CodeWarrior®
Targeting PlayStation OS

Because of last-minute changes to CodeWarrior, some of the information in this manual may be inaccurate. Please read the Release Notes on the CodeWarrior CD for the latest up-to-date information.

How to Contact Metrowerks:

U.S.A. and international	Metrowerks Corporation 2201 Donley Drive, Suite 310 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
Mail order	Voice: (800) 377–5416 Fax: (512) 873–4901
World Wide Web	`http://www.metrowerks.com`
Registration information	`register@metrowerks.com`
Technical support	`support@metrowerks.com`
Sales, marketing, & licensing	`sales@metrowerks.com`
America Online	keyword: `Metrowerks`
CompuServe	goto `Metrowerks`

Table of Contents

1

Introduction

This manual shows you how to install and use CodeWarrior to develop software for the PlayStation OS.

Read the Release Notes

If you haven't read the CodeWarrior release notes, please do so now. They contain important information about new features, bug fixes, and incompatibilities. You can find them on the CodeWarrior for PlayStation software development CD, in the Release Notes folder.

Use this chapter to prepare for PlayStation software development using Metrowerks CodeWarrior software. This chapter covers these topics:

- About Targeting PlayStation—the chapters in this manual
- Starting Points—where to begin in this manual, depending upon your level of familiarity with CodeWarrior
- System Requirements—a list of software and hardware you need
- Installing PlayStation Development Software—how to install PlayStation development software on your computer
- Where to Learn More—where to go to learn about PlayStation software programming

About Targeting PlayStation

PlayStation OS is the operating system for the PlayStation video game console developed by Sony Computer Entertainment, Inc.

This manual shows you how to install and use CodeWarrior to develop software for the PlayStation OS. This introduction includes instructions for how to install CodeWarrior software.

Additional chapters in this manual include:

- Installing Hardware—how to install and configure the hardware necessary for PlayStation software development
- Creating Projects—introduces the CodeWarrior tools for PlayStation software development and shows you how to build a PlayStation software title
- Debugging for PlayStation Software—how to use the CodeWarrior Debugger to debug a PlayStation software title
- MIPS Project Settings—customize your PlayStation software development tools
- Troubleshooting—common problems and how to solve them
- Using Assembly—how to include assembly code in your PlayStation C or C++ code
- Metrowerks Utility Library—describes `MWUtils.lib`, a new library that makes PlayStation OS development easier

Starting Points

This section discusses places where you can start your learning process in this manual.

For everyone:

- Read “Installing PlayStation Development Software” on page 11 in this introduction. After that you can read the chapters in this manual sequentially or as you need to refer to them. Chapter 3, “Creating Projects” on page 27 should be especially useful for learning how to use the CodeWarrior tools.
- See the *CodeWarrior IDE User’s Guide for* information about the CodeWarrior Integrated Development Environment.
- When you are ready to debug, be sure to read the *Debugger Manual*.

If you are new to CodeWarrior:

- See the *CodeWarrior QuickStart Guide* for an overview of CodeWarrior, descriptions of some CodeWarrior windows and shortcuts, and pointers to the references available to you. *Quick Start* is packaged with your CodeWarrior for PlayStation Software Development CD.
- See the *CodeWarrior IDE Tutorials* Apple Guide provided on the CodeWarrior for PlayStation CD. The tutorial is designed to give you practice with the common tasks you perform when writing software. Note that the Creating section in *CodeWarrior IDE Tutorials* suggests a series of tutorials especially helpful to new users. There is not a specific tutorial designed for PlayStation software development, but there is a tutorial for the C language. The purpose of these tutorials is to help introduce you to the tools. The Creating Projects chapter also has a brief tutorial.

If you are an experienced CodeWarrior user:

- Congratulations! You're ahead of the rest, and can now focus on writing some games.

System Requirements

Metrowerks CodeWarrior for PlayStation requires either a Macintosh or Windows computer, with these minimum requirements.

Macintosh

- A Mac OS computer with a PowerPC processor. A Power Macintosh with a PCI bus is required to use the PCI development card.
- Mac OS System 7.1.2 or later. CodeWarrior also needs Color QuickDraw, which is part of the Macintosh hardware ROM.

There are also requirements specific to both platforms. These are listed below the Windows requirements.

Windows

- An IBM-compatible 486 or higher computer. For best performance, a Pentium or equivalent processor is recommended.

- Windows 95, or Windows NT 4.0.

There are also requirements specific to both platforms. These are listed below.

Both

- Either the Net Yaroze Development System, or the Sony Programmer Tool Kit, both available exclusively from Sony Computer Entertainment, Inc.

NOTE: If you are using the Net Yaroze development system for Macintosh, you also need to purchase a serial cable with a Macintosh serial connection at one end and a male DB-9 connector at the other end. This cable is commonly referred to as a Hayes™ modem cable.

- PlayStation OS Runtime Libraries compatible with CodeWarrior. These are available exclusively from Sony Computer Entertainment, Inc.

NOTE: The runtime libraries that accompany the Net Yaroze development system and the Programmer Tool Kit are not compatible with the CodeWarrior for PlayStation tools.

- A minimum of 16 MB RAM. 24MB or 32MB RAM is preferable for PlayStation development. When using the CodeWarrior IDE and debugger, 16MB may not provide the best performance.
- A CD-ROM drive to install CodeWarrior software, documentation and examples.
- An external television monitor. If you are using the Sony Programmer Tool Kit and a Power Macintosh equipped with video I/O, you can use the Apple Video Player software as a PlayStation game console monitor on your Macintosh. The PlayStation card outputs NTSC as well as PAL.
- Finally, the CodeWarrior for PlayStation CD.

Our next topic discusses installing PlayStation development software. To learn how to install PlayStation development hardware on your computer, see Chapter 2, “Installing Hardware” on page 15.

Installing PlayStation Development Software

Your first step to writing a PlayStation software title is to install CodeWarrior for PlayStation Software Development. There are two components in this process:

- Installing CodeWarrior for PlayStation Software Development—the CodeWarrior development tools from Metrowerks
- Installing the PlayStation OS Runtime Library—the PlayStation OS support libraries

Installing CodeWarrior for PlayStation Software Development

The CodeWarrior Installer automatically installs all necessary components of the CodeWarrior for PlayStation development tools. Double-click the “CW for PlayStation Installer” icon at the top level of the CodeWarrior for PlayStation CD, and you’re on your way.

If you already have CodeWarrior Gold, the installer updates only the necessary components of CodeWarrior.

It is recommended that you use the CodeWarrior Installer to install the PlayStation development tools to make sure you have all the required files. If you have any questions regarding the installer, read the instructions built into the CodeWarrior Installer itself.

Installing the PlayStation OS Runtime Library

To develop a PlayStation software title, you need to install and convert a PlayStation OS runtime library. This converted library will be used in every PlayStation project you develop, so this section is critical reading.

The original library can only be obtained directly from Sony Computer Entertainment, Inc. This library is included with either the Net Yaroze Development System and Programmer Tool Kit. If you do not have the library mentioned in the tutorial below, contact Sony Computer Entertainment for information on obtaining the correct library for use with CodeWarrior for PlayStation.

There are two topics in this section:

- Installing the Library
- Converting the Library

Installing the Library

1. Install CodeWarrior for PlayStation.

If you haven't done this yet, do so now. See "Installing CodeWarrior for PlayStation Software Development" on page 11.

2. Copy the Sony folders to your hard drive.

Find the folders named "Include" and "Lib" on the CD-ROM supplied with your Sony development hardware (Net Yaroze, or PC or ISA development card). Copy these folders from your CD-ROM into the folder "PlayStation Support", located in the "CodeWarrior" folder.

3. Make sure it's there.

Now open the folder "Lib" in the "PlayStation Support" folder. You will see a file called `Libps.a`

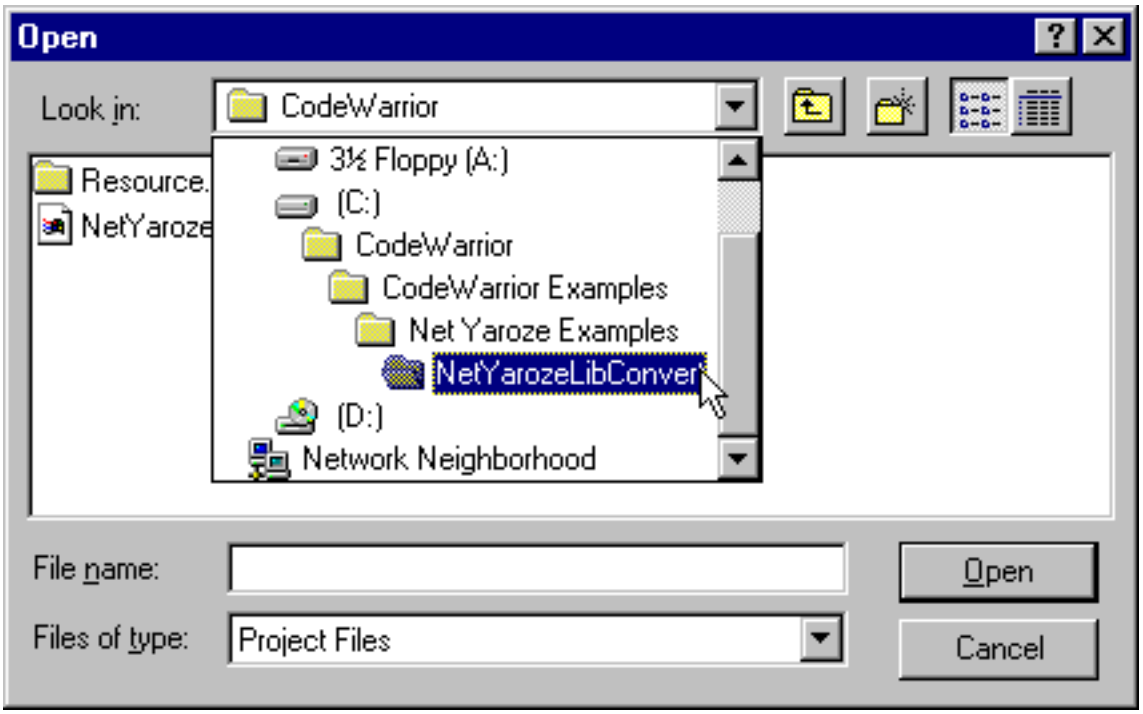
Converting the Library

1. Open the NetYarozeLibConvert project.

Locate and open the file `NetYarozeLibConvert.cwp` (or `NetYarozeLibConvert.μ` on Macintosh). The CodeWarrior project window will open, and you will see one source file contained in the project: `Libps.a`.

Figure 1.1 shows the path to the NetYarozeLibConvert project.

Figure 1.1 Path to NetYarozeLibConvert.cwp



2. Compile the project.

Now that you have the project window on top, click on the file `Libps.a` so that it is highlighted. Choose **Compile** from the **Project** menu.

3. Retrieve and archive the converted library.

Now that the compile is finished, the file `Libps.a.lib` has been created and placed in the folder “Lib”, located in the folder “PlayStation Support”. Get this file, and make a backup of it. You will need to add this file to your PlayStation projects in order for them to compile.

You’re now finished installing software! You can now shutdown your computer to begin installing the necessary hardware.

Where to Learn More

This manual discusses how to use the CodeWarrior tools.This manual does not discuss how to program the PlayStation game console, or how to use the PlayStation OS runtime libraries.

To learn more about the runtime libraries, see the documentation that comes with your Sony development kit.

2

Installing Hardware

This chapter discusses the hardware you need to program the PlayStation game console, and how to install it in your computer.

Hardware Overview

Programming for the PlayStation game console requires additional hardware. Some hardware setup is required before it can be used for CodeWarrior for PlayStation development.

The topics in this chapter include:

- Installing Hardware—how to install the necessary card or console, and connect it to a video display.
- Making Sure It Works—a simple test to ensure that everything has been installed and configured correctly.

Installing Hardware

This section discusses the steps you take to install the required hardware. The topics included are:

- Installing the PCI Card on Windows
- Installing the ISA Card on Windows
- Installing the Net Yaroze Development System on Windows
- Installing the PCI Card on Mac OS
- Installing the Net Yaroze Development System on Mac OS

Installing the PCI Card on Windows

1. **Insert the PCI card into the PCI slot.**

   Follow the installation instructions that are included with the Sony Programmer Tool Kit for installing the PCI card.

2. **Install the PlayStation PCI card driver.**

   Look at the read me file in the System Items directory on the CodeWarrior for PlayStation CD for instructions on installing the driver.

3. **Test it.**

   You are now ready to test your hardware installation. See “Making Sure It Works” on page 20 for details

   For further installation details, refer to the instruction manual for your computer.

Installing the ISA Card on Windows

1. **Install the ISA card.**

   Follow the installation instructions that are included with the Sony Programmer Tool Kit for installing the ISA card.

2. **Install the PlayStation ISA card driver.**

   Look at the read me file in the System Items directory on the CodeWarrior for PlayStation CD for instructions on installing the driver.

3. **Test it.**

   You are now ready to test your hardware installation. See “Making Sure It Works” on page 20 for details

   For further installation details, refer to the instruction manual for your computer.

Installing the Net Yaroze Development System on Windows

The Net Yaroze development system is equipped with the following items, which are necessary to setup the console for development.

- Black PlayStation game console
- Video and power cables
- Controller pads
- Access card (looks like a memory card)
- Black Net Yaroze CD-ROM
- Silver Net Yaroze CD-ROM
- 9-pin to 9-pin connector cable

1. Set up the black PlayStation console.

Connect the black PlayStation console to a monitor, connect the power and video cables, and connect the controller pads.

For detailed instructions, refer to the "Start Up Guide" documentation included in the Net Yaroze package from Sony Computer Entertainment, Inc.

2. Connect the PlayStation console to your PC.

Using the 9-pin to 9-pin connector cable, connect the female end of the cable to the male port on the PlayStation console. Next, connect the other end of the cable to the COM-1 or COM-2 port of your PC.

NOTE: If you are unsure about the location of the serial port on your PC, refer to the user's manual supplied with your computer. Depending on which type of PC you are using, you may need to purchase a 9- to 25-pin cable adapter if your COM ports are 25-pin instead of 9-pin.

3. Install the access card.

Plug the access card into memory slot 1 of the PlayStation console.

4. **Insert the black development CD-ROM.**

   The black Net Yaroze CD-ROM must be seated in the PlayStation console's CD tray in order to begin development.

5. **Test it.**

   You are now ready to test your hardware installation. Skip ahead to "Making Sure It Works" on page 20.

Installing the PCI Card on Mac OS

1. **Remove the jumper from the PCI card.**

WARNING! Do not touch the components and connectors on the board without first being grounded. You can damage the hardware with static discharge if you touch the card's components without a ground.

When looking at the PlayStation PCI card, you can see two slotted connectors at the base of the card. The connector on the left is longer than the short connector on the right. Above the left connector is a chip labeled "Altera", and to the left of this chip is a small black plastic rectangle pushed onto two gold pins. This is called a jumper. It needs to be removed from the card. If it is not removed, the card will not work in your Macintosh.

2. **Install the PCI card.**

   See the PlayStation PCI card installation instructions for complete details on installing the card properly. The installation instructions currently do not include Macintosh installation information. The procedure for installing the PCI card in a PCI-equipped Macintosh is essentially the same as the installation procedure for IBM-compatibles.

3. **Connect the monitor and peripherals.**

   Connect the monitor, controller pads, and other devices to the main board and the connector panel as stated in the PlayStation PCI card installation notes.

4. **Install the PlayStation PCI card driver.**

   Look in the System Folder Items folder on the CodeWarrior for PlayStation CD for the extension `PSXDriver`. This file needs to be dragged onto the System folder of your startup disk.

WARNING! Once you install the PlayStation PCI card driver, you need to first shutdown your computer, and then start it up. Doing a restart alone will not properly install the PCI card driver.

5. **Test it.**

   You are now ready to test your hardware installation. See "Making Sure It Works" below for details.

   For further installation details, refer to the instruction manual for your computer.

Installing the Net Yaroze Development System on Mac OS

The Net Yaroze development system is equipped with the following items, which are necessary to setup the console for development.

- Black PlayStation console
- Video and power cables
- Controller pads
- Access card (looks like a memory card)
- Black Net Yaroze CD-ROM
- Silver Net Yaroze CD-ROM
- 9-pin to 9-pin connector cable

NOTE: You will also need to purchase a 9- to 25-pin adapter cable, and a 25-pin to serial modem cable. A printer serial cable will not work.

1. **Setup the black PlayStation console.**

   Connect the black PlayStation console to a monitor, connect the power and video cables, and connect the controller pads.

   For detailed instructions, refer to the "Start Up Guide" documentation included in the Net Yaroze package from Sony Computer Entertainment, Inc.

2. **Connect the PlayStation console to your Macintosh.**

   First, take the 9- to 25-pin adapter cable and connect the female 9-pin end to the male PlayStation console port. Connect the other end of the adapter cable to the 25-pin male end of the serial cable. Finally, connect the modem serial cable to the Macintosh serial port. You may connect this cable to either the modem or printer port.

3. **Install the access card.**

   Plug the access card into memory slot 1 of the PlayStation console.

4. **Insert the black development CD-ROM.**

   The black Net Yaroze CD-ROM must be seated in the PlayStation console's CD tray in order to begin development.

5. **Test it.**

   You are now ready to test your hardware installation.

Making Sure It Works

After installing the hardware and software, you should make sure it all works. To this end, we have provided a fully-compiled and complete executable PlayStation game console program that should behave properly if you have setup everything correctly.

This section tells you how to go about:

- Testing your PCI- or ISA-based Development System
- Testing the Net Yaroze Development System

Testing your PCI- or ISA-based Development System

In this section you load and run an example project from the debugger. Please follow the instructions carefully.

This will also serve as a brief introduction to the debugger. The Creating Projects chapter has a more complete guided tour of the CodeWarrior environment. The *Debugger Manual*, included on the CodeWarrior for PlayStation CD, has a complete description of the debugger.

1. **Launch the Debugger.**

   Locate and launch the CodeWarrior debugger. The debugger is found in the Metrowerks CodeWarrior folder.

2. **Select your connection method.**

   As the debugger launches, it displays the standard “get file” dialog. Click “cancel” to continue. Select **Preferences** from the Edit Menu. Click the **PlayStation** tab in the Preferences dialog. Select the Connection pop-up menu, and choose the connection appropriate for your setup.

3. **Open the PSE file.**

   Select **Open** from the File Menu, locate the file `Diffuse.pse`, and open it. The Diffuse demo is located in the PlayStation Samples folder, in the folder named `Diffuse`.

4. **Run the executable.**

   Choose the **Run** command from the debugger’s **Control** menu. This starts the executable code running.

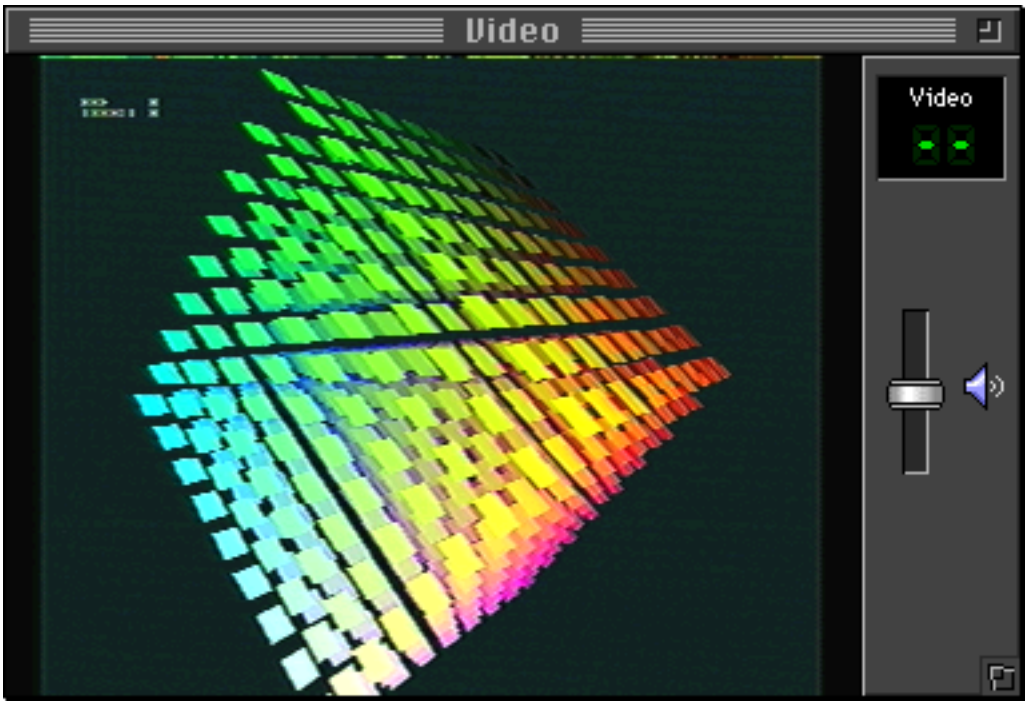
5. **Enjoy the show.**

   If you have properly installed your PlayStation development hardware, and connected the video to a display monitor (or to the Apple Video Player), you should see the Diffuse demo.

   The Diffuse demo is a program that draws zooming colored squares in a three dimensional space. You can rotate the squares, and change the squares to balls, with the game controller.

Figure 2.1 shows what should appear on your monitor.

Figure 2.1 The Diffuse demo



Congratulations! You are ready to begin programming the PlayStation game console.

To end the demo, press the “Select” button on the game controller.

Testing the Net Yaroze Development System

In this section you will configure and run an application to ensure your development hardware is setup properly. Please follow the instructions carefully.

1. **Launch the application PSComUtil.**

   Locate and launch PSComUtil. The purpose of this application is to setup and check your connection from the PlayStation console to your computer.

2. Select your connection method.

As PSComUtil launches, the Preferences dialog is displayed. Select the **Connection** pop-up menu. Figure 2.2 shows this preferences dialog.

If you are using Windows, choose serial as your connection method. Then, select a COM port number (1-4) from the **Port** pop-up menu.

If you are using a Macintosh, choose the port used to connect the PlayStation console to your computer. This is either the modem or printer port.

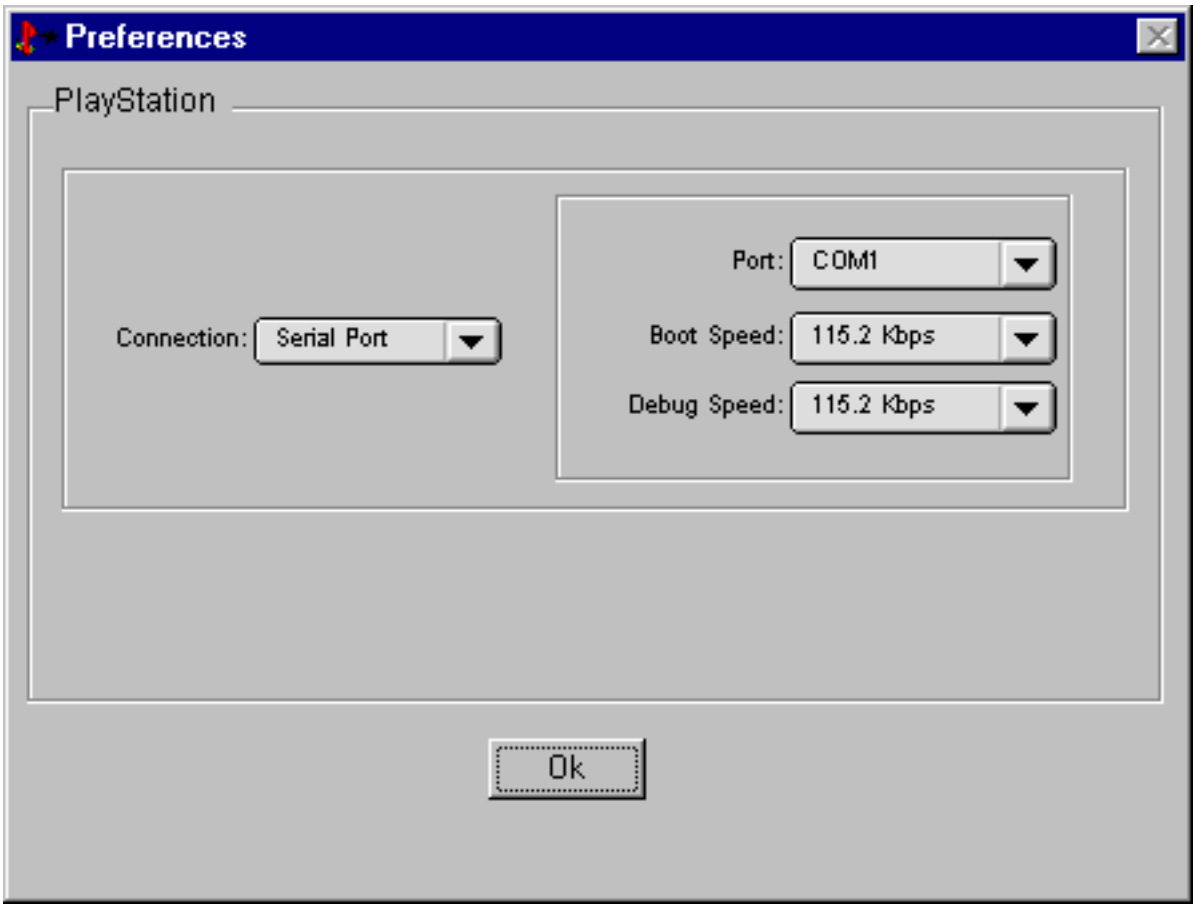
3. Select the Boot Speed.

Next, look at the pop-up menu labeled **Boot Speed**.

When you turn on the Net Yaroze console, you will see a picture on the monitor showing bricks. The last line of text at the top of the screen will have a message that says “Terminal Speed”, followed by a number. Set the boot speed pop-up menu to the value listed after the “Terminal Speed” text on your monitor.

If you don’t have a memory card installed, this will always be 9600 bps.

Figure 2.2 PSComUtil preferences dialog



4. Select the Debug connection speed.

The pop-up menu labelled **Debug Speed** is also visible in the preferences window. Click on the pop-up menu to choose 115.2 kbps. This is the fastest connection possible. If your development machine is a 68K Macintosh, you must choose a slower connection of 57.6 kbps. Click **Ok** to apply the changes.

5. Check the status.

Now that your connection preferences are chosen, the Driver Status Messages window will let you know if your connection was successful.

If the connection is successful, the message “Connection Established” will appear in the window. If unsuccessful, a message say-

ing “Connection Failed.” will appear. Click on the error message to get hints on how to correct the problem. This information will appear in the “Information” window.

Once you’ve corrected the problem, here’s how to reconnect. Select the Communication Prefs menu item from the Edit menu to change your communication settings and reconnect.

6. Download the demo program.

Select **Download Program** from the **Program Execution** menu.

Locate and open the file “Check”. It will be located in the “Check” folder, inside “Net Yaroze Examples”. The Driver Status Messages window will show a status message.

While the download is proceeding, dots will appear in the Driver Status Messages window to indicate that the download is progressing. Once the download is complete, the message “Download Successful” will appear.

If unsuccessful, the message “Download Failed!” will appear. Click on the error message to get hints on how to correct the problem. This information will appear in the “Information” window.

7. Run the demo.

Select **Run** from the **Program Execution** menu. In 5 to 10 seconds, the output of the program will appear on your monitor. For this example, you will see a bouncing ball. To increase or decrease the number of balls on the screen, press the buttons on the controller pad. If your PlayStation console is connected to speakers, you will hear music.

When you are finished viewing the demo, press the Reset button.

3

Creating Projects

This chapter walks you through all the principal steps required to begin developing with CodeWarrior for PlayStation Software Development.

Overview of Creating Projects

In this chapter, our focus is to briefly explain the CodeWarrior for PlayStation development environment, and to walk you through an example PlayStation software project. To create software for the PlayStation game console, you must use CodeWarrior IDE version 1.7 or later.

The topics in this chapter are:

- Introduction to the PlayStation Software Development Tools
- A First Tutorial for PCI and ISA Card Developers
- A First Tutorial for Net Yaroze Developers

Introduction to the PlayStation Software Development Tools

Programming for the PlayStation game console is much like programming for any other target in CodeWarrior. If you have never used CodeWarrior before, the tools you will need to become familiar with are:

- CodeWarrior IDE
- CodeWarrior Debugger for PlayStation Software Development

If you aren't familiar with CodeWarrior, below are the basic components of the development environment.

CodeWarrior IDE

The CodeWarrior IDE is the application that allow you to write your executable. It controls the project manager, the source code editor, and the compilers and linkers.

The CodeWarrior project manager may be new to those more familiar with command-line development tools. All files related to your project are organized in the project manager. This allows you to see your project at a glance, and eases the organization of and navigation between your source code files.

The CodeWarrior IDE has an extensible architecture that uses plugin compilers and linkers to target various operating systems and microprocessors. The CodeWarrior for PlayStation Software Development CD includes a C/C++ compiler for the MIPS™ family of processors. Other CodeWarrior packages include C, C++, Pascal, and Java compilers for Mac OS, Win32, and other platforms.

For more information about the CodeWarrior IDE, you should read the *CodeWarrior IDE User Guide.*

CodeWarrior Debugger for PlayStation Software Development

The CodeWarrior debugger is a separate application that controls your program's execution and allows you to see what's happening internally as your program runs. You use the debugger to find problems in your program's execution. The debugger can execute your program one statement at a time, and suspend execution when control reaches a specified point. When the debugger stops a program, you can view the chain of function calls, examine and change the values of variables, and inspect the contents of the processor's registers.

For information about the debugger, including all of its features and its visual interface, you should read the *CodeWarrior Debugger Manual.*

A First Tutorial for PCI and ISA Card Developers

WARNING! Before you can begin this tutorial, you must first convert a Sony library to Metrowerks format. See “Installing PlayStation Development Software” on page 11 to learn how to do this

In this section you walk through the CodeWarrior for PlayStation software development environment very quickly. You use a sample project to see the components of the development environment, and how to use them to program for the PlayStation game console.

NOTE: CodeWarrior compatible runtime libraries need to be obtained from Sony Computer Entertainment in order to build this tutorial project. The runtime libraries shipped with the PCI and ISA development systems are not compatible with the CodeWarrior for PlayStation tools. For details, see Chapter 1, “System Requirements” on page 9.

If you are not already familiar with CodeWarrior, you may wish to use the CodeWarrior tutorials before diving into the PlayStation software development tour. The CodeWarrior tutorials introduce you to the environment more thoroughly than this quick walk-through. On the Mac OS these tutorials take the form of Apple Guides. On Windows, they are in HTML format. Some of that information is duplicated here. However, what’s really significant about this particular walk-through is that you see the elements related to the PlayStation game console.

If you are already familiar with CodeWarrior, you may wish to read through the steps in this tour anyway. You will encounter the MIPS compiler and linker for the first time, as well as other features of CodeWarrior specific to the PlayStation game console.

This tour is divided into segments. In each segment you will perform a series of steps that introduce you to all the critical elements of this powerful programming environment. The segments are:

- Using the CodeWarrior IDE
- Working with the Debugger

Using the CodeWarrior IDE

In this section of the tour, you work with the CodeWarrior IDE to see a PlayStation demonstration project. This quick tour does not cover or explain all the features of CodeWarrior. It does introduce you to the important elements in CodeWarrior that you use when programming for the PlayStation game console.

In this section you:

- Launch the CodeWarrior IDE.
- Open a project.
- View a source file.
- Set the target.
- Set compiler options.
- Set linker options.
- Generate debugging information.
- Compile the code.

In the next section you work with the debugger.

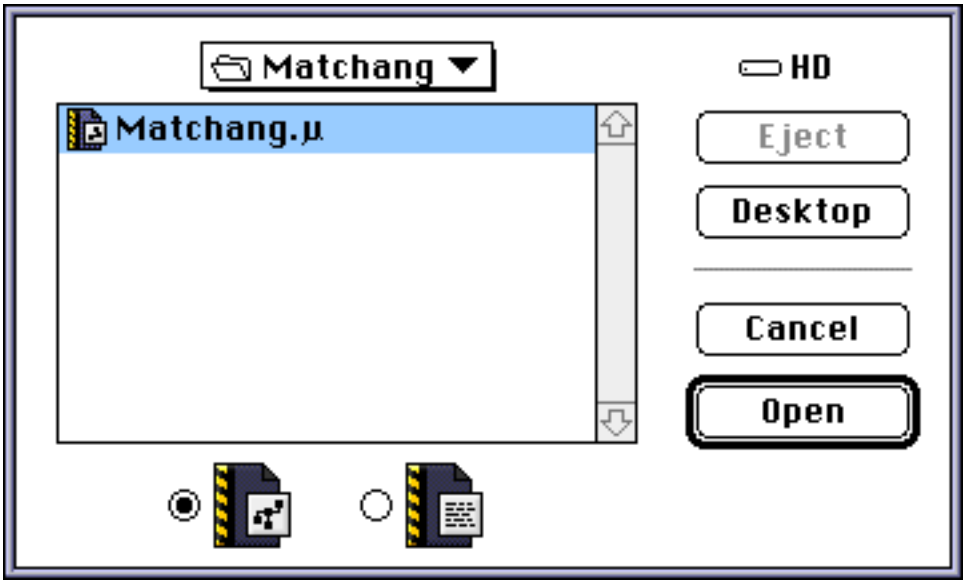
1. Launch the CodeWarrior IDE.

Locate the icon for the CodeWarrior IDE, and launch the CodeWarrior application. When you do, the menu bar changes to reflect the choices available to you in CodeWarrior.

2. Open a project.

From the **File** menu, choose the **Open** item. The dialog in Figure 3.1 appears. The icons at the bottom of the dialog let you choose between seeing available project files, and available source files.

Figure 3.1 The CodeWarrior Open dialog



Locate the project `Matchang.µ`. This project is located in the `Matchang` folder inside the `PlayStation Samples` folder at the top level of the CodeWarrior for PlayStation CD.

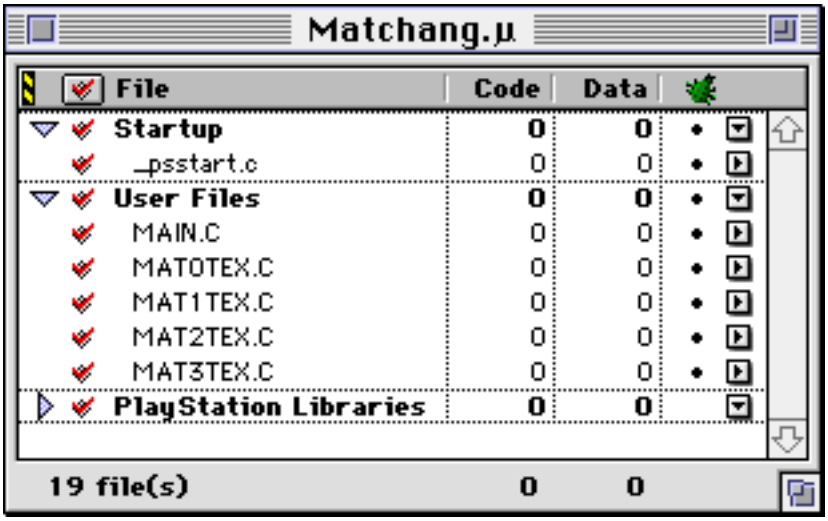
TIP: By convention, Macintosh CodeWarrior projects end with the `.µ` (mu) character. You can generate this character by typing Option-m. For Windows-hosted CodeWarrior projects, use the .cwp naming extension.

Select the project and open it. When you do, the CodeWarrior project window appears, as shown in Figure 3.2.

The project window is the central location from which you control development. This is where you can add or remove source files, add libraries of code, compile your code, generate debugging information, and much more. For full information on the CodeWarrior IDE and project manager, you should see the *CodeWarrior IDE User Guide*, and the tutorials.

NOTE: In this tour, the project file, source files, and other files you create when working with CodeWarrior already exist. You can create new files as well. The CodeWarrior tutorials show you how.

Figure 3.2 The CodeWarrior project window

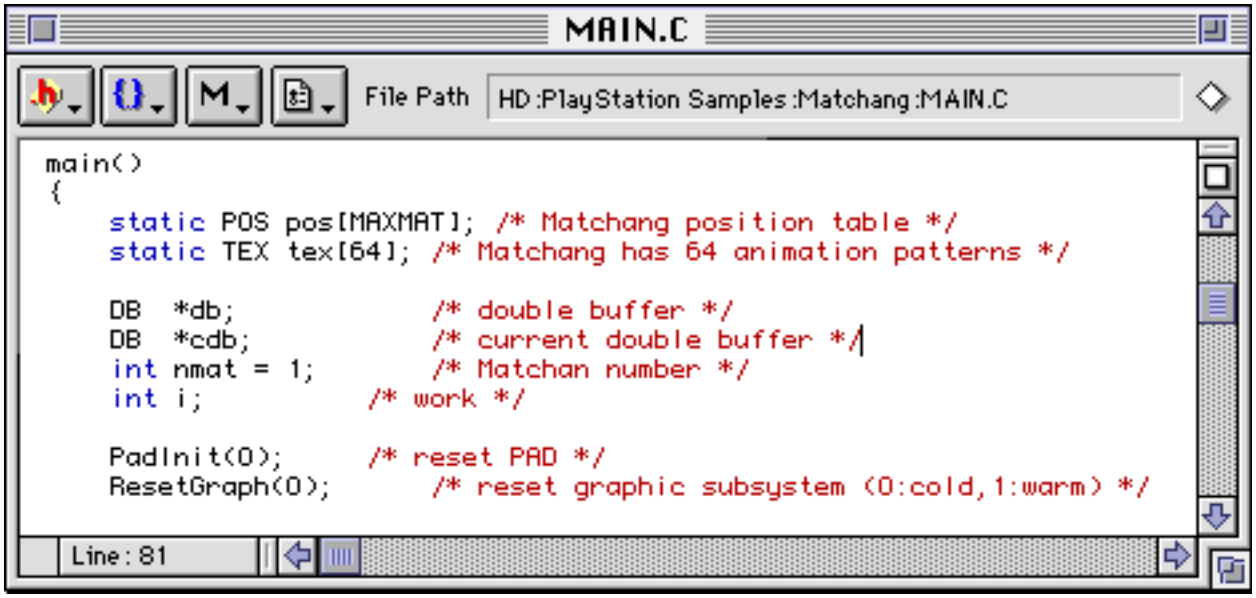


3. View a source file.

There are a variety of techniques you can use to open a source file. However, the simplest method may be to double-click the file name in the CodeWarrior project window.

Double-click the `MAIN.C` file in the project window. When you do, that file's source code appears in a CodeWarrior source code editor window, as shown in Figure 3.3.

Figure 3.3 The CodeWarrior editor window



The editor is where you write code. The editor has many powerful features that are fully explained in the *CodeWarrior IDE User Guide*, and the tutorials.

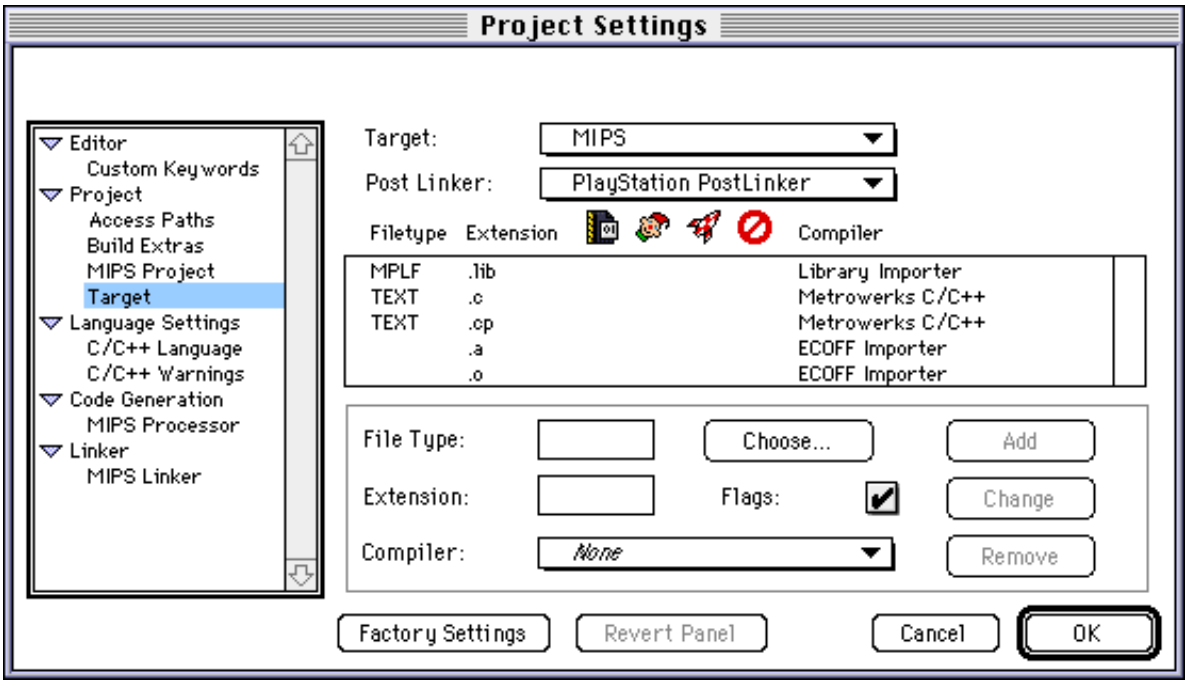
4. Set the target.

In this step you don't actually set the target, but you do see how it's done.

CodeWarrior lets you write code for a wide variety of microprocessors and operating systems. These are called targets in CodeWarrior terminology. When you work with a new CodeWarrior project, one of the first things you do is to tell CodeWarrior what your target platform is.

To do this, first choose the **Project Settings** item from the **Edit** menu. When you do, the CodeWarrior Project Settings dialog appears, as shown in Figure 3.4.

Figure 3.4 The CodeWarrior target settings



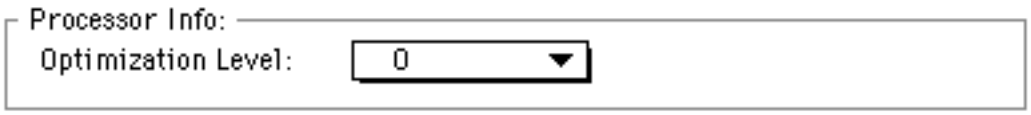
Select the Target item from the available list of setting panels. When you do, the main part of the settings dialog displays a panel with all the options related to choosing a target. Choose MIPS from the **Target** pop-up menu. This tells CodeWarrior that the code you're writing is intended for MIPS processors.

The Project Settings dialog is the location for all project-related options. Every panel and option is explained in the CodeWarrior documentation. Most of the general settings panels are explained in the *CodeWarrior IDE User Guide*. Target-specific panels are typically explained in a Targeting guide, such as this one. For example, the MIPS compiler settings panel is explained in Chapter 5, "MIPS Code Generation Settings" on page 82. The MIPS linker settings panel is explained in "MIPS Linker Settings" on page 84. You meet these two panels in the next steps.

5. Set compiler options.

In the **Project Settings** dialog, choose the MIPS Processor panel from the list of panels. When you do, the settings panel displays the processor info box, as shown in Figure 3.5.

Figure 3.5 MIPS compiler settings



You do not need to modify this setting for the tour. It has already been set in this project. The purpose of this step is simply to introduce you to the way you control compiler settings.

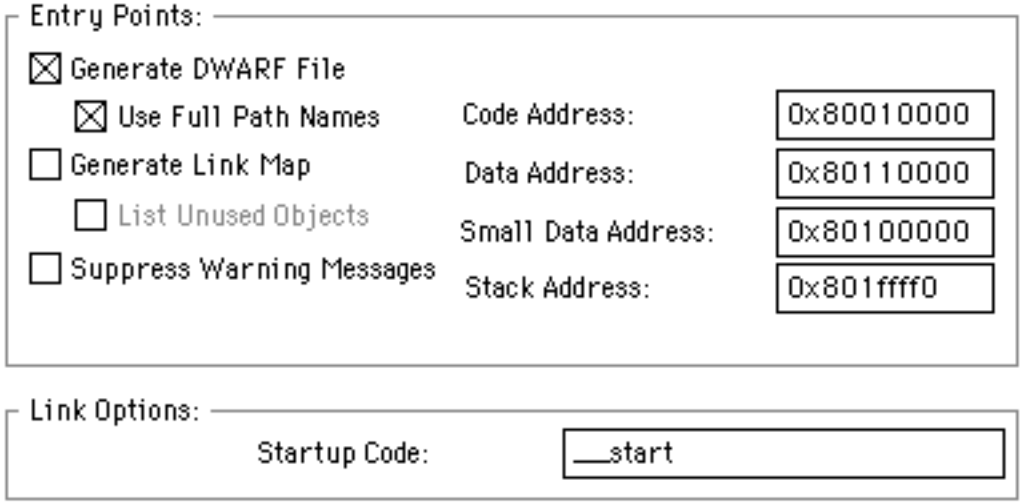
6. Set linker options.

In the Project Settings dialog, choose the MIPS linker. When you do, the settings panel switches to MIPS linker options, as shown in Figure 3.6. All of the options available to you in this panel are fully explained in “MIPS Linker Settings” on page 84.

Once again, you do not need to modify these settings for the tour. They have already been set in this project. The purpose of this step is simply to introduce you to the way you control linker settings.

Examine the settings, and close the Project Settings dialog when you’re done.

Figure 3.6 MIPS linker settings



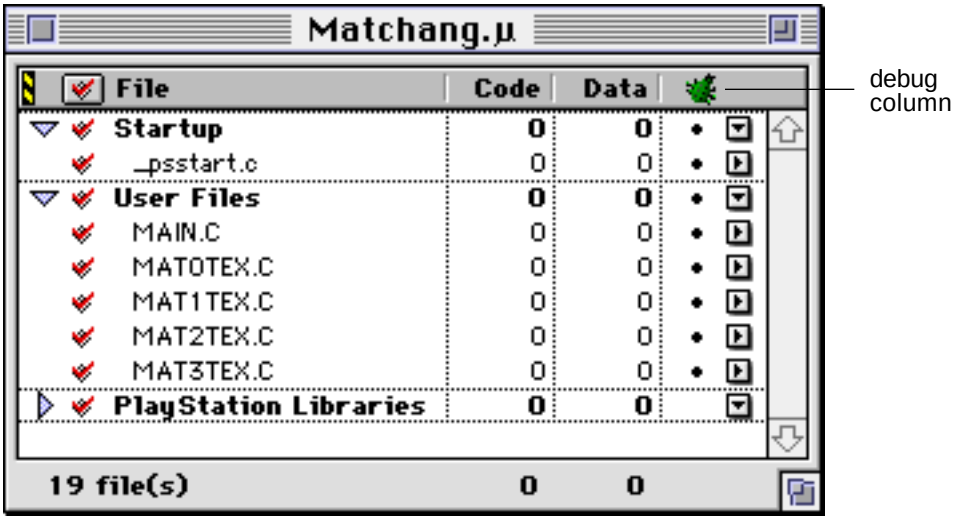
7. Generate debugging information.

For the debugger to work, it needs certain information from the IDE so that it can connect object code to source code. You must tell the CodeWarrior IDE to produce this information.

In the **Project** menu, choose **Enable Debugger**. This tells CodeWarrior that you want it to create debugging information, and sets certain debugger-related preferences automatically.

In the Project window, there is a debug-related column, as shown in Figure 3.7. A dot appears in this column for every file for which the IDE generates debugging information.

Figure 3.7 Turning on debugging per file



In typical use, you want to have debugging turned on for every source file. Then, when you walk through your code in the debugger, the debugger has access to the source code. If debugging information is unavailable for a source file, the debugger can display the object code (assembly) but not the source code.

All the files in the example project for this tour have debugging on. To turn debugging on or off, all you have to do is click in the debugging column next to the file name.

8. Compile the code.

This one's easy. In the CodeWarrior **Project** menu, choose the **Make** item. This tells the CodeWarrior IDE to update all files that need

compiling, and to relink the project. The IDE tracks these dependencies automatically.

TIP: The **Compile** item in the same menu compiles selected files, not all changed files. The **Bring Up To Date** item compiles all changed files but does not link the project into an executable.

When you issue the **Make** command, CodeWarrior compiles all the code. This may take a little while as the IDE locates the files, opens them, and generates the object code. If you see a warning about an uninitialized variable when compiling this code example, you can ignore it. When the compiler is through, the linker creates an executable from the objects. You can see progress in the Project window, and in the toolbar.

At this point you have met all the major components of the IDE, except the debugger. You have seen the project manager, the source code editor, and the preference panels.

Working with the Debugger

In this section you explore the CodeWarrior debugger. Once again, this quick tour does not explain all the features of the debugger. The *CodeWarrior Debugger Manual* covers all the features of the debugger. Individual Targeting guides, such as this one, cover any features of the debugger that are specific to a particular target platform.

This tour simply introduces you to the various components of the debugger. In this section you:

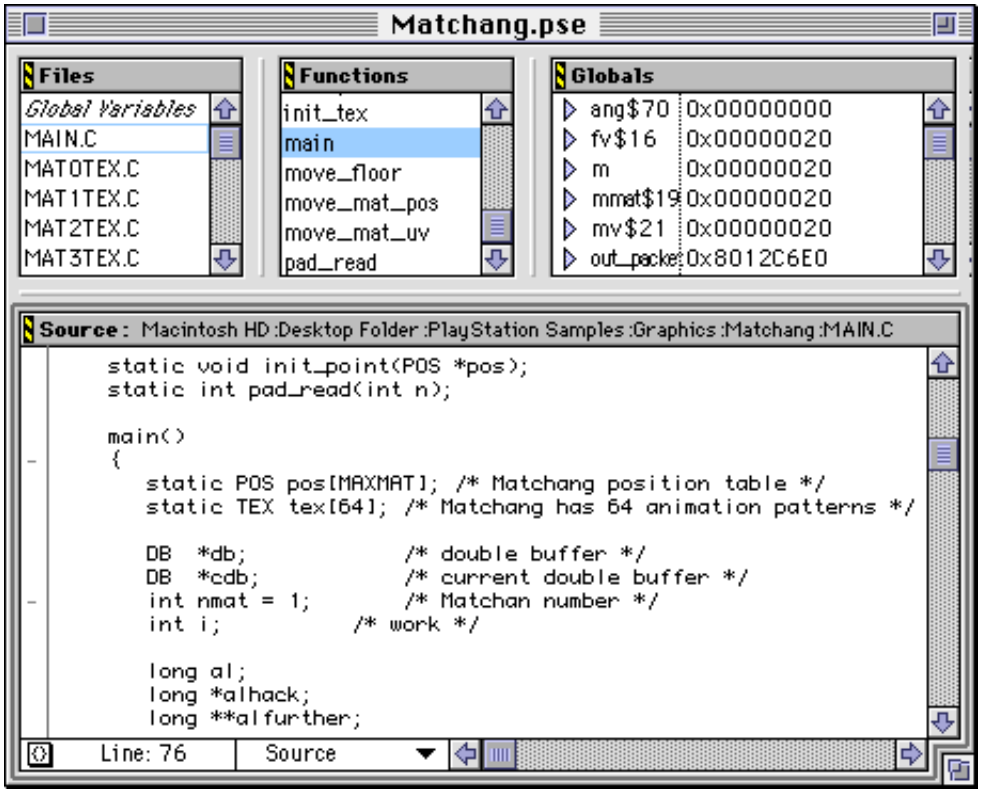
- Launch the debugger.
- Navigate code.
- Set a breakpoint.
- Run with the debugger.
- Display local variables.
- Quit the application.

1. Launch the debugger.

Locate the icon for the CodeWarrior debugger and launch the program. When you do, you'll see a file dialog asking you to locate a PSE (PlayStation executable) file. The PSE file contains the information required by the debugger and prepared by the IDE.

Locate and open the `Matchang.pse` file. When you do the debugger's browser window appears, as shown in Figure 3.8.

Figure 3.8 The debugger browser window



The top left panel of the browser contains all the files. When you select a file, the functions defined in that file appear in the function pane in the top center. Global and static variables (if any) appear in the globals pane in the top right. The source code appears in the bottom pane.

2. Navigate code.

In the **Window** menu, choose the **Show Toolbar** item. This shows the debugger toolbar. The toolbar contains a series of buttons that give access to the execution commands in the **Control** menu: **Run, Stop, Kill, Step Over, Step Into,** and **Step Out**.

To learn more about these buttons do, see Chapter 4,"Basic Debugging" of the *CodeWarrior Debugger Manual.*

3. Set a breakpoint.

Scroll the code in the debugger window to the FOR loop under the comment `/*Update Primitives*/`, in the `main()` function. Click the grey dash in the far left hand column of the debugger browser window. When you do a red circle appears, as shown in Figure 3.9. You have just set a breakpoint.

TIP: To view all breakpoints present in your code, in the **Window** menu choose the **Show Breakpoints** item.

4. Run with the debugger.

Choose **Run** from the **Control** menu. This causes your code to begin execution and continue until a breakpoint is reached.

5. Display local variables.

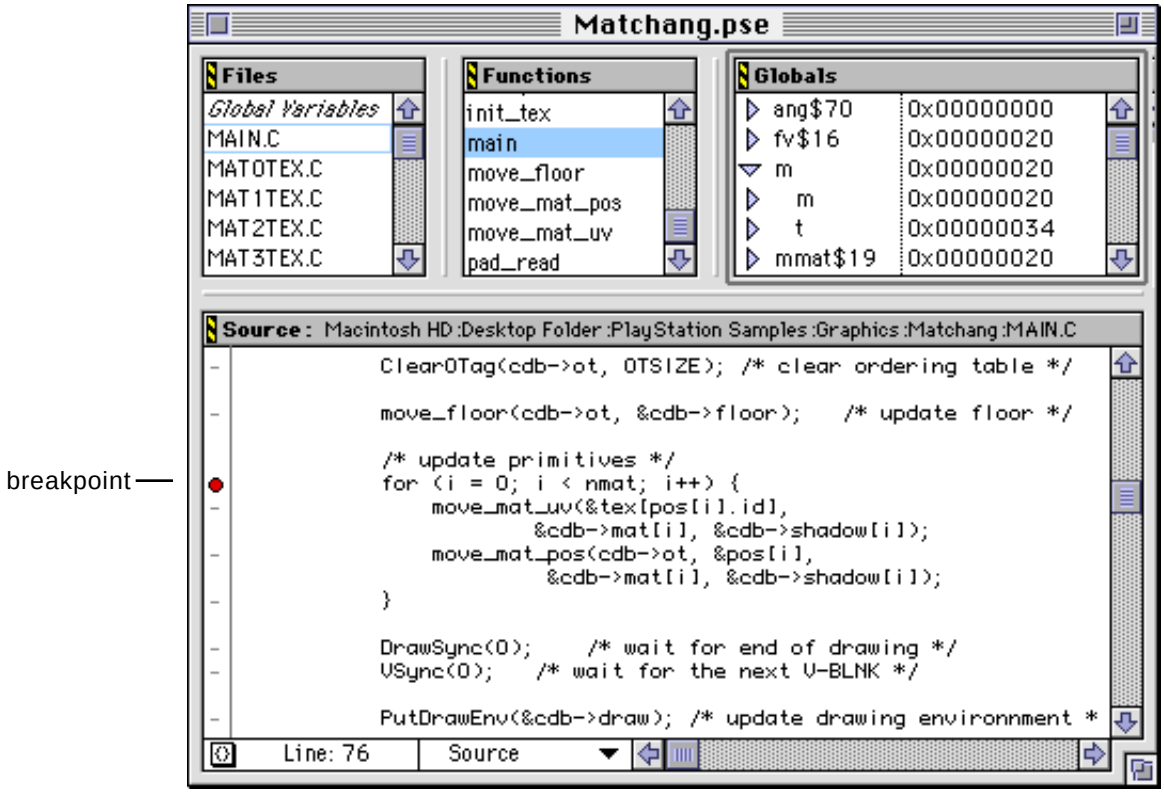
Local variables appear in the top right pane of the debugger browser window. To view the contents of a local variable, choose one from the list and click the disclosure triangle to the left of the variable.

6. Quit the application.

Choose **Kill** from the **Control** menu. This causes your code to cease execution.

Congratulations! You've completed the tour. You now have met the basic elements of the CodeWarrior for PlayStation development environment.

Figure 3.9 Breakpoint in the debugger browser window



There is a lot more to CodeWarrior than what you have seen on this quick tour. For example, CodeWarrior has an excellent browser that lets you navigate through your code quickly and easily. As you work with CodeWarrior, refer to the *CodeWarrior IDE User Guide* to learn about all of the powerful features available to you.

A First Tutorial for Net Yaroze Developers

WARNING! Before you can begin this tutorial, you must first convert a Sony library to Metrowerks format. See “Installing PlayStation Development Software” on page 11 to learn how to do this.

In this section you walk through the CodeWarrior for PlayStation software development environment very quickly. You use a sample project to see the components of the development environment, and how to use them to program for the PlayStation game console.

If you are not already familiar with CodeWarrior, you may wish to use the CodeWarrior tutorials before diving into the PlayStation software development tour. The CodeWarrior tutorials introduce you to the environment more thoroughly than this quick walk-through. Some of that information is duplicated here. However, what's really significant about this particular walk-through is that you see the elements related to the PlayStation game console.

If you are already familiar with CodeWarrior, you may wish to read through the steps in this tour anyway. You will encounter the MIPS compiler and linker for the first time, as well as other features of CodeWarrior specific to the PlayStation game console.

This tour is divided into segments. In each segment you will perform a series of steps that introduce you to all the critical elements of this powerful programming environment. The segments are:

- Using the CodeWarrior IDE
- Working with the Debugger

Using the CodeWarrior IDE

In this section of the tour, you work with the CodeWarrior IDE to see a PlayStation demonstration project. This quick tour does not cover or explain all the features of CodeWarrior. It does introduce you to the important elements in CodeWarrior that you use when programming for the PlayStation game console.

In this section you:

- Launch the CodeWarrior IDE.
- Open a project.
- View a source file.
- Set the target.
- Set compiler options.

- Set linker options.
- Generate debugging information.
- Compile the code.

In the next section you work with the debugger.

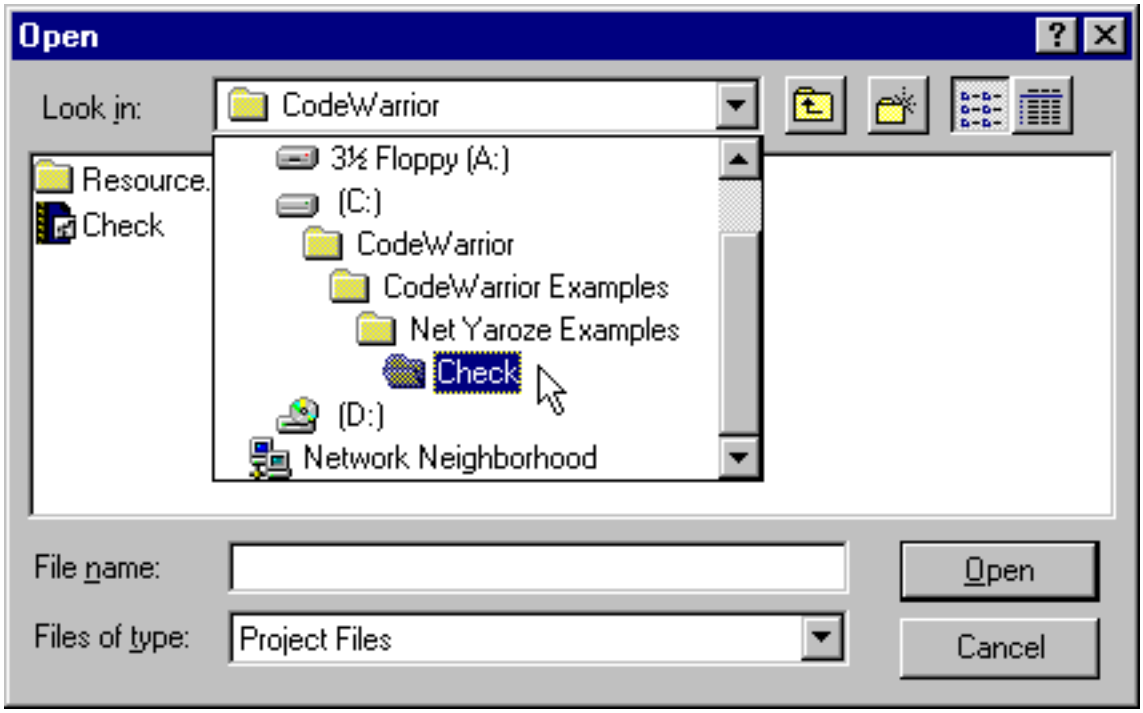
1. Launch the CodeWarrior IDE.

Locate the icon for the CodeWarrior IDE, and launch the CodeWarrior application. When you do, the menu bar changes to reflect the choices available to you in CodeWarrior.

2. Open a project.

From the **File** menu, choose the **Open** item. The dialog in Figure 3.10 appears. The icons at the bottom of the dialog let you choose between seeing available project files, and available source files.

Figure 3.10 The CodeWarrior Open dialog



Locate the project `Check`. This project is located in the `Check` folder, which is inside the `Net Yaroze Examples` folder.

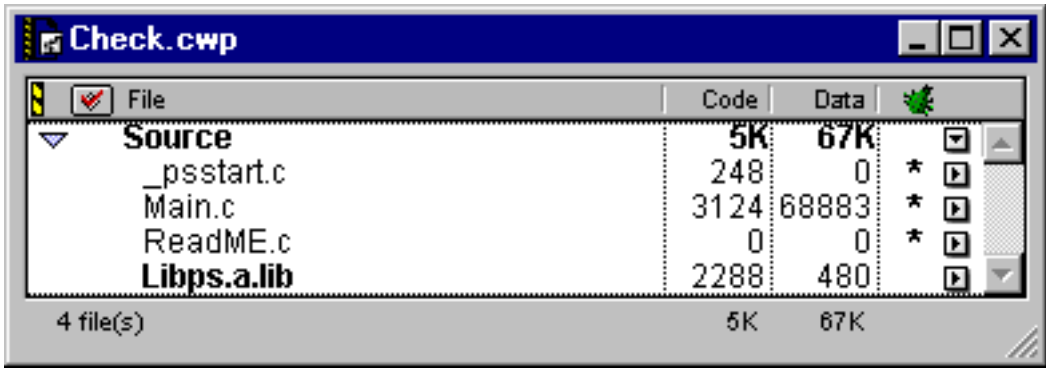
NOTE: By convention, Macintosh CodeWarrior projects end with the .µ (mu) character. You can generate this character by typing Option-m. For Windows-hosted CodeWarrior projects, use the .cwp naming extension.

Select the project and open it. When you do, the CodeWarrior project window appears, as shown in Figure 3.11.

The project window is the central location from which you control development. This is where you can add or remove source files, add libraries of code, compile your code, generate debugging information, and much more. For full information on the CodeWarrior IDE and project manager, you should see the *CodeWarrior IDE User Guide*, and the tutorials.

NOTE: In this tour, the project file, source files, and other files you create when working with CodeWarrior already exist. You can create new files as well. The CodeWarrior tutorials show you how.

Figure 3.11 The CodeWarrior project window

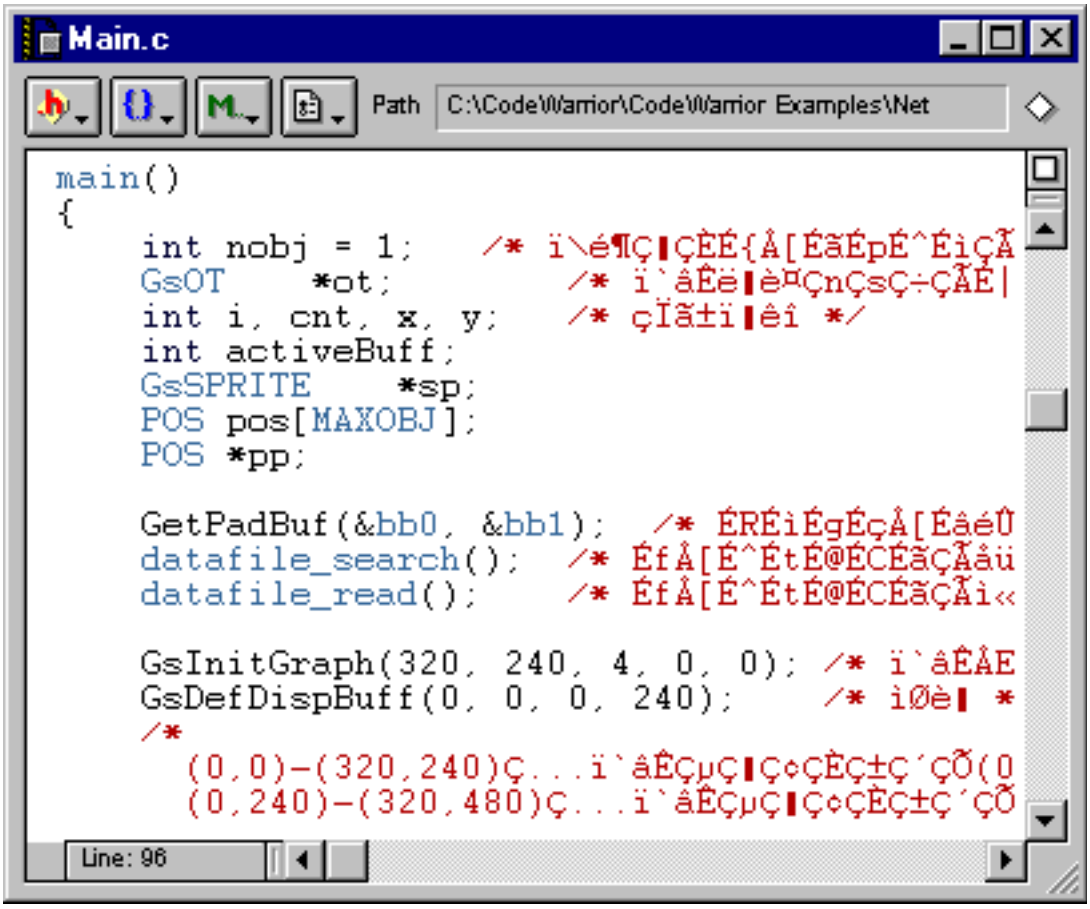


3. View a source file.

There are a variety of techniques you can use to open a source file. However, the simplest method may be to double-click the file name in the CodeWarrior project window.

Double-click the `Main.c` file in the project window. When you do, that file's source code appears in a CodeWarrior source code editor window, as shown in Figure 3.12.

Figure 3.12 The CodeWarrior editor window



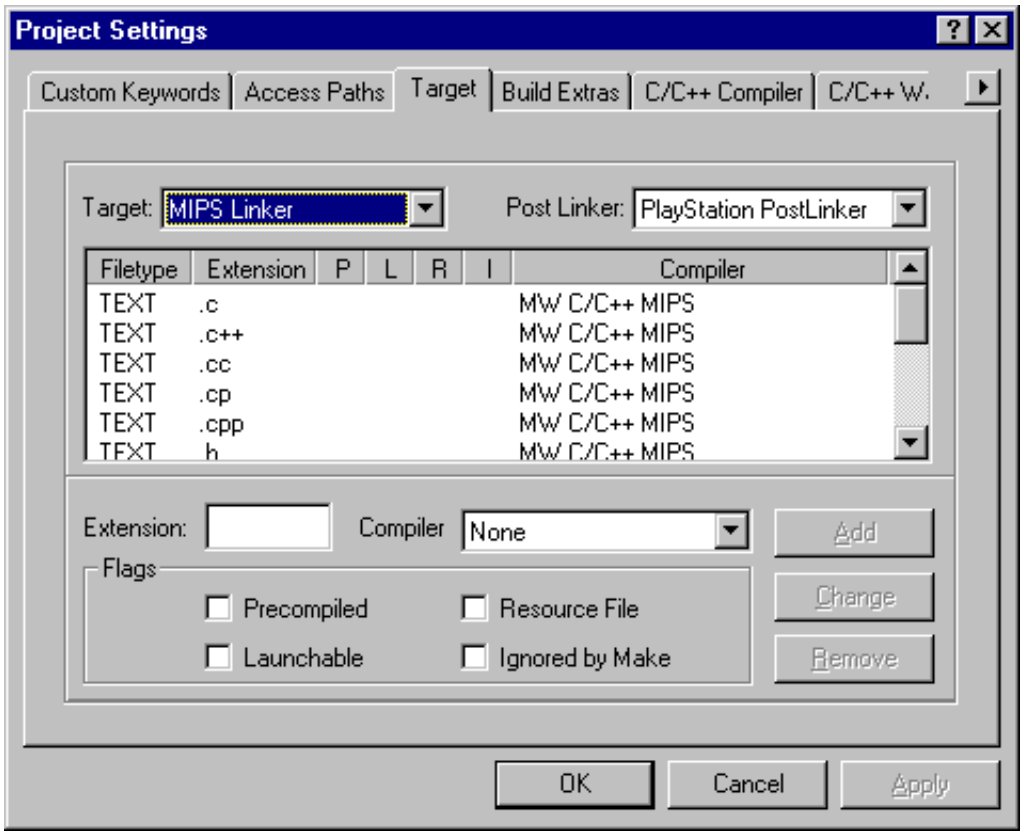
The editor is where you write code. The editor has many powerful features that are fully explained in the *CodeWarrior IDE User Guide*, and the tutorials.

4. Set the target.

CodeWarrior lets you write code for a wide variety of microprocessors and operating systems. These are called targets. When you work with a new CodeWarrior project, one of the first things you do is to tell CodeWarrior what your target platform is.

To do this, first choose the **Project Settings** item from the **Edit** menu. When you do, the CodeWarrior Project Settings dialog appears, as shown in Figure 3.13.

Figure 3.13 The CodeWarrior target settings



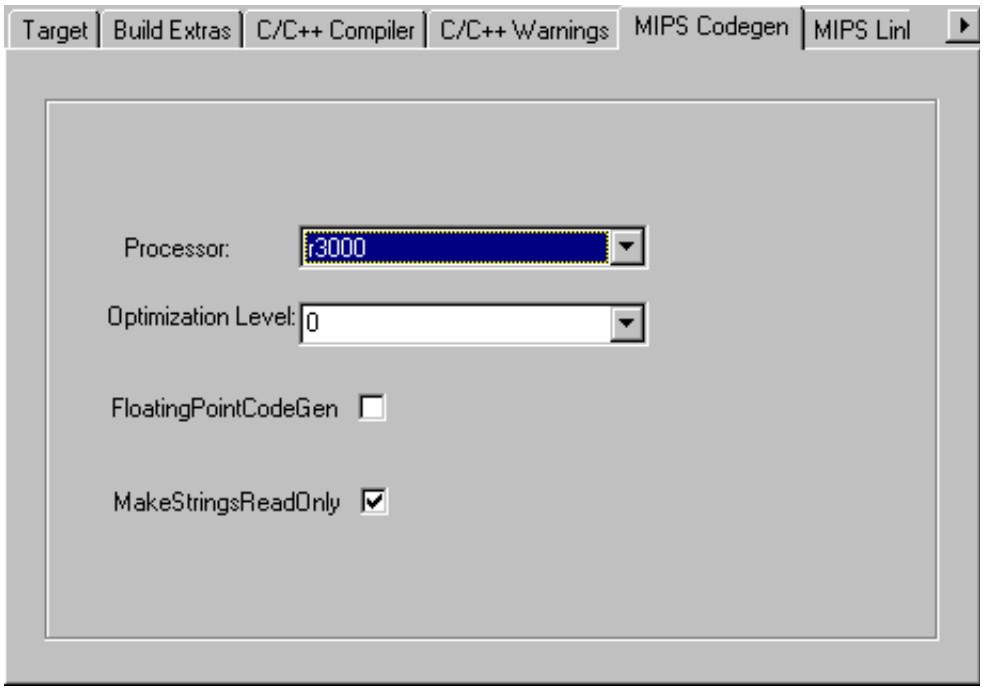
Select the Target panel. When you do, the main part of the settings dialog displays a panel with all the options related to choosing a target. Choose MIPS from the **Target** pop-up menu. This tells CodeWarrior that the code is intended for MIPS processors.

The Project Settings dialog is the location for all project-related options. Every panel and option is explained in the CodeWarrior documentation. Most of the general settings panels are explained in the *CodeWarrior IDE User Guide*. Target-specific panels are typically explained in a Targeting guide, such as this one. For example, the MIPS compiler settings panel is explained in Chapter 5, “MIPS Code Generation Settings” on page 82. The MIPS linker settings panel is explained in “MIPS Linker Settings” on page 84. You meet these two panels in the next steps.

5. Set compiler options.

In the **Project Settings** dialog, choose the MIPS Processor panel from the list of panels. When you do, the settings panel displays the processor info box, as shown in Figure 3.14.

Figure 3.14 MIPS compiler settings



You do not need to modify this setting for the tour. It has already been set in this project. The purpose of this step is simply to introduce you to the way you control compiler settings.

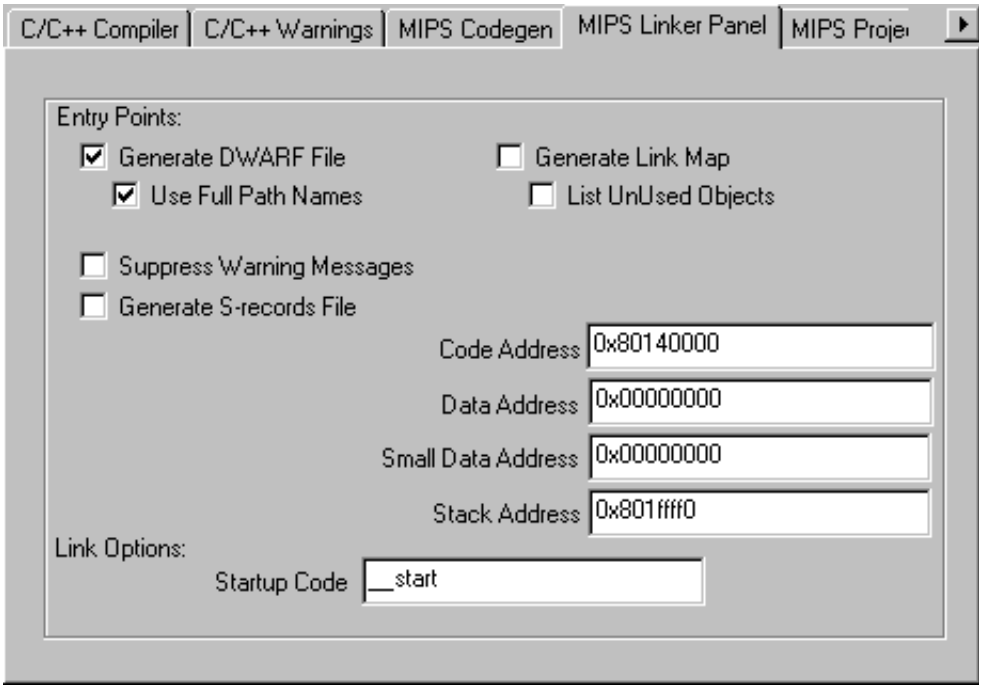
6. Set linker options.

In the Project Settings dialog, choose the MIPS linker. When you do, the settings panel switches to MIPS linker options, as shown in Figure 3.15. All of the options available to you in this panel are fully explained in "MIPS Linker Settings" on page 84.

Once again, you do not need to modify these settings for the tour. They have already been set in this project. The purpose of this step is simply to introduce you to the way you control linker settings.

Examine the settings, and close the Project Settings dialog when you're done.

Figure 3.15 MIPS linker settings



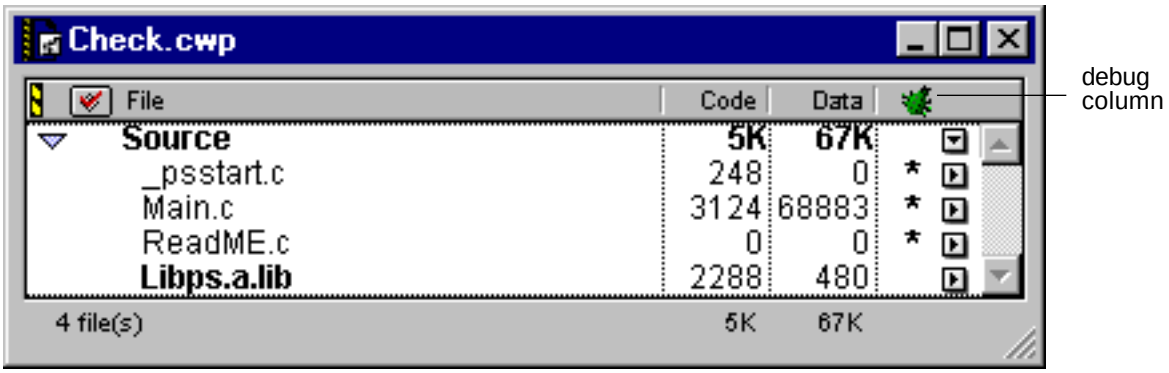
7. Generate debugging information.

For the debugger to work, it needs certain information from the IDE so that it can connect object code to source code. You must tell the CodeWarrior IDE to produce this information.

In the **Project** menu, choose **Enable Debugger**. This tells CodeWarrior that you want it to create debugging information, and sets certain debugger-related preferences automatically.

In the Project window, there is a debug-related column, as shown in Figure 3.16. A dot appears in this column for every file for which the IDE generates debugging information.

Figure 3.16 Turning on debugging per file



In typical use, you want to have debugging turned on for every source file. Then, when you walk through your code in the debugger, the debugger has access to the source code. If debugging information is unavailable for a source file, the debugger can display the object code (assembly) but not the source code.

All the files in the example project for this tour have debugging on. To turn debugging on or off, all you have to do is click in the debugging column next to the file name.

8. Compile the code.

This one's easy. In the CodeWarrior **Project** menu, choose the **Make** item. This tells the CodeWarrior IDE to update all files that need compiling, and to relink the project. The IDE tracks these dependencies automatically.

TIP: The **Compile** item in the same menu compiles selected files, not all changed files. The **Bring Up To Date** item compiles all changed files but does not link the project into an executable.

When you issue the **Make** command, CodeWarrior compiles all the code. This may take a little while as the IDE locates the files, opens them, and generates the object code. If you see a few warnings when compiling this code example, you can ignore them and close the error message window. When the compiler is through, the linker creates an executable from the objects. You can see progress in the Project window, and in the toolbar.

At this point you have met all the major components of the IDE, except the debugger. You have seen the project manager, the source code editor, and the preference panels.

Working with the Debugger

In this section you explore the CodeWarrior debugger. Once again, this quick tour does not explain all the features of the debugger. The *CodeWarrior Debugger Manual* covers all the features of the debugger. Individual Targeting guides, such as this one, cover any features of the debugger that are specific to a particular target platform.

This tour simply introduces you to the various components of the debugger. In this section you:

- Launch the debugger.
- Navigate code.
- Set a breakpoint.
- Run with the debugger.
- Display local variables.
- Quit the application.

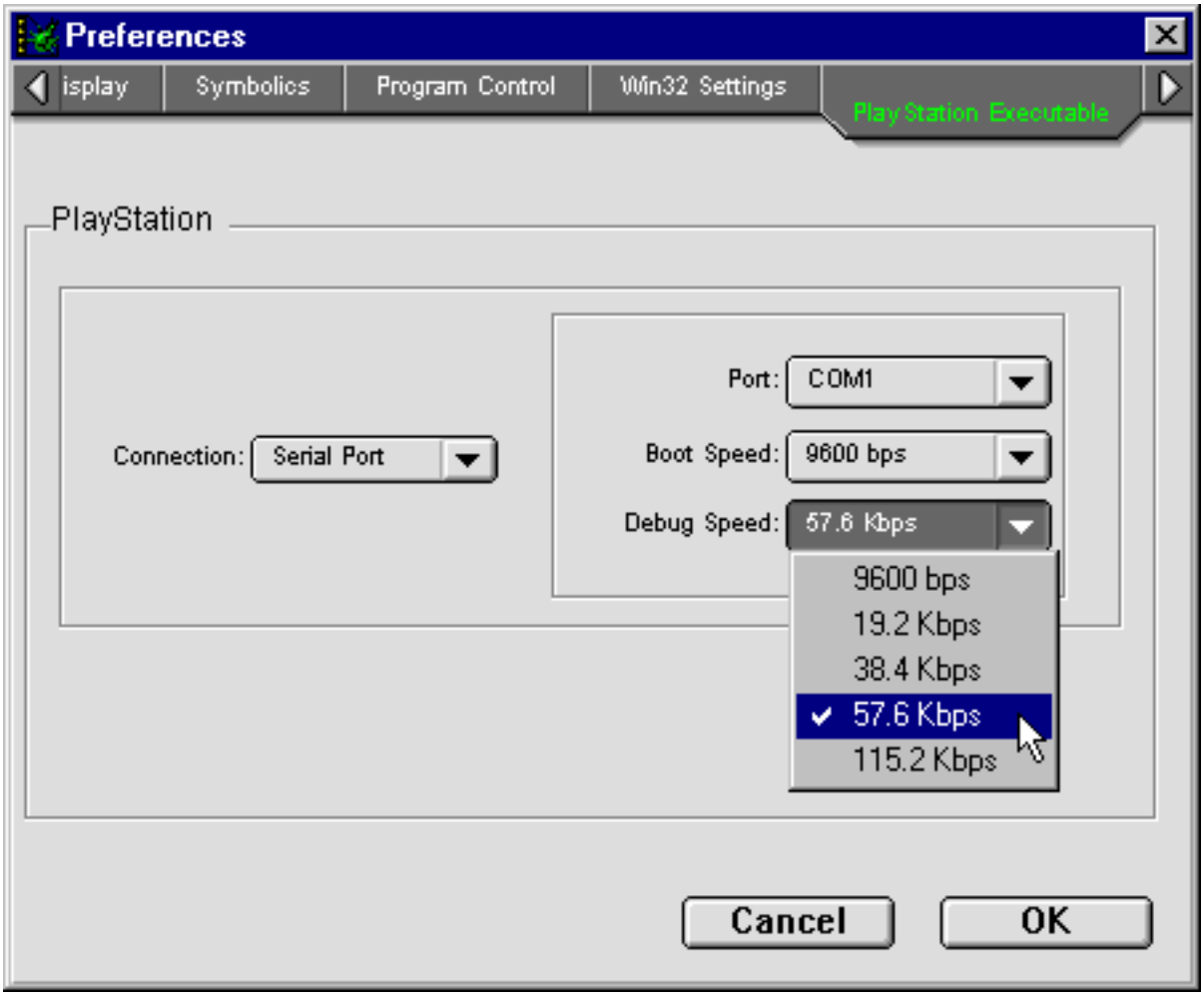
1. Launch the debugger.

Locate the icon for the CodeWarrior debugger and launch the program. When you do, you'll see an open file dialog. Click **Cancel** to continue.

2. Select the debugger preferences.

Choose Preferences from the Edit menu. Click on the PlayStation Executable panel, as shown in Figure 3.17

Figure 3.17 The PlayStation Executable panel



If you are using Windows, choose serial as your connection method. Then, select a COM port number (1-4) from the **Port** pop-up menu.

If you are using a Macintosh, choose the port used to connect the PlayStation console to your computer. This is either the modem or printer port.

a. Select the Boot Speed.

Next, look at the pop-up menu labelled **Boot Speed**.

When you turn on the Net Yaroze console, you will see a picture on the monitor showing bricks. The last line of text at the top of the screen will have a message that says “Terminal Speed”, followed by a number. Set the boot speed pop-up menu to the value listed after the “Terminal Speed” text on your monitor.

If you don’t have a memory card installed, this will always be 9600 bps.

NOTE: If you have a memory card installed, the debug speed that you chose when configuring PSComUtil is saved in the memory card. Therefore, the boot speed may not be the same for debugging as it was for PSComUtil. Double check the panel to be sure that your options are set as you intended.

b. Select the Debug connection speed.

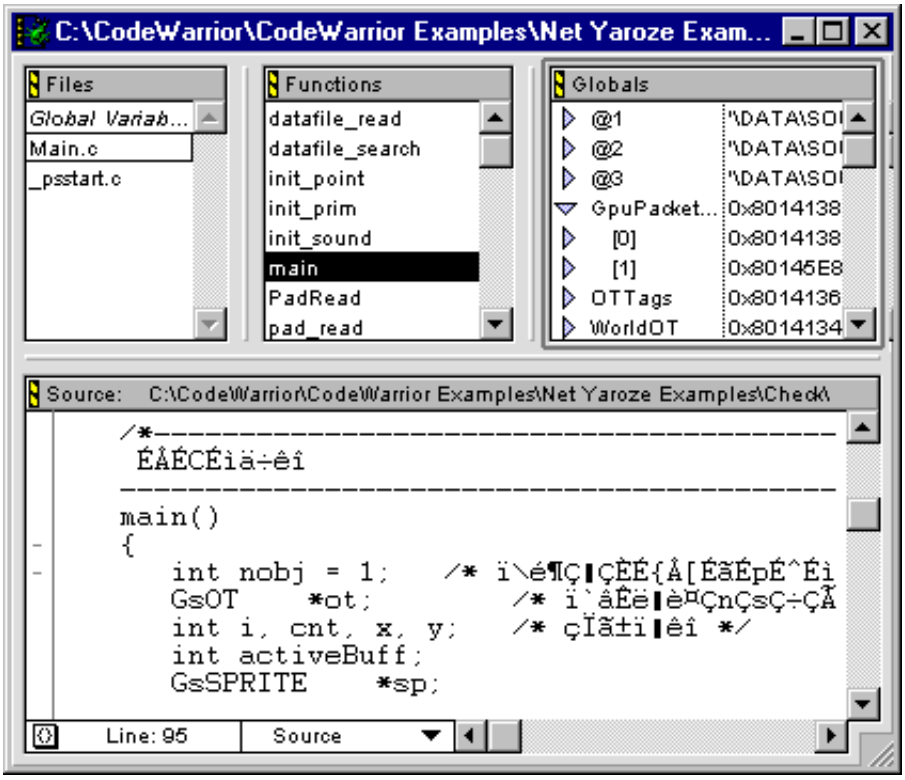
The pop-up menu labelled **Debug Speed** is also visible in the preferences window. Click on the pop-up menu to choose 115.2 kbps. This is the fastest connection possible. If your development machine is a 68K Macintosh, you must choose a slower connection of 57.6 kbps. Click **Ok** to apply the changes.

3. Prepare to debug.

Locate and open the `Check.pse` file. PSE files contain information required by the debugger and prepared by the IDE. When you do the debugger’s browser window appears, as shown in Figure 3.18.

If the browser should fail to appear and you get an error message asking you to reset the Net Yaroze console, double check your COM port or boot speed setting. These are the most common communications errors.

Figure 3.18 The debugger browser window



The top left panel of the browser contains all the files. When you select a file, the functions defined in that file appear in the function pane in the top center. Global and static variables (if any) appear in the globals pane in the top right. The source code appears in the bottom pane.

4. Navigate code.

Now switch to the second debugging window with the VCR-like toolbar between the upper and lower window. The toolbar contains a series of buttons that give access to the execution commands in the **Control** menu: **Run, Stop, Kill, Step Over, Step Into,** and **Step Out**.

To learn more about these buttons do, see Chapter 4,"Basic Debugging" of the *CodeWarrior Debugger Manual.*

5. Set a breakpoint.

Scroll the code in the debugger window to the FOR loop above the code

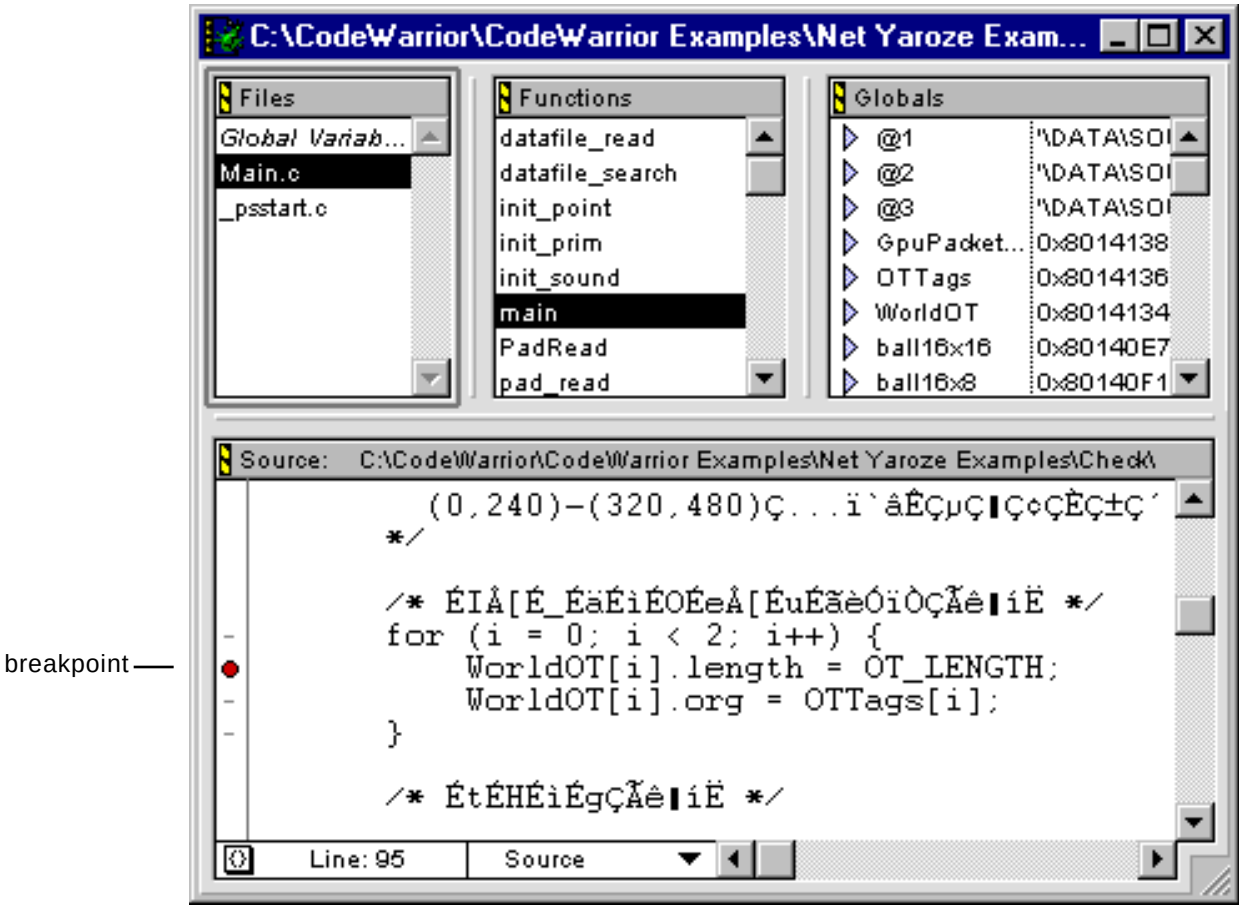
```
WorldOT[i].length = OT_LENGTH;
```

in the `main()` function.

Click the grey dash in the far left hand column of the debugger browser window. When you do a red circle appears, as shown in 3.19. You have just set a breakpoint.

TIP: To view all breakpoints present in your code, in the **Window** menu choose the **Show Breakpoints** item.

Figure 3.19 Breakpoint in the debugger browser window



6. Run with the debugger.

Choose **Run** from the **Control** menu. This causes your code to begin execution and continue until a breakpoint is reached.

On the monitor, you should see a ball bouncing around. This is the correct output of the `Check` program.

7. Display local variables.

Local variables appear in the top right pane of the debugger browser window. To view the contents of a local variable, choose one from the list and click the disclosure triangle to the left of the variable.

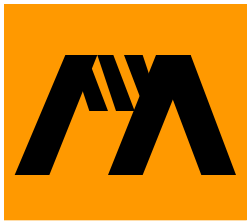
8. Quit the application.

To stop execution of the program, close the debugger's program window (the one with the toolbar). Press Reset on the Net Yaroze console, and your program is now stopped.

Congratulations! You've completed the tour. You now have met the basic elements of the CodeWarrior for PlayStation development environment.

There is a lot more to CodeWarrior than what you have seen on this quick tour. For example, CodeWarrior has an excellent browser that lets you navigate through your code quickly and easily. As you work with CodeWarrior, refer to the *CodeWarrior IDE User Guide* to learn about all of the powerful features available to you.

4



Debugging for PlayStation Software

This chapter discusses the features of CodeWarrior debugging that are unique to the PlayStation game console.

Overview of Debugging for PlayStation Software

This chapter assumes that you are generally familiar with the operation of the CodeWarrior Debugger. For information about the debugger, including all of its features and its visual interface, you should read the *Debugger Manual*, included on the CodeWarrior for PlayStation CD.

However, the debugger for PlayStation software development has some additional features, and is missing a few features found in the generic CodeWarrior debugger. This chapter covers the differences between the generic debugger described in the *Debugger Manual*, and the PlayStation software debugger.

The topics include:

- Debugger Setup—making sure everything is configured correctly before debugging begins
- Debugger Preferences—special preferences for the PlayStation software debugger
- Additional Debugging Features—features found in the PlayStation software debugger not described in the *Debugger Manual*
- Limitations of the PlayStation Debugger—features that are limited with the current PlayStation debugger

- Unimplemented Debugger Features—features described in the *Debugger Manual* that are not available for PlayStation software debugger
- The Debugging Library for PlayStation Software—a reference to assist you when you debug PlayStation OS code

Debugger Setup

Before you begin debugging, there are a few things to double check.

There are two topics in this section:

- For PCI and ISA Card Users
- For Net Yaroze Development System Users

For PCI and ISA Card Users

Be sure that the card has been installed correctly, and the appropriate driver has been installed. For details, see “Installing Hardware” on page 15.

For Net Yaroze Development System Users

First, be sure that all cables have been properly connected. If you are unsure what needs to be done, see “Installing Hardware” on page 15.

Second, before debugging is allowed with the CodeWarrior for PlayStation tools, you must seat the Net Yaroze black debugging CD in the console’s CD tray, and the access card must be plugged in to the memory card slot 1.

Finally, the Reset button on the console needs to be pressed before each debugging session to ensure that communications are established correctly.

Debugger Preferences

To properly debug PlayStation software, you should set certain preferences in the debugger's Preferences dialog.

The PlayStation software debugger actually has four preference-related panels: **Settings & Display**, **Symbolics**, **Program Control**, and **PlayStation Executable**. The Windows-hosted debugger has an extra panel labelled **Win32 Settings**.

The **Settings & Display**, **Symbolics**, and **Program Control** panels are fully described in the *Debugger Manual*. However, certain target preferences described in that manual are irrelevant to the PlayStation software debugger and have been removed from the dialog.

The topics in this section are:

- PlayStation Executable Preferences Panel
- PlayStation Executable Preference Panel

The **PlayStation Executable** panel is unique to the PlayStation software debugger.

PlayStation Executable Preferences Panel

To properly debug PlayStation code, you should set certain preferences in the debugger's PlayStation Executable preferences panel. The panel has three areas:

- Connection Selection
- Connection Options
- Video Mode

Connection Selection

The Connection Selection pop-up menu selects the method by which the debugger should communicate with the PlayStation game console.

The Net Yaroze version of CodeWarrior only supports one method: Serial, either through a COM port on Windows or the modem or

printer port on Mac OS. The Pro version of CodeWarrior supports PCI and ISA as well as Serial.

Table 4.1 Connection selection choices

Choice	For
Serial Port	Net Yaroze, Windows
Modem Port	Net Yaroze, Mac OS
Printer Port	Net Yaroze, Mac OS
PCI Card	Mac OS or Windows
ISA Card	Windows

Select the Serial Port option if you are connecting to a Net Yaroze development system from Windows. When the “Serial Port” item is selected in the Connection Selection, a pop-up menu will appear in the Connection Options area to control which port you use.

Select either the Modem Port or Printer Port option if you are connecting to a Net Yaroze development system from Mac OS.

Select either the PCI Card or ISA Card options if you have a card of the corresponding type installed in your computer.

Connection Options

The items that appear in this area depend upon the choice you make in the Connection Selection section.

If you choose Serial Port for Net Yaroze on Windows, you’ll see a pop-up menu to select which PC COM port to use. Choose the COM port to which your Net Yaroze is connected. If you pick the wrong port, or one that is in use, the debugger will generate an error when you attempt to connect to the Net Yaroze.

The two other pop-up menus are for serial speed. The first is the speed at which your Net Yaroze boots. If you do not have a memory card installed, the Net Yaroze will always boot at 9600 baud. You can determine the speed at which your Net Yaroze boots by looking

at the last line of text at the top of the TV monitor after the Net Yaroze has completed the boot process.

NOTE: The Net Yaroze development system for Macintosh does not have the option Boot Speed. This option only applies to the Windows development system, where it was necessary to provide backward compatibility with earlier PlayStation development systems.

The second baud rate pop-up is the speed that the debugger will use to communicate with the Net Yaroze while debugging. Typically, you should set this to the fastest speed that your computer can use reliably to talk to the Net Yaroze, either 57.6 kbps or 115 kbps. When the debugger connects to the Net Yaroze, it will automatically tell the Net Yaroze to change it's serial port speed from the boot speed to the debug speed.

TIP: If you install a memory card in slot 2 of your Net Yaroze, the debug baud rate will be saved in the card. The next time the Net Yaroze boots, it will default to that speed. If the boot speed and the debug speed are the same in the Connection Options, the debugger will connect to the Net Yaroze about 1-2 seconds faster.

NOTE: Not all Macintosh computers can support a serial speed of 115 kbps. The debugger will give you an error message if you choose 115 kbps and your computer does not support it.

Video Mode

The **Video Mode** radio buttons allow selection between NTSC and PAL video for the monitor when using a PCI card. Only one of these buttons can be selected at a time. When the NTSC button is on, the PlayStation development card generates NTSC video. When the PAL button is on, PAL video is generated by the card.

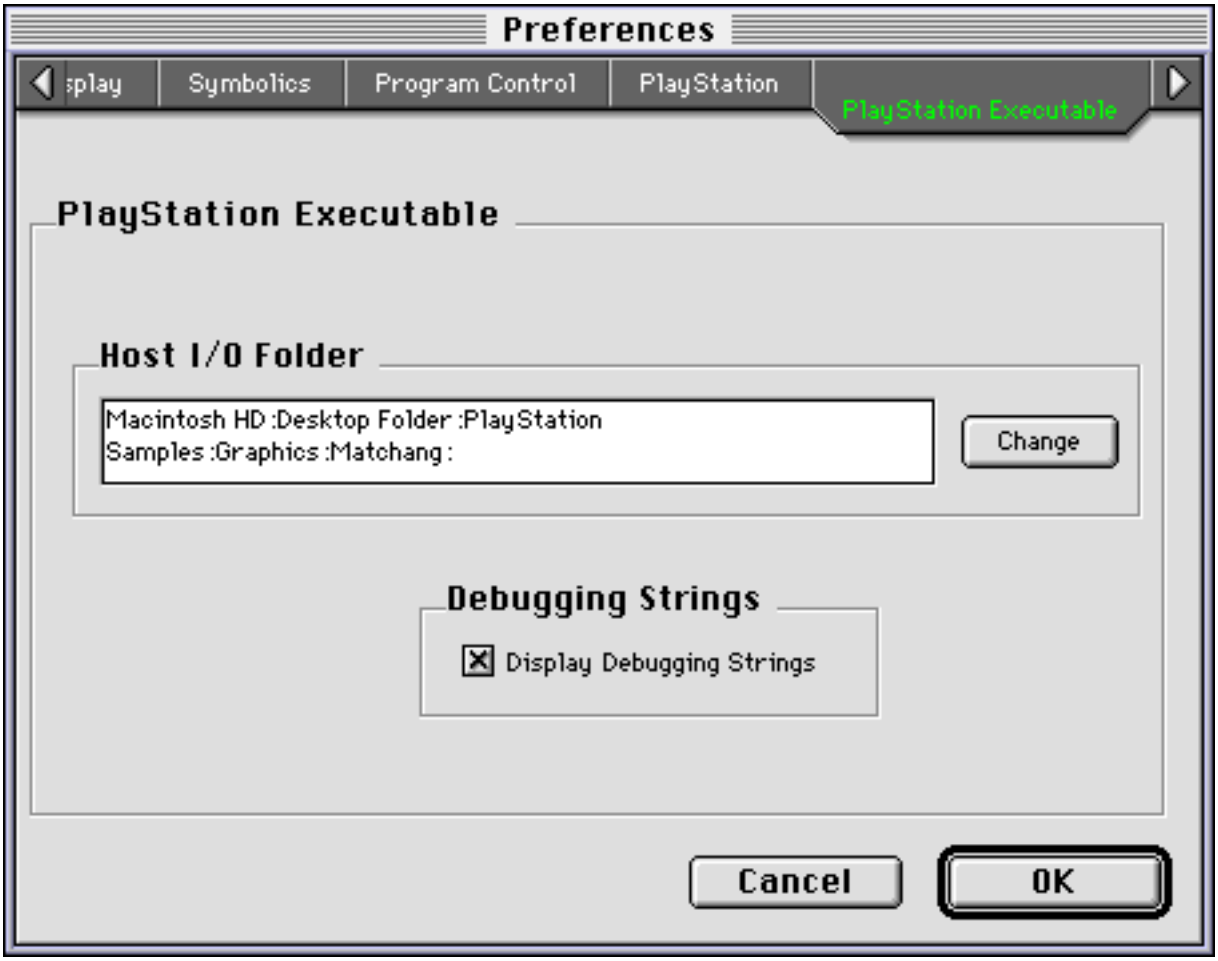
TIP: NTSC is the video format used in North America and Japan. The PAL video format is used in Europe.

NOTE: If you are using an ISA card on Windows, the NTSC/PAL option is selected via a hardware jumper on the ISA card.

PlayStation Executable Preference Panel

Figure 4.1 shows the **PlayStation Executable** panel.

Figure 4.1 PlayStation Executable dialog in the debugger preferences



The principal items in this dialog are:

- Host File I/O Folder
- Debugging Strings

Host File I/O Folder

The **Host File I/O Folder** section of the dialog identifies a directory used for file I/O in the debugging process.

To set or change this directory, click the **Change** button. When you do, a directory dialog appears that allows you to choose a new directory for file I/O.

See "Using MWDebugIO.lib" on page 73 for more information.

Debugging Strings

The **Display Debugging Strings** checkbox controls the display of debugging string information generated by your PlayStation application. Debug strings are generated in your PlayStation code by using the `printf` function.

When the checkbox is on, debugging strings are displayed in the debugger log window. When off, these strings are discarded.

Additional Debugging Features

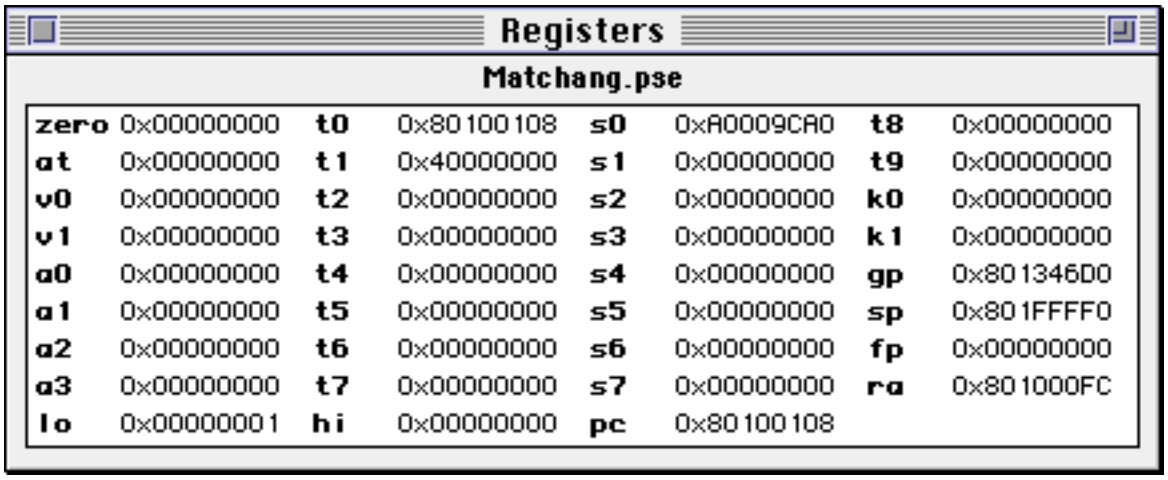
The PlayStation software debugger has certain additional features not present in the standard CodeWarrior debugger. Among these features are:

- Processor register window—displays the MIPS processor registers
- Transmission status display—shows progress when downloading to the PlayStation game console
- Metrowerks debugging library—greatly enhances your ability to work with PlayStation software

Processor register window

Figure 4.2 shows the MIPS processor register window. Each of the registers appears, along with its current value.

Figure 4.2 The MIPS processor register window



To see this window, choose the **Show Registers** item from the debugger's **Window** menu. To close the window, click in the close box or choose **Hide Registers** from the **Window** menu.

You can edit the value in any register. Click the value to select it, and enter a new value. Changing the value of a register can be very dangerous. Be careful.

For information on the registers, consult the *IDT R30xx Family Software Reference Manual* available from Integrated Device Technology, Inc., at http://www.idt.com.

Transmission status display

When you issue a run command from the debugger, the debugger downloads the executable code to the destination selected in the Debugger Preferences dialog.

When it does, a window appears showing you the progress of the download. This display is informational only. It has no other function.

Metrowerks debugging library

The Metrowerks debugging library provided for PlayStation software debugging adds significant and useful features to CodeWar-

rior. This library allows you to work with data files on your hard drive, reducing the need to prepare CD-ROM disks.

This library is fully discussed in “The Debugging Library for PlayStation Software” on page 65.

Limitations of the PlayStation Debugger

Some features present on other versions of CodeWarrior do not work the same way with the PlayStation debugger. These features are:

- Watchpoints
- Kill Command

Watchpoints

Watchpoints, while functional when debugging PlayStation applications, will seriously degrade the execution speed of the program you are debugging. Your application will execute approximately two orders of magnitude slower while watchpoints are active.

Use of watchpoints is not recommended, unless absolutely necessary to debug your application. Watchpoints are not available at all for Net Yaroze development hardware.

Kill Command

The Net Yaroze development hardware currently has no provision for killing applications from the debugger. Users of the Net Yaroze system should press the Reset button on the Net Yaroze console to kill a program.

Unimplemented Debugger Features

Certain features of the standard CodeWarrior debugger are unimplemented for PlayStation software debugging. Among these features are:

- Stop command
- Floating point registers

A few other features that are clearly irrelevant to PlayStation software development are disabled as well. For example, the **Break on Java exceptions** item in the debugger **Control** menu is disabled.

Stop command

The debugger's Stop command interrupts execution at the earliest opportunity after you issue the command.

For PlayStation software debugging, the Stop command is unimplemented at this time. The **Stop** button in the debugger tool bar and the **Stop** item from the debugger **Control** menu are non-functional.

To stop your executable, use a breakpoint instead.

Floating point registers

Because the MIPS processor has no floating point co-processor, the **Show FPU Registers** item in the debugger **Window** menu is disabled. There are no registers to display.

The Debugging Library for PlayStation Software

The Metrowerks debugging library for PlayStation software allows you to access data files resident on the host development computer. This allows you to quickly write and test games that rely on data files without creating CD-ROM disks containing the data files.

This section fully describes the debugging library. There are really two sets of functions in this library. One set is designed to be compatible with the file I/O functions resident in the PlayStation OS.

The other set is designed to be backward compatible with functions implemented in the Psy-Q™ development environment. All of these functions are in a library file named `MWDebugIO.lib`.

The topics in this section are:

- OS-Compatible File I/O Functions
- Psy-Q™ Compatible File I/O Functions
- Metrowerks Additional File I/O Functions
- Using MWDebugIO.lib

OS-Compatible File I/O Functions

The PlayStation OS has a series of functions that support data file access. These are designed to work only with data files on the game CD-ROM or on attached devices (such as backup memory). This can make game development difficult. It means you must create a CD-ROM disk every time you modify files.

The OS-compatible functions in `MWDebugIO.lib` are designed to be argument-compatible with the I/O services provided by the Play-Station OS. This makes using the OS-compatible functions very simple. See "Using MWDebugIO.lib" on page 73 for more information.

Each of the library functions is identical in name and argument list to the corresponding PlayStation OS function with one exception. The library function begins with the letters "MW."

This section is a reference to the functions. Table 4.2 lists each function and its purpose. After that, each function is detailed with prototype, parameters, return value, and a brief description

Table 4.2 MWDebugIO.lib OS-compatible I/O functions

MW Function	OS Function	Purpose
`MWopen()`	`open()`	open a file
`MWclose()`	`close()`	close a file
`MWread()`	`read()`	read data

MW Function	OS Function	Purpose
`MWwrite()`	`write()`	write data
`MWlseek()`	`lseek()`	reposition file position pointer
MWRedirectIO()	none	select devices for which to redirect file I/O

MWopen()

Prototype:
```
unsigned long MWopen(char *devname,
                     unsigned long flag)
```

Parameter(s): `devname`—name of device and file to open
`flag`—mode to use when opening file, one of
`O_RDONLY`—open for reading only
`O_WRONLY`—open for writing only
`O_RDWR`—open for reading and writing
`O_CREAT`—create a new file

Returns: file descriptor if successful, -1 otherwise

Notes: opens a file for reading or writing

MWclose()

Prototype:
```
long MWclose(unsigned long fd)
```

Parameter(s): `fd`—file descriptor of file to close

Returns: the value of `fd` if successful, -1 if not

Notes: closes the previously opened file which corresponds to the file descriptor supplied

MWread()

Prototype:
```
long MWread(unsigned long fd,
            void *buf, long n)
```

Parameter(s): `fd`—descriptor of file from which to read data
`buf`—address of buffer in which to place data
`n`—number of bytes to read from file

Returns: actual number of bytes read if successful, -1 otherwise

Notes: reads data from an open file

MWwrite()

Prototype:
```
long MWwrite(unsigned long fd,
             void *buf, long n)
```

Parameter(s): `fd`—descriptor of file to which to write data
`buf`—address of buffer from which to get data
`n`—number of bytes to write to file

Returns: actual number of bytes written if successful, -1 otherwise

Notes: writes data to an open file

MWlseek()

Prototype:
```
long MWlseek(unsigned long fd,
             long offset, long flag)
```

Parameter(s): `fd`—file in which to reposition pointer
`offset`—amount to move file pointer
`flag`—how to interpret the offset, one of
`SEEK_SET`—from start of file
`SEEK_CUR`—from current position

Returns: current (absolute) value of file pointer if successful, -1 if not

Notes: repositions the current read / write file position pointer in the specified file

MWRedirectIO()

Prototype: `void MWRedirectIO(unsigned long options)`

Parameter(s): `options`—devices for which file I / O should be redirected to host, pass zero to turn off redirection, or use the logical OR of
`__MWIO_CDROM`—redirects the "cdrom:" device
`__MWIO_MEMCARD`—redirects the "bu:" device

Returns: void

Notes: initializes the Metrowerks PlayStation software debugging library, and selects devices for which file I / O is redirected to the host computer

Psy-Q™ Compatible File I/O Functions

The Psy-Q development environment implemented a library similar in concept to `MWDebugIO.lib`. However, the Psy-Q functions do not mimic the PlayStation OS file I / O functions. In addition, the Psy-Q functions always access the host system. They do not allow you to turn off access to the host system, or redirect access for particular devices.

`MWDebugIO.lib` includes functions identical to the Psy-Q funtions. Each function name begins with the letters "PC." This allows you to use code that you wrote for the Psy-Q environment without making any changes, although the "MW" functions listed in Table 4.2 are more useful.

While the library functions have the same name as and are argument-compatible with the corresponding Psy-Q functions, these library functions are not argument-compatible with either the PlayStation OS or the other functions in the `MWDebugIO.lib` library.

This section is a reference to the `MWDebugIO.lib` functions that are Psy-Q-compatible. Table 4.3 lists each function and its purpose. After that, each function and its parameters are detailed with prototype, parameters, return type, and a brief description. For a full description of these functions, see the documentation that comes with the Psy-Q environment.

Table 4.3 MWDebugIO.lib Psy-Q-compatible I/O functions

Function	Purpose
`PCopen`	open a file on the host system
`PCclose()`	close a file on the host system
`PCread()`	read data
`PCwrite()`	write data
`PClseek()`	reposition file position pointer
PCcreat()	create a file on the host system

PCopen

Prototype:
```
unsigned long PCopen(char *name,
                     int flags, int perms)
```

Parameter(s): `name`—name of file to open
`flags`—mode to use when opening file, one of
`0`—open for reading only
`1`—open for writing only
`2`—open for reading and writing
`perms`—for UNIX compatibility, ignored

Returns: file descriptor if successful, -1 otherwise

Notes: opens a file on the host system for reading or writing

PCclose()

Prototype:
```
int PCclose(int fd)
```

Parameter(s): `fd`—file descriptor of file to close

Returns: negative value if unsuccessful

Notes: closes an open file on the host system

PCread()

Prototype:
```
long PCread(int fd, char *buff,
            int len)
```

Parameter(s): `fd`—descriptor of file from which to read data
`buff`—address of buffer in which to place data
`len`—number of bytes to read from file

Returns: actual number of bytes read if successful, -1 otherwise

Notes: reads data from an open file

PCwrite()

Prototype:
```
long PCwrite(int fd, char *buff,
             int len)
```

Parameter(s): `fd`—descriptor of file to which to write data
`buff`—address of buffer from which to get data
`len`—number of bytes to write to file

Returns: actual number of bytes written if successful, -1 otherwise

Notes: writes data to an open file

PClseek()

Prototype:
```
long PClseek(int fd, int offset,
             int mode)
```

Parameter(s): `fd`—file in which to reposition pointer
`offset`—amount to move file pointer

`mode`—how to interpret the offset, one of
`0`—from start of file
`1`—from current position
`2`—from end of file

Returns: current (absolute) value of file pointer if successful, -1 if not

Notes: repositions the current read / write file position pointer in the specified file

PCcreat()

Prototype:
```
int PCcreat(char *name, int perms)
```

Parameter(s): `name`—name of file to create
`perms`—permission flag, ignored

Returns: file descriptor if successful, -1 otherwise

Notes: creates a new file on the host system

Metrowerks Additional File I/O Functions

In addition to the debug file I/O functions that are compatible with those provided in the PlayStation OS and Psy-Q tools, CodeWarrior for PlayStation includes extra debugging I/O support.

NOTE: At this time, the additional debugging I/O support consists of `MWbload()`, found in `MWDebugIO.lib`

Table 4.4 MWDebugIO.lib Metrowerks Additional I/O function

Function	Purpose
`MWbload`	quickly loads a file into the memory of the host system

MWbload

Prototype:
```
long MWbload(char *devname,
             void *address)
```

Parameter(s):
```
devname-name of device and file to open
address-address at which to place loaded data
```

Returns: number of bytes read if successful, -1 otherwise

Notes: loads the entire contents of thenamed file into memory at the chosen address

Using MWDebugIO.lib

Using the `MWDebugIO.lib` requires that you:

1. add the `MWDebugIO.lib` library to your project
2. include the `MWDebugIO.h` library header file in any file that calls a library function
3. call `MWRedirectIO()` when you wish to redirect file I/O

You can redirect access to either the CD-ROM, the memory card, or both devices. To redirect access to the host system for both devices, the code would look like this:

```
MWRedirectIO( __MWIO_CDROM | __MWIO_MEMCARD);
```

To redirect one device, simply use that device's constant.

To turn off I/O redirection, that is, to access data files from the original device and not from the host system, call `MWRedirectIO()` and pass zero as the parameter, like this:

```
MWRedirectIO(0); // turn off file I/O redirection
```

You can then pick and choose which files you want to access from the host system, and which you want to access from the PlayStation game console hardware. Use the "MW" calls listed in Table 4.2 to work with data files on the host system.

Remember, all redirected file I/O works out of the directory specified in the Debugger Preferences dialog. See "Host File I/O Folder" on page 61 for details on how to set the directory.

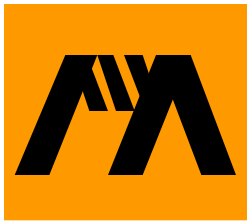
TIP: There's a very easy way to manage file I/O so that you use the host system during development, but you can switch to the real PlayStation game console hardware easily at any time.

Write all your code using the PlayStation OS functions. If you then define a symbol `DEBUG_IO`, the compiler will automatically change all your PlayStation OS I/O function calls to the corresponding `MWDebugIO.lib` function calls.

This transformation takes place in the `MWDebugIO.h` file, so you must define the constant before that file is included. A good place to define the constant would be in the `PSX_prefix.h` file. See "C/C++ Language Settings for MIPS" on page 81 for information about prefix files.

Simply undefine the `DEBUG_IO` constant to turn off the debugging library and use the PlayStation game console hardware.

5



MIPS Project Settings

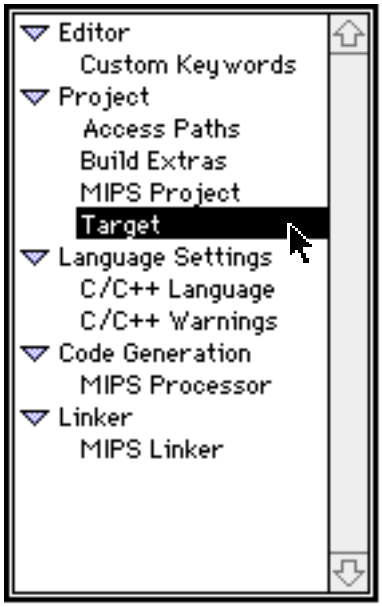
This chapter discusses the project settings for PlayStation software development with the MIPS compiler and linker.

Overview of MIPS Project Settings

You control compiler and linker behavior by modifying settings in various project settings panels. This chapter illustrates all panels that are specific to the MIPS processor used for PlayStation software development, and details the effect of each setting in each panel.

To open any settings panel, choose **Project Settings** from the **Edit** menu. When you do, the Project Settings dialog appears. On Mac OS (as shown in Figure 5.1) you select a panel from a hierarchical list of available panels. On Windows, select the tab for the panel of interest. When you select a panel, that panel's settings appear in the Project Settings dialog.

Figure 5.1 Choosing a settings panel for Mac OS



The Code Generation and Linker panels from which you can choose are dependent upon the target selection. Only panels appropriate for your target are available.

This chapter discusses each of the MIPS related panels: the Target panel, the MIPS project panel, the C / C++ Language panel, the MIPS processor panel, and the MIPS linker panel.

TIP: Use the PlayStation project stationery when you create a new project. The stationery already has all settings in all panels set to reasonable or default values. You can create your own stationery file with your preferred settings. Modify a new project to suit your needs. Then save it an place it in the stationery folder. See the *IDE User Guide* for details.

The topics are:

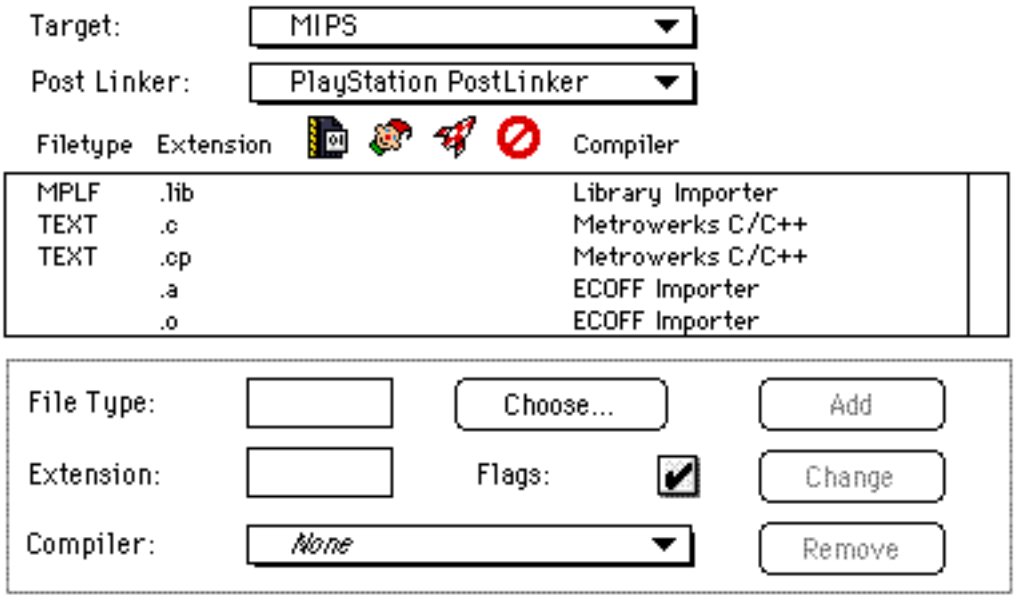
- Target Settings—choosing a target processor

- MIPS Project Settings—general environment and other settings
- C/C++ Language Settings for MIPS—setting the project prefix file
- MIPS Code Generation Settings—code generation settings for the MIPS compiler
- MIPS Linker Settings—settings for the MIPS linker
- PlayStation OS Address Spaces—information about legal addresses and how to set up memory

Target Settings

Target settings identify which processor or platform you intend to write code for. Figure 5.2 shows the target settings panel as it should appear for PlayStation software development.

Figure 5.2 Target settings for MIPS target



The settings of immediate interest to PlayStation software development are:

- Target
- Post Linker

The other settings and controls in this panel let you specify the relationship between a file and its compiler. How to use this panel to accomplish this task is fully explained in the *CodeWarrior IDE User Guide*.

Target

The **Target** pop-up menu specifies the chip or platform for which you want to compile code. Set this menu to MIPS, as shown in Figure 5.2. The items available to you in the Target pop-up menu depend upon the compilers you have available.

NOTE: You must set the target first, before changing other settings. Other MIPS-related settings panels are only available when MIPS is the selected target.

Post Linker

The **Post Linker** pop-up menu specifies a post-linker phase to modify the linker output.

The MIPS linker generates a debuggable version of your code. The linker generates a file with a name you set. You specify this name in the MIPS Project Settings with the File Name setting. By convention, this file should end with the extension `.pse`.

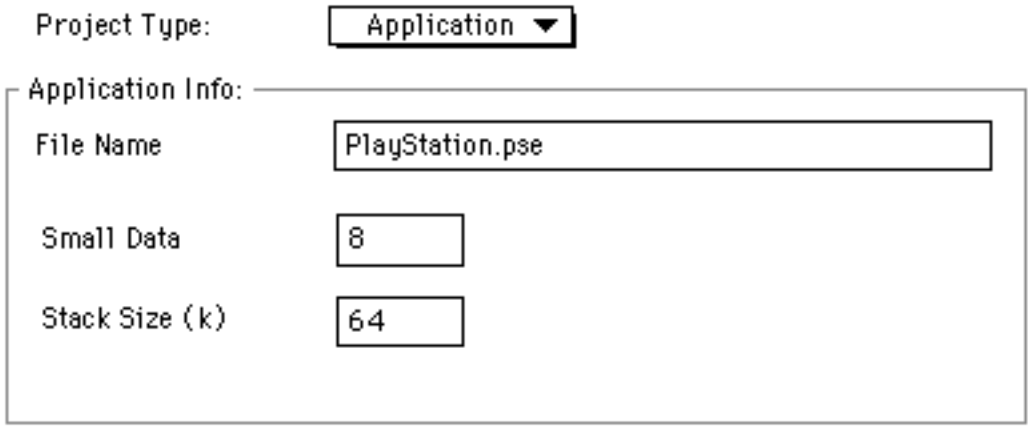
Set the Post Linker pop-up menu to *PlayStation PostLinker* when you want to prepare a project for a PlayStation CD-ROM, as shown in Figure 5.1. This post-linker converts the debuggable PlayStation executable prepared by CodeWarrior for placement on the CD-ROM, and performs the GTE instruction translation formerly done by the DMPSX utility. The post-linker adds the extension `.exe` to the linker's generated file name.

MIPS Project Settings

The MIPS project settings tell the compiler and linker certain things about the environment in which it is working, such as available memory, stack size, and so forth. This panel is only available when MIPS is the target processor. See “Target Settings” on page 77 for information on how to set the target.

Figure 5.3 shows the MIPS project settings panel.

Figure 5.3 MIPS project settings



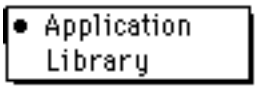
The items in this panel are:

- Project Type
- File Name
- Small Data
- Stack Size (k)

Project Type

The **Project Type** pop-up menu determines the kind of project you are creating. The available project types are shown in Figure 5.4.

Figure 5.4 MIPS project type options



Set this menu so that the selected menu item (visible in the panel when the pop-up menu is not open) reflects the kind of project you are building.

The option you choose in this pop-up menu also controls the visibility of other items in this panel. If you are making a Library project, the **Stack Size (k)** item disappears from this panel because it is not relevant.

File Name

The **File Name** edit field specifies the name of the debuggable executable, library, or layout you create. By convention, this name should end with the extension `.pse`.

Small Data

The **Small Data** edit field controls the threshold size (in bytes) for an item considered by the linker to be "small data." The linker stores small data items in the Small Data Address space. This space has faster access than the regular Data Address space.

Items whose byte size is less than or equal to the value in the **Small Data** edit field are considered to be small data.

See also: "PlayStation OS Address Spaces" on page 88.

Stack Size (k)

The **Stack Size** edit field controls the amount of RAM allocated for the stack. The value you enter is in kilobytes.

The stack size value is used by the linker for two reasons. One is to determine whether the various PlayStation OS address spaces collide. This value is also used to compute the size of the application

heap, since the heap occupies the space between the end of your application's code and data, and the bottom of the stack.

WARNING! The PlayStation OS has no method of verifying that your stack doesn't exceed the space you allocated. It is possible for the stack to grow beyond the space allowed, and overwrite items in the adjacent space with unpredictable results. Use caution when setting the stack size value.

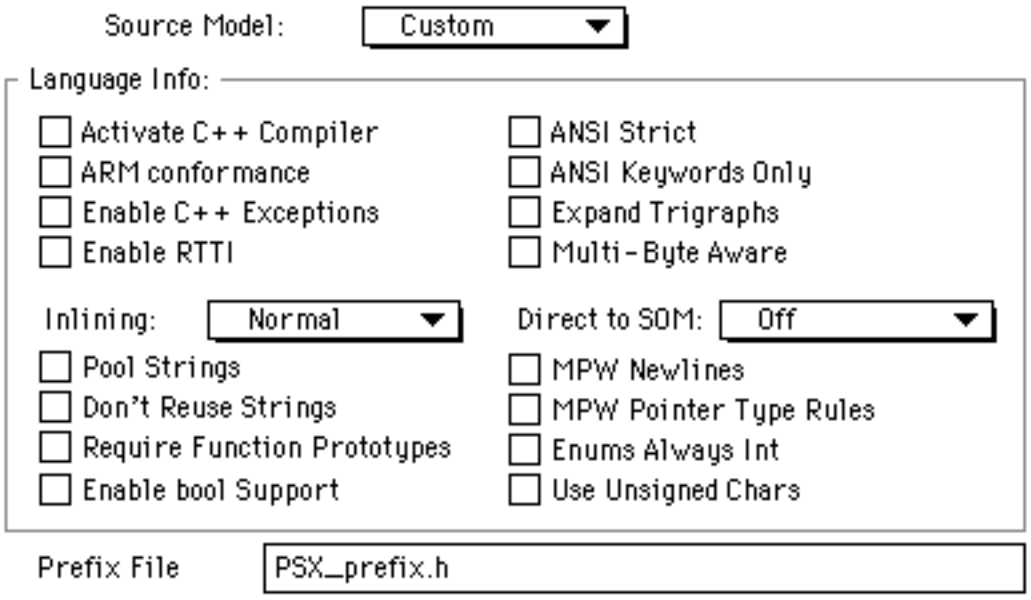
This setting is available when the **Project Type** setting is either Application or Layout. This setting is not available when the **Project Type** setting is Library.

See also: "PlayStation OS Address Spaces" on page 88.

C/C++ Language Settings for MIPS

The C/C++ language settings tells the compiler what rules to follow with respect to the programming language. Figure 5.5 shows the C/C++ language settings panel.

Figure 5.5 C/C++ language settings



All of the options in this panel are fully explained in the *CodeWarrior IDE User Guide*. However, the **Prefix File** item at the bottom of the panel deserves special note.

In CodeWarrior, the prefix file is automatically included in all source files. The contents of the prefix file are usually designed to perform some universal function. Then you don't have to repeat the code in a series of header files, or remember to include a special header file in every single source file.

For PlayStation software development, the default prefix file is `PSX_prefix.h`. If you use the PlayStation project stationery, this file is set for you. You can create and use your own prefix file if you wish. Simply set the name properly in this option.

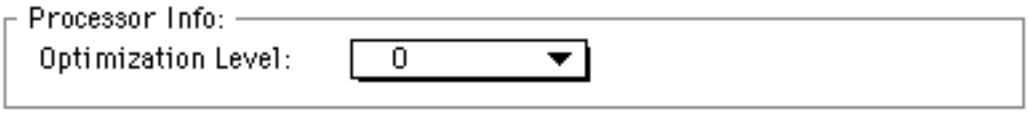
MIPS Code Generation Settings

The MIPS processor code generation settings determine what kind of code the compiler creates. This panel is only available when MIPS

is the target processor. See “Target Settings” on page 77 for information on how to set the target.

Figure 5.6 shows the MIPS processor code generation settings panel.

Figure 5.6 MIPS processor code generation settings



The only item in this panel is:

- Optimization Level

Optimization Level

The **Optimization Level** pop-up menu determines the level of optimization applied by the MIPS compiler. The available optimization levels are shown in Figure 5.7. They are simply numbers from 0 to 4.

Figure 5.7 Optimization level pop-up menu

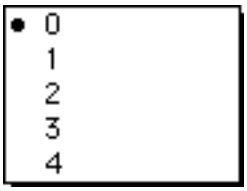


Table 5.1 details the effect of various optimization levels.

Table 5.1 MIPS optimizer levels

Level	Effect	Debugging
0	none	safe
1	simple optimizations	safe
2	aggressive optimization	safe

Level	Effect	Debugging
3	simple optimizations	not safe
4	aggressive optimizations	not safe

Levels 0-2 are safe for debugging. That is, choosing one of these optimization levels will not break the one-to-one correspondence between source code an object code. Levels 1-2 operate on a small area of code.

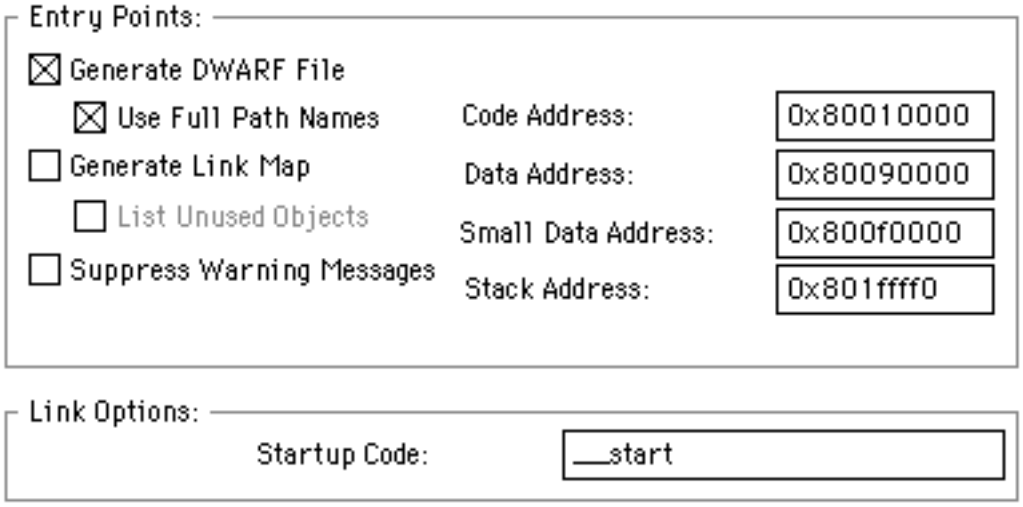
Levels 3-4 are analogous to levels 1-2. They perform the same kinds of optimizations, but operate on a larger area of code. As a result, you are likely to have difficulty debugging code optimized at levels 3-4. The optimizer may and probably will reorganize object code so that it no longer corresponds in a one-to-one relationship with source code. This can cause problems with breakpoints.

MIPS Linker Settings

The MIPS linker settings determine how the linker operates. This panel is only available when MIPS is the target processor. See “Target Settings” on page 77 for information on how to set the target.

Figure 5.8 shows the MIPS linker settings panel.

Figure 5.8 MIPS linker settings



The items in this panel are:

- Generate DWARF File
- Use Full Path Names
- Generate Link Map
- List Unused Objects
- Suppress Warning Messages
- Code Address
- Data Address
- Small Data Address
- Stack Address
- Startup Code

Generate DWARF File

The **Generate DWARF File** check box controls whether the linker generates debugging information.

When this setting is on the linker generates debugging information. The debugger information is included within the linked ELF file.

This setting does not generate a separate file. When this setting is off the linker does not generate debugging information.

When you choose the **Enable Debugger** item in the CodeWarrior **Project** menu, CodeWarrior turns this item on for you.

If this item is off, you will not be able to see source code in the debugger.

Use Full Path Names

The **Use Full Path Names** check box controls how the linker includes path information for source files in the debugging information.

When this setting is on, the linker includes full path names to the source files. When this setting is off, the linker uses project-relative path names. In typical usage, this setting is on.

This item is disabled and unavailable unless the Generate DWARF File setting is on.

Generate Link Map

The **Generate Link Map** check box controls whether the linker generates a link map.

When this setting is on the linker generates a link map. When this setting is off, the linker does not generate a link map.

The file name for the link map adds the extension `.xmap` to the linker's generated file name. See "File Name" on page 80. The file is put in the same folder as the CodeWarrior project file.

List Unused Objects

The **List Unused Objects** check box controls whether the linker includes unused objects in the link map.

When this setting is on, the linker includes unused objects. When this setting is off, the linker does not include unused objects in the link map.

Typically this item is off. However, you may want to turn it on in certain cases. For example, you may discover that an object you expect to be used is not in fact in use.

This item is disabled and unavailable unless the Generate Link Map setting is on.

Suppress Warning Messages

The **Suppress Warning Messages** check box controls whether the linker displays warnings.

When this setting is on, the linker will display warnings in the CodeWarrior message window. When this setting is off, the linker will not display warnings.

In typical usage, this setting is on.

Code Address

The **Code Address** edit field specifies the location in memory where the executable code resides.

You must specify the address in hexadecimal notation. See “PlayStation OS Address Spaces” on page 88 for details on legal addresses.

Data Address

The **Data Address** edit field specifies the location in memory where the program’s global data resides.

You must specify the address in hexadecimal notation. See “PlayStation OS Address Spaces” on page 88 for details on legal addresses.

If you specify the data address value as zero, the linker will calculate the value for you.

Small Data Address

The **Small Data Address** edit field specifies the location in memory where the small data resides.

You must specify the address in hexadecimal notation. See "PlayStation OS Address Spaces" on page 88 for details on legal addresses.

If you specify the small data address value as zero, the linker will calculate the value for you.

Stack Address

The **Stack Address** edit field specifies the location in memory where the program's stack resides.

You must specify the address in hexadecimal notation. The address you specify is the *top* of the stack.

See "PlayStation OS Address Spaces" on page 88 for details on legal addresses.

Startup Code

The **Startup Code** edit field specifies the function that the linker uses first when the program launches. This is the program's entry point.

The default `__start` function is Metrowerks bootstrap or glue code that sets up the environment before your code executes. This function is in the `_psstart.c` file. The final task performed by `__start` is to call your `main()` function.

PlayStation OS Address Spaces

The PlayStation PCI and ISA card-based development systems have eight megabytes of available RAM, addressed as 0x8000 0000 to 0x807F FFFF. The Hobbyist development system and the consumer PlayStation game console have two megabytes of available RAM, addressed as 0x8000 0000 to 0x801F FFFF.

Ultimately, the code you write must work within the smaller, two-megabyte space available in the PlayStation game console.

Within this memory space, PlayStation OS recognizes four principal memory address spaces. They are:

- code address space
- data address space
- small data address space
- stack address space

You specify the beginning point of each space in the MIPS Linker Settings panel. For the stack address space, you specify the top of the stack. For the other spaces, you specify the lowest point in memory at the beginning of the space.

There are two requirements that each address must meet.

- All addresses must be evenly divisible by four.
- The lowest legal address is 0x8001 0000 for the PCI and ISA cards. The lowest legal address for the Hobbyist development console is 0x8009 0000. The PlayStation OS reserves the memory below these respective addresses for itself.

Within these limitations, the address spaces can reside in any section of memory, in any order.

No matter what addresses you use, the small data address space will use no more than 64K of memory.

When the linker builds your code, it will determine if the code, data, small data, or stack spaces overlap (collide). If there is a collision, you will receive a linker error. You can then adjust the addresses so that everything fits.

To determine whether the stack address space collides with any other, the linker uses the **Stack Size (k)** setting in the MIPS Project Settings panel. You can still exceed the stack size at runtime and overwrite whatever is below the stack in memory, with unpredictable (but usually disastrous) results.

See also:

- “Small Data” on page 80
- “Stack Size (k)” on page 80
- “Code Address” on page 87

- “Data Address” on page 87
- “Small Data Address” on page 87
- “Stack Address” on page 88

For more information on how the PlayStation OS uses memory, you should consult the PlayStation OS documentation.

6

Troubleshooting

The goal of this chapter is to provide solutions to the common programming problems found when using the CodeWarrior for PlayStation tools.

Overview of TroubleShooting

If you run into problems with the CodeWarrior for PlayStation tools and find a useful solution, Metrowerks would be glad to publish your advice here. Please share your tips for using the CodeWarrior tools by sending email to <support@metrowerks.com>.

The only topic in this chapter is:

- Frequently Asked Questions

Frequently Asked Questions

This section covers general problems you may encounter.

No Video Output

Problem

My code runs, but I can’t see the video output.

Solution

Check the video cables to be certain they’re connected to the PCI card. Also, be sure that your monitor has power and is turned on.

For more information, review “Making Sure It Works” on page 20.

File I/O Problems

Problem

I'm using File I/O from my hard drive while debugging. The code can't find my files.

Solution

See "Debugger Preferences" on page 57. Make sure that the debugger preferences (especially the *Host File I/O Folder* preference) are set correctly.

Downloading Code Fails

Problem

I can't download my code to the PlayStation card.

Solution

Make sure that your address spaces are set correctly. See "PlayStation OS Address Spaces" on page 88 to learn what memory address spaces are allowed for PlayStation software development. See also "MIPS Linker Settings" on page 84, which shows where to set the address spaces.

Crashes When Code Runs

My PlayStation game console crashes immediately when I try to run my code.

Solution

Make sure that your address spaces are set correctly. See "PlayStation OS Address Spaces" on page 88 to learn what memory address spaces are allowed for PlayStation software development. See also "MIPS Linker Settings" on page 84, which shows where to set the address spaces.

7

Using Assembly

This chapter steps you through the process of incorporating MIPS assembly language routines into your PlayStation OS code.

Overview of Using Assembly

In this chapter, our focus is to explain the current Metrowerks implementation for the use of assembly language in your MIPS code. We use a tutorial to show, with example code, how you mix assembly language code with your C or C++ code.

The only topic in this chapter is:

- MIPS Intrinsic Functions

MIPS Intrinsic Functions

Metrowerks C / C++ for MIPS provides intrinsic functions to generate inline MIPS instructions. These intrinsic functions are faster than other functions, since the compiler translates them into inline assembly instructions instead of function calls.

NOTE: These intrinsic functions are not part of the ANSI C or C++ standards. The intrinsic functions are available only with the Metrowerks C / C++ for MIPS compiler.

The parts of this section include:

- Using Intrinsic Functions
- Special Case: __evaluate

Using Intrinsic Functions

Metrowerks has added an intrinsic function for each instruction in the MIPS R3000 instruction set. The format of the intrinsic function is:

`__I_<name> (argument list)`

- `__I_` appears before each assembler instruction's opcode word.
- `<name>` is the opcode word of the assembly instruction you want to use.
- `(argument list)` is a list of the arguments that are used with the MIPS assembly instruction you choose.

For example, when using the MIPS instruction `addi`, we have:

MIPS Instruction	`addi r0,r1,2`
Metrowerks intrinsic function	`__I_addi (r0, r1, 2)`
All other compilers	`asm ("addi r0, r1, 2")`

NOTE: The intrinsic function solution is a temporary one that works today. We are currently implementing inline assembly as shown in the last example.

Special Case: __evaluate

The intrinsic function __evaluate is a utility function that allows you to pass C/C++ variables and expressions into inline assembly. This makes it easier to use built-in assembly instructions. The format of this intrinsic function is:

`__evaluate (<virtual variable>, <evaluation>)`

- `__evaluate` is required for the compiler to know the presence of an argument.
- `<virtual variable>` is a virtual register created by the compiler.

- `<evaluation>` is the area for evaluating a C/C++ variable or expression.

Here is an example piece of code that uses both the `__evaluate` and normal intrinsic functions.

Note the presence of `__asm_start()` and `__asm_end()`. These markers tell the compiler to lock the values contained in the virtual variables (such as `__arg0`). These values are unlocked once `__asm_end()` is read by the compiler. It is only after `__asm_end()` that these virtual variables can be overwritten with new values.

```
#define ASMcode (Param1, Param2, Param3) \
  __evaluate (__arg0, (long)Param1);\
  __evaluate (__arg1, (long)Param2);\
  __evaluate (__arg2, (long)Param3);\
  __asm_start();\
  __I_lwc2 (0, 0, __arg0);\
  __I_lwc2 (1, 4, __arg0);\
  __I_lwc2 (2, 0, __arg1);\
  ...\
  __asm_end();
```

`__evaluate` will evaluate the parameter Param into a virtual hardware register `__arg0`. Note that `__arg0` is NOT `r0`--it is just a general purpose register. The compiler will make sure that the hardware register used for `__argi` is not destroyed between `__asm_start()` and `__asm_end()`.

Moreover, no dangerous optimization will take place between `__asm_start()` and `__asm_end()`.

For example:

```
  struct {
    int x;
    int y;
```

```
} POINT

__evaluate (__arg0, (long)&POINT);
```

will put `&POINT` into the virtual register `__arg0`.

```
__evaluate (__arg1, (long)POINT.x);
```

will put the contents of `POINT.x` into the virtual register `__arg1`.

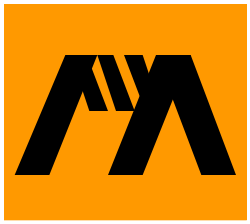
```
__I_lw (12, 4, __arg0);
```

will load into register 12 the contents of `POINT.y` (4 is the offset and `__arg0` is the address of the object).

For more information, see `intrinsics.h`, which is located in the Include folder on the CodeWarrior for PlayStation CD. For more information on MIPS assembler instructions, the following book is recommended:

MIPS Risc Architecture, by Gerry Kane and Joe Heinrich, published by Prentice Hall, 1992 (ISBN: 0-13-590472-2)

8



Metrowerks Utility Library

This chapter describes `MWUtils.lib`, the first Metrowerks custom library for PlayStation OS development.

Overview of Metrowerks Utility Library

Programming for the PlayStation game console requires libraries for dealing with opening, saving and closing files. Sony has provided file I/O library functions to facilitate this.

To make programming for the PlayStation OS as simple as possible, Metrowerks has written `MWUtils.lib`. This release of the library contains one function, `bload()`, which allows you to do a fast binary load.

NOTE: At this time, `bload()` is the only CodeWarrior for PlayStation utility function available. More functions are in development at this time. They will be documented as they become available.

bload()

The function bload() opens and loads an entire file into memory.

Prototype:

```
long bload(char *devname,
           void *address)
```

Parameter(s): `devname-name of device and file to open`
`address-address at which to place loaded data`

Returns: number of bytes read if successful, -1 otherwise

Notes: loads the entire contents of the named file into memory at the chosen address

For information on the debug version of this function, see Chapter 4, “Metrowerks Additional File I/O Functions” on page 72.

Index

CodeWarrior

Targeting PlayStation OS

Credits

writing lead: John Roseborough

other writers: Jim Trudeau

engineering: Bobby Clarke, Pascal Cleve, Matt Cole, Soren Groenbech, Fred Peterson, Khurram Qureshi, Peter Speck, Joel Sumner, Lawrence You

frontline warriors: Bobby Clarke, Pascal Cleve, Matt Cole, Joel Sumner

Guide to CodeWarrior Documentation

If you need information about...	See this
Installing updates to CodeWarrior	*QuickStart Guide*
Getting started using CodeWarrior	*QuickStart Guide*; *Tutorials* (Apple Guide)
Using CodeWarrior IDE (Integrated Development Environment)	*IDE User's Guide*
Debugging	*Debugger Manual*
Important last-minute information on new features and changes	*Release Notes* folder
Controlling CodeWarrior through AppleScript	*IDE User's Guide*
C, C++ programming	*C, C++, and Assembly Reference*; *MSL C Reference*; *MSL C++ Reference*
Fixing compiler and linker errors	*Errors Reference*
Contacting Metrowerks about registration, sales, and licensing	*Quick Start Guide*
Contacting Metrowerks about problems and suggestions using CodeWarrior software	*email Report Forms* in the *Release Notes* folder
Problems other CodeWarrior users have solved	*Internet newsgroup [docs]* folder

Revision code 97/01/23 jet