# Discovery options

For projects where the IDE generates a makefile to build the project automatically, the IDE has more information about the internal build state of the make project compared to those projects where you provide a makefile to build the project.

For example, a makefile includes build information and other settings, such as include file paths (-I) and macro definitions (-D), which are standard to the build tools (such as the compiler and linker). When the makefile is automatically created for you, this information is known to Eclipse to a greater extent then when you create and manage the makefile for a project yourself. The type of information affects the search capabilities and Code Assist abilities of Eclipse. Therefore, in this example, the purpose of Discovery Options is for improved search and Code Assist capability for projects where Eclipse does not manage the makefile for you. For example, in an open source file of an editor view, to see the declaration for a function that the code calls, you select the function, right click, and select **Open Declaration** from the context menu. If the location of the include file (that was coded in the makefile for the project) containing the function declaration was in some directory, the CDT would not find the declaration because it has no visibility for that include path. Consequently, you can use Discovery Options in the CDT to enhance the IDE build state by parsing the build process output to extract build path information that the CDT searching mechanism uses to locate and open the include file. By default, the CDT uses GNU\* tools (gcc, etc.). If you want to build your projects using another compiler, use the settings described here.

Scanner configuration discovery is tightly linked to project's build process. The first part of scanner discovery begins during the make build for make projects where you provide the makefile. The Eclipse CDT parses the build output for compiler commands with options that specify the definition of the preprocessor symbols and include search paths (for the gcc compiler, -D and -I), and then it stores the information as the project's discovered scanner configuration.

Next, after the build process completes, it is implemented as a separate Eclipse builder where it runs a generate scanner info command, and then parses the output (properties specified on the Discover Options tab for Builds in the Project Properties window).

For C++, the default generate scanner information command is **gcc -E -P -v myfile.c | myfile.cpp**. This command reads the compiler's configuration file and prints the information that includes compiler's internally defined preprocessor symbols and include search paths.

A single scanner configuration is applicable to all the files in a project. Although Eclipse discovers the information for each compilation unit, it stores the scanner configuration on a per project basis. This means that Eclipse applies a single, cumulative scanner configuration to all files in a project.

The following situations identify circumstances when certain parts of the scanner configuration are in conflict:

* Symbol definitions that contradict
* An incorrect order for the include paths
* Internal scanner information may not be applicable to all the files in a project (for a single compiler), because some files might be compiled with a different compiler, or have target specific options specified.

**Note:** Only basic command line options are supported. In addition, only basic scanner configuration related command line options are recognized (for example, -D and -I for gcc). For some of the commands, their relative position in the command line is important. For information about these options, see the documentation for the utilities you are using.



[CDT Overview](http://docs.google.com/cdt_c_over_cdt.htm)

[C/C++ perspectives](http://docs.google.com/cdt_c_perspectives.htm)



[Views](http://docs.google.com/reference/cdt_o_views.htm)



