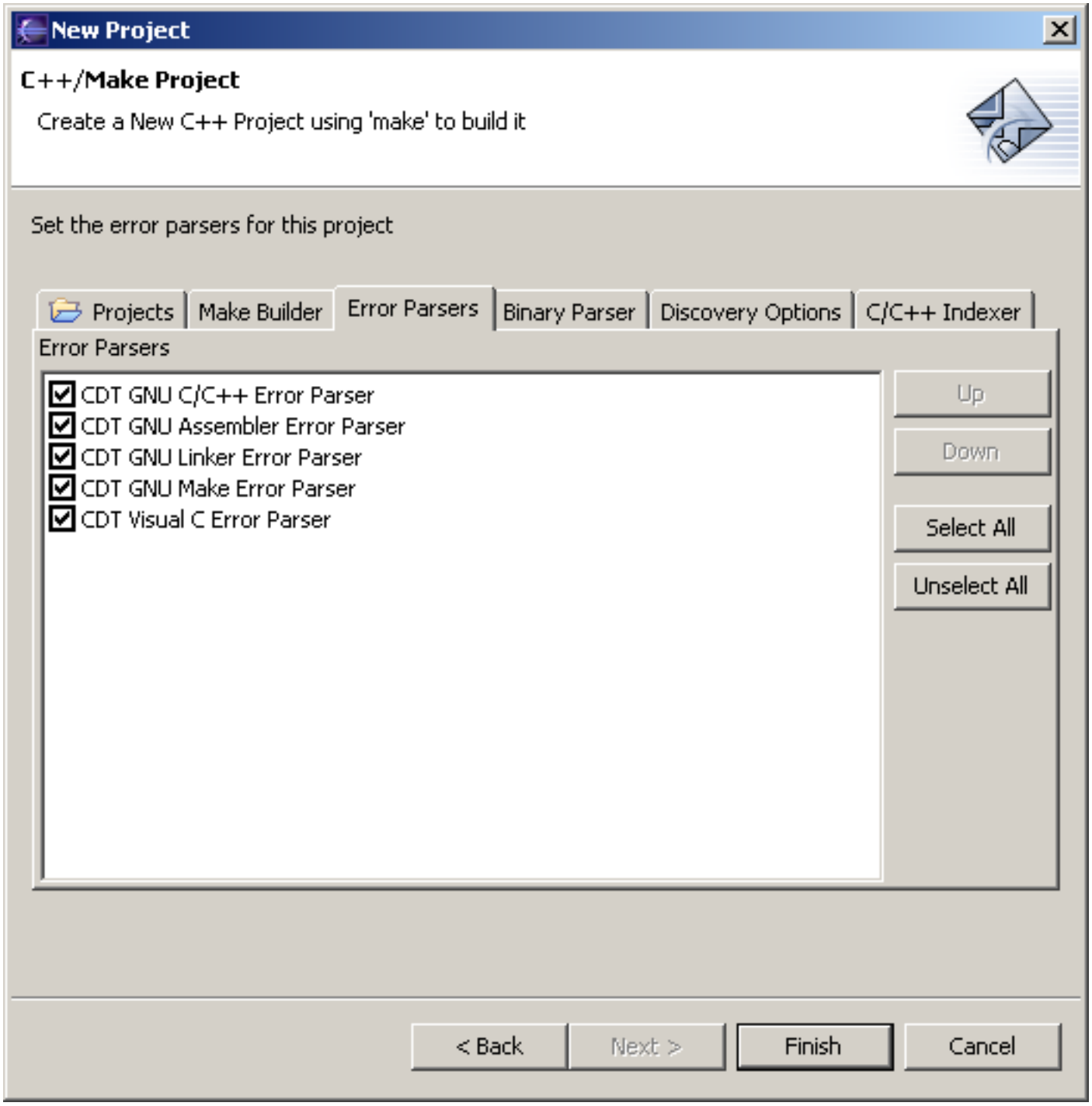
Standard Make, Binary Parser

Standard Make, Discovery Options

Standard Make, C/C++ Indexer

New Project Wizard - Standard Make, Error Parsers

You can select which error parsers to use and in which order they are used for your project.



Name	Function
Error Parsers	You can select which Error Parsers to enable from this window.
Up	Moves the currently selected Error Parser higher in the ordered list.
Down	Moves the currently selected Error Parser lower in the ordered list.
Select All	Selects all Error Parsers.

Unselect All	Unselects all Error Parsers.

Related concepts

CDT projects

Related tasks

CDT Managed Make Tutorial

CDT Standard Make Tutorial

Related reference

New Project Wizard

Managed Make, Select a Target

Managed Make, Referenced Projects

Managed Make, Error Parsers

Managed Make, C/C++ Indexer

Standard Make, Name

Standard Make, Referenced Projects

Standard Make, Make Builder

Standard Make, Binary Parser

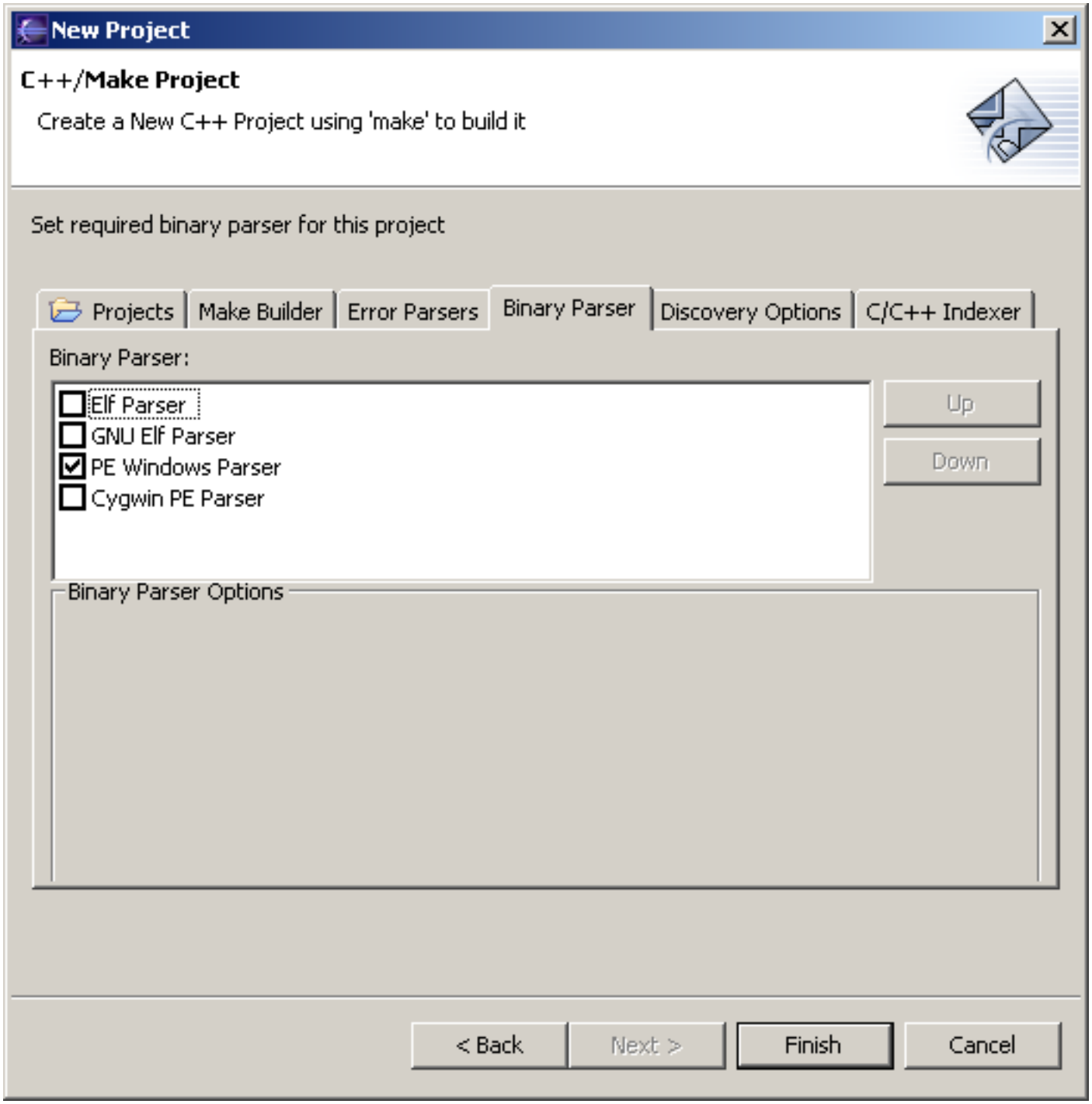
Standard Make, Discovery Options

Standard Make, C/C++ Indexer

New Project Wizard - Standard Make, Binary Parser

You can select the Binary Parsers you require for the project.

Select the appropriate binary parser to ensure the accuracy of the C/C++ Projects view and the ability to successfully run and debug your programs. After you select the correct parser for your development environment and build your project, you can view the symbols of the .o file in the C/C++ Projects view.



Name	Function

Binary Parser	Select a binary parser from the list.
Binary Parser Options	If a binary parser has parser options you can define them in this section.

Related concepts

CDT projects

Related tasks

CDT Managed Make Tutorial

CDT Standard Make Tutorial

Related reference

New Project Wizard

Managed Make, Select a Target

Managed Make, Referenced Projects

Managed Make, Error Parsers

Managed Make, C/C++ Indexer

Standard Make, Name

Standard Make, Referenced Projects

Standard Make, Make Builder

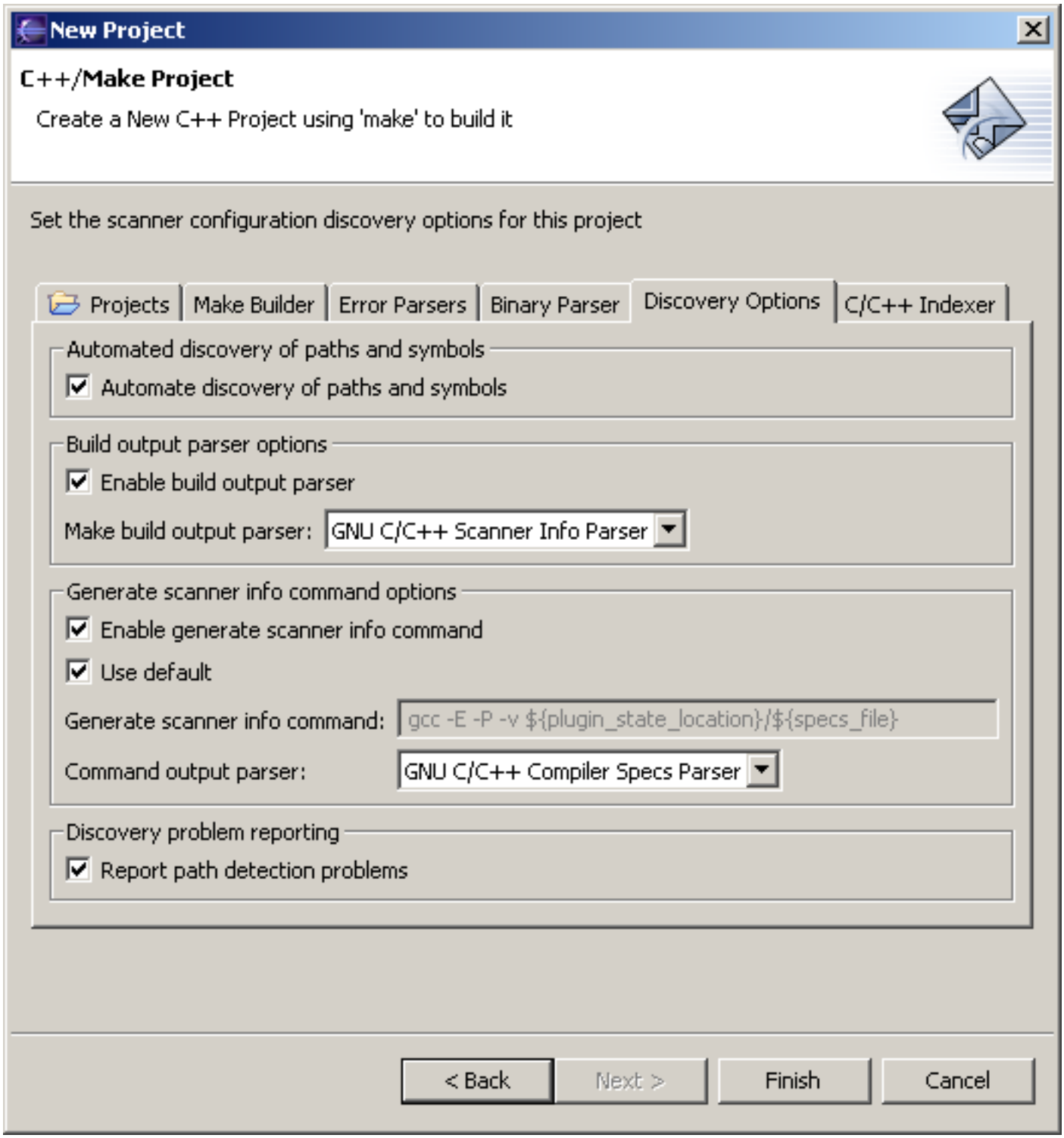
Standard Make, Error Parsers

Standard Make, Discovery Options

Standard Make, C/C++ Indexer

New Project Wizard - Standard Make, Discovery Options

You can define the discovery options on the Discovery Options page of the C/C++ Preferences window.



Name	Function

Automate scanner configuration discovery	Select this checkbox to configure the scanner discovery to run automatically.
Build output parser options	This section allows you to select the output parser.
Generate scanner info command options	This section allows you to select the scanner info settings.

Related concepts

CDT projects

Related tasks

CDT Managed Make Tutorial

CDT Standard Make Tutorial

Related reference

New Project Wizard

Managed Make, Select a Target

Managed Make, Referenced Projects

Managed Make, Error Parsers

Managed Make, C/C++ Indexer

Standard Make, Name

Standard Make, Referenced Projects

Standard Make, Make Builder

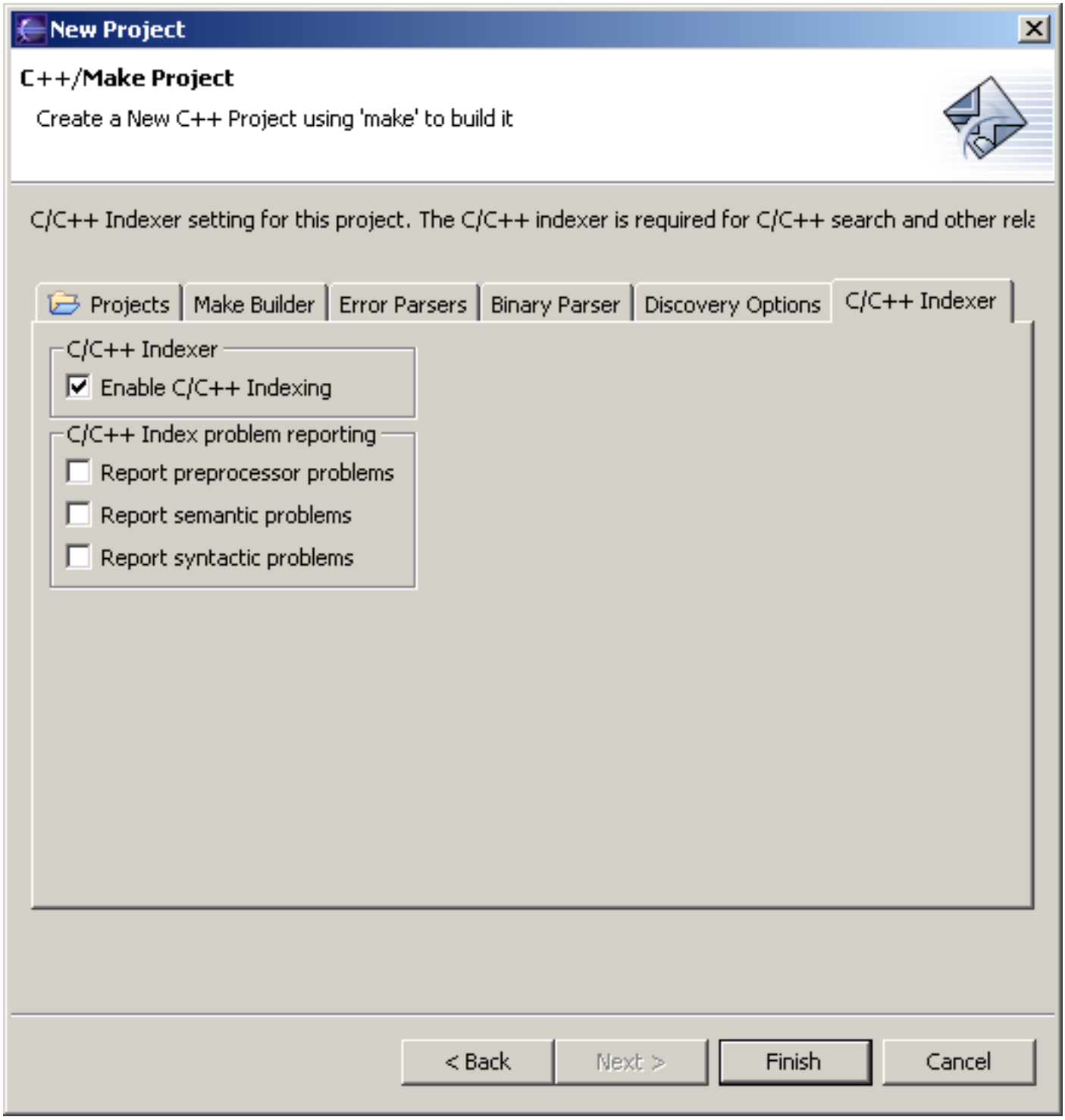
Standard Make, Error Parsers

Standard Make, Binary Parser

Standard Make, C/C++ Indexer

New Project Wizard - Standard Make, C/C++ Indexer

You can select which C/C++ Indexers to use for your project from this page of the wizard. The indexer is necessary for search and related features, like content assist.



Name	Function
Enable C++ Indexing	When selected C++ Indexing is enabled for this project.
Enable C++ Index problem reporting	When selected, any index related errors detected will be reported.

Related concepts
CDT projects

Related tasks
CDT Managed Make Tutorial
CDT Standard Make Tutorial

Related reference
New Project Wizard
Managed Make, Select a Target
Managed Make, Referenced Projects
Managed Make, Error Parsers
Managed Make, C/C++ Indexer
Standard Make, Name
Standard Make, Referenced Projects
Standard Make, Make Builder
Standard Make, Error Parsers
Standard Make, Binary Parser
Standard Make, Discovery Options

Run and Debug dialog boxes

This section describes the Run and Debug dialog boxes.

Main page, Run or Debug dialog boxes

The Main page of the Run dialog box and the Debug dialog box, identifies the project and application you want to run or debug.

Project
Specifies the name of the project.
C/C++ Application
Specifies the name of the application.

Related concepts

Debug overview
Debug information

Related tasks

Running and debugging

Related reference

Run and Debug dialog box

Arguments page, Run or Debug dialog boxes

The Arguments page of the Run dialog box and the Debug dialog box lets you specify the execution arguments that an application uses and the working directory for a run configuration.

C/C++ Project Arguments
Specifies the arguments that are passed on the command line.
Use default working directory
Clears the check box to specify a local directory or a different project in your workspace.
Local directory
Specifies the path of, or browse to, a local directory.
Workspace
Specifies the path of, or browse to, a workspace relative working directory.

Related concepts
Debug overview
Debug information

Related tasks
Running and debugging

Related reference
Run and Debug dialog box

Environment page, Run or Debug dialog box

The Environment page of the Run dialog box and the Debug dialog box lets you set environment variables and values to use when an application runs.

Name
Displays the name of environment variables.
Value
Displays the value of environment variables.
New
Creates a new environment variable.
Edit
Modifies the name and value of an environment variable.
Remove
Removes selected environment variables from the list.

Related concepts
Debug overview
Debug information

Related tasks
Running and debugging

Related reference
Run and Debug dialog box

Debugger page, Run or Debug dialog box

The Debugger page of the Run dialog box and the Debug dialog box lets you select a debugger to use when debugging an application.

Debugger
: Selects debugger from the list.

Run program in debugger
: Runs the program in debug mode.

Attach to running process
: Prompts you to select a process from a list at run-time.

Stop at main() on startup
: Stops program at main().

Enable variable bookkeeping
: Updates variables in the Variables view. Individual variables can be updated manually in the Variables view.

Related concepts

Debug overview

Debug information

Related tasks

Running and debugging

Related reference

Run and Debug dialog box

Source page, Run or Debug dialog box

The Source page of the Run dialog box and the Debug dialog box lets you specify the location of source files used when debugging a C or C++ application. By default, this information is taken from the build path of your project.

Generic Source Locations
: Displays the location of the project selected in the C/C++ Projects view and in any referenced projects.

Additional Source Locations
: Lists projects and directories added to the debugger search list.

Select All
: Selects all items in the Generic Source Locations list.

Deselect All
: Deselects all items in the Generic Source Locations list.

Add
: Adds new projects and directories to the debugger search list.

Up
: Moves selected items up the Additional Source Locations list.

Down
: Moves selected items down the Additional Source Locations list.

Remove
: Removes selected items from the Additional Source Locations list.

Related concepts

Debug overview

Debug information

Related tasks

Running and debugging

Related reference

Run and Debug dialog box

Common page, Run or Debug dialog box

The Main page of the Run dialog box and the Debug dialog box lets you specify the location in which to store your run configuration and how you access it and what perspective to open when running an application.

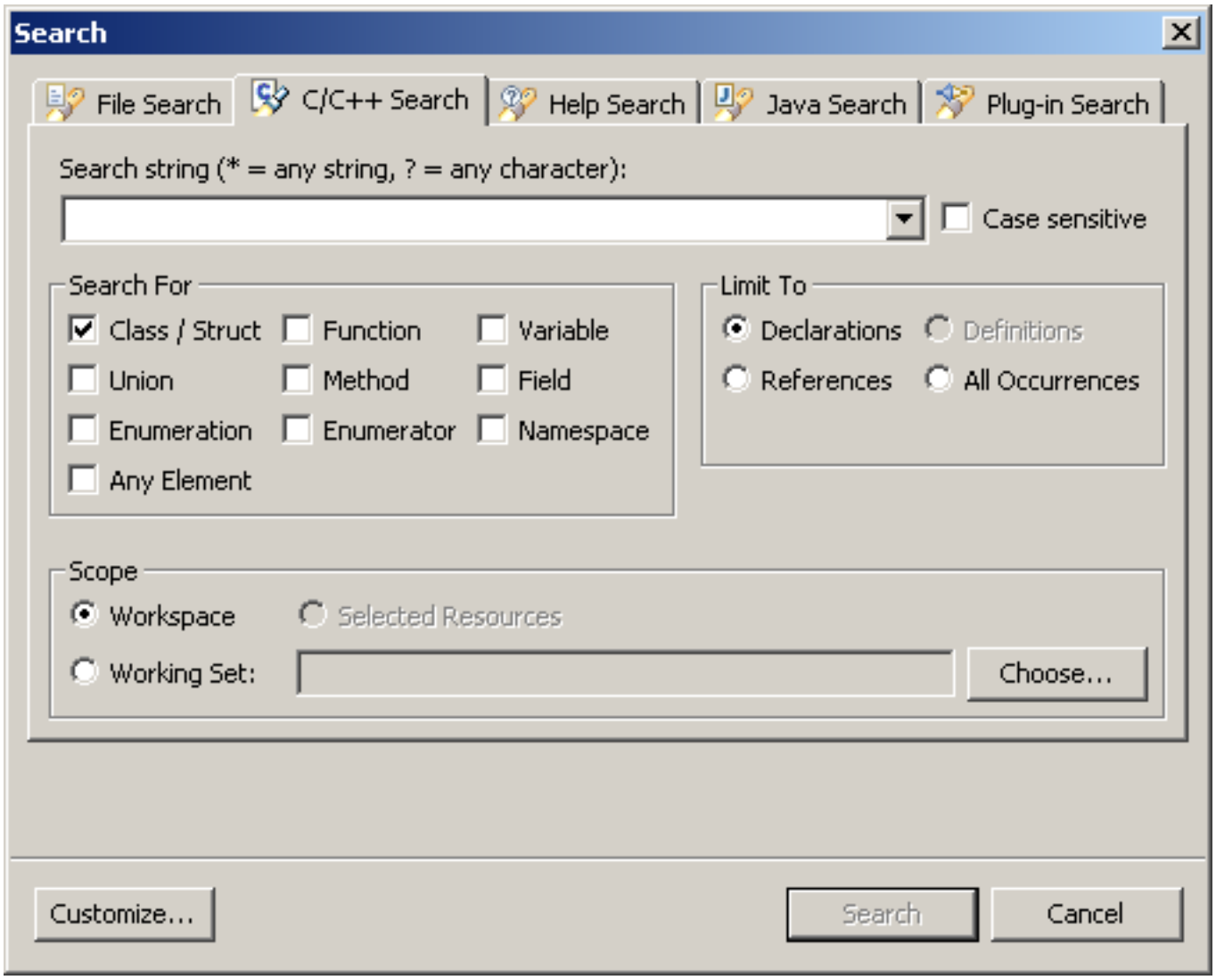
Local
Saves the launch configuration locally.
Shared
Saves the launch configuration to a project in your workspace, and be able to commit it to CVS.
Location of shared configuration
Specifies the location of a launch configuration.
Run mode
Selects a perspective to switch to when you run an application.
Debug mode
Selects a perspective to switch to when you debug an application.
Run
Displays "Run" in favorites menu.
Debug
Displays in "Debug" favorites menu.

Related concepts
Debug overview
Debug information

Related tasks
Running and debugging

Related reference
Run and Debug dialog box

C/C++ search



Search string

Specifies a search string.

The search functions support the following wildcards:

- Use * to represent any series of characters
- Use ? to represent any single character
- Use * to find an asterisk character (*)

Case sensitive

Searches will return results which match capitalization specified in the search string precisely.

Search for

Specify the types of elements to include in the search:

Search for	Description

Class/Struct	Includes classes and structs in your search.
Function	Searches for global functions or functions in a namespace (functions that are not members of a class, struct, or union).
Variable	Searches for variables that are not members of a class, struct, or union.
Union	Searches for unions.
Method	Searches for methods that are members of a class, struct, or union.
Field	Searches for fields that are members of a class, struct, or union.
Enumeration	Searches for enumerations.
Enumerator	Searches for enumerators.
Namespace	Searches for namespaces.
Any Element	Includes all elements in the search.

Limit to

Specify the types of context to search:

Limit to	Description
Declarations	Limits the search to declarations.
Definitions	Limits the search to definitions (for functions, methods, variables, and fields).
References	Limits the search to references.
All Occurrences	Includes declarations, definitions, and references in the search.

Scope

Choose the scope of the search:

Scope	Availability	Description
Workspace	all elements	Searches in the full workspace
Selected Resources	all elements	Searches the project selected in the **Projects** view
Workings Set	all elements	Searches in a working set

Working sets can be created and used from within the search dialog.

Related concepts

Coding aids

C/C++ search

Related tasks

Searching for C/C++ elements

Customizing the C/C++ editor

Related reference

C/C++ editor preferences
Search action
Search view