Eclipse SIB-SDK

Manual

REV. 1.1

EMAC, inc.
EQUIPMENT MONITOR AND CONTROL
2390 EMAC Way, Carbondale, IL 62902
World Wide Web: http://www.emacinc.com
(618) 529-4525 FAX: (618) 457-0110

Table of Contents

Disclaimer

1 Introduction and Installation

Eclipse is a full-featured Open-Source IDE that can organize the development of software for use on EMAC products such as the Server In a Box (SIB), Single Board Server (SBS), iPac, and System on Module (SoM) product lines. Eclipse is based on Java, and can be used to develop applications in many languages and environments.

This document will detail the steps of installing and configuring Eclipse in a Linux environment, compiling and running applications, and remote debugging using GDB and GDBServer both on the Linux shell and through Eclipse. This document refers to Eclipse version 3.2 with Java 2 version 1.5.0 running on a Debian GNU/Linux system, but is not Debian specific and should describe operation on other versions of Linux as well; it assumes some familiarity with the Linux environment and file system.

1.1 Installing Java

Because Eclipse depends on Java, ensure that a suitable version of Java has been installed before using Eclipse. Release 3.2 of Eclipse requires at least Java 2 version 1.4.2. EMAC has tested versions up to 1.5.0_09 at the time of this writing, although newer versions of Java should be compatible.

Minimally, a Java Runtime Environment (JRE) is required for Eclipse. However, a Java Software Development Kit (SDK) may be required depending on your intended use of Eclipse. See the Eclipse page at http://www.eclipse.org/ and the Sun Java page at http://sun.java.com/ to determine which Java package best suits your needs.

Before downloading and installing a new version of Java, check to see if a suitable version is already present on the machine. The easiest way to do this is by issuing the command `java -version` at the shell. It is important to note that the GNU interpreter for Java bytecode (GIJ) was not fully compatible with Eclipse at the time of testing.

There are several ways to install a new version of Java. For Eclipse compatibility, EMAC recommends that you install version 1.4.2 or newer directly from Sun Microsystems at http://sun.java.com/. A redistributable version of the JRE is located on the CD under \Development_Kits \EMAC_Open_Tools \Linux which is known to work with the Eclipse version contained on the CD. Download the package required for your platform, unpack it, and place a link to the executable in your path if necessary. Refer to the Java installation instructions available on the Java website and in the downloaded package for more information on installing Java.

Finally, test your Java installation with the `java -version` command to ensure that it has been installed properly.

1.2 Installing Eclipse

Following a successful Java installation, the Eclipse IDE must be installed on your machine. EMAC recommends that you install version 3.2, the most recent stable version as of this testing. This will prevent problems due to untested versions and ensure that the operation of Eclipse matches the instructions in this document. Versions earlier than Eclipse 3.2 are not compatible with the EMAC SDK's.

Eclipse 3.2 is available on the distribution CD under \Development_Kits \EMAC_Open_Tools \Linux directory or you may download version 3.2 (or the latest tested version) of Eclipse from the EMAC FTP site at ftp://ftp.emacinc.com/PCSBC/Development_Kits/EMAC_Open_Tools/Linux/ directory, or directly from Eclipse at http://download.eclipse.org/eclipse/downloads/. After downloading the package, unpack the contents of the gzipped tar file to the desired location.

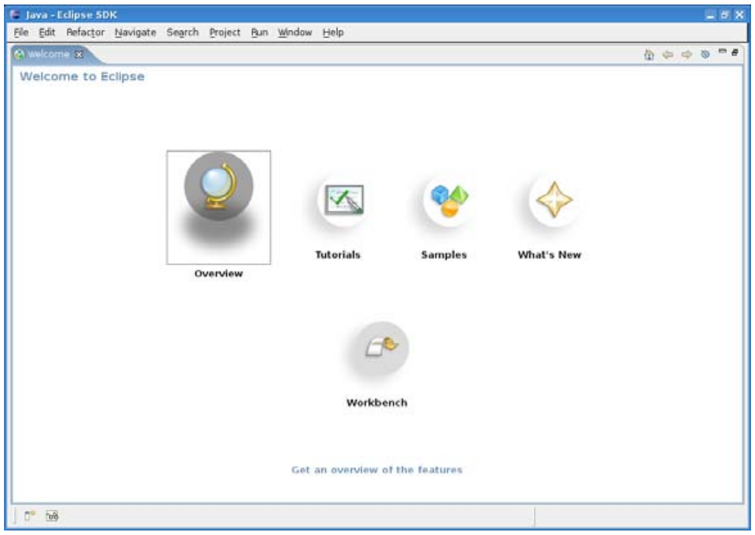
Note that some Linux distributions are releasing Eclipse packages as well. If you decide to use one of these packages, be aware that the installation procedure will differ slightly from this document. Also, ensure that the correct version of Eclipse is installed.

After installation is complete, navigate to the top level of the Eclipse directory. The compiled Eclipse executable is named `eclipse`. Executing this file will start the Eclipse IDE. For ease of use, place a link to this file somewhere in your path or add the Eclipse directory to your path. See the Eclipse documentation in the package and online for more information about command line arguments and other options.

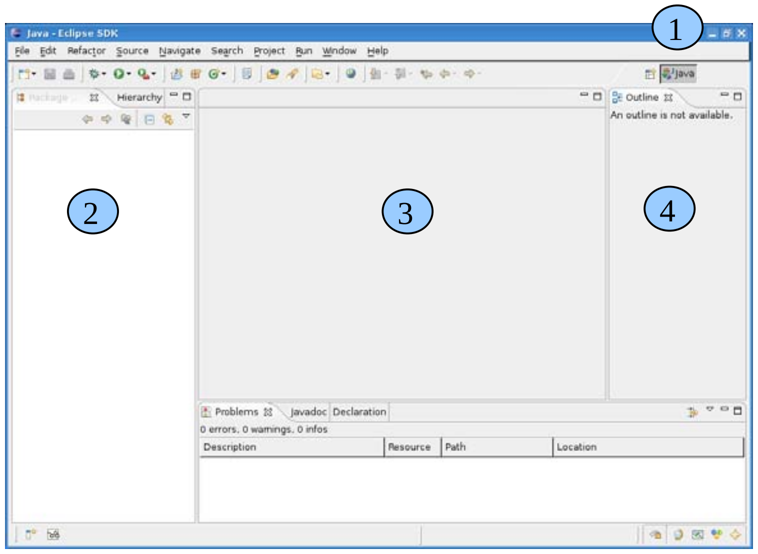
1.3 Initial Configuration

After installing Eclipse and running it, you will be prompted for a workspace location. Eclipse uses this location to store projects and save information about the state of your projects from previous sessions. Because Eclipse stores this data in the actual workspace, the workspace location must be a valid Eclipse workspace or a path for Eclipse to create a new workspace. Eclipse will not recognize a folder without this data as a valid project. Choose a workspace location and continue.

Once Eclipse loads, you should see the Eclipse welcome screen, similar to the one below. Navigate through the various options in the top row to become more familiar with Eclipse and its operation.


Java - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
Welcome
Welcome to Eclipse
Overview
Tutorials
Samples
What's New
Workbench
Get an overview of the features

The *Workbench* link on the welcome screen will open the Eclipse Java development perspective, shown below. The *Java* tab (1) in the upper right hand corner indicates the current perspective. The far left window (2) is used to navigate the current workspace and project, the center window (3) will display any open files, and the far right window (4) will display an outline of the current file or executable. Program output, errors, and other information are displayed in the window at the bottom center of the screen (5).

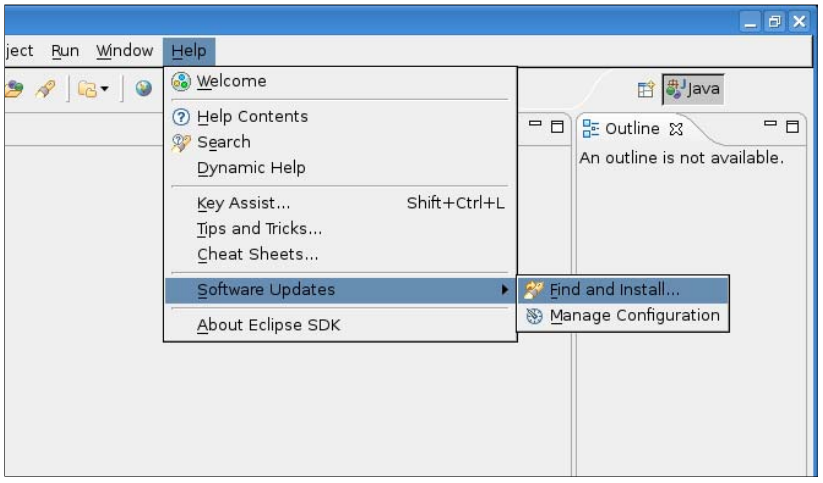


An outline is not available.

0 errors, 0 warnings, 0 infos

1.4 Installing the CDT Plugin

In order to develop applications in C or C++ through Eclipse, you must first install the C Development Tools Plugin (CDT). The utility to locate and install new features in Eclipse is located under the *Help* menu. Select *Software Updates* and then *Find and Install*, as shown below.



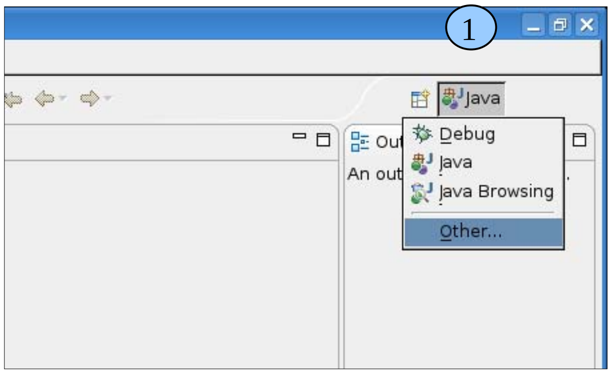
An outline is not available.

This will bring up a new window, which prompts for the type of features to search for. Select *Search for new features to install* and click on *Next.*

The next screen is important because it determines which sites to search for features. Select the *Callisto Discovery Site*, and click on *Finish*. Eclipse will begin to search for updates, and may prompt you to select the mirror to download from. Select a mirror and wait for searching to complete.

Once Eclipse has determined which features are available for your version, it will list them on a new screen. Expand the *Callisto Discovery Site* list, check the box next to *C/C++ Development Tools*, and click *Next*. You will be prompted to accept a license agreement for the new features. Following this screen, check to ensure that the download location is correct, and click *Finish*. Answer any necessary questions during installation. After the tools have downloaded and installed, you will be prompted to restart Eclipse for the changes to take effect.

When Eclipse restarts, navigate to the Workbench and attempt to change your Eclipse perspective to C/C++ by clicking the icon (1) next to the *Java* tab in the upper right hand corner and choosing *Other*, as shown below.



This will bring up the *Open Perspective* screen; choose *C/C++* and click *OK*. Eclipse should now switch to the C/C++ perspective, changing the windows accordingly.

1.5 GCC Versions

In order to develop applications through Eclipse, you will first need a compiler. Different versions and configurations of the GNU Compiler Collection (GCC) are required depending on the platform that you are developing for. EMAC provides software development kits (SDKs) for these versions of GCC, called cross-compilers, and other tools through the EMAC FTP site as well as on the CD-ROM included with your product.

If you are developing for the EMAC SoM-5282, look on the CD under \SoM \SoM-5282M \Tools. Run the install script `m68k-elf-tools-20031003` (or newer version). Installation the executable should be located in the directory `/usr/local/bin/`. After installing, extract the *SoM5282EM-SDK* from the *Tools* directory directly into your Eclipse workspace.

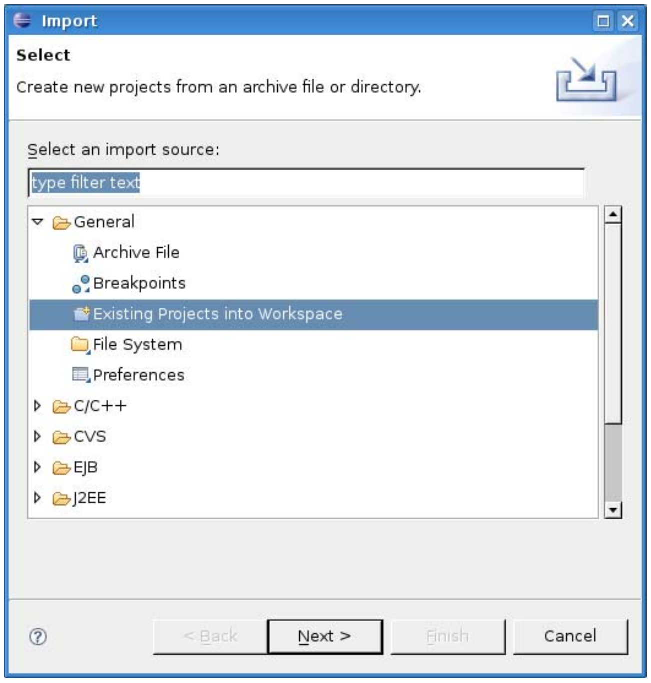
For the EMAC SIB/SBS, the SDK is located on the CD under \EMAC_Linux \SIB \EMAC_Linux_3 \Eclipse_SIB-SDK. Extract *SIB-SDK-2.2* (or newer version) directly into your Eclipse workspace.

Although they handle libraries differently, the required libraries for the SoM and SIB/SBS are built into the cross-compilers to ensure that the correct versions of these libraries are used. For the SIB/SBS, these are located under the `SIB-SDK-2.2/gcc-4.0.0-i486-D/sysroot` directory. For the SoM, the installer places these libraries in the `/usr/local/m68k-elf` directory on the host development machine. To ensure compatibility, it is important that the correct libraries are used.

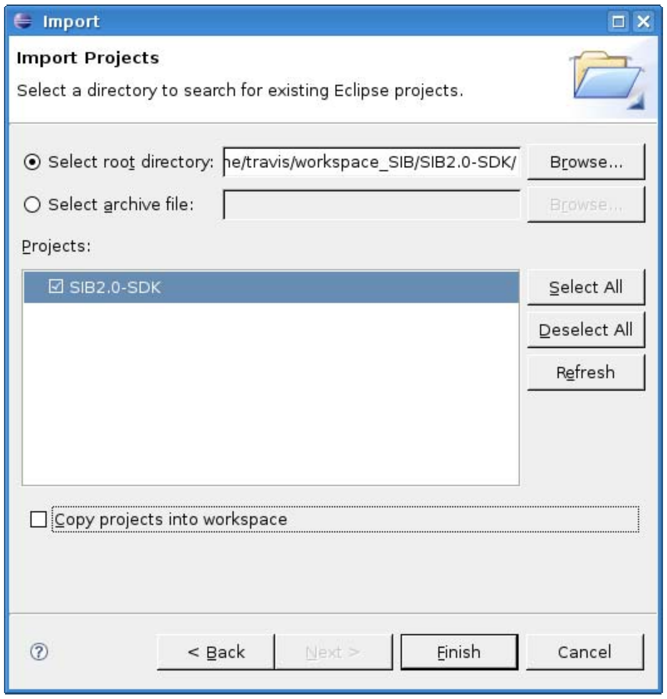
1.6 Importing and Creating Projects

Having installed the CDT Plugin and the necessary cross GCC version, you are ready to develop projects in C or C++ for use on the SoM or SIB/SBS. EMAC provides sample Eclipse projects that will help to familiarize you with the Eclipse development process. These projects are located under the provided Software Development Kit (SDK) for your device as described in section 2.5.

Each EMAC SDK is a valid Eclipse project, and may be imported directly into the Eclipse workspace. First, switch to the C/C++ perspective if necessary. Right click in the white area of the *C/C++ Projects* window in the far left of the screen and select *Import*. This will bring up a new window to specify the import type and location. Expand the *General* list and select *Existing Projects into Workspace*, as shown below.



After clicking *Next*, you will be prompted for the root directory of the project to import. Select the appropriate SDK directory and ensure that it is selected under the *Projects* section. To preserve symbolic links, make sure that the SDK is in your Eclipse workspace and that the *Copy projects into workspace* option is not checked before proceeding.


Import

Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory: he/travis/workspace_SIB/SIB2.0-SDK/

Select archive file:

Projects:

☑ SIB2.0-SDK

☐ Copy projects into workspace

Finally, click on *Finish*, and allow Eclipse to import the project.

When you are ready to develop your own applications, you may create a new project in the existing workspace by right clicking on the white space in the *C/C++ Projects* window and selecting *New*. Selecting *Project* will open the project wizard and allow you to customize the project. Create a new C or C++ *Standard Make Project.*

Once you have successfully imported a project, browse the projects stored in the *projects* directory. Double click on a file to open it with the Eclipse editor. Notice the syntax highlighting and outline that Eclipse provides automatically.

2 Application Development

Developing applications through Eclipse is efficient and simple once you become familiar with the process. Compiling and building applications through Eclipse relies on makefiles and the `make` command, which simplify the use of different GCC versions when compiling for a certain platform. This section will detail the process of compiling and running an application through Eclipse.

Before continuing, ensure that both the Make package and GCC have been installed on your host development machine. For more information on installing these packages, see the links in section 5.

2.1 Makefiles

The projects in the EMAC SDK provide makefiles with targets for compiling, cleaning, and uploading a given application. These makefiles can be easily modified for a new application. To locate a makefile, navigate to one of the projects in the *projects* directory within the appropriate SDK. Simply double click on the file titled *Makefile* (note the capital *M*) to open it with the Eclipse editor.

Notice that the correct GCC and compiler flags have been written into the makefile. Towards the end of the makefile, you will notice several headings followed by indented lines. Each of these headings is called a target, and performs a specific function.

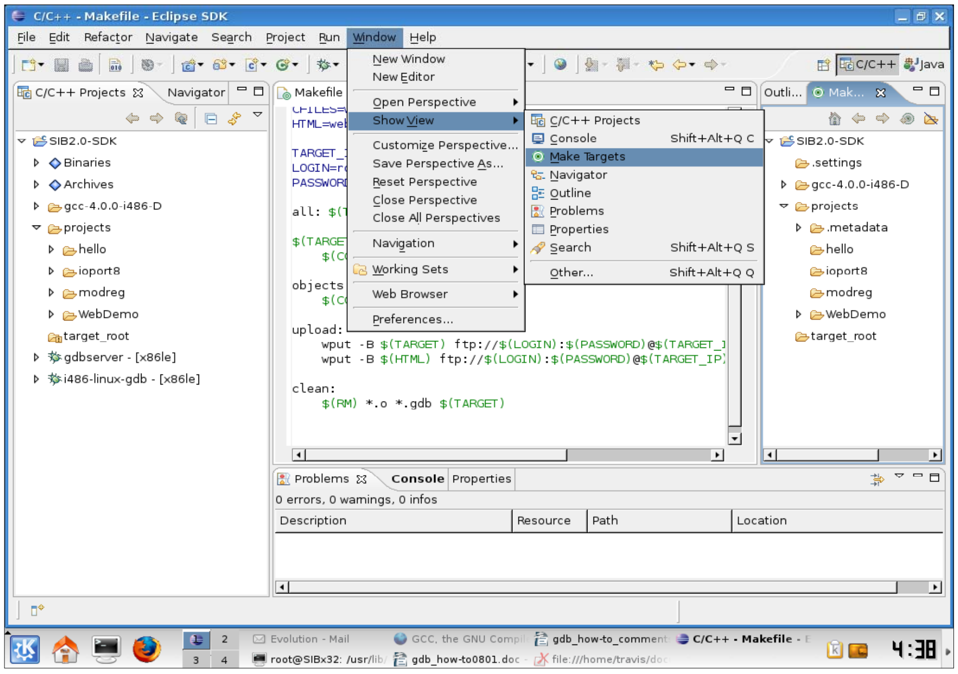
The `all` target will compile all of the specified source code for the given project using the compiler and compiler flags specified in the makefile. The `clean` target will delete all files created from the `all` target, if they exist. The `upload` target will transfer the specified files to the device as specified by the `TARGET_IP`, `LOGIN`, and `PASSWORD` entries in the makefile. Editing the `wput` commands will change the location of the files on the target board.

From the shell on the host development machine, issuing the command `make all`, `make clean`, or `make upload` would execute all of the commands in the respective target. For example, issuing the command `make all` followed by `make upload` would first compile the specified source files and upload the files to the target board.

For more information on makefiles, see the link in section 5.

2.2 Creating and Using Make Targets

Eclipse requires that you create a *Make Target* for each target in the makefile in order to execute the various commands. To do this, first check that your *Make Targets* view is visible by using the *Window* menu and selecting *Show View* followed by *Make Targets*. The *Make Targets* view will be located on the far right of the screen, as shown below.

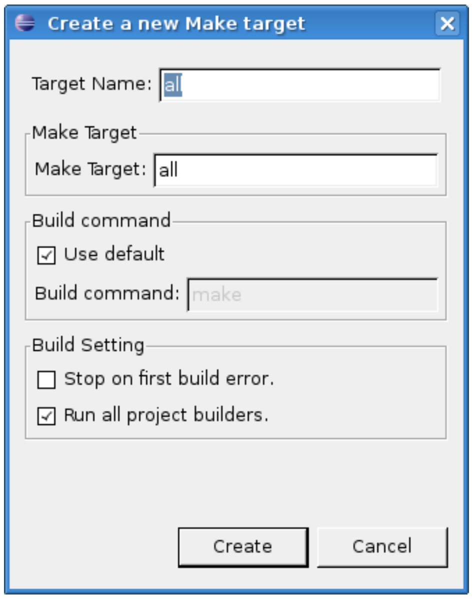


```
CFILES=
HTML=web

TARGET_
LOGIN=r
PASSWORD

all: $(T

$(TARGET
    $(C

objects
    $(C

upload:
    wput -B $(TARGET) ftp://$(LOGIN):$(PASSWORD)@$(TARGET_I
    wput -B $(HTML) ftp://$(LOGIN):$(PASSWORD)@$(TARGET_IP)

clean:
    $(RM) *.o *.gdb $(TARGET)
```

0 errors, 0 warnings, 0 infos

Description	Resource	Path	Location

To create a new *Make Target*, navigate to the project that you would like to create the target for under the *Make Targets* view. Right click on the project and select *Add Make Target*. This will bring up a new window that will specify the name for the target and the command to call. An example is shown below.



Target Name: all

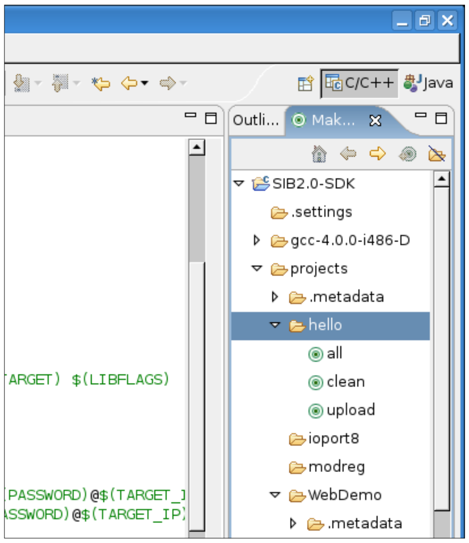
Make Target
Make Target: all

Build command
[✓] Use default
Build command: make

Build Setting
[ ] Stop on first build error.
[✓] Run all project builders.

Target Name specifies the Eclipse name for the target. This can be anything you wish, but it is easiest if the name is the same as the actual target in the makefile. *Make Target* specifies the target to call in the makefile. For example, the window above will create a *Make Target* called *all* that will call the command `make all` when executed.

After creating the Eclipse *Make Targets*, they will become visible under the project in the *Make Targets* view. For instance, after creating targets for `make all`, `make upload`, and `make clean`, Eclipse will display them all as shown below.



Following the creation of an Eclipse *Make Target*, you may execute it by simply double clicking the target in the *Make Targets* view. For example, double clicking *all* in the above screen shot would compile all of the source files specified in the makefile for the `hello` project. The *Console* tab in the bottom center window will show the output of all make commands that are executed.

2.3 Executing Applications

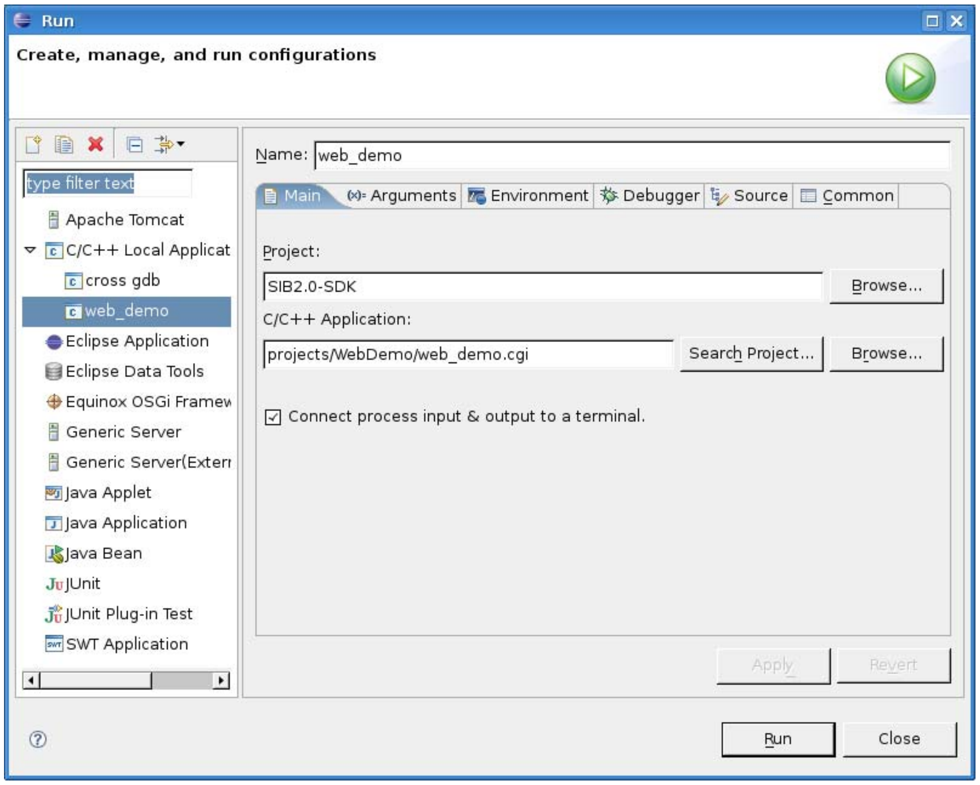
Eclipse can also be used to execute applications on the host development machine. This can be useful when it is practical to test an application or algorithm on the host machine before uploading it to the target board for testing, or when the remote target board is not available for testing. Also, this is valuable when developing companion software to be used in cooperation with the target board.

Eclipse can only execute applications that have been compiled using the compiler for the environment on the host development machine. In other words, Eclipse will not be able to run applications that have been compiled with `m68k-elf-gcc` or other incompatible compilers.

With some applications written for a specific target board, running applications through Eclipse may be as simple as creating a copy of the original makefile and changing the compiler to the host GCC. In cases where the program is more platform or device dependent, the code may need to be broken down and modified considerably before it will compile and execute on the host development machine. In these instances, it will be more advantageous to test applications for the target board after physically uploading them and executing the file, using the remote debugging process described in section 4 to work out any problems with the application.

To create and run applications for use on the host development machine only, simply create a makefile that calls the GCC version on the host system with the appropriate flags. Follow the same process of creating and executing *Make Targets* and makefiles as described in sections 3.1 and 3.2. Also, ensure that GCC has been installed on your system first. For more information on installing GCC, see the links in section 5.

After compiling an application using the Eclipse *Make Targets*, you may run it using the *Run* menu. Select *Run...* and specify how the program should be executed through the configuration screen that Eclipse will bring up. A sample view is shown below.


Run

Create, manage, and run configurations

Name: web_demo

Project: SIB2.0-SDK

C/C++ Application: projects/WebDemo/web_demo.cgi

☑ Connect process input & output to a terminal.

Once run, any program input and output is facilitated through the *Console* tab in the bottom center window of the Eclipse workbench.

3 Remote Debugging

When developing applications for an embedded environment, it is important to be able to debug them remotely, as this is generally difficult on the embedded device itself. The combination of the GNU Debugger (GDB) and its companion server (GDBServer) allow for this to be done quite easily and efficiently. Eclipse integrates remote debugging using this software combination, allowing for an excellent graphical debugging session.

3.1 Compiling for Debugging

In order to facilitate remote debugging, only the executable must be present on the target board. On the host machine, however, the source code for the application must be present, as well as the executable.

The executable on the host machine must have been compiled with debugging symbols included in order for GDB to work properly. This is done with the use of the `-g` option to GCC. Although it is possible to compile debugging symbols into the remote executable as well, this may cause large applications to become too big for the embedded device, and will not provide any benefits. The debugging symbols option should be the only compiler flag that differs when compiling the executables for the target board and the host machine.

It is also important to ensure that no optimization is preformed during compilation, as this will make debugging the code nearly impossible. This may be specified with the `-O0` (capital "O" followed by "zero") option to GCC, but should be implied by default as long as no other optimizations are specified.

Compile and upload the files using the Eclipse *Make Targets* created in section 3.2. You may have to modify the Makefile to reflect the required compiler flags.

3.2 Establishing a Connection

Before continuing to set up the required software for the debugging session, establish and test a connection with the remote target board. Ensuring that a reliable connection has been secured between the host development machine and the target board will make troubleshooting any problems within the debugging session easier.

This connection can be established via an Ethernet or serial port connection. For serial connections, follow the documentation included with your specific target board and use a terminal program such as Minicom to communicate with the device.

For the SIB/SBS, Ethernet connections should be established through the SSH server. This will ensure that all traffic is encrypted. There is also an optional SSH module available for the SoM-5282. Most SoM products do not currently support SSH connections, requiring Ethernet connections to be made using Telnet and FTP.

Currently, there are no public Eclipse plugins to allow Telnet or terminal program sessions to be established cleanly through Eclipse, although support is planned in the near future. For this reason, Telnet and terminal connections must be managed directly from the shell on the host machine. SSH connections, however, can be managed via Eclipse using the Target Management / Remote Systems Explorer Plugin.

The Target Management (TM/RSE) Plugin is currently under development, but has already released several stable versions. At the time of testing, the most recent version was Milestone 4 (1.0M4), with the first official release scheduled for October of 2006. The plugin will allow you to copy and paste files between the target board and the host machine, as well as edit files and issue commands on the target board through a shell window.

To install the Target Management Plugin, follow the same process used in section 2.4 to install the CDT Plugin. From the *Help* menu, go to *Software Updates* and select *Find and Install.* Select *Search for new features to install* and click *Next*. Create a *New Remote Site* named "RSE" with a URL of "http://download.eclipse.org/dsdp/tm/updates/". Check the box next to the RSE site and select *Finish*. Complete the process to install the entire RSE SDK and examples. Restart Eclipse to complete the installation.

Following a successful installation, browse the RSE documentation to familiarize yourself with its operation. This should be accessible through the *Help* menu under *Help Contents*. The *RSE User Guide* link is located in the left pane of the help window.

If you have not already created a new user on your target board, you will need to do this before connecting to the SIB through the Target Management Plugin because Eclipse will not display the menu system on the SIB properly. Connect to the SIB using SSH via the shell and use the menu configuration system to create a new user. You should then be able to login through SSH with the new user name (using a shell command like `ssh new_user@192.168.0.101`). Leave this connection open for now to allow access to the shell on the SIB. You will need to have an SSH client installed on your machine to enable this connection. More information about SSH can be found in the links in section 5.

You may generate a specific SSH key to be used for the newly created user. The following instructions will help you to create a new key to use with the SIB/SBS. To create key pairs for the SoM-5282 with optional SSH, the commands are slightly different. See the documentation that came with your SoM-5282 for more information about how to do this. To create a new key on the SIB/SBS, generate a key pair on the host development machine with the command `ssh-keygen -t dsa`. Using the system defaults, this should create the files `~/.ssh/id_dsa` and `~/.ssh/id_dsa.pub`. The `.pub` file contains the public key, and must be securely copied to the target board. A command similar to `scp ~/.ssh/id_dsa.pub root@192.168.0.101:/home/new_user/` will copy the file to the home directory of the newly created user (replace the IP address and directories with the appropriate values).

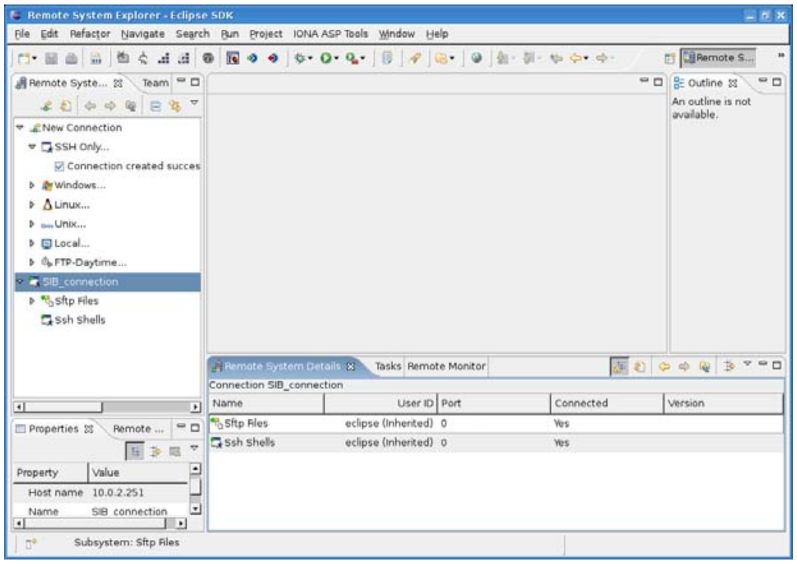
The final step in using the new key is to copy it into the keys file. On the SIB/SBS, if the directory `~/.ssh` and the file `~/.ssh/authorized_keys` do not exist, create them. Next, add the key to the authorized keys file with the command `cat ~/id_dsa.pub >> ~/.ssh/authorized_keys` on the SIB/SBS. Close the SSH session with the SIB/SBS, and attempt to login as the newly created user. The password (if any) used in the creation of the public key will be prompted for before login. Delete the `id_dsa.pub` file in the new user's home directory on the SIB/SBS.

You are now ready to open a connection with the SIB/SBS or the SoM-5282 with optional SSH through the Target Management Plugin. Begin by launching the RSE perspective using the *Window* menu. Select *Open Perspective* followed by *Other*. Select *Remote System Explorer* in the new window and click on *OK*.

When the new perspective opens, you should see the *Remote Systems* tab to the left of the screen. In this window, create a new *SSH Only* connection. This will bring up a new screen that will allow you to configure the connection. Ensure that the host name or IP address of the target board is entered correctly in the *Host Name* field.

The newly created connection should now appear in the *Remote Systems* window on the left of the screen. Monitor this connection with Eclipse by right clicking on the connection and selecting *Show in table*. Also, right click on the connection and select *Monitor*. You should see the connection added to the windows at the bottom center of the screen. Finally, right click on the connection and select *Connect*.

The system should then prompt you for your user name and password. Enter the user name and password for the user created earlier in this section. The resulting configuration should be similar to the following screen shot. You will also want to open the local connection to allow browsing of the local file system and shells. This connection should be opened automatically.



Connection SIB_connection

Name	User ID	Port	Connected	Version
Sftp Files	eclipse (Inherited)	0	Yes	
Ssh Shells	eclipse (Inherited)	0	Yes	

Property	Value
Host name	10.0.2.251
Name	SIB_connection

You may use the *Sftp* files link to browse, edit, copy, and paste files to and from the file system on the target board as if they were located on the development host. From the other perspectives, connections may be monitored and established by activating the various *Remote System Explorer* windows in the current perspective through the *Window* menu under *Show View*. Dragging and dropping files between any of the open windows is also supported.

To open a SSH shell with the target board, right click on *Ssh Shells* and select *Launch Shell*. This will bring up a new view that will allow you to execute commands on the target board. Local shells may also be invoked in the same way through the Local connection.

3.3 GDBServer and GDB Versions

Once a connection has been successfully established with the target board, the required programs for debugging on the target board and local development host must be located and installed. The required GDBServer programs are included with the EMAC Linux distributions for the SoM-5282 and the SIB/SBS. The appropriate version of GDB must be installed on the host machine, however.

For the SIB, GDB is located in the same location as GCC in the SIB-SDK-2.2 directory. To find it, look for `i486-linux-gdb` in `SIB-SDK-2.2/gcc-4.0.0-i486-D/bin`. Place a link to this executable somewhere in your path for easier use.

For the SoM-5282, EMAC provides a different configuration of GDB than the version included with the m68k-elf-tools toolchain to allow for compatibility with Eclipse. This can be located on both the EMAC ftp site and the CD-ROM. Refer to the instructions in the README.htm file within the SoM-5282EM-SDK to build this new version of GDB. Replace the version of `m68k-elf-gdb` installed from the m68k-elf-tools toolchain with this version. It should be located in `/usr/local/bin` on the host development machine.

3.4 Using GDBServer

After compiling and uploading the required programs and locating and installing the appropriate version of GDB for your target board, GDBServer must be invoked on the target board. This will allow the target board to accept incoming connections from GDB on the development machine.

This can be done through a TCP connection or a dedicated serial connection from the host machine to the remote machine. In either case, first navigate to the directory of the executable on the target board and ensure that execute permissions have been enabled for the program (using the `chmod` command). GDBServer should be run from this directory.

The general form of the command to initiate GDBServer is `gdbserver COMM PROGRAM [ARGS]` where `COMM` specifies the connection and `PROGRAM [ARGS]` specifies the program to debug along with any command line arguments that the program should be passed.

When debugging over a TCP connection, `COMM` takes the form of `HOST:PORT` to describe the IP address and port to listen for connections from. For example, if the development host was at IP address 192.168.0.100, and you wanted to debug the `web_demo.cgi` program from the EMAC project examples, you could enter a command like `gdbserver 192.168.0.100:2828 web_demo.cgi` to listen for incoming connections on port 2828. This port number can be anything that you like, as long as it is not a reserved port number. Leaving the `HOST` portion of the command blank, as in `gdbserver :2828 web_demo.cgi` will allow GDBServer to accept connections from any machine with access to it from the network.

Debugging over a serial connection requires that the `COMM` portion of the GDBServer command denote the serial port to listen for connections on. For example, if the host development machine and the target board have a serial connection on port `/dev/ttyS1` on the target board, the command `gdbserver /dev/ttyS1 web_demo.cgi` will allow GDBServer to listen for incoming connections over this serial line.

3.5 Shell Operation

After compiling the executable with debugging symbols for the host machine and starting GDBServer on the target board, you are ready to start debugging using GDB. Because it is easy to start GDB and establish a connection using the shell on the host machine, this is a good way to test your GDB and GDBServer setup before using Eclipse.

From the shell on the development machine, navigate to the directory containing the executable to be debugged as well as all required source code. From here, issue the command for the appropriate version of GDB.

For example, if the executable to be debugged were `web_demo.cgi.gdb`, issue the command `m68k-elf-gdb web_demo.cgi.gdb` or `i486-linux-gdb web_demo.cgi.gdb` depending on the platform you are developing for. This assumes that the GDB executable is located in your path as described in section 4.2.

GDB should start and return a prompt for commands. Next, initiate a connection to the remote machine for debugging by issuing a `target` command. This command will differ according to the type of connection (serial or TCP).

To connect over a TCP connection, you must know the IP address of the target board as well as what port GDBServer is listening for connections on. Following the earlier examples and assuming the IP address of the target board is 192.168.0.101 listening on port 2828, issue the following command at the GDB prompt: `target remote 192.168.0.101:2828` to establish a connection with the target board.

When debugging over a serial connection, the command becomes even simpler. For example, if the target board is connected on the host development machine port `/dev/ttyS1` and GDBServer is listening for connections on the respective serial port on the target board, issue the command `target remote /dev/ttyS1` at the GDB prompt to establish a connection.

After the GDB prompt returns, the shell on the target board should report that a connection has been successfully established. To test your connection, first add a breakpoint in the main function of the program with the command `b main` at the GDB prompt. To execute the code up to the breakpoint, continue by typing `c` at the GDB prompt. The program should advance and stop. Set other breakpoints in the program and step through the application. You may use the `list` command to show the line numbers of surrounding code that you wish to set a breakpoint at.

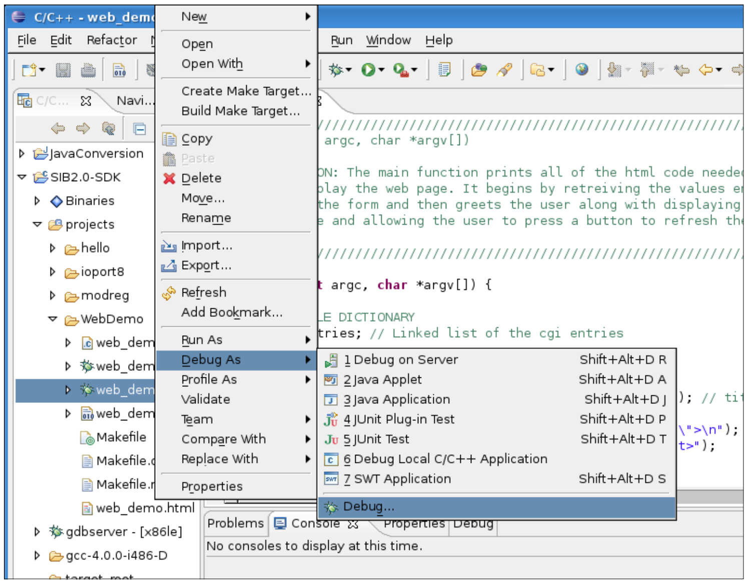
The output of the program will be displayed on the shell of the target board through the output of GDBServer. Any required input for the program will also need to be entered on the target board shell within the running GDBServer.

3.6 Remote Debugging with Eclipse

The next step after successfully debugging an application through the shell using GDB directly is to use Eclipse to set up a remote connection with GDBServer. This will allow for a much more user-friendly interface and make debugging more efficient.

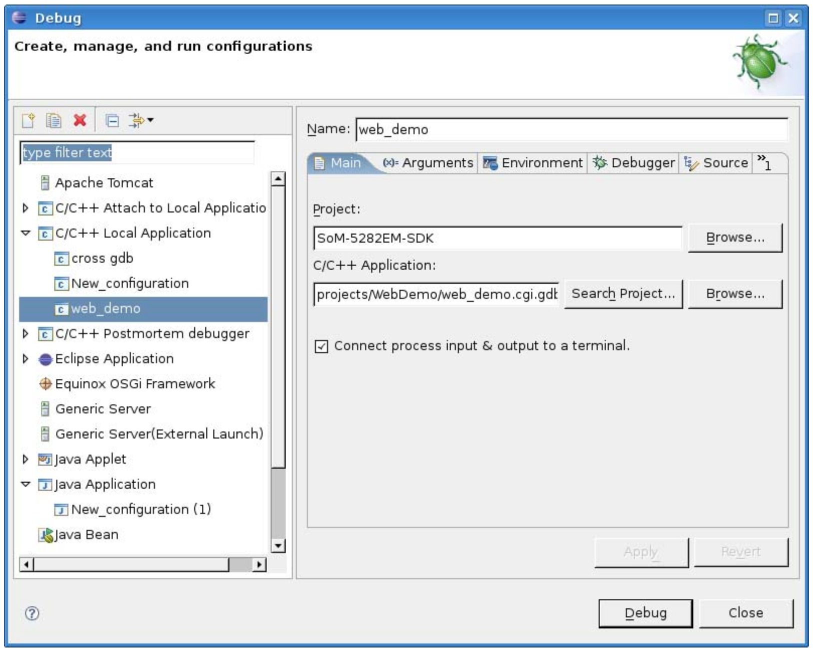
The GDBServer command will be exactly the same on the target board as it was when debugging through the shell. After compiling and uploading the necessary applications, start GDBServer on the target board according to section 4.3.

On the Eclipse workbench, right click on the application that you are trying to debug. Make sure that this file has been compiled with debugging symbols. From the menu that is brought up, select *Debug As* followed by *Debug...* as shown below. This will bring up a new window to allow you to configure the way the program is debugged. This can also be accessed from the *Run* menu.



No consoles to display at this time.

On the new window, select *C/C++ Local Application* on the left and name the configuration on the right side of the screen so that your changes are saved for future sessions. Select the project and application to be debugged if necessary.



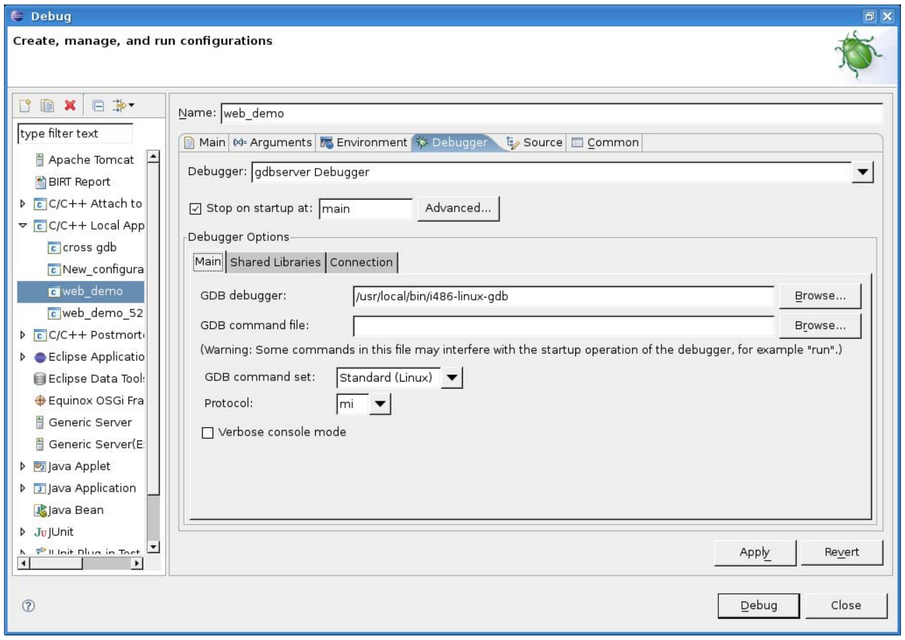
Name: web_demo

Project: SoM-5282EM-SDK

C/C++ Application: projects/WebDemo/web_demo.cgi.gdb

☑ Connect process input & output to a terminal.

The *Debugger* tab on this window is where most of the configuration will take place. Select this tab and choose *gdbserver Debugger* as the debugger. Next, on the *Main* tab under *Debugger Options*, choose the appropriate *GDB Debugger* for your platform as described in section 4.2. Also, set the *GDB command set* to *Standard (Linux)* and the *Protocol* to *mi*. The EMAC versions of GDB are not configured to use a *GDB command file* by default, so this setting is not necessary. The final setup should be similar to the following screen shot.



Name: web_demo

Debugger: gdbserver Debugger

☑ Stop on startup at: main

Debugger Options

Main | Shared Libraries | Connection

GDB debugger: /usr/local/bin/i486-linux-gdb

GDB command file:

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

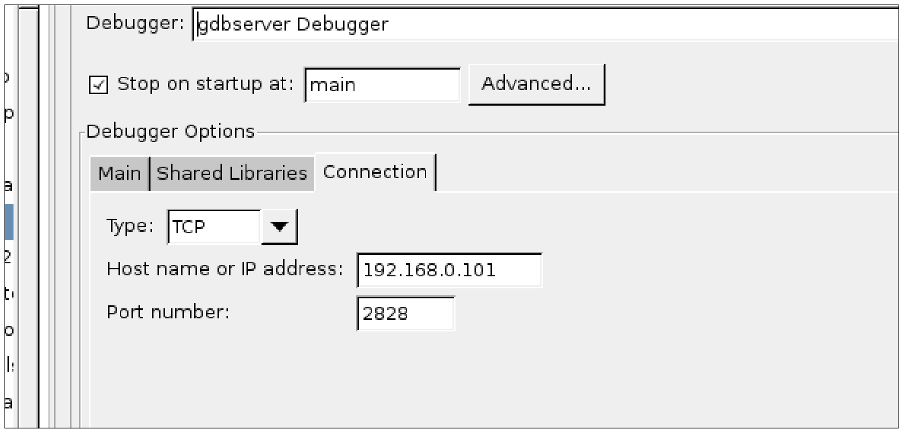
GDB command set: Standard (Linux)

Protocol: mi

☐ Verbose console mode

The *Shared Libraries* tab can be useful in telling GDB exactly where to search for shared libraries that are utilized by your program. If GDB cannot find the required libraries, it will give you errors while executing, and you will need to specify their location directly here.

On the *Connection* tab, you must specify the connection type and parameters for Eclipse to use to locate the target board. For example, to specify a TCP connection to IP address 192.168.0.101 on port 2828, you would use the settings shown below.



Debugger: gdbserver Debugger

☑ Stop on startup at: main

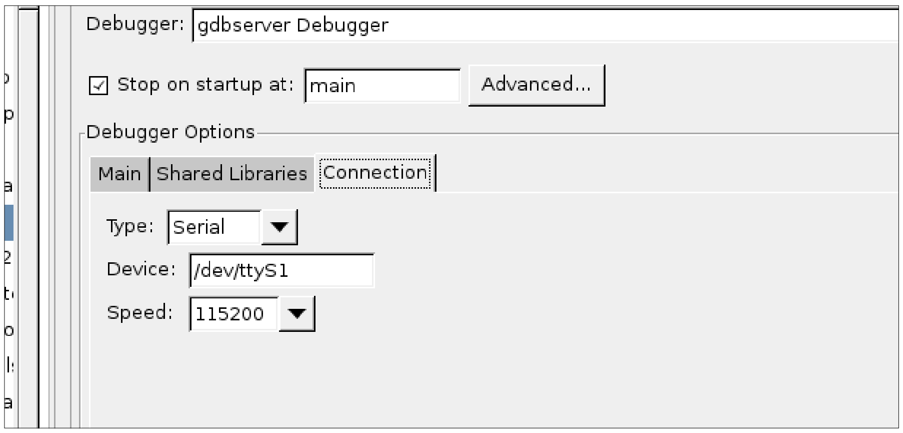
Debugger Options

Type: TCP

Host name or IP address: 192.168.0.101

Port number: 2828

Connection over a serial line requires you to set the device and line speed as follows, according to the settings of your serial connection.



Debugger: gdbserver Debugger

☑ Stop on startup at: main

Debugger Options

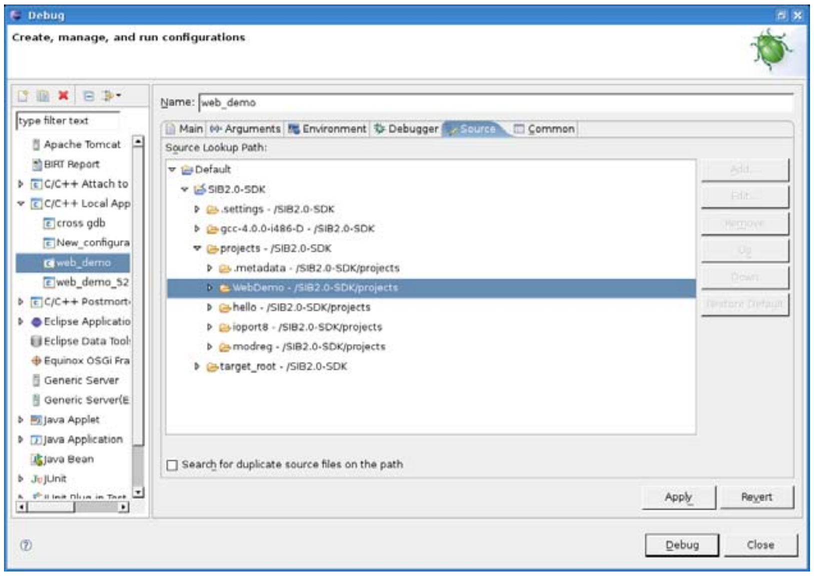
Tabs: Main | Shared Libraries | Connection

Type: Serial

Device: /dev/ttyS1

Speed: 115200

Use the *Source* tab to specify the search path for the program source files. Check to ensure that this is set so that the directory containing the source files for the program is located in the search path as shown below. If not, add this location to the path.



Debug

Create, manage, and run configurations

Name: web_demo

Tabs: Main | Arguments | Environment | Debugger | Source | Common

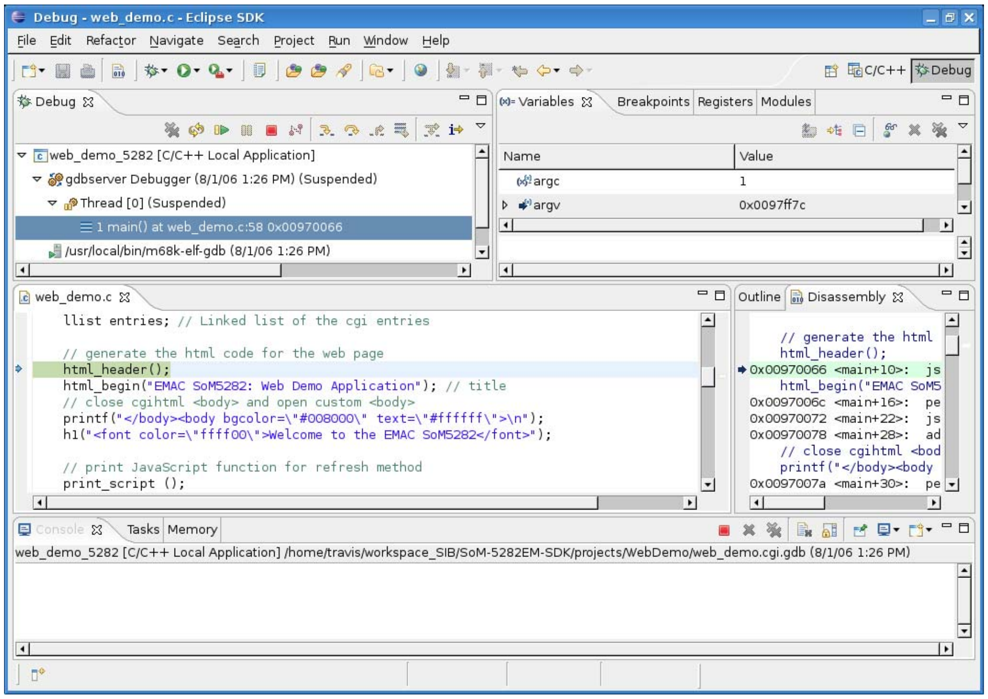
Source Lookup Path:

```
Default
  SIB2.0-SDK
    .settings - /SIB2.0-SDK
    gcc-4.0.0-i486-D - /SIB2.0-SDK
    projects - /SIB2.0-SDK
      .metadata - /SIB2.0-SDK/projects
      WebDemo - /SIB2.0-SDK/projects
      hello - /SIB2.0-SDK/projects
      ioport8 - /SIB2.0-SDK/projects
      modreg - /SIB2.0-SDK/projects
    target_root - /SIB2.0-SDK
```

☐ Search for duplicate source files on the path

When you are finished with the configuration and have started GDBServer on the target board, press *Apply*. Press *Debug* to initiate the connection and begin debugging.

Eclipse should automatically open the *Debug* perspective, locate the source code, and wait for you to debug the program. You should see something similar to the screen shot shown below. The GDBServer running on the shell of the target board should report that a connection has been established.



```
web_demo_5282 [C/C++ Local Application]
  gdbserver Debugger (8/1/06 1:26 PM) (Suspended)
    Thread [0] (Suspended)
      1 main() at web_demo.c:58 0x00970066
  /usr/local/bin/m68k-elf-gdb (8/1/06 1:26 PM)
```

Name	Value
argc	1
argv	0x0097ff7c

```
    llist entries; // Linked list of the cgi entries

    // generate the html code for the web page
    html_header();
    html_begin("EMAC SoM5282: Web Demo Application"); // title
    // close cgihtml <body> and open custom <body>
    printf("</body><body bgcolor=\"#008000\" text=\"#ffffff\">\n");
    h1("<font color=\"ffff00\">Welcome to the EMAC SoM5282</font>");

    // print JavaScript function for refresh method
    print_script ();
```

```
    // generate the html
    html_header();
0x00970066 <main+10>:  js
    html_begin("EMAC SoM5
0x0097006c <main+16>:  pe
0x00970072 <main+22>:  js
0x00970078 <main+28>:  ad
    // close cgihtml <bod
    printf("</body><body
0x0097007a <main+30>:  pe
```

```
web_demo_5282 [C/C++ Local Application] /home/travis/workspace_SIB/SoM-5282EM-SDK/projects/WebDemo/web_demo.cgi.gdb (8/1/06 1:26 PM)
```

In the view above, the current location in the program is highlighted in green in the source code window. Directly to the right of the source code, the current location in the assembly code is also highlighted under the *Disassembly* tab. Any output from the debugger is displayed in the Console window at the bottom of the screen. Note that program I/O will still take place on the shell of the target board. This can be accessed from any of the connection types described in section 4.2, including the RSE SSH shell within Eclipse.

Use the various buttons under the *Debug* tab in the upper left of the screen to manage your debugging session. You can use these to step through the program, resume and pause execution, and toggle the stepping mode, along with other options.

To add a breakpoint, simply double click in the gray margin t the left of the source code on the line that you would like it to be placed. Breakpoints, variables, modules, and registers are monitored in the window in the upper right of the screen. After adding a breakpoint, it will be marked with a breakpoint pin to the left of the line on the source code as well as an entry under the *Breakpoints* tab in the upper right window.

Step through the application and familiarize yourself with the operation of the Eclipse Debugging perspective. This will also allow you to test your configuration by ensuring that everything is working properly before attempting to develop your own applications.
For more information on GDB and debugging within Eclipse, see the links in section 5.

4 Further Information

The combination of Eclipse and the EMAC development tools should allow you to easily and efficiently develop, debug, and test custom applications for use on and with EMAC products such as the SoM-

5282M and the SIB. For more information about the software mentioned in this document, start with the links below.

Eclipse:	www.eclipse.org
Eclipse Wiki:	wiki.eclipse.org
Java:	java.sun.com
GCC:	gcc.gnu.org
GNU Make:	www.gnu.org/software/make
Makefiles:	www.wlug.org.nz/MakefileHowto
Open SSH:	www.openssh.com
Target Management:	www.eclipse.org/dsdp/tm/
GDB:	www.gnu.org/software/gdb
GDBServer:	www.linuxmanpages.com/man1/gcc.1.php
Remote Debugging:	linuxdevices.com/articles/AT6046208714.html
EMAC Software:	ftp://ftp.emacinc.com/PCSBC/Development_Kits/EMAC_Open_Tools/ ftp://ftp.emacinc.com/Controllers/Development_Kits/EMAC_Open_Tools/
EMAC Documents:	ftp://ftp.emacinc.com/PCSBC/Development_Kits/EMAC_Open_Tools/Manuals/ ftp://ftp.emacinc.com/Controllers/Development_Kits/EMAC_Open_Tools/

If you have any questions regarding the information described in this document contact EMAC support (support@emacinc.com) with the subject heading “Eclipse Development Support.”