Symphony Studio Eclipse for Symphony DSPs

User's Guide

Document Number: DSPSTUDIOUG
Rev. 1.1
07/2008

freescale semiconductor

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. Microsoft and Windows are registered trademarks of Microsoft Corporation.

© Freescale Semiconductor, Inc. 2008. All rights reserved.

About This Book

Use this guide to get started with cross development for the Freescale Symphony DSP Platform.

NOTE

This guide assumes development on the Windows host operating system unless otherwise stated.

Audience

This document is intended for software, hardware, and system engineers who are planning to use the product and for anyone who wants to understand more about the product.

Organization

This document contains the following chapters.

Chapter 1 "Chapter 1, "Installation," provides instructions for the initial installation and setup of the Symphony DSP Platform Tools.

Chapter 2 "Chapter 2, "Tutorials," includes tutorials to allow you to create a simple Symphony Studio C project, shows you how to build it using the development tools, and how to debug the dual-core application through Eclipse, and also allow you to create a simple Symphony Studio Assembly project, shows you how to build it using the development tools, and how to debug the dual-core application through Eclipse.

Chapter 3 "Chapter 3, "Advanced Topics," goes into further details about debugging tasks, such as reading and writing to/from registers and memory. If you are a returning user wanting only to brush-up on the debugging procedure, this section is most relevant.

Chapter 4 "Chapter 4, "Troubleshooting," goes into further details about debugging tasks, such as reading and writing to/from registers and memory. If you are a returning user wanting only to brush-up on the debugging procedure, this section is most relevant.

Chapter 1
Installation

1.1 Java Runtime Environment

The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine (JVM), and other components to run applets and applications written in the Java programming language. Since Eclipse is written in Java, a compatible JRE must first be installed before Eclipse can be used.

There are many commercial vendors that supply JREs. A list of supported JREs is usually available from the Eclipse download page at http://www.eclipse.org/downloads/. Follow the product installation instructions and complete the installation.

NOTE

The recommended release for DSP56720 development is JRE 1.5.x

NOTE

It is possible to have more than one JRE installed on your machine. To see if you have JRE installed on your system check your installed programs (For Windows this is under Start -> Settings -> Control Panel and then Add or remove Programs). As a general safeguard, it is recommended to always explicitly specify the JVM to be used for launching Eclipse. For more information on how to specify a particular JVM, see the following:

http://wiki.eclipse.org/index.php/FAQ_How_do_I_run_Eclipse%3F

1.2 Eclipse for Symphony DSPs

Eclipse is an open source community whose projects are focused on providing a vendor-neutral open development platform and application frameworks for building software, and is the chosen platform for Symphony DSP development. Symphony Studio is made up of two main components, the Eclipse platform SDK 3.2.2 and the Eclipse C/C++ Development Toolkit 3.1.1. Together these provide an industrial strength C/C++ IDE. For more information on Eclipse see http://www.eclipse.org.

Both Eclipse 3.2.2 and the Eclipse C/C++ Development Toolkit 3.1.1 are Eclipse projects but have been patched specifically for the Symphony DSPs. Therefore you are unable to download the components

from the Eclipse website. To install Symphony Studio simply execute the setup file Symphony-Studio-1.1.0-Windows-Setup.exe and follow the on-screen instructions to choose an installation path.

NOTE

A directory with spaces in the absolute path (such as My Documents or Program Files) is not the recommended install location for Eclipse due to potential problems with directory/file names with spaces. The recommended place to install Eclipse is in C:\Symphony-Studio.

1.3 Open On-Chip Debugger

Open On-Chip Debugger (openOCD) is a GDB-JTAG server, which allows the host computer to communicate with the DSP EVB. The openOCD binary comes packaged with your Symphony Studio release. We will refer to Symphony Studio installation directory as TOOLSDIR.

This section details how to install OpenOCD with a parallel port driver, or with the D2XX USB driver from Future Technology Devices International Ltd. “

See http://www.ftdichip.com/

1.3.1 Install FTDI D2XX Drivers

If you are using the SoundBite board the FTDI D2XX drivers need to be installed on your system. By default, these drivers are automatically installed with the installation of Symphony Studio. If for some reason the drivers need to be reinstalled, execute the following:

```
TOOLSDIR\dsp56720-devtools\dist\openocd\driver\CDM\CDM 2.02.04.exe
```

1.3.2 Install Parallel port Drivers

For the parallel port driver open a command prompt then:

1. Change directory to TOOLSDIR\dsp56720-devtools\dist\openocd\driver\parport
2. Run “`install_giveio.bat`”

The output should look like:

```
Copying the driver to the windows directory
target file: C:\WINDOWS\giveio.sys
      1 file(s) copied.
Remove a running service if needed...
Installing Windows NT/2k/XP driver: giveio
```

```
installing giveio from C:\WINDOWS\giveio.sys... ok.
starting giveio... ok.
set start type of giveio to auto... ok.
Success
```

NOTE

The driver can be unloaded by running `remove_giveio.bat` in the same directory.

To verify the installation run "`status_giveio.bat`"

You should see the following output:

```
status of giveio:
Type:          [0x01] Kernel driver.
Start Type:    [0x02] Automatic
Error Control:    [0x01] NORMAL: Display a message box.
Binary path:    \??\C:\WINDOWS\giveio.sys
Load order grp:
Dependencies:
Start Name:
ok.
```

Chapter 2
Tutorials

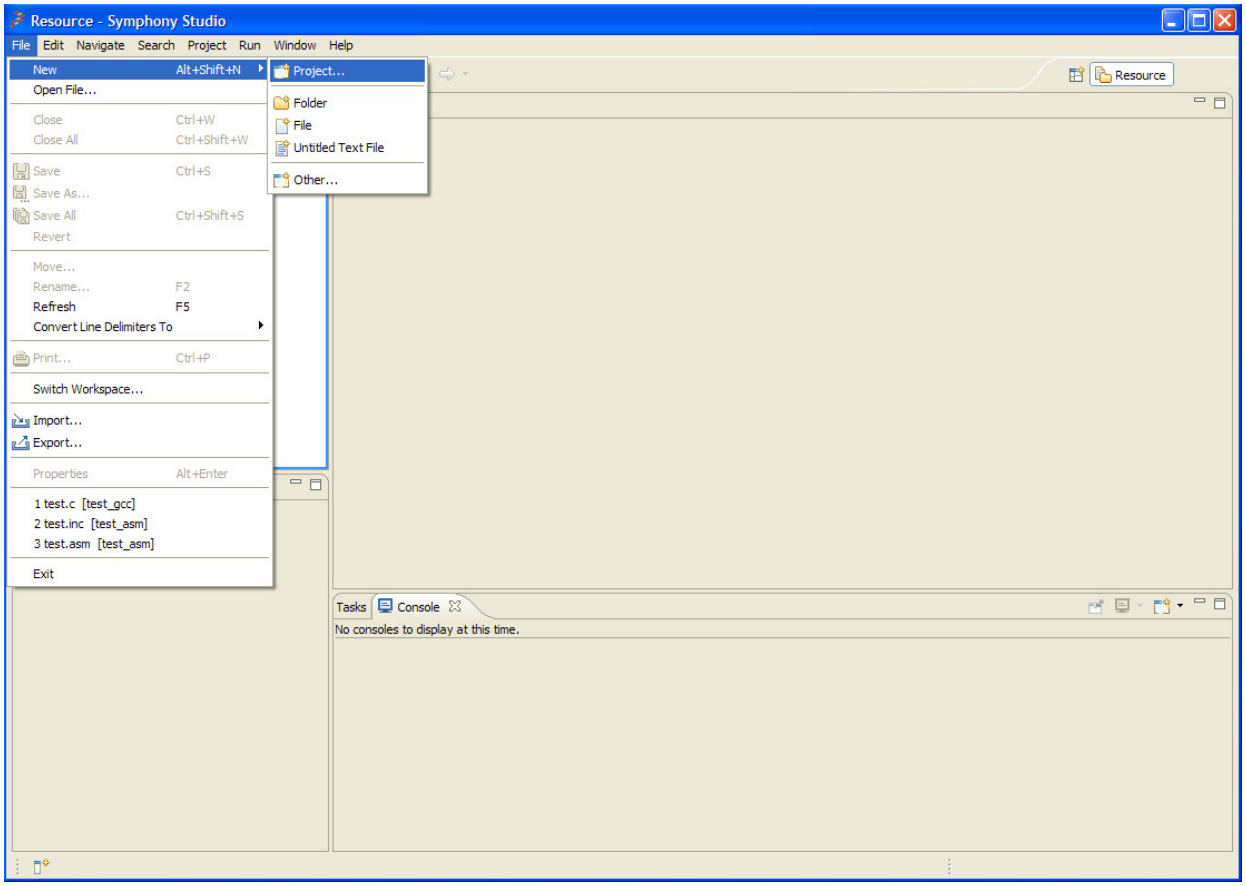
You are now ready to start developing with Symphony Studio. This chapter shows a C and assembly application compiled using the Symphony Studio Development Tools and then debugged on the DSP56720 simulator.

2.1 C Tutorial

This section describes the specific steps of the C tutorial.

2.1.1 Creating a New Project

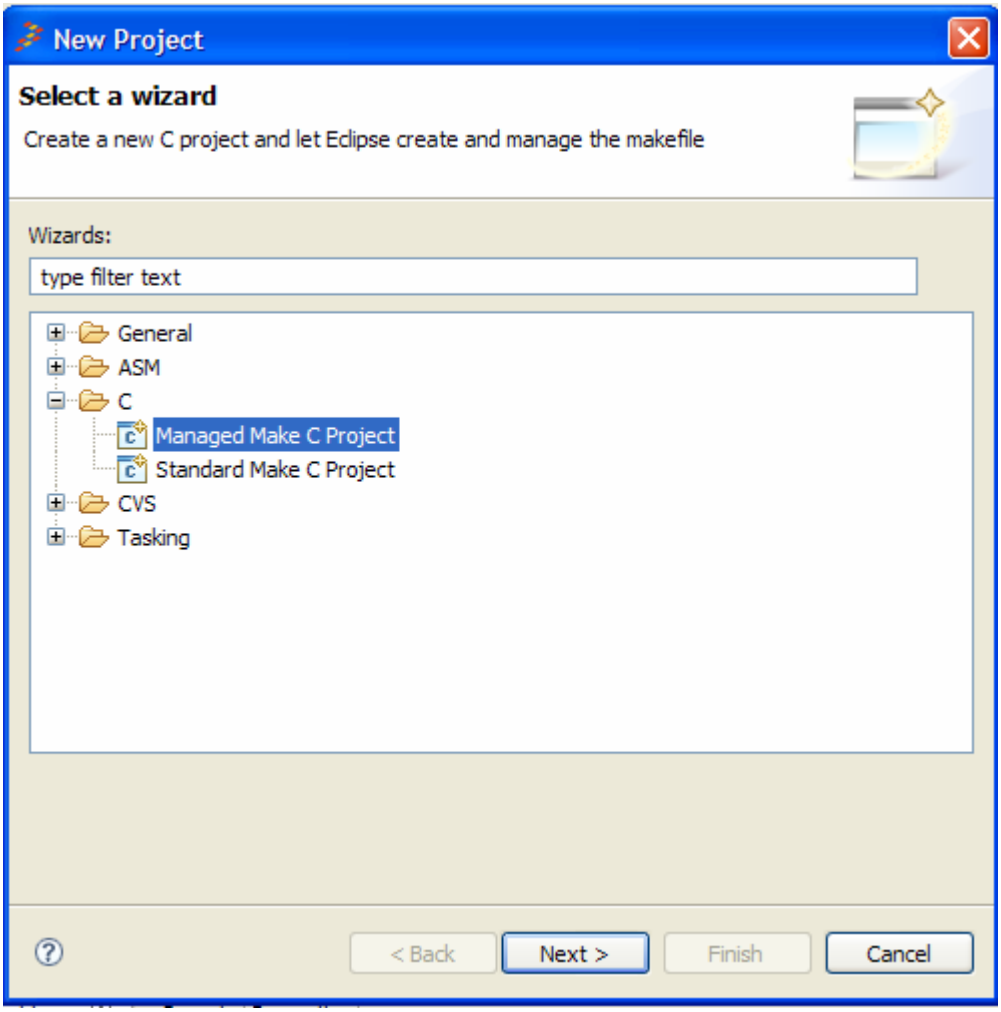
1. Select File -> New -> Project.



TIP

In the C/C++ Perspective you can quickly open a new Managed Make C Project by right click in the C/C++ Projects View and selecting New->Managed Make C Project.

2. Expand the *C* folder and select the *Managed Make C Project* option. Click *Next*.

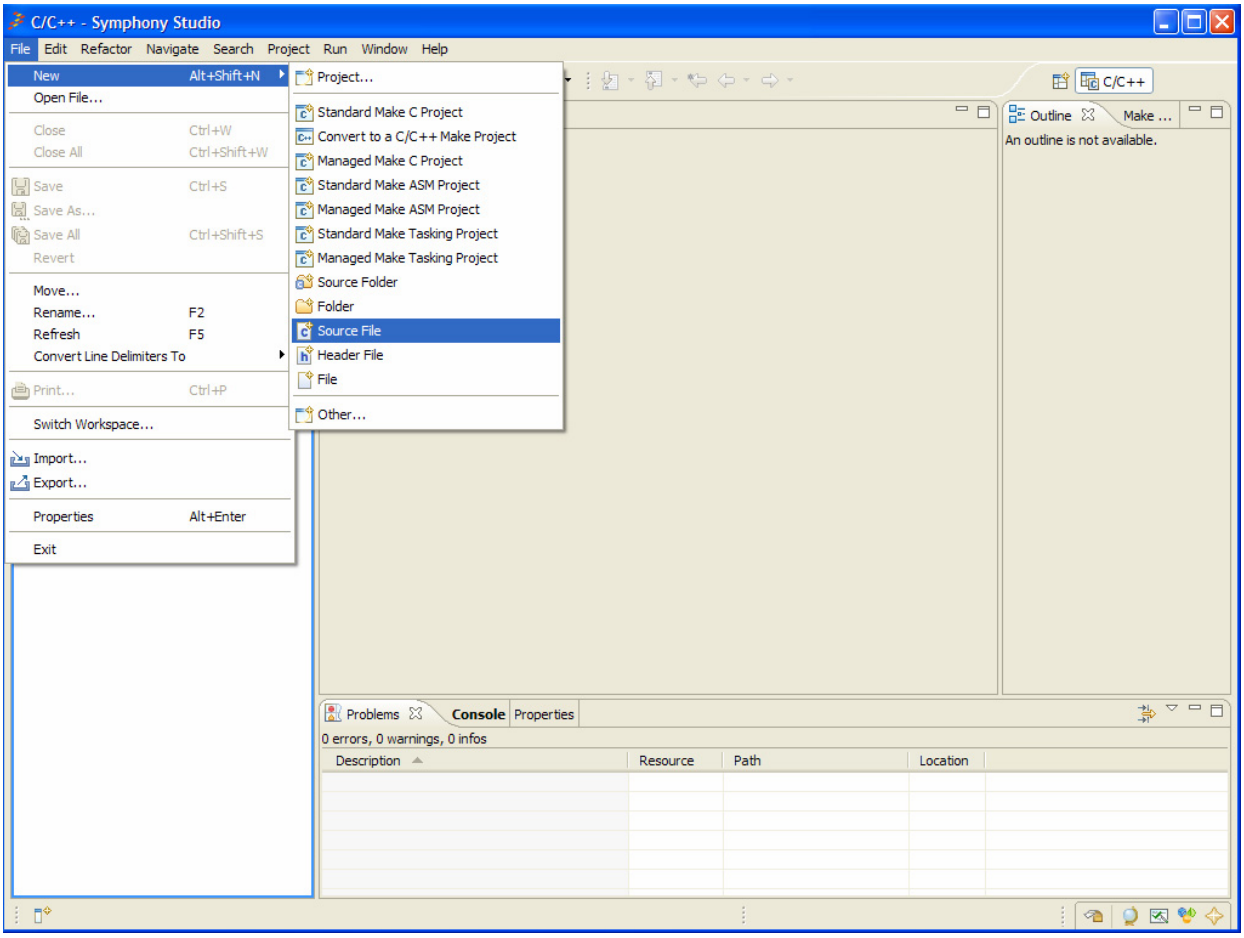


3. Select a suitable workspace directory if prompted (this is used to store all your projects)
4. Set the *Project name* to `C-Tutorial`

5. Click the *Next* button and the *Project Type* should be automatically set to `56K GCC COFF`
6. Click the *Finish* button.
7. Click *Yes* if you are offered the option to open the C/C++ perspective now. The *C-Tutorial* project appears in the *C/C++ Projects* tab.

2.1.2 Adding Source Files

1. Right-click on the *C-Tutorial* project in the *C/C++ Projects* tab and select **New -> Source File**.



2. Use the following settings:
   - Source Folder: `C-Tutorial` (default)
   - Source File: `tutorial.c`

3. The empty *tutorial.c* file appears under the *C-Tutorial* project and is automatically opened by Symphony Studio for editing. Note that an error is reported initially as it tries to compile the empty file into a CLD file.

4. Add the following C source lines to the *tutorial.c* file:

```
/* Data Types */
typedef struct customtype
{
      int       val1;
      long int  val2;
      long int  val3;
} ctype;


/* Function Prototypes */
int       func1(int arg1, int arg2);
long int  func2(int arg1, int arg2);
void      func3(ctype *arg1);

/* Function Definitions */
int main(void)
{
      int       a = func1(7, 3);
      long int  b = func2(a, 19);
      long int  c = a * b;
      ctype     d;

      d.val1 = a;
      d.val2 = c;
      d.val3 = 0;

      func3(&d);

      if(d.val3 == b)
            return 1;

      return 0;
}

int func1(int arg1, int arg2)
{
      int rv = func2(arg1, arg2);
      rv += arg1;
      return rv;
}
```

```
long int func2(int arg1, int arg2)
{
      return arg1 - arg2;
}

void func3(ctype *arg1)
{
      arg1->val3 = arg1->val2 / arg1->val1;
}
```

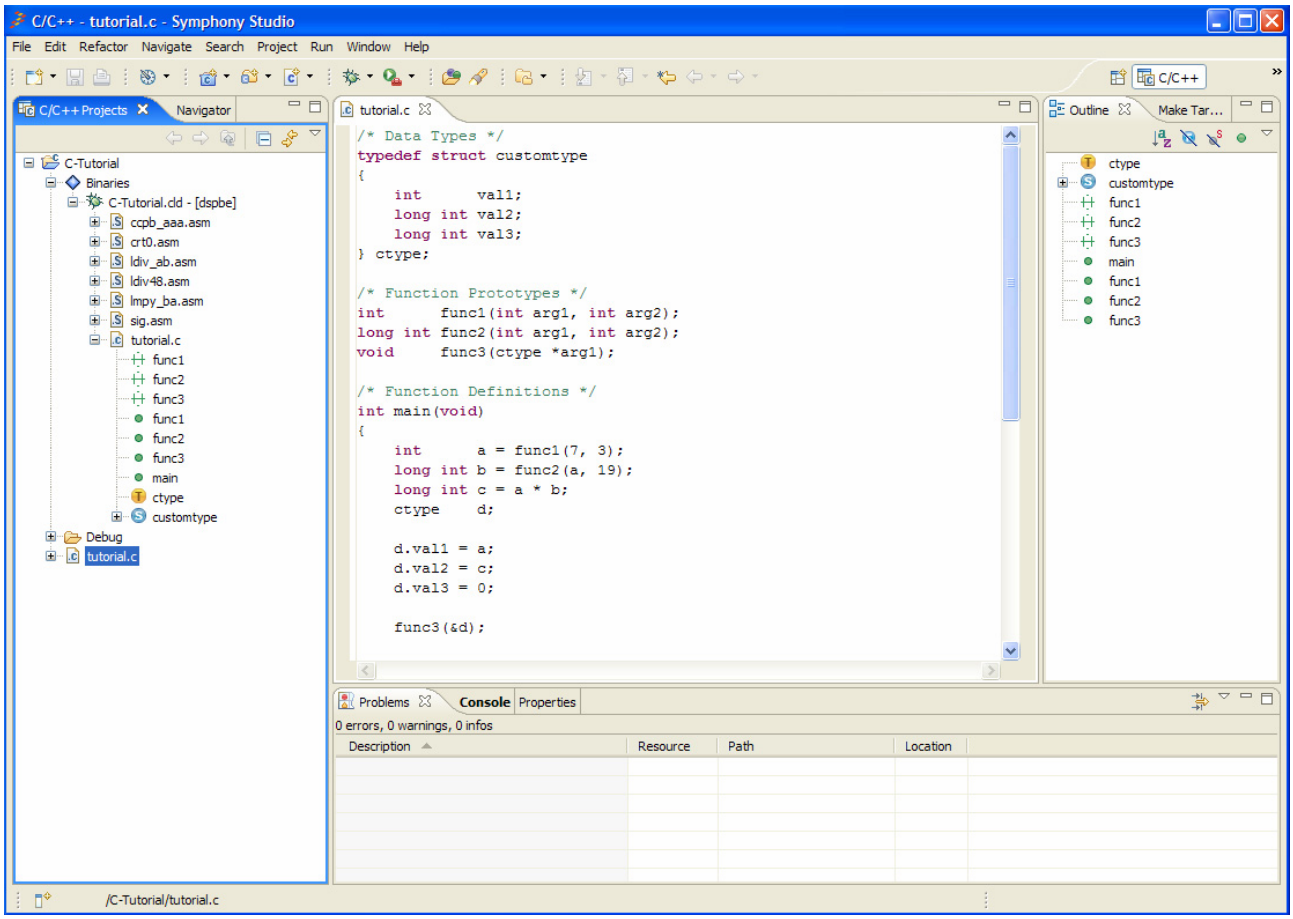
5. Save your work by clicking the *Save* button. Symphony Studio automatically compiles the project when you do this and the earlier reported error should not be present anymore.

NOTE

If you want to disable automatic building, uncheck **Project -> Build Automatically**. To clean your project, select **Project -> Clean**, select the projects you want to clean and hit the **OK** button.

TIP

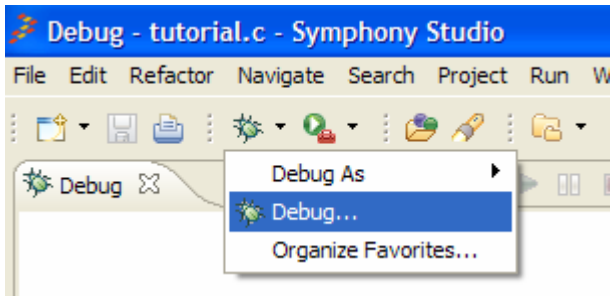
Symphony Studio includes a Freescale DSP COFF Binary parser and identifies all valid COFF files in a Binaries tree element in the project folder. You can expand the COFF file to see which source files it references.



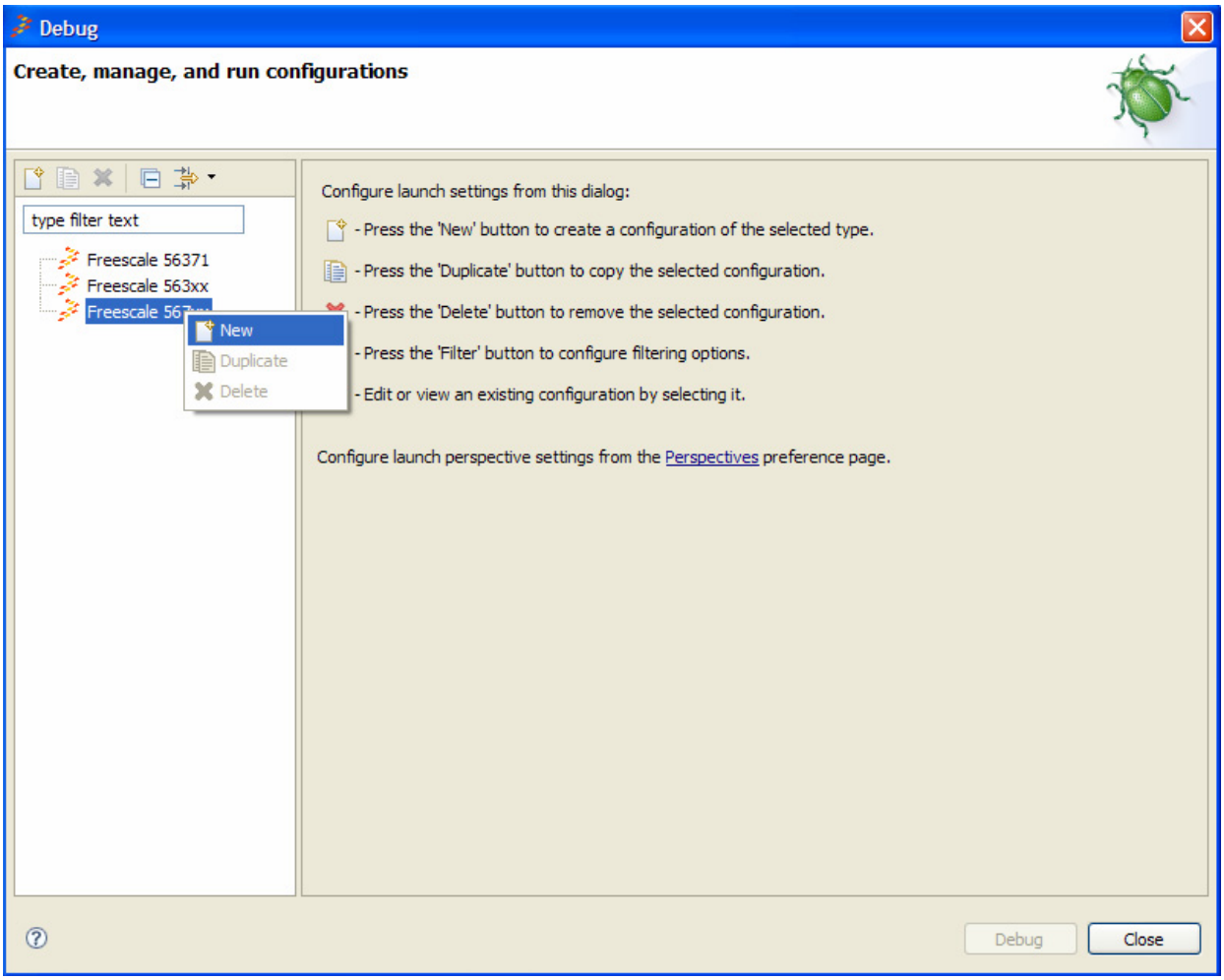
2.1.3 Creating a Debug Configuration

You will now create two debug configurations (one for each DSP56720 core).

1. Switch to the *Debug* perspective by selecting **Window -> Open Perspective -> Other** and choosing the **Debug** option.
2. Select **Run->Debug**. Alternately you can use the debug pull down menu in the tool bar.



3. Right click on the *Freescale 567xx* target and select **New**.

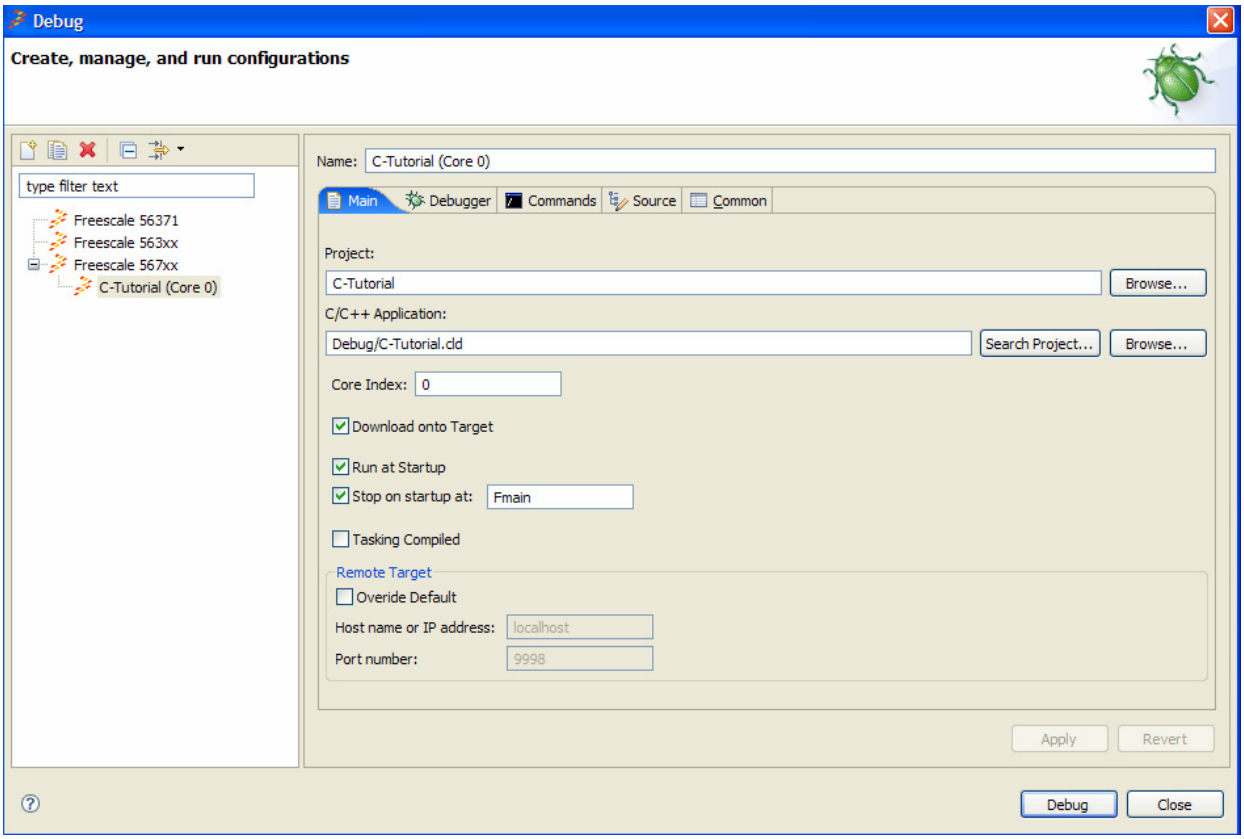


4. In the *Name* field change the text so it reads: `C-Tutorial (Core0)`

5. The Project field should currently be pointing to *C-Tutorial*. Click the *Search project* button on the C/C++ Application field and select the *C-Tutorial.cld* file that should be listed there. Set the Core Index to 0 if it isn't already.

NOTE

If you want to debug a CLD file that is not part of the current project you can use the **Browse** button, which lets you select any CLD file available on your computer.



Repeat steps 1-5 to create a debug target for Core1 (changing the name and Core Index appropriately).

More details on the other options available for the Freescale 56xxx debugger target are documented in Section 3.9, “Freescale 56xx Debug Target Options.”

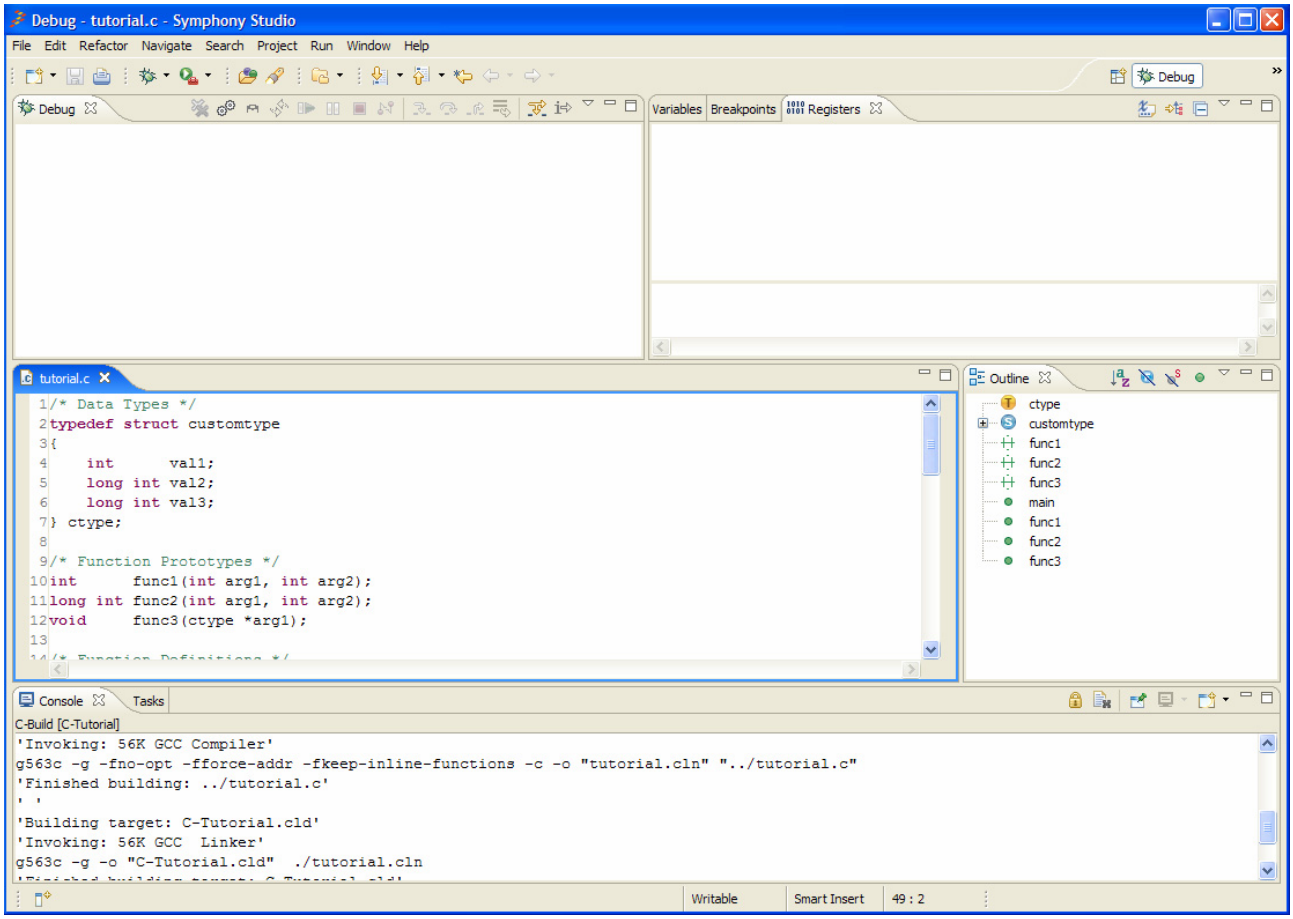
2.1.4 Debugging your Application

You are now ready to debug your application running on the DSP56720 simulator. Following is a summary of what we have achieved thus far:

- Written an application for the DSP56720 and built it using the integrated DSP compiler and tools
- Created two debug configurations (one for each core inside the DSP56720 device) to debug the application code running on inside each core

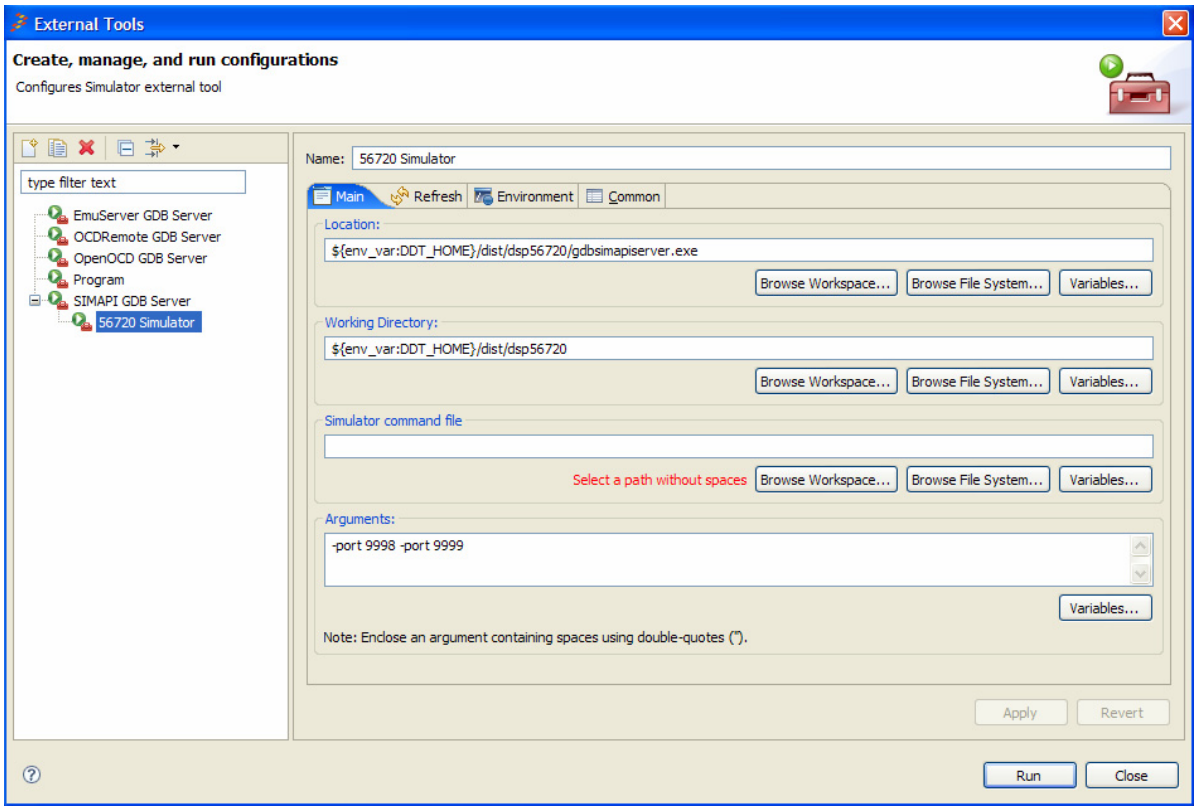
To debug, follow these steps:

1. Select **Window -> Open Perspective -> Debug** to select the debug perspective. Resize your panes as appropriate. You will want a reasonable sized main window for debugging the source code.
2. Select **Window -> Show View -> Registers** to open the registers window.
3. Right-click in the column to the left of the source lines in the source window and select **Show Line Numbers**.

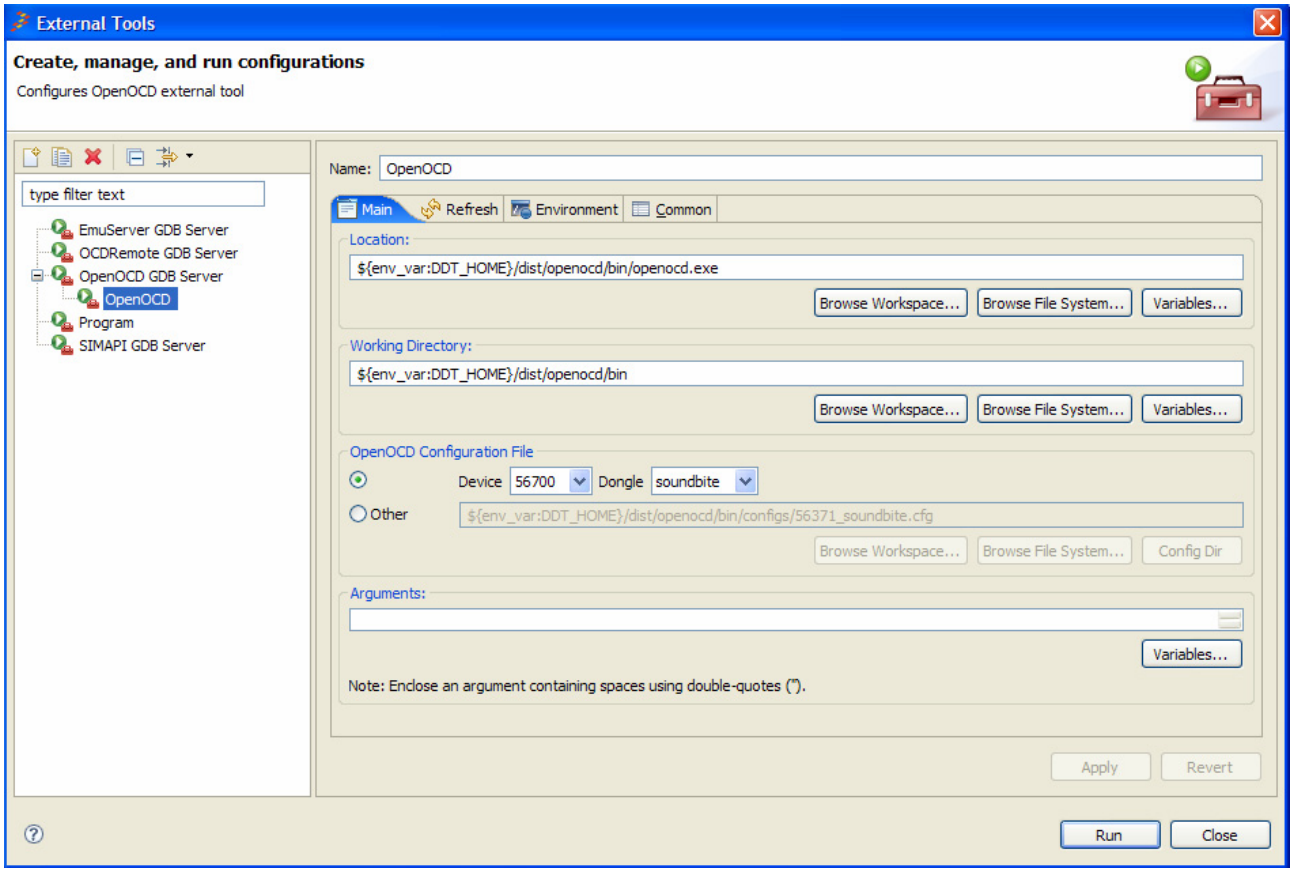


4. Launch the *GDB SIMAPI Server* to debug via the simulator or the *GDB OpenOCD Server* to debug via the EVB.

a) To launch the SIMAPI GDB Server select **Run -> External Tools -> External Tools**. Right-click *SIMAPI GDB Server* in the side pane to create a new configuration. Use the default values and **Run**

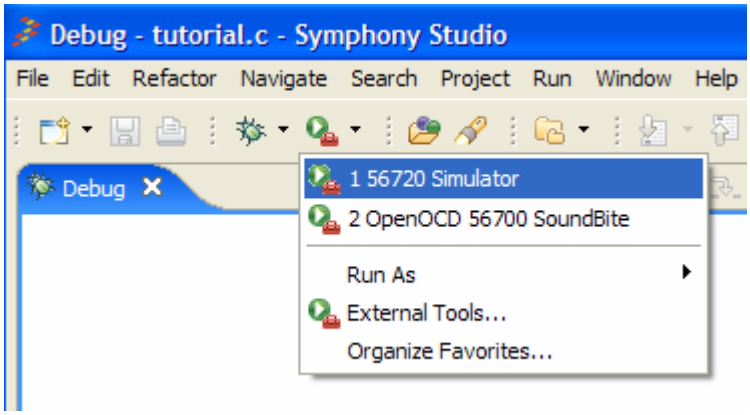


b) To launch the OpenOCD Server select **Run -> External Tools -> External Tools**. Right-click on OpenOCD GDB Server in the side pane to create a new configuration. In the "OpenOCD Configuration File" group select the appropriate debug dongle and DSP. Otherwise the user can specify the exact location of the OpenOCD Configuration file by selecting other. They can browse the current workspace. Browse the file system if the file is outside of the workspace, or use one of the in build Eclipse variables.Use the default values for the remaining fields and **Run**.

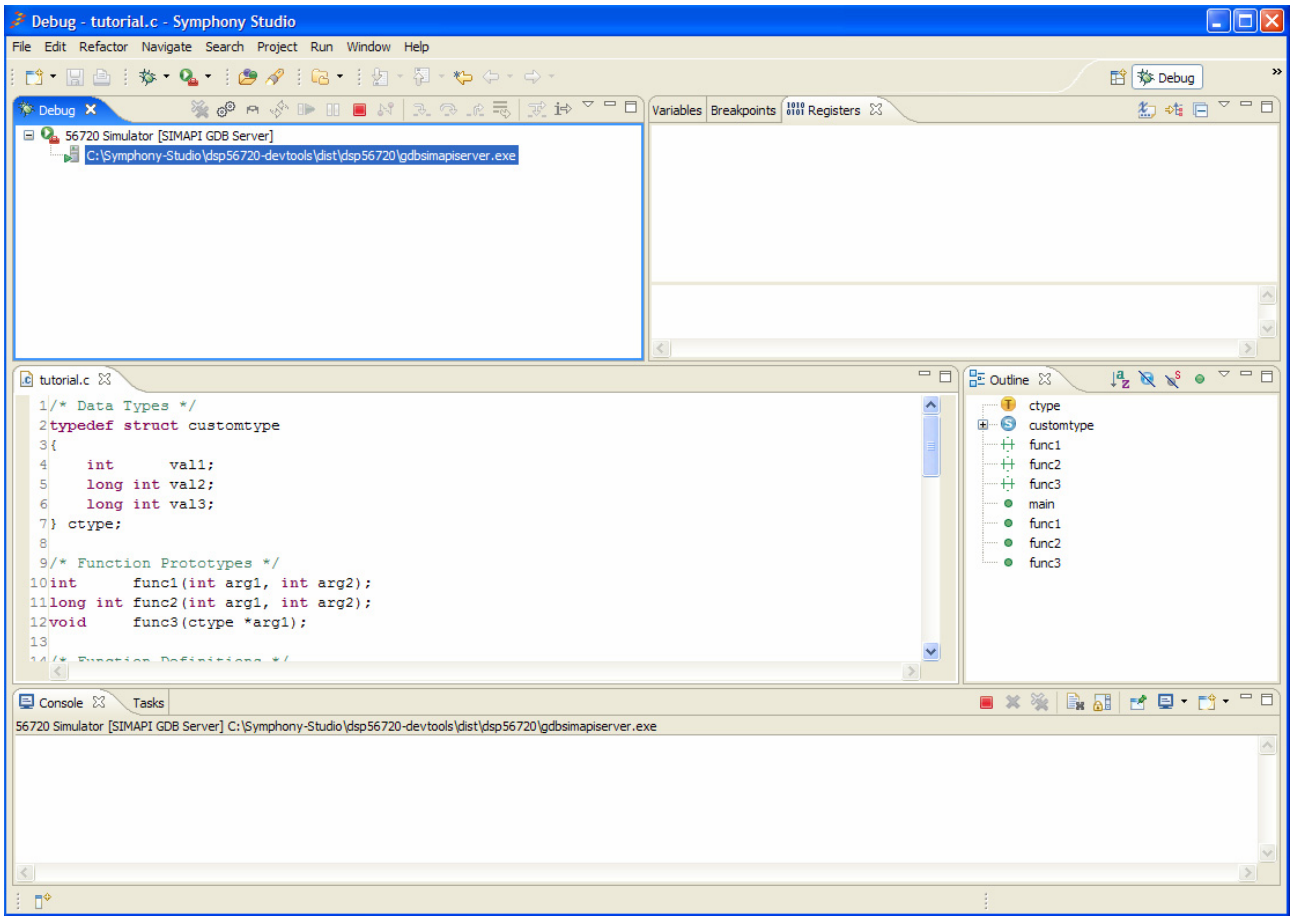


TIP

Once an External tool is created and launched you can quickly relaunch it by selecting it in the External Tools pull down menu in the tool bar.



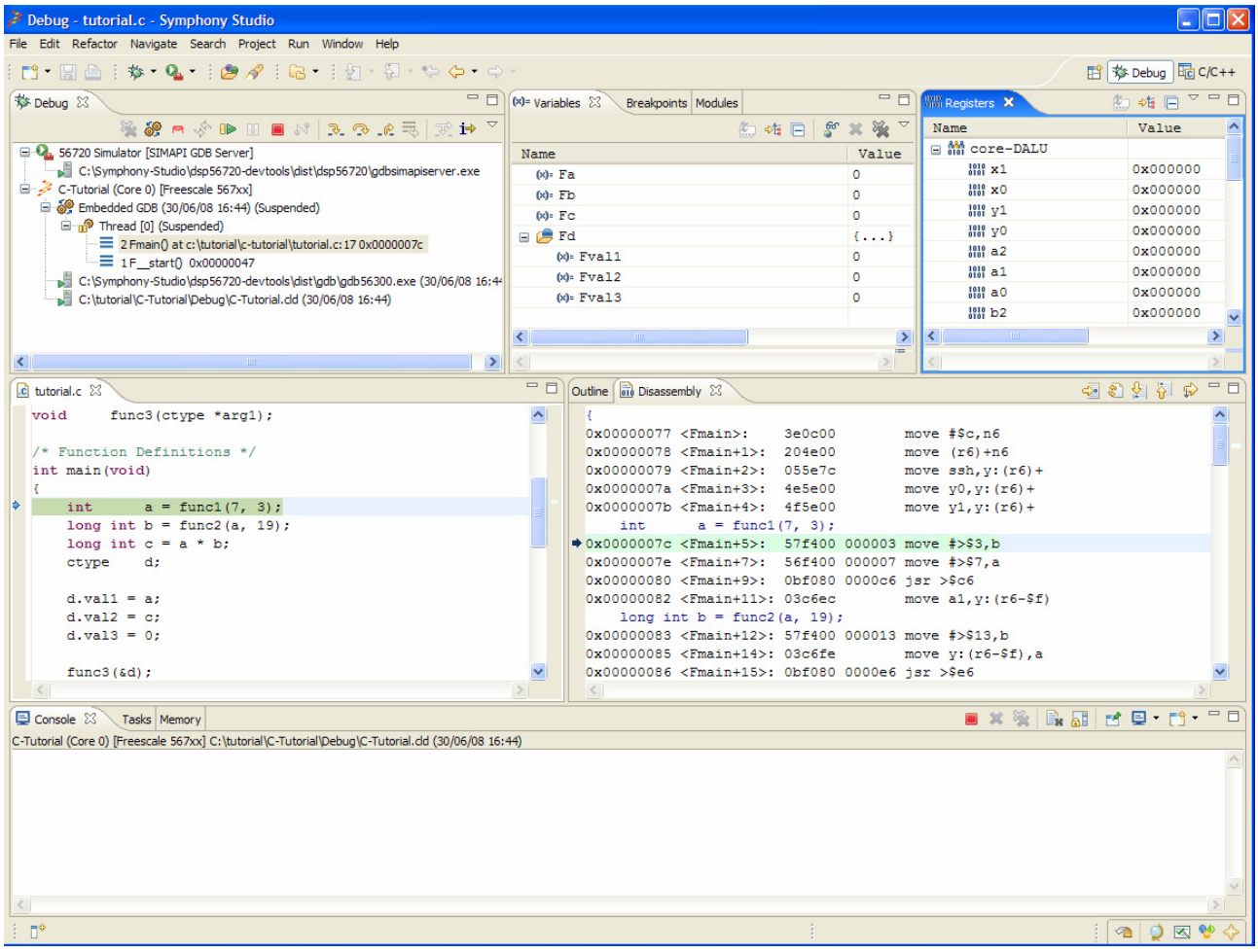
You should now have something that looks like the window below:



5. Launch GDB by selecting **Run -> Debug**, then selecting *C-Tutorial (Core0)* in the side pane and hitting **Debug**. Behind the scenes, GDB makes a TCP connection to the port on which the

GDB SIMAPI Server is listening. It then instructs the server to load the application binary `C-Tutorial.cld` into the connected DSP56720 device (Core0).

6. The simulator runs till it hits the start of *main()* then stops (If you get a debug window with a message like *source code not found*, please see Chapter 4, "Troubleshooting," for details on how to fix it). You can now open the disassembly window by Selecting **Window -> Show View -> Disassembly**. Use the **Step Into** icon to step through the code.

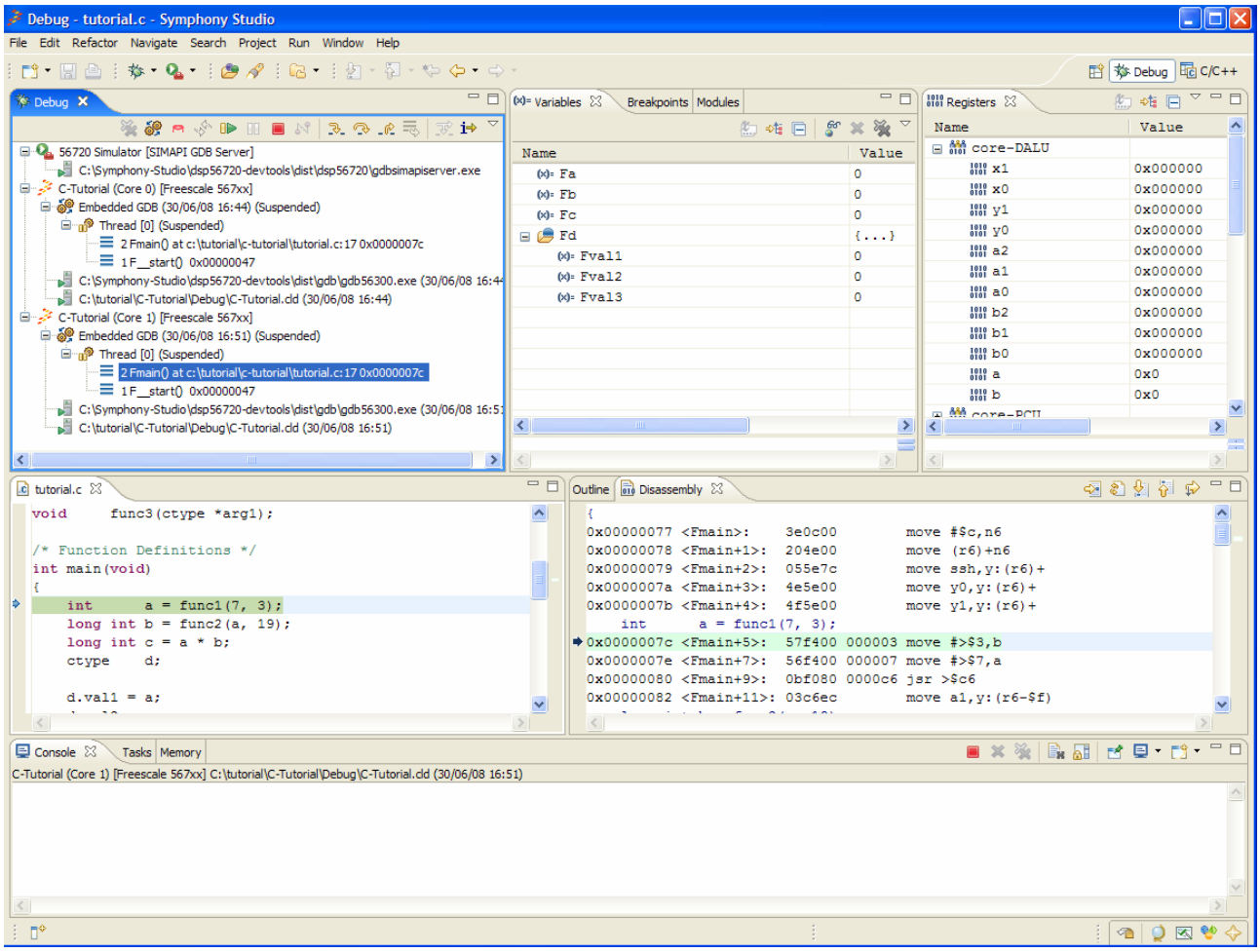


At this point you are debugging the code running in the private memory of Core0. Let's now also load the program to the private memory of Core1 and debug both sessions simultaneously.

1. Launch the second GDB session by selecting **Run -> Debug**, then selecting *C-Tutorial (Core1)* in the side pane and hitting **Debug**. The debug session starts and the `C-Tutorial.cld` file gets loaded into the connected DSP56720 device (Core1).

2. Notice carefully the information in the *Debug* tab. Here you can see the information on the two sessions running (such as *C-Tutorial (Core0)* and *C-Tutorial (Core1)*). You can now switch context between the two cores. Select the *Thread* under *C-Tutorial (Core1)*. All subsequent commands will now be performed on Core1.

3. Press **Step Over** a few times, you should now be presented with a window looking something like the following:



4. Switch threads again and make sure you know how to control each core.
5. Each debug session can be killed by hitting the **Terminate** buttons in the respective threads.

NOTE

You can also right-click on a session in the *Debug* window and select *Terminate All* and then again with *Remove All Terminated*.

That's it. Play around to become familiar with the Eclipse development and debugging environment. Make changes to the source code or write your own application, recompile and debug again. Check out the *Disassembly* window, the *Breakpoints* window, the *Variables* window the *Registers* and *Memory* windows. Add and remove breakpoints, kill then restart the debug session.

More details on specific items discussed in this tutorial can be found in Chapter 4, "Advanced Topics." Also to gain a further understanding on how Eclipse works in regards to your workbench, views and

perspectives you can view the on line help system, Help->Help Contents and select Workbench User Guide.

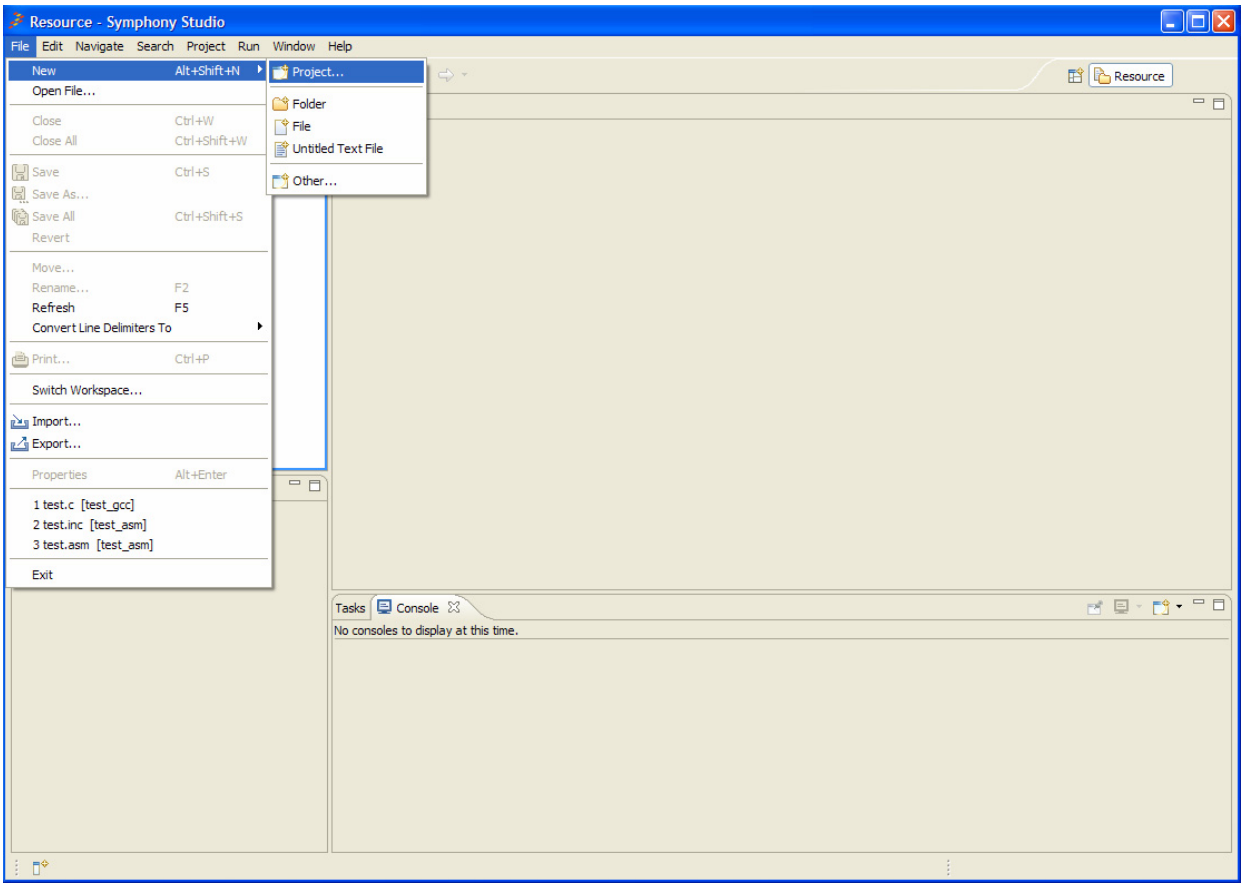
For a copy of this tutorial, see /sample-projects/ *Symphony Studio* Installation/ *.*.

2.2 ASM Tutorial

This section describes the specific steps for the ASM tutorial.

2.2.1 Creating a New Project

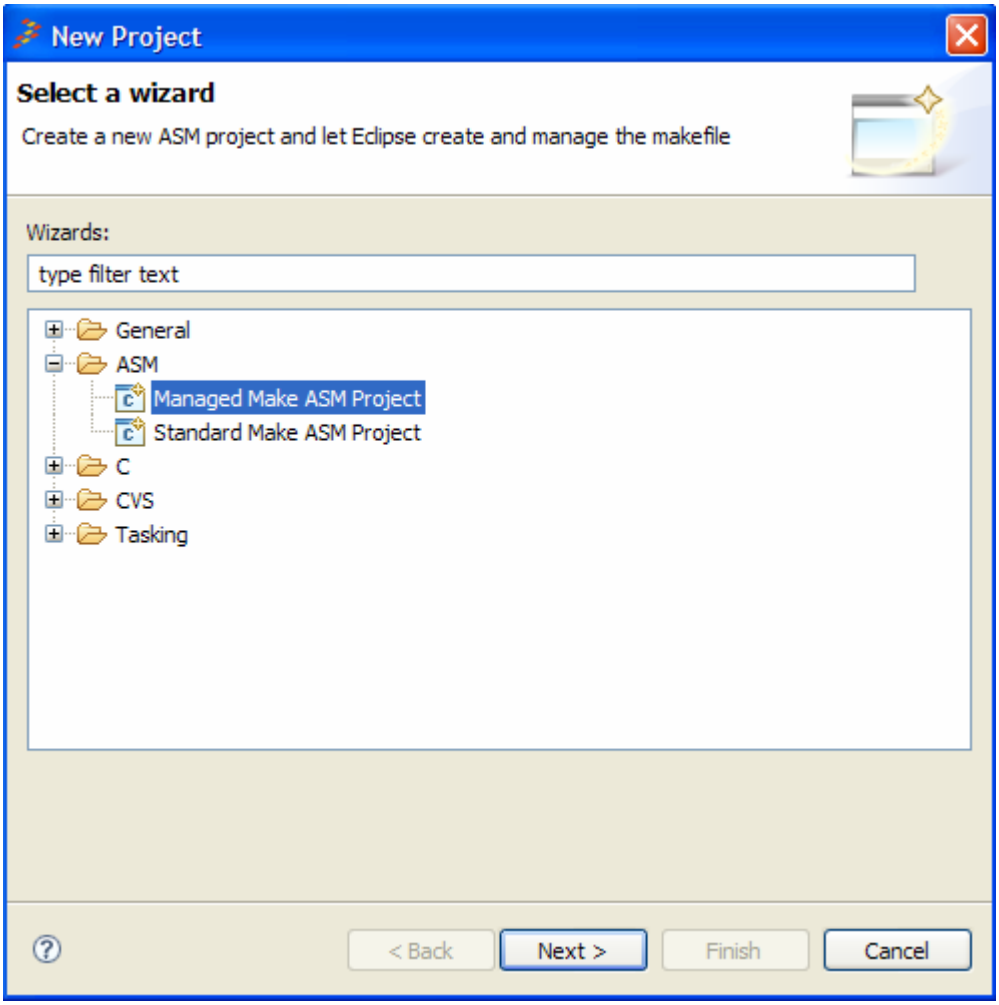
1. Select File -> New -> Project.



TIP

In the C/C++ Perspective you can quickly open a new Managed Make ASM Project by right click in the C/C++ Projects View and selecting New->Managed Make ASM Project.

2. Expand the *ASM* folder and select the *Managed Make ASM Project* option. Click *Next*.

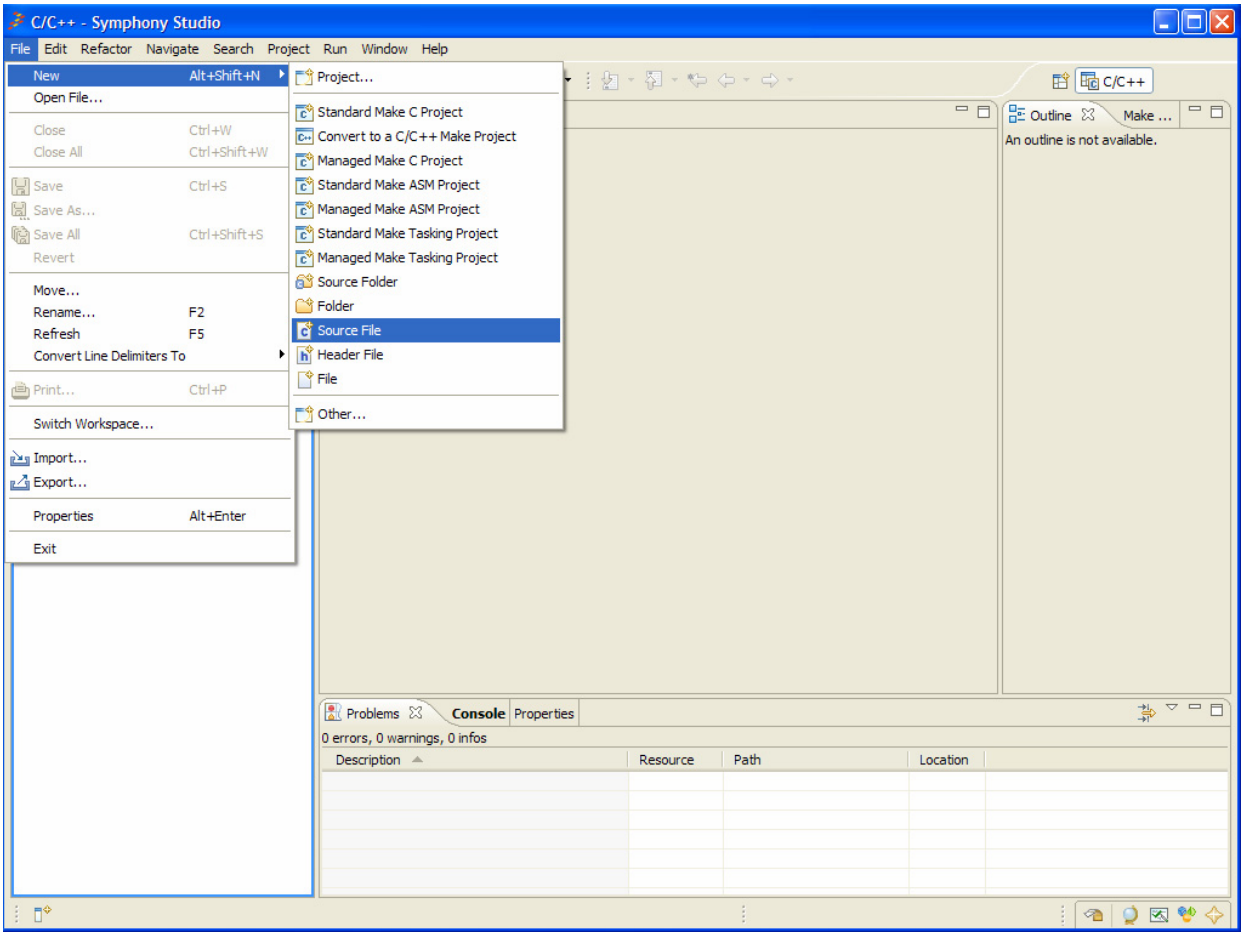


3. Select a suitable workspace directory if prompted (this is used to store all your projects)
4. Set the *Project name* to `ASM-Tutorial`
5. Click the *Next* button and the *Project Type* should be automatically set to `56K ASM COFF`
6. Click the *Finish* button.

7. Click *Yes* if you are offered the option to open the C/C++ perspective now. The *ASM-Tutorial* project appears in the *C/C++ Projects* tab

2.2.2 Adding Source Files

1. Right-click on the *ASM-Tutorial* project in the *C/C++ Projects* tab and select **New -> File**.



2. Use the following settings:
   - Source Folder: `ASM-Tutorial` (default)
   - Source File: `tutorial.asm`

3. The empty *tutorial.asm* file appears under the *ASM-Tutorial* project and is automatically opened by Symphony Studio for editing. Note that an error is reported initially as it tries to compile the empty file into a CLD file.

4. Add the following source lines to the *tutorial.asm* file:

```
M_ID   EQU      $FFFFF5          ; X space:ID Register
```

```
;*****************************************************************************
      ; Init data storage
      ;*****************************************************************************

            org     x:$00
      TEST_BASE      equ     *
      Test_offset_1  ds      1
      Test_offset_2  ds      1



      ;*****************************************************************************
      ; Init interrupt vectors
      ;*****************************************************************************

            org     p:$00
            jmp     START

;*****************************************************************************
      ;  Start
      ;*****************************************************************************
          org     p:$100
      START
            ori     #$03,mr                 ; mask interrupts
            move    #0,omr
            movec   #0,sp                   ; reset hardware stack pointer

            ;move ID register to x:$10
            move    #$10,r0
            move    x:M_ID,x0
            move    x0,x:(r0)+

            ;Initialize data storage
            move    #$0,r0
            move    r0,x:TEST_BASE
            move    r0,x:TEST_BASE+1
      LOOP
            move    x:TEST_BASE,a
            move    x:TEST_BASE+1,b
```

```
        jsr     ADD
        move    a,x:TEST_BASE
        move    b,x:TEST_BASE+1
        jmp     LOOP
ADD
        add      a,b                          ;a = a + b
        add      #$1,a                        ;a++
        rts
```

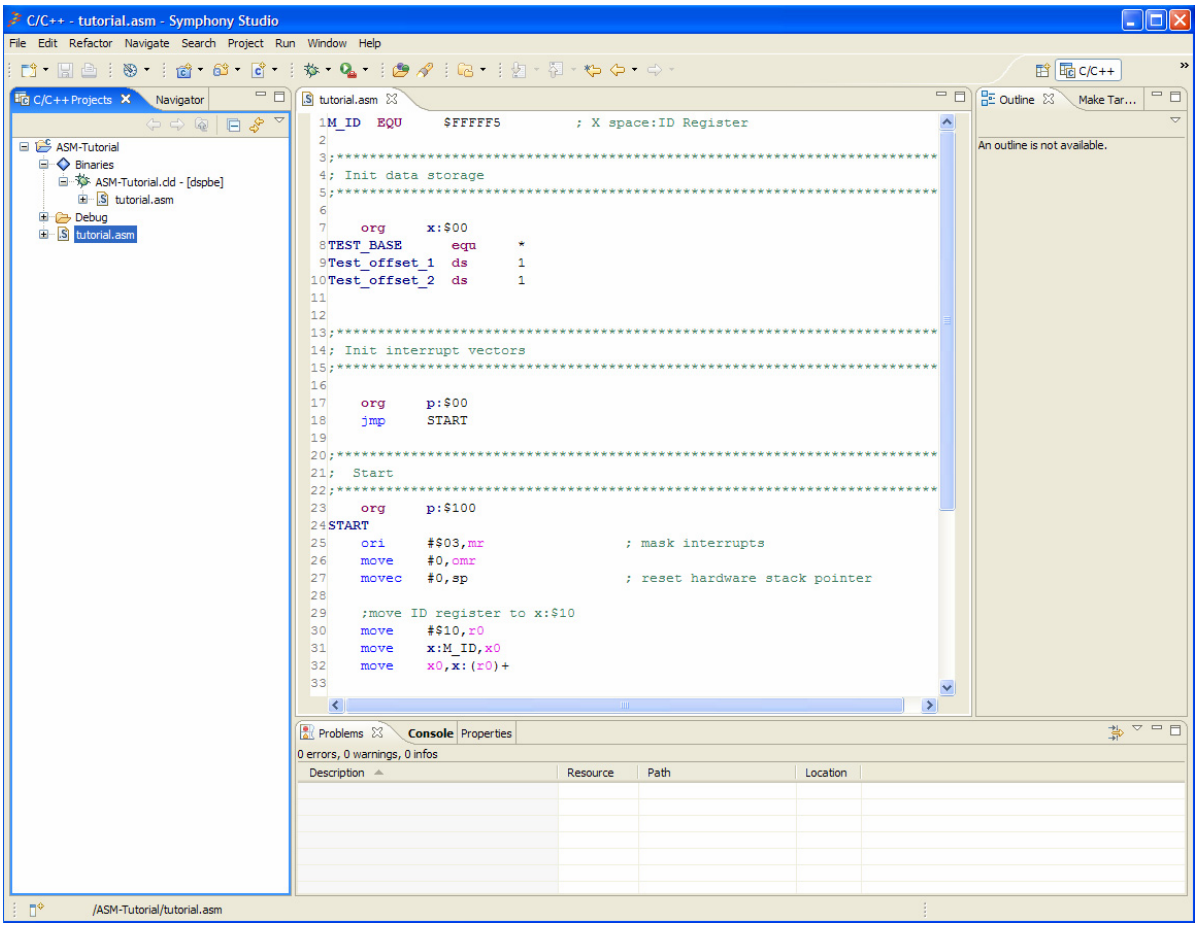
5. Save your work by clicking the *Save* button. Eclipse automatically compiles the project when you do this and the earlier reported error should no be preset anymore.

NOTE

If you want to disable automatic building, uncheck **Project -> Build Automatically**. To clean your project, select **Project -> Clean**, select the projects you want to clean and hit the **OK** button.

TIP

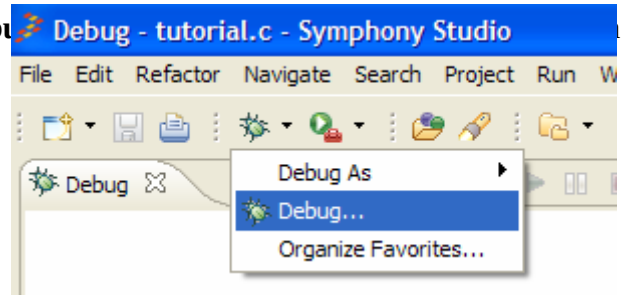
Symphony Studio includes a Freescale DSP COFF Binary parser and identifies all valid COFF files in a Binaries tree element in the project folder. You can expand the COFF file to see which source files it references.



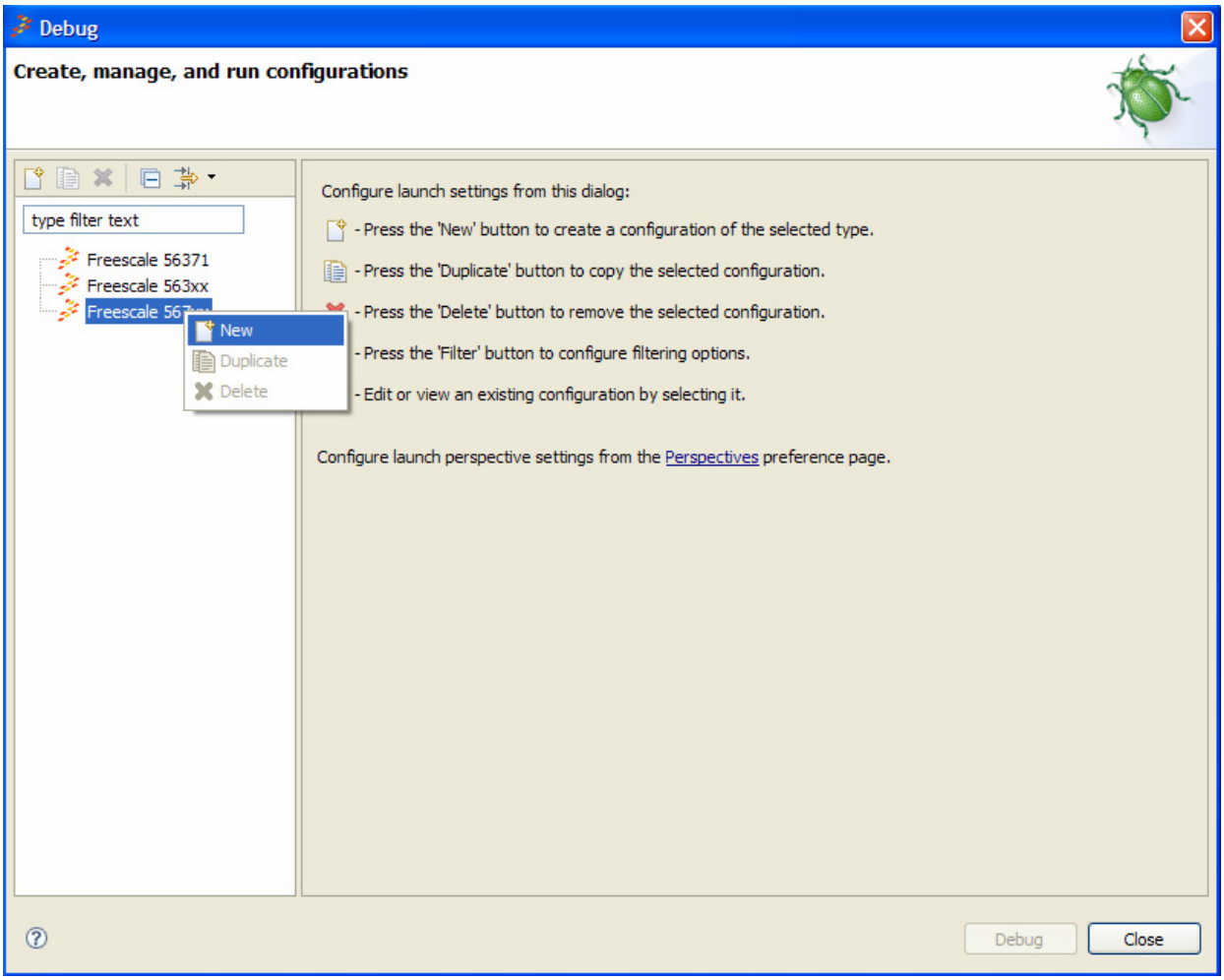
2.2.3 Creating a Debug Configuration

You will now create two debug configurations (one for each DSP56720 core).

1. Switch to the *Debug* perspective by selecting **Window -> Open Perspective -> Other** and choosing the **Debug** option.
2. Select **Run -> Debu** menu in the tool bar.



1. Right click on the *Freescale 567xx* target and select **New**.

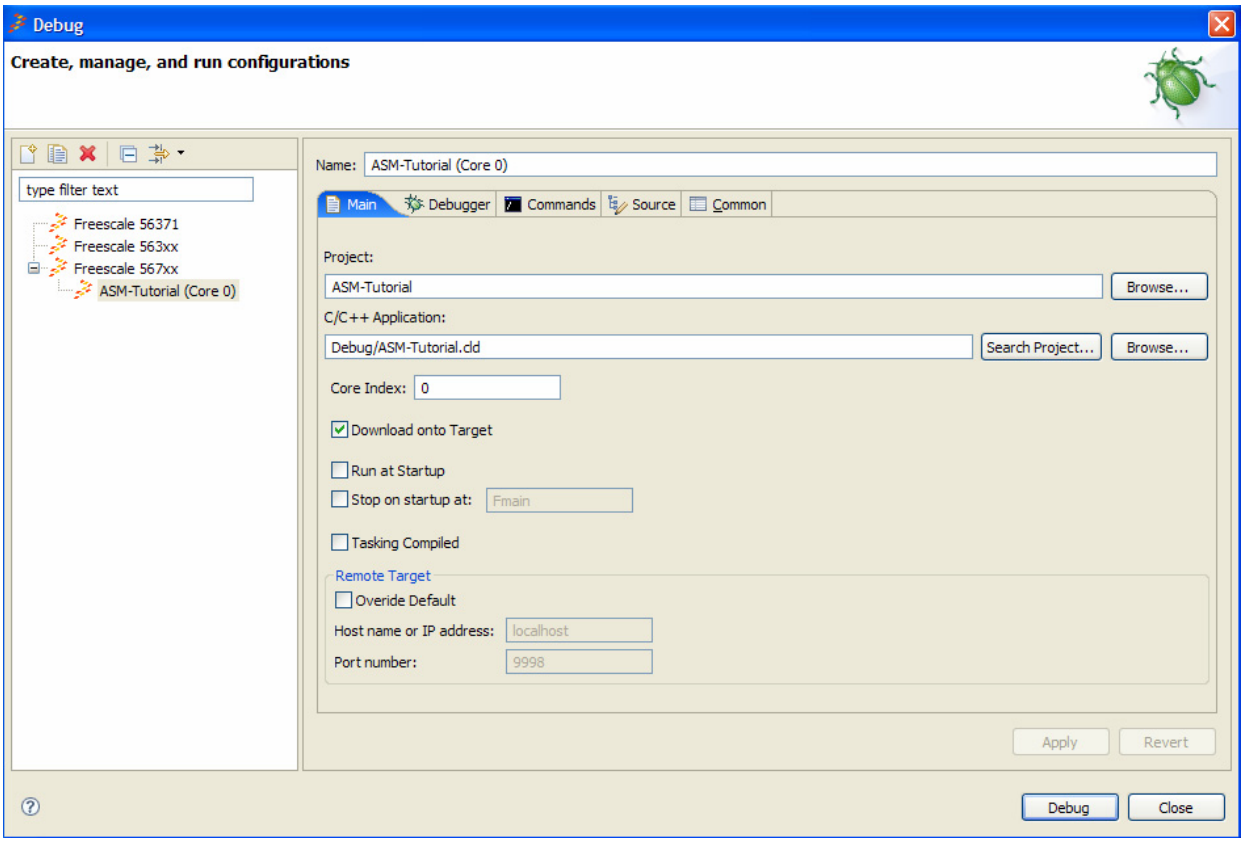


2. In the *Name* field change the text so it reads `ASM-Tutorial (Core0)`

3. The project field should currently be pointing to *ASM-Tutorial*. Click the *Search project* button on the C/C++ Application field and select the *ASM-Tutorial.cld* file that should be listed there. Set the Core Index to 0 if it isn't already.

NOTE

If you want to debug a CLD file that is not part of the current project you can use the **Browse** button, which lets you select any CLD file available on your computer.



3. Repeat steps 1-5 to create a debug target for Core1 (changing the name and Core Index appropriately).

NOTE

Symphony Studio recognizes that it is an ASM project and automatically de-selects “Run at Startup” and “Stop on Startup”. As a result when you launch your application, the PC points to the entry point of the cld file, usually 0.

More details on the other options available for the Freescale 56xxx debugger target are documented in Section 3.9, “Freescale 56xx Debug Target Options.”

2.2.4 Debugging your Application

You are now ready to debug your application running on the DSP56720 simulator. Follow Section 2.1.4, “Debugging your Application,” from the previous tutorial, which outlines the procedure when debugging an application created from a C file. The steps are identical when the application is created

from an ASM file except for steps 5 and 7 select the debug connections you created in Section 2.2.3, “Creating a Debug Configuration.”

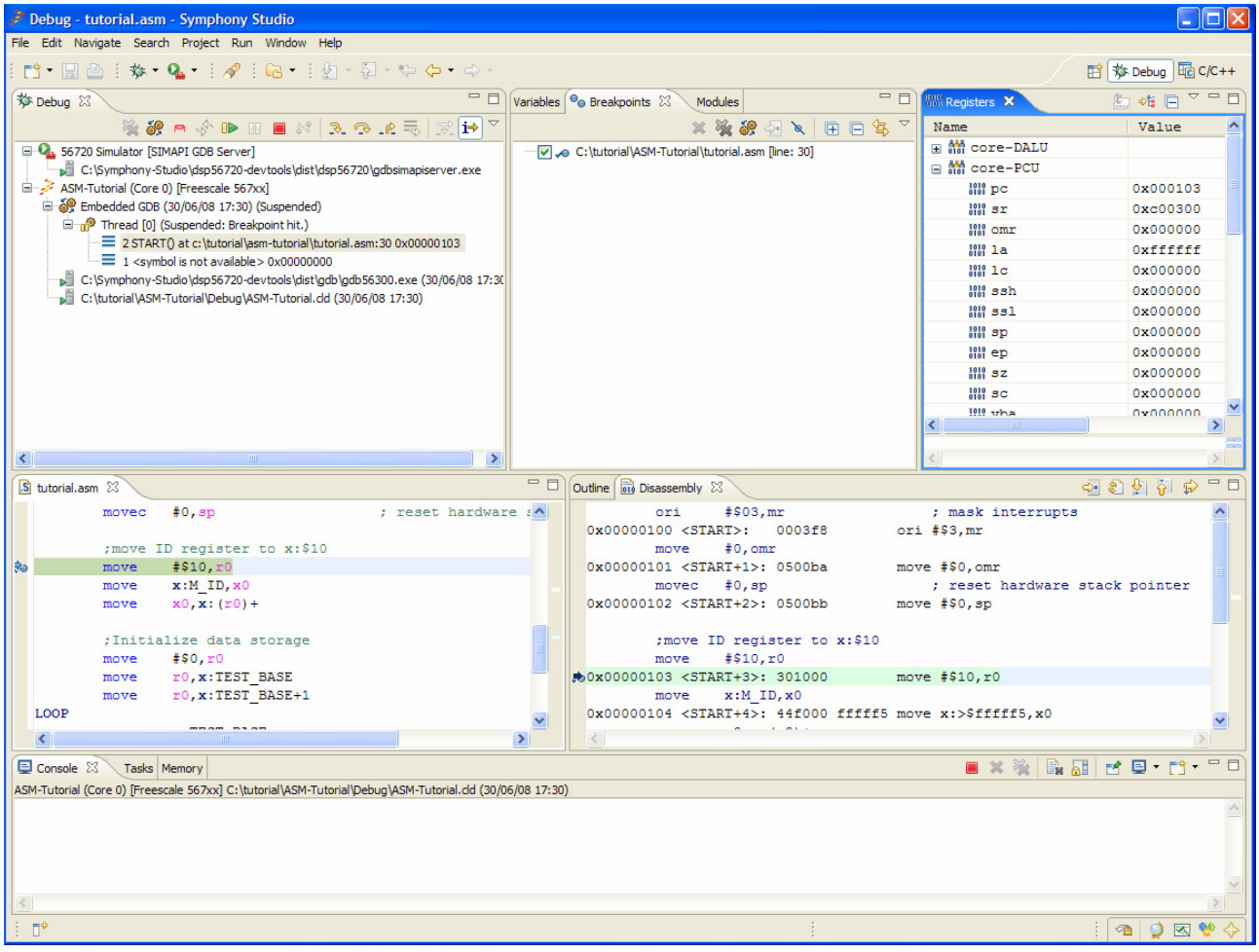
A copy of this tutorial project can be found in the *Symphony Studio* Installation directory under the sample-projects directory.

Chapter 3
Advanced Topics

This chapter provides advanced topics.

3.1 Breakpoints

You can apply breakpoints in either the *Source Code* window or in the *Disassembly* window. You do this by *Double Clicking* on the left hand border. A Blue Circle is displayed where the breakpoint occurs. Breakpoints are also listed in the Breakpoint view. They can be disabled or removed using the breakpoint view or by double clicking on the blue circles.

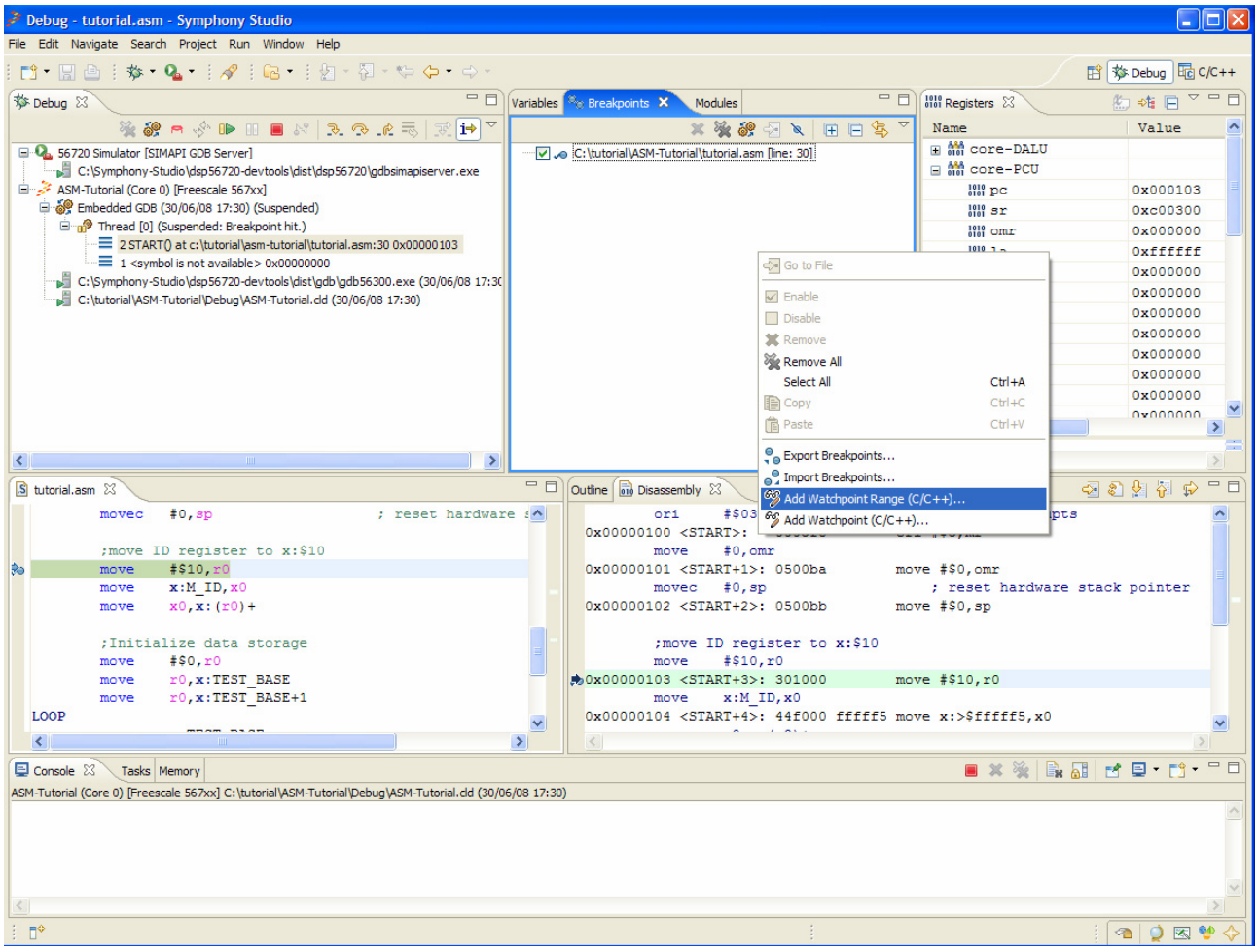


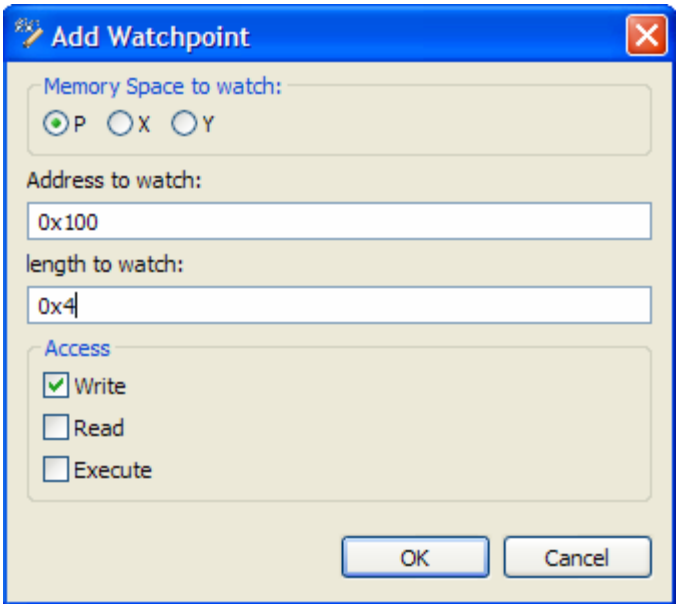
3.1.1 Hardware Breakpoints

By default when a breakpoint is inserted a software breakpoint is chosen. If a hardware breakpoint is desired, instead of *Double Clicking* on the left hand border of either the *Source Code* window or in the *Disassembly* window, right click and select *Toggle Hardware Breakpoint*. A different icon is displayed to indicate a hardware breakpoint is chosen. Hardware breakpoints are disabled and removed in the same manner as software breakpoints.

3.1.2 Hardware Watchpoints

You can apply a hardware watchpoint which spans over an address range. These watchpoints break the execution of the core when read, write, execute or either read or write accesses to a specified memory region occur. To create simply *Right-Click* in the breakpoint view and select *Add Watchpoint Range.* A *Add Watchpoint* dialog appears where you select the memory space (P,X or Y), starting address, length and access for the watchpoint.





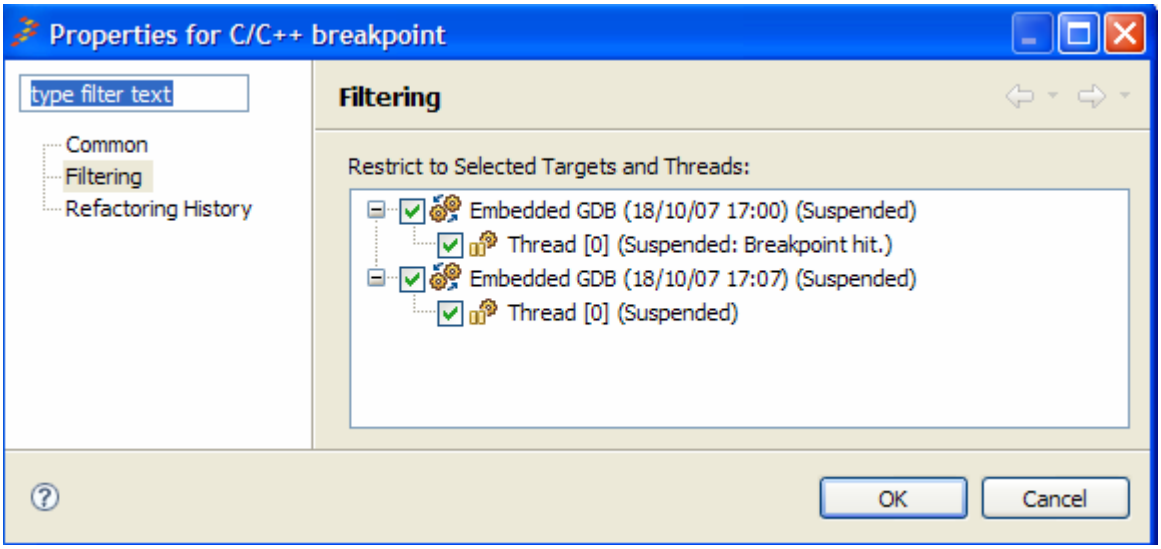
3.1.3 Multi-Core Breakpoints

For a multi-core simulation an enhanced breakpoint implementation is desirable which allows:

1. Apply a breakpoint to all cores such that either core stops if it reaches the breakpoints
2. Apply a breakpoint to a specific core such that only the specified core stops if it reaches the breakpoint.
3. Apply a breakpoint to a specific core such that both cores stops if the specified core reaches the breakpoint.

By default all breakpoints are triggered as per the first option, however it is possible to specify breakpoints as per the second option from the Eclipse interface. To do this:

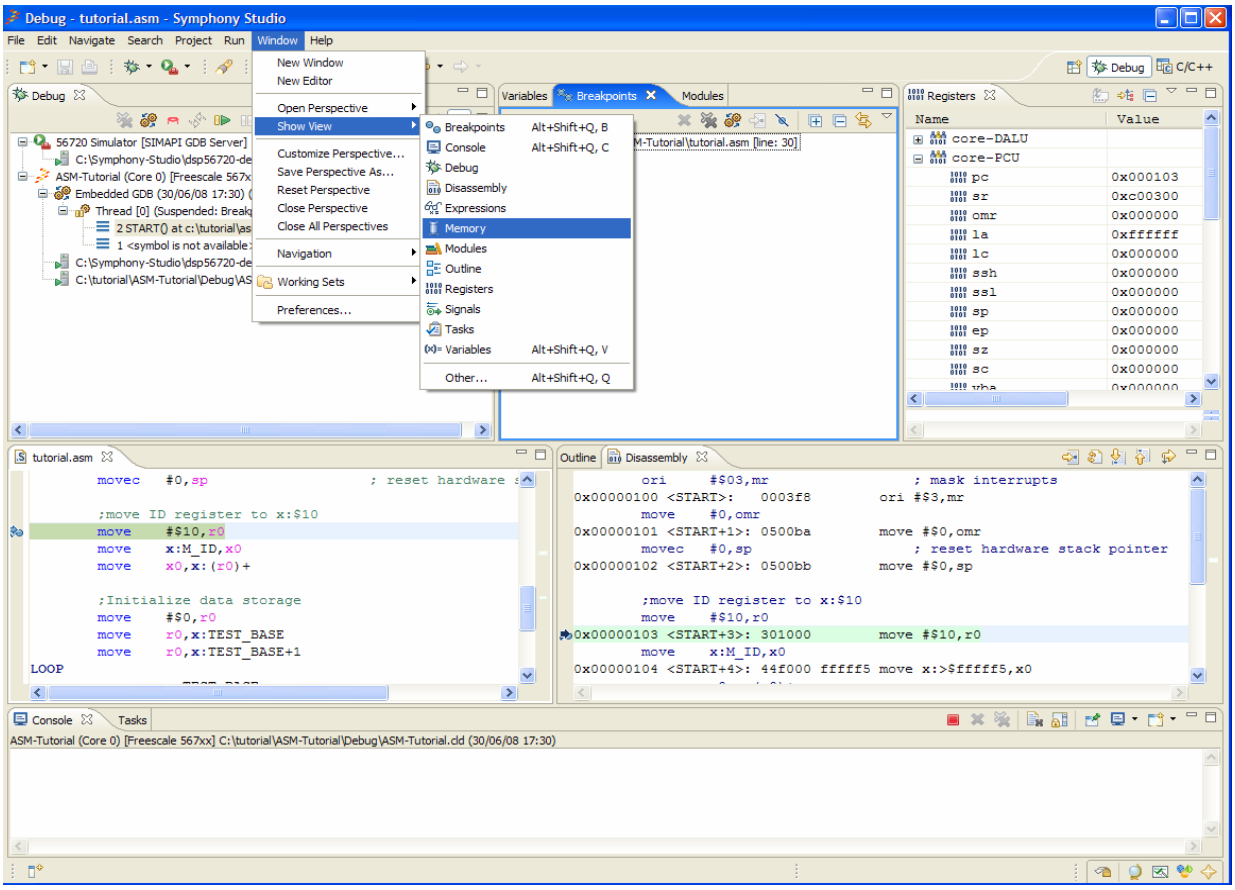
1. Right-Click on a previously set breakpoint and select *Breakpoint Properties...*. The **Properties for C/C++ breakpoint** window appears.
2. Click on *Filtering* in the left navigation pane to bring the **Filtering** tab to the foreground.
3. De-select the *gdb Debugger* (ie Core) that you do not wish to apply the breakpoint to. The following figure illustrates the setup.
4. Click **OK**.



3.2 Memory

It is possible to view memory sections from the four memory regions P,X , Y and L. In Symphony Studio the view which shows a section of memory is called a memory render. To open a memory render do this:

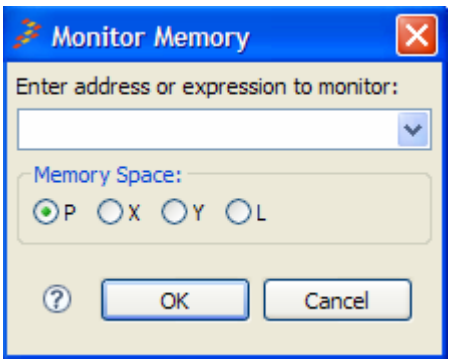
1. In the Debug view, select a debug session. Selecting a thread or stack frame automatically selects the associated session.
2. Open the memory view by selecting **Window->Show View->Memory** as shown below



3. Click on the green plus sign in the Memory Monitor pane, which brings up a Monitor Memory pop up and select the starting address and which memory region you wish to view

TIP

You can also right click in the Memory Monitor pane to add and delete renders.



You now have completed the setup, and while you debug your code, any changes in memory are highlighted in red. It is also possible to have several memory renders open simultaneously from different or the same memory region (P, X, Y and L). To do this click on the *New Memory View* Button in the Memory view tool bar (this is the first button). Another Memory View is created, which lists all current memory renders. Follow the steps above to create a new memory render.

3.2.1 Memory Render Format

To view memory in a different format:

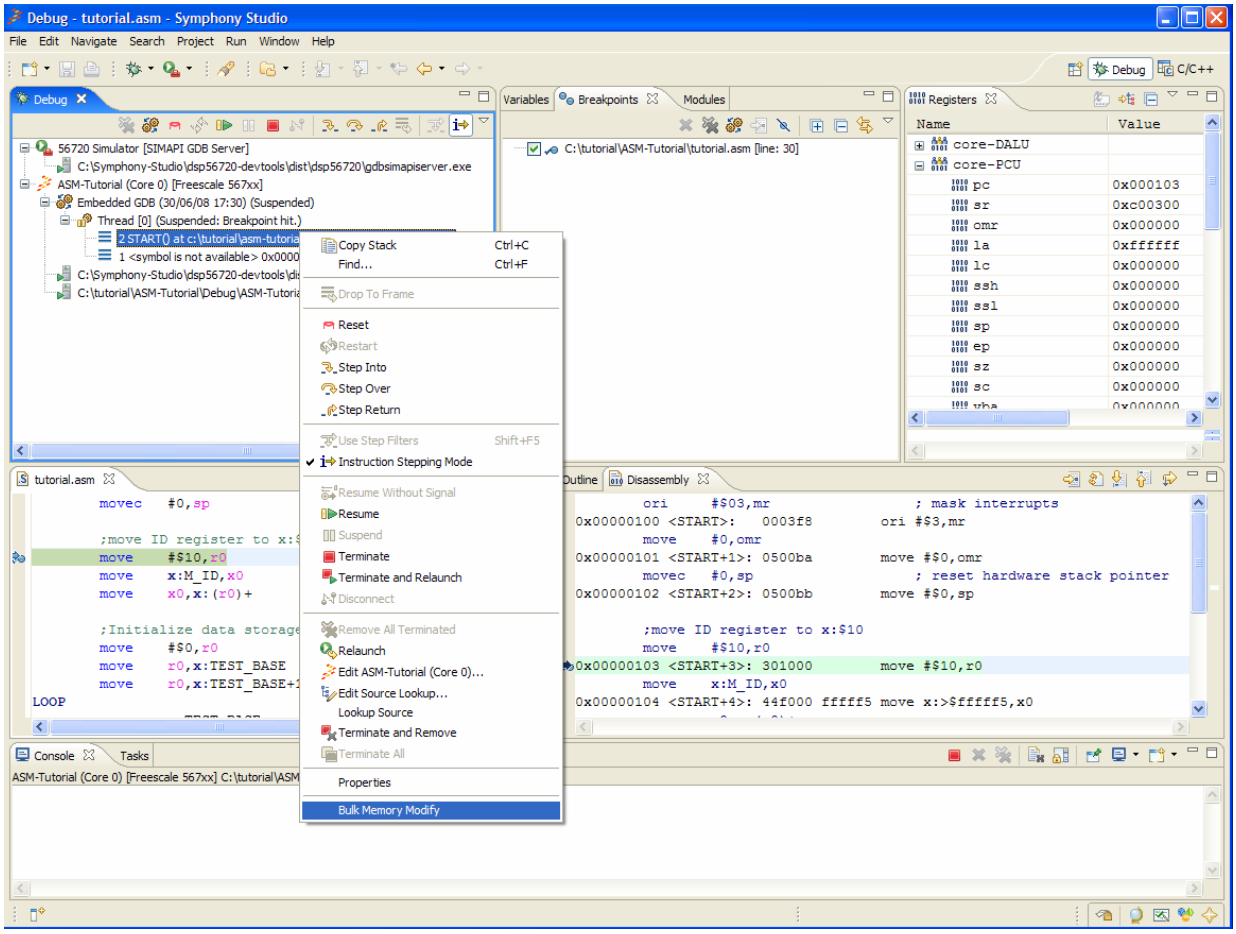
1. Select **"Add Rendering"** in the context menu of the Memory Renderings pane. The **"Add Memory Rendering"** dialog appears. Alternately you can use the green plus sign in the Memory Renderings Pane.
2. Select the formats from the list and press **"OK"**. It is possible to have multiple formats open of the same memory render.
3. You can delete a render format by selecting "**Remove Rendering**" in the context menu of the Memory Renderings pane. Alternately you can use the grey x sign in the Memory Renderings Pane.

3.2.2 Modifying Memory

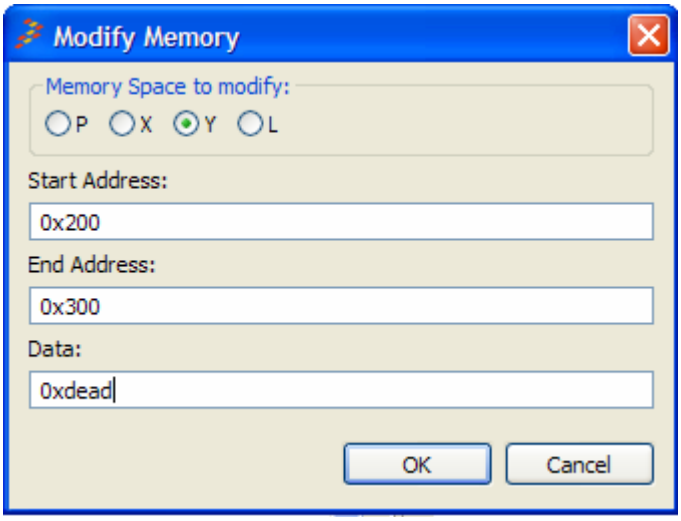
To change the contents of memory simply click the word of memory you wish to change and enter in the value in hex (without the "0x" prefix). If you have chosen the fractional format you enter in fractional values. To get the cursor to appear in the cell double click the cell instead of a single click. To cancel your change before you have finished entering in the value press the escape key. Once you have finished entering in the value press the enter key and wait for the memory view to update to reflect your change.

You may also do a bulk memory modify which modifies multiple words in memory with the same data word. To do this:

1. In the Debug view, select a debug session. Selecting a thread or stack frame automatically selects the associated session. Right click and select *Bulk Memory Modify*



2. A Modify Memory dialog appears and selects the memory region (P,X,Y and L), start address, optional end address and the data word.



3.3 Registers

The register view is automatically opened for the debug perspective. If the registers view has been closed to re-open it select **Window->Show View->Registers**. Registers are displayed by default in Hex format. To change the format select either one or multiple registers and right click, select Format and choose the desired format.

To change a register click the Value cell for the register you wish to change and enter in the value. By default the value is decimal and to enter the value in hex use the “0x” prefix. To enter the value in fractional format, suffix the value with *fr*, i.e -0.5*fr* or 1*fr*. To cancel your change before you have finished entering in the value press the escape key. Once you have finished entering in the value press the enter key and wait for the register view to update to reflect your change.

3.4 Variables

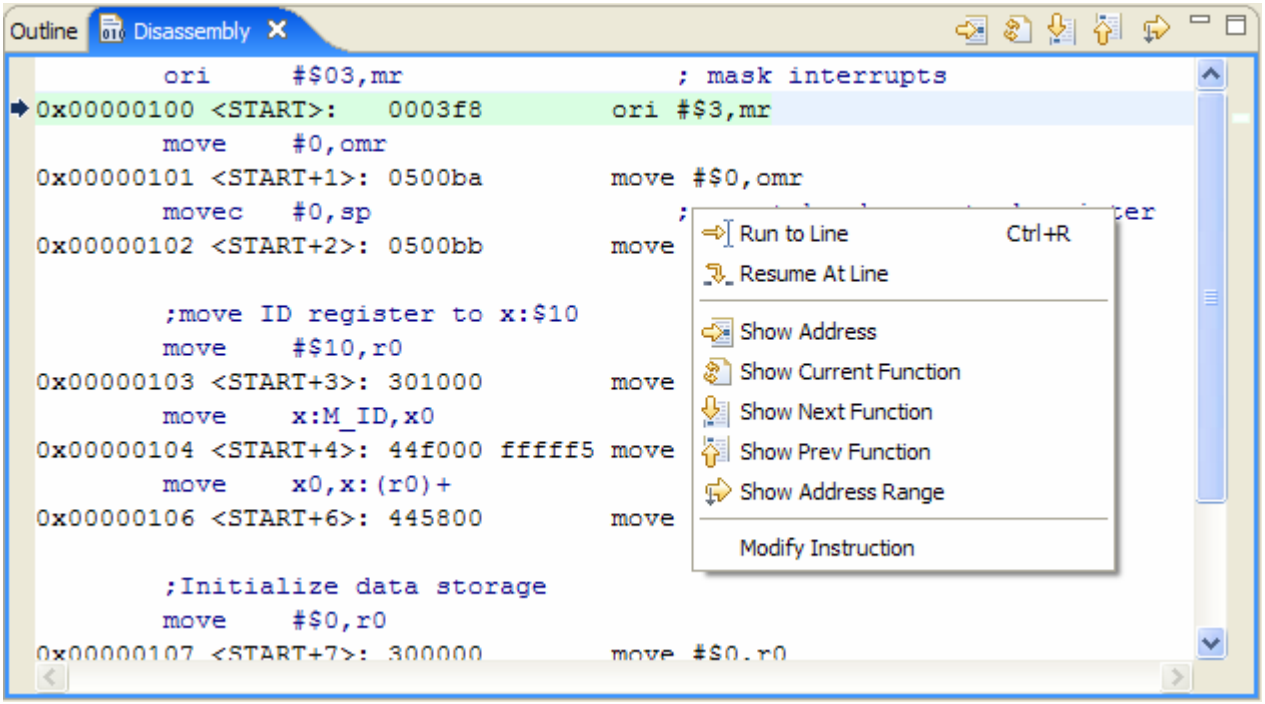
The variable view is automatically opened for the debug perspective. If the variable view has been closed to re-open it select **Window->Show View->Variables**. Variables are displayed by default in the natural format. To change the format select either one or multiple variables and right click, select Format and choose the desired format.

Similar to modifying registers to change a value click the Value cell for the variable you wish to change and enter in the value. By default the value is decimal and to enter the value in hex use the “0x” prefix. To enter the value in fractional format, suffix the value with *fr*, i.e -0.5*fr* or 1*fr*. For long values, to enter in a long fractional value, suffix the value with *lfr*, i.e -0.5*lfr* or 1*lfr*. To cancel your change before you have finished entering in the value press the escape key. Once you

have finished entering in the value press the enter key and wait for the register view to update to reflect your change.

3.5 Disassembly View

The disassembly view is automatically opened for the debug perspective when instruction stepping is enabled. To open the view, manually select **Window->Show View->Disassembly**. The disassembly view shows a mixture of source code and the corresponding disassembled instructions. The disassembly view's range for instructions disassembled is the start and end address for the function for the current program counter. Symphony Studio has extended the disassembly view to allow the user to watch an arbitrary function, not only the current function. To watch an arbitrary function either right-click inside the disassembly view or alternately you can use the buttons on the disassembly view's tool bar.



- Show Address

This opens a dialog for the user to enter in an address (prefix with 0x for hex values). The view then disassembles the function for which this address lies in.

TIP

Show Address is also in any source code window's context menu. When selected, it disassembles the function for the selected line of code, assuming the target is running.

- Show Current Function

This disassembles the function for the current program counter.

- Show Next Function

This disassembles the function that starts directly after the currently shown function.

- Show Prev Function

This disassembles the function that ends directly before the currently shown function.

- Show Address Range

This opens a dialog for the user to enter in the start and end address (prefix with 0x for hex values) to be disassembled.

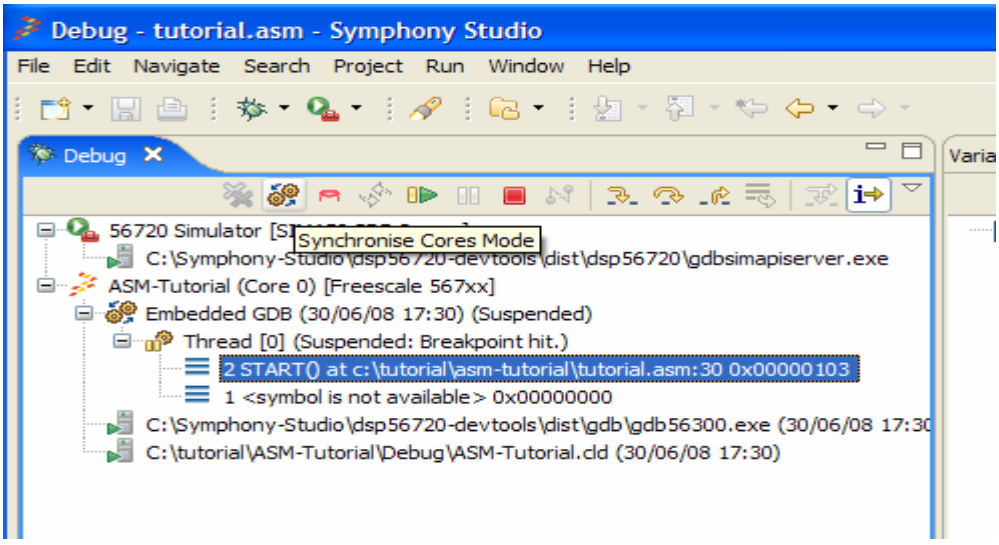
NOTE

The user must be careful with this option as it does not have any protection if a start address is chosen in the middle of a double word instruction.

The Modify Instruction option allows the user to do in line assembling. When selected, a dialog appears containing the currently selected instruction. The user can then modify it or enter in any mnemonic which is assembled.

3.6 Synchronize Control of DSP56720 Cores

If you are debugging two applications simultaneously on the dsp56720 you can use synchronize control to send commands to both cores with the one set of commands. Once both applications have been launched, in the Debug view, select a debug session. Selecting a thread or stack frame



automatically selects the associated session. Then select the *Synchronize Cores Mode* button.

The cores are now synchronized and each step, run and suspend are sent to both cores. Once one core stops execution the other is stopped as well.

3.7 Creating a Standard Make Project

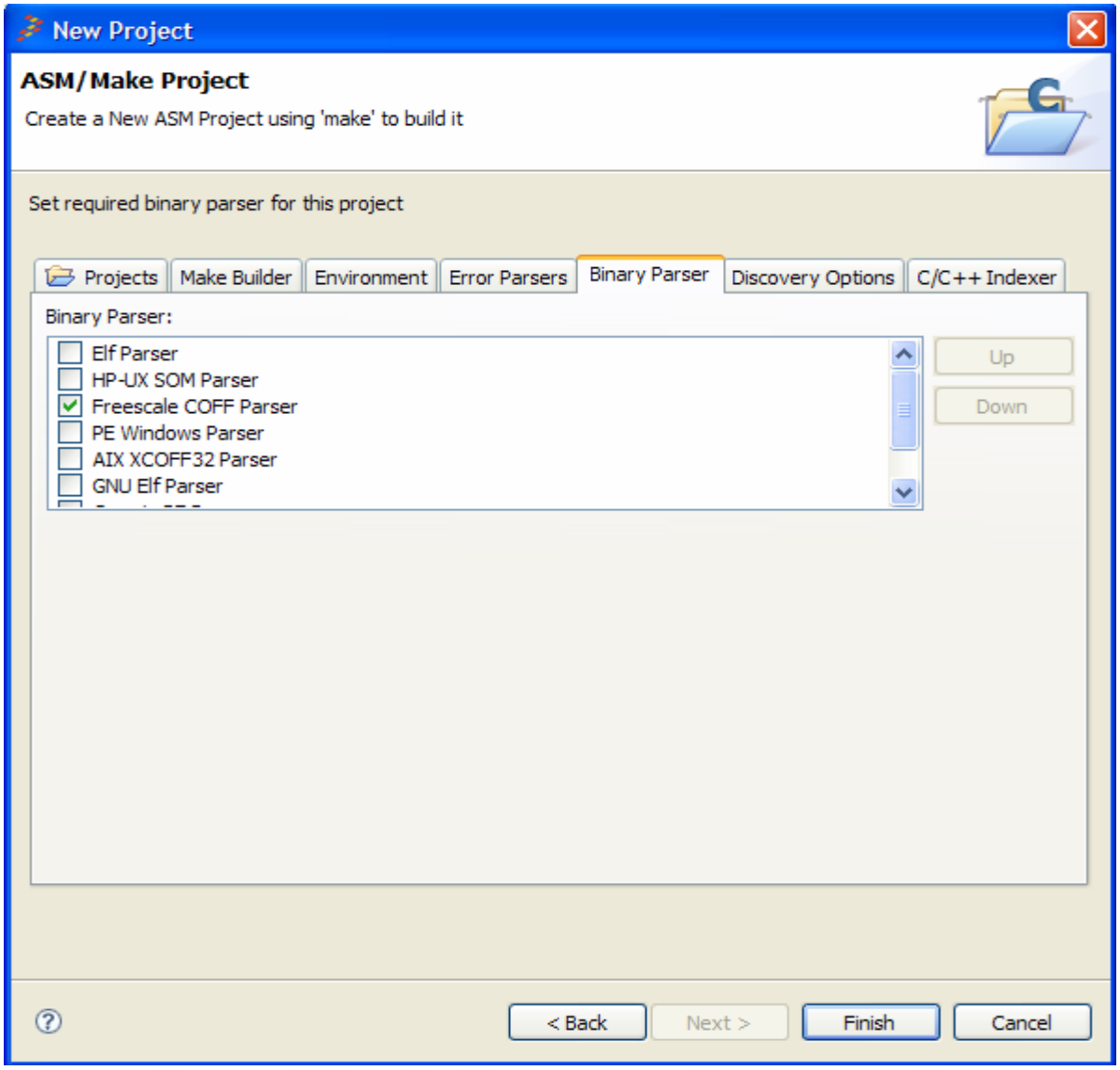
If you want to import an existing project into Symphony Studio, or if you have a complex build process which requires more features than Managed Make can provide, then you may need to create a *Standard Make Project*.

To create a *Standard Make Project* select **File -> New -> Project**, expand the *C* Folder, select *Standard Make C Project* and click *Next*. Or if you are creating a Tasking C project select **File -> New -> Project**, expand the Tasking Folder, select *Standard Make Tasking Project* and click *Next*. Or if you are creating an ASM project select **File -> New -> Project**, expand the ASM Folder, select *Standard Make ASM Project* and click *Next*.

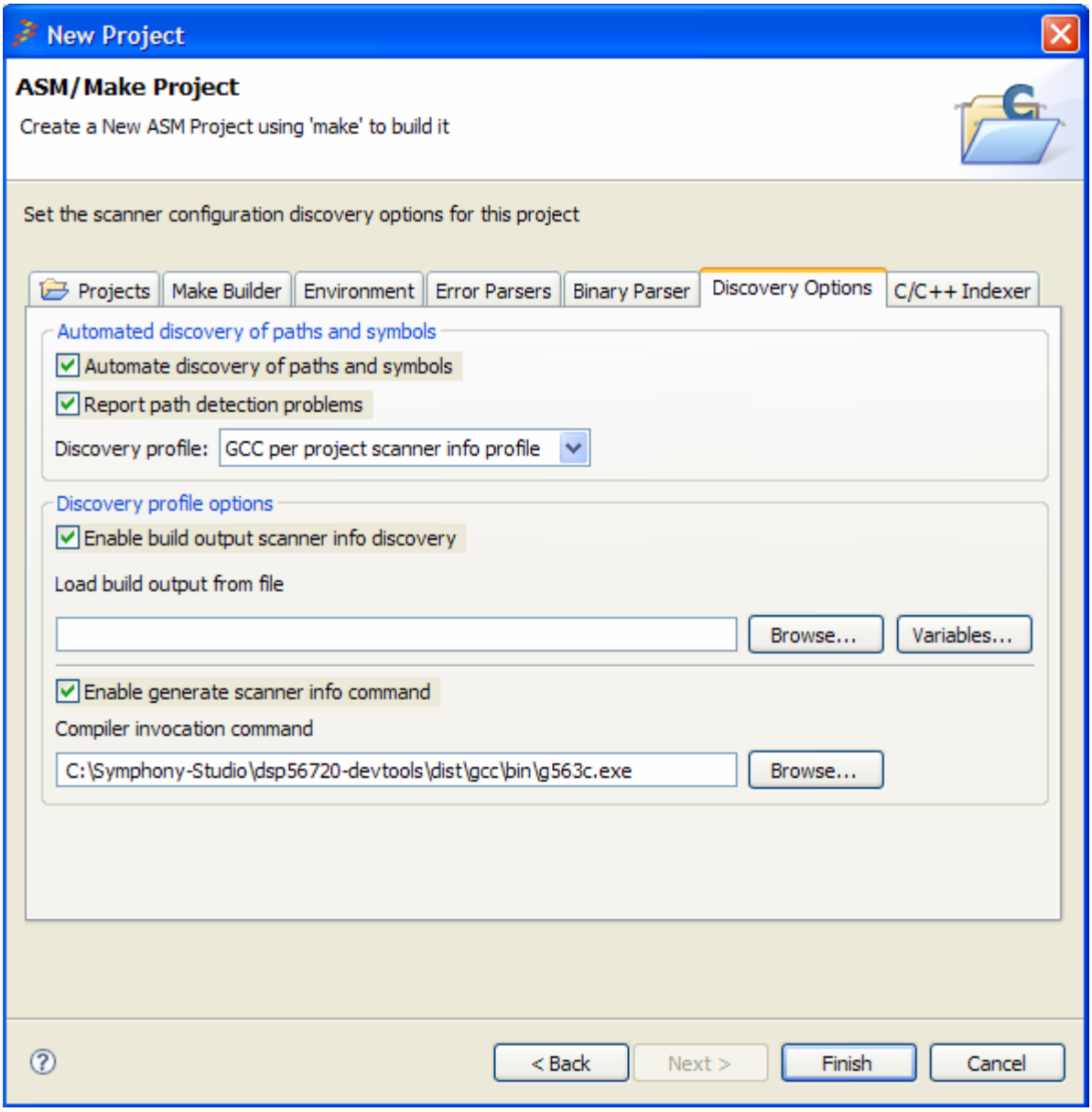
TIP

In the C/C++ Perspective you can quickly open a new Standard Make C, Tasking or ASM Project by right click in the C/C++ Projects View and selecting New->Standard Make C Project or New->Standard Make Tasking Project or New->Standard Make ASM Project.

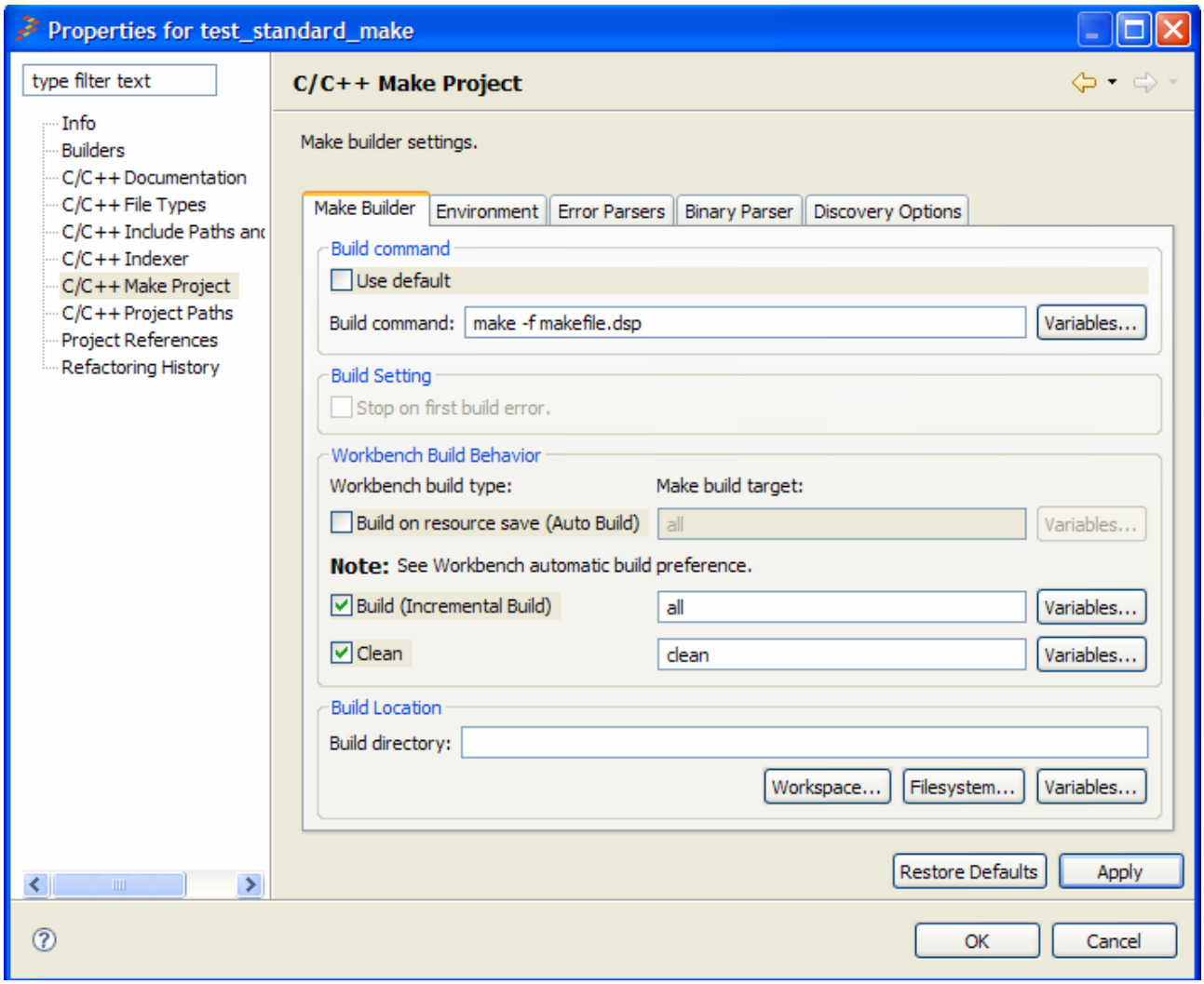
Provide whatever name you want for the project and click *Next* again. On this dialog you need to select the *Binary Parser* tab and make sure that the **Freescale COFF Parser** box is checked.



In the Discovery Tab select g563c for the *compiler invocation command* by using the browse button.



You can now add whatever files you want to the project. At the moment the command that is run to build the project is *make*. You can change this by selecting **Project -> Properties** and selecting the *C/C++ Make Project* option in the left panel. On this page you can change the build command to use a different makefile, or select different targets for *Build* and *Clean*.



3.8 Managed Make Options

3.8.1 Build Configurations

When creating either the “56K GCC COFF” or “56K ASM COFF” managed make projects two build configurations are defined, Debug and Release. These configurations allow the user to specify different command line options for the build tools.

By Default the debug configuration enables the required flags to produce COFF debugging information while the release configuration has no flags set.

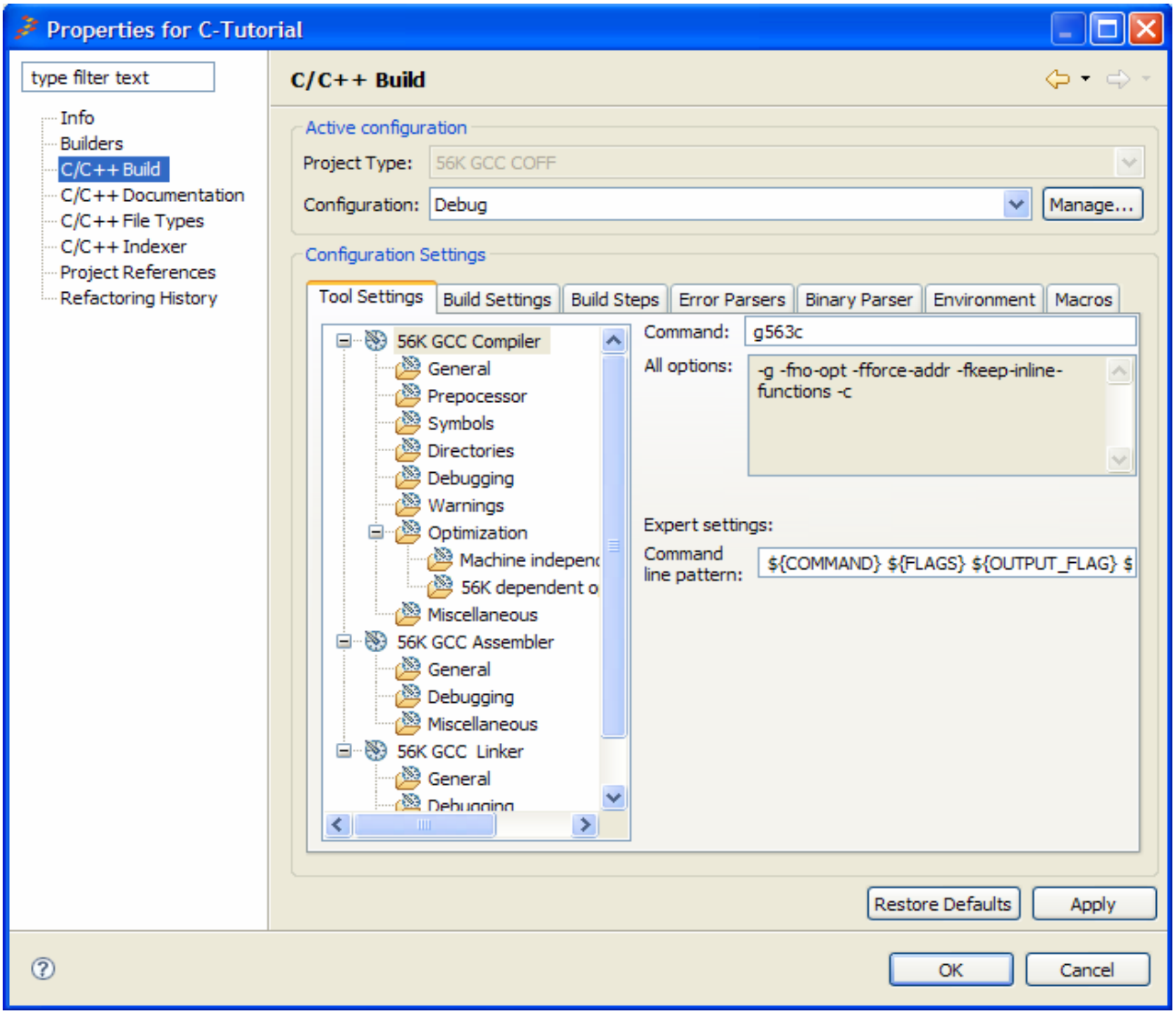
A build or clean is only performed on the active configuration. To change the active configuration, go to **Project -> Active Build Configuration.**

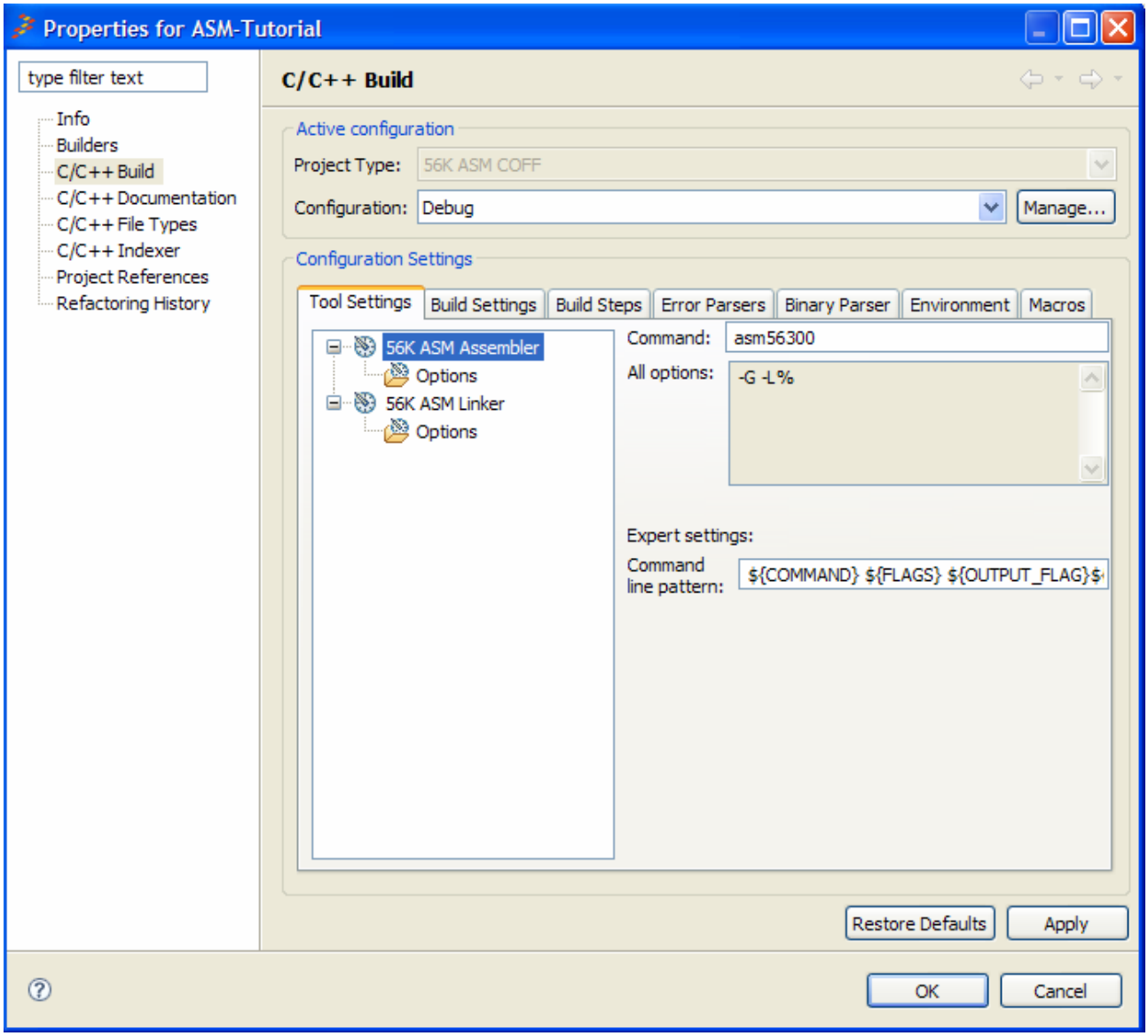
3.8.2 Build Options

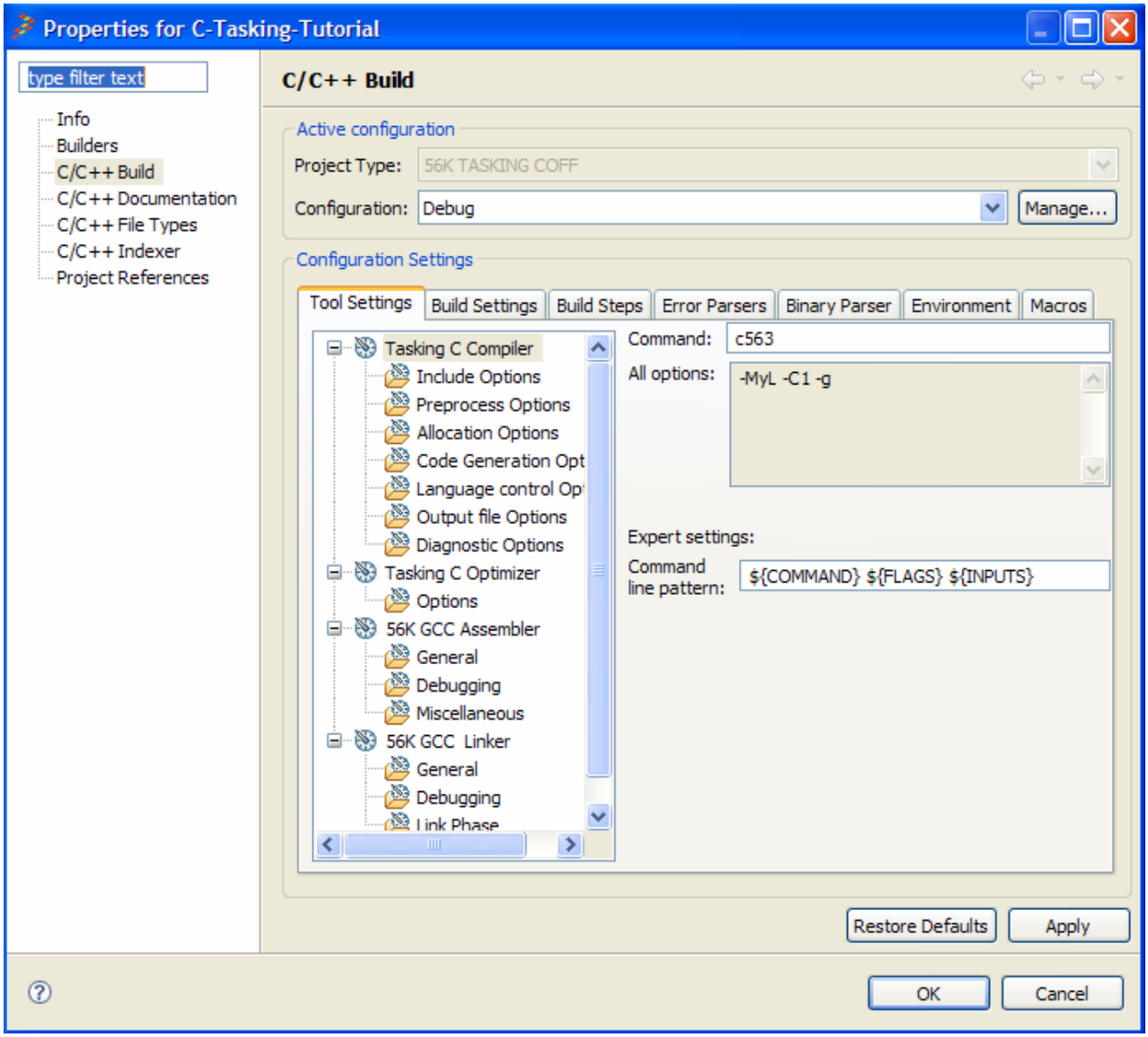
To change any of the GCC Compiler, 56300 Assembler or Linker options do the following:

1. Highlight the project you wish to change. This may be a “56K GCC COFF” or “56K ASM COFF” managed make project.
2. Select Project -> Properties.
3. In the left pane select *C/C++ Build*.
4. Select the configuration to edit, or create a new configuration by selecting Manage...
5. In the Tool Settings tab there is a hierarchal display for each tool. The options are grouped into categories. When changing all desired options, a handy tip is to select the tool and look at the All options box. For more information on each option please refer to *DSP563CCC Manual, DSP563ASM Manual or DSP563LNK Manual*.
6. Select Apply and OK.

You can change the final generated coff filename in the Build Settings tab next to the Tool Settings tab. Below shows you the tools and their category's for “56K GCC COFF”, “56K ASM COFF” and “56K TASKING COFF” managed make projects.



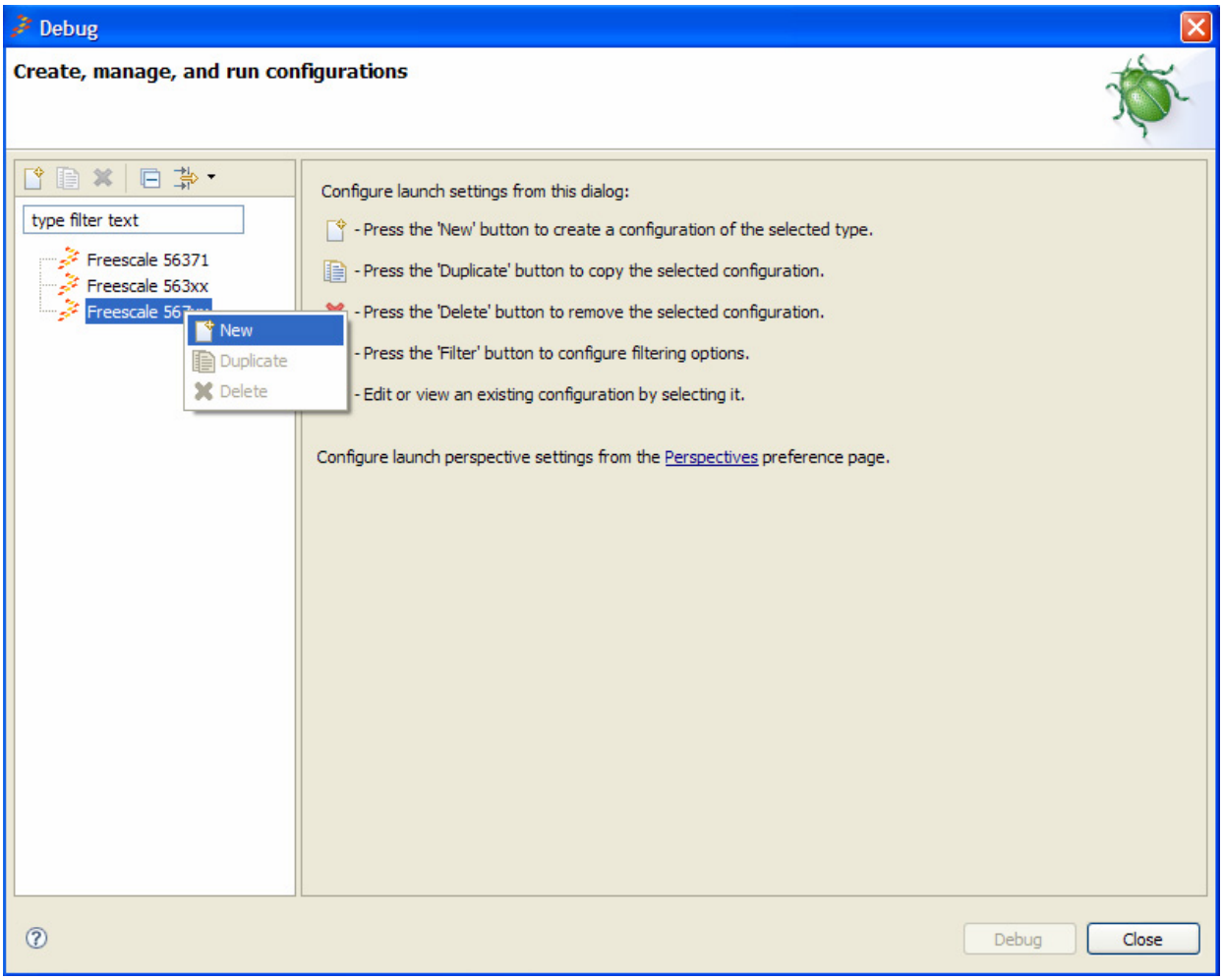

Properties for ASM-Tutorial
type filter text
Info
Builders
C/C++ Build
C/C++ Documentation
C/C++ File Types
C/C++ Indexer
Project References
Refactoring History
C/C++ Build
Active configuration
Project Type:
56K ASM COFF
Configuration:
Debug
Manage...
Configuration Settings
Tool Settings
Build Settings
Build Steps
Error Parsers
Binary Parser
Environment
Macros
56K ASM Assembler
Options
56K ASM Linker
Options
Command:
asm56300
All options:
-G -L%
Expert settings:
Command line pattern:
${COMMAND} ${FLAGS} ${OUTPUT_FLAG}$
Restore Defaults
Apply
OK
Cancel


Properties for C-Tasking-Tutorial
type filter text
Info
Builders
C/C++ Build
C/C++ Documentation
C/C++ File Types
C/C++ Indexer
Project References
C/C++ Build
Active configuration
Project Type:
56K TASKING COFF
Configuration:
Debug
Manage...
Configuration Settings
Tool Settings
Build Settings
Build Steps
Error Parsers
Binary Parser
Environment
Macros
Tasking C Compiler
Include Options
Preprocess Options
Allocation Options
Code Generation Opt
Language control Op
Output file Options
Diagnostic Options
Tasking C Optimizer
Options
56K GCC Assembler
General
Debugging
Miscellaneous
56K GCC Linker
General
Debugging
Link Phase
Command:
c563
All options:
-MyL -C1 -g
Expert settings:
Command line pattern:
${COMMAND} ${FLAGS} ${INPUTS}
Restore Defaults
Apply
OK
Cancel

3.9 Freescale 56xx Debug Target Options

This section explains how to create a Freescale 56xxx Target debug connection. To create a debug connection, do the following:

1. In the C/C++ Projects view, select a project.
2. Click Run->Debug.
3. In the Configurations box, either select Freescale 56371, Freescale 563xx or Freescale 567xx, right click and select new. The difference between the two configurations is what registers are displayed. Freescale 567xx displays all of the 56720 peripheral registers while Freescale 563xx only shows the core registers.



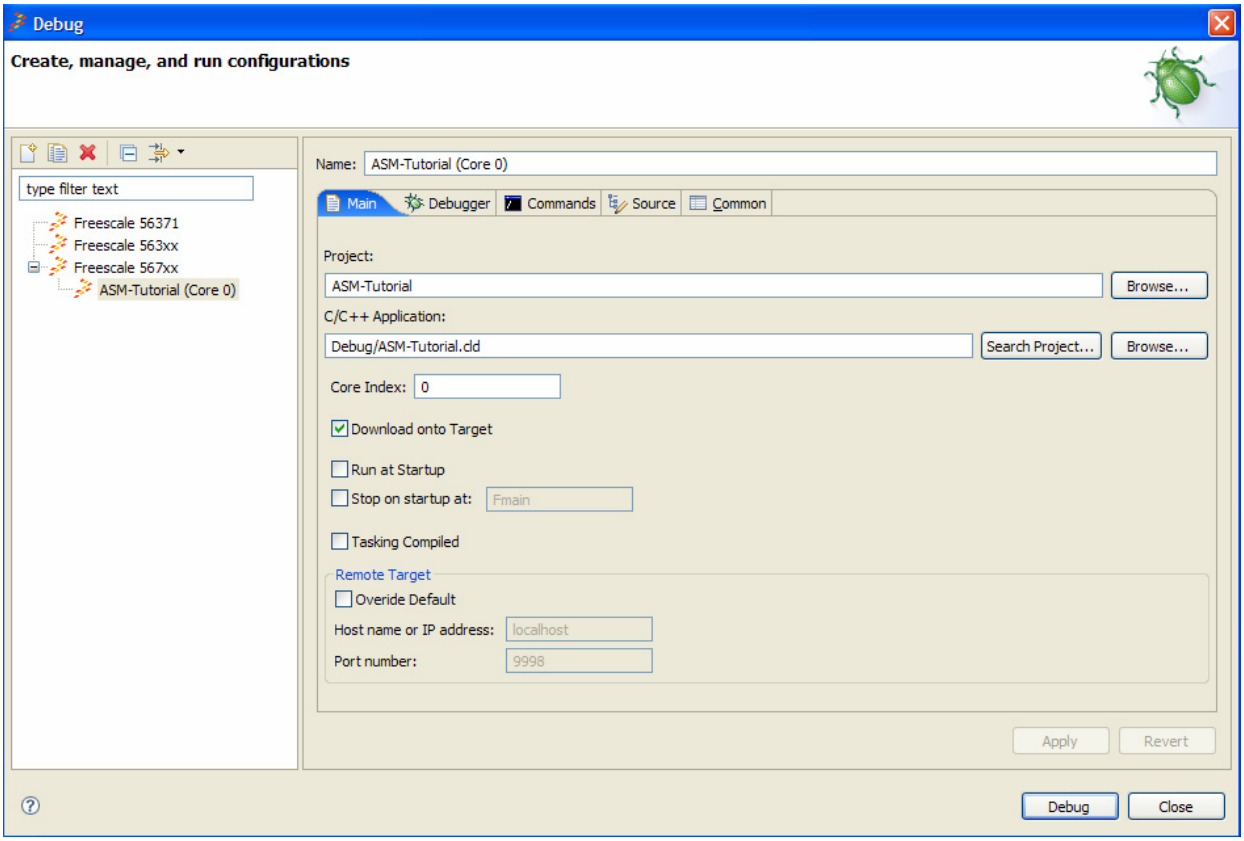
The debug dialog box contains the following tabs:

- Main
- Debugger
- Commands
- Source
- Common

3.9.1 Setting an Application to Debug

To set an application to debug, use these steps:

1. Select the **Main** tab.



2. Do the following:
   - In the **Name** box, type a descriptive name for this launch configuration.
   - In the **Project** box, type the name of the project containing the application that you want to debug. Alternately you can use the *Browse* button.

— In the **C/C++ Application** box, type the name of the CLD file that you want to debug. Alternately you can use the *Search Project* button to search within the project entered above. If the CLD file is not within the selected project use the *Browse* button to search for any CLD file available on your computer.

— In the **Core Index** box, select which core to debug the application on. If you have a chain of devices it is here where you specify which index of the chain to debug the application on.

3. The *Download onto target* check box instructs GDB whether to load the selected CLD file onto the target i.e. the EVB. By default it is enabled and **may** be deselected, for example, if your application is already loaded in FLASH.

4. The *Run at Startup* check box, if enabled, performs a resume at startup to run the target to the first breakpoint. When deselected the target's PC should equal the reset vector after launch or the entry point of the cld file.

5. To let your program run until you interrupt it manually, or until it hits a user breakpoint, clear the ***Stop at startup*** check box. However if selected you can change the symbol which to stop at on startup, by default this is Fmain.

NOTE

Symphony Studio recognizes that it is an ASM project and automatically de-selects “Run at Startup” and “Stop on Startup”. As a result, when you launch your application the PC points to the entry point of the cld file, usually 0.

6. The *Tasking Compiled* checkbox should be selected for C projects which have been compiled with the Tasking Compiler.

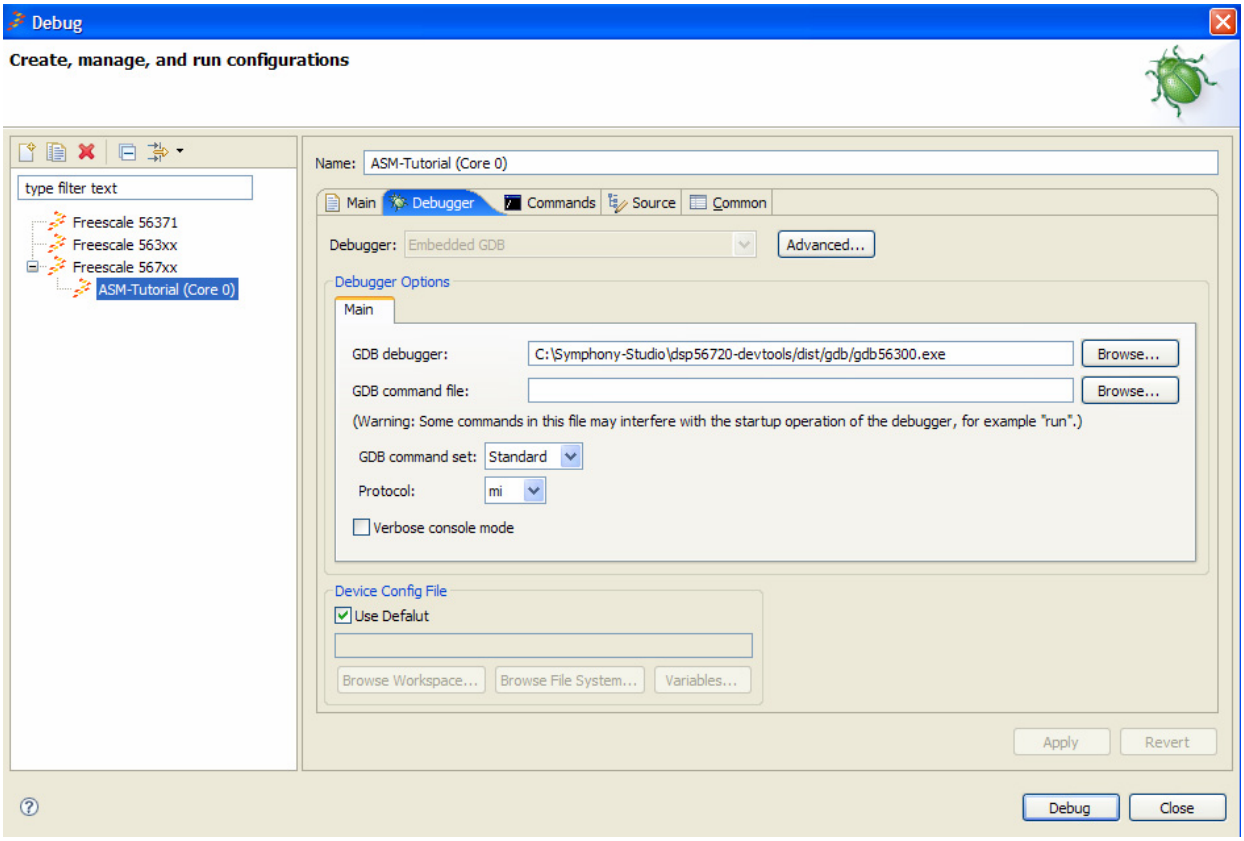
NOTE

Symphony Studio automatically selects the Tasking checkbox if the project was created as a new “Managed Make Tasking project” or “Standard Make Tasking Project”.

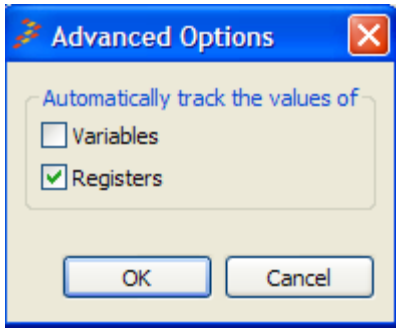
7. The Remote Target group allows you override the default and specify the host name (or IP address) and port number for the GDB remote server (such as, openocd, remoteocd or emuserver). To do this, select *Overide Default*, which enables the host name and port number text boxes.

3.9.2 Setting Debug Settings

To select a debugger to use when debugging an application, select the **Debugger** tab.

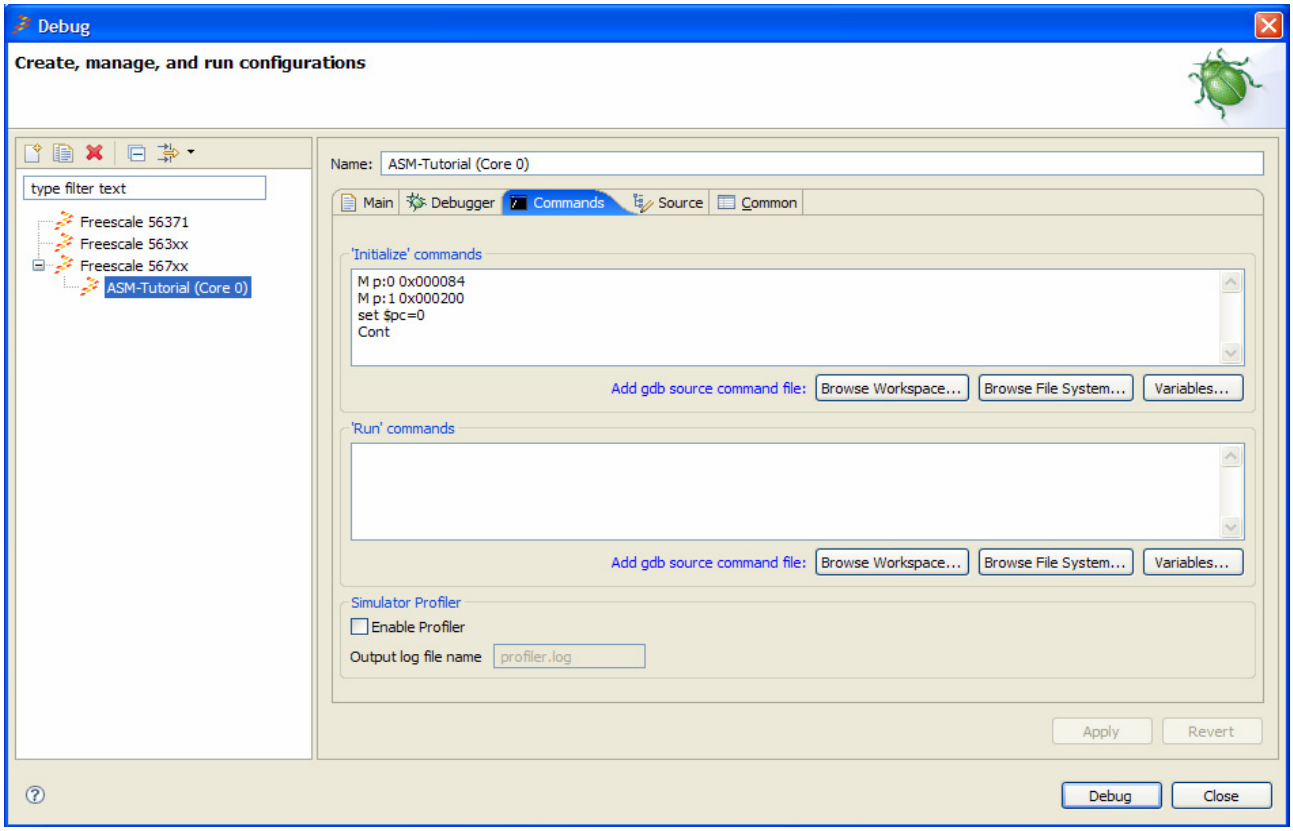


1. In the **GDB debugger** box, type the GDB to use to debug the application. By default the GDB debugger that is choosen is the gdb56300 that is part of the Symphony Studio release. You may use the *Browse* button to change this.
2. The *GDB command set* box should be left as *Standard*.
3. The *Protocol* box should be left as *mi*.
4. If *Verbose console mode* is selected, Symphony Studio lists all mi commands send to GDB and their replies.
5. The *Use Default* checkbox for “Device Config File” should be unselected if the user wants to specify the location of the configuration file. They can browse the current workspace. Browse the file system if the file is outside of the workspace, or use one of the in build Eclipse variables. For more information on Device Configuration files, see Section 3.10, “Device Configuration File.”
6. You can turn off the tracking of registers and variables by selecting the Advanced button, which brings up a dialog box. By turning off these options you minimise the amount of data that is read from the target. This may be useful if your C project contains some large arrays or data structures.



3.9.3 Setting GDB Commands

To add additional GDB commands to be executed at initialization or at run time select the **Commands** tab.



1. All “Initialize” commands are executed after connecting to the target but before downloading the cld onto it.

2. The “run” commands are executed directly after downloading the cld file. However they are executed before the target has hit the first breakpoint. So if you want to add to the Run commands you need to ensure the “Run at startup” in the main tab is de-selected. You can insert a cont as the last run command to do the same behavior.

A useful gdb command that has been created specifically for Symphony DSPs is a modify memory command M:

M <memspace>:<start_addr>[..<end_addr>] <data>

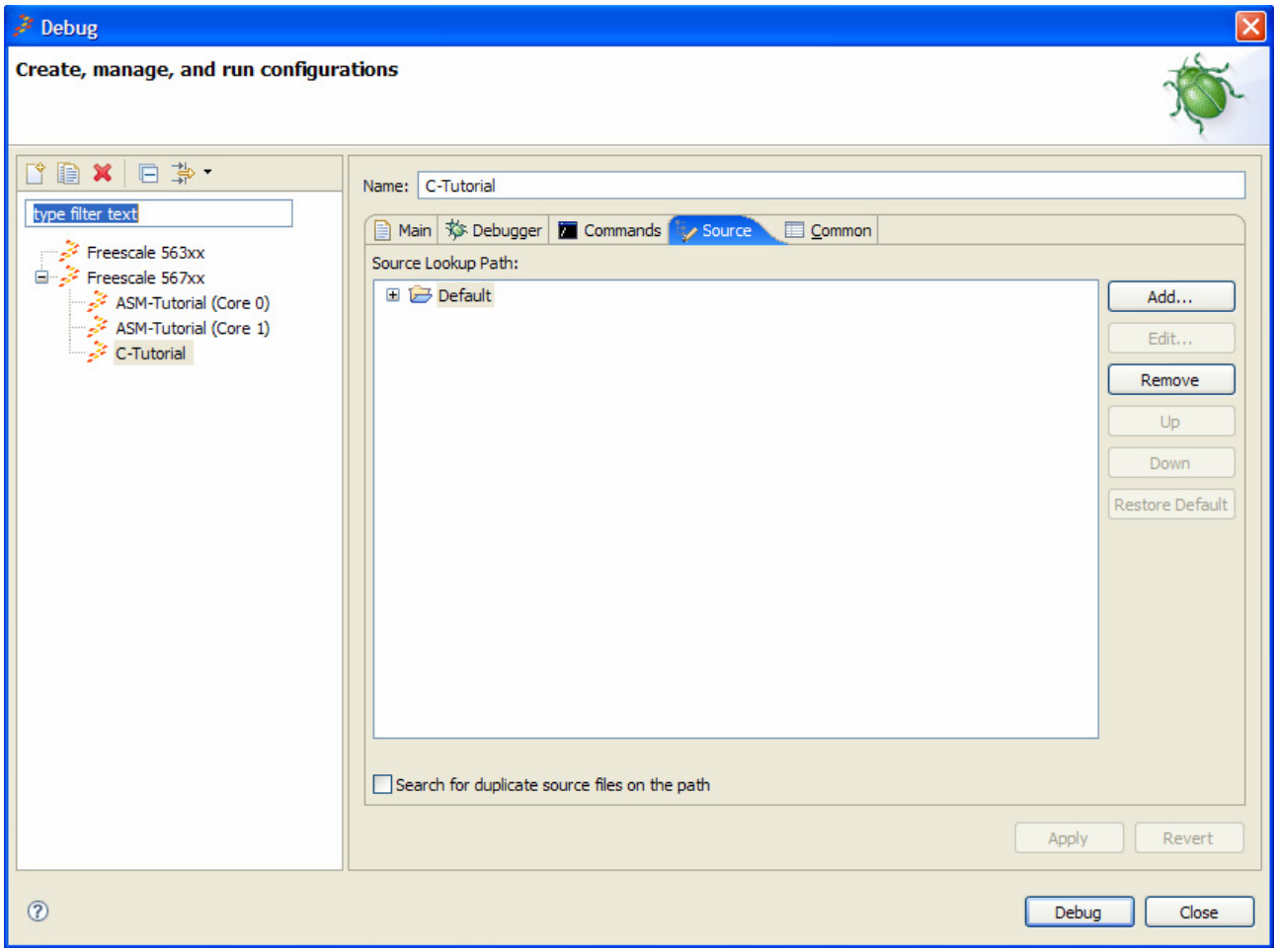
<memspace>: p | x | y | l
<start_addr>: start address, can be in hex by prefixing with 0x
<end_addr>: optionally end address to do a bulk memory modify of the same data word
<data>: The data word to write to memory, can be in hex by prefixing with 0x

For both “Initialize” and “Run” command boxes you can have 1 or more files that contain the gdb commands (or even a combination of files and commands). Use the “Add gdb source command file” buttons to source gdb command files. Use the “Browse Workspace...” button to select a file within your workspace. If the file is outside of the workspace use the “Browse File System..” button or the user can use the “Variables..” button to use one of the in build Eclipse variables.

3. The simulator profiler group allows the user to enable the profiler and specify the output file name. The profiler can only be used when running the simulator, SIMAPI GDB Server External tool, not when running on real hardware. The output file is stored in the project folder. When using the profiler the user must terminate the debug connection first before terminating the simulator so that the results can be flushed to the file.

3.9.4 Specifying the Location of Source Files

You can specify the locations of source files used when debugging a C application. By default, this information is taken from your project. To specify the locations of source file select the **Source** tab.

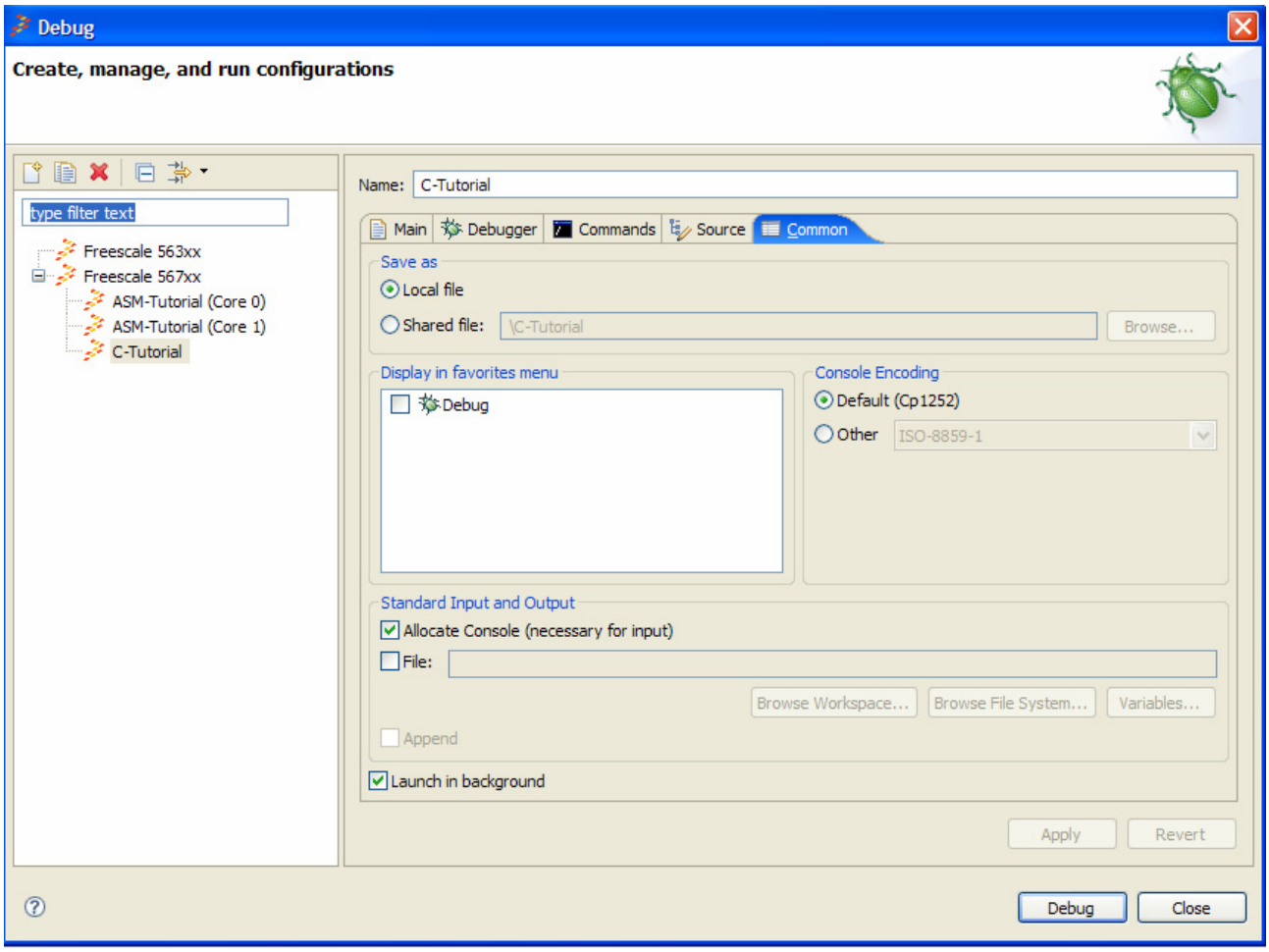


1. To add a source container to the source locations list:
   - Click **Add** to open the **Add Source** dialog box.
   - Select a container type.
   - Select a container from the list of available containers of the selected type.
2. You can remove or modify a source container by selecting a container and clicking the **Remove** or **Edit** button.
3. You can change the order of source containers by selecting a container and clicking the **Up** and **Down** buttons.

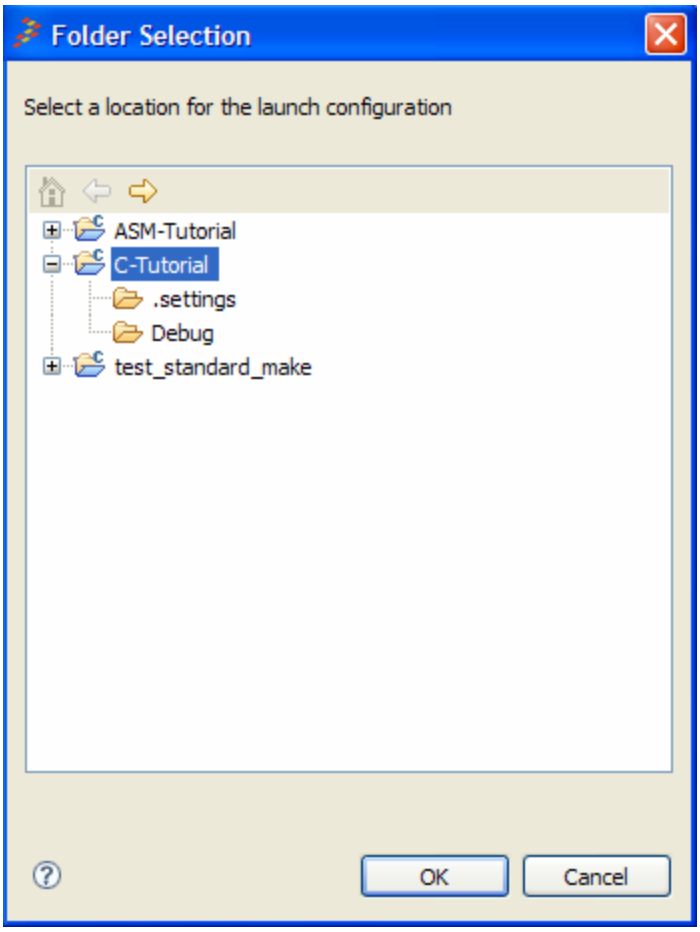
4. To search for duplications in your source locations select the **Search for duplicate source files on path** check box.

3.9.5 Specifying the Location of the Debug Configuration

When you create a debug configuration, it is saved with the extension `.launch` in `org.eclipse.debug.core`. You can specify an alternate location in which to store your debug configuration. To specify the location of the debug configuration select the **Command** tab.



1. To save `.launch` to a project in your workspace, and to be able to commit it to CVS, click **Shared file**.
2. Click *Browse* and in the **Folder Selection** window, select a project, and click **OK**.



3.10 Device Configuration File

A Device configuration file is used by GDB to know the device's peripheral registers and the device's memory map. The configuration file is a XML document.

3.10.1 Register Groups

Peripheral register groups are defined as follows:

<reggroups>

groups...

</reggroups>

There can be multiple groups and each group is defined as

<reggroup name=”name” base=”address” memspace=”P|X|Y” size=”size”>

regs...

</reggroup>

There can be multiple regs and each reg is defined as

<reg name=”name” offset=”offset” />

The device's peripheral registers are displayed in the Register View.

3.10.2 Memory Map

The memory map is defined as follows:

<memory-map>

region...

</memory-map>

There can be multiple regions and each region can be either:

A region of RAM starting at addr and extending for length bytes from there:

<memory type="ram" start="addr" length="length"/>

A region of read-only memory:

<memory type="rom" start="addr" length="length"/>

A region of flash memory, with erasure blocks blocksize bytes in length:

<memory type="flash" start="addr" length="length">

<property name="blocksize">blocksize</property>

</memory>

The memory map is used so that any breakpoints in read only memory are implemented as hardware breakpoints. If this feature is not desired it can be overridden by using the following gdb command in the *“initialize” commands* in the command tab in the debug connection:

set breakpoint auto-hw 0

3.10.3 Example Configuration File

```
<reggroups>
   <reggroup name="cim"       base="0xfffff0" memspace="X" size="0x10" >
      <reg name="ogdb"  offset="0x0c" />
      <reg name="dmas"  offset="0x08" />
      <reg name="coidr" offset="0x06" />
      <reg name="chidr" offset="0x05" />
   </reggroup>

   <reggroup name="cgm"       base="0xffff7c" memspace="X" size="0x4"  >
      <reg name="ascdr"  offset="0x02" />
      <reg name="pctl"   offset="0x01" />
      <reg name="spena"  offset="0x00" />
   </reggroup>
</reggroups>

<memory-map>
  <memory start="0x0" length="0x2000" memspace="P" type="ram" />
  <memory start="0x40000" length="0x1000" memspace="P" type="flash" >
       <property name="blocksize">0x100</property>
```

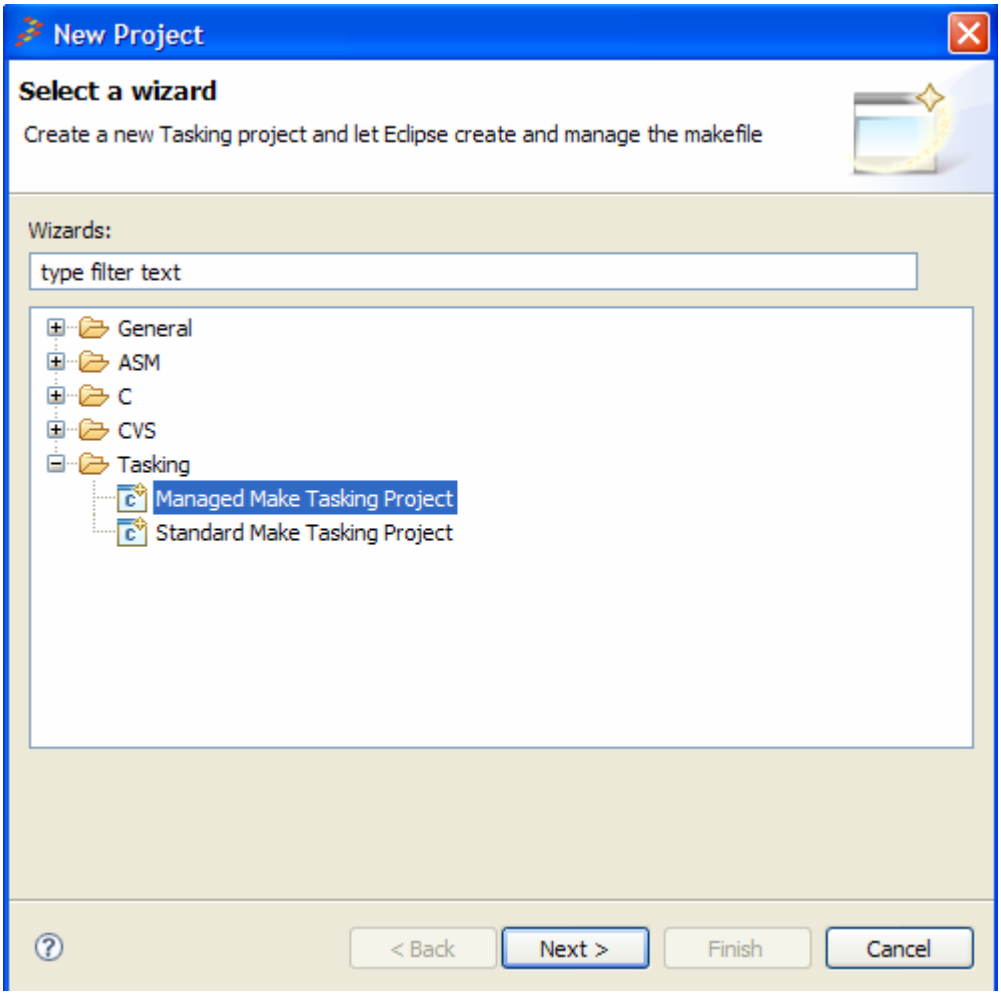
```
</memory>
<memory start="0xfce000" length="0x32000" memspace="P" type="rom"/>
</memory-map>
```

3.11 Tasking Projects

This section describes the specific steps for tasking projects.

3.11.1 Creating a Tasking Project

To create a Tasking C project, instead of selecting a Managed Make C Project under the C Folder select *Managed Make Tasking Project* under the Tasking folder.



TIP

In the C/C++ Perspective you can quickly open a new Managed Make Tasking Project by right click in the C/C++ Projects View and selecting New->Managed Make Tasking Project.

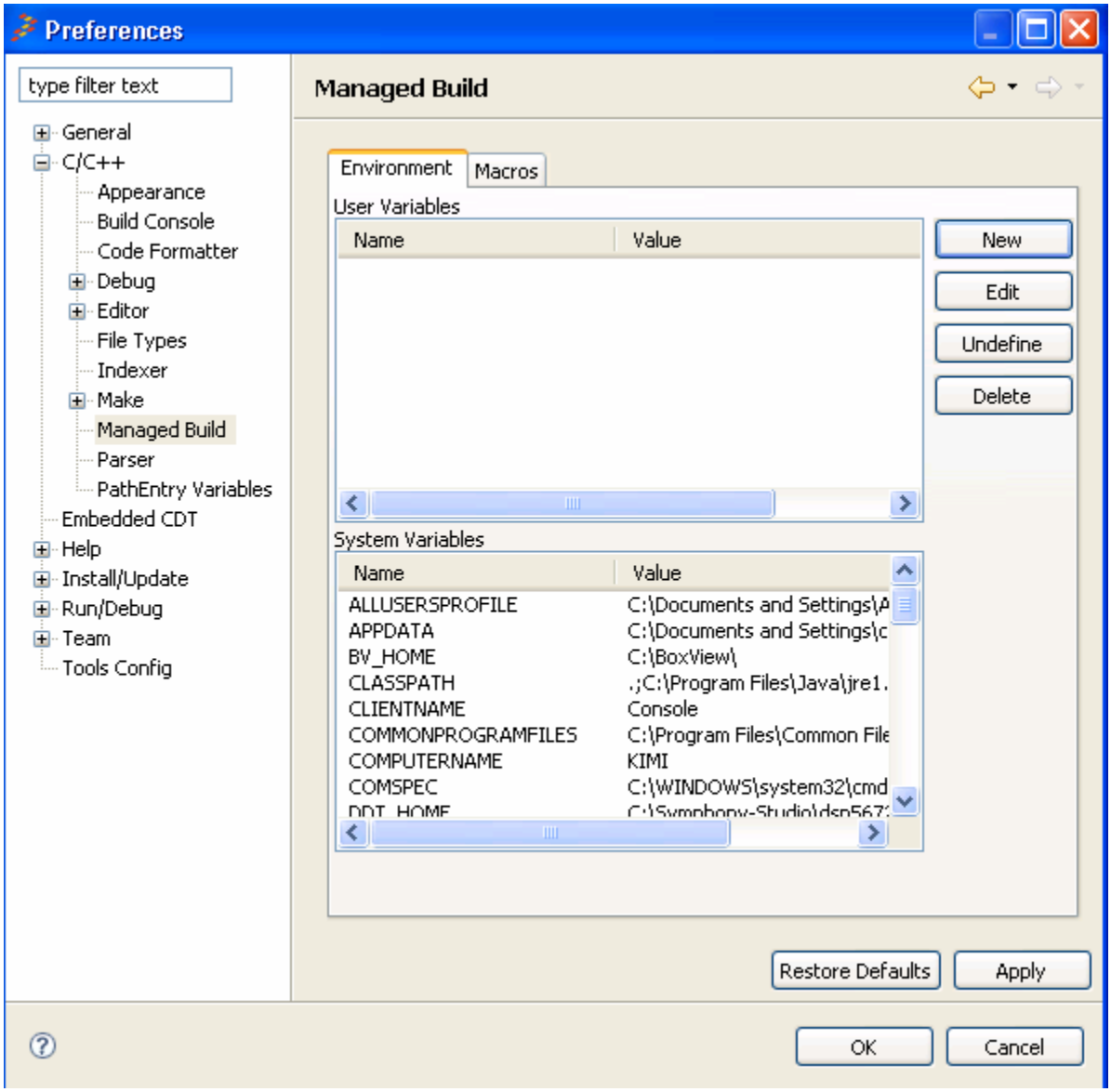
The tool chain for managed Tasking projects uses the Tasking C compiler and Tasking ASM optimizer to generate the assembler files and then to assemble and link the files using the gcc tools:

Such as, Tasking C compiler -> Tasking ASM Optimizer -> GCC Assembler -> GCC linker

3.11.2 Tasking Setup

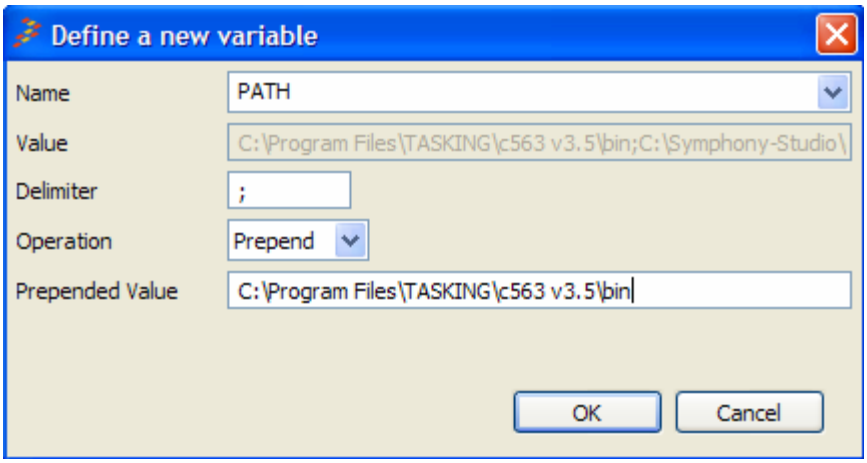
Symphony Studio assumes the Tasking tools are on your PATH environment variable. If this has not happen when installing the Tasking tools you can use Symphony Studio to add it to your path. To use Symphony Studio go to **Window->Preferences**

1. Expand C/C++ on the left pane
2. Select Managed Build under C/C++



3. In the Environment Tab in the right pane, select the “New” button on the right
   This pops up “Define a new variable” window
4. Select the down arrow in the name text box and choose PATH in the pull down menu
   The current contents of PATH should populate the Value text box
5. Enter ; in the Delimiter box
6. Select the down arrow in the operation text box and choose prepend
   This adds a Prepended Value text field and grey out the Value text box above
7. Enter in the bin directory for the Tasking tools in the Prepended Value text field.

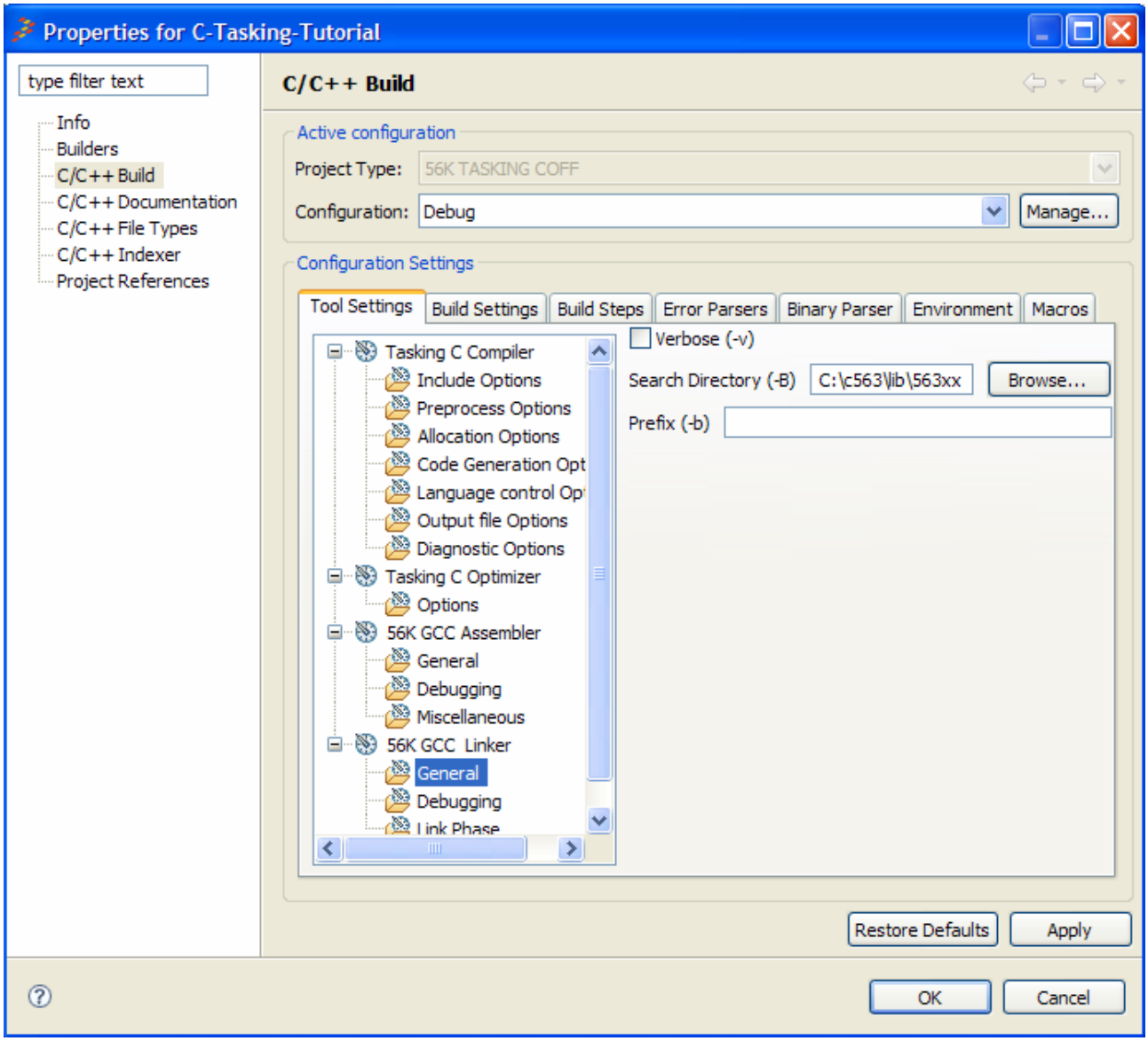
8. Select OK and select OK again.



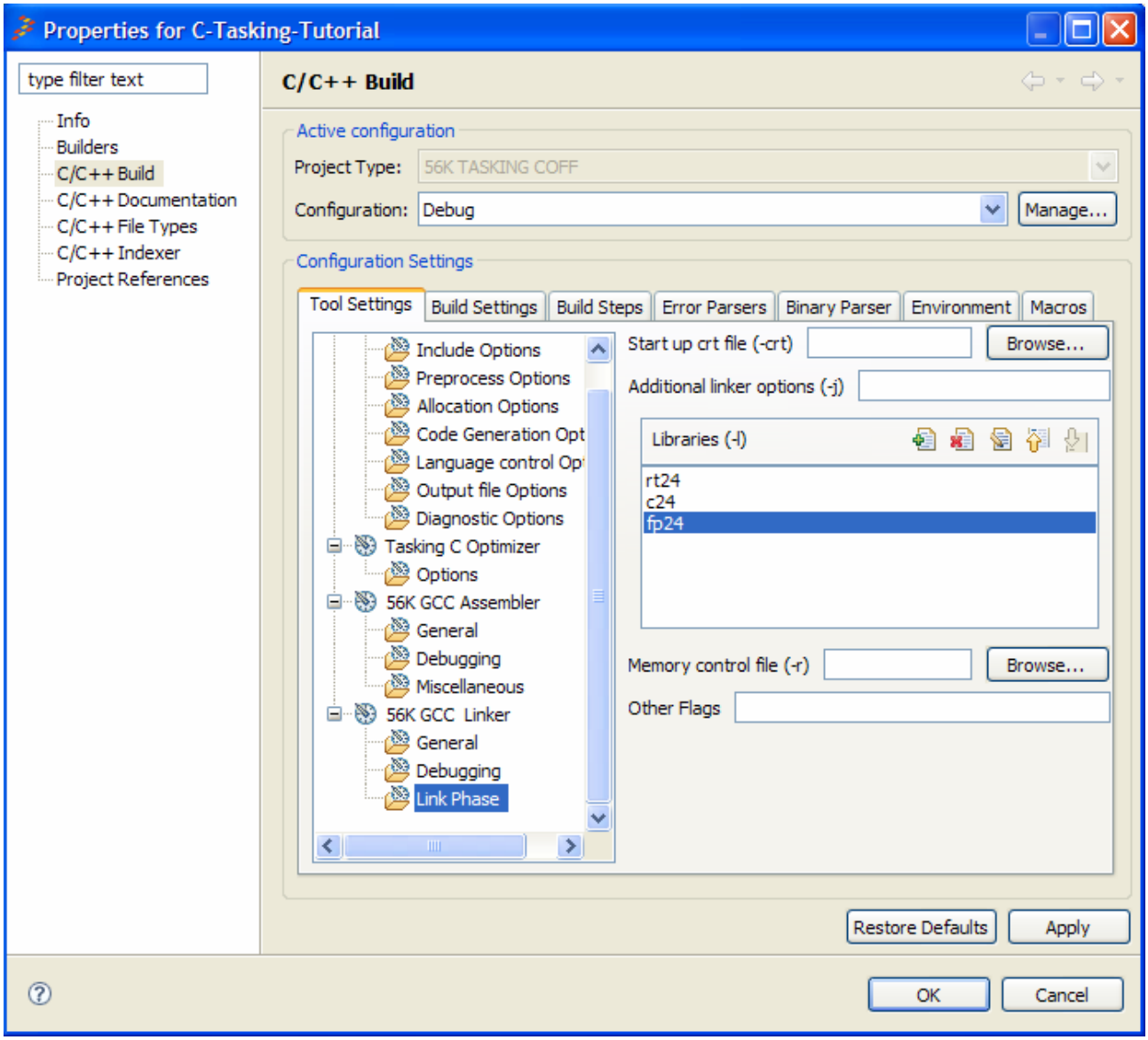
3.11.3 Adding Tasking Libraries

To add Tasking libraries, you first need to know the directory where Tasking has stored its libraries. You then add this directory to the linker search directory and add the libraries you wish to link with. You need to do this if you wish to compile the tutorial project to access Tasking's divide function.

1. Select the project, right click and go to properties.
2. Select "C/C++ Build" on the left pane and select the "Tools Settings" tab on the right pane.
3. Under the "56K GCC Linker" tool select the "general" category and use the browse button to add the library path to the -B option.



4. Then under the “link phase” category, add rt24,c24 and fp24 to the libraries.

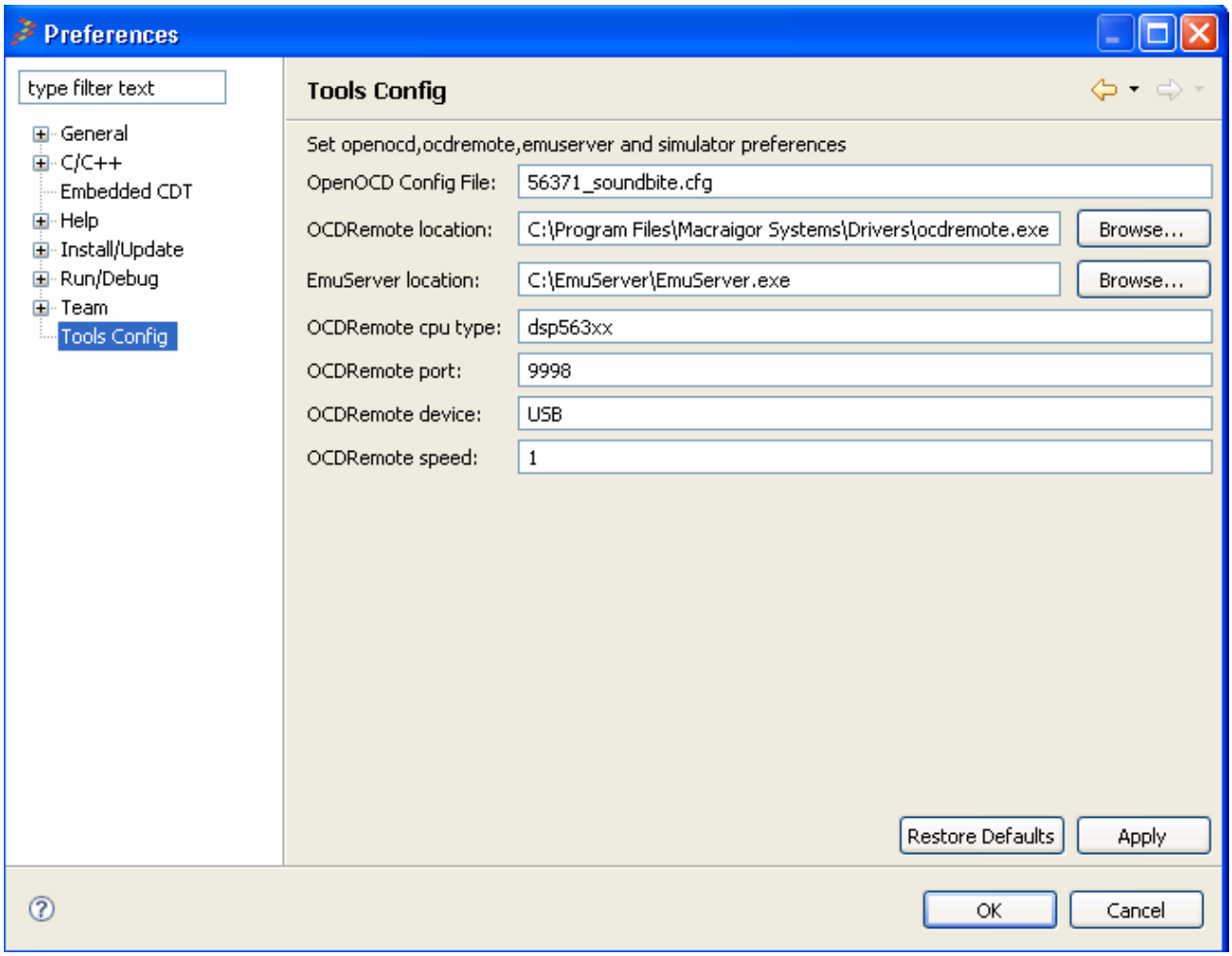


For more information on Tasking tools please visit:
http://www.tasking.com/products/dsp/dsp56xxx/

3.12 Third Party Tools

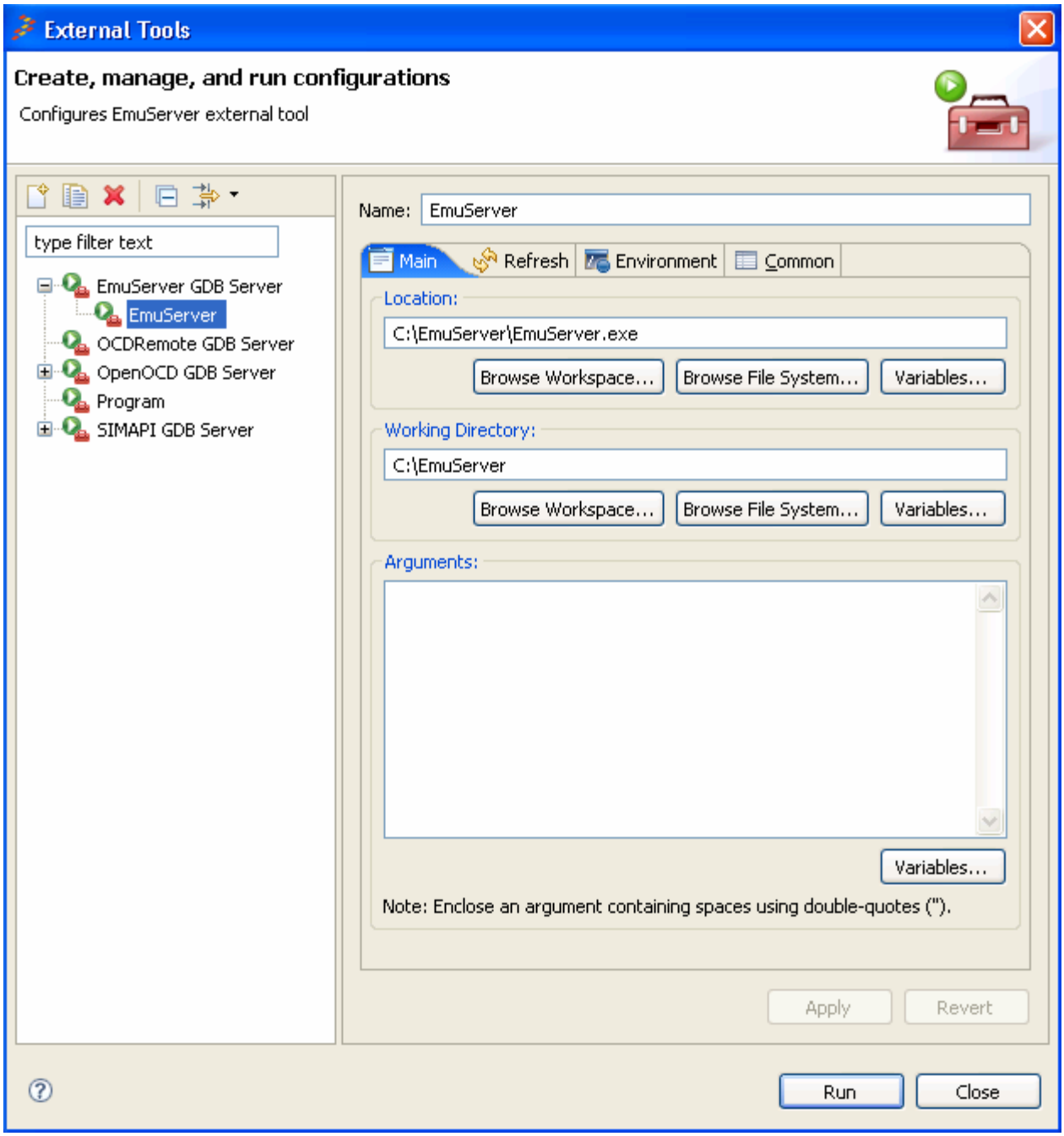
Symphony Studio has support for several third party debuggers. Currently Macraigor's OCDRemote and Domain Technologies EmuServer are supported. Both these tools act as remote gdb servers so that Symphony Studio can talk to each company's debug dongles. Templates are provided for both tools so that they can be run as external tools within Symphony Studio. Before these templates can be used they must be configured. To do this, go to Window->Preferences and

select Tools Config in the left pane. On the right pane, using the Browse button specify the location for either OCDRemote or EmuServer. If OCDRemote is to be used the user must also specify the cpu type, port number, device and speed. Default values are given for each which should work for USB Macraigor dongles for a single core dsp. If using the 56720 dual core dsp please change the cpu type to dsp563xx, dsp563xx



3.12.1 EmuServer

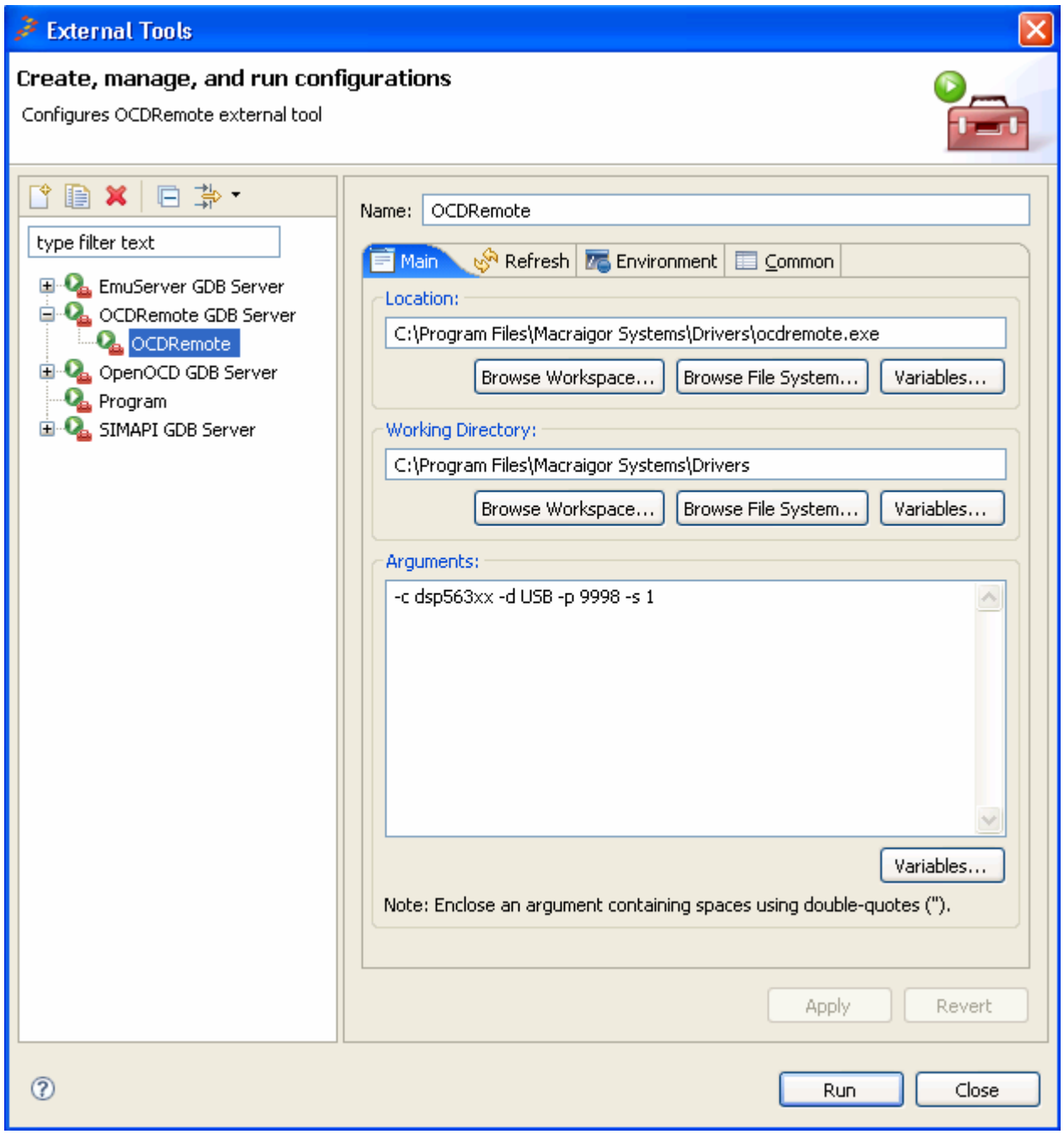
To run Domain Technologies EmuServer select **Run -> External Tools -> External Tools**. Right-click *EmuServer GDB Server* in the side pane to create a new configuration. Use the default values and **Run**



For more information on Domain Technologies, visit http://www.domaintec.com/

3.12.2 OCDRemote

To run Macraigor's OCDRemote select **Run -> External Tools -> External Tools**. Right-click OCDRemote *GDB Server* in the side pane to create a new configuration. Extra parameters may be needed depending on your hardware configuration. These can be added in the “Arguments:” box and then select **Run**



For more information on OCDRemote, visit
http://www.macraigor.com/full_gnu.htm#OcdRemote

Chapter 4
Troubleshooting

This chapter covers the main issues you may experience using the *Eclipse for DSP56720 Platform* and how to fix them.

4.1 Troubleshooting Eclipse

1. While debugging with the EVB if strange behavior is observed such as the core(s) stepping randomly or inconsistencies in the memory it is recommend to try and slow down the JTAG clock. Examples of this would be if the disassembly window suggests the next instruction is a non jump and the core executes a random jump or if the memory render window highlights changing words which should not be. Usually inconsistencies in the memory occur with words changing by a single bit and then changing back to their original value. To decrease the JTAG clock you need to modify the OpenOCD configuration file.

The OpenOCD configuration file is the file that you specify with the -f argument. In this file look for

```
jtag_speed <value>
```

For ftdxxx jtag devices 0 = 6 MHz,1 = 3 MHz, 2 = 1.5 MHz .... For parallel jtag devices the JTAG clock is = (maximum speed)/(value +1) where the maximum speed is approx 1/6 x CPU-Clock.

2. Use the `-loud` argument to the GDB SIMAPI Server (in the External Tools configuration) to get verbose debugging output from the server. This is useful if you are having trouble connecting to the simulator.

3. You may notice that some things do not quite work as you expect. Some of these are known issues. For a complete list of known issues see the README file that comes with your Symphony Studio package.

4. If you have problems launching your configured External Tool (GDB Server), then it is possible that the ports you specified may already be in use by a different application. In this case, please try different non-conflicting port numbers.

5. While Debugging, if a "Source not found" error appears try the following
   - Click on the "Edit source lookup" button in the editor, or right click on the launch node in Debug View and select "Edit source lookup"
   - Click on the "Add..." button

- Select "Path Mapping" and click OK.
- Select the new "Path Mapping" source container and click the "Edit..." button.

— Once again, click the "Add..." button to create a mapping.

- Enter the path to map from. Look at the stack frame label in Debug view, if the filename is something like "/cygdrive/c/workspace/hello/hello.c", enter the path to the first real directory "/cygdrive/c/workspace".
- Enter the correct path to the directory entered above, in the file system. In example above, it would be "C:\workspace".
- Click OK three times and you are back in the Debug view.
- If the source does not show up right away, try stepping once.

NOTE

Please note you can also set the path mapping in **Window -> Preferences -> C/C++ -> Debug -> Common Source Lookup Path**. If you do use the this method to set the path mapping while a debug is in progress you need to re-start the debug for it to take effect.

Appendix A
GDB SIMAPI Server

The following command line options are available from the gdbsimapi server:

```
usage: gdbsimapi [option]
  -port <number>         : Set the listening port (one for each core)
  -showpc                : Display program counter during execution
  -showregs              : Display registers during execution
  -fastmode              : Enable the simulator the leagacy Suite56 fast mode
  -dll <filename>        : Name of SIMAPI dynamic library to load
                         : Default is 'libdsp56720simapi'
  -config <filename>     : Config file to use for DSP56720 simulator
                         : Default is 'default.cnfg'
  -cmdfile <filename>    : Simulator command file input
  -loud                  : Show debugging output
  -h                     : Print this usage string and exit
```

The simulator command file format is described in Appendix B. A command file can easily be added by using the Simulator command file group when creating the SIMAPI GDB Server External tool. The user can browse the current workspace. Browse the file system if the file is outside of the workspace, or use one of the in build Eclipse variables.

NOTE

No spaces are allowed in the path to the command file. Also a carriage return is needed at the end of the file.

The *showpc* argument also in addition to displaying the pc, displays the cycle count and number of instructions executed.

The *config* argument is needed if a different on-chip memory configuration is used. All four configurations as well as the defalut for the 56720 are supported. To change the on-chip memory configuration do the following:

```
for MS 1, MSW1 0, MSW0 0    use     -config dsp56720-MS1-00.cnfg
for MS 1, MSW1 0, MSW0 1    use     -config dsp56720-MS1-01.cnfg
for MS 1, MSW1 1, MSW0 0    use     -config dsp56720-MS1-10.cnfg
for MS 1, MSW1 1, MSW0 1    use     -config dsp56720-MS1-11.cnfg
```

NOTE

In addition to the -config argument, you must also use the gdb command “set $omr=data” in the “initialize” commands, in the command tab in the debug connection, with the appropriate memory configuration bits set.

Appendix B
Simulator Command Files

A simulator command file is a text file with a list of commands to perform on the simulator during startup. Each line of the command file must follow this format:

```
<model name>.<model instance> <command to perform> <parameters>
```

The following models are available in the DSP56720 simulator:

```
core.0   - SONYX core 0
core.1   - SONYX core 1
icc.0    - ICC module
asrc.0   - ASRC module
lbiu.0   - LBIU module
spdif.0  - SPDIF module
memgen.0 - Memory SRAM0
```

Currently only the core and spdif models have supported commands. These commands are listed in the following subsections.

B.1 CORE Commands

INPUT: Assign Input File

```
{I}NPUT [#n] [{T}(timed)] addr/port/periph/pin[_group] file
             [{-RD}/{-RF}/{-RH}/{-RU}]
{I}NPUT [#n] pin [{DVn:}]pin
{I}NPUT [#n] addr [{DVn:}]addr

{input x:$0 xfile}
Get values for memory location x:0 from input file 'xfile.io'.

{input host hfile}
Get values for the Host peripheral from input file 'hfile.io'.  Note
that on core peripherals the name of the peripheral is used, but for
shared peripherals (icc, asrc, lbiu, spdif) the keyword 'periph' is used
for this type of input.

{input reset rfile}
Input values for the RESET pin from input file 'rfile.io'.

{input t irqa tfile}
Input time&data pairs from input file 'tfile.tio' for the device IRQA
pin.

{input t x:$ffcf xfile -rd}
Input time and data pairs for memory location X:$FFCF from 'xfile.tio'.
The data portion of the time-data pair is read as a decimal integer.
```

OUTPUT: Assign Output File

```
{O}UTPUT [#n] [{T}] addr/port/pin[_group] file
              [{-RD}/{-RF}/{-RH}/{-RU}/{-RS}] [{-A}/{-O}/{-C}]

{output x:$0 xfile}
Store values written to memory location x:0 in output file 'xfile.io'.

{output host hfile -a}
Store values from the Host peripheral to output file 'hfile.io'.
Append to the file if it already exists.

{output pd0..pd7 adfile}
Store output values for pins PD0 through PD7 to output file 'adfile.io'.

{output rw rwfile -o}
Output values for the RW pin to file 'rwfile.io'.
Overwrite the file if it already exists.

{output t bg bgfile}
Output time and data pairs from the device BG pin to 'bgfile.tio'.

{output t dbus dfile -rf}
Output time and data pairs from Port A data bus to file 'adfile.tio'.
The data portion of the time-data pair is output in floating point.

{output t x:$ffcf xfile -rd}
Output time and data pairs for memory location X:$FFCF to 'xfile.tio'.
The data portion of the time-data pair is output in decimal form.
```

BREAK: Break Command

```
 {B}REAK [{#}bn] [expression] [break_action]
   [break_action] = [{H}(halt)/{In}(increment CNTn)]
```

A breakpoint expression may be any logical expression that is valid for the DSP Macro Assembler. The following is a list of operators that may be used in the breakpoint expression:

```
  {<}   less than                 {&&}    logical 'and'
  {<=}  less than or equal to     {||}    logical 'or'
  {==}  equal to                  {! }    logical 'negate'
  {>=}  greater than or equal to  {& }    bitwise 'and'
  {> }  greater than              {| }    bitwise 'or'
  {!=}  not equal to              {~ }    bitwise one's complement
  {+ }  addition                  {^ }    bitwise 'exclusive or'
  {- }  subtraction               {<<}    shift left
  {/ }  division                  {>>}    shift right

{eof}  Is TRUE if an end-of-file condition occurs in an input file
       assigned to a peripheral or memory location.

 {----------- Breakpoint Actions -----------}
  {H }    Halt execution.  This is the default.
  {In}    Increment counter variable CNTn (n=1/2).


{----------- Examples -----------}
 {break pc>=$500 }
```

```
Halt DSP program simulation and display enabled registers and memory
when the program counter register is greater than or equal to
hexadecimal 500.

{break (lc<10)&&(pc>100)}
Halt if the loop counter is less than 10 and the program counter is
greater than 100.

{break eof}
Halt if an end of file condition occurs in an assigned peripheral input
file.

{break r0==r1}
Halt when the value of register r0 equals the value of register r1.
```

B.2 SPDIF Commands

Note that SPDIF currently has a limitation where only 1 input and output file (2 total) can be open at a time.

CHANGE: Change Peripheral Attribute

```
change {tx_freq / devattr / int_addr / chan_ctrl / dma_trigg_mask /
 warnings_break_ctrl} value

{change tx_freq $2}
Change the tx_frequency attribute to HEX value 2

{change devattr $012345}
Change the devattr attribute to HEX value 012345


{change int_addr 0}
Change the int_addr attribute to DEC value 0

{change dma_trigg_mask 1}
Change the dma_trigg_mask attribute to DEC value 1

{change warnings_break_ctrl $0}
Change the warnings_break_ctrl attribute to HEX value 0
```

OUTPUT: Assign Output File

```
output [t] periph/pin pinname/reg regname/mem filename

{output t periph file.out}
Writes timed peripheral output to file.out

{output t pin apin pin.out}
Writes the values of apin (and times it changed) to pin.out

{output reg STL reg.out}
Writes the values of the STL register to the file reg.out
```

INPUT: Assign Input File

```
input [t] periph/pin pinname/reg regname/mem filename

{input t periph file.in}
Reads timed peripheral input from file.in

{input t pin apin pin.in}
Reads timed input for the pin apin from pin.in

{input reg STL reg.in}
Reads input for the register STL from reg.in
```