



eGui Eclipse™ User Guide

Imperas Software Limited

Imperas Buildings, North Weston,
Thame, Oxfordshire, OX9 2HA, UK
docs@imperas.com



Author:	Imperas Software Limited
Version:	2.0
Filename:	eGui_Eclipse_User_Guide.doc
Project:	Imperas Eclipse Project eGui
Last Saved:	Tuesday, 25 August 2015
Keywords:	

Copyright Notice

Copyright © 2015 Imperas Software Limited All rights reserved. This software and documentation contain information that is the property of Imperas Software Limited. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Imperas Software Limited, or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Imperas permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

IMPERAS SOFTWARE LIMITED, AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents

1	Preface.....	5
1.1	Notation.....	5
1.2	Related OVP Documents	5
2	Introduction.....	6
2.1	OVP Virtual Platforms.....	6
2.2	Prerequisites	6
3	Installing Imperas eGui.....	8
3.1	Installing the eGui_Eclipse package.....	8
3.2	eGui_Eclipse Package Contents	8
3.3	eGui plugin installation into existing Eclipse	8
4	Starting a debug session.....	11
4.1	Specifying platform options.....	11
4.2	Starting a standalone debug session.....	11
4.2.1	The eGui port file.....	12
4.3	Starting a debug session from Eclipse	13
4.3.1	Starting the Imperas eGui Eclipse product	13
4.3.1.1	The Java Runtime Environment (JRE)	13
4.3.1.2	Selecting a workspace.....	14
4.3.2	Eclipse launch configurations	14
4.3.2.1	The <i>Imperas - Connect</i> debug configuration dialog	16
5	Standalone eGui Debug Demos	19
5.1	eGui in GDB mode on Cortex-A15UP single core demo.....	19
5.1.1	Example Debug Session for Dhrystone Benchmark application.....	23
5.1.2	Removing Breakpoints.....	26
6	An Eclipse Project Example	27
6.1	Starting Imperas eGui	27
6.2	Selecting the C/C++ Perspective	27
6.3	Preparing an Eclipse Project	28
6.3.1	Creating a New Project	28
6.3.2	Configuring Project Build Commands (Windows Only).....	29
6.3.2.1	Using MINGW ‘make’	29
6.3.2.2	Adding MinGW tools onto PATH.....	29
6.3.3	Deselect ‘Autobuild’ from Eclipse	31
6.4	Importing a Demo to the Project.....	32
6.5	Building the applications locally	36
6.6	Virtual Platform Configuration for Application Debug.....	36
6.6.1	Command Line arguments for debugging on a C Platform.....	36
6.6.1.1	Specifying the debugger connection details	36
6.6.1.2	Nominating the debugged processor.....	36
6.6.2	ICM C Platform API calls for enabling debug	37
6.6.2.1	Specifying the debugger connection details	38
6.6.2.2	Nominating the debugged processor.....	38
6.6.3	Debugging applications with the Imperas ISS.....	39
6.7	Creating Launch Configurations.....	39

6.7.1	Virtual Platform launch.....	39
6.7.2	Create an Imperas MPD Debug Configuration.....	41
6.7.3	Create an OVPsim CDT Debug Configuration	42
6.7.4	Starting to Debug using eGui.....	44
6.7.5	Switching Console Output	48
6.7.6	Setting breakpoints without source	50
6.7.7	Terminating the Debug Session.....	52
6.7.8	Failing to launch correctly	53
7	An Example Debug Session.....	54

1 Preface

This document describes how to debug an application running on the OVP or Imperas Professional simulator using the Imperas eGui (based on Eclipse™) Integrated Development Environment.

The Imperas eGui is based upon Eclipse Luna version 4.4.2 and provided as the installation package eGui_Eclipse. This provides a standalone version of Eclipse and also includes an Eclipse plugin archive that can be used to add the Imperas eGui feature to other compatible Eclipse installations.

This package must be installed and used in conjunction with a standard OVP or Imperas product package installation.

The example in this document demonstrates debugging of ARM and MIPS32 applications but the same approach is valid for applications running on processor models for any architecture supported by the OVP simulator.

1.1 Notation

Code Code and command extracts

1.2 Related OVP Documents

- Imperas Installation and Getting Started Guide
- OVPSim and CpuManager User Guide

2 Introduction

This document describes how to install and use the Imperas eGui (based on Eclipse) debugger for interactive debugging of OVP virtual platforms.

The Imperas eGui (based on Eclipse) debugger (hereafter referred to as eGui) is based on the popular Eclipse IDE and adds the eGui feature which enhances Eclipse to connect to an OVP platform for debugging. See the OVPSim and CpuManager User Guide for information on OVP platforms.

The Imperas eGui consists of 2 parts:

1. A standalone Eclipse-based product with the eGui feature included.
2. A plugin that includes the eGui feature that may be added to an existing Eclipse version 4.4.2 (Luna) installation.

Non-Eclipse users may use the eGui product as a standalone debugger, while continuing to do their software development in their environment of choice.

Current users of Eclipse may add the eGui plugin to their Eclipse installation (version 4.4.2) to add OVP platform debugging capabilities to their existing Eclipse IDE.

2.1 OVP Virtual Platforms

The eGui feature supports connecting to OVP virtual platforms to debug application software and/or peripheral model behavioral code. The platform and models require either the OVPSim or Imperas professional simulator which provide different debugging capabilities as discussed below.

The OVPSim simulator supports:

- Debug of a single processor in a platform using the Gnu debugger (GDB).

Additionally, the Imperas simulator supports:

- Simultaneous debug of all processors and peripheral models in a platform using the Imperas Multi Processor Debugger (MPD).
- Programmers View of registers and other model elements provided by the processor and peripheral models.
- VAP tools which provide additional powerful debugging capabilities.

2.2 Prerequisites

The following OVPSim and/or Imperas packages must be installed to use eGui:

- One of: OVPSim, Imperas_DEV or Imperas_SDK products

- The appropriate example pre-built Cross Compiler toolchain for your target processor. These include the appropriate version of the target processor gdb for use by eGui.
- The peripheral simulation toolchain package, OVPpse.toolchain, if the debug of peripheral model behavioral code is to be performed.

The *Imperas Installation and Getting Started Guide* provides a step-by-step guide to obtaining and installing the OVP files, native compiler and cross compiler tools. It is strongly recommended that you follow that guide until you are able to build, compile and simulate a platform using OVPsim before attempting to debug using eGui.

In addition the eGui_Eclipse package must be installed as described in the next section.

3 Installing Imperas eGui

3.1 Installing the eGui_Eclipse package

eGui can be added to your Imperas installation by installing the eGui_Eclipse package. This package is provided in a self installing executable file available for download from the OVPworld.org or imperas.com websites (registration required).

The installer is a file named:

eGui_Eclipse.<version>.<arch>.exe

(Note, the file name includes the .exe suffix on Linux as well as Windows for consistency.)

For Linux it may be necessary to make the file executable with the command:

chmod +x <fn>

To install, simply execute the installer and follow the directions. See the Imperas Installation and Getting Started Guide for additional information.

3.2 eGui_Eclipse Package Contents

The eGui installation installs the following into the directory \$IMPERAS_HOME/lib/\$IMPERAS_ARCH/eGui:

1. The Imperas eGui (based on Eclipse) standalone product.
2. A Java Runtime Executable (JRE).
3. An archive file named com.imperas.egui.releng.p2-<version>.zip that contains the eGui plugin for installation into an existing Eclipse.

Additional utilities and documentation are installed in various places in the Imperas installation.

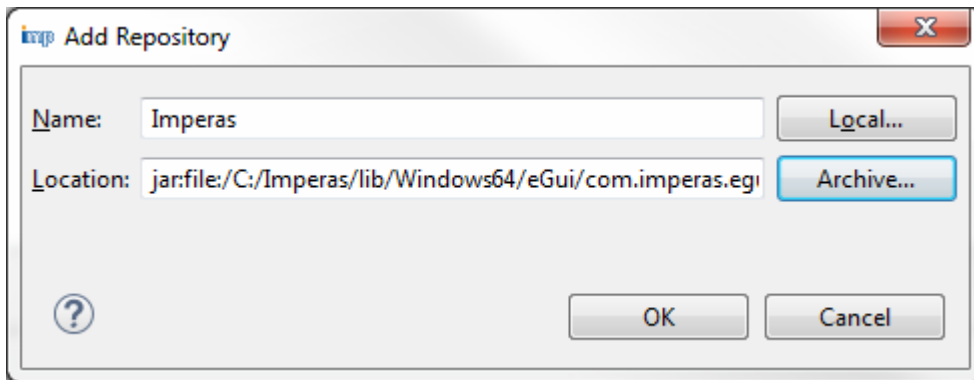
3.3 eGui plugin installation into existing Eclipse

To just use the standalone eGui product, installing the eGui_Eclipse package as described above is sufficient. Users wishing to add the Imperas eGui plugin to an existing Eclipse installation must perform the following additional steps.

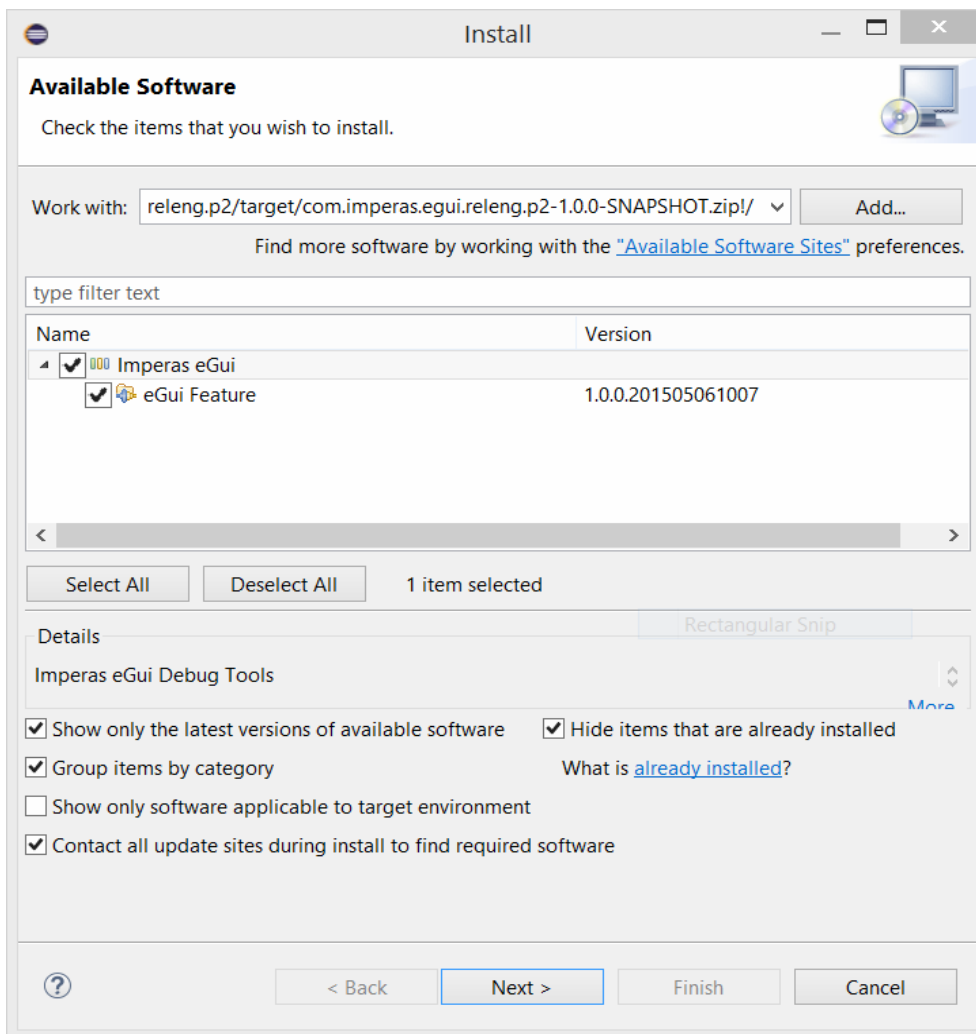
The Eclipse version must be Eclipse Luna version 4.4.2 to use the Imperas eGui plugin. The Imperas eGui plugin is a standard Eclipse plugin and is installed in the normal manner:

1. In a running Eclipse, select *Help-> Install New Software*.
2. In the Install dialog select *Add*.
3. In the *Add Repository* dialog set *Name* to Imperas and select *Archive*.

4. In the *Repository archive* dialog browse to:
 <install directory>/lib/<arch>/eGui
 and select the *com.imperas.egui.releng.p2-<version>.zip* file.



5. Select *OK*.
6. Expand the category *Imperas eGui* and then select the *eGui Feature*.



7. Select *Next* twice, accept terms of the license and select *Finish*.
8. Eclipse will install the eGui feature, downloading any dependencies it may need in the process.
9. Select *OK* when prompted to install unsigned content.
10. Restart Eclipse when prompted to complete the install.

4 Starting a debug session

There are several ways to start a debug session depending on whether you want to just run a standalone debug session or are using Eclipse projects, and also whether you have a license for the Imperas Multi-Processor Debug (MPD) utility.

With Imperas MPD support all of the processors and peripherals in the platform may be debugged simultaneously. Without Imperas MPD support GDB is used directly and only a single processor or peripheral may be debugged at a time.

4.1 Specifying platform options

Debugging is enabled when starting an OVP platform by specifying command line or control file options, or can be built into the platform source.

If the platform includes the Imperas Command Line Parser (CLP) then the platform executable file will support command line options to enable debugging without the need to make any changes to the platform source. See the *OVPsim and CpuManager User Guide* for information on adding the CLP to a platform. The examples in this document assume that the CLP is present in the platform.

An alternative to the CLP is to use a control file to specify platform options at runtime. See the *OVP Control File User Guide* for information on using control files.

The platform source may also enable debugging using ICM or OP API calls. See the respective API documentation for information, although the CLP makes the use of these unnecessary in most cases.

4.2 Starting a standalone debug session

This is the simplest way to start a debug session. Standalone debugging sessions are started by the platform, and do not require using Eclipse projects. eGui will be started if it is not already running.

The following platform options are available for starting a standalone debug session:

--mpdegui

This will connect to eGui in MPD mode, starting it up if it is not already running. MPD mode supports simultaneous debug of all processors and peripherals in the platform. This requires an MPD license.

--gdbegui

This will connect to eGui in GDB mode, starting it up if it is not already running. GDB mode supports debug of a single processor. In platforms with multiple processors the `--debugprocessor <platform>/<cpu>` option will also be needed to

specify the processor to be debugged, where platform and cpu are the respective platform and processor names.

As discussed in section 4.1, these may be specified on the command line of the platform executable (if the CLP processor is present) or in a control file.

4.2.1 The eGui port file

A standalone eGui debug session is initiated from the simulator. The simulator determines whether eGui is already running by looking for the eGui Port file. This file is created when an Eclipse with the eGui feature starts up and it contains the TCP port number that eGui listens to for connections.

The file name for the eGui port file is determined by:

1. The value of the IMPERAS_EGUI_PORT_FILE environment variable, if specified.
2. If that environment variable is not set then the file name defaults to:
 `$HOME/.egui.port` on Linux
 `%USERPROFILE%\egui.port` on Windows

If the eGui port file exists and a listener is found on the port, then a standalone debug session is started in the running eGui which connects to the newly started platform.

If the eGui port file does not exist, or if no listener is found on the port, then the eGui product is launched which then connects to the newly started platform.

Note:

- If the eGui feature has been added to an existing Eclipse installation then using `--mpdegui` or `--gdbegui` will connect to it *if* it is already running, since the eGui feature in that Eclipse will have created an eGui port file.
- When no valid eGui port file exists the platform will only start the eGui product in the Imperas installation - it will not start an Eclipse that has added the eGui feature. Thus if you wish to use your own Eclipse installation with the eGui feature for standalone debugging it must already be running when the platform is started.
- Multiple instances of the eGui feature may run simultaneously. The eGui port file will contain the port for the one started most recently. The IMPERAS_EGUI_PORT_FILE environment variable may be used to control connecting to different instances running simultaneously.

4.3 Starting a debug session from Eclipse

For users who use Eclipse as their IDE, starting a debug session from within Eclipse may be more convenient, but requires setting up Eclipse launch configurations to both run the platform and start the debug session.

4.3.1 Starting the Imperas eGui Eclipse product

The eGui_Eclipse package includes a fully functional Eclipse, referred to here as eGui. Users who do not already use Eclipse or use a version that is incompatible with the Imperas eGui feature may wish to use this as their IDE in addition to using it as a standalone debugger as described in section 4.2. This section contains information on running the Eclipse installation provided by Imperas.

Users who wish to use an existing Eclipse installation should add the eGui feature as described in section 3.3 and start their Eclipse as usual, and can disregard the rest of this section.

eGui is started by executing the command *egui.exe* from any Linux shell or a Windows Command Prompt or MSys shell. *egui.exe* is a wrapper program that verifies the environment and launches the eGui product found in:

`$IMPERAS_HOME/lib/$IMPERAS_ARCH/eGui/eguieclipse{.exe}`

egui.exe arguments include:

--help

Print a help message listing all arguments

--version

Print the version and exit

--verbose

Print additional information that may be useful for troubleshooting

(There are several additional arguments intended for internal use by the simulator when launching standalone debugging sessions that are not documented here.)

Any additional arguments on the command line will be passed to the *eguieclipse* executable, which accepts all the standard Eclipse arguments. Documentation for these may be found by searching the Eclipse documentation for "Running Eclipse".

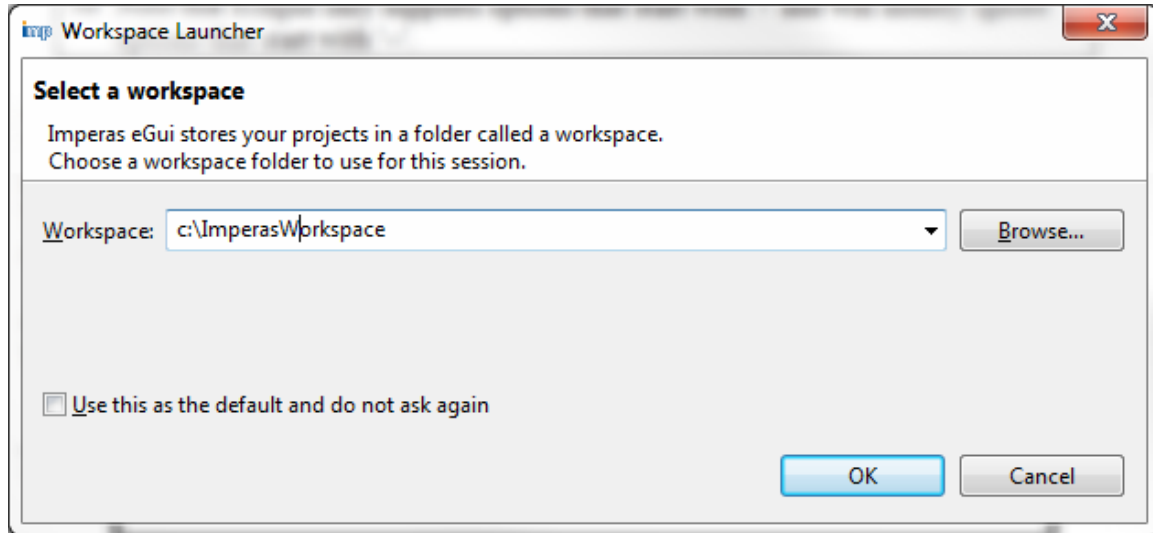
⇒ Note that Eclipse only supports options that start with '-' and will silently ignore options that start with '--'. *egui.exe* will issue a warning if it sees options starting with '--' being passed to *eguieclipse*

4.3.1.1 The Java Runtime Environment (JRE)

The eGui_Eclipse package includes a Java Runtime Environment (JRE) whose root is in the same directory as the *eguieclipse* executable and will be used by Eclipse by default. The Eclipse *-vm* option may be used to override this default JRE. See the Eclipse documentation for additional details.

4.3.1.2 Selecting a workspace

When eGui is started a workspace prompt will appear (unless the *-data* option specifying the workspace directory is specified):



The workspace is where Eclipse stores settings for the session. Settings from a previous session may be used again by using the same workspace.

For standalone debugging you may want to just use a temporary workspace, perhaps in your working directory.

If you are using Eclipse as your IDE you probably want to use a permanent workspace that will save projects and other information that you configure, so select an appropriate location for the workspace, perhaps in your home directory.

See the Eclipse documentation for additional information about workspaces.

4.3.2 Eclipse launch configurations

To start a debug session from within Eclipse you use launch configurations. There are several different types of launch configurations. The following is a brief introduction to the ones relevant to starting debugging sessions for OVP platforms.

An Eclipse *External Tools Configuration* is used to start an OVP simulation platform from within Eclipse. The platform arguments should include the *--port <port number>* command line argument telling the simulator to open a debug port on the specified port number. (If the port number 0 is specified the simulator will open the next available system port and report the port number it selected.)

An Eclipse *Debug Configuration* is then used to launch a debug session that connects to the simulation platform. If an MPD license is available this may use the *Imperas - Connect* debug configuration provided by the eGui feature. If an MPD license is not

available a standard Eclipse *CDT C/C++ Remote Application* debug configuration may be used to connect using GDB.

An Eclipse *Launch Group* may be used to launch both the external tools and debug configurations together with a single button for convenience.

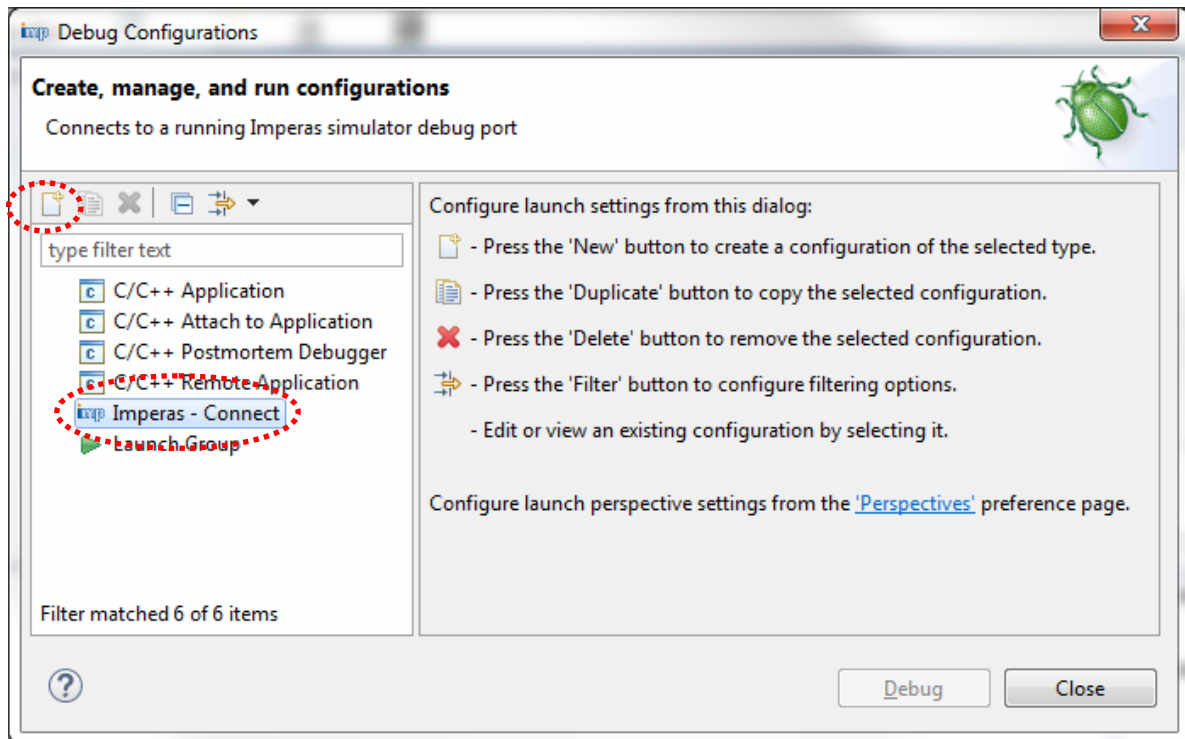
The *Imperas - Connect* debug configuration is specific to eGui and is described in section 4.3.2.1, below.

The *External Tools Configuration*, *C/C++ Remote Application* and *Launch Group* are standard Eclipse features and information on them may be found in the Eclipse documentation. An example of their use is also included later in this document, for those not familiar with this feature of Eclipse.

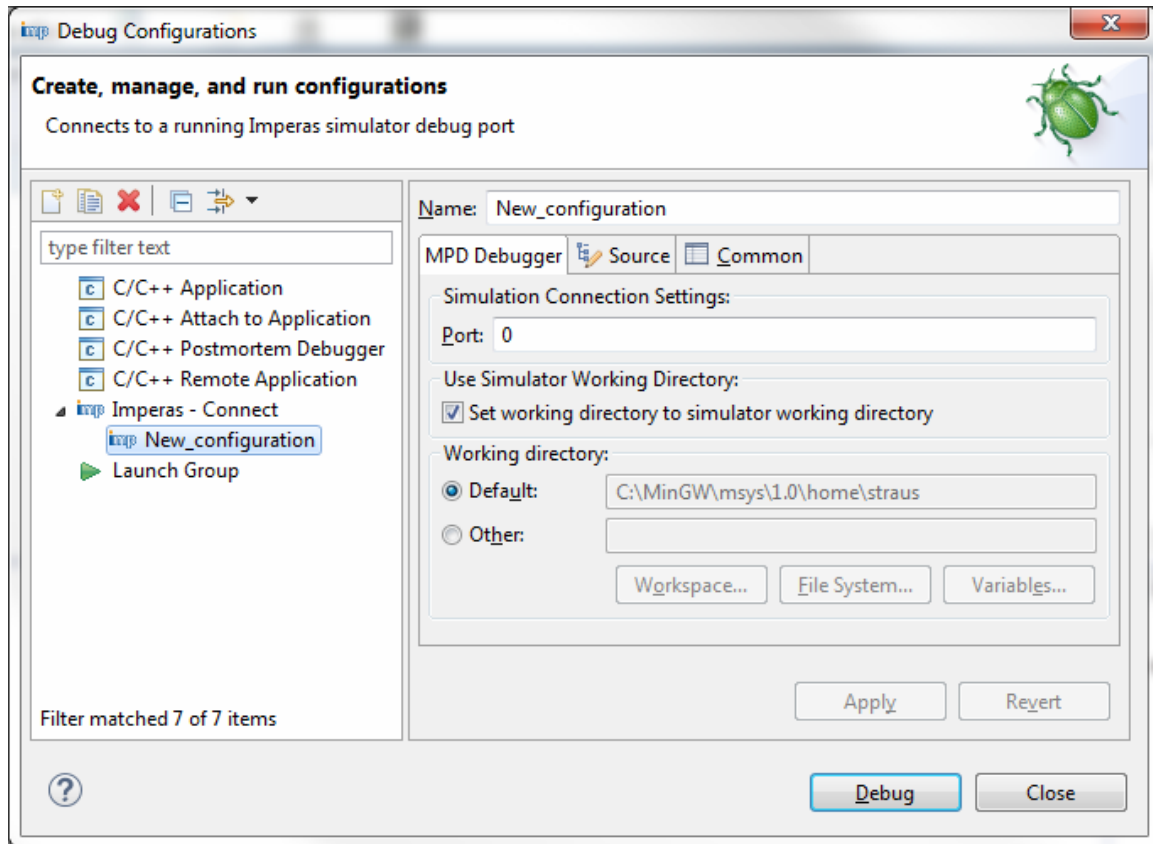
4.3.2.1 The *Imperas - Connect* debug configuration dialog

The *Imperas - Connect* debug configuration is used to start a debug session that connects to an OVP simulation platform using the Imperas Multi-Processor Debugger (MPD). This debug configuration is added to Eclipse by the eGui feature.

Start by selecting *Run->Debug configurations...* to bring up the Debug Configurations dialog. Then select *Imperas - Connect* and click the *New launch configuration* button:



This will bring up the *Imperas - Connect* dialog:



The fields are as follows:

Name

This is the name assigned to the debug configuration, similar to standard Eclipse configurations.

Port

This is the port that the platform is listening to, waiting for a connection. It should match the value specified when the platform was launched, or if a port of 0 was specified for the platform it should match the port number reported by the platform.

Set working directory to simulator working directory

When this is checked the debug session will run with the current directory set to the same working directory as the simulation that it connects to. This should normally be checked.

Working Directory

If the working directory is not obtained from the simulator it may be specified here.

The rest of the tabs function the same way as any other Eclipse debug configuration dialog and the Eclipse documentation should be consulted for information on them.

Examples of the use of this debug configuration may be found below.

<p>⇒ To use the <i>Imperas - Connect</i> debug configuration you must have a license for the Imperas MPD.</p>

5 Standalone eGui Debug Demos

The Imperas installation includes a number of demos of processors running simple applications. These demos may be easily run in standalone debugging sessions by running with additional debug options specified.

The following examples require installation of the prerequisites described in section 2.2, including the armv7.toolchain package, and the eGui_Eclipse package described in section 3.1.

Also, the Imperas environment must be setup according to the directions in the *Imperas Installation and Getting Started Guide*.

5.1 eGui in GDB mode on Cortex-A15UP single core demo

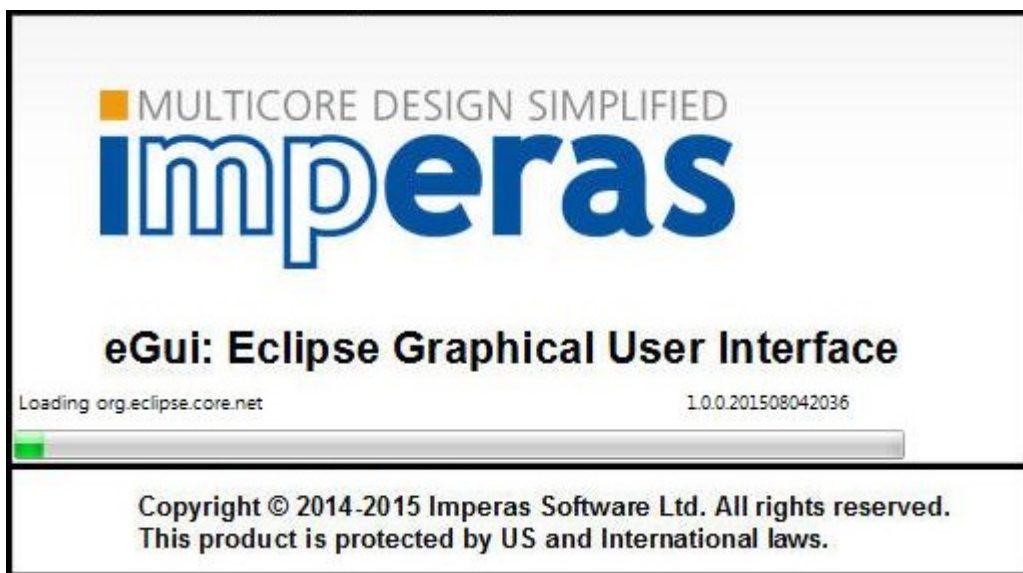
First we must change to the demo directory and start the demo in MPD mode. From a Linux shell or a Windows MSys shell:

```
cd $IMPERAS_HOME/Demo/Processors/ARM/Cortex/Cortex-A15UP/single_core  
./Run_Dhrystone.sh --gdbgui
```

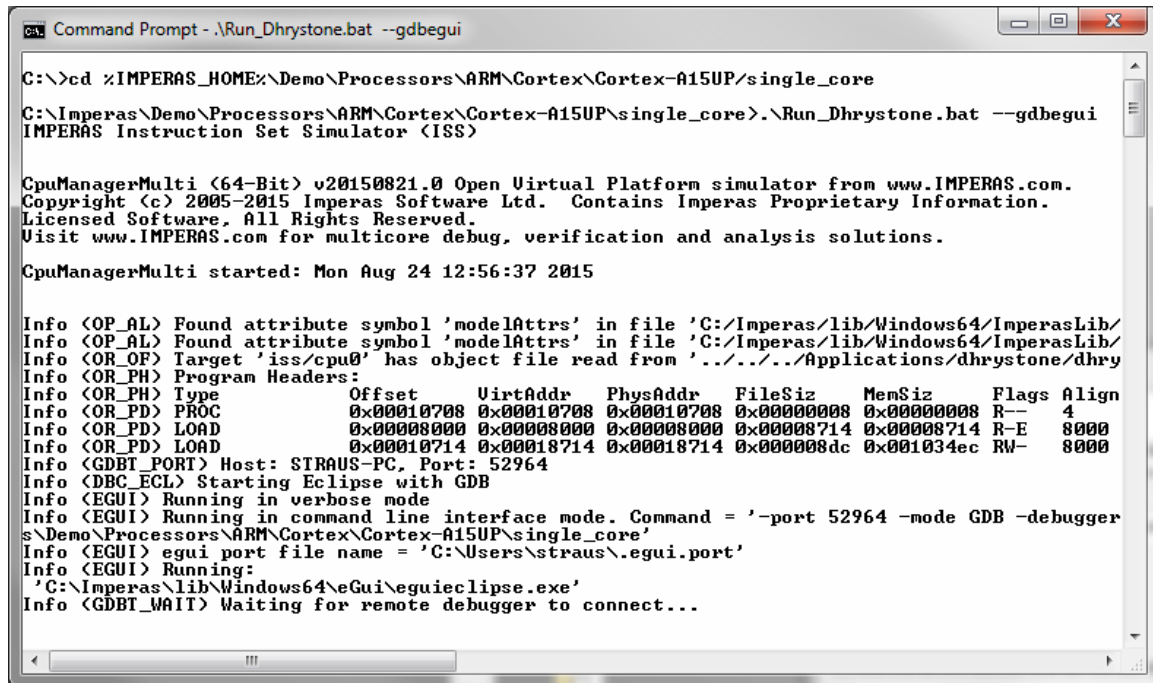
or, from a Windows command shell:

```
cd %IMPERAS_HOME%\Demo\Processors\ARM\Cortex\Cortex-A15UP\single_core  
.\Run_Dhrystone.bat --gdbgui
```

When starting the Imperas eGui splash screen will be displayed with loading status bar



The shell from which the script was invoked will show the connection messages:



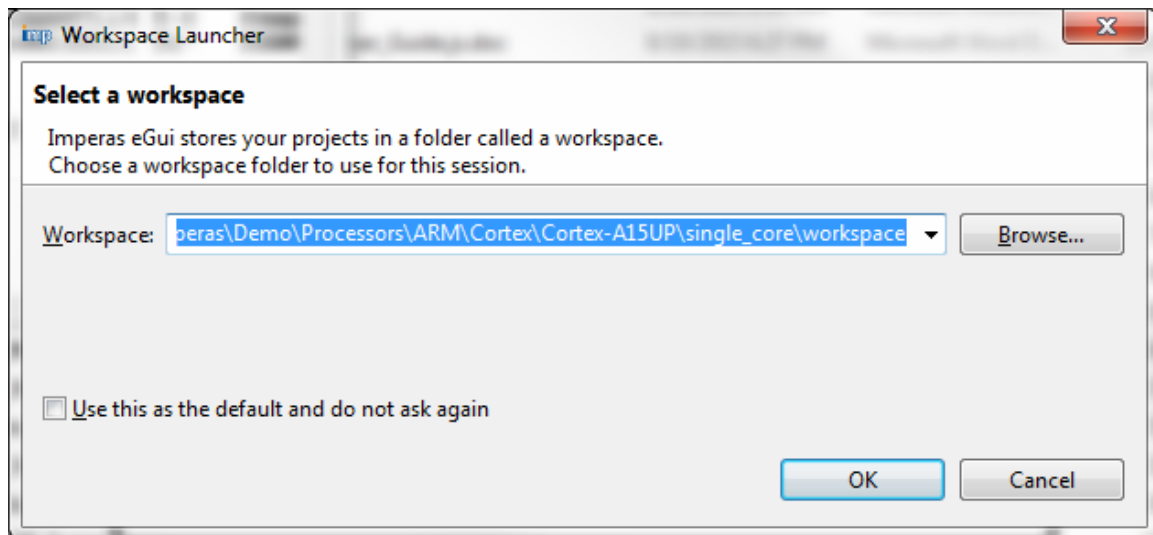
```

C:\>cd %IMPERAS_HOME%\Demo\Processors\ARM\Cortex\Cortex-A15UP\single_core
C:\Imperas\Demo\Processors\ARM\Cortex\Cortex-A15UP\single_core>.\Run_Dhrystone.bat --gdbgui
IMPERAS Instruction Set Simulator (ISS)

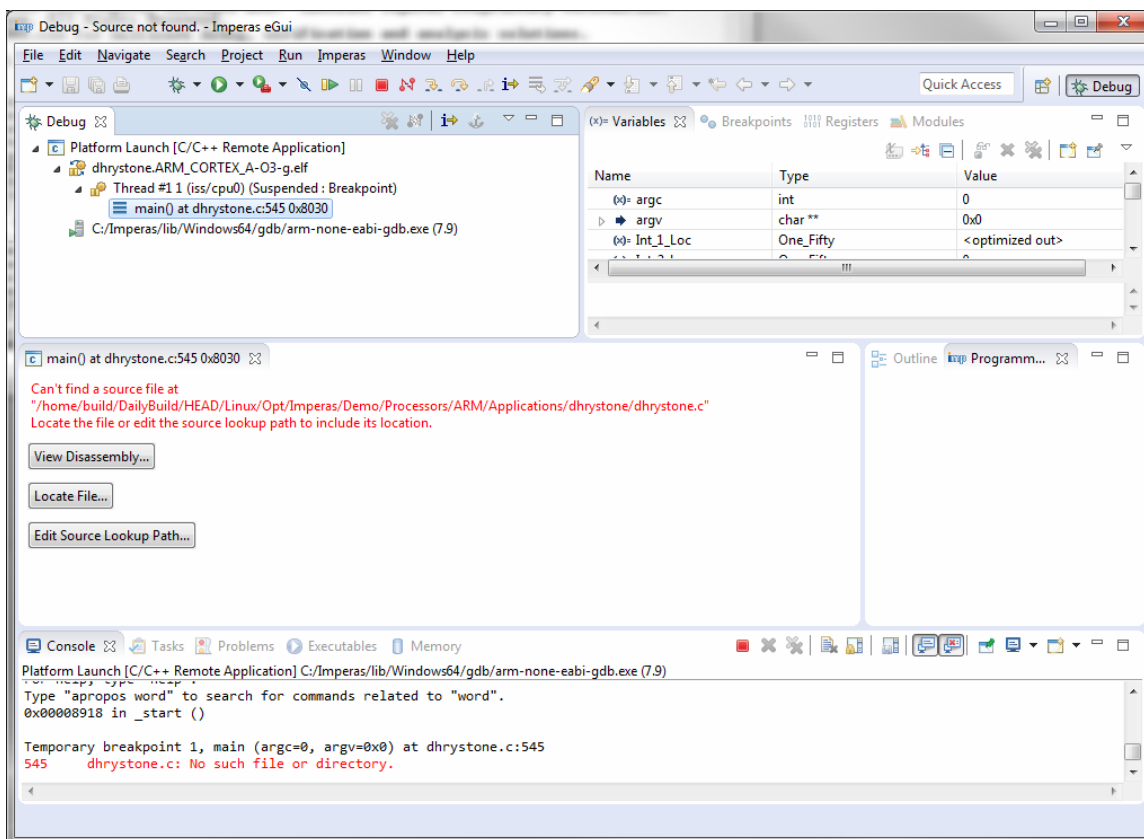
CpuManagerMulti <64-Bit> v20150821.0 Open Virtual Platform simulator from www.IMPERAS.com.
Copyright (c) 2005-2015 Imperas Software Ltd. Contains Imperas Proprietary Information.
Licensed Software. All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.
CpuManagerMulti started: Mon Aug 24 12:56:37 2015

Info <OP_AL> Found attribute symbol 'modelAttrs' in file 'C:/Imperas/lib/Windows64/ImperasLib/
Info <OP_AL> Found attribute symbol 'modelAttrs' in file 'C:/Imperas/lib/Windows64/ImperasLib/
Info <OR_OF> Target 'iss/cpu0' has object file read from '.....Applications/dhrystone/dhry
Info <OR_PH> Program Headers:
Info <OR_PH> Type          Offset      VirtAddr  PhysAddr  FileSiz  MemSiz   Flags  Align
Info <OR_PD> PROC          0x00010708 0x00010708 0x00010708 0x00000008 0x00000008 R--   4
Info <OR_PD> LOAD          0x00008000 0x00008000 0x00008000 0x00008714 0x00008714 R-E   8000
Info <OR_PD> LOAD          0x00010714 0x00018714 0x00018714 0x000008dc 0x001034ec RW-   8000
Info <GDBT_PORT> Host: STRAUS-PC, Port: 52964
Info <DBC_ECL> Starting Eclipse with GDB
Info <EGUI> Running in verbose mode
Info <EGUI> Running in command line interface mode. Command = '-port 52964 -mode GDB -debugger
s\Demo\Processors\ARM\Cortex\Cortex-A15UP\single_core'
Info <EGUI> egui port file name = 'C:\Users\straus\.egui.port'
Info <EGUI> Running:
'C:\Imperas\lib\Windows64\egui\eguiclipse.exe'
Info <GDBT_WAIT> Waiting for remote debugger to connect...
  
```

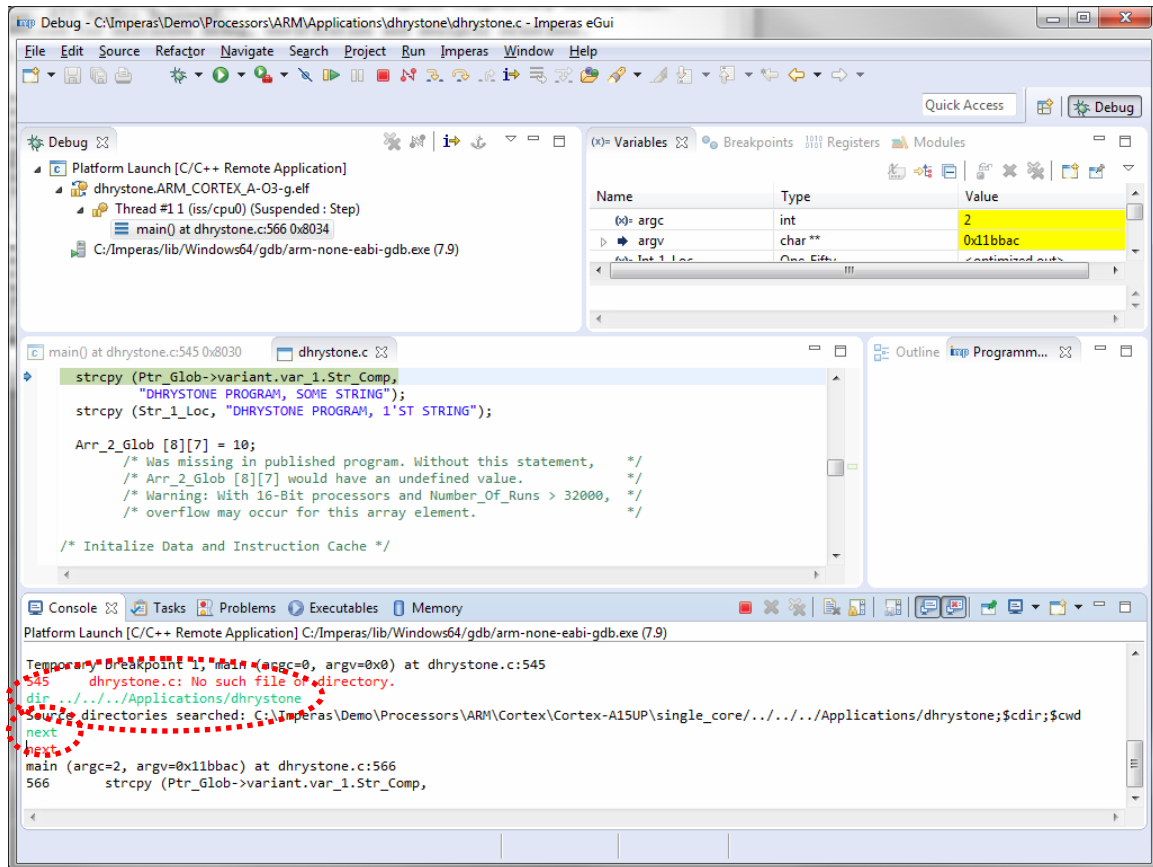
Eclipse will prompt you for a workspace. Since we are just running a standalone debugging session we can create a new workspace in the current directory:



Once started Eclipse will be in the debug perspective stopped at main in the Dhrystone application waiting for debug commands:



However the debugger will not be able to find the source file so we need to tell it what directory it resides in and then step one line to read the source file in the new location (this would not be necessary if we were using an application compiled locally as opposed to one provided as part of the Imperas installation):

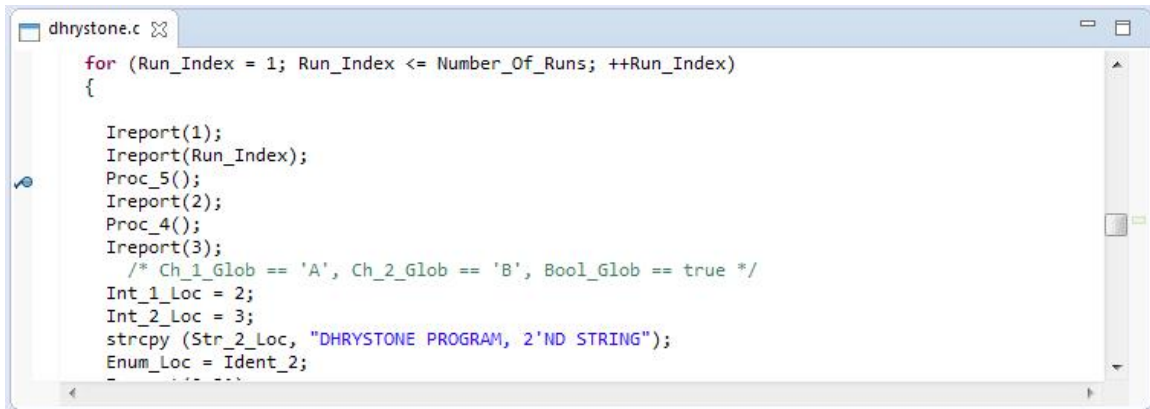


Now we can debug the program normally using the standard Eclipse features as described in the following section.

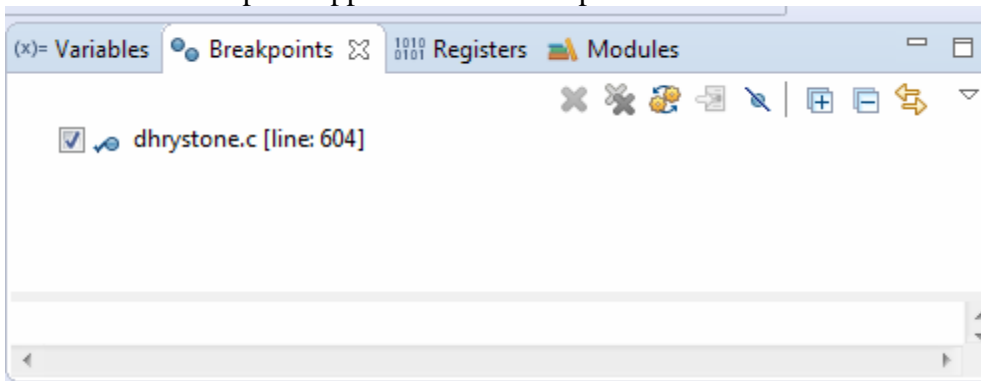
5.1.1 Example Debug Session for Dhrystone Benchmark application

You should be familiar with the general debug control 'buttons' within Eclipse in order to work through the following

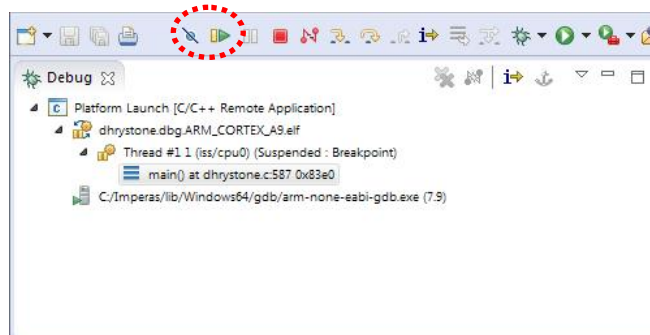
1. Set a breakpoint on the line containing the call to 'Proc_5();' by double clicking beside this line; this is found on line 604:



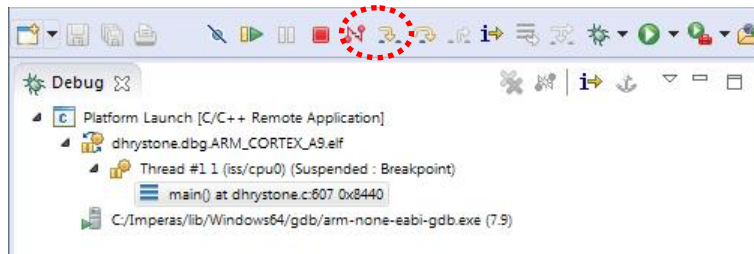
Once set the breakpoint appears in the Breakpoints window:



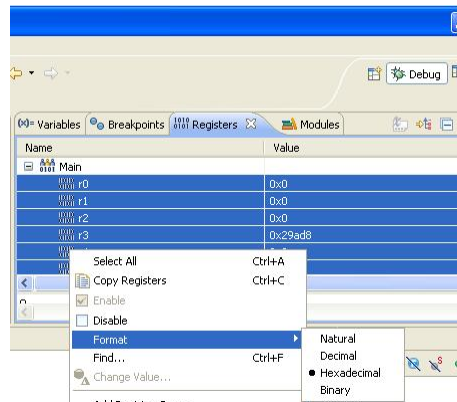
2. Continue the simulation using the green 'Resume (F8)' symbol



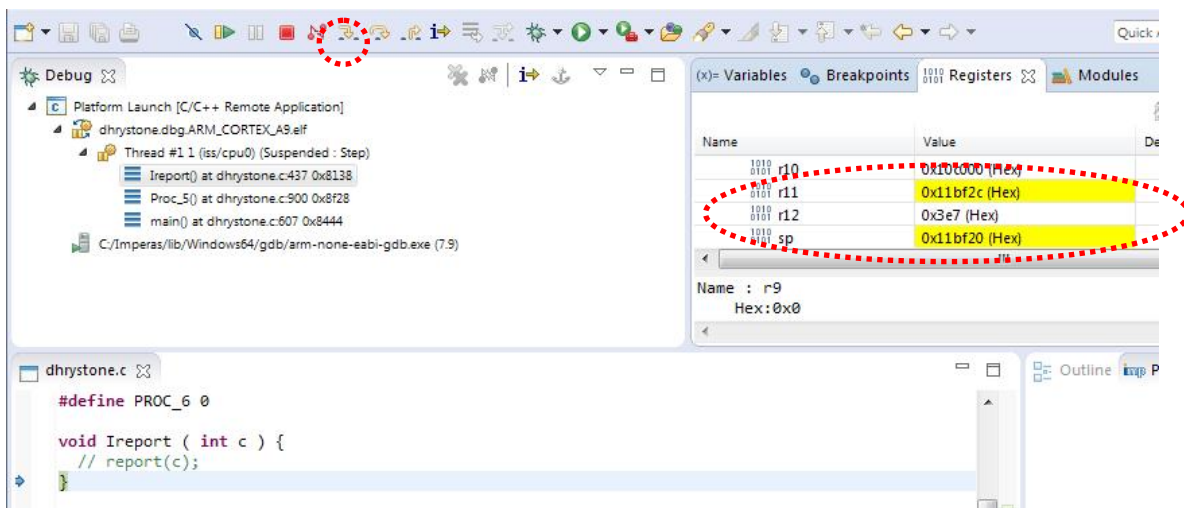
3. Step the application forward using the 'Step Into' button



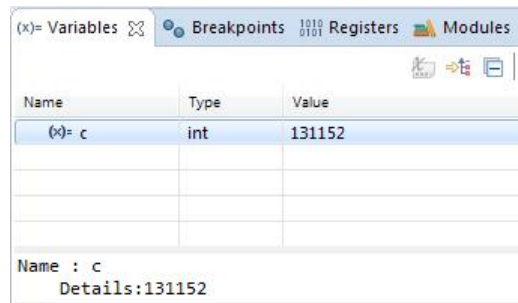
4. Open the register view; expand 'General Registers', select registers and set the format to hexadecimal



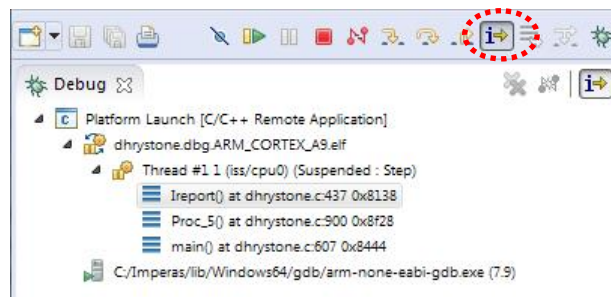
5. Step the application forward using the 'Step Into' button. This will show you the colored highlighting of the registers that change



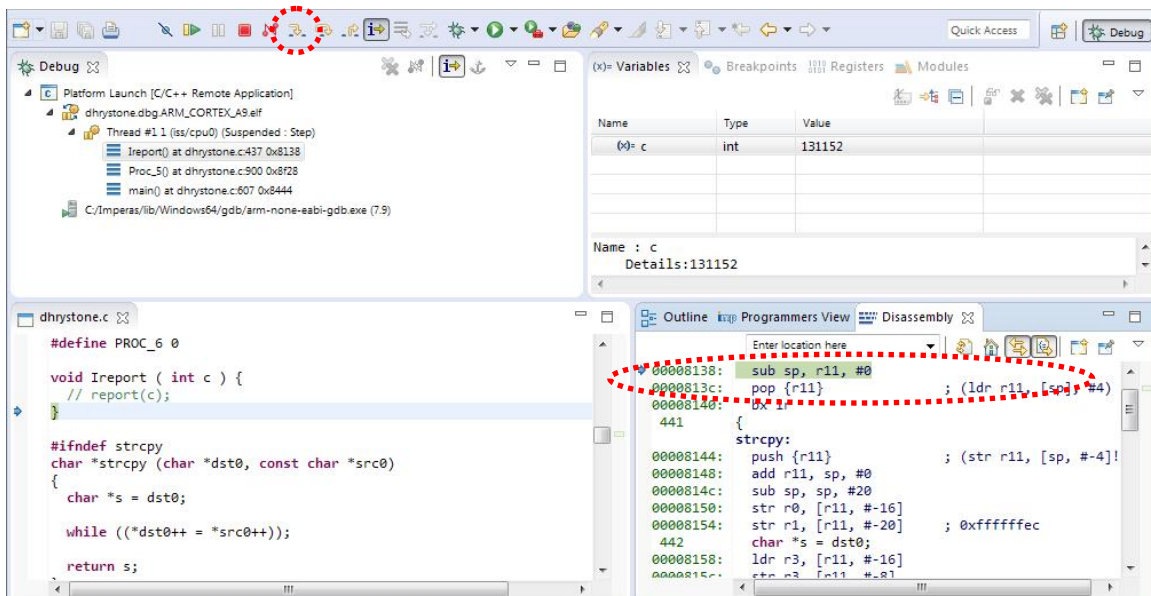
- Open the Variables view; showing the current local variables



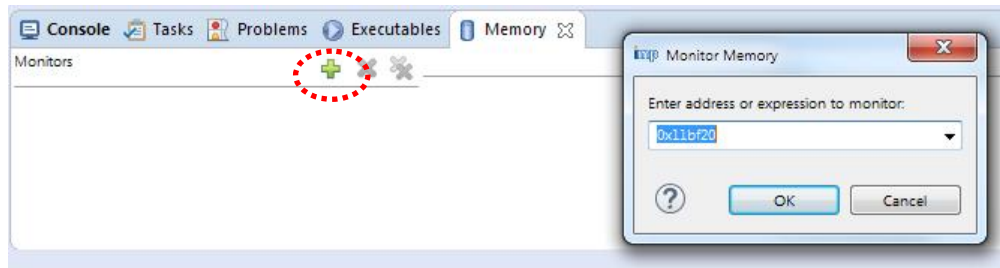
- Select 'Instruction Stepping Mode' using the button.



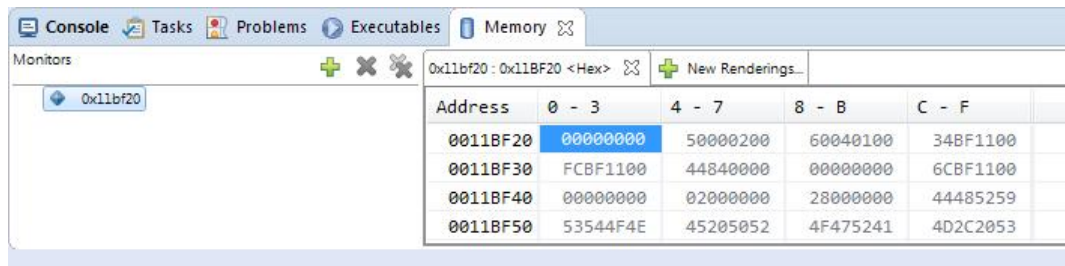
- Step the application forward using the 'Step Into' button this will open the disassembly view and highlight the instructions as they are stepped.



9. In the 'Memory' window, select 'Add Memory Monitor' and use the value of the register 'SP' (ensure the bottom two bits are zeros) to set the memory monitor base address so that it will show the program stack.



10. Move the application forward using the 'Step Return' button. The memory locations that are modified will be highlighted.



11. To finish the simulation you may either:
 - a. Use 'Resume' to continue to the end of the application
 - b. Use 'Terminate'
 After either of these the debug perspective can be cleaned up with the 'Remove All terminated Launches' button

5.1.2 Removing Breakpoints

Breakpoints are persistent between runs within Eclipse. It is not essential but it can be a good idea to remove all breakpoints prior to closing down Eclipse so that breakpoints are not applied erroneously on files when a different Eclipse session is invoked.

6 An Eclipse Project Example

The following section provides information for the launching of the Imperas eGui to be used for a software development project using an Imperas/OVP virtual platform.

NOTE:

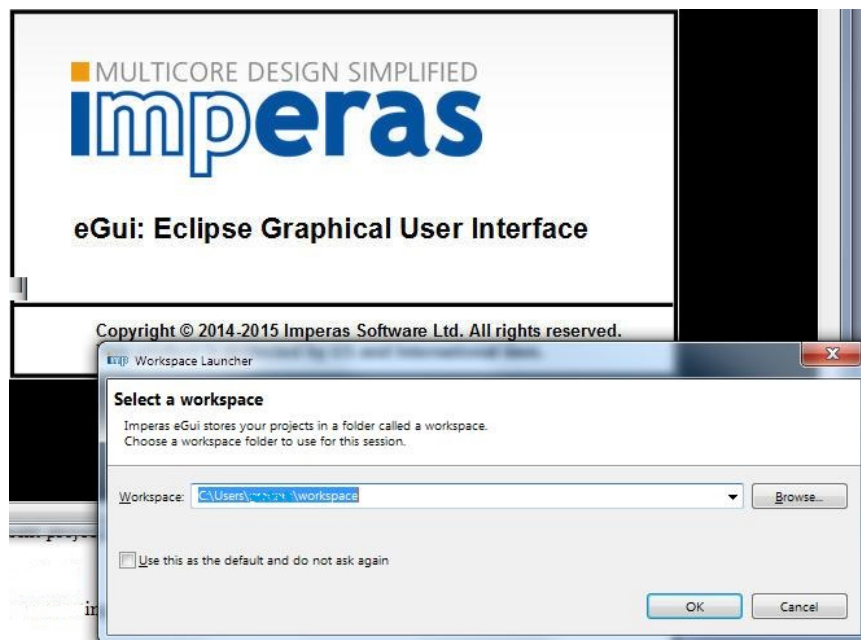
The debug connection launch from within Eclipse requires that the Imperas Professional simulator and MPD debugger are installed. The GDB only debug connection is only supported for a debug connection invoked from the platform command line.

To launch a standard single GDB debug session please refer to section 6.7.3 "Create an OVPSim CDT Debug Configuration"

6.1 Starting Imperas eGui

The Imperas eGui is started by executing the egui.exe program from a shell on Linux or Windows.

You should select a workspace in which Eclipse will keep all your local information



6.2 Selecting the C/C++ Perspective

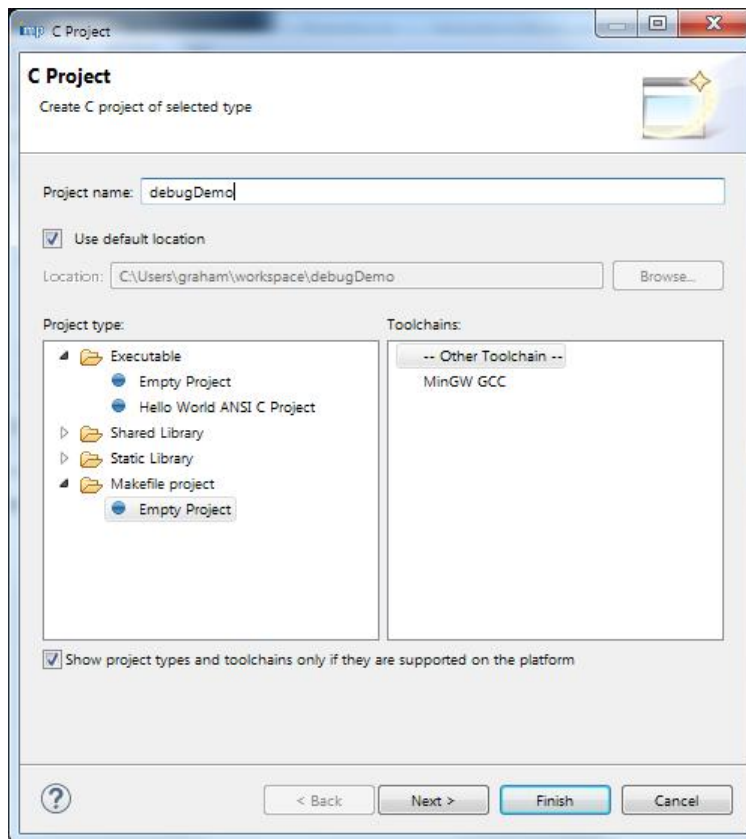
When the eGui starts it opens in the 'Debug' perspective. Select the C/C++ perspective using the tab at the screen top right corner or 'Open Perspective' under the Windows menu.

6.3 Preparing an Eclipse Project

This section shows all the steps needed to create an Eclipse project which allows the example platform and target application to be built and debugged in the Imperas Eclipse development environment.

6.3.1 Creating a New Project

Create a new C project by selecting “New” from the “File” menu, then selecting “C Project”. If this option is not present you may not have the C development plugins (CDT) for Eclipse – see section 2.2.



Select “Makefile project” as we will use an external Makefile within the project. Specify a project name and optionally a location. Select 'Other Toolchain' as this is the application cross compiler toolchain that will be configured in the demo Makefile.

NOTE:

The project location path must not include spaces as Eclipse passes the project directory to an underlying Gnu debugger in a way that does not handle spaces correctly.

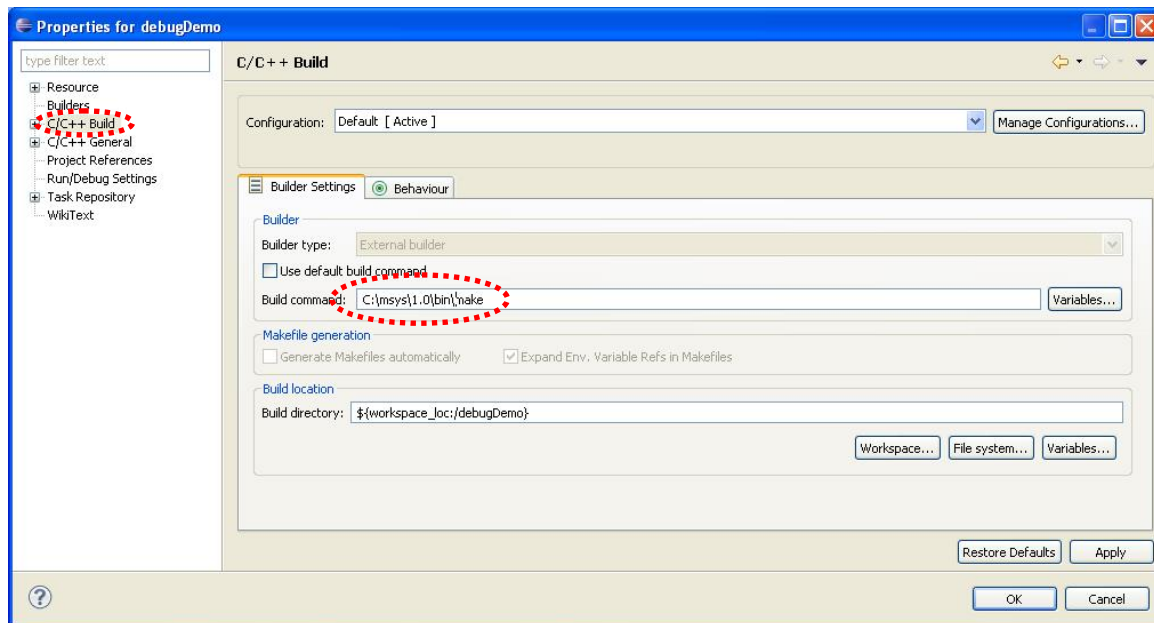
The default workspace will be setup when you invoked Imperas eGui Eclipse. This may be used or an alternative specified by unchecking the “Use default location” checkbox and specify a location. Click the “Finish” button to create the project.

The new project, at this stage, contains no Makefile or source files so Eclipse will show an error message if it attempts to build at this point.

6.3.2 Configuring Project Build Commands (Windows Only)

6.3.2.1 Using MINGW ‘make’

Change the project build configuration so that Eclipse can find the MinGW “make” tool used to build the project. This step may not be necessary if you already have a version of GNU “make” on your system path. Select “Properties” from the “Project” menu and select the “C/C++ Build” pane.



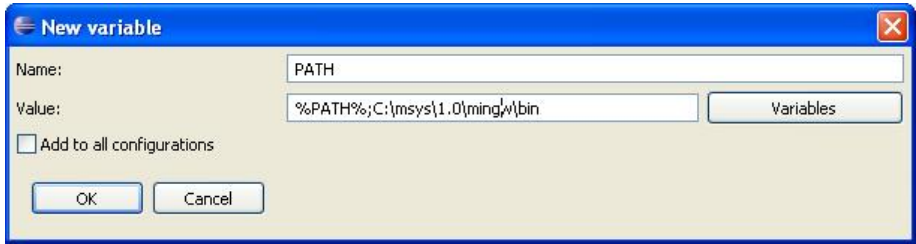
Uncheck the “Use default build command” checkbox and specify the full path of the MinGW/MSYS make executable on your system in the “Build command:” text box. In the example shown the default MSYS install location of “c:\msys\1.0\bin\make” is used.

6.3.2.2 Adding MinGW tools onto PATH

It is recommended that your MinGW binary path is appended to the system path within your Windows environment. In the example the MinGW install location of “c:\msys\1.0\mingw\bin” is used.

However, it may be added into your Eclipse environment by adding a variable “PATH” which includes the current value of PATH (given by %PATH%) and the MinGW path separated by a semi-colon.

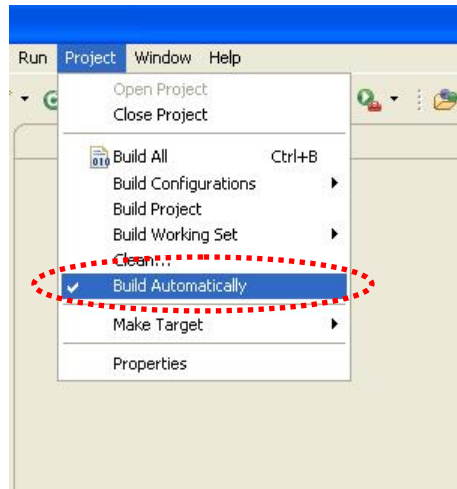
Add the MinGW native compilation tools to the path by selecting the “Environment” tab of the same dialog and pressing the “Add...” button.



6.3.3 Deselect 'Autobuild' from Eclipse

By default, a new project in Eclipse will have autobuild selected. This can be useful, but when debugging cross compiled applications on an ICM platform this may cause the build process to be re-invoked between the launching of the simulation platform and the attaching of the debugger to allow application debug.

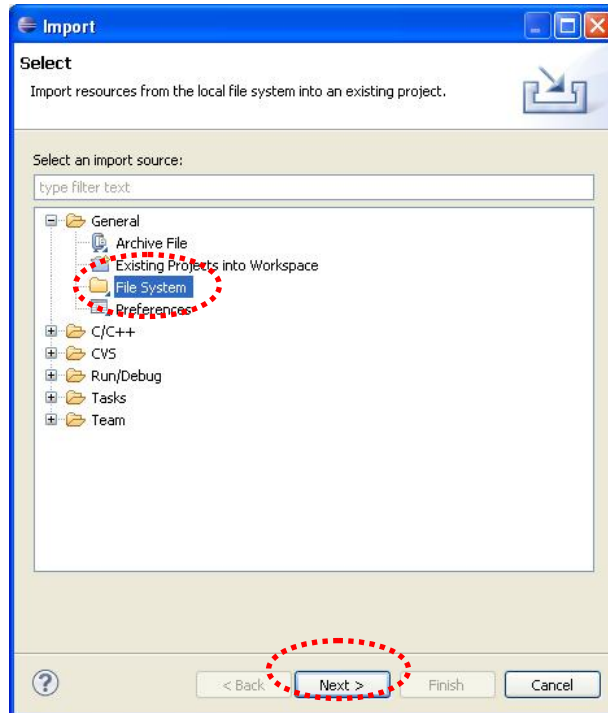
Select the "Project" menu and click on 'build automatically' to de-select it.



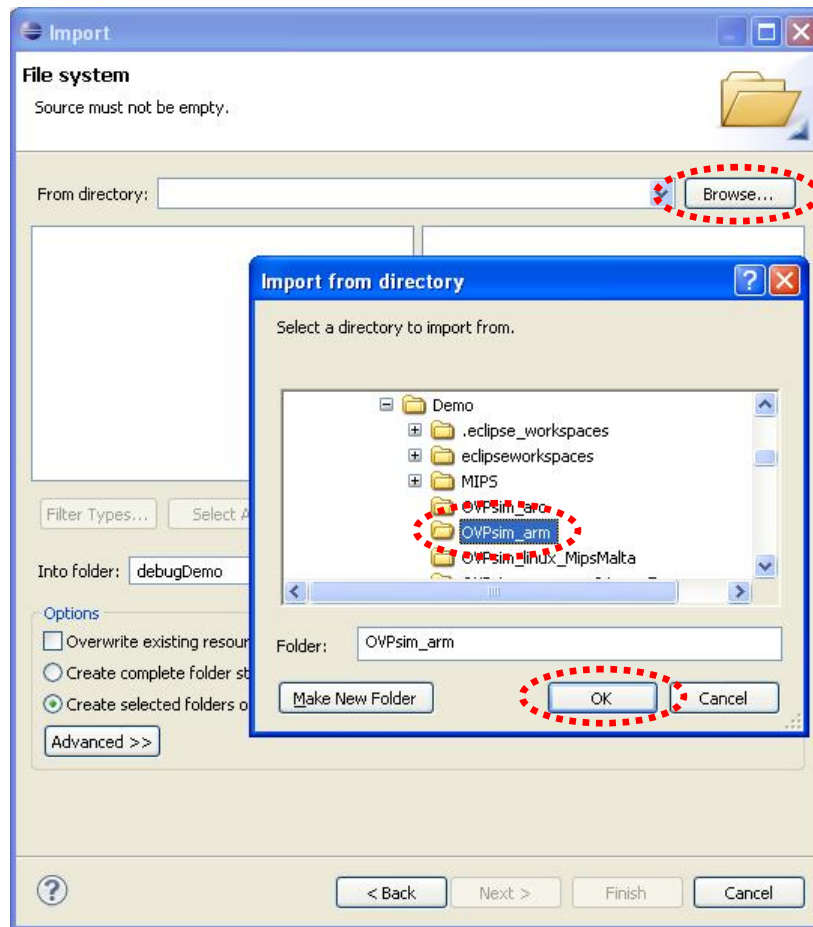
6.4 Importing a Demo to the Project

For the purposes of this document we will import the demo provided as part of a product installation for the ARM processor, that is, OVPSim_arm. We could equally choose the OVPSim_mips32 or other processor demo. The only difference is the Cross Compiler toolchain setting and the GDB debugger selected.

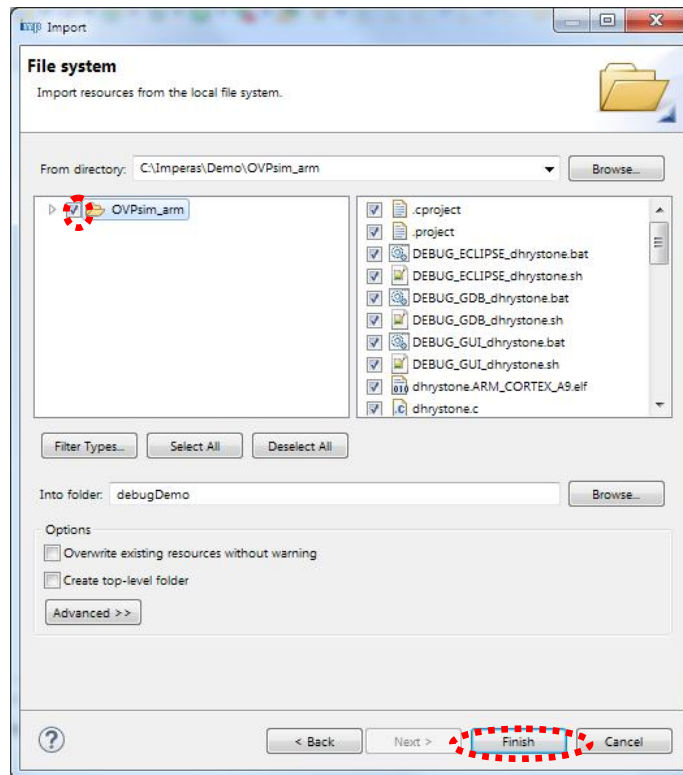
The files are imported to a project by selecting the File->Import menu and selecting 'File System' and followed by 'Next'



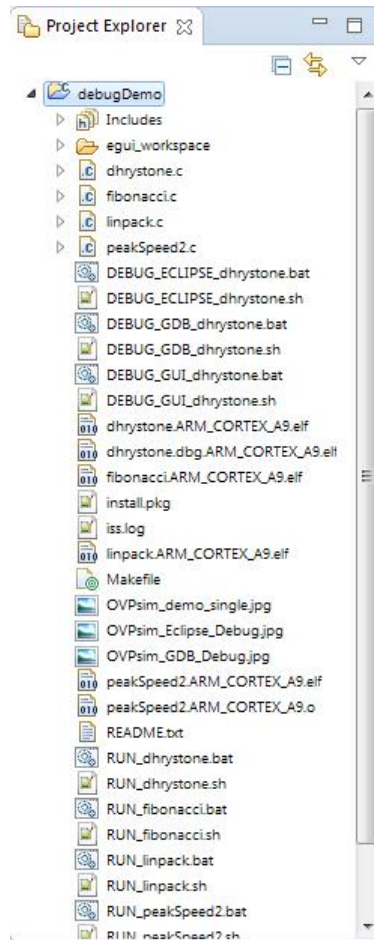
This brings up a further menu from which you may browse to the Imperas installation and the installed Demos. Select the demo directory that you wish to import. This may be any demo supplied as part of an OVP installation.



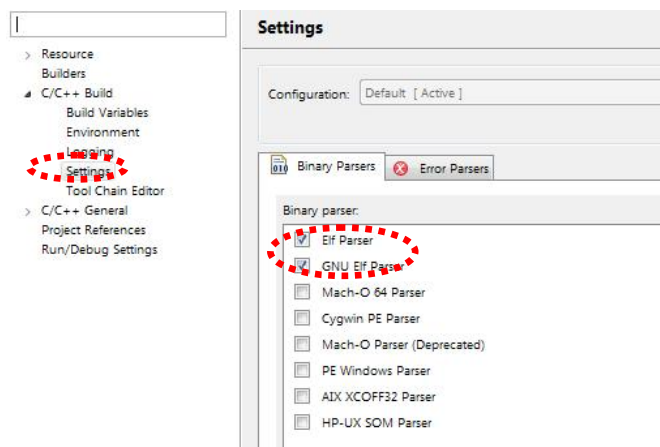
Once the directory containing the demo files has been selected, you will be prompted for which files you wish to import. In general this is ALL files, if you wish to exclude some files, please ensure that the source files and the exe and elf files are imported.



Once imported your project will contain all the files



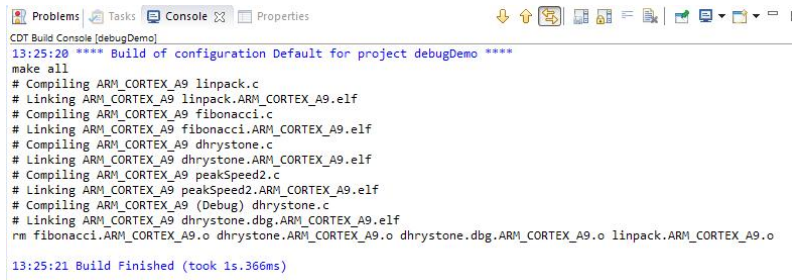
Select the project properties and ensure the C/C++ Build, Settings for the Binary Parser has the Elf parsers ticked.



6.5 Building the applications locally

It is recommended to build the applications locally so that the build paths are correct. This is accomplished using the Makefile provided in the demo directory and an installed cross compiler.

Select the build 'hammer' icon or in the project right click and select 'Build Project...', you may need to clean the project to ensure binaries are re-built.



```
CDT Build Console [debugDemo]
13:25:20 **** Build of configuration Default for project debugDemo ****
make all
# Compiling ARM_Cortex_A9 linpack.c
# Linking ARM_Cortex_A9 linpack.ARM_Cortex_A9.elf
# Compiling ARM_Cortex_A9 fibonacci.c
# Linking ARM_Cortex_A9 fibonacci.ARM_Cortex_A9.elf
# Compiling ARM_Cortex_A9 dhrystone.c
# Linking ARM_Cortex_A9 dhrystone.ARM_Cortex_A9.elf
# Compiling ARM_Cortex_A9 peakSpeed2.c
# Linking ARM_Cortex_A9 peakSpeed2.ARM_Cortex_A9.elf
# Compiling ARM_Cortex_A9 (Debug) dhrystone.c
# Linking ARM_Cortex_A9 dhrystone.dbg.ARM_Cortex_A9.elf
rm fibonacci.ARM_Cortex_A9.o dhrystone.ARM_Cortex_A9.o dhrystone.dbg.ARM_Cortex_A9.o linpack.ARM_Cortex_A9.o
13:25:21 Build Finished (took 1s.366ms)
```

6.6 Virtual Platform Configuration for Application Debug

We now have a project containing the application we wish to debug and scripts to run this application on a platform.

We may need to modify the platform so that it can be launched from the Eclipse and connect to the Eclipse debugger. This section shows how the platform can be modified, using the standard command line parser, using the C API or how to use the Imperas ISS.

6.6.1 Command Line arguments for debugging on a C Platform

If the platform has the Imperas Command Line Parser (CLP) included it makes available command line options to enable and control debugging of an application on a processor in the platform without the need to make any changes to the platform source.

6.6.1.1 Specifying the debugger connection details

The debug port is enabled by specifying the argument `--port <port number>` on the command line. A specific port number may be specified or by setting port number to 0 the next available port is opened.

6.6.1.2 Nominating the debugged processor

In an OVPsim simulation only a single processor may be connected to a GDB debugger¹. This requires that the processor is selected using the `--debugprocessor <processor name>`. In this case the processor name is the instance name in the platform, for example `platform/cpu0`.

¹ The Imperas Professional products allow the ability to attach a GDB debugger to any or all the processors defined in a platform. Imperas also provide alternative debugging solutions.

The instance names can be found by executing the platform with the option `--showoverrides` specified on the command line. The platform will not execute but will provide a list of all the instances and their parameter overrides, from which the instance names can be obtained.

6.6.2 ICM C Platform API calls for enabling debug

NOTE:

It is recommended to use the Imperas Command Line Parser (CLP) on platforms rather than using this approach. The use of the CLP provides more flexibility for the platform usage.

The following platform shows the modifications required to open a debug port for connection:

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

    int portNum;

    if(argc != 2) {
        icmPrintf("usage: %s <port number>\n", argv[0]);
        return -1;
    }

    sscanf(argv[1], "%d", &portNum);

    // initialize CpuManager
    icmInitPlatform(ICM_VERSION, 0, "rsp", portNum, "platform");

    // create a processor
    icmProcessorP processor = icmNewProcessor(
        "CPU_0",          // processor name
        CPU_TYPE,         // CPU type
        0,                // processor cpuId
        0,                // processor model flags
        32,               // address bits
        MODEL_FILE,       // model file
        "modelAttrs",     // morpher attributes
        ICM_ATTR_DEBUG,   // DEBUG THIS PROCESSOR
        0,                // user-defined attributes
        SEMIHOST_FILE,    // semi-hosting file
        "modelAttrs"      // semi-hosting attributes
    );

    // load the processor object file
```

```
icmLoadProcessorMemory(processor, "application.elf", False, False);

// run simulation
icmSimulatePlatform();

// terminate simulation
icmTerminate();

return 0;
}
```

For a full explanation of OVPsim platform construction please see the OVPsim and CpuManager User Guide. This section describes only those aspects of platform construction that relate to debugging.

6.6.2.1 Specifying the debugger connection details

The ICM kernel is initialized by calling `icmInitPlatform`:

```
void icmInitPlatform(ICM_VERSION, Uns32 simAttrs, const char *dbgProtocol,
Uns32 dbgPort, char *platform)
```

The second and third arguments of `icmInitPlatform` are used for single processor debugging:

```
icmInitPlatform (ICM_VERSION, 0, "rsp", portNum, "platform");
```

Eclipse uses the GNU debugger for C/C++ debugging. GDB Remote Serial Protocol (RSP) debugging as supported by OVPsim uses standard operating system sockets on the host running OVPsim and GDB.

If a NULL value is given for `dbgProtocol`, debugging is disabled. To enable debugging, specify the protocol for the debug connection, currently only the 'rsp' protocol is supported.

The `dbgPort` argument specifies the socket port on which to accept a debugger connection. The special value of zero allows OVPsim to choose any free port on the host, otherwise the specified port number is used. For the example platform the port number is given by the first command line argument.

6.6.2.2 Nominating the debugged processor

A single instance of a processor is defined by calling `icmNewProcessor`:

```
icmProcessorP processor0 = icmNewProcessor(
```

```
"CPU_0",      // processor name
CPU_TYPE,     // CPU type
0,           // processor cpuId
0,           // processor model flags
32,          // address bits
MORPHER_FILE, // model file
"modelAttrs", // morpher attributes
ICM_ATTR_DEBUG, // DEBUG THIS PROCESSOR
0,           // user-defined attributes
SEMIHOST_FILE, // semi-hosting file
"modelAttrs" // semi-hosting attributes
);
```

The processor to be debugged is indicated by specifying the ICM_ATTR_DEBUG processor attribute.

6.6.3 Debugging applications with the Imperas ISS

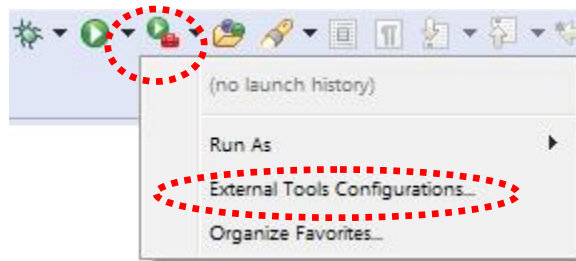
The Imperas ISS allows simple applications to be cross compiled and easily executed on a virtual platform containing one or more of the target processors.

6.7 Creating Launch Configurations

6.7.1 Virtual Platform launch

Before each debug session the virtual platform must be launched on the host machine.

The Eclipse “External Tools” facility is a convenient way to do this. Select “Open External Tools Dialog...” from the “External Tools” menu.



Create a new external tool configuration and give it a meaningful name.

The platform executable could be a local platform executable within the demo directory, a platform executable from the Imperas VLNV library or the ISS (found in the binary directory).

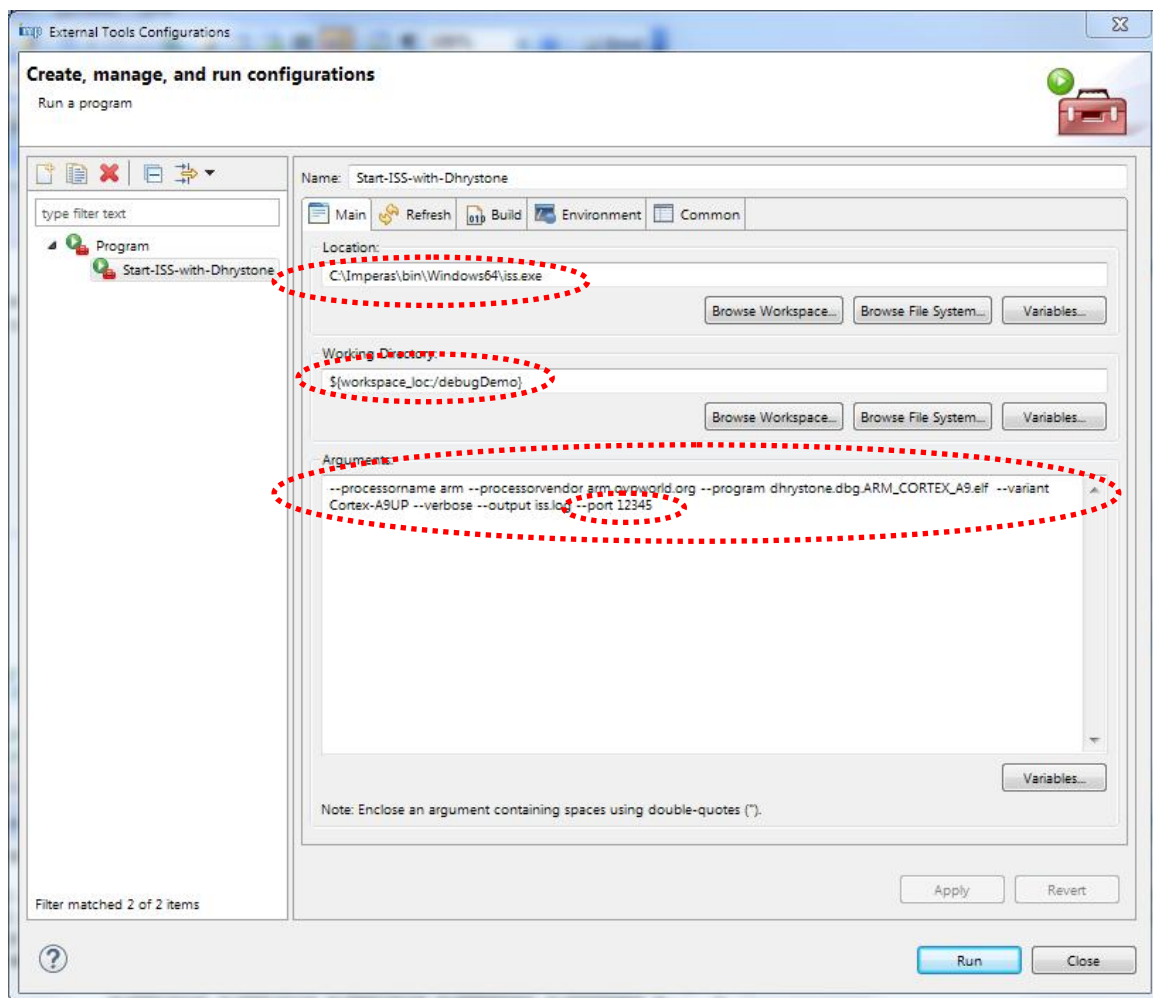
It is useful to examine one of the standard "RUN" scripts provided in the demo directory which will show you the location of the platform to be used and also the command line arguments to be passed to the platform executable.

Press one of the "Browse Workspace..." or "Browse File System..." buttons to go and select the platform executable.

The demo, OVPsim_arm, we have imported makes use of the ISS so we will select the ISS through the Browse File System button.

The arguments added into the arguments section must match the arguments that the executable platform expects.

In addition to the standard arguments required to execute the application on the virtual platform we also add the --port <port number> argument to open the port for attaching a debug connection. In this case we specify the port 12345, matching the port we chose in the debug configuration in section 6.7.2.

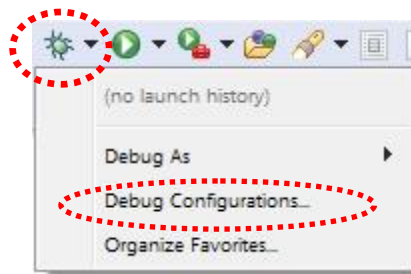


Press “Apply”, then “Close” to save the external tool configuration without starting it.

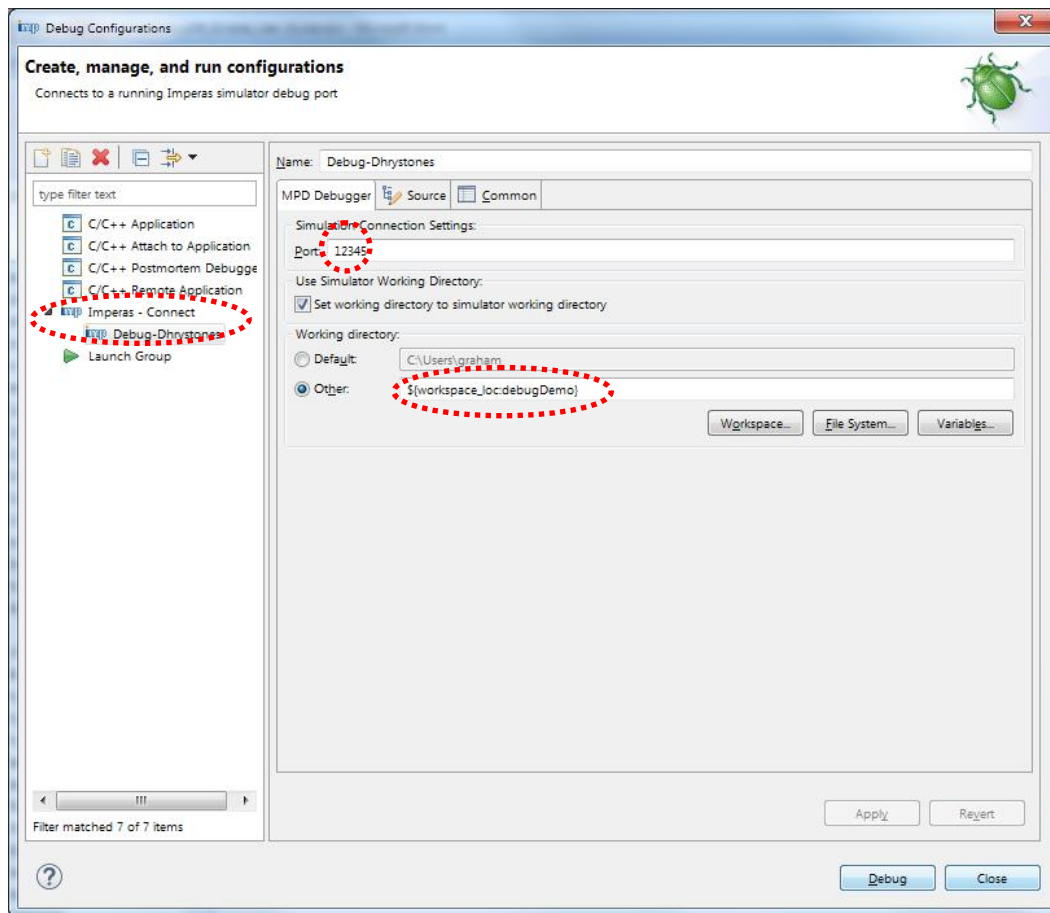
6.7.2 Create an Imperas MPD Debug Configuration

If the Imperas_SDK package is installed, the Imperas professional simulator is used and the Imperas Multi-Processor Debug environment is available. This allows us to utilize the full features of the eGui debug environment for multiple application and peripheral behavioral model development.

Create a debug configuration for the simulated application by selecting “Debug Configurations...”



Create a new “Imperas Connect” configuration.



The port number should be the same as the port specified in the virtual platform launch configuration and the working directory should be selected as that of the project directory containing the application files.

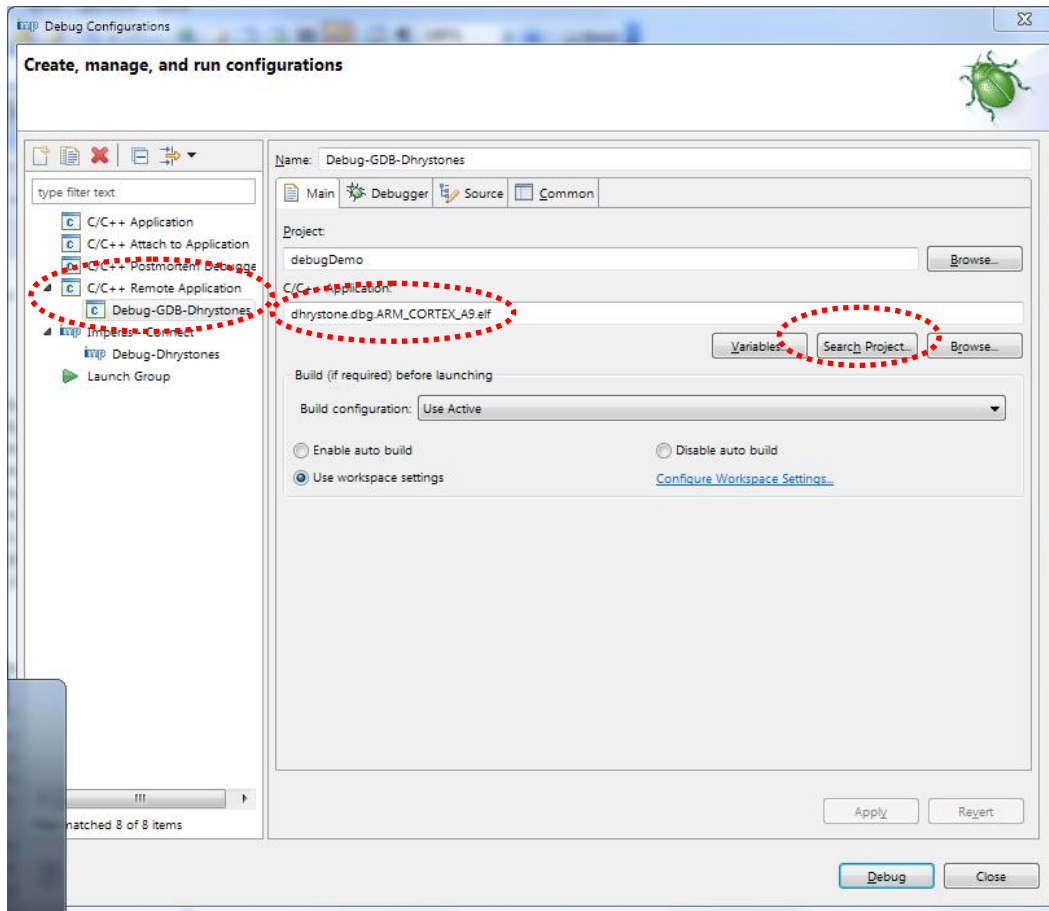
6.7.3 Create an OVPSim CDT Debug Configuration

If the OVPSim package is installed, the OVPSim simulator is used for the debug of an application running on a single processor². As such the connection used to the virtual platform is a standard CDT debug connection.

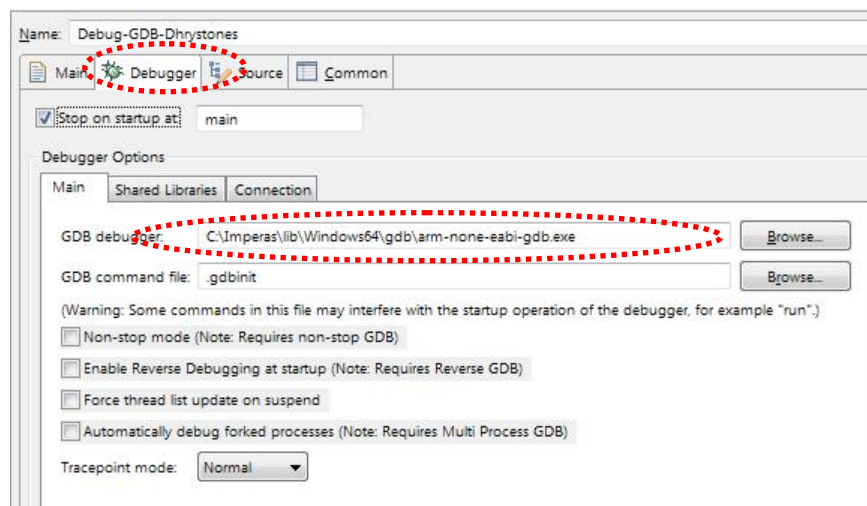
Create a debug configuration for the simulated application by selecting “Open Debug Dialog...”. Create a new “C/C++ Remote Application” configuration.

Select the application executable as the C application file using the “search project” button

² A 'single' processor can be an instance of a multicore processor, for example ARM Cortex-A15MPx4 or MIPS proAptiv MP cores.



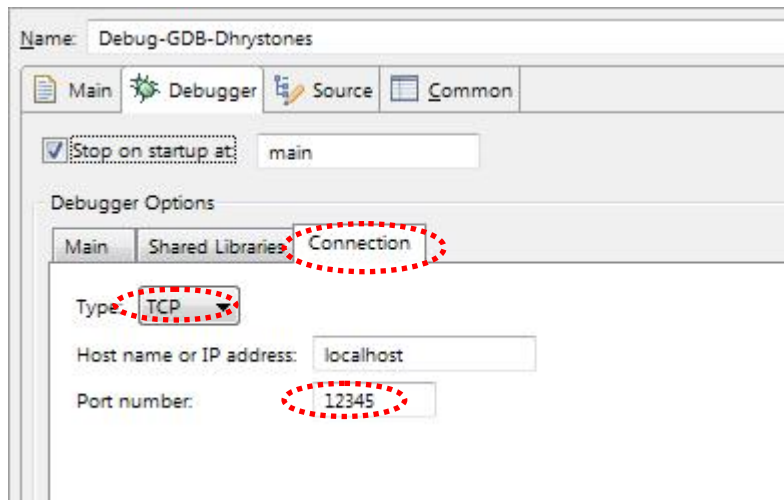
On the “Debugger” tab of this dialog browse for the GDB executable for the simulated processor type of the processor we wish to debug on



Instead of using the full path to the fixed location, for example on Windows the default installation location is C:\Imperas; it may instead be replaced with the environment variable IMPERAS_HOME.

In Eclipse the environment variable is accessed using the `${env_var:IMPERAS_HOME}` construct, as shown.

Finally, specify the target connection settings:



Select “TCP” as the connection type and choose the port number that we have chosen to use in the platform launch script; here we have initially chosen 12345. You may need to change this if there is a conflict on your host.

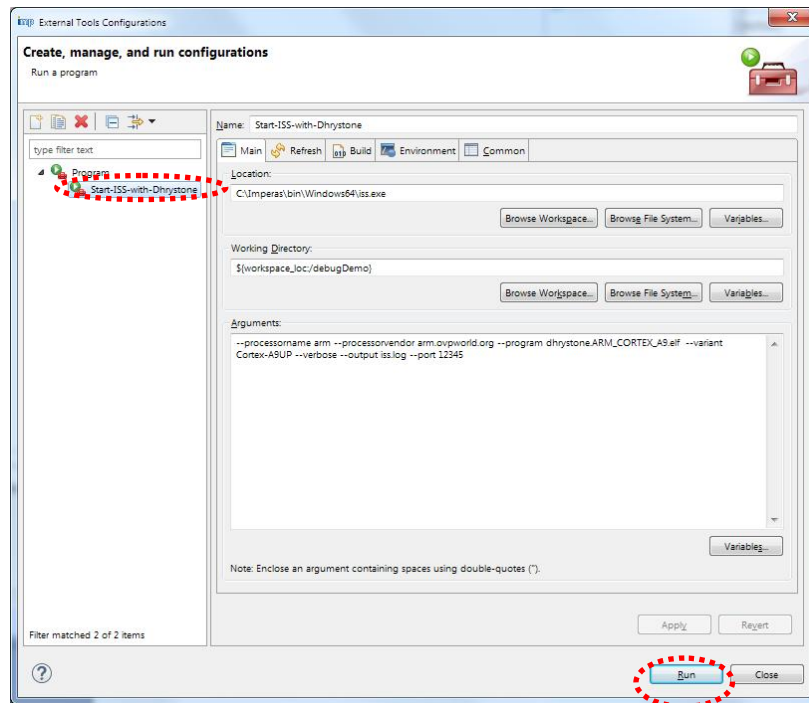
Press “Apply” then “Close” to save the debug configuration without starting it.

6.7.4 Starting to Debug using eGui

Earlier sections showed how to create a debug configuration for a simulated application and an external tool shortcut to launch a platform executable. To start the debug session, first we must launch the platform by running the external tool shortcut. The easiest way to do this is to press the external tool icon on the Eclipse toolbar.

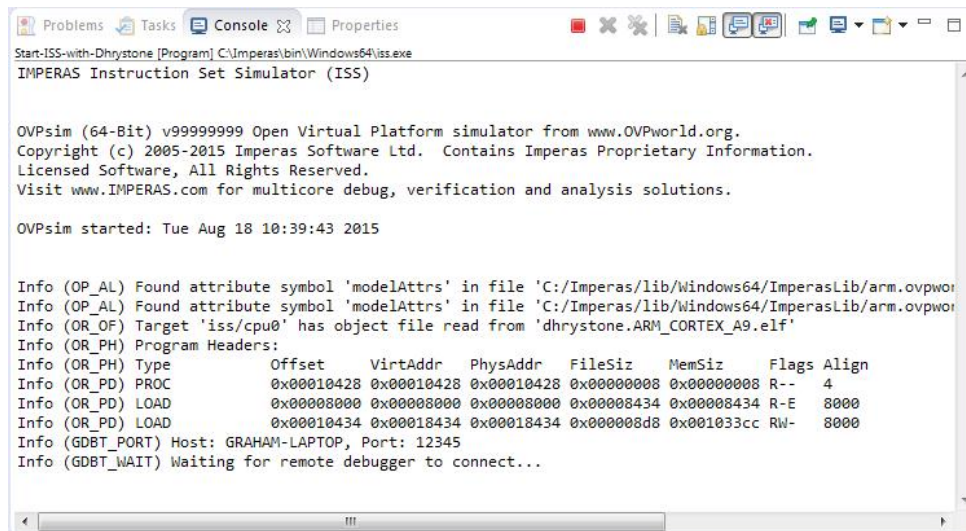


The first time you press this icon you will get the management screen in which you need to select the platform from the list in the external tool dialog, and then press the “Run” button.



Subsequently it will automatically re-run the external tool you ran last.

The Eclipse Console view will show the output from the platform executable as it starts and opens a port to await a debug connection:

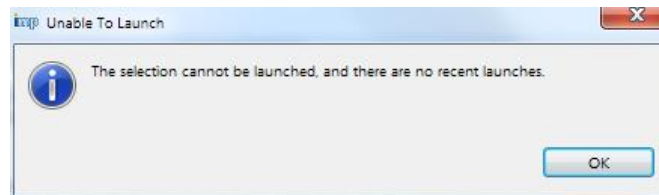


If port 12345 is already in use on your system the console will show an error message and you will need to modify both the debug configuration and the external tool configuration to select a new port number.

Once the OVPSim platform is running and waiting for a debugger to connect, launch the debug configuration we created earlier. Again, the easiest way to do this is to press the debug icon on the Eclipse toolbar.

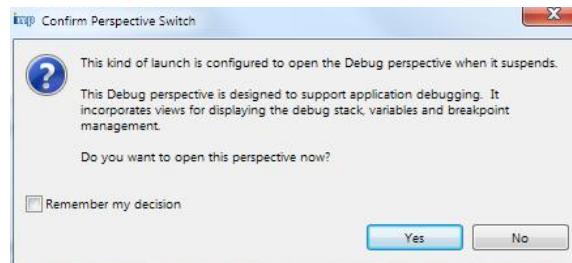


If you press this icon without having previously executed the debug configuration, you will get an error message

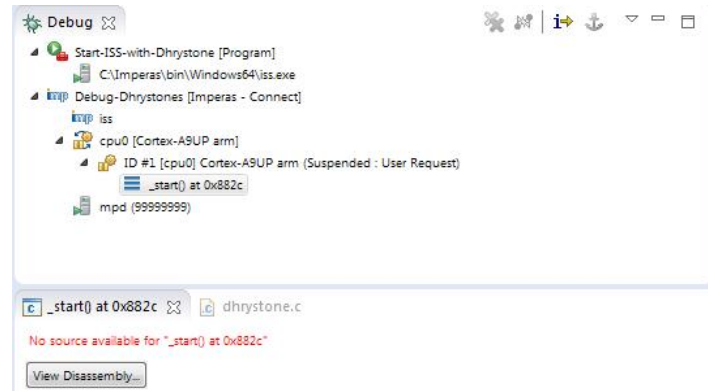


You must select 'Debug Configurations' and choose the appropriate Imperas connect configuration, then press the “Debug” button.

If debugging is configured and started correctly, Eclipse will switch (or prompt to switch) to the debug perspective, with the simulated application stopped at the entry to its main function.

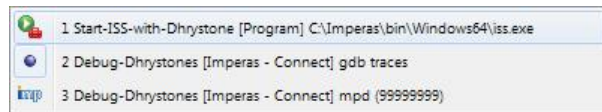


Select 'Remember my decision' so that the debug perspective is automatically switched next time. This will leave us in the debug perspective stopped at the reset or entry point.

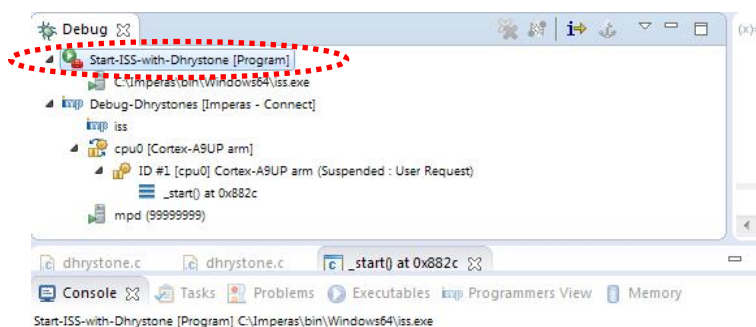


6.7.5 Switching Console Output

The output from the console terminal window depends upon the selection made in the Debug window or by using the console select dialogue, shown.



Here we have the output from the platform executing after selecting the platform executable.

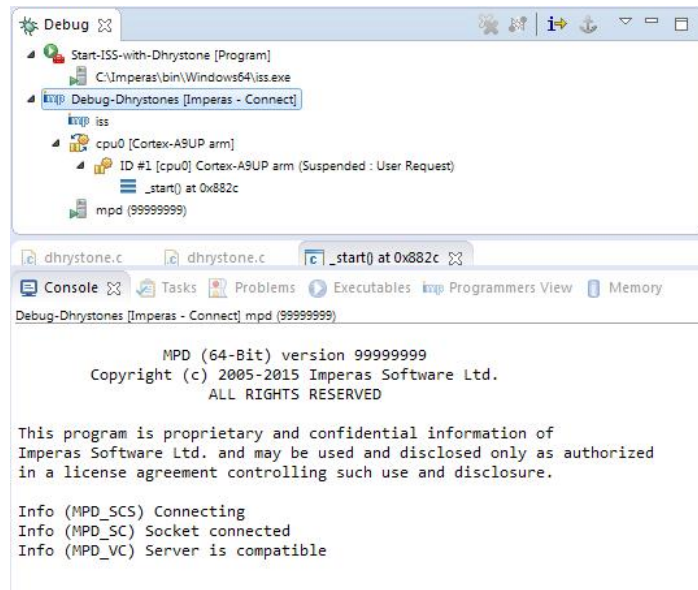


```
CpuManagerMulti (64-Bit) v99999999 Open Virtual Platform simulator from www.IMP
Copyright (c) 2005-2015 Imperas Software Ltd. Contains Imperas Proprietary Inf
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.
```

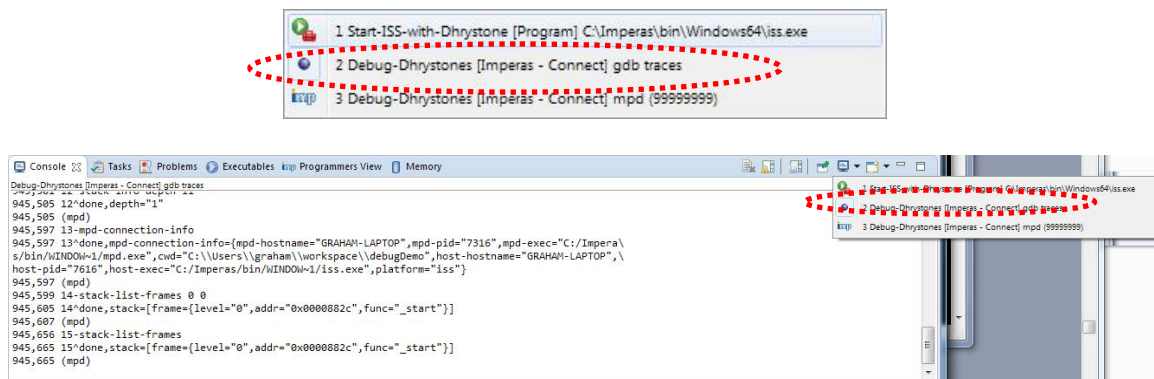
```
CpuManagerMulti started: Tue Aug 18 13:45:43 2015
```

```
Info (OP_AL) Found attribute symbol 'modelAttrs' in file 'C:/Imperas/lib/Window
Info (OP_AL) Found attribute symbol 'modelAttrs' in file 'C:/Imperas/lib/Window
Info (OR_OF) Target 'iss/cpu0' has object file read from 'dhrystone.ARM_CORTEX_
Info (OR_PH) Program Headers:
Info (OR_PH) Type      Offset      VirtAddr  PhysAddr  FileSiz  MemSiz
Info (OR_PD) PROC      0x00010428 0x00010428 0x00010428 0x00000008 0x000000
Info (OR_PD) LOAD      0x00008000 0x00008000 0x00008000 0x00008434 0x000008
Info (OR_PD) LOAD      0x00010434 0x00018434 0x00018434 0x00008d8 0x000103
Info (GDBT_PORT) Host: GRAHAM-LAPTOP, Port: 12345
Info (GDBT_WAIT) Waiting for remote debugger to connect...
Info (GDBT_MPD) Client connected to platform
```

This is the output from the debugger interface; this is actually an interactive interface.



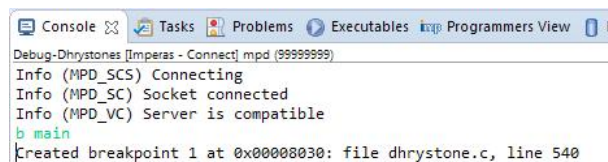
In the next pictures we see the GDB output when we select the 'gdb traces' console from the console dialogue. This shows the commands passed between Eclipse and the debugger.



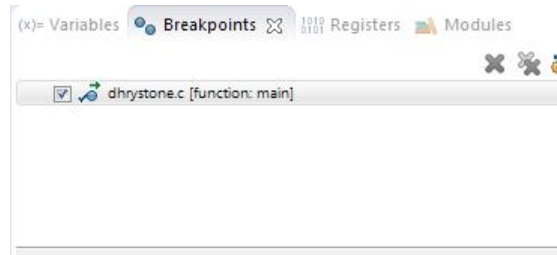
6.7.6 Setting breakpoints without source

The MPD console may be used to set breakpoints or interact directly with the Imperas debugger; including enabling and controlling the Imperas professional tools.

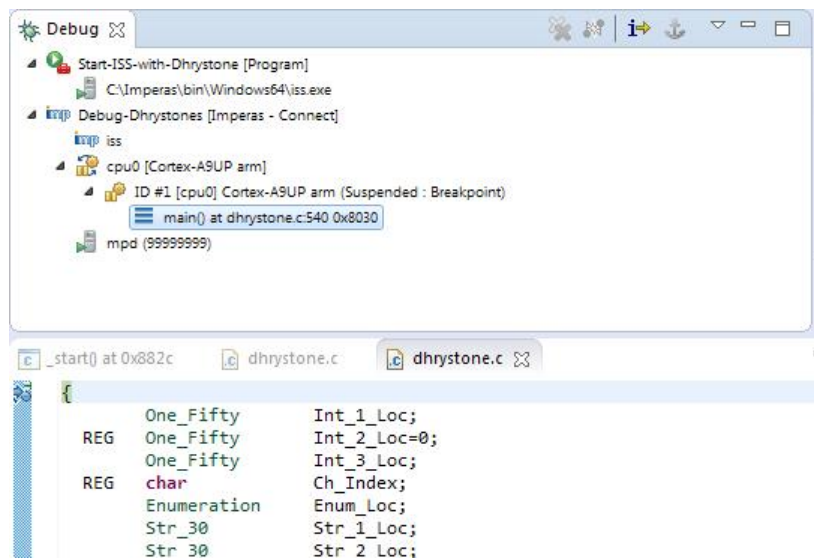
We can set a breakpoint on main by typing 'break main' or 'b main' in the console



The debugger responds with confirmation that the breakpoint has been set. It is also seen in the Eclipse breakpoint window

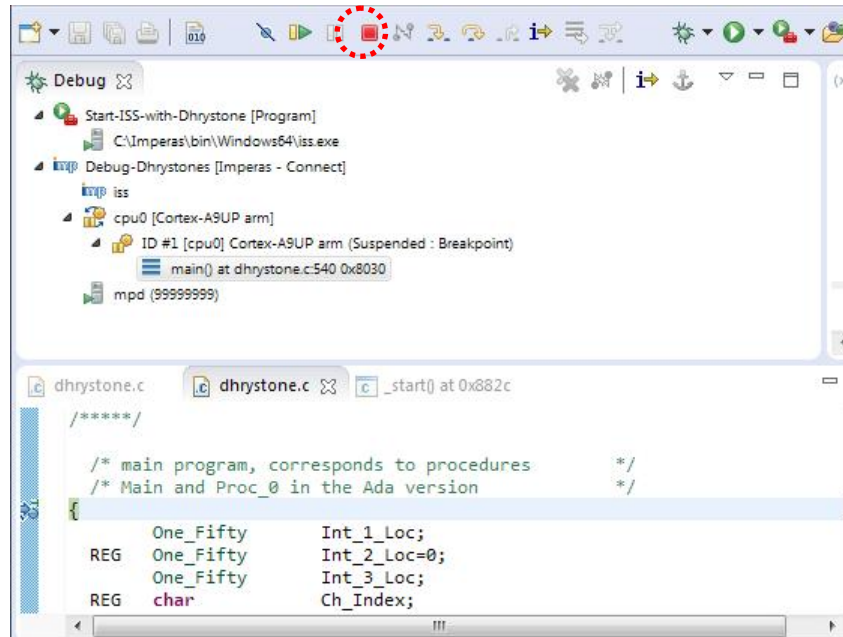


We can now execute code to this breakpoint using the run icon. Once we reach the breakpoint execution terminates and the source is displayed.

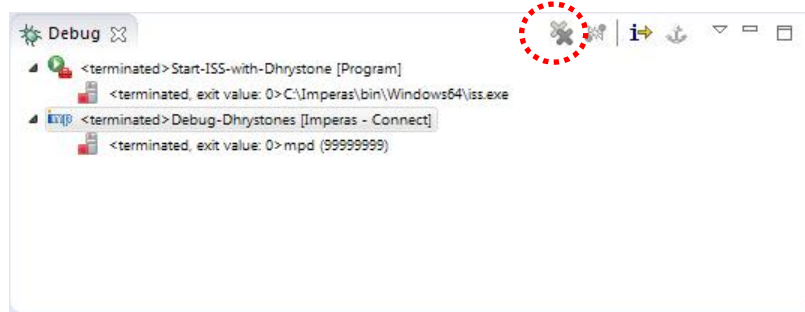


6.7.7 Terminating the Debug Session

Press the stop icon to stop debugging, then the “C/C++” perspective icon to return to the normal C project editing perspective.

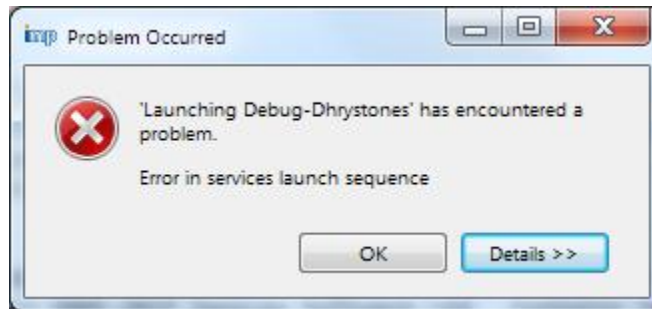


There will be a number of terminated processes shown in the debug session window that can be cleaned up using the icon shown.



6.7.8 Failing to launch correctly

If the debugger is unable to connect to the OVPsim platform you will see an error dialog.



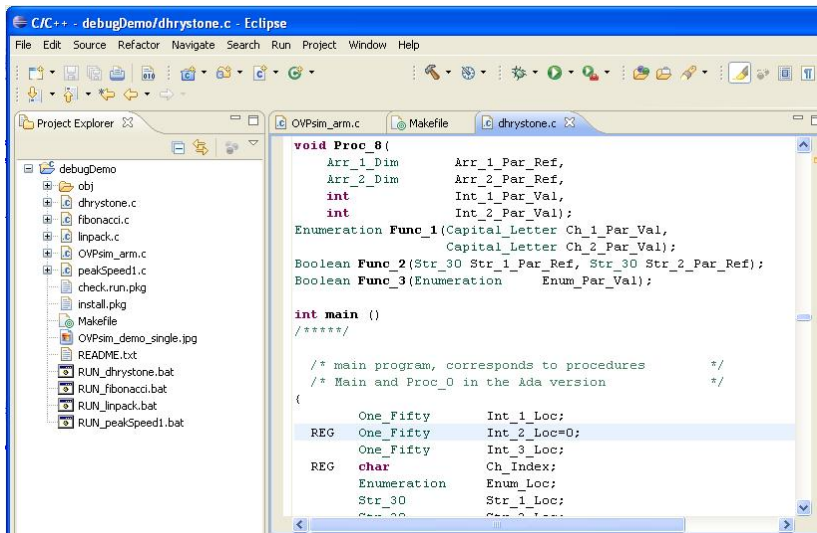
Check that the platform is running and waiting for a connection, and that the port number it is waiting for (section 6.6.2.1) matches the port number in the debug configuration settings (section 6.7.2).

By opening the Details tab further information about the problem is reported.

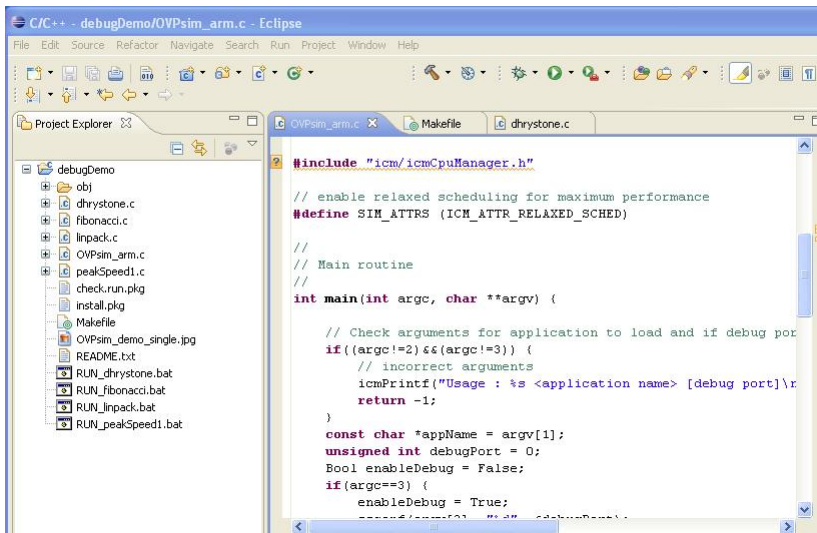
7 An Example Debug Session

Section 6.3 showed how to prepare an Eclipse project to debug an application running on a processor in an OVPsim virtual platform. This section provides a quick overview of some common debugging tasks in Eclipse. For a full explanation of Eclipse C/C++ debugging please see the C/C++ Development User Guide available in the Eclipse help system or online at www.eclipse.org.

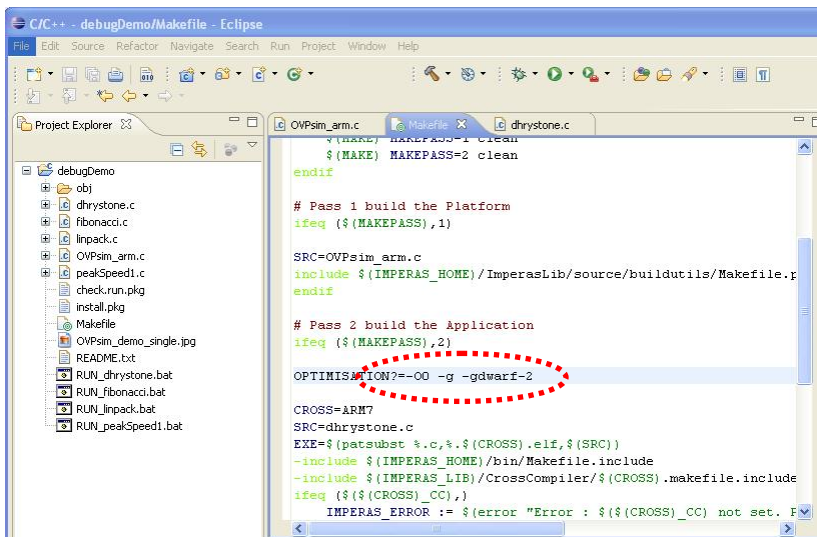
The application we will look at is the Dhrystone benchmark code.



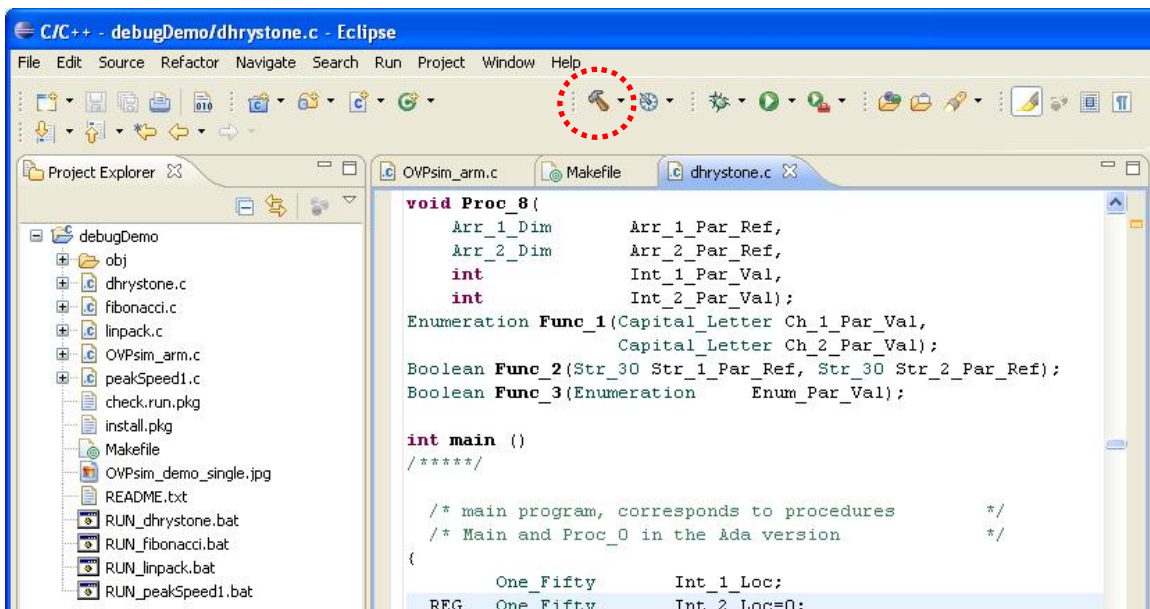
With a single processor platform setup to receive a port number for opening the debug connection.



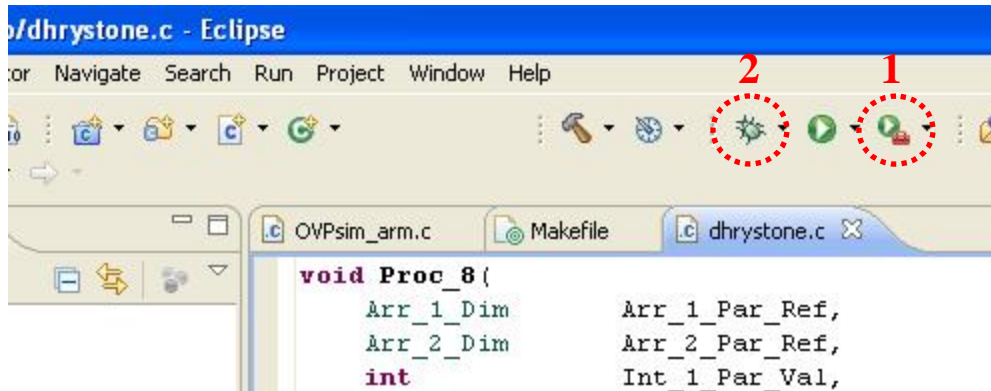
We also need to ensure that the external Makefile that is being used is setup to build an application elf file containing debug information.



After any editing in the project of the application or platform, rebuild by pressing the build icon on the Eclipse toolbar. Any syntax errors will be reported in the console view and highlighted in the editor window.



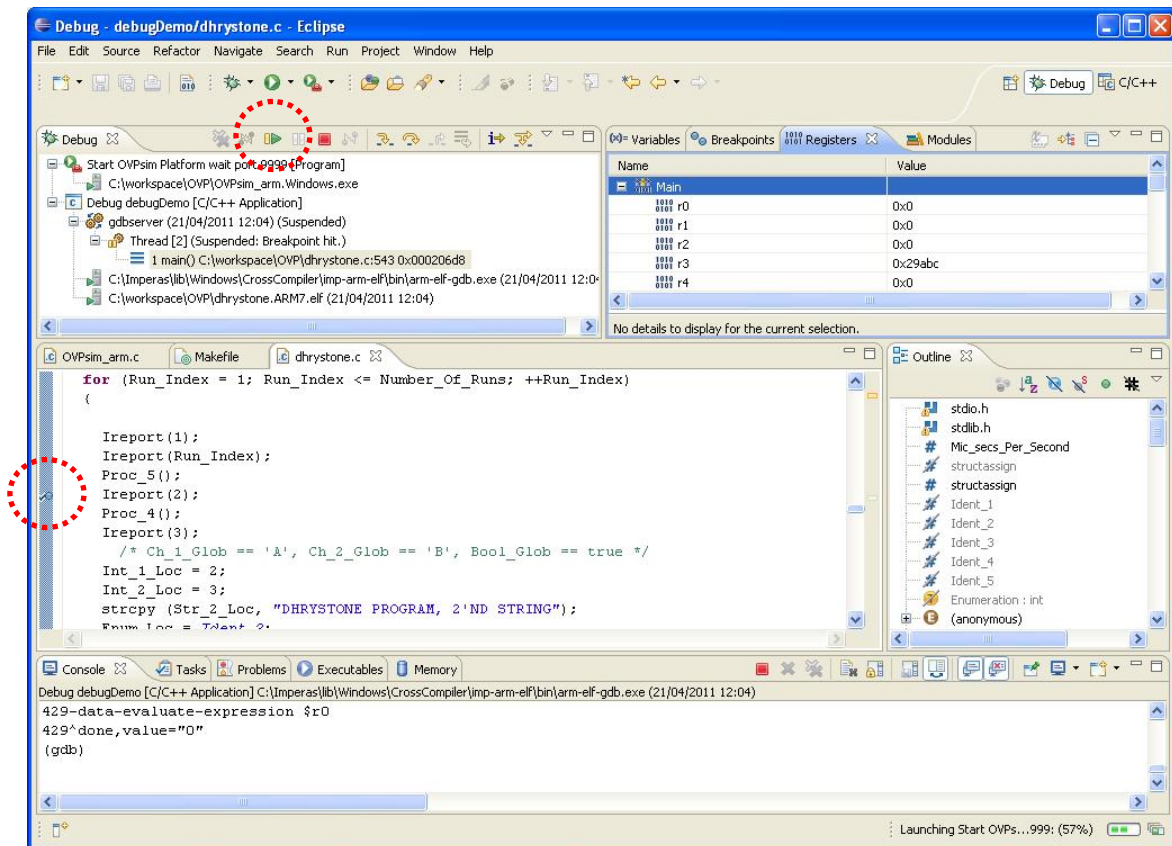
Start debugging by clicking the external tool toolbar button (1), then the debug toolbar button (2).



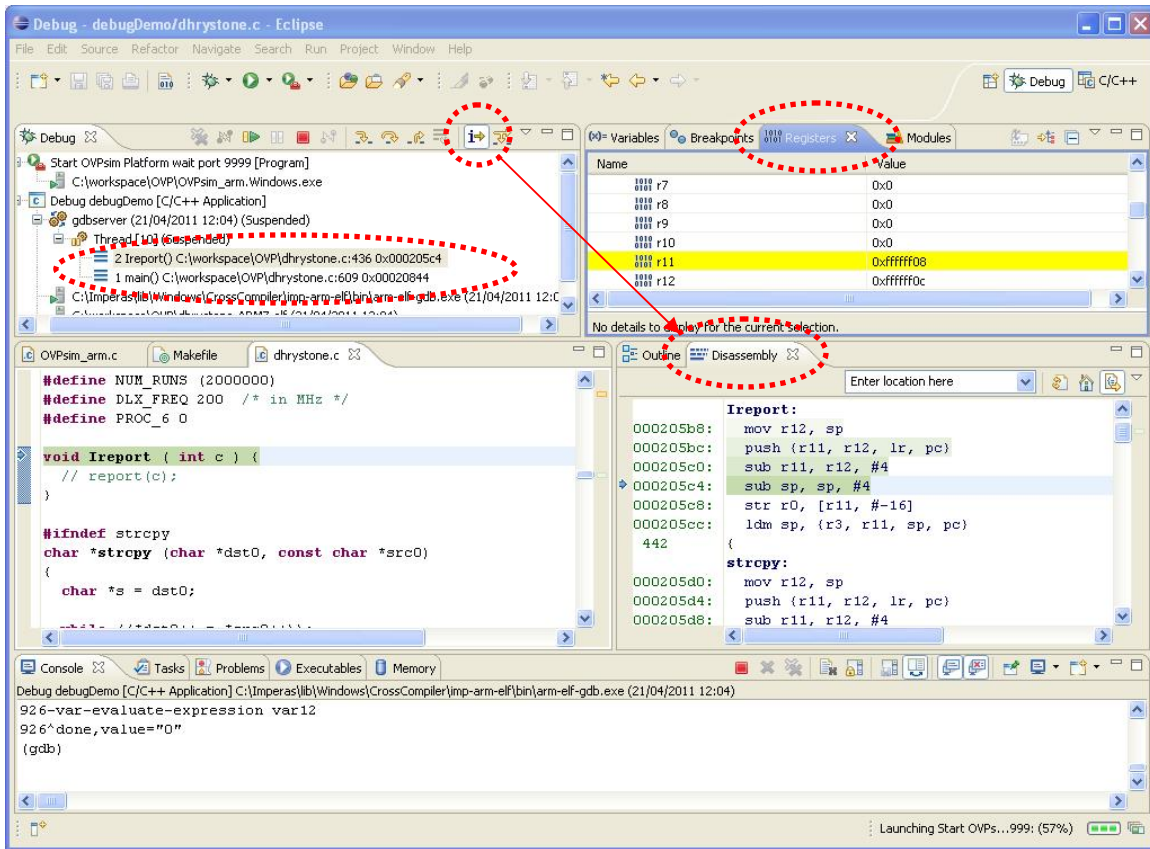
Eclipse will switch to the debug perspective automatically.

Set a breakpoint on the call to Ireport(2) by double-clicking in the margin next to the call.

Run to the breakpoint by pressing the resume toolbar button.



Show the processor registers by switching to the registers pane. Bring up the disassembly view by clicking the instruction stepping mode toolbar button. The debug view shows the call stack for the debugged application.



As you step, you will see the registers that are modified being highlighted.

Finally, run to completion by double clicking on the breakpoint to remove it, then pressing the resume button again. Program output is shown in the Eclipse platform console view.

###