JBLopen

Embedded Software Insight

GCC Toolchain Eclipse Setup Guide

WP0001 Version 5

September 23, 2020

Contents

Chapter

1 Overview

This document provides a series of step-by-step setup guides to install and integrate a GCC toolchain within the Eclipse IDE on Microsoft Windows® and Linux. This guide is written with embedded software development for MCU and SoCs in mind but can be used with any GCC toolchain regardless of the intended target. Note that there are many combinations of Eclipse versions, plugins and supporting tools that can be used to successfully build C and C++ applications using the GCC toolchain, this document is the recommended way by JBLopen but customers are free to modify their development environment to suit their needs.

The focus of this document is on the Eclipse setup which includes installation and setup of Eclipse and the chosen toolchain, creation of either a Managed CDT project or a Makefile project and finally how to create a debug configuration for GDB debugging.

1.1 About Eclipse

Eclipse is a cross-platform IDE that supports multiple programming languages. This document will mainly use the Eclipse CDT environment for C and C++ code development as well as the GDB integration offered by Eclipse.

Chapter

2 Eclipse Setup Guide (Windows)

This section will go over the Eclise IDE and toolchain setup for Windows. Instructions for Linux are available in the next chapter. The following step-by-step instructions will go through the installation and configuration of the following components in order:

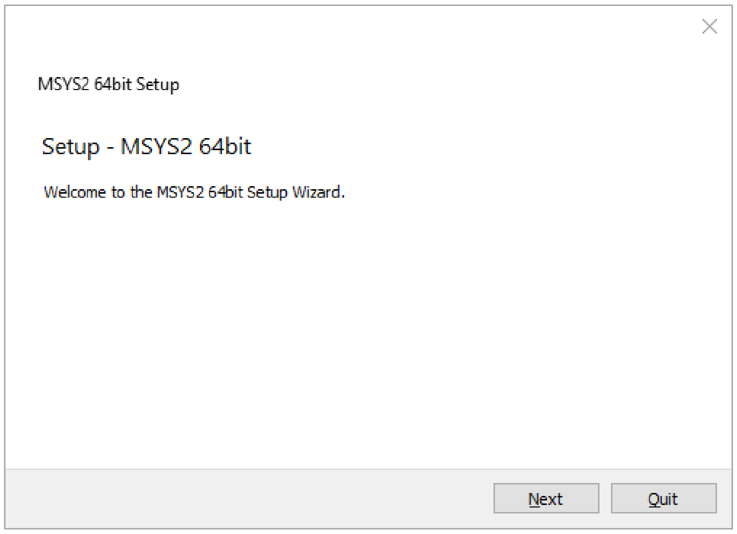
- MSYS2 – 20200903 – msys2-x86_64-20200903.exe
  www.msys2.org
- Eclipse – 2020.06 – eclipse-inst-win64.exe
  www.eclipse.org
- GCC - 9-2020q2 - gcc-arm-none-eabi-9-2020-q2-update-win32.zip
  https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm

The versions cited above are those that were used when writing this guide. Installation instructions for newer versions, if available, should be similar.

2.1 MSYS2 Installation

On Windows a basic UNIX environment is required to run the toolchain and more importantly supply the GNU Make utility. There are multiple options including the well-known Cygwin. This guide will use MSYS2 since it offers superior build performance and good compatibility with native Windows paths. If compatible with the host system, the 64-bit version of MSYS2 is recommended, which is labelled x86_64.

The installation can be started by executing the executable installation file, in this example `msys2-x86_64-20180531.exe`, the following screen should appear:

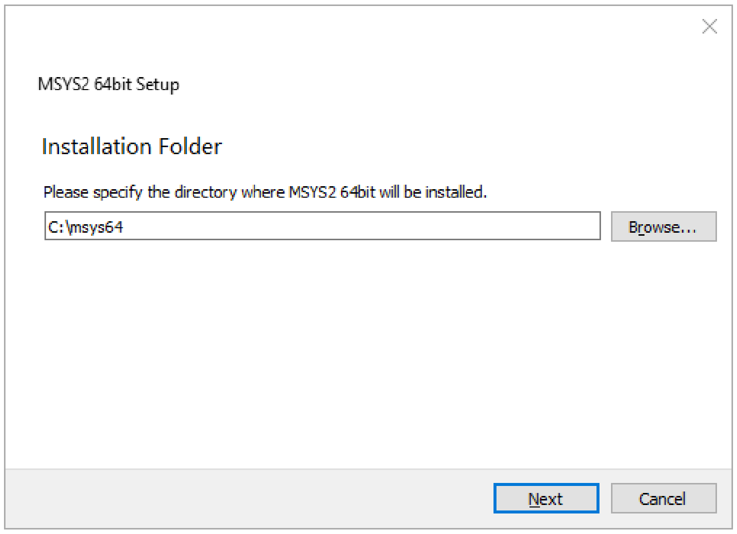

MSYS2 64bit Setup

Setup - MSYS2 64bit

Welcome to the MSYS2 64bit Setup Wizard.

Figure 1 – MSYS2 installation welcome prompt

Click Next.


MSYS2 64bit Setup

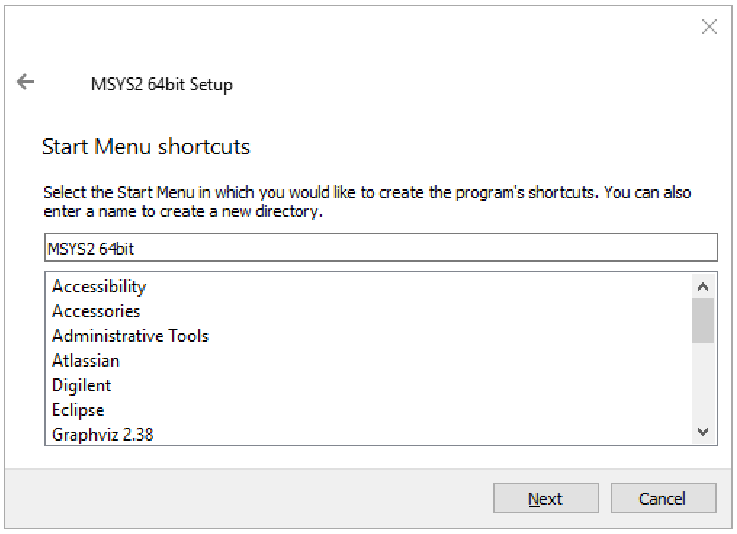
Installation Folder

Please specify the directory where MSYS2 64bit will be installed.

C:\msys64

Figure 2 – MSYS2 installation directory prompt

Enter the desired installation directory, the installation directory should be short with no spaces in the path. This guide will assume that the default installation directory of `C:/msys64` is used. Click Next.


MSYS2 64bit Setup

Start Menu shortcuts

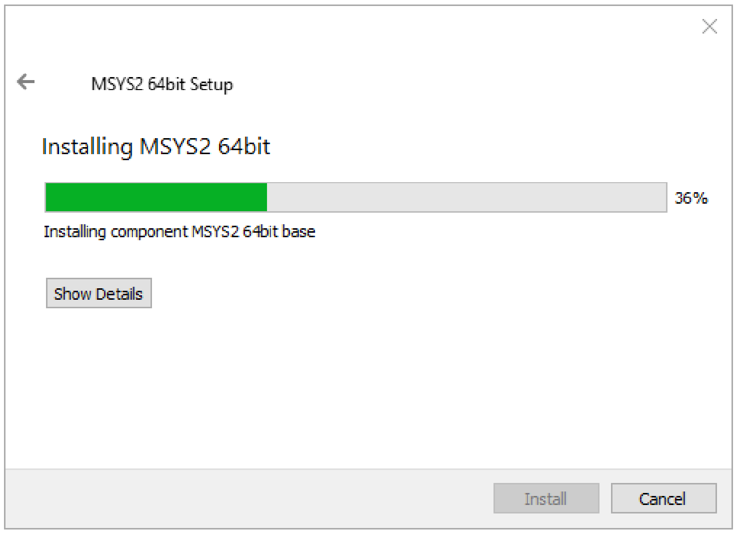
Select the Start Menu in which you would like to create the program's shortcuts. You can also enter a name to create a new directory.

MSYS2 64bit

Accessibility
Accessories
Administrative Tools
Atlassian
Digilent
Eclipse
Graphviz 2.38

Figure 3 – MSYS2 installation start menu shortcut prompt

The installer will now prompt for the name of the start menu shortcut. It is recommended to use the default name. Click Next.


MSYS2 64bit Setup

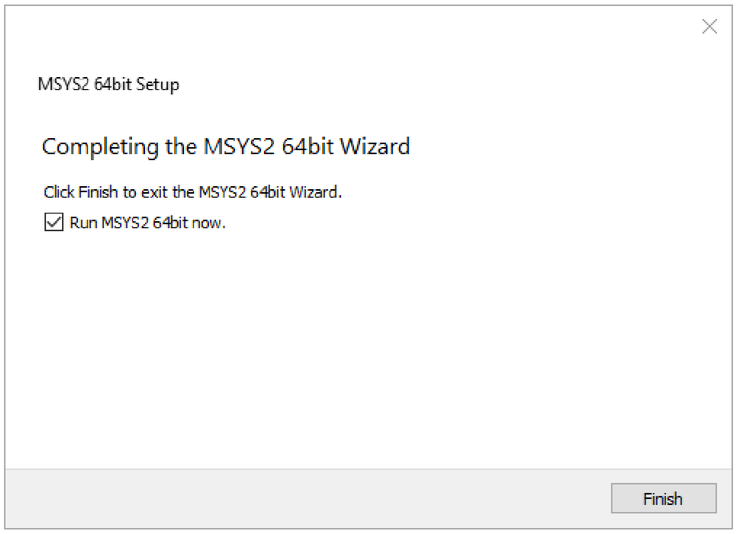
Installing MSYS2 64bit

36%

Installing component MSYS2 64bit base

Figure 4 – MSYS2 installation progress screen

Installation should begin.



MSYS2 64bit Setup

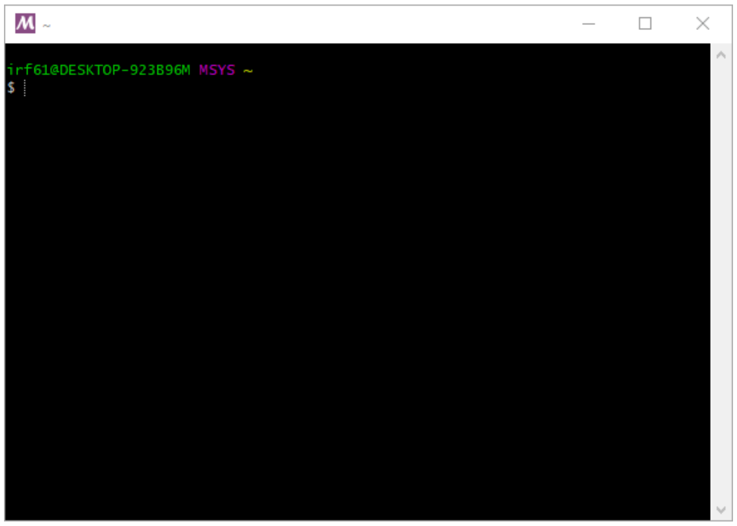
Completing the MSYS2 64bit Wizard

Click Finish to exit the MSYS2 64bit Wizard.

☑ Run MSYS2 64bit now.

Figure 5 – MSYS2 installation finished screen

The installation should finish successfully. Make sure that ”Run MSYS2 64bit now.” is checked.



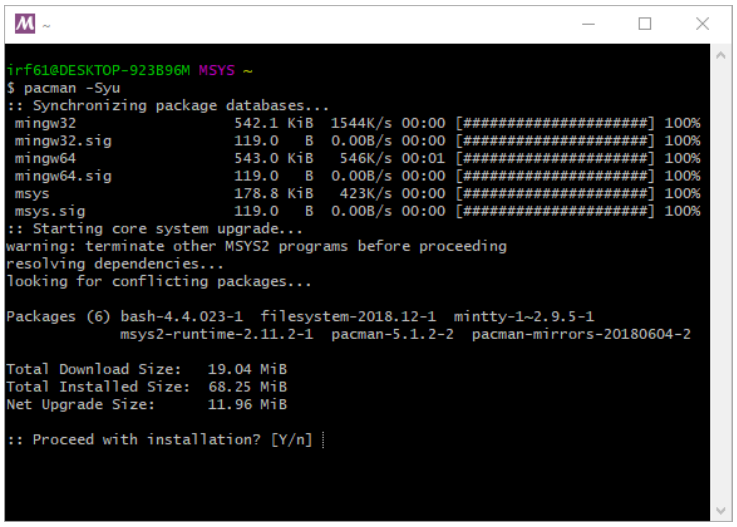
```
irf61@DESKTOP-923B96M MSYS ~
$
```

Figure 6 – MSYS2 console

The MSYS2 console should open. It is now necessary to perform an update of the MSYS package manager by typing:

```
pacman -Syu
```

And press enter.

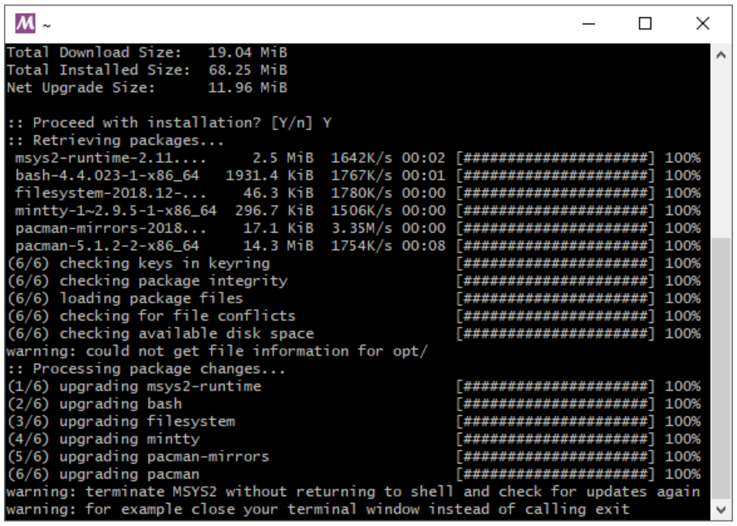


```
irf61@DESKTOP-923B96M MSYS ~
$ pacman -Syu
:: Synchronizing package databases...
 mingw32               542.1 KiB  1544K/s 00:00 [######################] 100%
 mingw32.sig           119.0   B  0.00B/s 00:00 [######################] 100%
 mingw64               543.0 KiB   546K/s 00:01 [######################] 100%
 mingw64.sig           119.0   B  0.00B/s 00:00 [######################] 100%
 msys                  178.8 KiB   423K/s 00:00 [######################] 100%
 msys.sig              119.0   B  0.00B/s 00:00 [######################] 100%
:: Starting core system upgrade...
warning: terminate other MSYS2 programs before proceeding
resolving dependencies...
looking for conflicting packages...

Packages (6) bash-4.4.023-1  filesystem-2018.12-1  mintty-1~2.9.5-1
             msys2-runtime-2.11.2-1  pacman-5.1.2-2  pacman-mirrors-20180604-2

Total Download Size:   19.04 MiB
Total Installed Size:  68.25 MiB
Net Upgrade Size:      11.96 MiB

:: Proceed with installation? [Y/n]
```

Figure 7 – MSYS2 package manager update prompt

After downloading the update information, a prompt will appear to ask for permission to proceed with the installation. Type Y and press enter to continue.

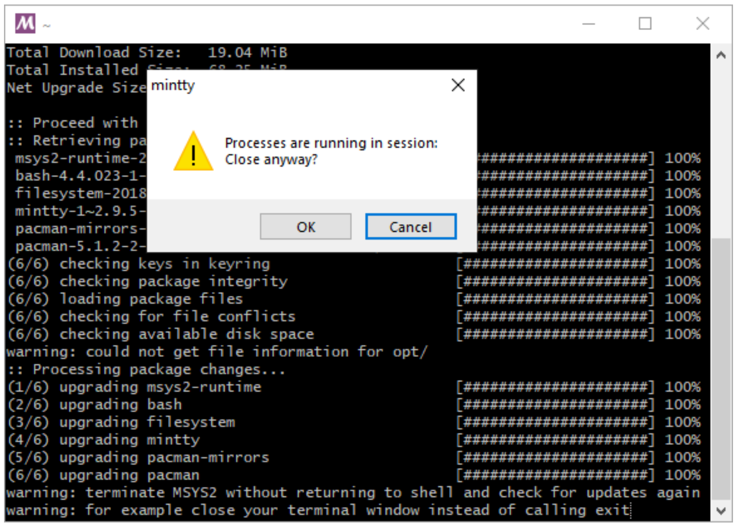
Download and installation will proceed, this can take several minutes depending on network connection and computer performance.



```
Total Download Size:   19.04 MiB
Total Installed Size:  68.25 MiB
Net Upgrade Size:      11.96 MiB

:: Proceed with installation? [Y/n] Y
:: Retrieving packages...
 msys2-runtime-2.11....      2.5 MiB  1642K/s 00:02 [######################] 100%
 bash-4.4.023-1-x86_64    1931.4 KiB  1767K/s 00:01 [######################] 100%
 filesystem-2018.12-...     46.3 KiB  1780K/s 00:00 [######################] 100%
 mintty-1~2.9.5-1-x86_64  296.7 KiB  1506K/s 00:00 [######################] 100%
 pacman-mirrors-2018...     17.1 KiB  3.35M/s 00:00 [######################] 100%
 pacman-5.1.2-2-x86_64      14.3 MiB  1754K/s 00:08 [######################] 100%
(6/6) checking keys in keyring                      [######################] 100%
(6/6) checking package integrity                    [######################] 100%
(6/6) loading package files                         [######################] 100%
(6/6) checking for file conflicts                   [######################] 100%
(6/6) checking available disk space                 [######################] 100%
warning: could not get file information for opt/
:: Processing package changes...
(1/6) upgrading msys2-runtime                       [######################] 100%
(2/6) upgrading bash                                [######################] 100%
(3/6) upgrading filesystem                          [######################] 100%
(4/6) upgrading mintty                              [######################] 100%
(5/6) upgrading pacman-mirrors                      [######################] 100%
(6/6) upgrading pacman                              [######################] 100%
warning: terminate MSYS2 without returning to shell and check for updates again
warning: for example close your terminal window instead of calling exit
```

Figure 8 – MSYS2 package update restart request

It is likely that a warning message is displayed at this point asking to return to the shell and restart the update process. When this occurs exit the MSYS2 console by pressing the X button on the top right corner of the window. It is important to quit the console that way otherwise the warning may reappear in the next invocation.



```
Total Download Size:   19.04 MiB
Total Installed
Net Upgrade Size

:: Proceed with
:: Retrieving pa
 msys2-runtime-2                                  ####################] 100%
 bash-4.4.023-1-                                  ####################] 100%
 filesystem-2018                                  ####################] 100%
 mintty-1~2.9.5-                                  ####################] 100%
 pacman-mirrors-                                  ####################] 100%
 pacman-5.1.2-2-                                  ####################] 100%
(6/6) checking keys in keyring                   [######################] 100%
(6/6) checking package integrity                 [######################] 100%
(6/6) loading package files                      [######################] 100%
(6/6) checking for file conflicts                [######################] 100%
(6/6) checking available disk space              [######################] 100%
warning: could not get file information for opt/
:: Processing package changes...
(1/6) upgrading msys2-runtime                    [######################] 100%
(2/6) upgrading bash                             [######################] 100%
(3/6) upgrading filesystem                       [######################] 100%
(4/6) upgrading mintty                           [######################] 100%
(5/6) upgrading pacman-mirrors                   [######################] 100%
(6/6) upgrading pacman                           [######################] 100%
warning: terminate MSYS2 without returning to shell and check for updates again
warning: for example close your terminal window instead of calling exit
```

mintty

Processes are running in session:
Close anyway?

Figure 9 – MSYS2 window close confirmation

A warning message will be displayed, press OK to proceed with the close.

Open the MSYS2 console again by navigating to the start menu item for MSYS2 and clicking MSYS2 MSYS.

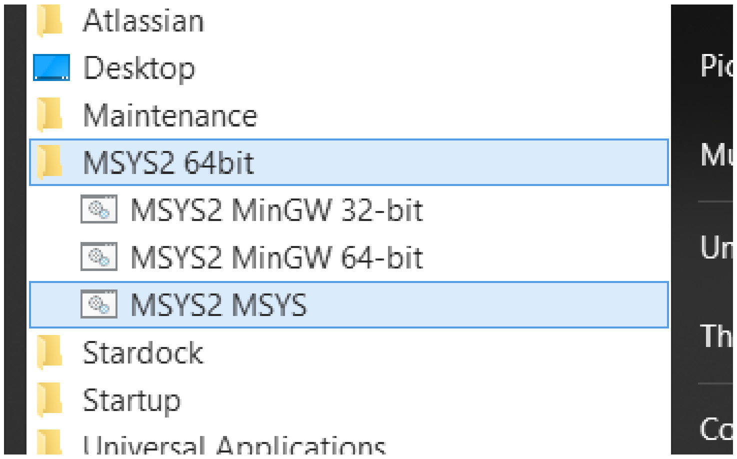
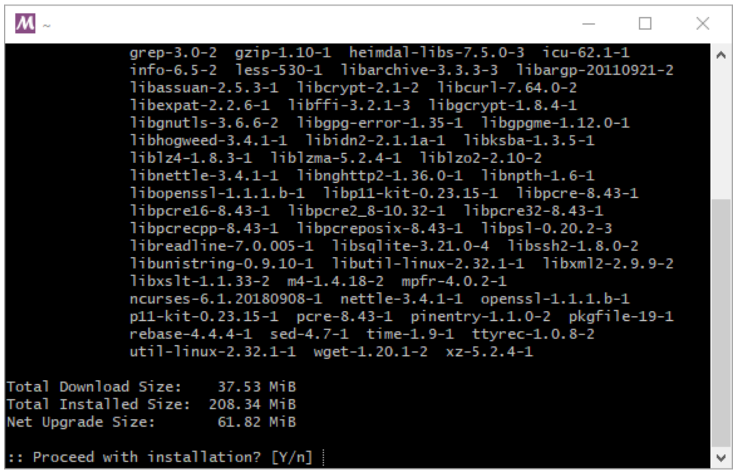


Figure 10 – MSYS2 start menu entry

Once opened type:

```
pacman -Su
```

And press enter to continue the update process.



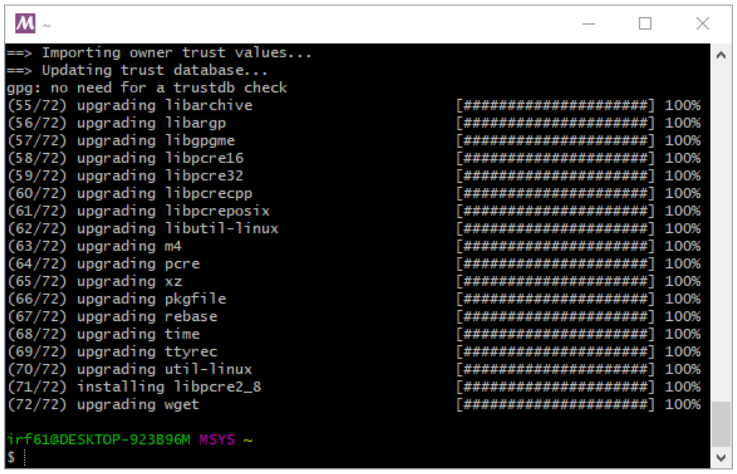
```
              grep-3.0-2  gzip-1.10-1  heimdal-libs-7.5.0-3  icu-62.1-1
              info-6.5-2  less-530-1  libarchive-3.3.3-3  libargp-20110921-2
              libassuan-2.5.3-1  libcrypt-2.1-2  libcurl-7.64.0-2
              libexpat-2.2.6-1  libffi-3.2.1-3  libgcrypt-1.8.4-1
              libgnutls-3.6.6-2  libgpg-error-1.35-1  libgpgme-1.12.0-1
              libhogweed-3.4.1-1  libidn2-2.1.1a-1  libksba-1.3.5-1
              liblz4-1.8.3-1  liblzma-5.2.4-1  liblzo2-2.10-2
              libnettle-3.4.1-1  libnghttp2-1.36.0-1  libnpth-1.6-1
              libopenssl-1.1.1.b-1  libp11-kit-0.23.15-1  libpcre-8.43-1
              libpcre16-8.43-1  libpcre2_8-10.32-1  libpcre32-8.43-1
              libpcrecpp-8.43-1  libpcreposix-8.43-1  libpsl-0.20.2-3
              libreadline-7.0.005-1  libsqlite-3.21.0-4  libssh2-1.8.0-2
              libunistring-0.9.10-1  libutil-linux-2.32.1-1  libxml2-2.9.9-2
              libxslt-1.1.33-2  m4-1.4.18-2  mpfr-4.0.2-1
              ncurses-6.1.20180908-1  nettle-3.4.1-1  openssl-1.1.1.b-1
              p11-kit-0.23.15-1  pcre-8.43-1  pinentry-1.1.0-2  pkgfile-19-1
              rebase-4.4.4-1  sed-4.7-1  time-1.9-1  ttyrec-1.0.8-2
              util-linux-2.32.1-1  wget-1.20.1-2  xz-5.2.4-1

Total Download Size:    37.53 MiB
Total Installed Size:  208.34 MiB
Net Upgrade Size:       61.82 MiB

:: Proceed with installation? [Y/n]
```

Figure 11 – MSYS2 package manager update prompt

A prompt asking for permission to continue will appear again. Type Y and press enter to continue.

Download and installation will proceed, like the previous step this can take several minutes depending on network and computer performance.

Installation should complete without any error or warning messages.



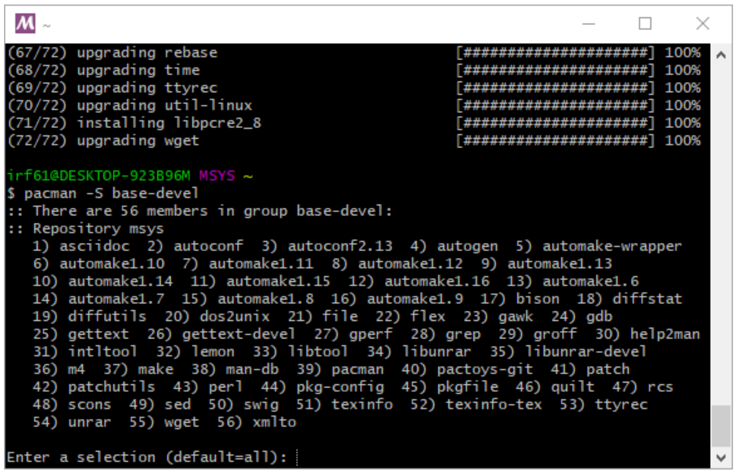
```
==> Importing owner trust values...
==> Updating trust database...
gpg: no need for a trustdb check
(55/72) upgrading libarchive                      [######################] 100%
(56/72) upgrading libargp                         [######################] 100%
(57/72) upgrading libgpgme                        [######################] 100%
(58/72) upgrading libpcre16                       [######################] 100%
(59/72) upgrading libpcre32                       [######################] 100%
(60/72) upgrading libpcrecpp                      [######################] 100%
(61/72) upgrading libpcreposix                    [######################] 100%
(62/72) upgrading libutil-linux                   [######################] 100%
(63/72) upgrading m4                              [######################] 100%
(64/72) upgrading pcre                            [######################] 100%
(65/72) upgrading xz                              [######################] 100%
(66/72) upgrading pkgfile                         [######################] 100%
(67/72) upgrading rebase                          [######################] 100%
(68/72) upgrading time                            [######################] 100%
(69/72) upgrading ttyrec                          [######################] 100%
(70/72) upgrading util-linux                      [######################] 100%
(71/72) installing libpcre2_8                     [######################] 100%
(72/72) upgrading wget                            [######################] 100%

irf61@DESKTOP-923B96M MSYS ~
$
```

Figure 12 – MSYS2 package manager update done

With the package manager updated and synced it is time to install the basic development utilities. This can be achieved by typing:

```
pacman -S base-devel
```

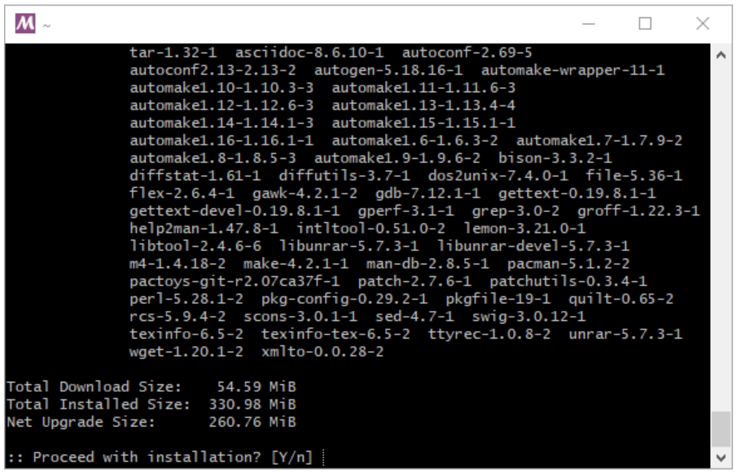
A list of packages to be installed will appear.



```
(67/72) upgrading rebase                                   [#####################] 100%
(68/72) upgrading time                                     [#####################] 100%
(69/72) upgrading ttyrec                                   [#####################] 100%
(70/72) upgrading util-linux                               [#####################] 100%
(71/72) installing libpcre2_8                              [#####################] 100%
(72/72) upgrading wget                                     [#####################] 100%

irf61@DESKTOP-923B96M MSYS ~
$ pacman -S base-devel
:: There are 56 members in group base-devel:
:: Repository msys
   1) asciidoc  2) autoconf  3) autoconf2.13  4) autogen  5) automake-wrapper
   6) automake1.10  7) automake1.11  8) automake1.12  9) automake1.13
   10) automake1.14  11) automake1.15  12) automake1.16  13) automake1.6
   14) automake1.7  15) automake1.8  16) automake1.9  17) bison  18) diffstat
   19) diffutils  20) dos2unix  21) file  22) flex  23) gawk  24) gdb
   25) gettext  26) gettext-devel  27) gperf  28) grep  29) groff  30) help2man
   31) intltool  32) lemon  33) libtool  34) libunrar  35) libunrar-devel
   36) m4  37) make  38) man-db  39) pacman  40) pactoys-git  41) patch
   42) patchutils  43) perl  44) pkg-config  45) pkgfile  46) quilt  47) rcs
   48) scons  49) sed  50) swig  51) texinfo  52) texinfo-tex  53) ttyrec
   54) unrar  55) wget  56) xmlto

Enter a selection (default=all):
```

Figure 13 – MSYS2 base-devel prompt

Press Enter to continue.

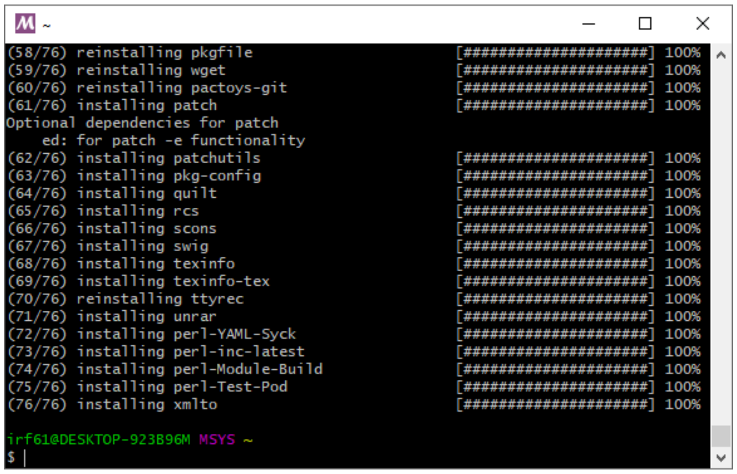


```
              tar-1.32-1  asciidoc-8.6.10-1  autoconf-2.69-5
              autoconf2.13-2.13-2  autogen-5.18.16-1  automake-wrapper-11-1
              automake1.10-1.10.3-3  automake1.11-1.11.6-3
              automake1.12-1.12.6-3  automake1.13-1.13.4-4
              automake1.14-1.14.1-3  automake1.15-1.15.1-1
              automake1.16-1.16.1-1  automake1.6-1.6.3-2  automake1.7-1.7.9-2
              automake1.8-1.8.5-3  automake1.9-1.9.6-2  bison-3.3.2-1
              diffstat-1.61-1  diffutils-3.7-1  dos2unix-7.4.0-1  file-5.36-1
              flex-2.6.4-1  gawk-4.2.1-2  gdb-7.12.1-1  gettext-0.19.8.1-1
              gettext-devel-0.19.8.1-1  gperf-3.1-1  grep-3.0-2  groff-1.22.3-1
              help2man-1.47.8-1  intltool-0.51.0-2  lemon-3.21.0-1
              libtool-2.4.6-6  libunrar-5.7.3-1  libunrar-devel-5.7.3-1
              m4-1.4.18-2  make-4.2.1-1  man-db-2.8.5-1  pacman-5.1.2-2
              pactoys-git-r2.07ca37f-1  patch-2.7.6-1  patchutils-0.3.4-1
              perl-5.28.1-2  pkg-config-0.29.2-1  pkgfile-19-1  quilt-0.65-2
              rcs-5.9.4-2  scons-3.0.1-1  sed-4.7-1  swig-3.0.12-1
              texinfo-6.5-2  texinfo-tex-6.5-2  ttyrec-1.0.8-2  unrar-5.7.3-1
              wget-1.20.1-2  xmlto-0.0.28-2

Total Download Size:    54.59 MiB
Total Installed Size:  330.98 MiB
Net Upgrade Size:      260.76 MiB

:: Proceed with installation? [Y/n]
```

Figure 14 – MSYS2 base-devel installation confirmation prompt

Type Y and then Enter to continue.

Download and installation will proceed, this can take several minutes depending on network and computer performance.



```
(58/76) reinstalling pkgfile                 [#####################] 100%
(59/76) reinstalling wget                    [#####################] 100%
(60/76) reinstalling pactoys-git             [#####################] 100%
(61/76) installing patch                     [#####################] 100%
Optional dependencies for patch
    ed: for patch -e functionality
(62/76) installing patchutils                [#####################] 100%
(63/76) installing pkg-config                [#####################] 100%
(64/76) installing quilt                     [#####################] 100%
(65/76) installing rcs                       [#####################] 100%
(66/76) installing scons                     [#####################] 100%
(67/76) installing swig                      [#####################] 100%
(68/76) installing texinfo                   [#####################] 100%
(69/76) installing texinfo-tex               [#####################] 100%
(70/76) reinstalling ttyrec                  [#####################] 100%
(71/76) installing unrar                     [#####################] 100%
(72/76) installing perl-YAML-Syck            [#####################] 100%
(73/76) installing perl-inc-latest           [#####################] 100%
(74/76) installing perl-Module-Build         [#####################] 100%
(75/76) installing perl-Test-Pod             [#####################] 100%
(76/76) installing xmlto                     [#####################] 100%

irf61@DESKTOP-923B96M MSYS ~
$
```

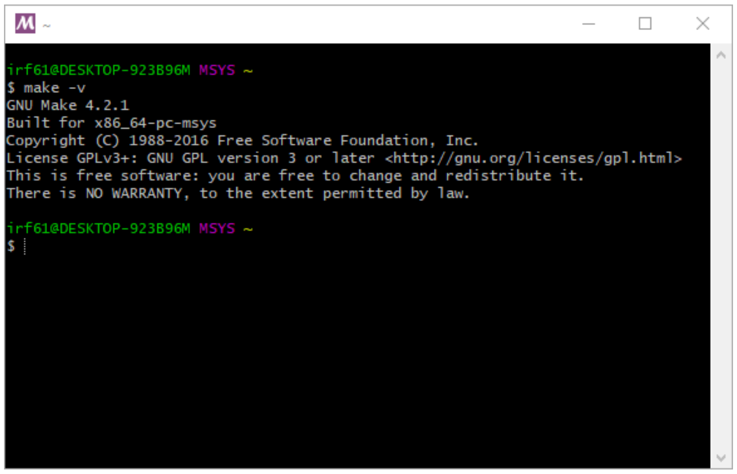
Figure 15 – MSYS2 base-devel installation completed

The installation should complete without any error or warning messages.

To test that the installation is functional, let's run a simple test by invoking the make utility as such:

```
make -v
```

The make utility should output its version information in the console.



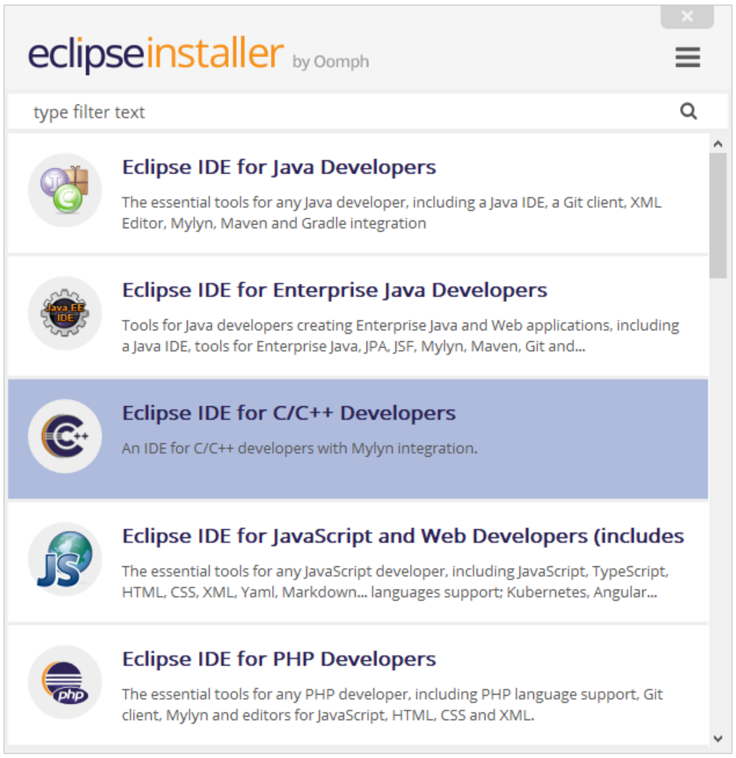
```
irf61@DESKTOP-923B96M MSYS ~
$ make -v
GNU Make 4.2.1
Built for x86_64-pc-msys
Copyright (C) 1988-2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

irf61@DESKTOP-923B96M MSYS ~
$
```

Figure 16 – MSYS2 make test output

MSYS2 and the required utilities are now installed, the console window can be closed.

2.2 Eclipse Installation

The next package to install is the Eclipse IDE. Start by executing the installer file. Note that if possible the 64-bit version of Eclipse is strongly recommended. In this example `eclipse-inst-win64.exe`, the following screen should appear.



eclipseinstaller by Oomph

type filter text

Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration

Eclipse IDE for Enterprise Java Developers
Tools for Java developers creating Enterprise Java and Web applications, including a Java IDE, tools for Enterprise Java, JPA, JSF, Mylyn, Maven, Git and...

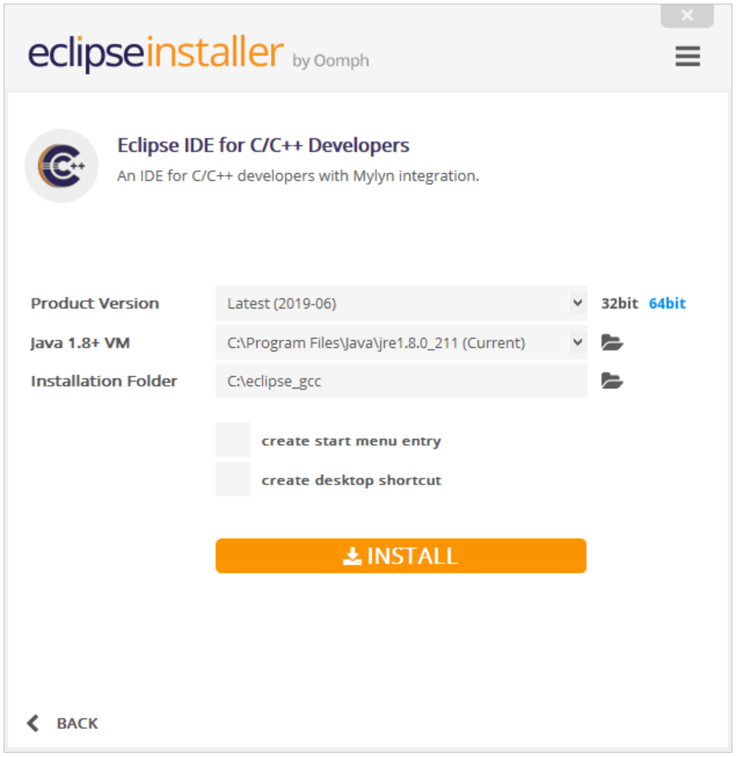
Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers with Mylyn integration.

Eclipse IDE for JavaScript and Web Developers (includes
The essential tools for any JavaScript developer, including JavaScript, TypeScript, HTML, CSS, XML, Yaml, Markdown... languages support; Kubernetes, Angular...

Eclipse IDE for PHP Developers
The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, HTML, CSS and XML.

Figure 17 – Eclipse install screen

Select "Eclipse IDE for C/C++ Developers" by clicking it. The install configuration screen should appear.

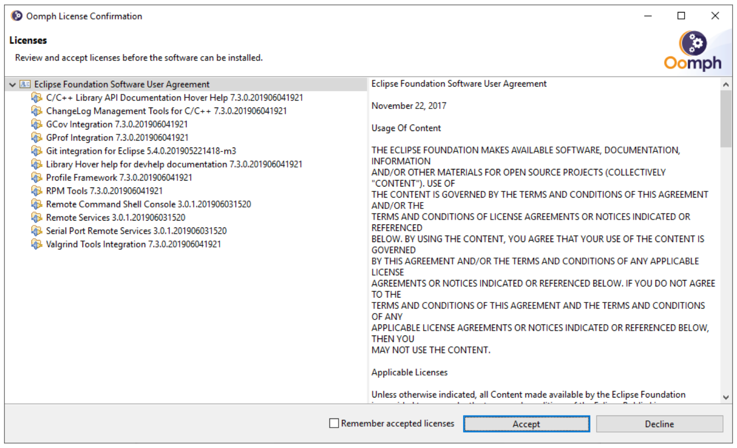

eclipseinstaller by Oomph
Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers with Mylyn integration.
32bit 64bit
create start menu entry
create desktop shortcut
INSTALL
BACK

Product Version: Latest (2019-06)
Java 1.8+ VM: C:\Program Files\Java\jre1.8.0_211 (Current)
Installation Folder: C:\eclipse_gcc

Figure 18 – Eclipse installation configuration dialog

The Product Version should be set to latest unless a specific version of Eclipse is desired. Right next to the version is the selection for 32 or 64-bit. 64 Bit is strongly recommended if it is supported by the host computer. Next is the Java virtual machine selection. If a suitable JVM is found, it will usually be selected. Note that the JVM must match with the 32 or 64-bit selection. For the installation directory, it is recommended to use a unique directory that is short with no spaces in the name as the eclipse install directory. In this example we will use `C:/eclipse_gcc`" as the install directory. The eclipse installer will automatically create an eclipse subdirectory inside the specified installation path. Finally, it is possible to create a start menu entry and/or a desktop shortcut if desired.

Click install to continue.



Oomph License Confirmation

Licenses

Review and accept licenses before the software can be installed.

- Eclipse Foundation Software User Agreement
  - C/C++ Library API Documentation Hover Help 7.3.0.201906041921
  - ChangeLog Management Tools for C/C++ 7.3.0.201906041921
  - GCov Integration 7.3.0.201906041921
  - GProf Integration 7.3.0.201906041921
  - Git integration for Eclipse 5.4.0.201905221418-m3
  - Library Hover help for devhelp documentation 7.3.0.201906041921
  - Profile Framework 7.3.0.201906041921
  - RPM Tools 7.3.0.201906041921
  - Remote Command Shell Console 3.0.1.201906031520
  - Remote Services 3.0.1.201906031520
  - Serial Port Remote Services 3.0.1.201906031520
  - Valgrind Tools Integration 7.3.0.201906041921

Eclipse Foundation Software User Agreement

November 22, 2017

Usage Of Content

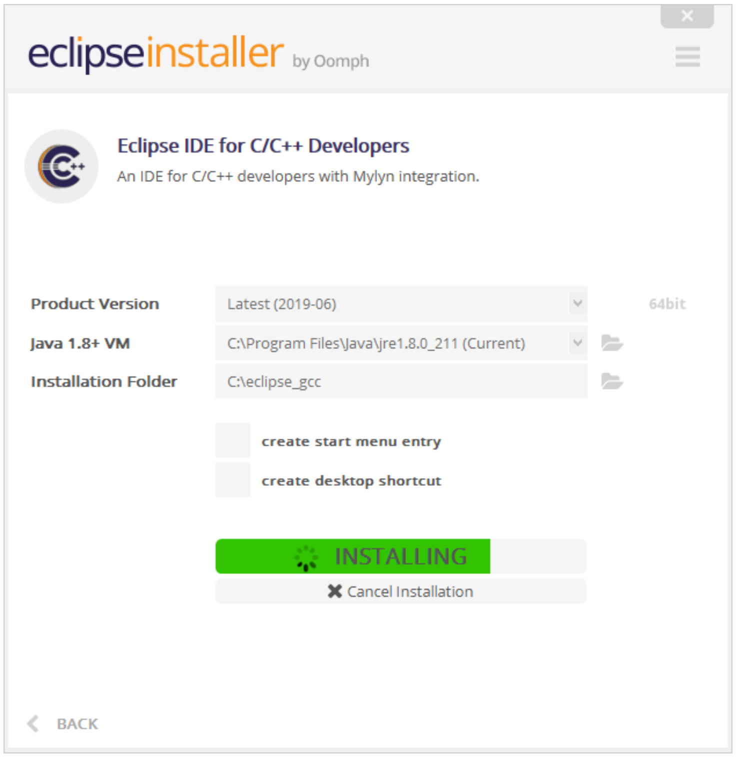
THE ECLIPSE FOUNDATION MAKES AVAILABLE SOFTWARE, DOCUMENTATION, INFORMATION
AND/OR OTHER MATERIALS FOR OPEN SOURCE PROJECTS (COLLECTIVELY "CONTENT"). USE OF
THE CONTENT IS GOVERNED BY THE TERMS AND CONDITIONS OF THIS AGREEMENT AND/OR THE
TERMS AND CONDITIONS OF LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED
BELOW. BY USING THE CONTENT, YOU AGREE THAT YOUR USE OF THE CONTENT IS GOVERNED
BY THIS AGREEMENT AND/OR THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE
AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW. IF YOU DO NOT AGREE TO THE
TERMS AND CONDITIONS OF THIS AGREEMENT AND THE TERMS AND CONDITIONS OF ANY
APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED BELOW, THEN YOU
MAY NOT USE THE CONTENT.

Applicable Licenses

Unless otherwise indicated, all Content made available by the Eclipse Foundation

Figure 19 – Eclipse installation licence dialog

The licence confirmation prompt should appear. Click "Accept" to continue.



Eclipse IDE for C/C++ Developers

An IDE for C/C++ developers with Mylyn integration.

Product Version: Latest (2019-06) 64bit

Java 1.8+ VM: C:\Program Files\Java\jre1.8.0_211 (Current)

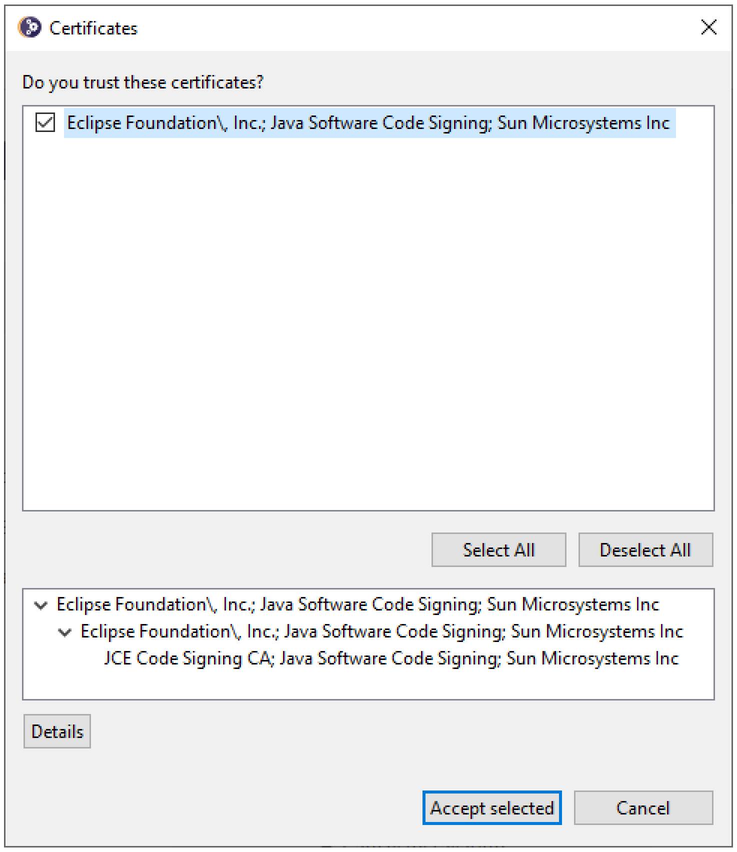
Installation Folder: C:\eclipse_gcc

create start menu entry

create desktop shortcut

Figure 20 – Eclipse installation progress dialog

The installation should progress. This may take some time depending on network performance.


Certificates

Do you trust these certificates?

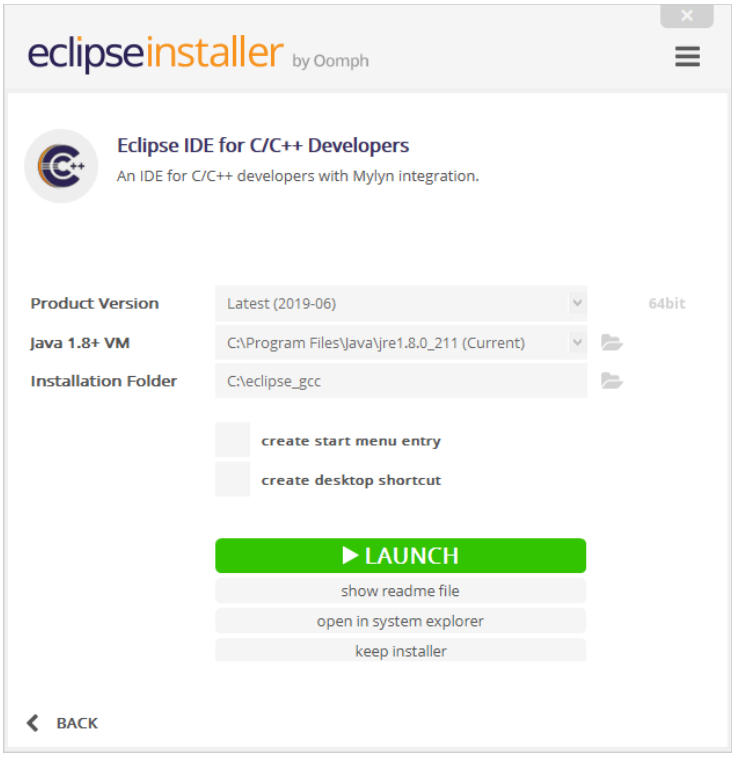
☑ Eclipse Foundation\, Inc.; Java Software Code Signing; Sun Microsystems Inc

Eclipse Foundation\, Inc.; Java Software Code Signing; Sun Microsystems Inc
Eclipse Foundation\, Inc.; Java Software Code Signing; Sun Microsystems Inc
JCE Code Signing CA; Java Software Code Signing; Sun Microsystems Inc

Figure 21 – Eclipse certificate trust prompt

The installer will likely prompt to trust of the remote server certificate where the Eclipse files will be downloaded. Check the checkbox next to the certificate name and click accept selected.

Installation should proceed.

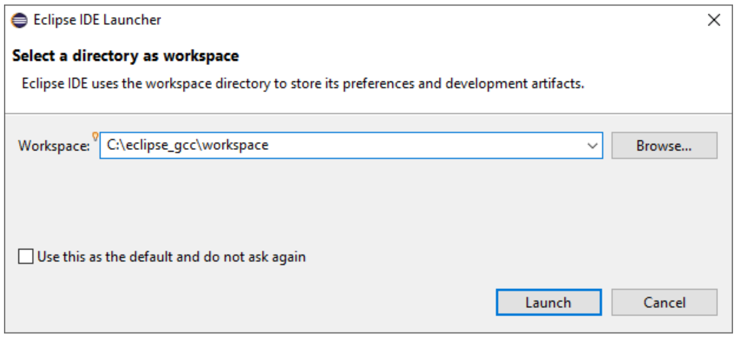


Product Version: Latest (2019-06)
Java 1.8+ VM: C:\Program Files\Java\jre1.8.0_211 (Current)
Installation Folder: C:\eclipse_gcc

Figure 22 – Eclipse installation complete and launch prompt

Launch the installation to check if it's working properly and create the first workspace. Do so by clicking Launch.

The workspace selection prompt should appear.



Select a directory as workspace
Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\eclipse_gcc\workspace

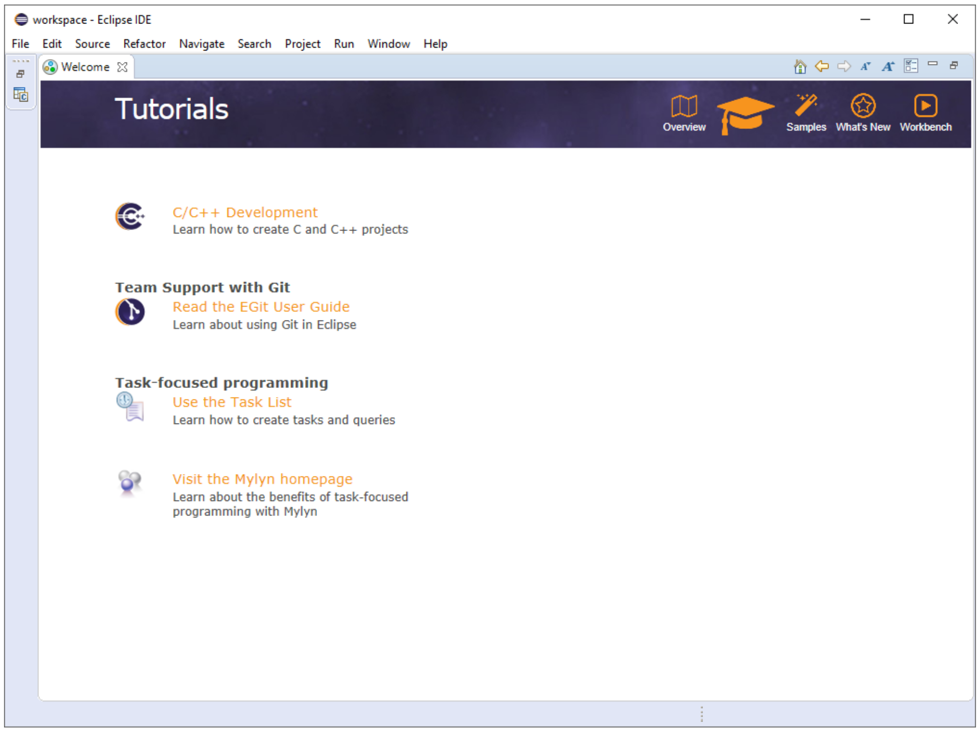
☐ Use this as the default and do not ask again

Figure 23 – Eclipse workspace selection prompt

It is strongly recommended to create a unique workspace that will be used exclusively for the projects related to the GCC toolchain that will be installed in this guide. The workspace selected should not have a path name that is excessively long and it is recommended not to use spaces in the path name either. To keep everything well contained and ordered this example will use `C:/eclipse_gcc/eclipse_workspace` as the workspace.

Click Launch.

The Eclipse welcome screen should appear.



C/C++ Development
Learn how to create C and C++ projects

Team Support with Git
Read the EGit User Guide
Learn about using Git in Eclipse

Task-focused programming
Use the Task List
Learn how to create tasks and queries

Visit the Mylyn homepage
Learn about the benefits of task-focused programming with Mylyn

Figure 24 – Eclipse welcome screen

Eclipse can be closed at this point as there are more packages to install before setting up the Eclipse IDE.

2.3 Toolchain Installation

This guide will use as an example the GNU Embedded Toolchain for Cortex-R and M distributed by ARM. However the procedure should be very similar to other embedded GCC toolchains. On Windows, if given a choice a simple ZIP or tarball archive is easier than an executable installer, this guide will use the former as an example.

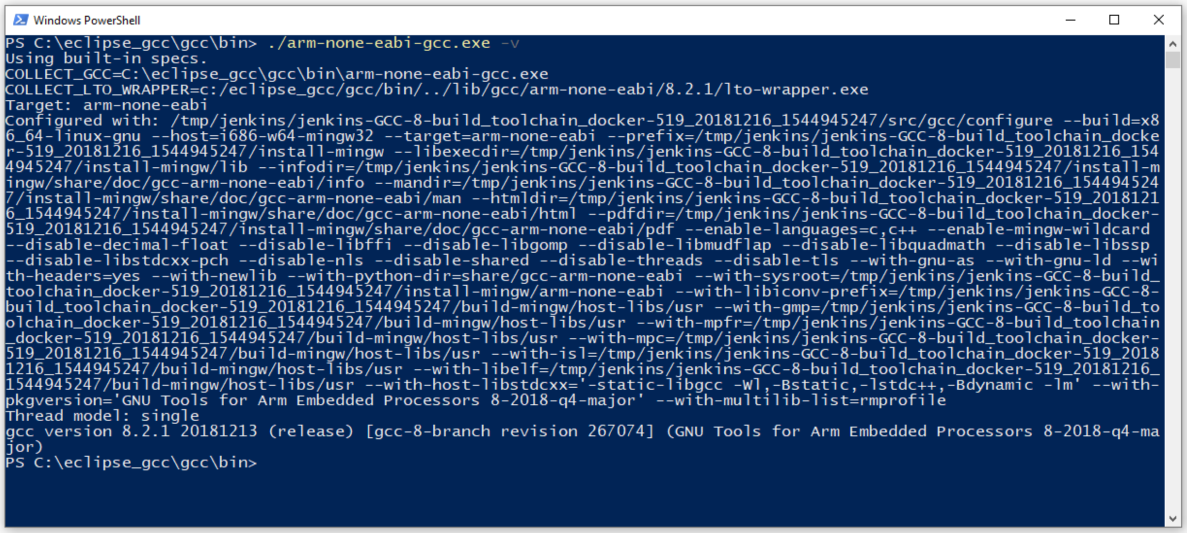
To install the toolchain it should simply be extracted to a suitable directory. As usual the full directory path should be short with no spaces. This example will use `C:/eclipse_gcc/gcc`.

The installation can be tested by opening a windows command prompt, changing the directory to the bin subdirectory of GCC installation directory and running the gcc executable. The full name of the GCC executable will be prefixed by the target triplet. For example `arm-none-eabi-gcc.exe`

After finding the name of the GCC executable, it can be executed as follows:

```
arm-none-eabi-gcc.exe -v
```

GCC should output various versions and build information to the command prompt.



```
PS C:\eclipse_gcc\gcc\bin> ./arm-none-eabi-gcc.exe -v
Using built-in specs.
COLLECT_GCC=C:\eclipse_gcc\gcc\bin\arm-none-eabi-gcc.exe
COLLECT_LTO_WRAPPER=c:/eclipse_gcc/gcc/bin/../lib/gcc/arm-none-eabi/8.2.1/lto-wrapper.exe
Target: arm-none-eabi
Configured with: /tmp/jenkins/jenkins-GCC-8-build_toolchain_docker-519_20181216_1544945247/src/gcc/configure --build=x8
6_64-linux-gnu --host=i686-w64-mingw32 --target=arm-none-eabi --prefix=/tmp/jenkins/jenkins-GCC-8-build_toolchain_docke
r-519_20181216_1544945247/install-mingw --libexecdir=/tmp/jenkins/jenkins-GCC-8-build_toolchain_docker-519_20181216_154
4945247/install-mingw/lib --infodir=/tmp/jenkins/jenkins-GCC-8-build_toolchain_docker-519_20181216_1544945247/install-m
ingw/share/doc/gcc-arm-none-eabi/info --mandir=/tmp/jenkins/jenkins-GCC-8-build_toolchain_docker-519_20181216_154494524
7/install-mingw/share/doc/gcc-arm-none-eabi/man --htmldir=/tmp/jenkins/jenkins-GCC-8-build_toolchain_docker-519_2018121
6_1544945247/install-mingw/share/doc/gcc-arm-none-eabi/html --pdfdir=/tmp/jenkins/jenkins-GCC-8-build_toolchain_docker-
519_20181216_1544945247/install-mingw/share/doc/gcc-arm-none-eabi/pdf --enable-languages=c,c++ --enable-mingw-wildcard
--disable-decimal-float --disable-libffi --disable-libgomp --disable-libmudflap --disable-libquadmath --disable-libssp
--disable-libstdcxx-pch --disable-nls --disable-shared --disable-threads --disable-tls --with-gnu-as --with-gnu-ld --wi
th-headers=yes --with-newlib --with-python-dir=share/gcc-arm-none-eabi --with-sysroot=/tmp/jenkins/jenkins-GCC-8-build_
toolchain_docker-519_20181216_1544945247/install-mingw/arm-none-eabi --with-libiconv-prefix=/tmp/jenkins/jenkins-GCC-8-
build_toolchain_docker-519_20181216_1544945247/build-mingw/host-libs/usr --with-gmp=/tmp/jenkins/jenkins-GCC-8-build_to
olchain_docker-519_20181216_1544945247/build-mingw/host-libs/usr --with-mpfr=/tmp/jenkins/jenkins-GCC-8-build_toolchain
_docker-519_20181216_1544945247/build-mingw/host-libs/usr --with-mpc=/tmp/jenkins/jenkins-GCC-8-build_toolchain_docker-
519_20181216_1544945247/build-mingw/host-libs/usr --with-isl=/tmp/jenkins/jenkins-GCC-8-build_toolchain_docker-519_2018
1216_1544945247/build-mingw/host-libs/usr --with-libelf=/tmp/jenkins/jenkins-GCC-8-build_toolchain_docker-519_20181216_
1544945247/build-mingw/host-libs/usr --with-host-libstdcxx='-static-libgcc -Wl,-Bstatic,-lstdc++,-Bdynamic -lm' --with-
pkgversion='GNU Tools for Arm Embedded Processors 8-2018-q4-major' --with-multilib-list=rmprofile
Thread model: single
gcc version 8.2.1 20181213 (release) [gcc-8-branch revision 267074] (GNU Tools for Arm Embedded Processors 8-2018-q4-ma
jor)
PS C:\eclipse_gcc\gcc\bin>
```

Figure 25 – GCC version information

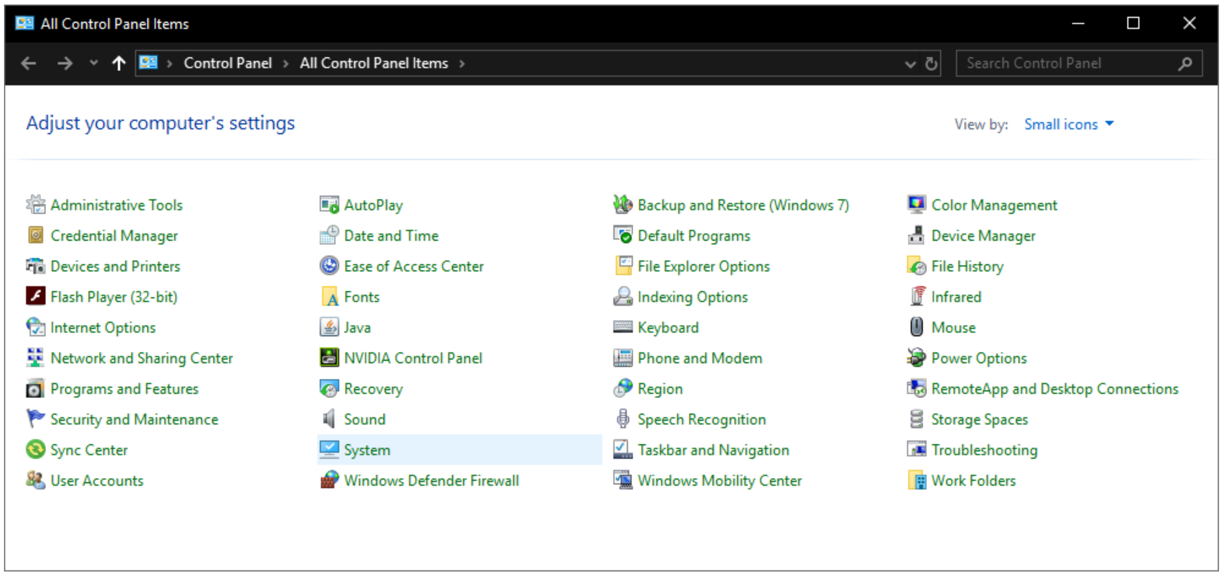
2.4 Environment Variable Setup

For the toolchain to be easily accessible by Eclipse, Make and other utilities it is preferable to have the toolchain binary location in the `PATH` environment variable.

One can notice that they are usually two `bin` directories available within a GCC toolchain distribution. One directory contains all of the toolchain's executables prefixed with the target triplet, i.e. `arm-none-eabi-gcc.exe` while the other contains generic names without the host triplets. When adding the toolchain to the `PATH` environment variable, it is important to use the directory where the executables are prefixed with the target triplet. First the other directory doesn't contain all the necessary executables, but may interfere with the OS native toolchain.

2.4.1 PATH Environnement Variable Setup

Start by opening the Windows Control Panel.


All Control Panel Items

Control Panel > All Control Panel Items >

Search Control Panel

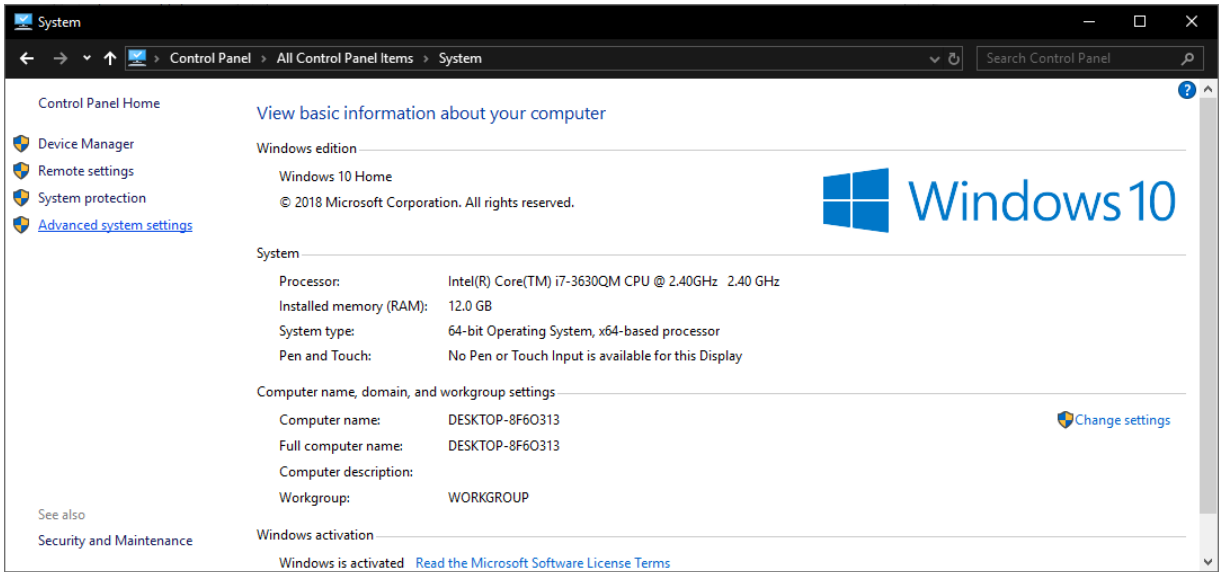
Adjust your computer's settings

View by: Small icons

Administrative Tools	AutoPlay	Backup and Restore (Windows 7)	Color Management
Credential Manager	Date and Time	Default Programs	Device Manager
Devices and Printers	Ease of Access Center	File Explorer Options	File History
Flash Player (32-bit)	Fonts	Indexing Options	Infrared
Internet Options	Java	Keyboard	Mouse
Network and Sharing Center	NVIDIA Control Panel	Phone and Modem	Power Options
Programs and Features	Recovery	Region	RemoteApp and Desktop Connections
Security and Maintenance	Sound	Speech Recognition	Storage Spaces
Sync Center	System	Taskbar and Navigation	Troubleshooting
User Accounts	Windows Defender Firewall	Windows Mobility Center	Work Folders

Figure 26 – Windows Control Panel

Click "System".


System

Control Panel > All Control Panel Items > System

Search Control Panel

Control Panel Home

Device Manager
Remote settings
System protection
Advanced system settings

See also
Security and Maintenance

View basic information about your computer

Windows edition

Windows 10 Home
© 2018 Microsoft Corporation. All rights reserved.

System

Processor:	Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz 2.40 GHz
Installed memory (RAM):	12.0 GB
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name:	DESKTOP-8F6O313
Full computer name:	DESKTOP-8F6O313
Computer description:	
Workgroup:	WORKGROUP

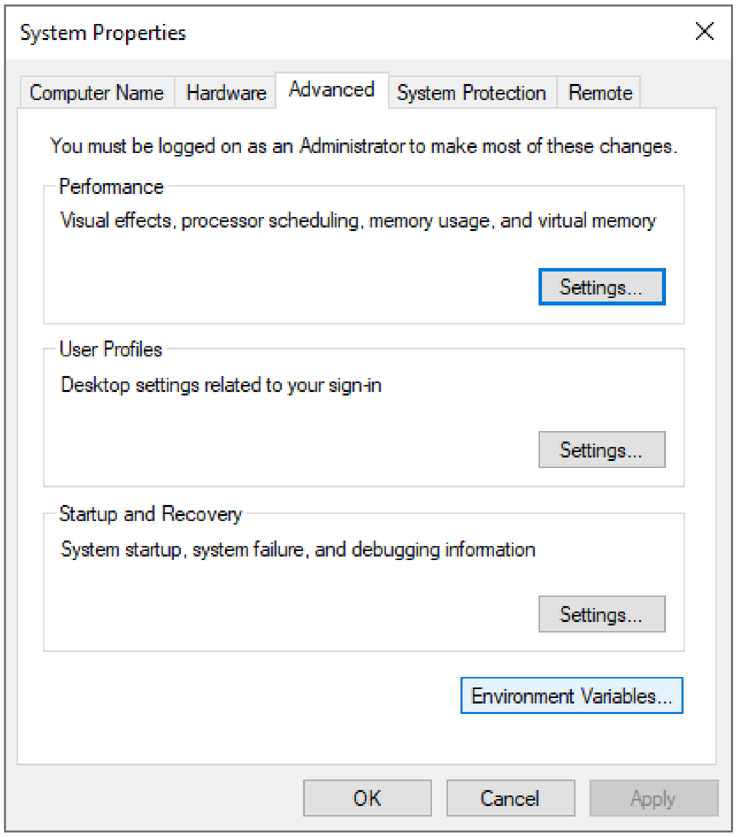
Change settings

Windows activation

Windows is activated Read the Microsoft Software License Terms

Figure 27 – Windows system settings

Click "Advanced System Settings".



You must be logged on as an Administrator to make most of these changes.

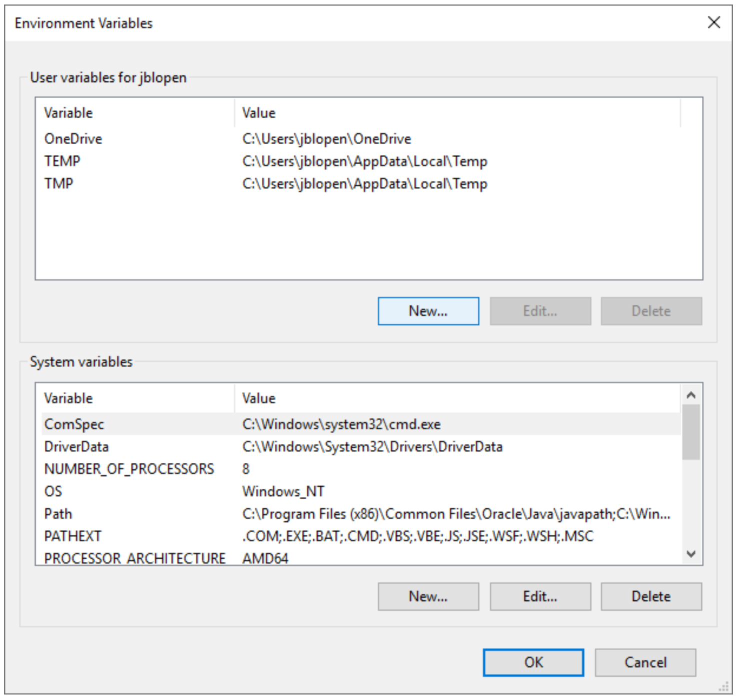
Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your sign-in

Startup and Recovery
System startup, system failure, and debugging information

Figure 28 – Windows Advanced System Settings

Click "Environment Variables".


Environment Variables

User variables for jblopen

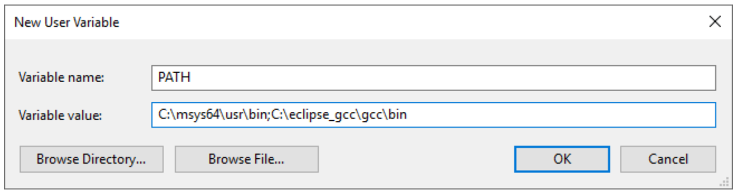
Variable	Value
OneDrive	C:\Users\jblopen\OneDrive
TEMP	C:\Users\jblopen\AppData\Local\Temp
TMP	C:\Users\jblopen\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Win...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

Figure 29 – Windows environnement variable configuration panel

In this screen, if an existing PATH environment variable exists, click it and then click "Edit..." otherwise click "New..."


New User Variable

Variable name: PATH

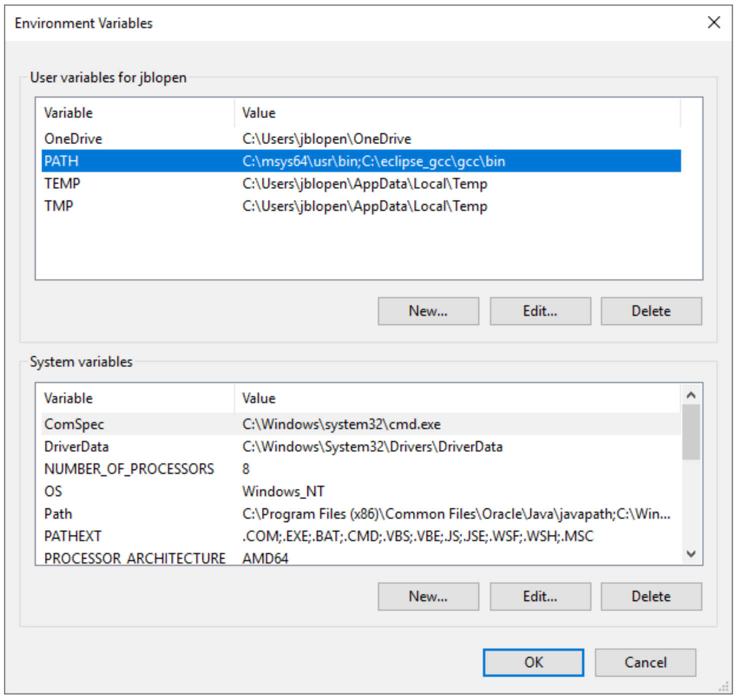
Variable value: C:\msys64\usr\bin;C:\eclipse_gcc\gcc\bin

Figure 30 – Windows new environment variable dialog

Two paths should be added either to the existing PATH variable or the new one. The `usr/bin` subdirectory of the MSYS2 installation and the bin subdirectory of the GCC installation. For the install directories chosen for this example, this results in the following two paths:

```
C:/msys64/usr/bin;C:/eclipse_gcc/gcc/bin;
```

Then click OK.


Environment Variables

User variables for jblopen

Variable	Value
OneDrive	C:\Users\jblopen\OneDrive
PATH	C:\msys64\usr\bin;C:\eclipse_gcc\gcc\bin
TEMP	C:\Users\jblopen\AppData\Local\Temp
TMP	C:\Users\jblopen\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Win...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

Figure 31 – Configured environment variables

Then OK again to exit the Environment Variables window.

Chapter

3

Eclipse Setup Guide (Linux)

This section will go over the Eclise IDE and toolchain setup for Linux. Instructions for Windows are available in the previous chapter. The following step-by-step instructions will go through the installation and configuration of the following components in order:

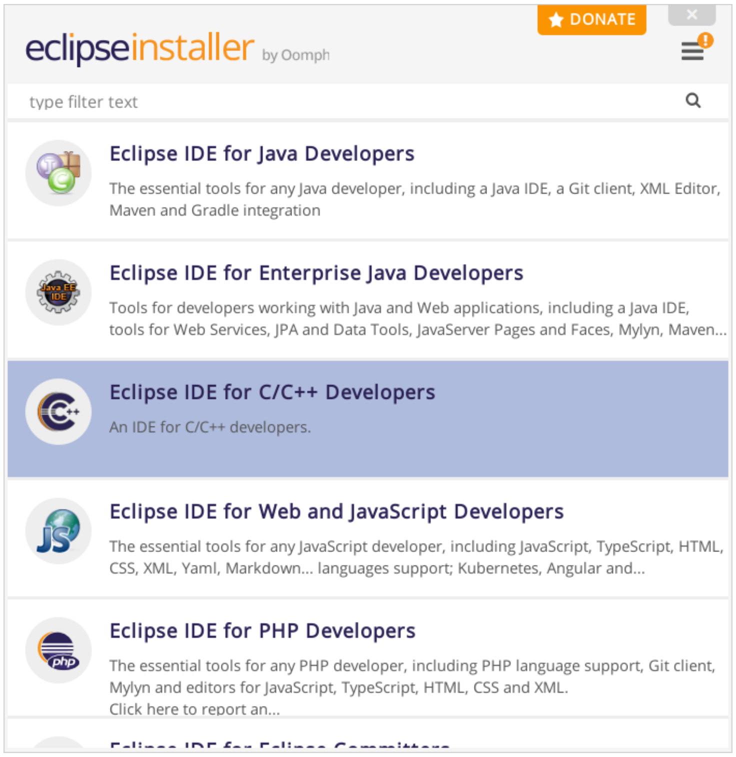
- Eclipse – 2020.06 – eclipse-inst-linux64.tar.gz
  www.eclipse.org
- GCC - 9-2020q2 - gcc-arm-none-eabi-9-2020-q2-update-x86_64-linux.tar.bz2
  https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm

The versions cited above are those that were used when writing this guide. Installation instructions for newer versions, if available, should be similar.

3.1 Eclipse Installation

The first package to install is the Eclipse IDE. As is usual with Linux, there's more than one way to install Eclipse. This guide will use the official Eclipse installer from eclipse.org which will install Eclipse within the current user's home directory. This is the simplest method and is also useful since it isolates the Eclipse installation from any native installation that may exist.

Start by extracting the installation archive and executing the installer file. Note that if possible the 64-bit version of Eclipse is strongly recommended. In this example `eclipse-inst`, the following screen should appear.



Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration

Eclipse IDE for Enterprise Java Developers
Tools for developers working with Java and Web applications, including a Java IDE, tools for Web Services, JPA and Data Tools, JavaServer Pages and Faces, Mylyn, Maven...

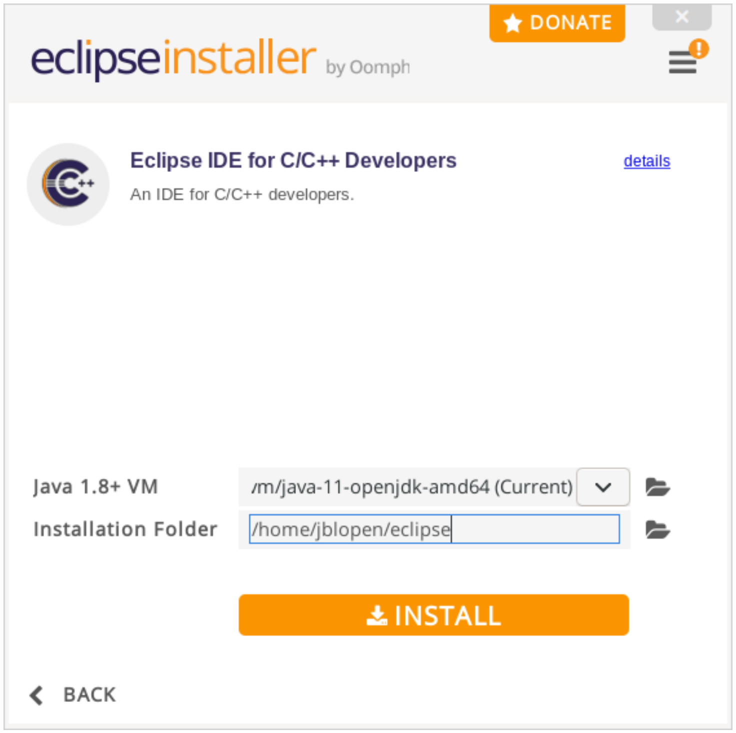
Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers.

Eclipse IDE for Web and JavaScript Developers
The essential tools for any JavaScript developer, including JavaScript, TypeScript, HTML, CSS, XML, Yaml, Markdown... languages support; Kubernetes, Angular and...

Eclipse IDE for PHP Developers
The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, TypeScript, HTML, CSS and XML.
Click here to report an...

Figure 32 – Eclipse install screen

Select ”Eclipse IDE for C/C++ Developers” by clicking it. The install configuration screen should appear.



Eclipse IDE for C/C++ Developers

An IDE for C/C++ developers.

Java 1.8+ VM: /m/java-11-openjdk-amd64 (Current)

Installation Folder: /home/jblopen/eclipse

Figure 33 – Eclipse installation configuration dialog

Set the installation directory, it is recommended to use a unique directory that is short with no spaces in the name as the eclipse install directory. In this example we will use `/home/jblopen/eclipse` as the installation directory. The eclipse installer will automatically create an eclipse subdirectory inside the specified installation path.

Click install. Installation should start and a progress bar should be displayed.

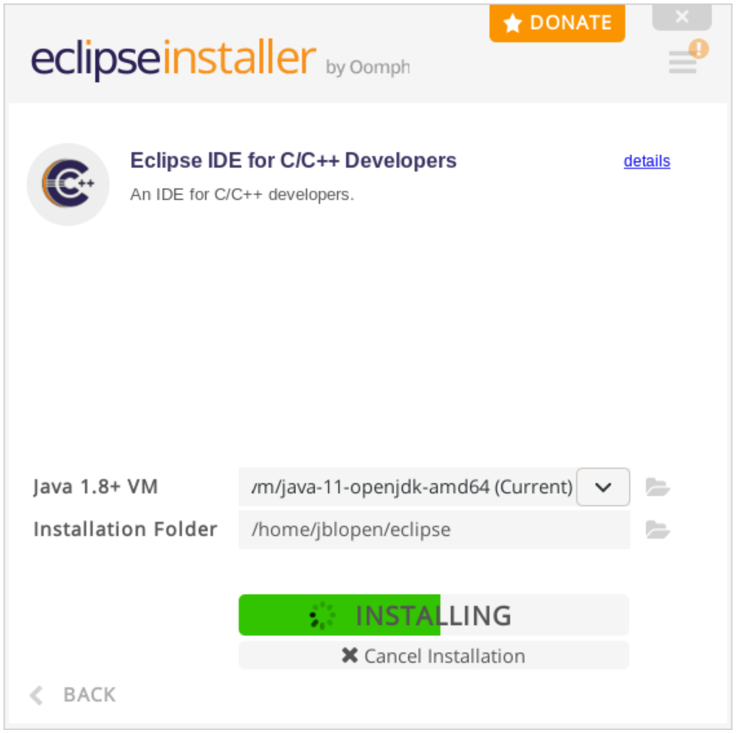
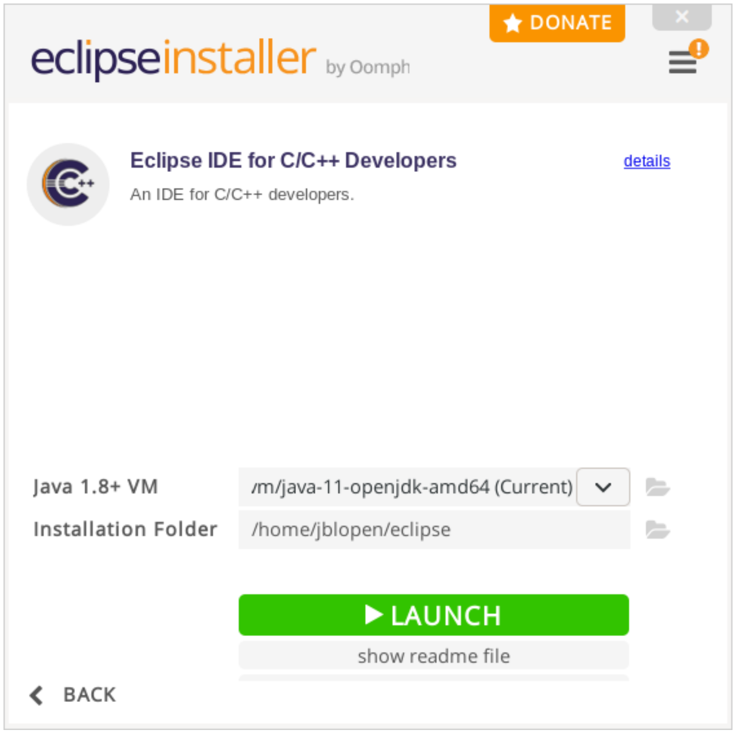

eclipseinstaller by Oomph
DONATE
Eclipse IDE for C/C++ Developers
details
An IDE for C/C++ developers.
Java 1.8+ VM
/m/java-11-openjdk-amd64 (Current)
Installation Folder
/home/jblopen/eclipse
INSTALLING
Cancel Installation
BACK

Figure 34 – Eclipse installation progress dialog

Installation should proceed.

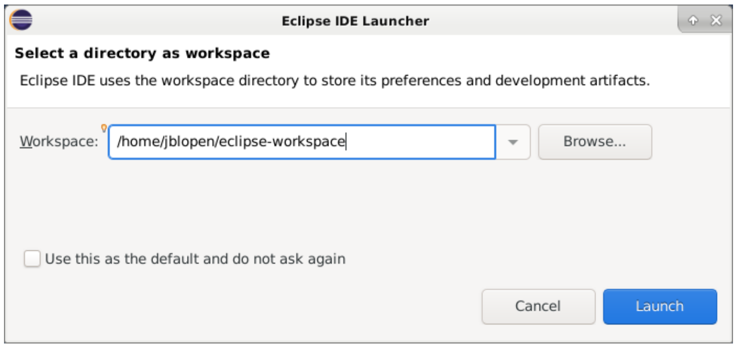

DONATE
eclipseinstaller by Oomph
Eclipse IDE for C/C++ Developers
details
An IDE for C/C++ developers.
Java 1.8+ VM
Installation Folder
LAUNCH
show readme file
BACK

Java 1.8+ VM: /m/java-11-openjdk-amd64 (Current)
Installation Folder: /home/jblopen/eclipse

Figure 35 – Eclipse installation complete and launch prompt

Launch the installation to check if it's working properly and create the first workspace. Do so by clicking Launch.

The workspace selection prompt should appear.



Select a directory as workspace
Eclipse IDE uses the workspace directory to store its preferences and development artifacts.
Workspace: /home/jblopen/eclipse-workspace
Use this as the default and do not ask again

Figure 36 – Eclipse workspace selection prompt

It is strongly recommended to create a unique workspace that will be used exclusively for the projects related to the GCC toolchain that will be installed in this guide. The workspace selected should not have a path name that is excessively long and it is recommended not to use spaces in the path name either. To keep everything well contained and ordered this example will use `/home/jblopen/eclipse-workspace` as the workspace.

Click Launch.

The Eclipse welcome screen should appear.

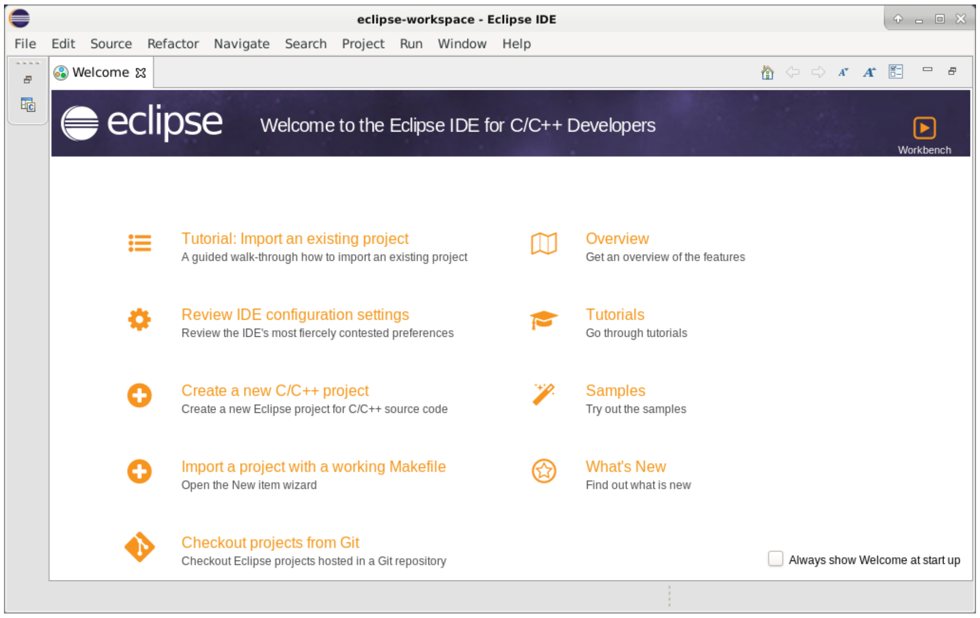

eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
eclipse
Welcome to the Eclipse IDE for C/C++ Developers
Workbench
Tutorial: Import an existing project
A guided walk-through how to import an existing project
Overview
Get an overview of the features
Review IDE configuration settings
Review the IDE's most fiercely contested preferences
Tutorials
Go through tutorials
Create a new C/C++ project
Create a new Eclipse project for C/C++ source code
Samples
Try out the samples
Import a project with a working Makefile
Open the New item wizard
What's New
Find out what is new
Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository
Always show Welcome at start up

Figure 37 – Eclipse welcome screen

Eclipse can be closed at this point as there are more packages to install before setting up the Eclipse IDE.

3.2 Toolchain Installation

This guide will use as an example the GNU Embedded Toolchain for Cortex-R and M distributed by ARM. However the procedure should be very similar to other embedded GCC toolchains.

First the downloaded toolchain archive should be extracted to a suitable location. In this example we will create a directory named gcc in the current user home directory and then extract the toolchain into that directory. This can be done, for example, by executing the following commands from the directory that contains the downloaded archive.

```
$ mkdir ~/gcc
$ tar -xvf gcc-arm-none-eabi-9-2020-q2-update-x86_64-linux.tar.bz2 -C ~/gcc/
```

That's about it for the toolchain installation on Linux. Optionally, the toolchain can be added to the PATH variable as shown in the next section.

3.3 GNU Make Installation

In addition to the toolchain and Eclipse it is usually necessary to at least have the GNU Make utility to process makefiles. This utility may already be installed, if not there's usually a package containing the minimal set of applications required to build.

Under Debian, Ubuntu and many others one can simply install the `build-essential` package. For example by running the following command:

```
$ sudo apt-get install build-essential
```

3.4 PATH Environment Variable Setup

It is not necessary to add the toolchain to the `PATH` environment variable to use the newly installed toolchain in Eclipse. However, it can be useful for building directly from the command line.

One can notice that they are usually two `bin` directories available within a GCC toolchain distribution. One directory contains all of the toolchain's executables prefixed with the target triplet, i.e. `arm-none-eabi-gcc` while the other contains generic names without the host triplets. When adding the toolchain to the PATH environment variable, it is important to use the directory where the executables are prefixed with the target triplet. First the other directory doesn't contain all the necessary executables, but may interfere with the OS native toolchain.

To add the toolchain to the PATH, add or edit the following line of the `.bashrc` file located in the home directory of the current user. It's usually a good idea to add it to the end of the list to prevent conflicts.

```
export PATH=$PATH:/home/jblopen/gcc/gcc-arm-none-eabi-9-2020-q2-update/bin
```

When updating the `PATH` variable like this Eclipse will not see the new toolchain if Eclipse is launched from the desktop environment. This can be fixed by launching Eclipse from the command line, for example:

```
$ nohup ./eclipse &
```

Additional methods of adding the toolchain to Eclipse are discussed in the management and makefile build projects chapters of this guide.

Chapter

4

Eclipse Managed Build Project Setup

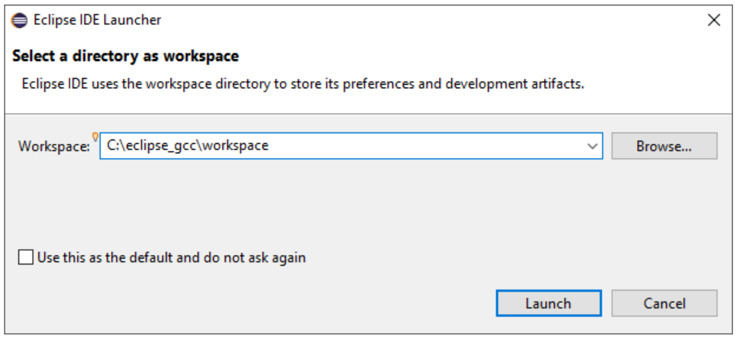
There are two main types of CDT projects that can be used in Eclipse. The first one is called a managed build where Eclipse CDT will manage the build. The other option is a Makefile project where a user-supplied Makefile is used with little help from Eclipse. Both methods have their pros and cons and this guide covers both.

This chapter will go over the basic steps required to create and set up a managed build project using the toolchain and Eclipse installation done in the first chapter. Note that this guide will not go over the project configuration for specific MCU or SoC as these settings can vary wildly and are beyond the scope of this guide.

4.1 Workspace Setup

Not much configuration is required at the workspace level when using a managed build, but it's a good idea to see if Eclipse has detected the toolchain that was added to the system PATH environment variable earlier in this guide. Note that the toolchain will only appear if it was added to the PATH variable.

First launch Eclipse if it's not already started. The workspace selection prompt should appear.


Eclipse IDE Launcher

Select a directory as workspace

Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\eclipse_gcc\workspace

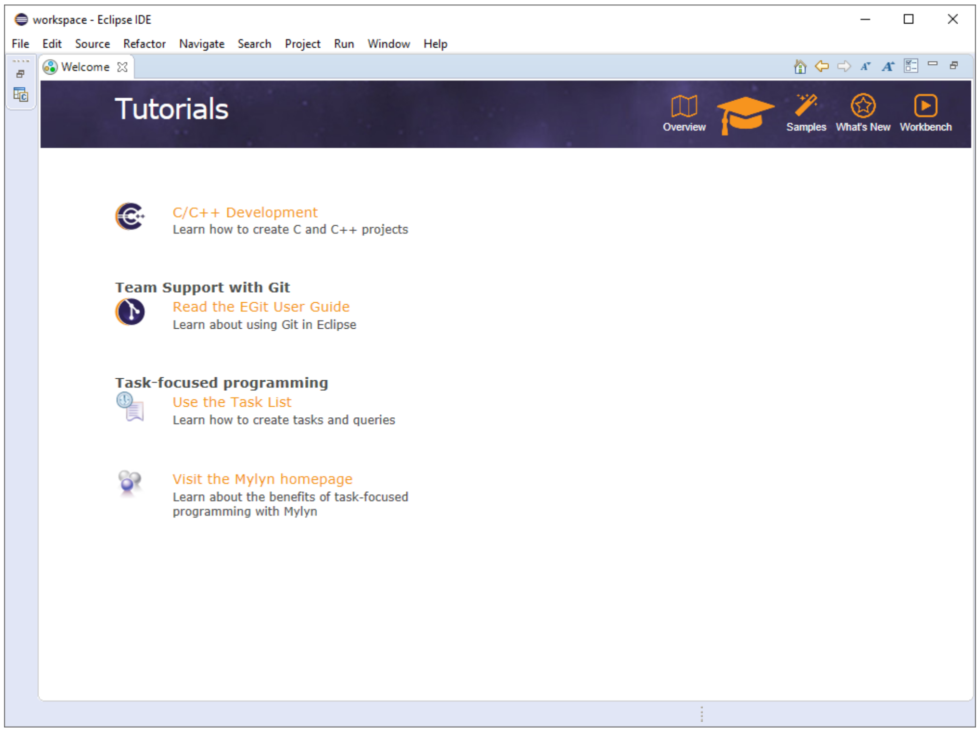
☐ Use this as the default and do not ask again

Figure 38 – Eclipse workspace selection prompt

If a workspace was created during the Eclipse installation select it, otherwise create it now. It is strongly recommended to create a unique workspace that will be used exclusively for the projects related to the GCC toolchain that will be installed in this guide. The workspace selected should not have a path name that is excessively long and it is recommended not to use spaces in the path name either. To keep everything well contained and ordered this example will use `C:/eclipse_gcc/eclipse_workspace` as the workspace.

Click Launch.

The Eclipse welcome screen should appear.



C/C++ Development
Learn how to create C and C++ projects

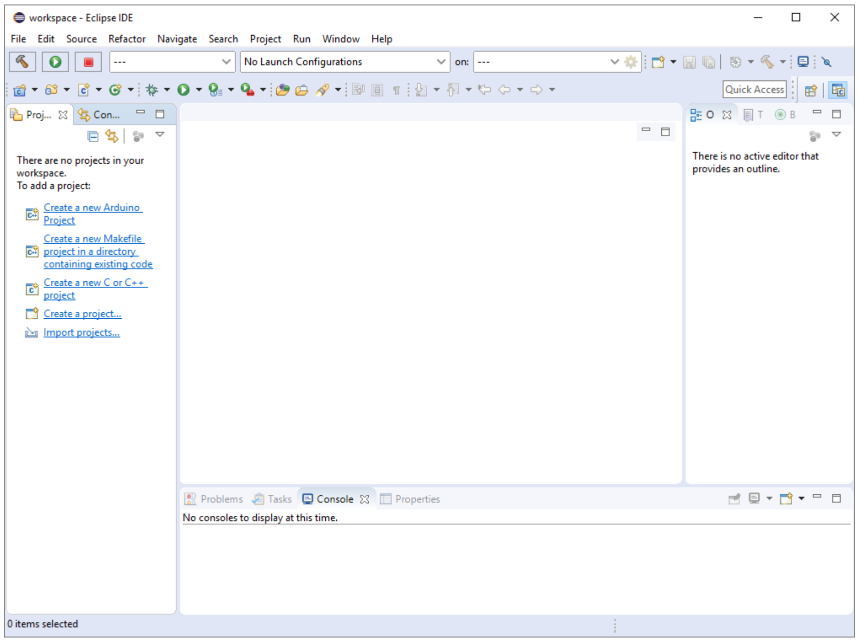
Team Support with Git
Read the EGit User Guide
Learn about using Git in Eclipse

Task-focused programming
Use the Task List
Learn how to create tasks and queries

Visit the Mylyn homepage
Learn about the benefits of task-focused programming with Mylyn

Figure 39 – Eclipse welcome screen

The welcome screen can be closed by pressing the X next to the Welcome tab. This reveals the main Eclipse view, which should be empty if this is a new workspace.



There are no projects in your workspace.
To add a project:
Create a new Arduino Project
Create a new Makefile project in a directory containing existing code
Create a new C or C++ project
Create a project...
Import projects...

There is no active editor that provides an outline.

No consoles to display at this time.

Figure 40 – Eclipse main view

Open the workspace preferences by clicking the Window->Preferences menu entry.

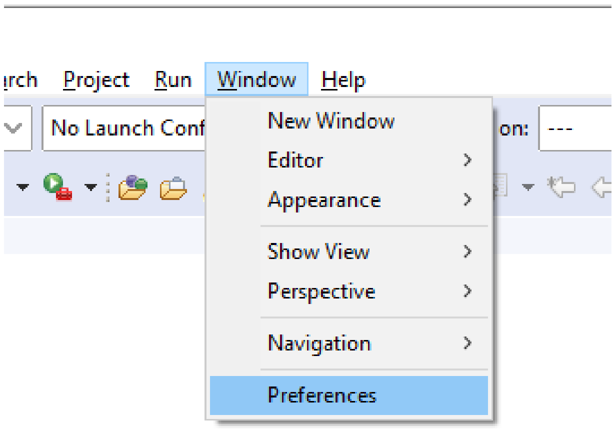
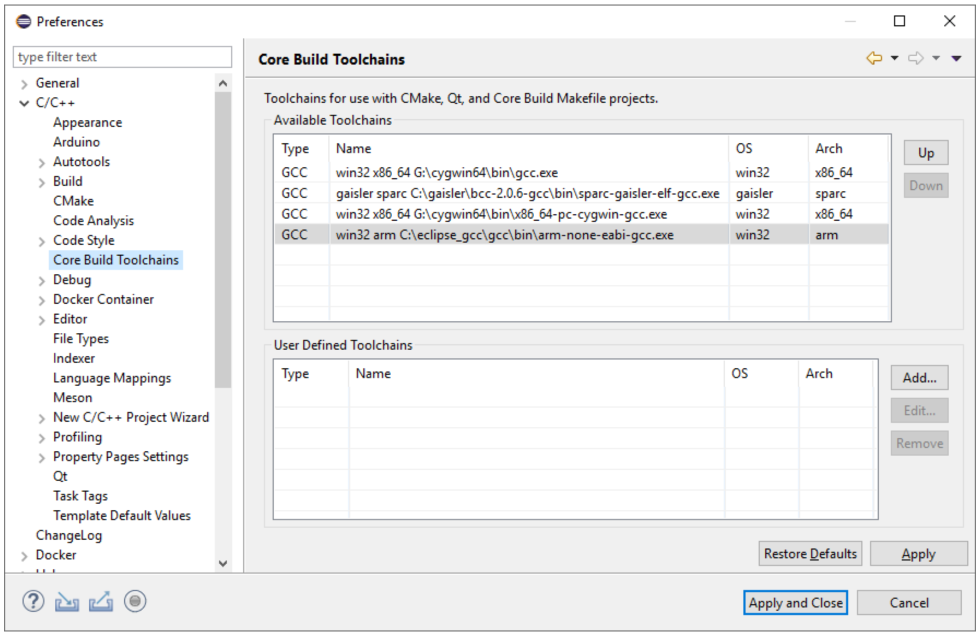


Figure 41 – Eclipse workspace preferences menu entry

Navigate to the ”Core Build Toolchains” subsection of the ”C/C++” section of the Preferences panel. This should display a list of detected toolchains. If the toolchain was installed correctly and was added to the PATH environment variable, it should appear in the list.



Toolchains for use with CMake, Qt, and Core Build Makefile projects.

Type	Name	OS	Arch
GCC	win32 x86_64 G:\cygwin64\bin\gcc.exe	win32	x86_64
GCC	gaisler sparc C:\gaisler\bcc-2.0.6-gcc\bin\sparc-gaisler-elf-gcc.exe	gaisler	sparc
GCC	win32 x86_64 G:\cygwin64\bin\x86_64-pc-cygwin-gcc.exe	win32	x86_64
GCC	win32 arm C:\eclipse_gcc\gcc\bin\arm-none-eabi-gcc.exe	win32	arm

Figure 42 – CDT Core Build Toolchains

In this example the toolchain for ARM installed earlier was detected with the path to the main GCC executable being C:/eclipse_gcc/gcc/bin/arm-none-eabi-gcc.exe correctly displayed.

That is all the workspace level items of importance for a managed build. The next step is to create the managed build project proper.

4.2 Managed Build Project Setup

To create a new managed build project select the File->New->C/C++ Project entry from the File Menu.

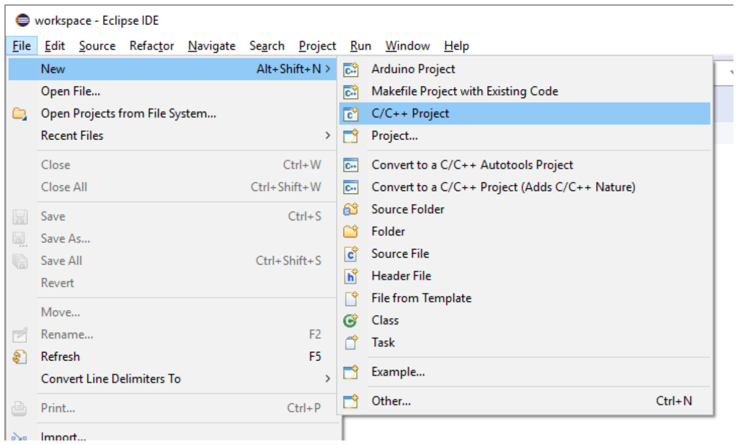
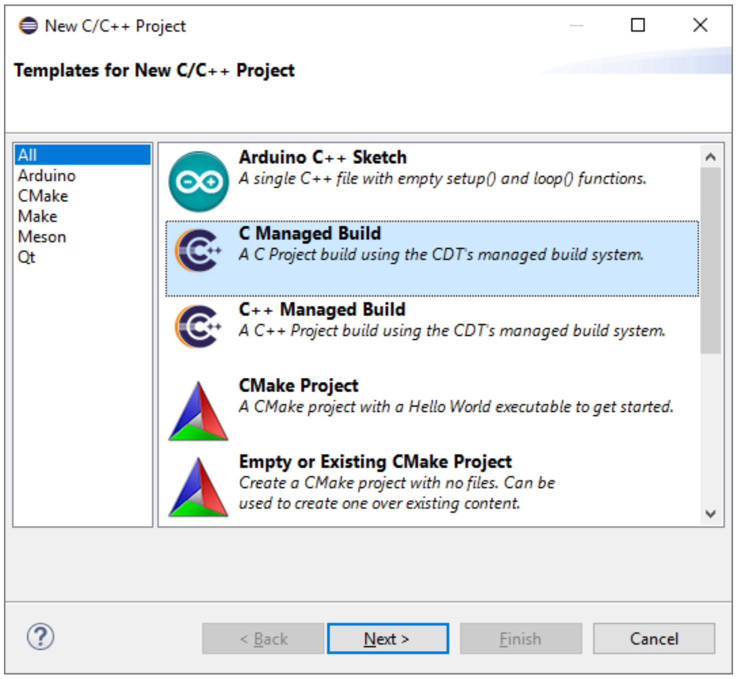


Figure 43 – New C/C++ Project menu entry

In the New C/C++ project dialog select ”C Managed Build”.



Arduino C++ Sketch
A single C++ file with empty setup() and loop() functions.

C Managed Build
A C Project build using the CDT's managed build system.

C++ Managed Build
A C++ Project build using the CDT's managed build system.

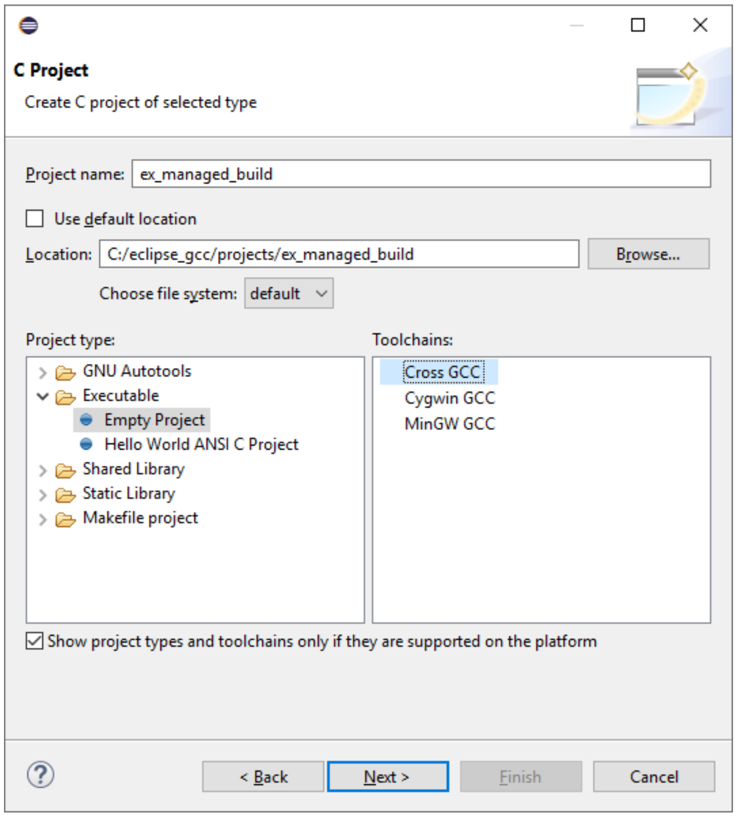
CMake Project
A CMake project with a Hello World executable to get started.

Empty or Existing CMake Project
Create a CMake project with no files. Can be used to create one over existing content.

Figure 44 – New C/C++ Project Dialog

Click Next. In this next screen, the project name and location should be set as well as the toolchain. Type a name for the project, for example ”ex_managed_build” and select a location. If the ”Use default location” box is checked the project will be created inside the current workspace, otherwise a directory can be typed or selected using the ”Browse” button. As usual it is recommended to keep the project path short without any spaces.

It is also important to select an empty executable project along with the ”Cross GCC” toolchain as displayed in the following screenshot.


C Project

Create C project of selected type

Project name: ex_managed_build

☐ Use default location

Location: C:/eclipse_gcc/projects/ex_managed_build

Choose file system: default

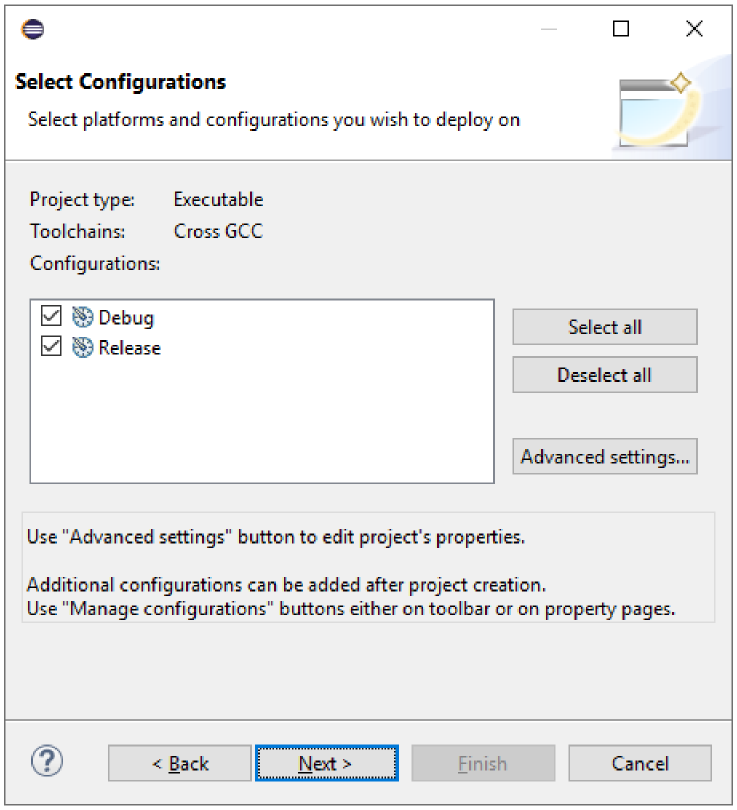
Project type:
- GNU Autotools
- Executable
  - Empty Project
  - Hello World ANSI C Project
- Shared Library
- Static Library
- Makefile project

Toolchains:
- Cross GCC
- Cygwin GCC
- MinGW GCC

☑ Show project types and toolchains only if they are supported on the platform

Figure 45 – New C/C++ Project name and toolchain selection

Click Next. The Select Configurations dialog will display.


Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Executable
Toolchains: Cross GCC
Configurations:

- [x] Debug
- [x] Release

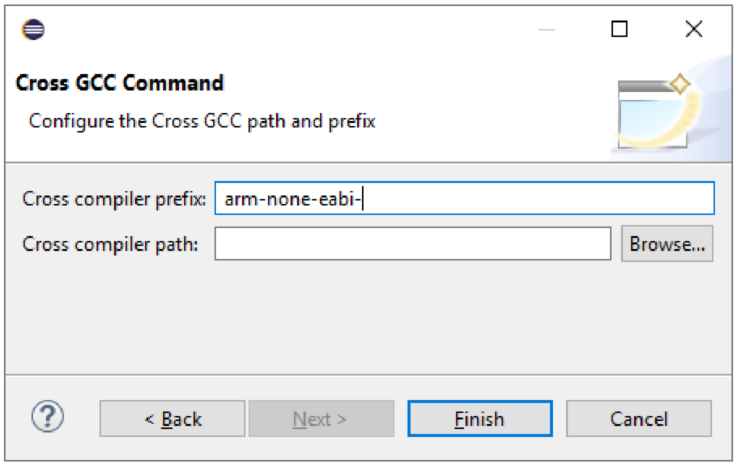
Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

Figure 46 – Project Configurations dialog

There is nothing to change on this screen other than clicking next.

The next ”Cross GCC Command” dialog is rather important, this is where we select the toolchain to use for building. To do so it’s necessary to type in the prefix of the GCC toolchain. Most of the time all the executables from a GCC cross compiler will be prefixed with a target triplet . For example, in our ARMv7 toolchain the target triplet is `arm-none-eabi` and the full prefix is ”arm-none-eabi-”. The full prefix should be typed in the ”Cross Compiler prefix:” box. If the toolchain was added to the `PATH` environment variable, it is not necessary to enter the path to the toolchain. Otherwise the toolchain bin directory should be entered.


Cross GCC Command

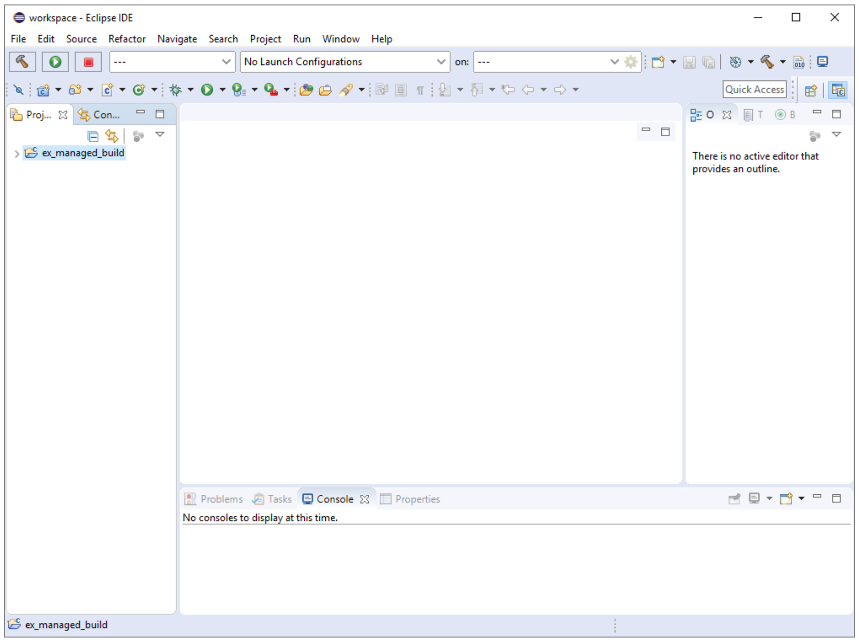
Configure the Cross GCC path and prefix

Cross compiler prefix: arm-none-eabi-

Cross compiler path:

Figure 47 – Cross GCC Command dialog

Click Finish. The project should now be created and available in the workspace.



There is no active editor that provides an outline.

No consoles to display at this time.

Figure 48 – Eclipse main view with new project

Files can now be added and the build settings set for the particular target for the project. The exact settings are outside the scope of this guide but let's at least add a simple main file to test the build.

To add a new file right-click on the project, or select the project and click the Project->New->Files menu item.

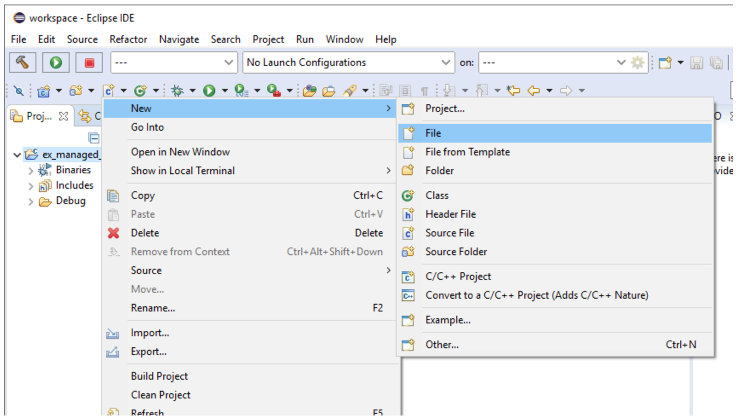


Figure 49 – Project new file menu entry

Type "main.c" in the "Create New File" Dialog and click "Finish".

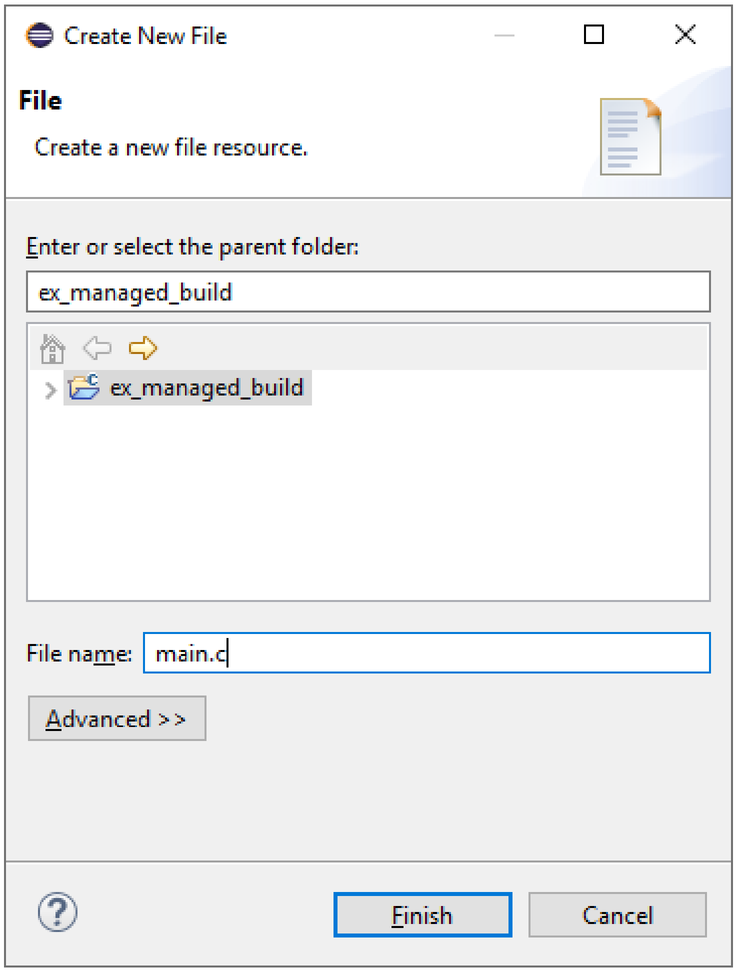
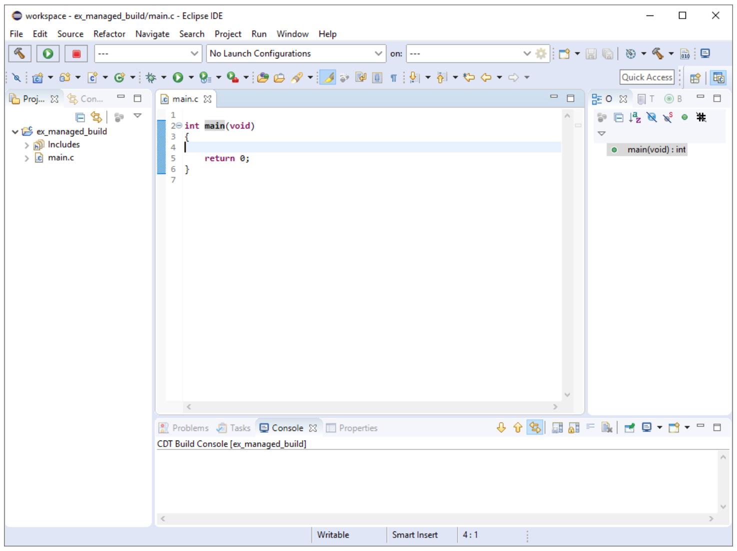


Figure 50 – Create new file dialog

Let’s fill the new source file with a minimal C main function.



```
int main(void)
{

    return 0;
}
```

workspace - ex_managed_build/main.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
No Launch Configurations
ex_managed_build
Includes
main.c
main(void) : int
CDT Build Console [ex_managed_build]
Writable
Smart Insert
4 : 1

Figure 51 – C Main function

If you try building the minimal C program, it’s quite likely that an error similar to the following linker error is displayed.

```
exit.c:(.text.exit+0x2c): undefined reference to `_exit'
```

To fix that common issue `--specs=nosys.specs` must be added to the linker command-line options. Note that if the error doesn’t happen it is important not to add the linker option.

To add the option first open the project properties, by right-clicking on the project or by selecting the project and click the Project->Properties menu entry.

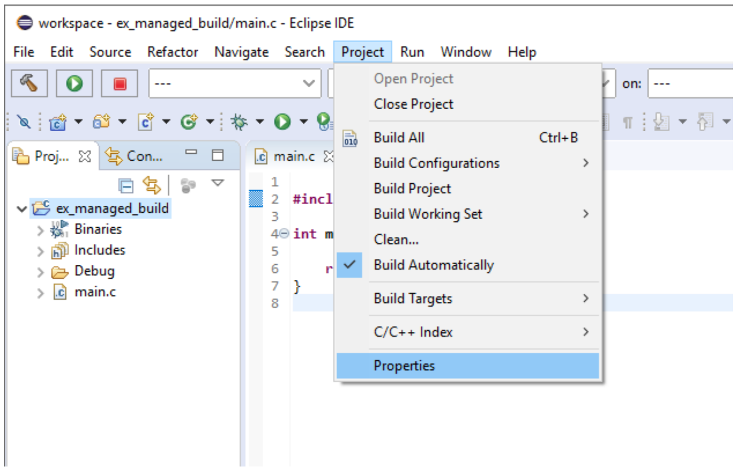
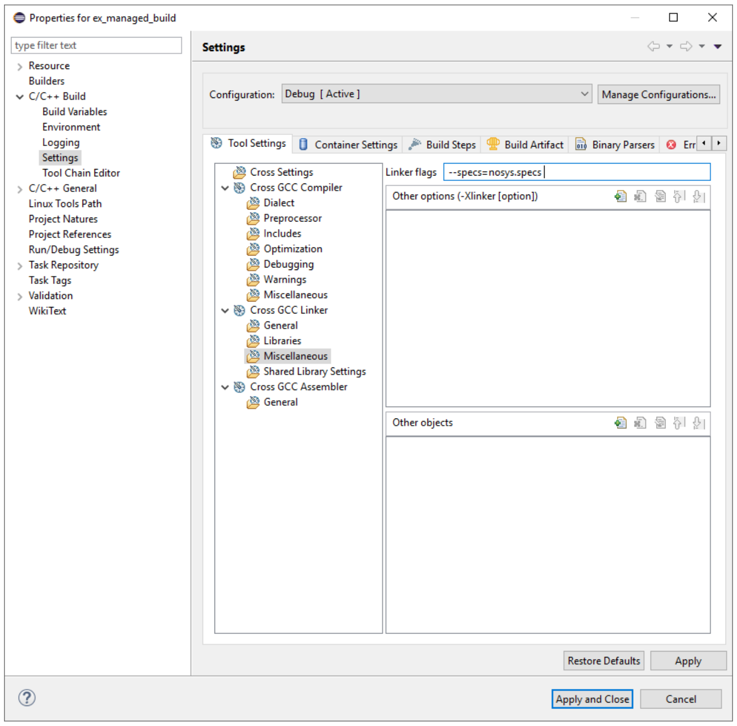


Figure 52 – Project Properties menu item

Navigate to the ”Miscellaneous” category under ”Cross GCC Linker” and add the option to the ”Linker flags” box.



Configuration: Debug [Active]

Linker flags: --specs=nosys.specs

Figure 53 – Project linker configuration

At this point the minimal program should build and everything is set up to start development of an embedded project using the chosen toolchain.

Chapter 5

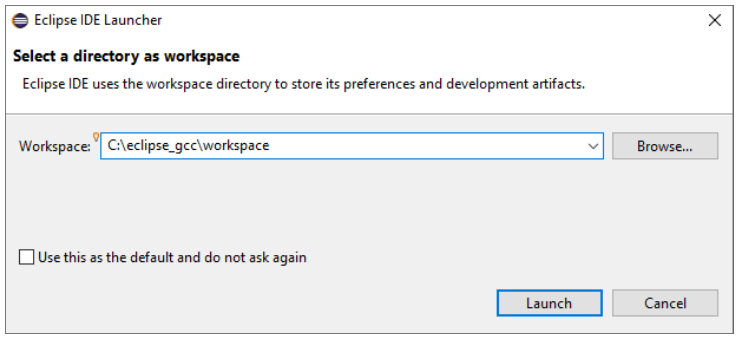
Eclipse Makefile Project Setup

The second type of Eclipse project, a Makefile project, is more complicated to setup but offers greater control and flexibility over the build process. A Makefile based project is also independent of Eclipse and can be built and maintained easily outside of Eclipse. This can be useful for automated builds.

5.1 Workspace Setup

Some configuration is helpful at the workspace level when using a Makefile build, it is also a good idea to see if Eclipse has detected the toolchain that was added to the system PATH environment variable earlier in this guide.

First launch Eclipse if it's not already started. The workspace selection prompt should appear.


Eclipse IDE Launcher

Select a directory as workspace

Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

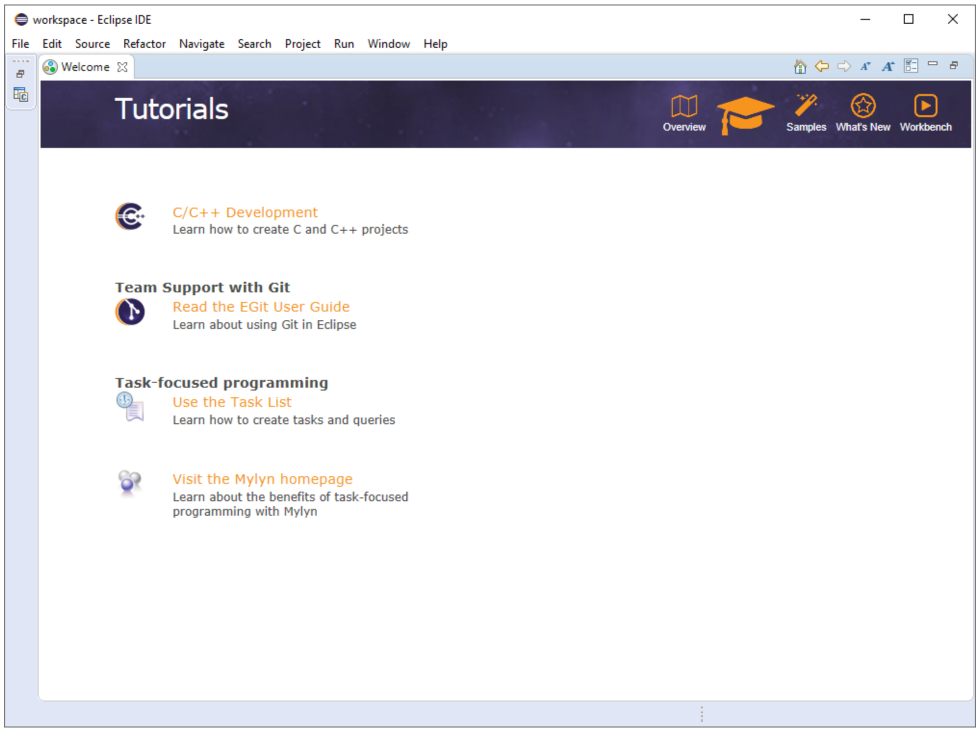
Workspace: C:\eclipse_gcc\workspace

Figure 54 – Eclipse workspace selection prompt

If a workspace was created during the Eclipse installation select it, otherwise create it now. It is strongly recommended to create a unique workspace that will be used exclusively for the projects related to the GCC toolchain that will be installed in this guide. The workspace selected should not have a path name that is excessively long and it is recommended not to use spaces in the path name either. To keep everything well contained and ordered this example will use ”C:/eclipse_gcc/eclipse_workspace” as the workspace.

Click "Launch".

The Eclipse welcome screen should appear.



C/C++ Development
Learn how to create C and C++ projects

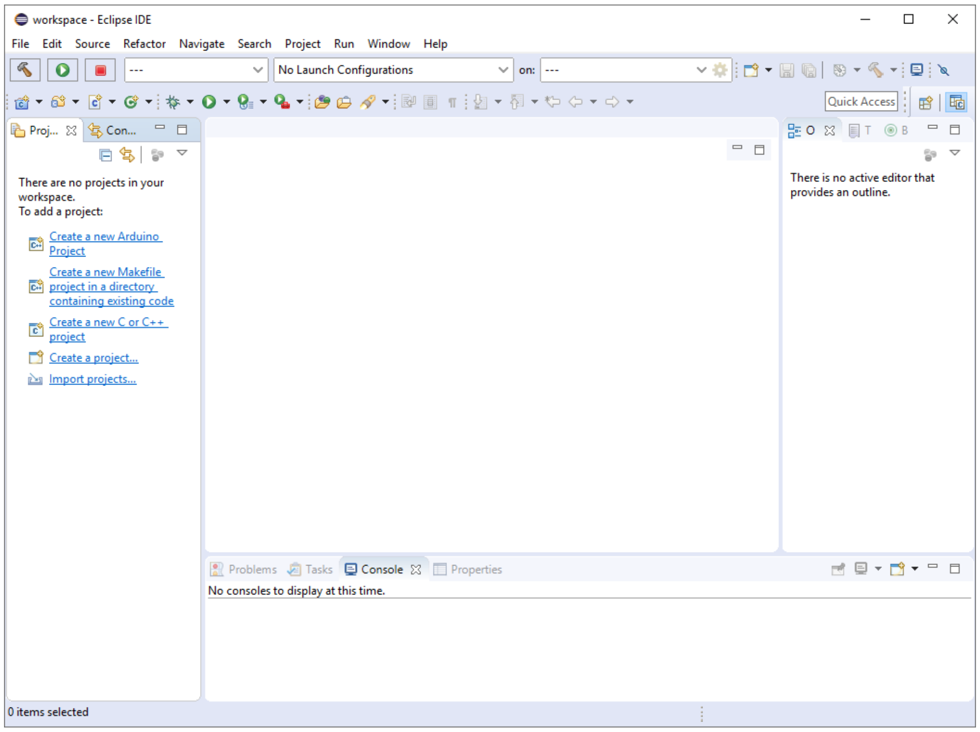
Team Support with Git
Read the EGit User Guide
Learn about using Git in Eclipse

Task-focused programming
Use the Task List
Learn how to create tasks and queries

Visit the Mylyn homepage
Learn about the benefits of task-focused programming with Mylyn

Figure 55 – Eclipse welcome screen

The welcome screen can be closed by pressing the X next to the Welcome tab. This reveals the main Eclipse view, which should be empty if this is a new workspace.



There are no projects in your workspace.
To add a project:
Create a new Arduino Project
Create a new Makefile project in a directory containing existing code
Create a new C or C++ project
Create a project...
Import projects...

There is no active editor that provides an outline.

No consoles to display at this time.

Figure 56 – Eclipse main view

Open the workspace preferences by clicking the Window->Preferences menu entry.

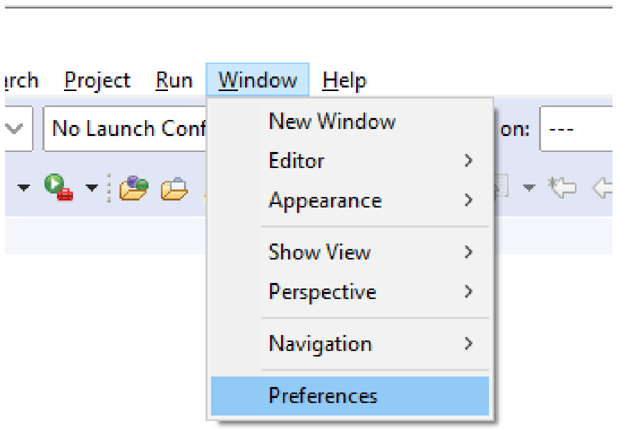
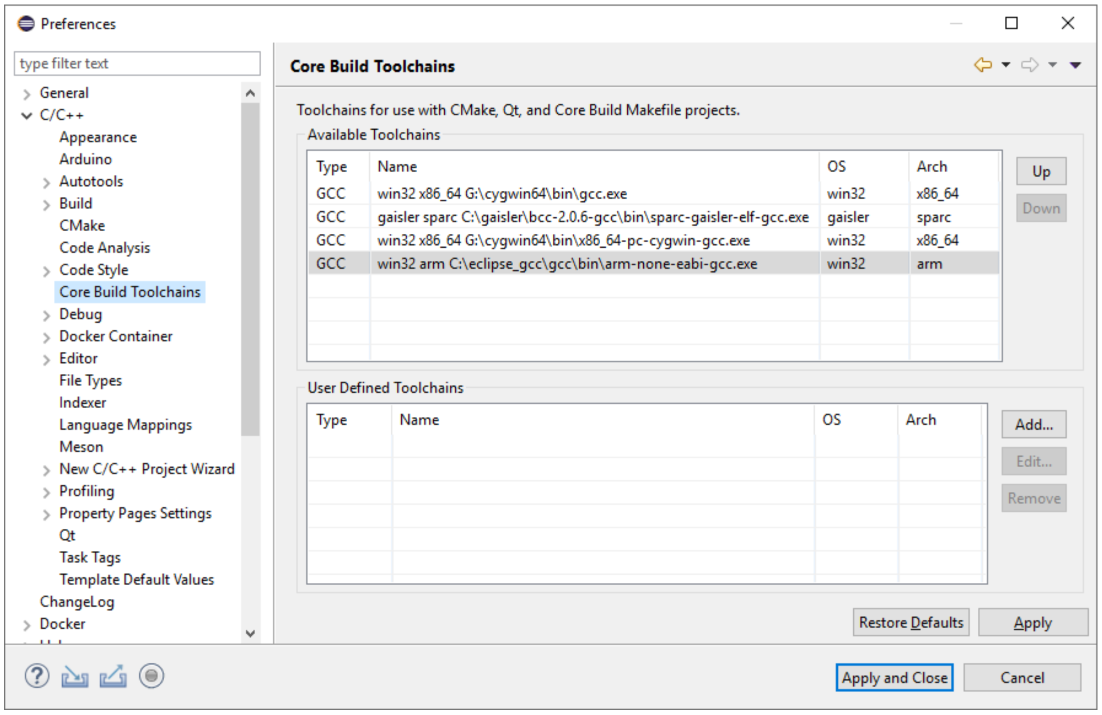


Figure 57 – Eclipse workspace preferences menu entry

Navigate to the "Core Build Toolchains" subsection of the "C/C++" section of the Preferences panel. This should display a list of detected toolchains. If the toolchain was installed correctly and was added to the PATH environment variable, it should appear in the list.



Toolchains for use with CMake, Qt, and Core Build Makefile projects.

Available Toolchains

Type	Name	OS	Arch
GCC	win32 x86_64 G:\cygwin64\bin\gcc.exe	win32	x86_64
GCC	gaisler sparc C:\gaisler\bcc-2.0.6-gcc\bin\sparc-gaisler-elf-gcc.exe	gaisler	sparc
GCC	win32 x86_64 G:\cygwin64\bin\x86_64-pc-cygwin-gcc.exe	win32	x86_64
GCC	win32 arm C:\eclipse_gcc\gcc\bin\arm-none-eabi-gcc.exe	win32	arm

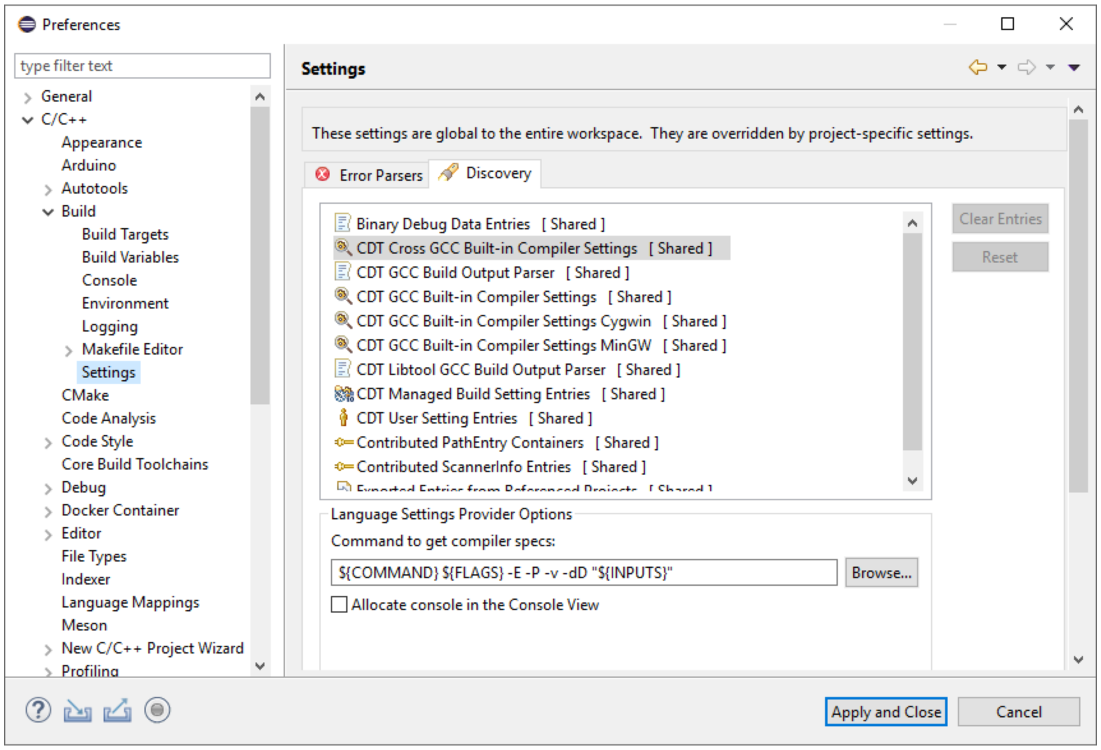
User Defined Toolchains

Type	Name	OS	Arch

Figure 58 – CDT Core Build Toolchains

In this example the toolchain for ARM installed earlier was detected with the path to the main GCC executable being C:/eclipse_gcc/gcc/bin/arm-none-eabi-gcc.exe correctly displayed.

Next it is helpful to change the Cross GCC compiler settings discovery for Eclipse CDT. This while help with some indexing issue later. To do so navigate to the ”C/C++->Build->Settings” section of the Preferences panel and open the ”Discovery” tab.



These settings are global to the entire workspace. They are overridden by project-specific settings.

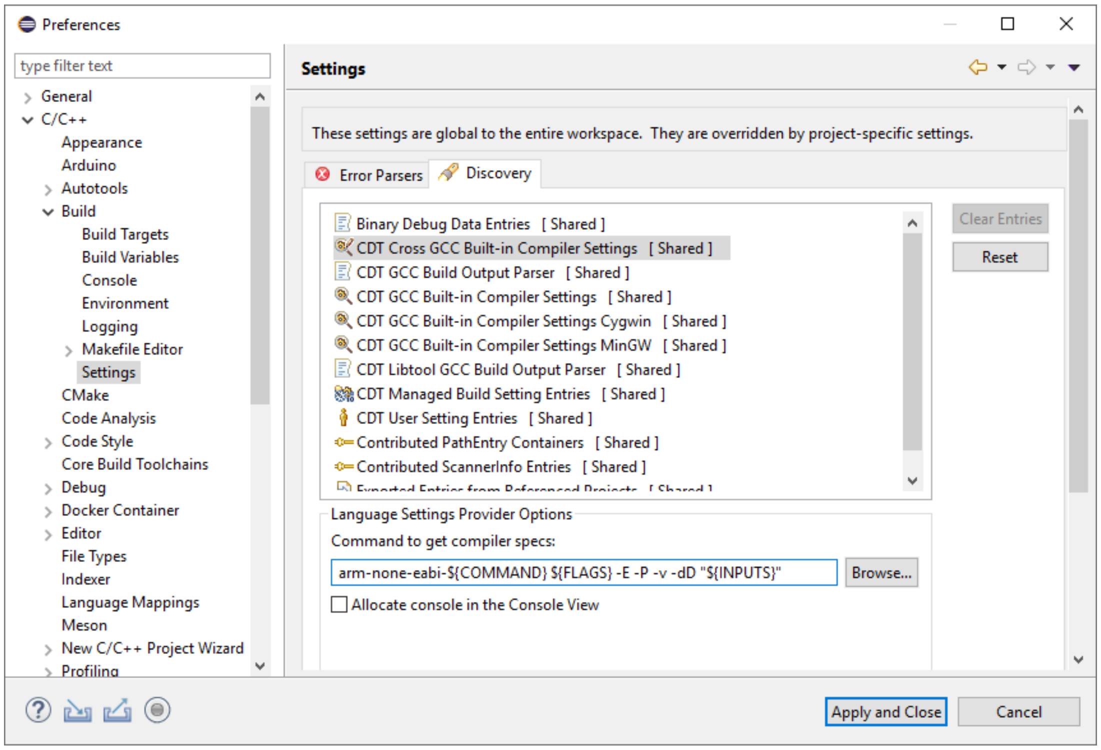
Binary Debug Data Entries [Shared]
CDT Cross GCC Built-in Compiler Settings [Shared]
CDT GCC Build Output Parser [Shared]
CDT GCC Built-in Compiler Settings [Shared]
CDT GCC Built-in Compiler Settings Cygwin [Shared]
CDT GCC Built-in Compiler Settings MinGW [Shared]
CDT Libtool GCC Build Output Parser [Shared]
CDT Managed Build Setting Entries [Shared]
CDT User Setting Entries [Shared]
Contributed PathEntry Containers [Shared]
Contributed ScannerInfo Entries [Shared]

Language Settings Provider Options
Command to get compiler specs:
```
${COMMAND} ${FLAGS} -E -P -v -dD "${INPUTS}"
```
☐ Allocate console in the Console View

Figure 59 – CDT Discovery Settings panel

Click the ”CDT Cross GCC Built-in Compiler Settings” item from the list. The ”Command to get compiler specs:” box should be modified to add the toolchain prefix in front of the command. In our example ”arm-none-eabi-”.

The result should look as follows:



These settings are global to the entire workspace. They are overridden by project-specific settings.

Binary Debug Data Entries [Shared]
CDT Cross GCC Built-in Compiler Settings [Shared]
CDT GCC Build Output Parser [Shared]
CDT GCC Built-in Compiler Settings [Shared]
CDT GCC Built-in Compiler Settings Cygwin [Shared]
CDT GCC Built-in Compiler Settings MinGW [Shared]
CDT Libtool GCC Build Output Parser [Shared]
CDT Managed Build Setting Entries [Shared]
CDT User Setting Entries [Shared]
Contributed PathEntry Containers [Shared]
Contributed ScannerInfo Entries [Shared]
Exported Entries from Referenced Projects [Shared]

Language Settings Provider Options
Command to get compiler specs:
```
arm-none-eabi-${COMMAND} ${FLAGS} -E -P -v -dD "${INPUTS}"
```
☐ Allocate console in the Console View

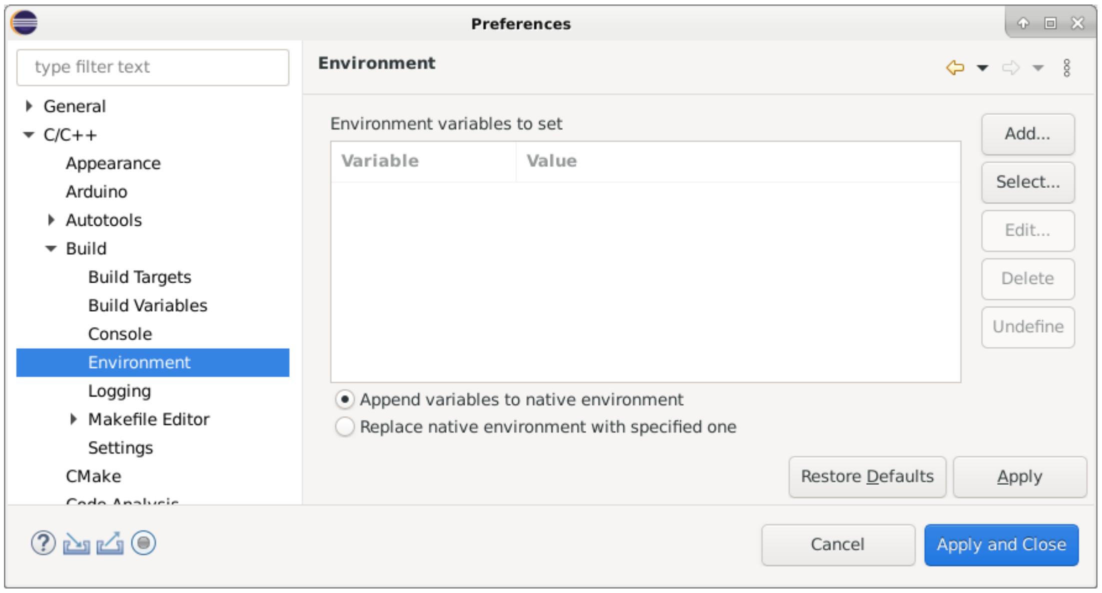
Figure 60 – Modified CDT Cross GCC Built-in Compiler Settings

That is all for the workspace configuration, click ”Apply and Close” to apply and save the configuration.

5.2 Alternative Workspace Specific PATH Variable

If the toolchain is not accessible through the default environment PATH variable, it is possible to either add it to the workspace specific environment variables or to each project's environment. This section will go over the workspace environment while the project specific method is discussed in the next section.

First navigate to the workspace preferences by selecting the Window->Preferences menu item. This will open the preferences panel. From that panel select the C/C++ -> Build -> Environment from the left panel. This will display the workspace environment variable settings.



Environment variables to set

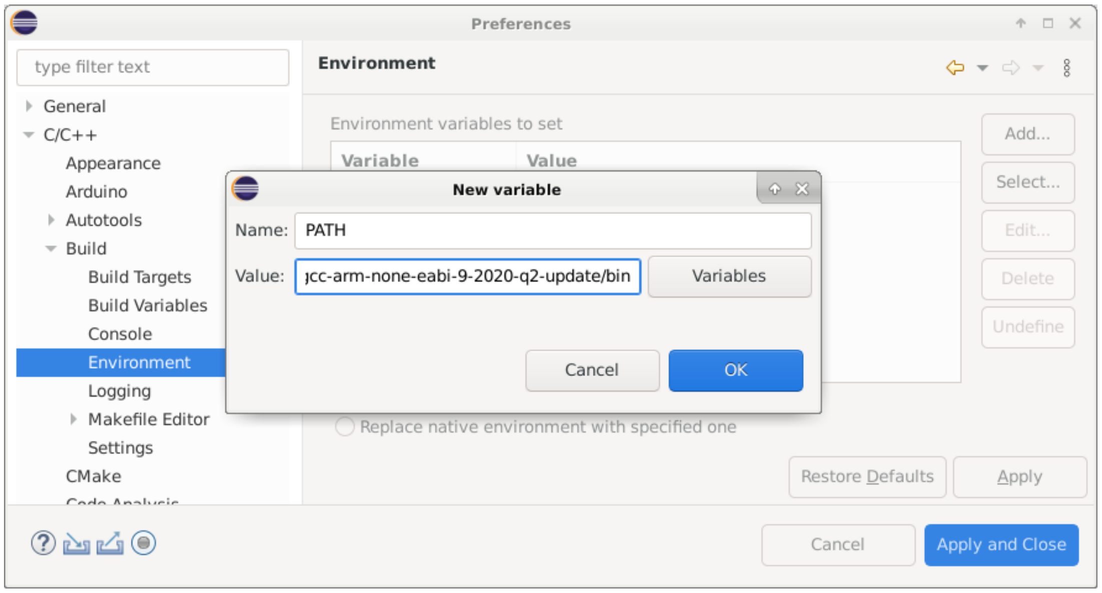
Variable	Value

(● Append variables to native environment) Replace native environment with specified one

Figure 61 – Workspace environment variable configuration panel

Make sure that the "Append variables to the native environment" option is checked.

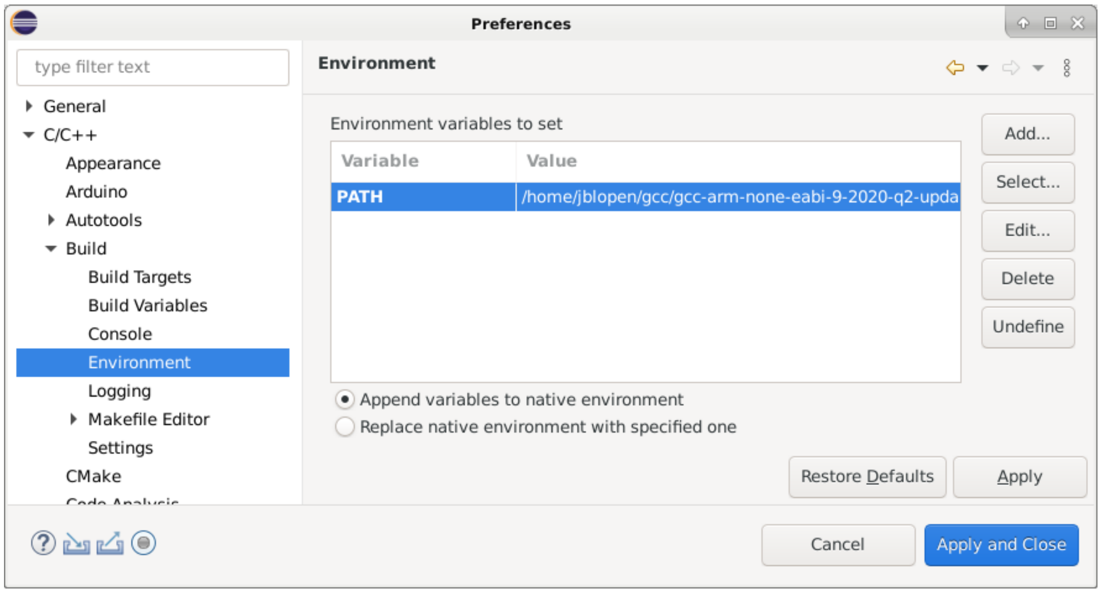
Click Add. In the new dialog enter `PATH` as the name and the full path to the desired toolchain `bin` directory. Note that since Eclipse will append the variables added in this configuration panel to the native environment it's important not to add the usual `$PATH` reference.



Name: PATH
Value: jcc-arm-none-eabi-9-2020-q2-update/bin

Figure 62 – Workspace environment variable configuration Add/Edit Variable dialog

Click OK.



Environment

Environment variables to set

Variable	Value
PATH	/home/jblopen/gcc/gcc-arm-none-eabi-9-2020-q2-upda

(● Append variables to native environment) Replace native environment with specified one

Figure 63 – New variable added

Click "Apply and Close", Now every new project built within the current workspace will have the toolchain path within the `PATH` environment variable.

5.3 Makefile Project Setup

When creating a new Makefile project, it is advisable to create it in a directory that already contains at least an empty Makefile. This is to ensure that the correct project type and nature are set when creating the project. The makefile should be named either "makefile" or "Makefile" with a capital M.

For this example we'll use a directory named "ex_make_build" for the project with an empty Makefile named "makefile".

To create a new Makefile project select the File->New->Makefile Project With Existing Code menu entry

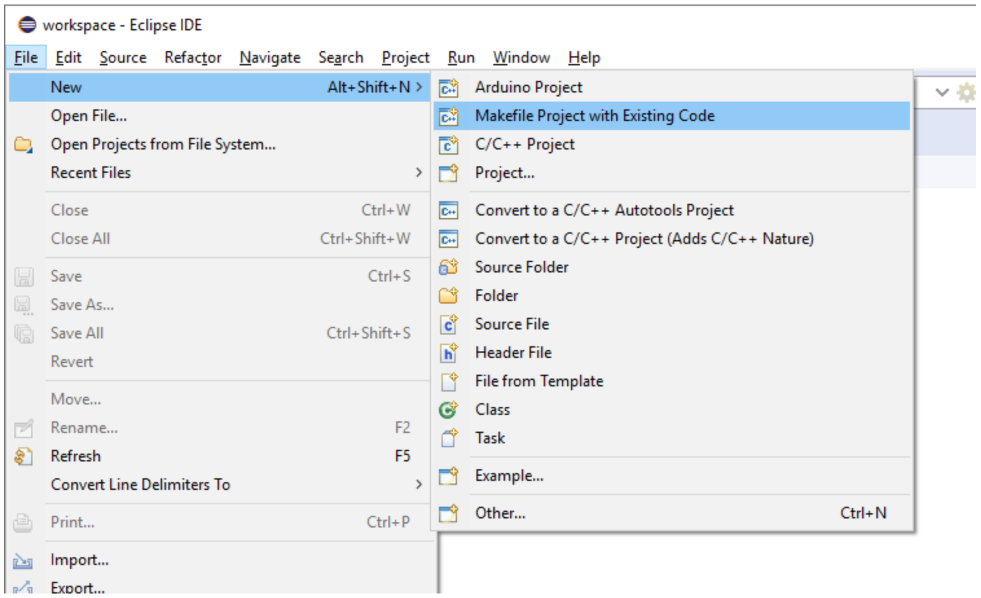
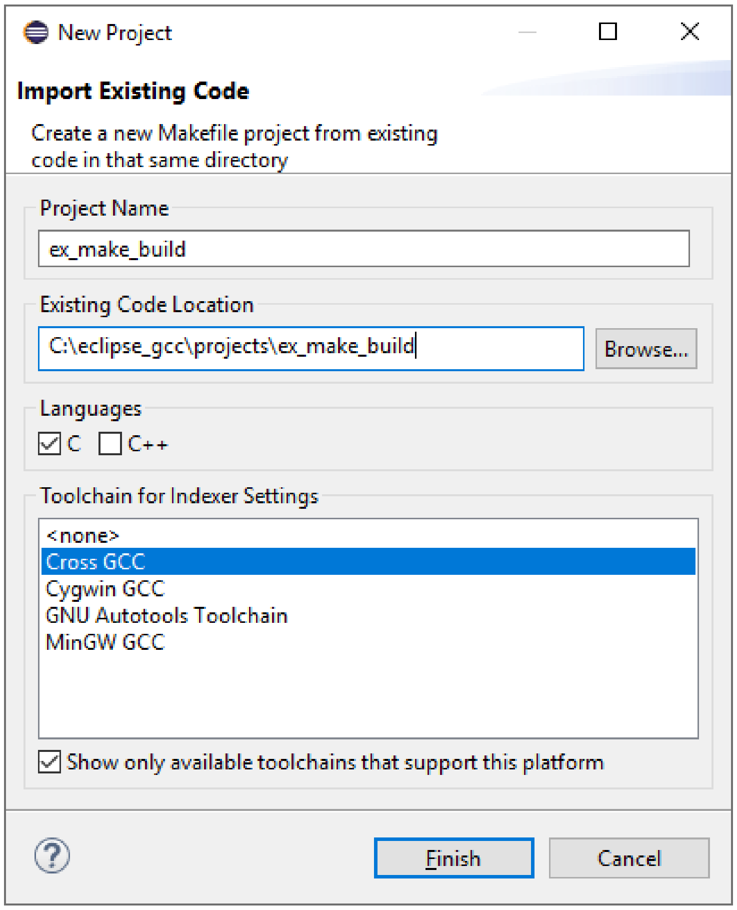


Figure 64 – New Makefile project menu item

This should open the New Project dialog. Browse or type the path to the existing makefile created earlier or you source code to build. By default the project name is the same as the directory name but can be changed if desired. Finally, select ”Cross GCC” as the toolchain.

The C and C++ checkboxes can be used to enable disable indexing of C++ or C code. If either of those will be found in your project, the corresponding box should be checked.



Import Existing Code

Create a new Makefile project from existing code in that same directory

Project Name: ex_make_build

Existing Code Location: C:\eclipse_gcc\projects\ex_make_build

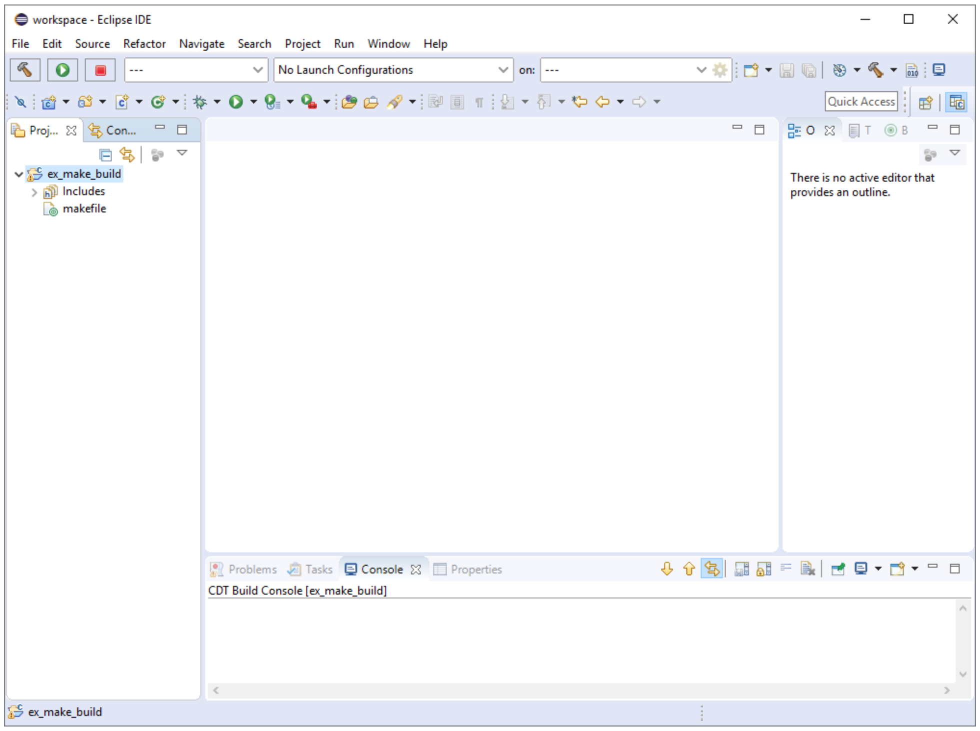
Languages: [✓ C] [ ] C++

Toolchain for Indexer Settings:
- <none>
- Cross GCC [✓]
- Cygwin GCC
- GNU Autotools Toolchain
- MinGW GCC

[✓] Show only available toolchains that support this platform

Figure 65 – New Makefile Project dialog

Click Finish. Eclipse will return to the main view with the newly created project in the workspace.



CDT Build Console [ex_make_build]

Figure 66 – Eclipse main view with new project

Let's add a file to build named main.c with a minimal main function. To do so right-click on the project then click the New->file menu item.

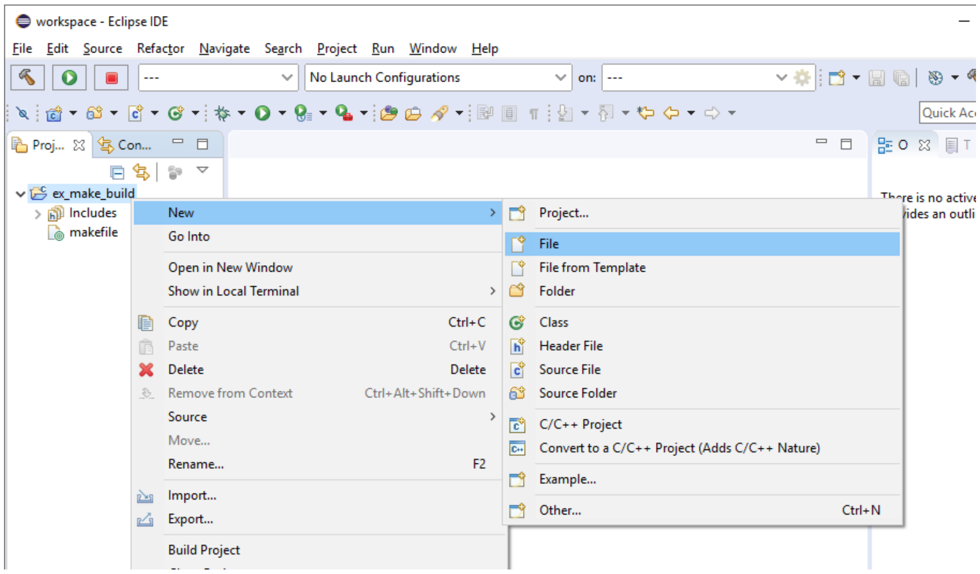
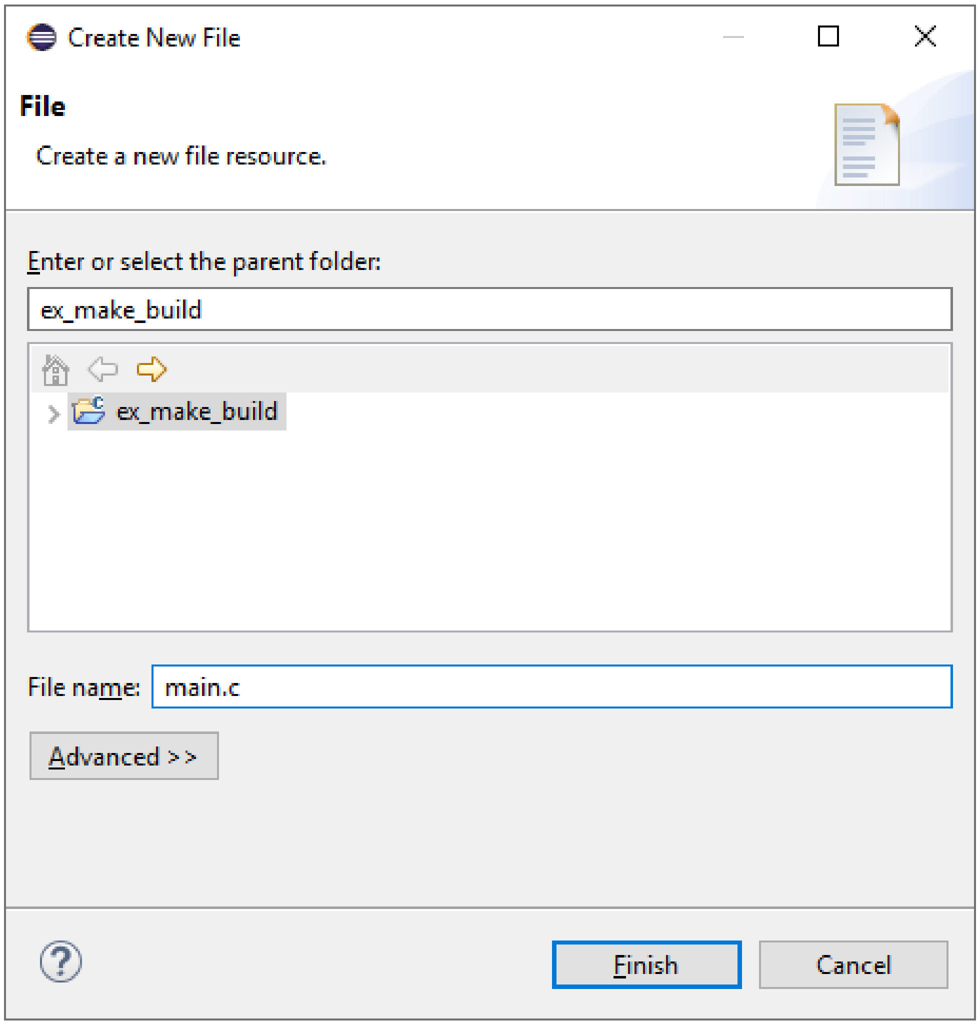


Figure 67 – Project new file menu item

In the "Create New File" dialog enter the name of the file, "main.c" in this example.


Create New File

File

Create a new file resource.

Enter or select the parent folder:

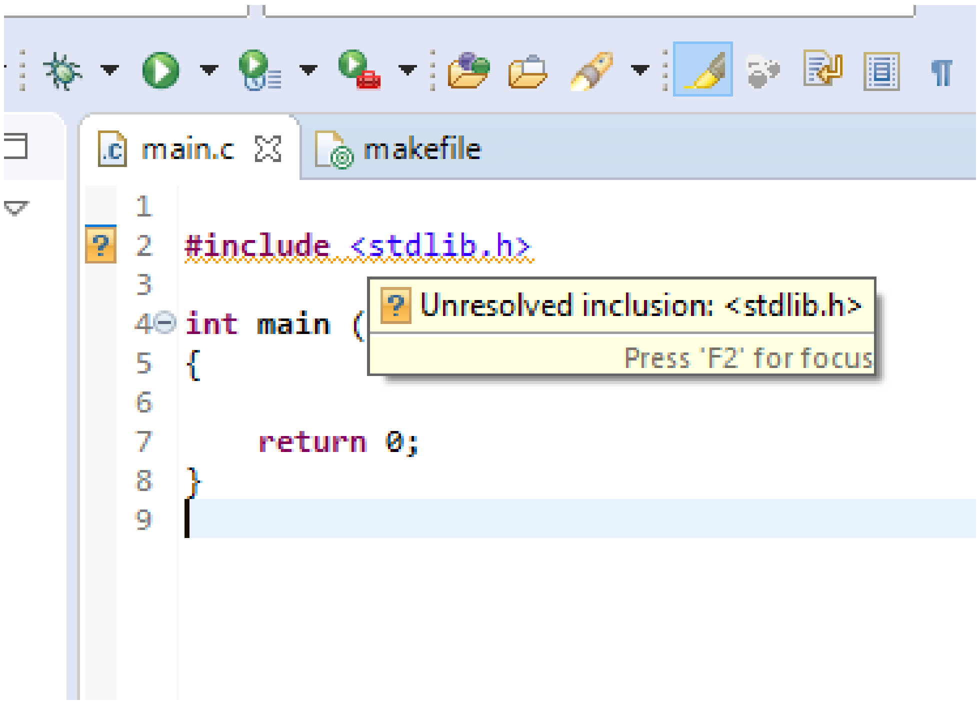
ex_make_build

File name: main.c

Figure 68 – New File dialog

At this moment if a C or C++ include directive of a standard header is written in the C file, it will most probably be underlined stating that there is an unresolved inclusion. This is due to the CDT indexer being unable to invoke the cross compiler.

Note that if you have a native GCC compiler available in your environment PATH it might be used by default for indexing. In which case there won't be unresolved inclusions reported by the indexer. However, since these headers do not match the current toolchain it can cause a few issues with code completion and indexing.



```
#include <stdlib.h>

int main (
{

    return 0;
}
```

Figure 69 – Example of an unresolved inclusion

To fix this issue open the Project Properties, either by right-clicking on the project or selecting the project and opening the Project->Properties menu item.

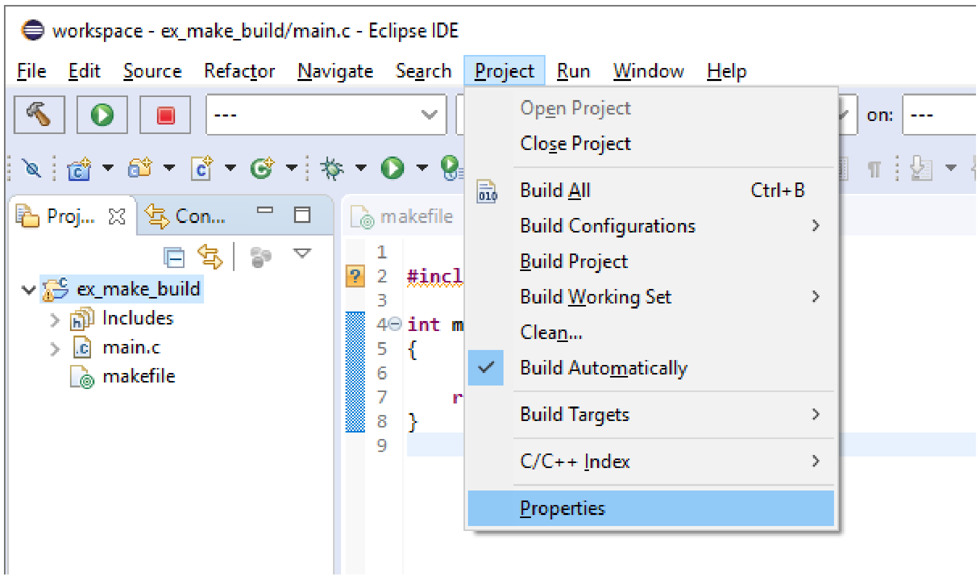
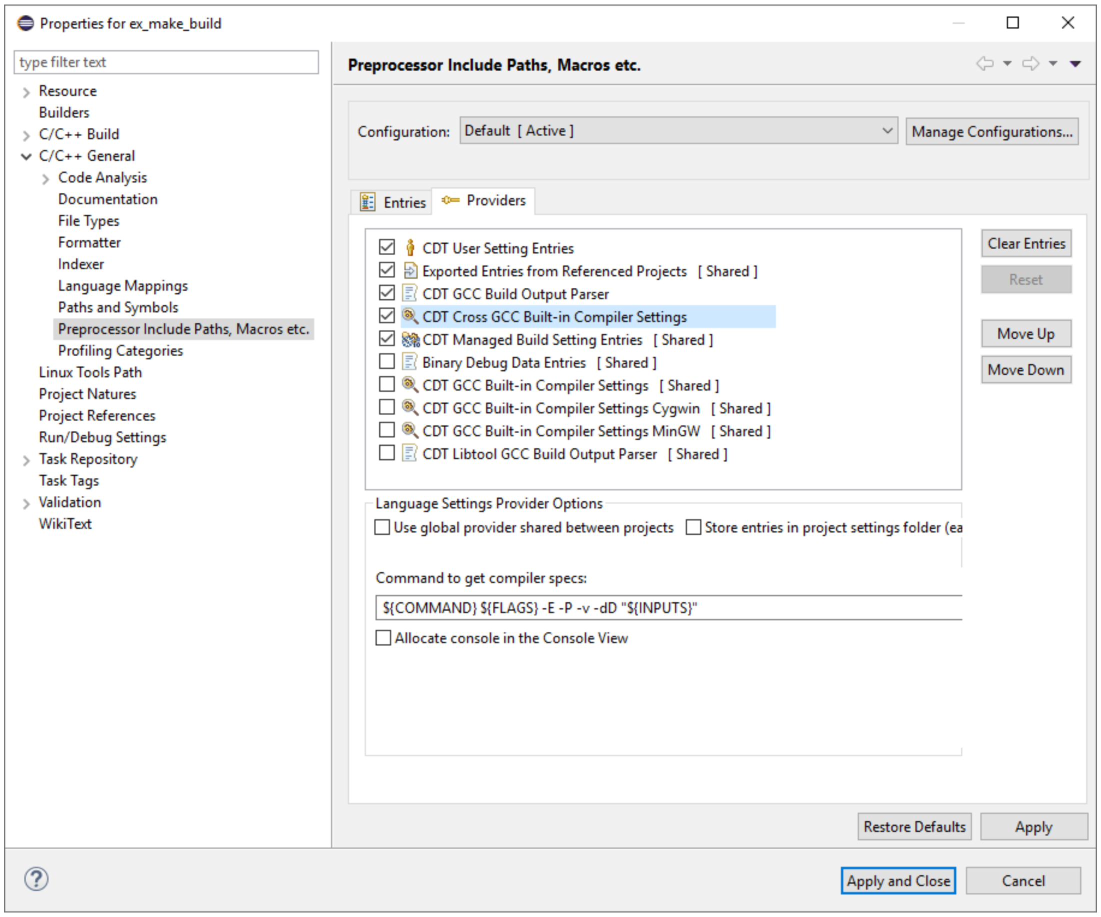


Figure 70 – Project Properties menu item

In the Properties panel navigate to the "Preprocessor Include Paths, Macros etc." section of the "C/C++ General" category.



Preprocessor Include Paths, Macros etc.

Configuration: Default [Active]

- [✓] CDT User Setting Entries
- [✓] Exported Entries from Referenced Projects [Shared]
- [✓] CDT GCC Build Output Parser
- [✓] CDT Cross GCC Built-in Compiler Settings
- [✓] CDT Managed Build Setting Entries [Shared]
- [ ] Binary Debug Data Entries [Shared]
- [ ] CDT GCC Built-in Compiler Settings [Shared]
- [ ] CDT GCC Built-in Compiler Settings Cygwin [Shared]
- [ ] CDT GCC Built-in Compiler Settings MinGW [Shared]
- [ ] CDT Libtool GCC Build Output Parser [Shared]

Language Settings Provider Options

[ ] Use global provider shared between projects [ ] Store entries in project settings folder (ea

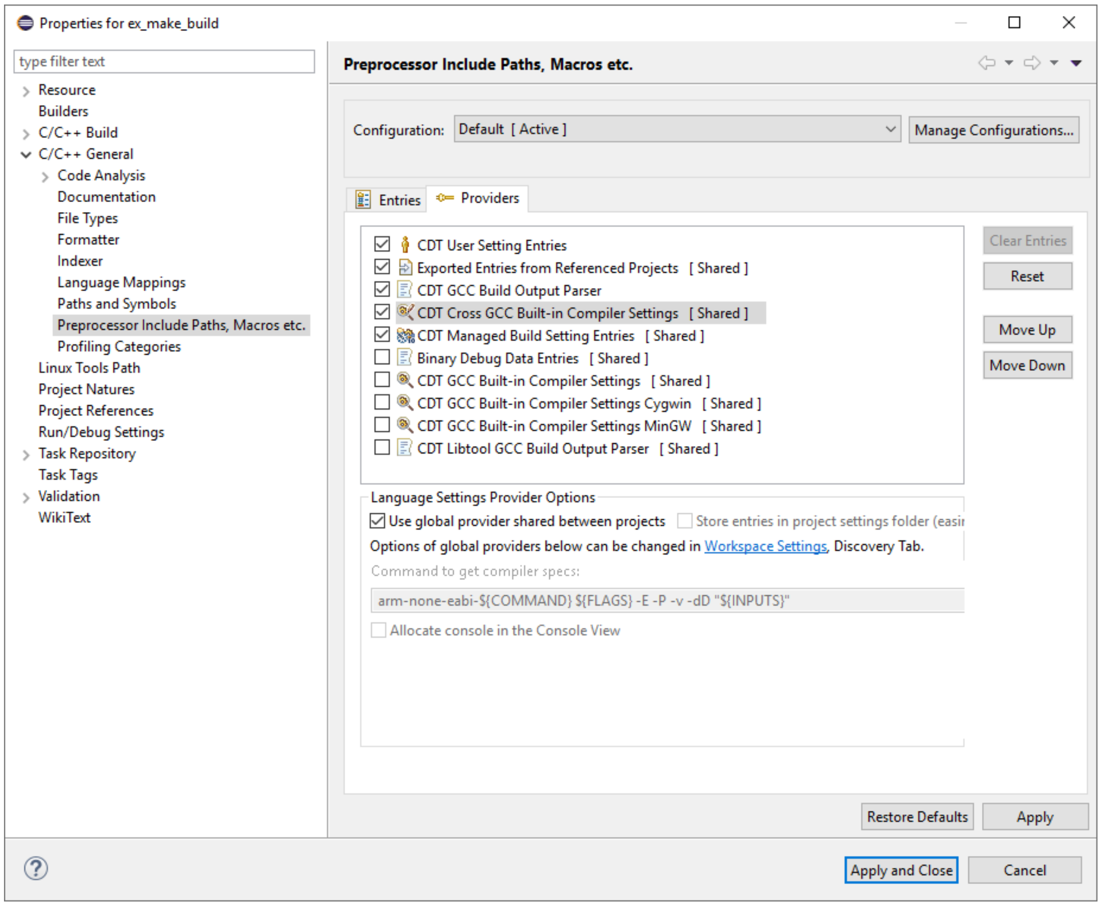
Command to get compiler specs:

```
${COMMAND} ${FLAGS} -E -P -v -dD "${INPUTS}"
```

[ ] Allocate console in the Console View

Figure 71 – C/C++ Providers Settings

Assuming that the workspace setup was done, on this screen select the ”CDT Cross GCC Built-in Compiler Settings” item and check the ”Use global provider shared between projects” checkbox. This will change the provider’s configuration to the one we set in the workspace.



Properties for ex_make_build

Preprocessor Include Paths, Macros etc.

Configuration: Default [Active]

Providers:

- [✓] CDT User Setting Entries
- [✓] Exported Entries from Referenced Projects [Shared]
- [✓] CDT GCC Build Output Parser
- [✓] CDT Cross GCC Built-in Compiler Settings [Shared]
- [✓] CDT Managed Build Setting Entries [Shared]
- [ ] Binary Debug Data Entries [Shared]
- [ ] CDT GCC Built-in Compiler Settings [Shared]
- [ ] CDT GCC Built-in Compiler Settings Cygwin [Shared]
- [ ] CDT GCC Built-in Compiler Settings MinGW [Shared]
- [ ] CDT Libtool GCC Build Output Parser [Shared]

Language Settings Provider Options

[✓] Use global provider shared between projects [ ] Store entries in project settings folder (easi

Options of global providers below can be changed in Workspace Settings, Discovery Tab.

Command to get compiler specs:

arm-none-eabi-${COMMAND} ${FLAGS} -E -P -v -dD "${INPUTS}"

[ ] Allocate console in the Console View

Figure 72 – Modified C/C++ Providers Settings

At this point the include files should no longer be unresolved.

Now that the inclusions are fixed, we need a Makefile. Here is a minimal example Makefile:

```
# Intermediary objects working directory.
OBJDIR := obj

# Name of the output binary executable file.
LINK_TARGET := a.out

# Application source files list.
APP_SOURCE := main.c

# Name of toolchain executables to use.
CC := arm-none-eabi-gcc
AS := arm-none-eabi-gcc
AR := arm-none-eabi-ar
LD := arm-none-eabi-gcc

# C Build flags.
```

```
C_FLAGS := -mcpu=cortex-m4 -g -ffunction-sections --specs=nosys.specs

# C Build warning flags.
WARNING_FLAGS := -Wall -pedantic -Wconversion

# Configure include directories.
INCLUDEDIR = $(subst \,/,$(PWD)) \
             $(BSP_SRC_DIR) \

# Extract all the C source file names.
SOURCE += $(notdir $(APP_SOURCE))

# Compute intermediary object names.
OBJ += $(SOURCE:%.c=$(OBJDIR)/%.o)

# Build everything.
all : $(OBJDIR) $(LINK_TARGET)
    @echo All done

# Firmware binary linking.
$(LINK_TARGET) : $(OBJ)
    $(LD) -o $@ $^ $(C_FLAGS)

# C files build.
$(OBJDIR)/%.o : %.c | $(OBJDIR)
    $(CC) $(C_FLAGS) $(WARNING_FLAGS) -o $@ -c $<

$(OBJDIR) :
    mkdir $(OBJDIR)

.PHONY: clean
clean:
    rm -rf ${OBJDIR}
    rm -rf $(LINK_TARGET)
```

The project can now be built using the ”Build Project” item from the context menu.

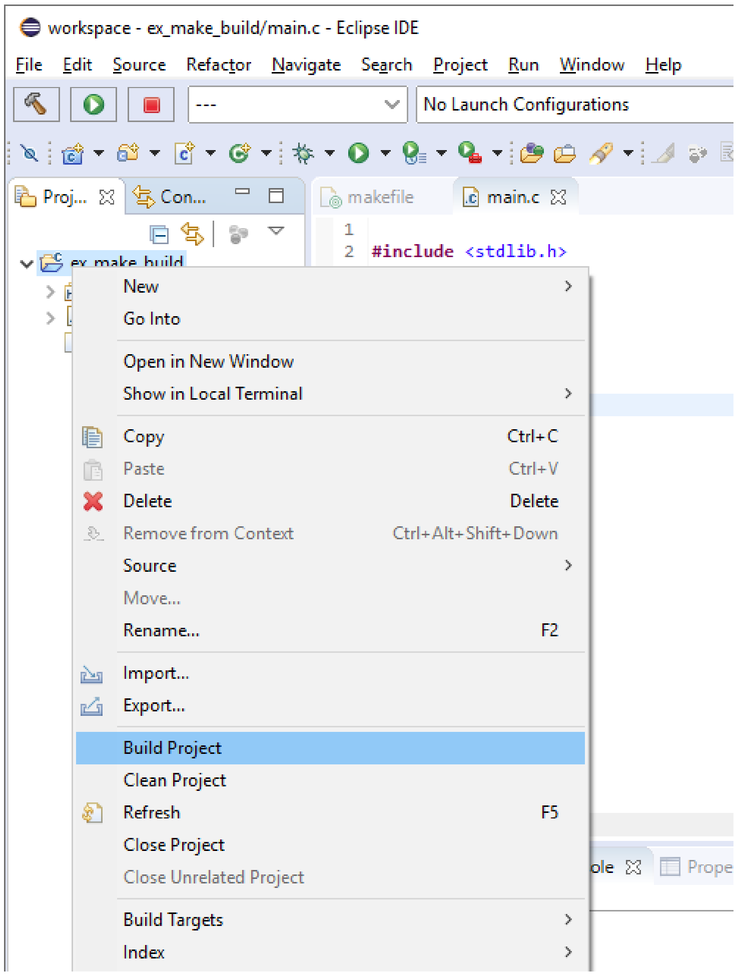
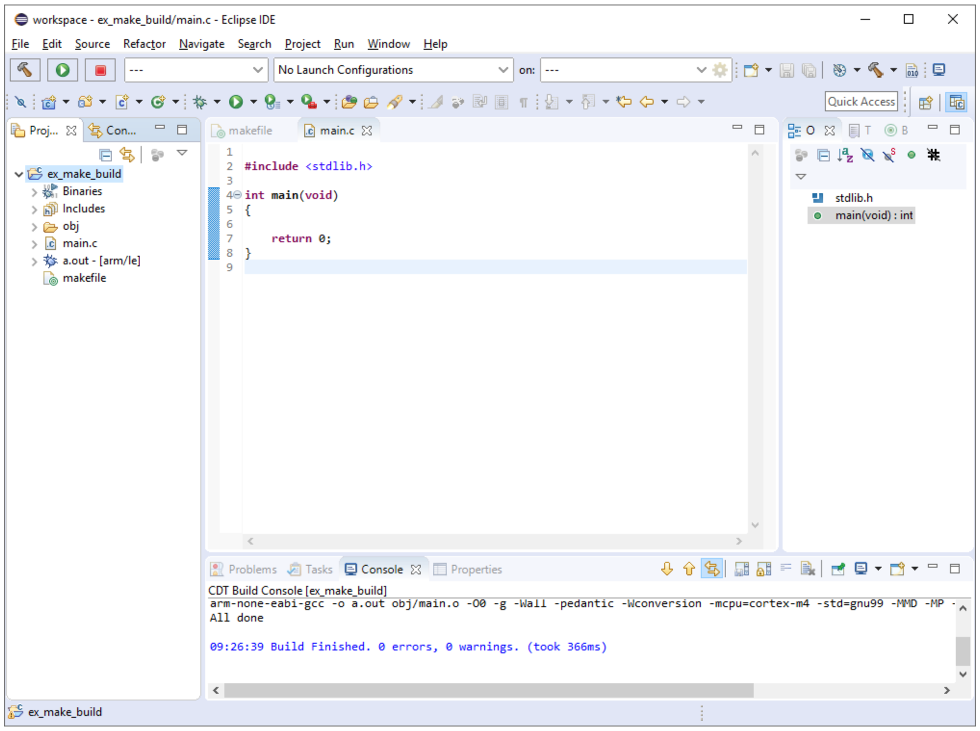


Figure 73 – Build Project menu item

The Makefile project is now ready for development.


workspace - ex_make_build/main.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
No Launch Configurations
Quick Access
ex_make_build
Binaries
Includes
obj
main.c
a.out - [arm/le]
makefile
stdlib.h
main(void) : int
Problems Tasks Console Properties

```

#include <stdlib.h>

int main(void)
{

    return 0;
}

```

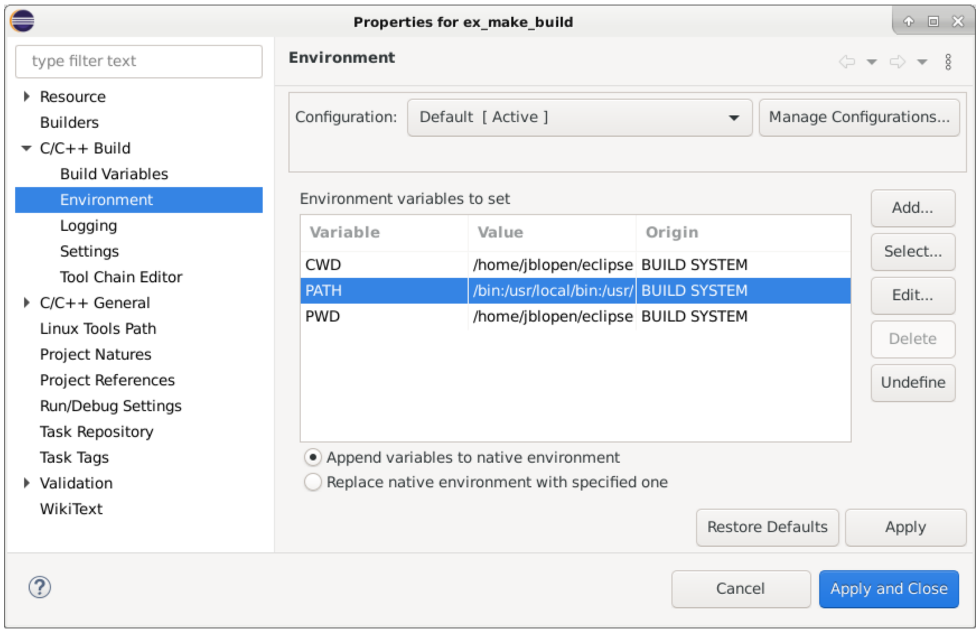
```
CDT Build Console [ex_make_build]
arm-none-eabi-gcc -o a.out obj/main.o -O0 -g -Wall -pedantic -Wconversion -mcpu=cortex-m4 -std=gnu99 -MMD -MP -
All done

09:26:39 Build Finished. 0 errors, 0 warnings. (took 366ms)
```

Figure 74 – Built project

5.4 Project Specific Toolchain Path

The previous instructions assumed that the toolchain was reachable through the build environment's PATH variable. If the toolchain path wasn't added to the eclipse environment or not added through the workspace environment settings then it must be added to the project environment itself.

This can be achieved by opening the project's properties and navigating to the C/C++ Build -> Environment panel.



Configuration: Default [Active]

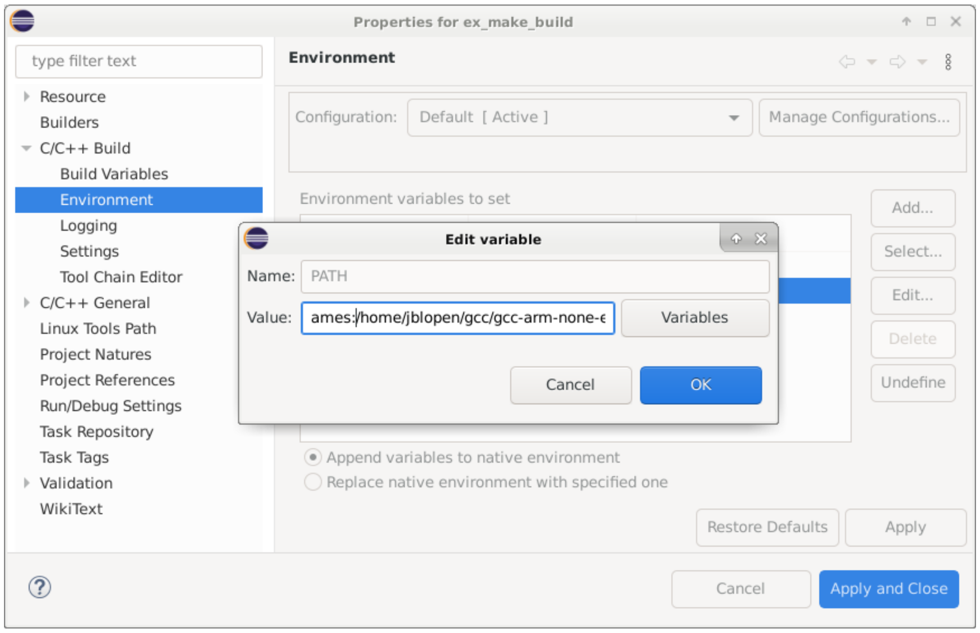
Environment variables to set

Variable	Value	Origin
CWD	/home/jblopen/eclipse	BUILD SYSTEM
PATH	/bin:/usr/local/bin:/usr/	BUILD SYSTEM
PWD	/home/jblopen/eclipse	BUILD SYSTEM

[✓ Append variables to native environment]
Replace native environment with specified one

Figure 75 – Project build environment configuration panel

An existing PATH variable is most probably already defined from the build environment. Click edit to modify it.

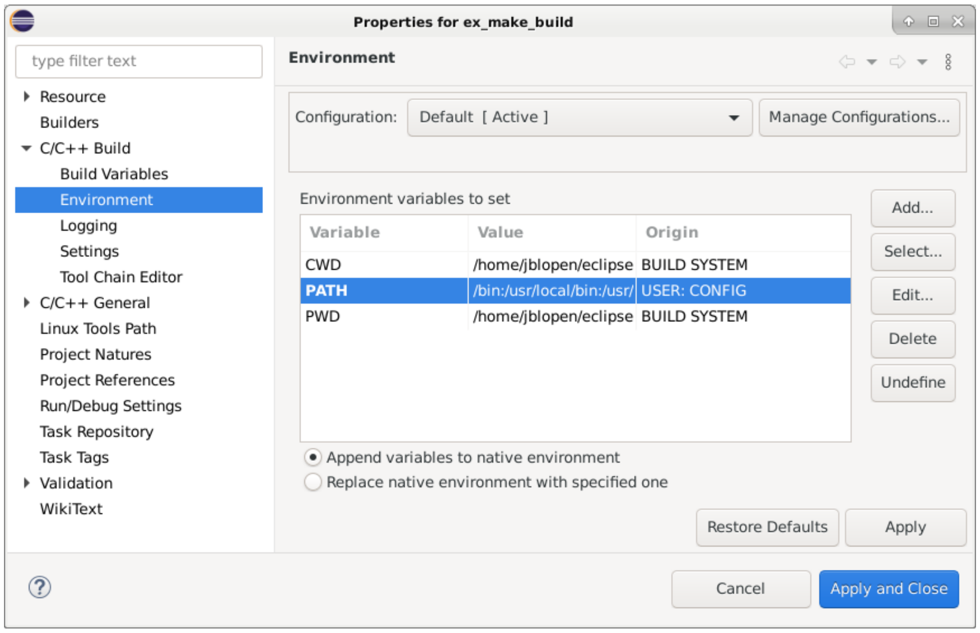


Name: PATH
Value: ames:/home/jblopen/gcc/gcc-arm-none-e

Figure 76 – Edit build environment variable dialog

In the edit dialog append the path to the toolchain bin directory at the end of the existing paths by adding a colon followed by the path.

Click OK.



Configuration: Default [Active] — Manage Configurations...

Environment variables to set

Variable	Value	Origin
CWD	/home/jblopen/eclipse	BUILD SYSTEM
PATH	/bin:/usr/local/bin:/usr/	USER: CONFIG
PWD	/home/jblopen/eclipse	BUILD SYSTEM

(●) Append variables to native environment
( ) Replace native environment with specified one

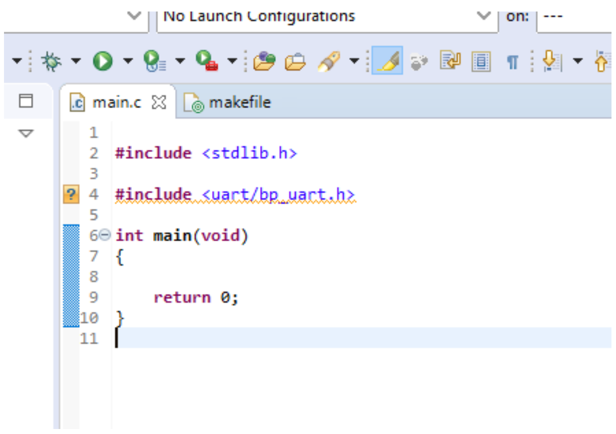
Figure 77 – PATH variable edited

Click Apply and Close.

At this point the toolchain should be usable from this specific project. These last steps must be repeated for each individual projects when using this method.

5.5 Indexing External Source Files

Oftentimes when using a Makefile project it is desirable to build source files that are outside the project directory. In that situation the indexer may not be able to find those files automatically which leads to unresolved inclusions warnings, code-completion issues and difficulty with debugging.

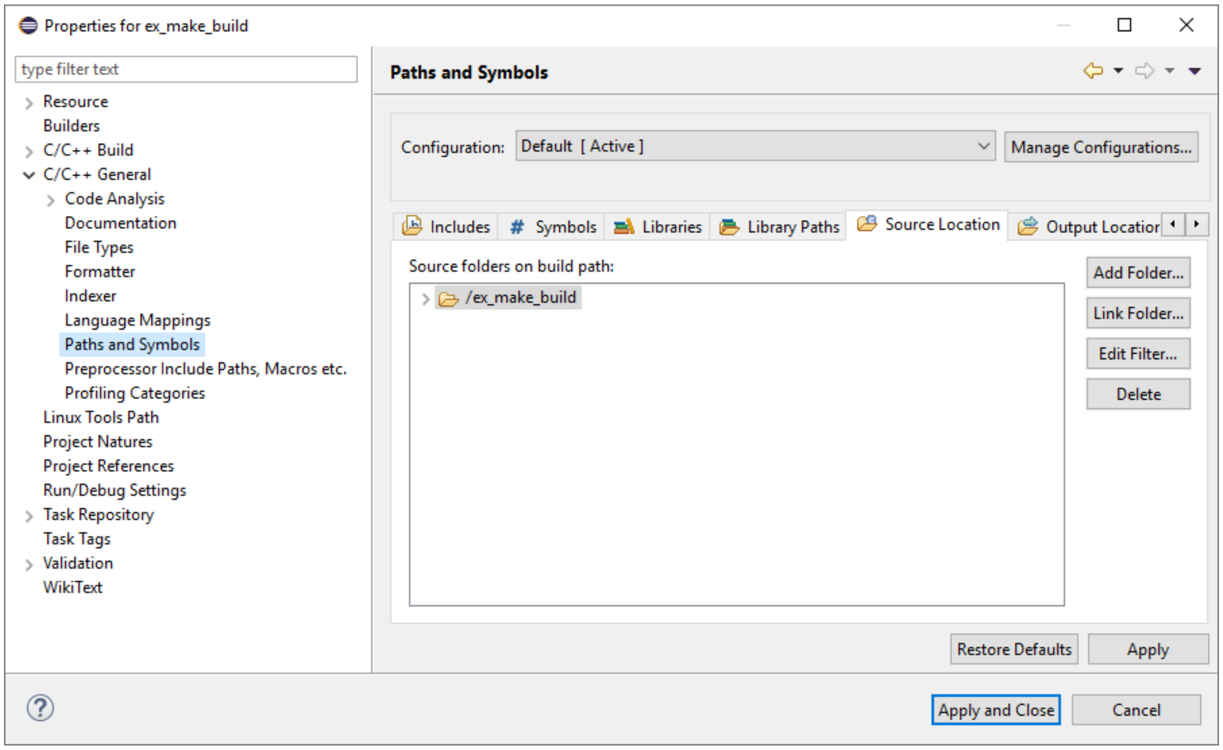


```

#include <stdlib.h>

#include <uart/bp_uart.h>

int main(void)
{

    return 0;
}

```

Figure 78 – Unresolved inclusion of an external file

For example, in the previous screenshot the external include file `uart/bp_uart.h` is underlined in yellow, meaning it hasn't been found by the indexer.

To solve this problem the path to the source files must be added in two locations. To do so open the project Properties and navigate to the "Paths and Symbols" On that screen open the "Source Location" tab. If it is not displayed use the arrows on the right which will show any hidden tabs on this panel.



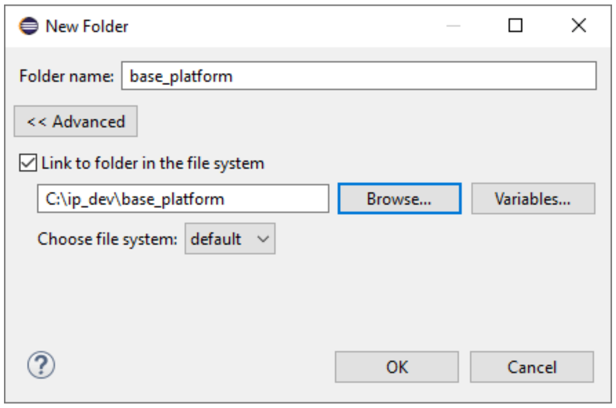
Configuration: Default [Active]

Source folders on build path:

/ex_make_build

Figure 79 – Project Source Location configuration

To add a new external source location click "Link Folder".



Folder name: base_platform

[✓] Link to folder in the file system

C:\ip_dev\base_platform

Choose file system: default

Figure 80 – Link directory dialog

Check the ”Link folder in the file system” box and then browse or type the location of the external source. In this example the external source code location is the ”C:/ip_dev/base_platform” directory. Click OK to add the directory.

Once the source location is linked, it is necessary to add the location to the indexer’s include path. To do so open the ”Includes” tab. And select either GNU C or GNU C++ depending on your project type.

Note that in the case both C and C++ was selected when initially creating the project, CDT will use the C++ compiler for indexing. In this case it’s important to add the include files paths to the GNU C++ language and note GNU C.

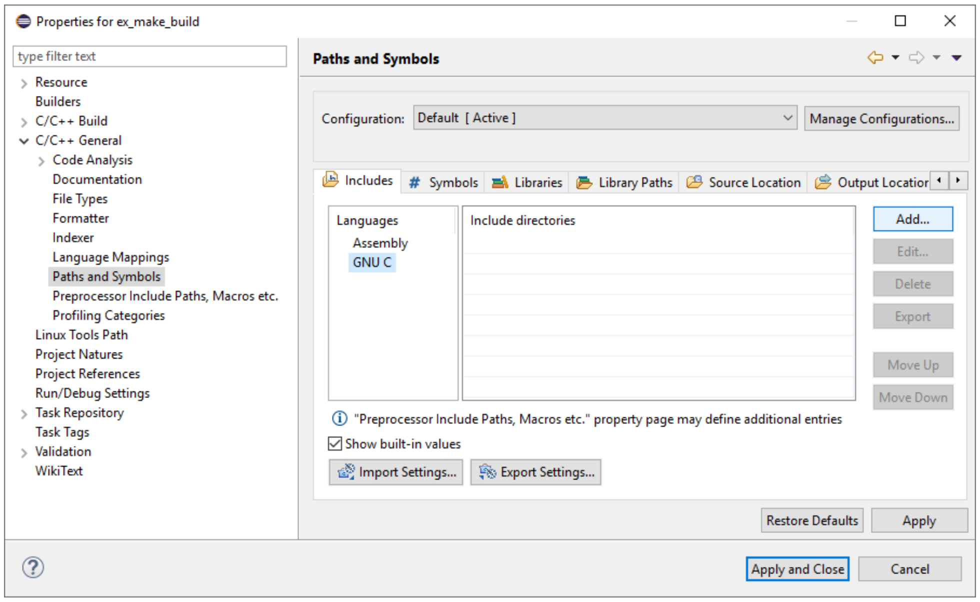


Figure 81 – Includes configuration panel

Click ”Add...” to add a new directory to search for include files. This will open the ”Add directory path” dialog. Since the source location was linked in the previous steps, it’s possible to use the workspace link in this dialog.

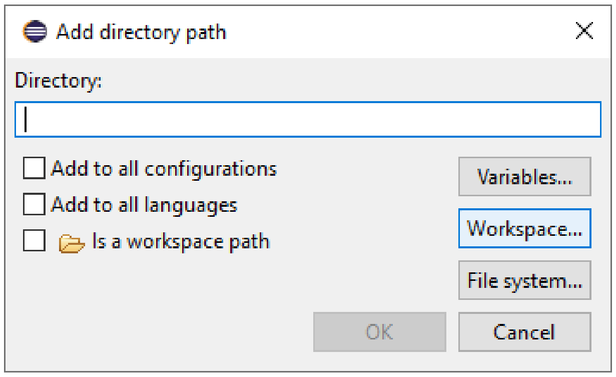


Figure 82 – Add directory path dialog

Click ”Workspace...” to add a workspace location.

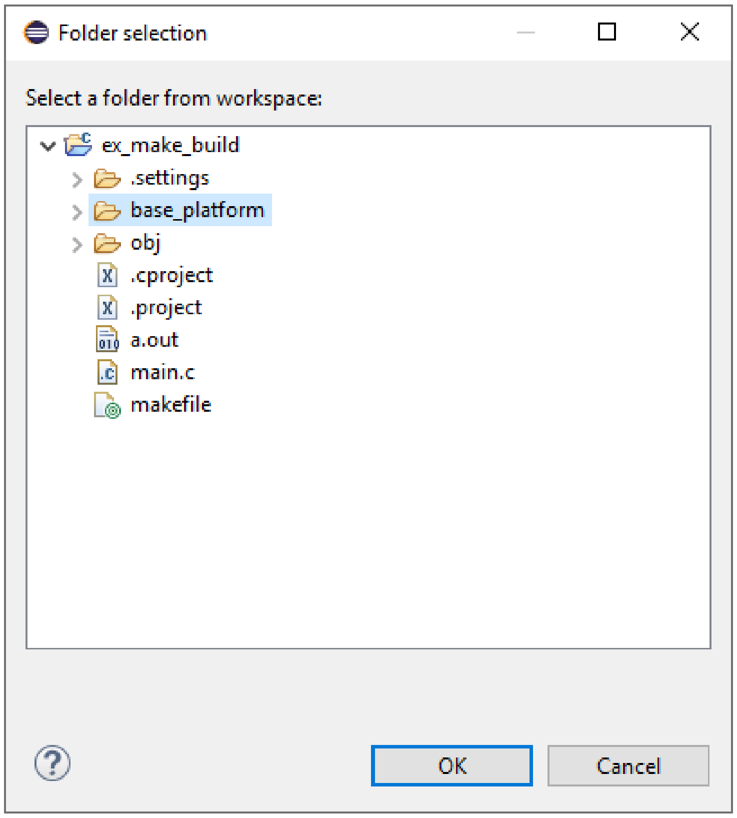


Figure 83 – Add workspace path dialog

Select the previously created workspace link, in this example ”base_platform”. Then click OK to add the path.

After doing the procedure for each external source location, they should be properly indexed and reachable during a debugging session.

Chapter
6

Debug Configuration

Many debug probes and targets supports acting as a GDB server. Which means that they can be debugged using GDB either remotely or from the same host. The exact steps required to start a GDB server are outside the scope of this guide but once the GDB server is running it's easy to connect through it from Eclipse. This guide assumes that the GDB server is started on the local host, in which case it's important to note the port number in use by the GDB server. The port number is usually prominent in the server configuration if a GUI tool was used to start the server. Otherwise the port number is usually specified on the command line when starting gdbserver from a shell.

First open the "Debug Configurations" menu by right-clicking on the project and selecting the "Debug As->Debug Configurations" menu entry. It's recommended to use this method to open the "Debug Configurations" dialog as it will make it easier to create a new debug configuration for the project.

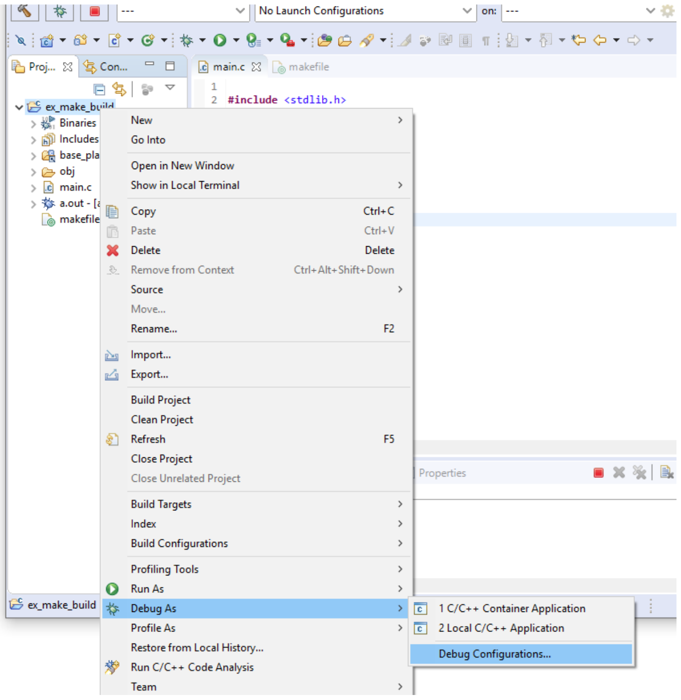
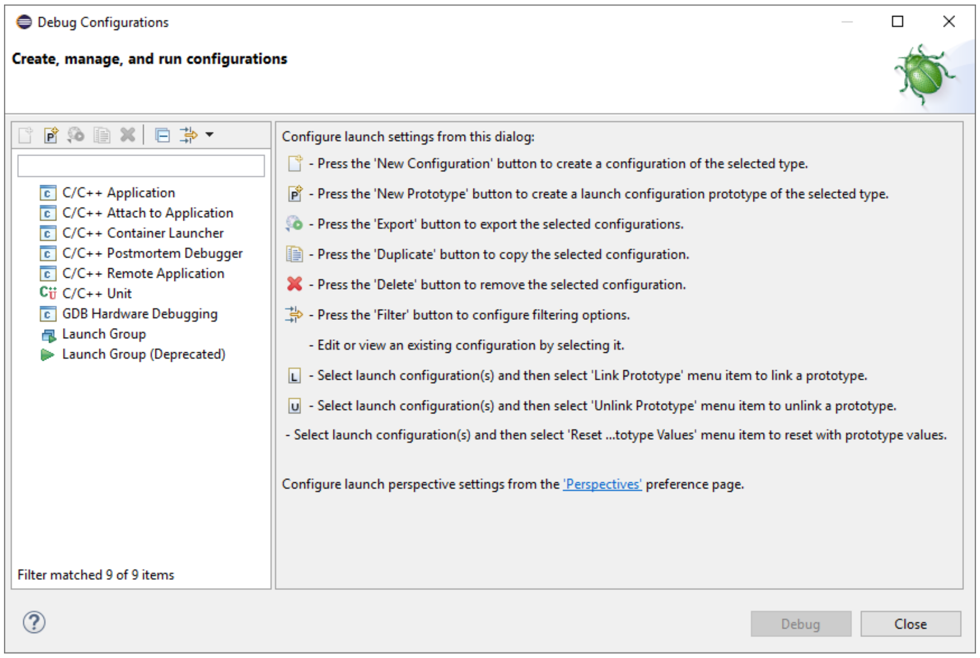


Figure 84 – Debug Configurations menu item

This will open the ”Debug Configurations” panel.


Debug Configurations
Create, manage, and run configurations
C/C++ Application
C/C++ Attach to Application
C/C++ Container Launcher
C/C++ Postmortem Debugger
C/C++ Remote Application
C/C++ Unit
GDB Hardware Debugging
Launch Group
Launch Group (Deprecated)
Filter matched 9 of 9 items
Debug
Close

Configure launch settings from this dialog:

- Press the 'New Configuration' button to create a configuration of the selected type.
- Press the 'New Prototype' button to create a launch configuration prototype of the selected type.
- Press the 'Export' button to export the selected configurations.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.
- Select launch configuration(s) and then select 'Link Prototype' menu item to link a prototype.
- Select launch configuration(s) and then select 'Unlink Prototype' menu item to unlink a prototype.
- Select launch configuration(s) and then select 'Reset ...totype Values' menu item to reset with prototype values.

Configure launch perspective settings from the 'Perspectives' preference page.

Figure 85 – Debug Configurations panel

Right click on ”GDB Hardware Debugging” and select ”New Configuration”.

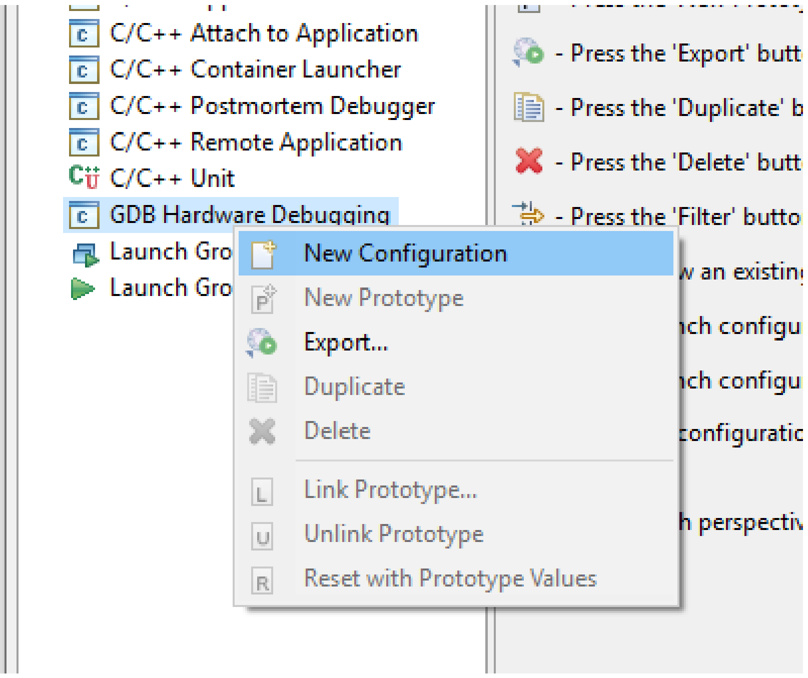
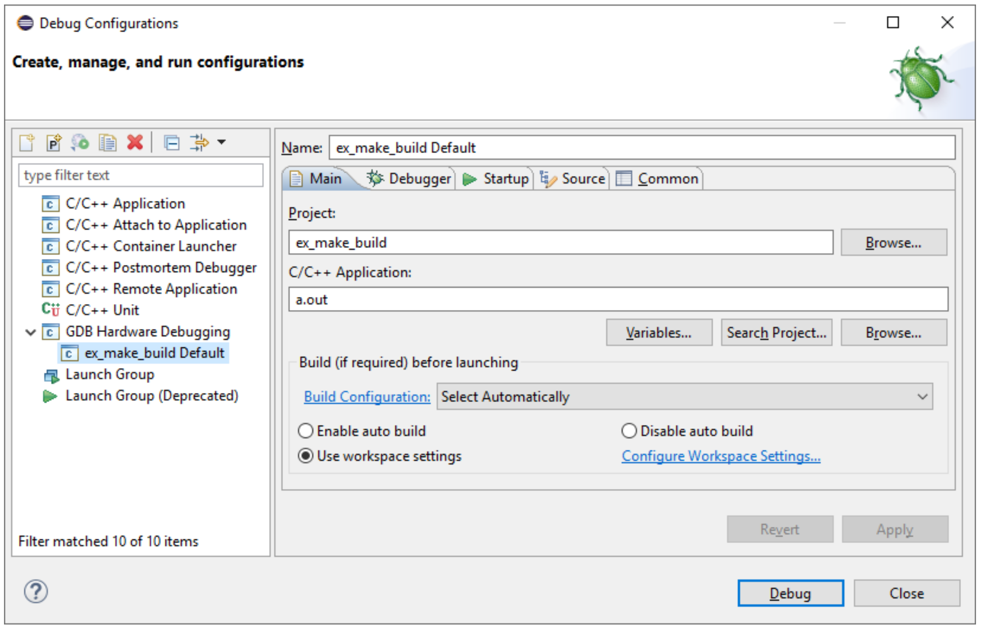


Figure 86 – Debug Configurations context menu

This will create and populate a new debug configuration for the selected project. Eclipse CDT will attempt to guess the name of the executable file. Check that the content of the "C/C++ Application" box match the target executable to debug.


Debug Configurations

Create, manage, and run configurations

type filter text

C/C++ Application
C/C++ Attach to Application
C/C++ Container Launcher
C/C++ Postmortem Debugger
C/C++ Remote Application
C/C++ Unit
GDB Hardware Debugging
ex_make_build Default
Launch Group
Launch Group (Deprecated)

Filter matched 10 of 10 items

Name: ex_make_build Default

Main | Debugger | Startup | Source | Common

Project:
ex_make_build

C/C++ Application:
a.out

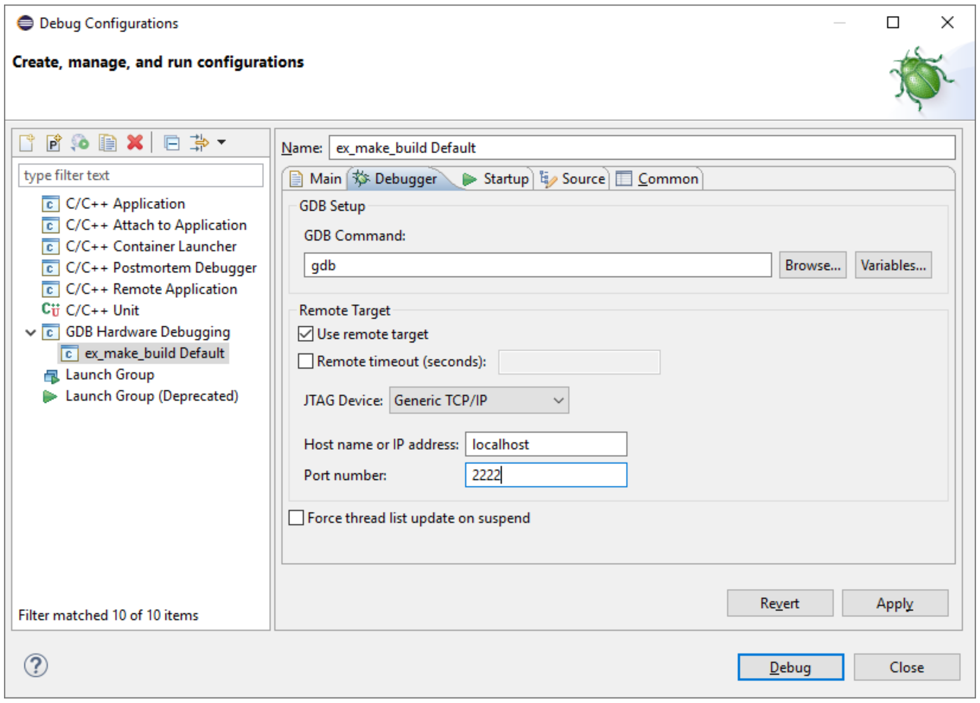
Build (if required) before launching

Build Configuration: Select Automatically

( ) Enable auto build
( ) Disable auto build
(●) Use workspace settings
Configure Workspace Settings...

Figure 87 – Newly created debug configuration

Open the ”Debugger” tab for further configuration.



Name: ex_make_build Default

GDB Setup
GDB Command: gdb

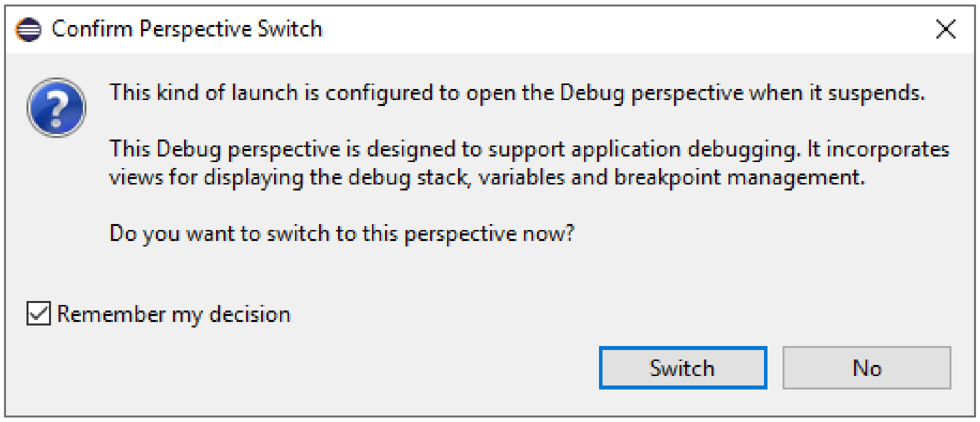
Remote Target
☑ Use remote target
☐ Remote timeout (seconds):
JTAG Device: Generic TCP/IP
Host name or IP address: localhost
Port number: 2222

☐ Force thread list update on suspend

Figure 88 – Debugger configuration

In this panel the host name and port number of the GDB server should be entered. If the server was started on the local host then locahost can be entered or ’127.0.0.1’. The port number should be the same as the one used when launching the GDB server. In most cases this is sufficient for a debug configuration to work.

Click ”Debug” to launch the configuration.



Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

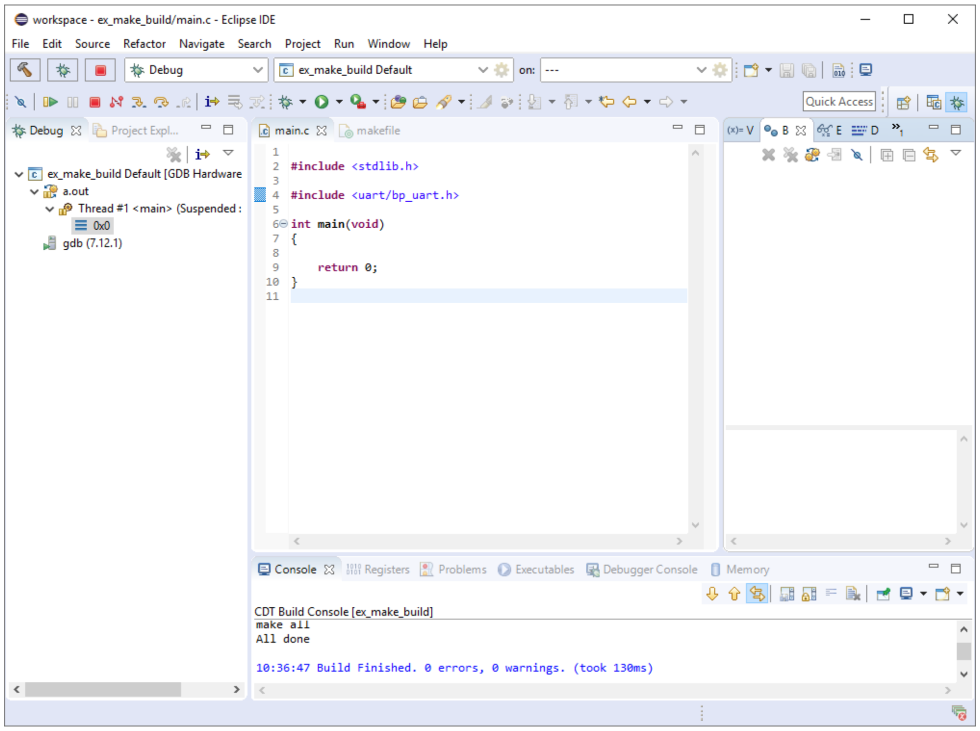
This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to switch to this perspective now?

☑ Remember my decision

Figure 89 – Confirm Perspective Switch dialog

The perspective switch confirmation dialog usually appears to warn the user that the view will be switched to the debugging view. Using the ”Remember my decision” checkbox will prevent this dialog from appearing further.


workspace - ex_make_build/main.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
ex_make_build Default
on: ---
Quick Access
Debug
Project Expl...
ex_make_build Default [GDB Hardware
a.out
Thread #1 <main> (Suspended :
0x0
gdb (7.12.1)
main.c
makefile
(x)= V
B
E
D
Console
Registers
Problems
Executables
Debugger Console
Memory

```
#include <stdlib.h>

#include <uart/bp_uart.h>

int main(void)
{

    return 0;
}
```

```
CDT Build Console [ex_make_build]
make all
All done

10:36:47 Build Finished. 0 errors, 0 warnings. (took 130ms)
```

Figure 90 – Debug session

And from this point everything should be ready to build and debug your embedded project using Eclipse.

Chapter

7

Document Revision History

Version	Release Date	Description
1	2019-07-03	- Initial release.
2	2019-08-20	- Added example Makefile to the Makefile project setup chapter.
3	2019-11-21	- Fixed formatting issue in the code example of the Makefile project setup chapter. - Added notes on the GDB server port number in the Makefile project setup chapter.
4	2020-09-14	- Added Linux instructions. - Clarified some sections. - Added additional methods of appending the toolchain to the PATH for makefile projects.
5	2020-09-23	- Fixed incorrect image in the Linux instructions.

Table 1 – Document Revision History