
TECHNICAL SPECIFICATION

How to move project from winIDEA to Eclipse

Introduction

This article describes the Eclipse build configuration and how to move build settings from winIDEA to Eclipse.

The build process in CDT is performed by tool-chains. A tool-chain is a collection of tools, which process source files to produce some kind of output. In our case the output will be an executable file. A tool-chain usually contains at least a compiler and a linker, sometimes also an assembler and other tools. CDT comes with a few tool-chains installed, but there are many more compilers on the market. If you have a compiler (assembler, linker), which is not supported by the available tool-chains, this article may help you. It describes, how to configure the Cygwin tool-chain on Windows to use other tools.

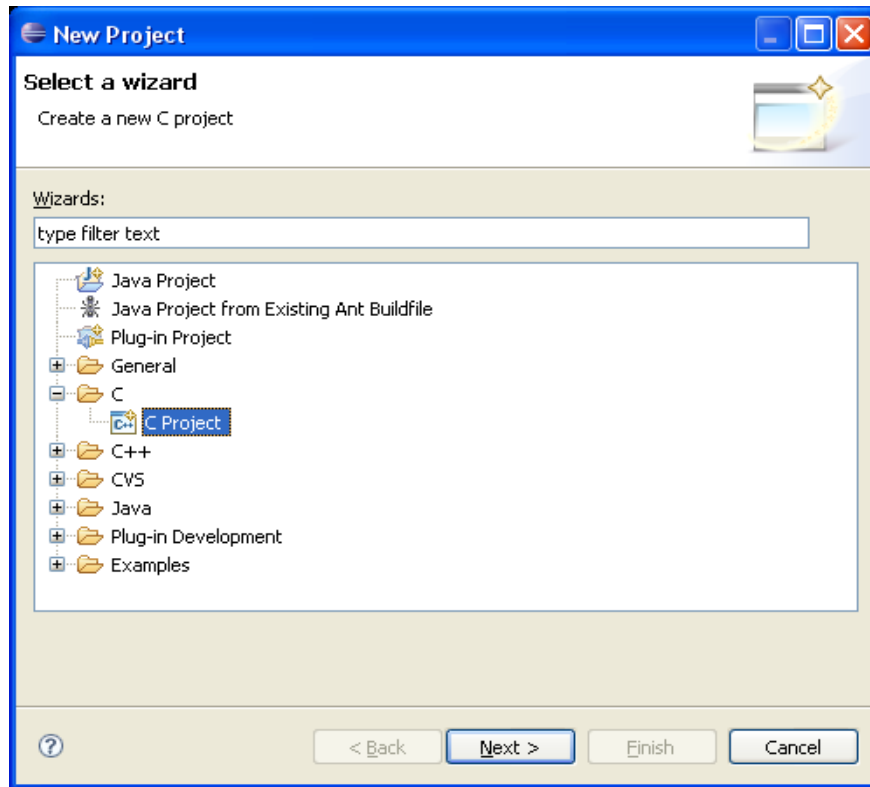
Requirements

- Eclipse 3.4.x
- CDT 5.01
- Compiler and linker for your target platform.
- (Optional) [RegExErrorParser plug-in](#)

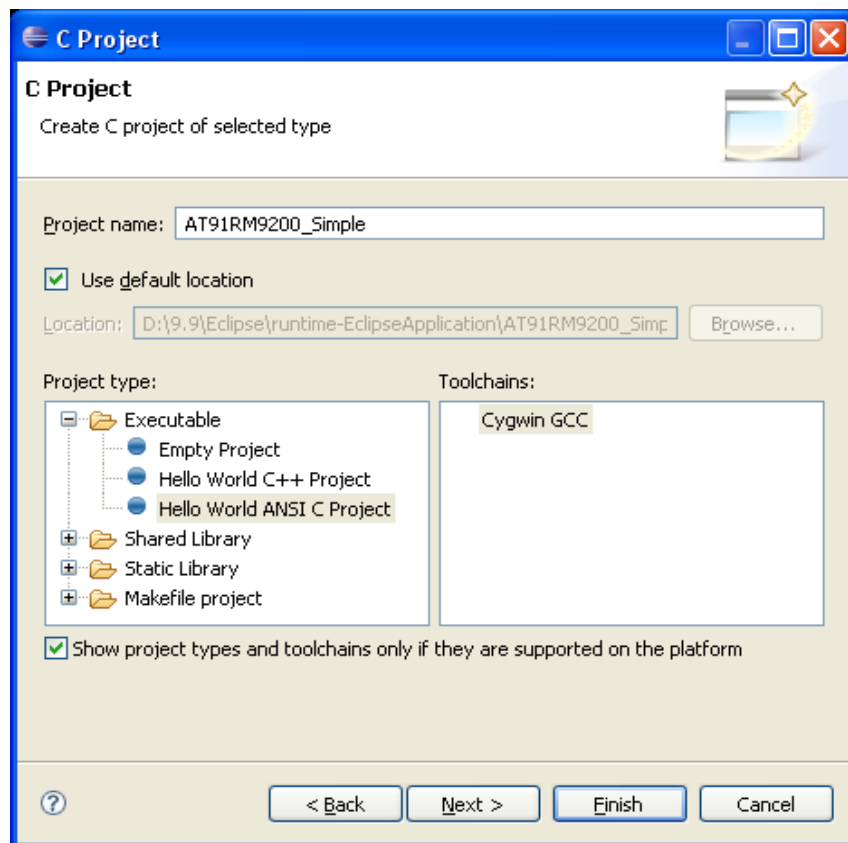
Creating a CDT Project

For the purpose of this article, you'll need a small C project:

1. Select **File > New > Project**.
2. Select the type of project to create. For this tutorial, expand the **C** folder and select **C Project**.



The **C Project** wizard opens. Enter the project name in the **Project name** field and select one of the **Executable** templates. Leave the default tool-chain for now.



3. Leave the **Use Default Location** option selected.
4. Click **Finish**.
5. If a message box prompts you to change the perspectives, click **Yes**.

Your new project displays in the C/C++ Projects view, and in the Navigator view. You can now start writing the code for your program.

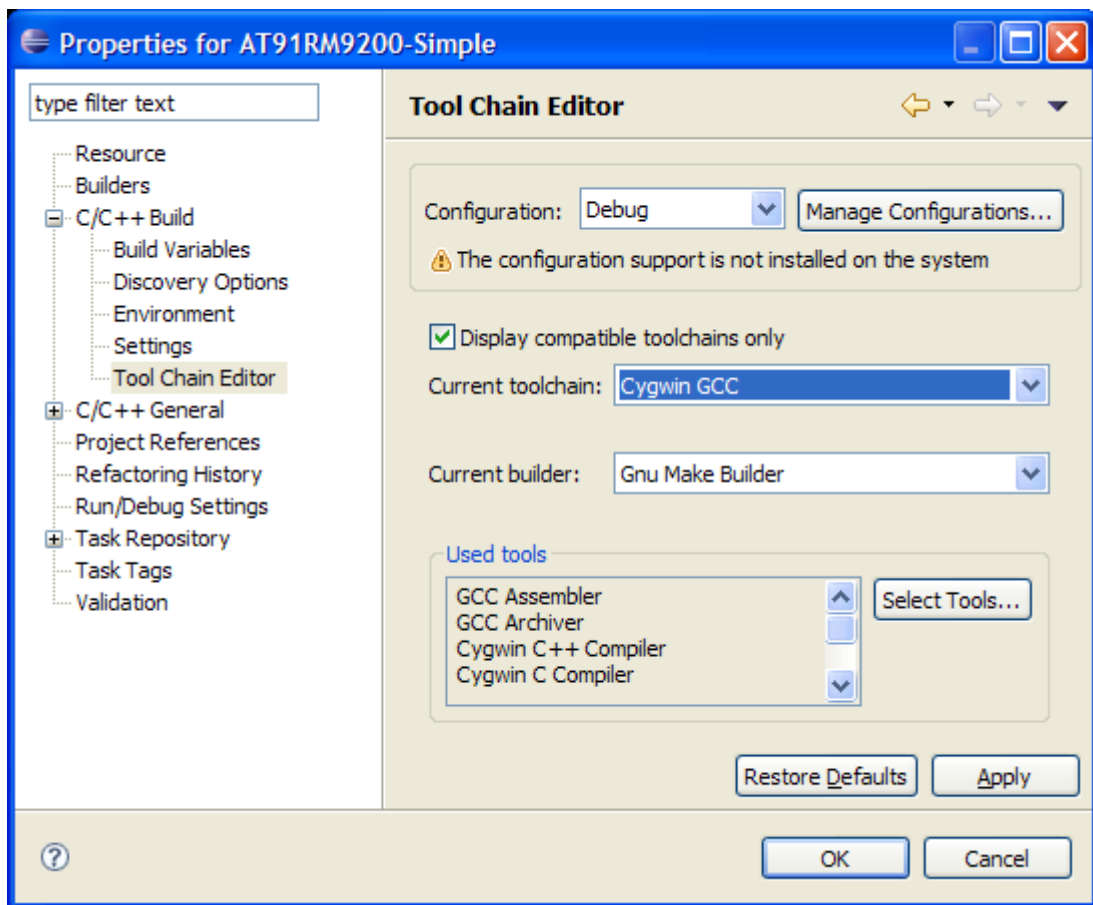
The creation of projects is also described in the CDT on-line documentation http://help.eclipse.org/stable/index.jsp?topic=/org.eclipse.cdt.doc.user/concepts/cdt_o_home.htm, Step 1.

Project Configuration

Once the project is created you must configure it, so that CDT knows how to produce an executable. To edit the configuration, select the project in Project explorer and open its properties (**Project | Properties**).

Tool Chain Editor

First you should select a tool-chain as shown on the image below.

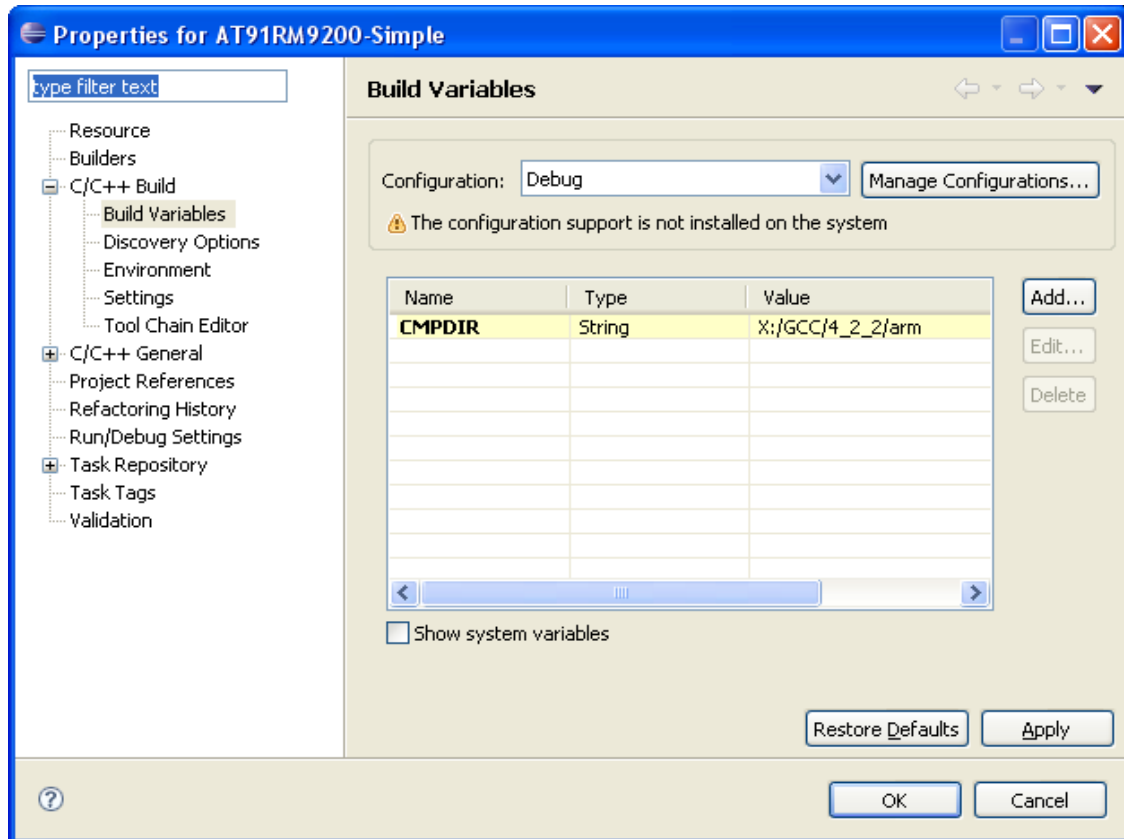


The warning, that the configuration support is not installed on the system, can be ignored.

Build variables

Build variables are handy for defining properties used more than once. You can assign property to variables and then use the variables latter in the settings. If the value of the property changes, you have to modify it in one place only.

For the purpose of this article let's define the property for the tools directory. Name it **CMPDIR** and assign it the directory, where the compiler and linker are located.



Settings

Here you will define commands and parameters for the assembler, the compiler and the linker.

Tool settings

This page defines settings for all tools: assembler, compiler and linker. When you click a tool name in the left pane the settings summary shows on the right.

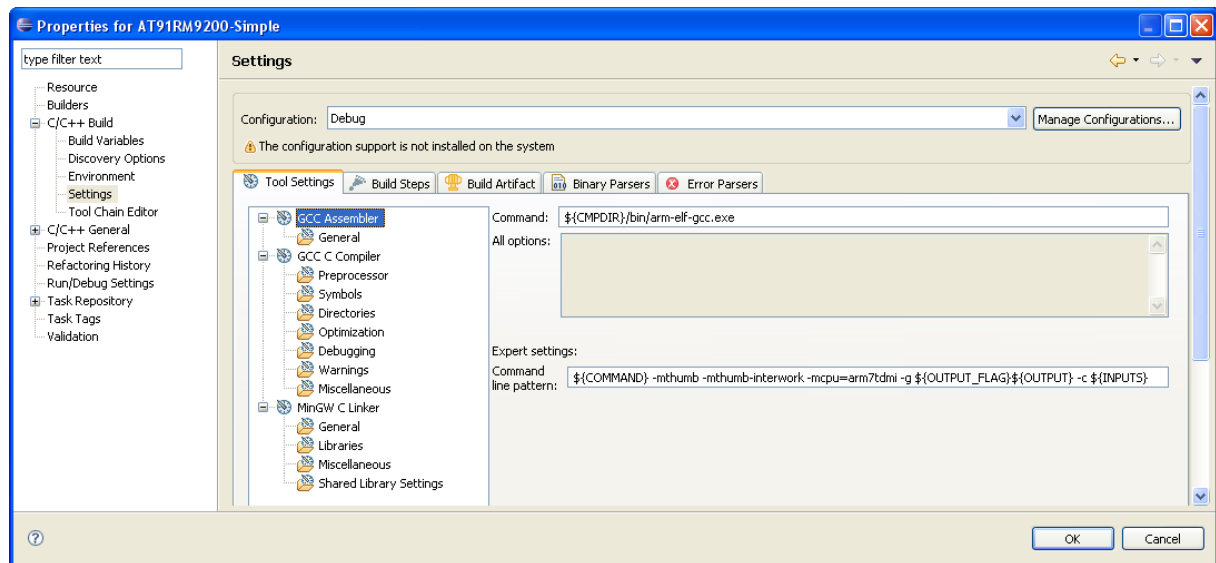
All tools have a very similar summary page, with **Command** and **Command line pattern** input fields. Although **Command line pattern** field originally consists of macros, you can remove some of them and replace them with literal text. This has to be done, when you can not define macros as required. For example, in **Compiler settings** the optimization flag can not be removed.

Assembler

First you enter the **Command** parameter, which contains the name of the tool including path . The macro **CMPDIR** can be used here.

Then enter the flags in the **GCC Assembler | General** page, or directly in the **Command line pattern** input field. These flags are tool specific. Consult the tool documentation to learn how to set them.

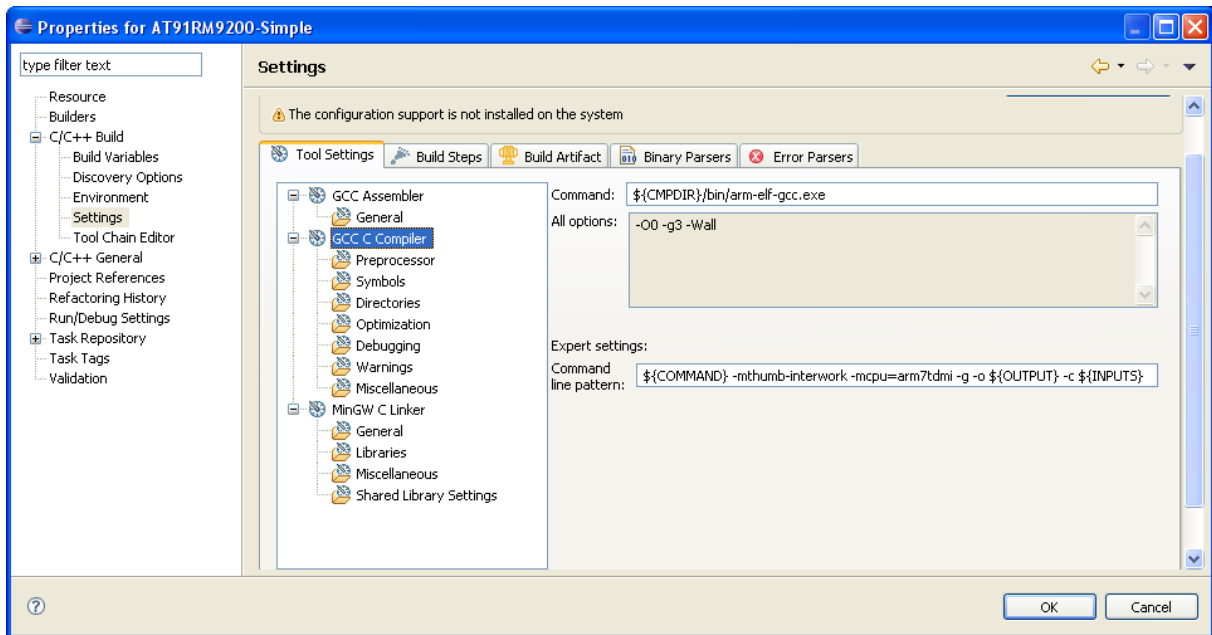
Last, provide the assembler configuration by specifying input and output parameters as macros. The value of these macros are set according to the source file name by the CDT build manager.



Tip: If you resize the dialog window, the **Command line pattern** input field gets wider and is then easier to edit.

Compiler

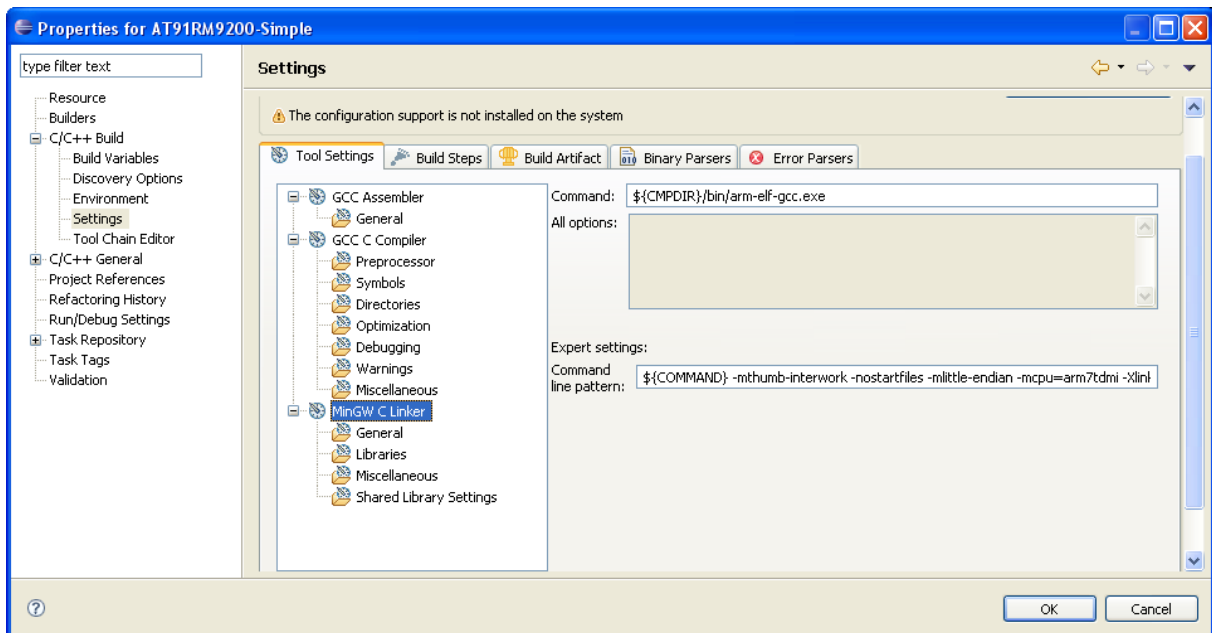
Configuration of compiler is similar. Again you specify a **Command** and a **Command line pattern**. In this example the executable is the same as for the assembler, but it is not always the case. Consult your documentation to find the right executable for compiler and its parameters.



Please note, that contents of the **All options** field is ignored in this case, because the **FLAGS** macro is not used in the **Command line pattern** input field.

Linker

As before, enter the tool name and parameters.



Build steps

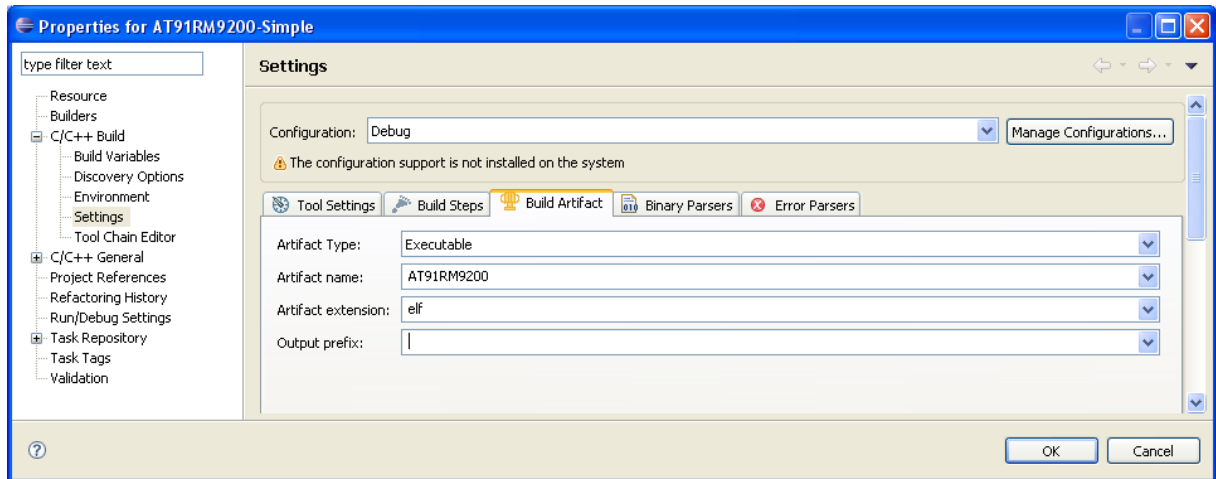
You can leave fields on this page empty.

Build Artifact

This page allows you to define values for macros **OUTPUT** and **OUTPUT_PREFIX**, which can be used in the linker command line. These macros are defined as:

OUTPUT = **Artifact name** + “. ” + **Artifact extension**

OUTPUT_PREFIX = **Output prefix**



Binary parsers

We can select a parser, which can parse the binary output file. If none of the available parsers is useful, select none of them. They are not required for build process.

Error parsers

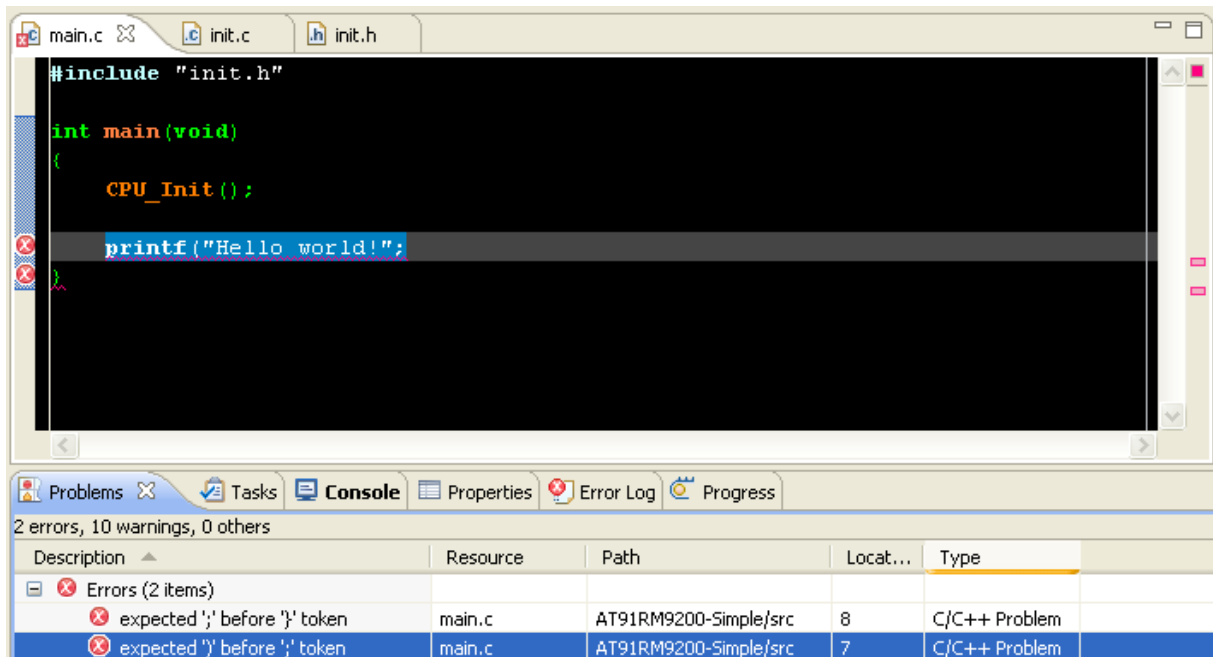
Error parsers parse output generated by the build process. When they detect a warning or error message, they extract the file name, line number and description, and notify the editor about problems. The editor then marks the line with error or warning marker. All errors and warnings are also added to the **Problems** view (**Window | Show view | Problems**). Since not all warning and error messages contain file and line number information (for example most linker warnings and errors), it is always a good idea to check also the **Problems** view.

Example:

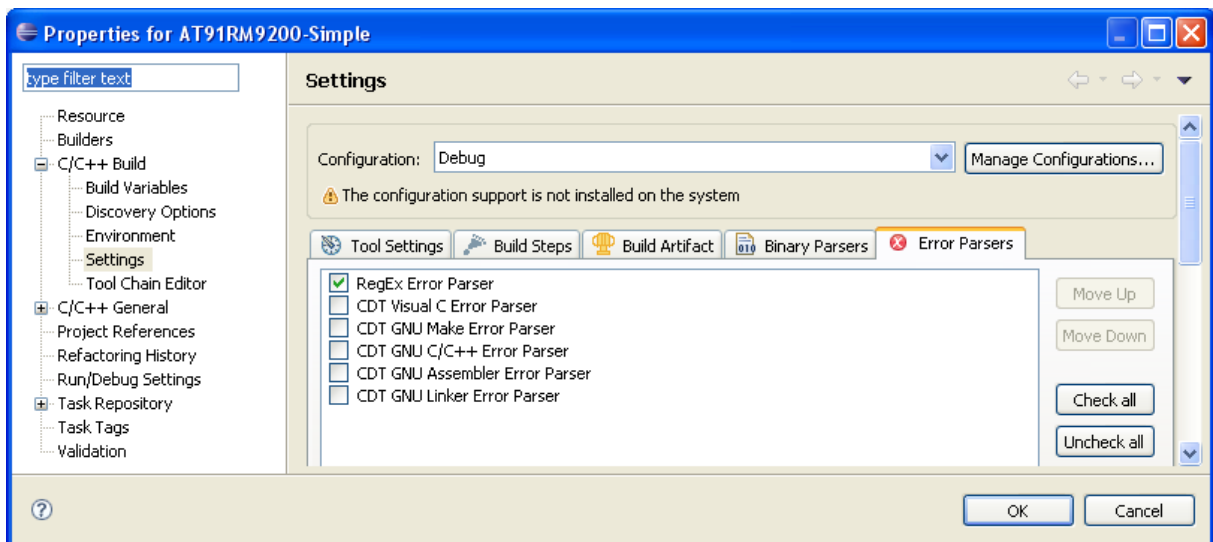
Output produced by compiler:

```
X:\GCC\4_2_2\arm\bin\arm-elf-gcc.exe -mthumb-interwork -mcpu=arm7tdmi -g -o src\main.o
-c ..\src\main.c
..\src\main.c: In function 'main':
..\src\main.c:7: warning: incompatible implicit declaration of built-in function
'printf'
..\src\main.c:7: error: expected ')' before ';' token
..\src\main.c:8: error: expected ';' before '}' token
Build error occurred, build is stopped
Time consumed: 703 ms.
```

This text goes to error parser, which then notifies the editor, and Eclipse shows the following:



Unfortunately the output format of tools is not standard, which means that error parsers available with the default CDT installation will not be useful for other tools. The solution is to write a custom error parser as an Eclipse plugin. Although this task is not complex, a better option is to use a generic parser that reads regular expressions from a text file in Java properties format. ISYSTEM provides a parser called **RegExErrorParser**. It is available at http://www.isystem.com/5653/Develop_Software/Products/Eclipse.html/#Parser, together with instructions. Once it is installed, we can select it from the list of available error parsers. If more than one error parser is selected, they are called in order.



Project configuration is now finished.

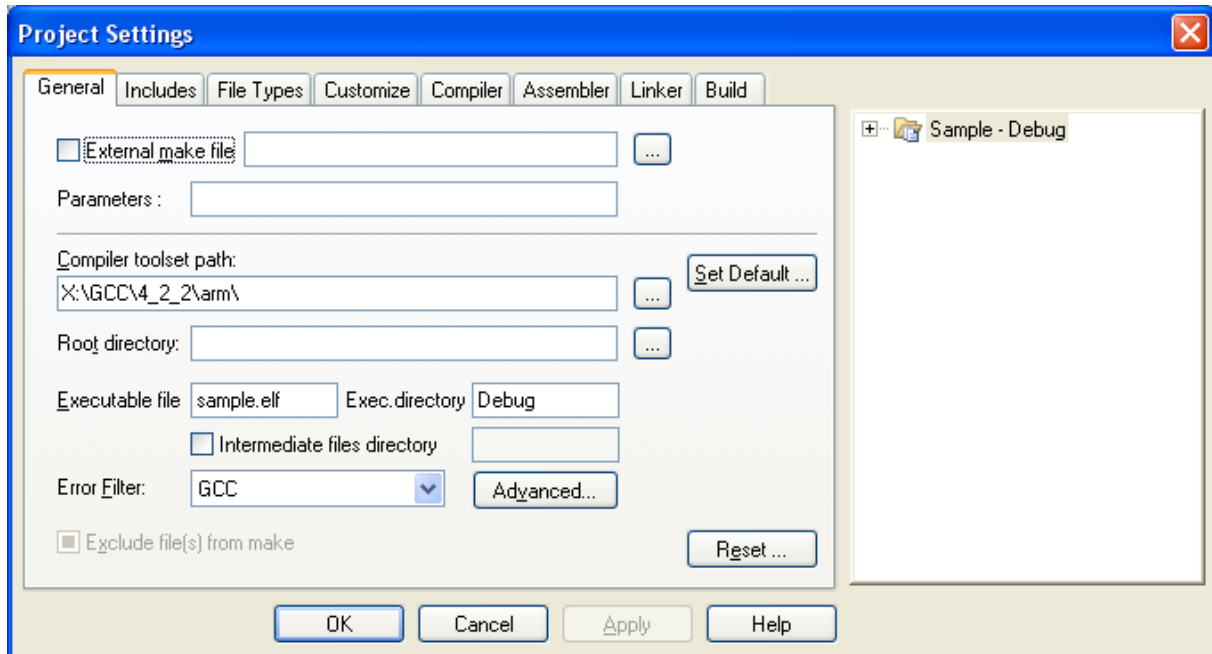
Building

To build a project select the command **Project | Build Project**. You can open the **Console** view, to see the exact input and output for the tools. This information is essential when fixing bugs in the configuration.

Moving project from winIDEA to Eclipse

Once you know how to configure the Eclipse build settings, moving a project from winIDEA to Eclipse is straightforward. This section describes how to read the required information from winIDEA.

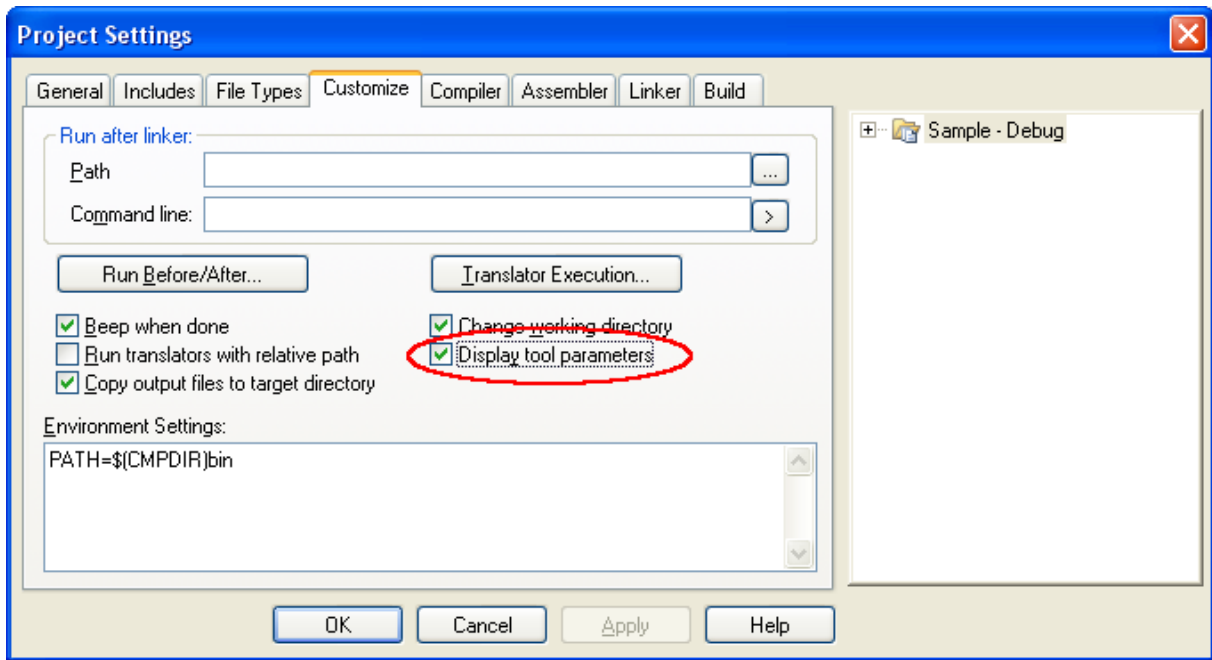
All build settings are accessible in the dialog, which is opened from the **Project | Settings** menu option.



Some hints:

- **Compiler toolset path** contains a path, which you can use for **CMPDIR** build variable in Eclipse.
- **Assembler**, **Compiler**, and **Linker** tabs contain options entered in the **Tool Settings** tab in Eclipse
- The **Executable file** is composed of the **Artifact name** and **Artifact extension** in Eclipse.
- The **Error filter** is called **Error parser** in Eclipse.

Since the commands used to run the tools are composed of many settings, it is not always obvious what the final command will look like. In such case it is best to run a full build (**Project | Rebuild**) and observe the output in the **Output view (View | Output, Alt+2)**. If there are no commands displayed, make sure, that the **Display tool parameters** option in **Customize** tab is checked.



Example of output line displayed in the winIDEA's Output view (file paths are truncated):

```
main.c ... with parameters "X:\GCC\4_2_2\arm\bin\arm-elf-gcc.exe -mthumb-interwork -mcpu=arm7tdmi -g -o Debug\main.o -c main.c"
```

Conclusion

Build parameters are stored in different locations in each IDE, but once we know where to set them, it is not a difficult task.