Sourcery G++ Lite

MIPS ELF

Sourcery G++ Lite 4.4-58

Getting Started

Sourcery G++ Lite: MIPS ELF: Sourcery G++ Lite 4.4-58: Getting Started

CodeSourcery, Inc.

Abstract

This guide explains how to install and build applications with Sourcery G++ Lite, CodeSourcery's customized, validated, and supported version of the GNU Toolchain. Sourcery G++ Lite includes everything you need for application development, including C and C++ compilers, assemblers, linkers, and libraries.

When you have finished reading this guide, you will know how to use Sourcery G++ from the command line.

Table of Contents

Preface

This preface introduces the Sourcery G++ Lite Getting Started guide. It explains the structure of this guide and describes the documentation conventions used.

1. Intended Audience

This guide is written for people who will install and/or use Sourcery G++ Lite. This guide provides a step-by-step guide to installing Sourcery G++ Lite and to building simple applications. Parts of this document assume that you have some familiarity with using the command-line interface.

2. Organization

This document is organized into the following chapters and appendices:

Chapter 1, “Quick Start”	This chapter includes a brief checklist to follow when installing and using Sourcery G++ Lite for the first time. You may use this chapter as an abbreviated guide to the rest of this manual.
Chapter 2, “Installation and Configuration”	This chapter describes how to download, install and configure Sourcery G++ Lite. This section describes the available installation options and explains how to set up your environment so that you can build applications.
Chapter 3, “Sourcery G++ Lite for MIPS ELF”	This chapter contains information about using Sourcery G++ Lite that is specific to MIPS ELF targets. You should read this chapter to learn how to best use Sourcery G++ Lite on your target system.
Chapter 4, “Using Sourcery G++ from the Command Line”	This chapter explains how to build applications with Sourcery G++ Lite using the command line. In the process of reading this chapter, you will build a simple application that you can use as a model for your own programs.
Chapter 5, “CS3™: The CodeSourcery Common Startup Code Sequence”	CS3 is CodeSourcery's low-level board support library. This chapter describes the organization of the system startup code and tells you how you can customize it, such as by defining your own interrupt handlers. This chapter also documents the boards supported by Sourcery G++ Lite and the compiler and linker options you need to use with them.
Chapter 6, “Next Steps with Sourcery G++”	This chapter describes where you can find additional documentation and information about using Sourcery G++ Lite and its components. It also provides information about Sourcery G++ subscriptions. CodeSourcery customers with Sourcery G++ subscriptions receive comprehensive support for Sourcery G++.
Appendix A, “Sourcery G++ Lite Release Notes”	This appendix contains information about changes in this release of Sourcery G++ Lite for MIPS ELF. You should read through these notes to learn about new features and bug fixes.
Appendix B, “Sourcery G++ Lite Licenses”	This appendix provides information about the software licenses that apply to Sourcery G++ Lite. Read this appendix to understand your legal rights and obligations as a user of Sourcery G++ Lite.

3. Typographical Conventions

The following typographical conventions are used in this guide:

`> command arg ...`	A command, typed by the user, and its output. The “`>`” character is the command prompt.
command	The name of a program, when used in a sentence, rather than in literal input or output.
`literal`	Text provided to or received from a computer program.
`placeholder`	Text that should be replaced with an appropriate value when typing a command.
`\`	At the end of a line in command or program examples, indicates that a long line of literal input or output continues onto the next line in the document.

Chapter 1
Quick Start

This chapter includes a brief checklist to follow when installing and using Sourcery G++ Lite for the first time. You may use this chapter as an abbreviated guide to the rest of this manual.

Sourcery G++ Lite for MIPS ELF is intended for developers working on embedded applications or firmware for boards without an operating system, or that run an RTOS or boot loader. This Sourcery G++ configuration is not intended for Linux or uClinux kernel or application development.

Follow the steps given in this chapter to install Sourcery G++ Lite and build and run your first application program. The checklist given here is not a tutorial and does not include detailed instructions for each step; however, it will help guide you to find the instructions and reference information you need to accomplish each step.

You can find additional details about the components, libraries, and other features included in this version of Sourcery G++ Lite in Chapter 3, "Sourcery G++ Lite for MIPS ELF".

1.1. Installation and Set-Up

Install Sourcery G++ Lite on your host computer. You may download an installer package from the Sourcery G++ web site[1], or you may have received an installer on CD. The installer is an executable program that pops up a window on your computer and leads you through a series of dialogs to configure your installation. If the installation is successful, it will offer to launch the Getting Started guide. For more information about installing Sourcery G++ Lite, including host system requirements and tips to set up your environment after installation, refer to Chapter 2, "Installation and Configuration".

Install drivers for your debug device. Sourcery G++ Lite supports third-party debug devices that communicate via the GDB remote serial protocol. If you plan to use one of these devices, follow the manufacturer's directions to connect the device and install any required drivers or software.

1.2. Configuring Sourcery G++ Lite for the Target System

Identify your target board. On bare-metal targets, you must explicitly specify a linker script for your target board on your link command line. Supported boards are listed in Chapter 5, "CS3™: The CodeSourcery Common Startup Code Sequence". You can also choose a simulator as your target board.

1.3. Building Your Program

Build your program with Sourcery G++ command-line tools. Create a simple test program, and follow the directions in Chapter 4, "Using Sourcery G++ from the Command Line" to compile and link it using Sourcery G++ Lite. On bare-metal targets, you must specify a linker script using the `-T` option on your link command line. Supported boards and linker scripts are listed in Chapter 5, "CS3™: The CodeSourcery Common Startup Code Sequence".

1.4. Running and Debugging Your Program

The steps to run or debug your program depend on your target system and how it is configured. Choose the appropriate method for your target.

Run or debug your program in the simulator. Sourcery G++ Lite includes an instruction-set simulator, which provides an easy way to run or debug your program without requiring target hard-

[1] http://www.codesourcery.com/gnu_toolchains/

ware. The simulator can be run directly from the command line (see Section 4.3, “Running Applications in the Simulator”) or via the debugger (see Section 4.4, “Running Applications from GDB”).

Debug your program on the target using MDI. You can load and execute your program on the remote target from the debugger using the MDI protocol. Refer to Section 4.4.3, “Connecting with MDI” for instructions on configuring MDI and using MDI from the GDB command line.

Run your program on the target using YAMON. You can run programs built with Sourcery G++ Lite on MIPS ELF targets via the YAMON boot monitor. For instructions, refer to Section 3.5, “Using Sourcery G++ with YAMON”. Note that you must select a YAMON linker script profile when building your program.

Debug your program on the target using a third-party debug device. Sourcery G++ supports debugging programs on the remote target using third-party debug devices that can communicate via the GDB remote serial protocol. For command-line GDB instructions, see Section 4.4, “Running Applications from GDB”.

Chapter 2
Installation and Configuration

This chapter explains how to install Sourcery G++ Lite. You will learn how to:

1. Verify that you can install Sourcery G++ Lite on your system.
2. Download the appropriate Sourcery G++ Lite installer.
3. Install Sourcery G++ Lite.
4. Configure your environment so that you can use Sourcery G++ Lite.

2.1. Terminology

Throughout this document, the term *host system* refers to the system on which you run Sourcery G++ while the term *target system* refers to the system on which the code produced by Sourcery G++ runs. The target system for this version of Sourcery G++ is `mips-sde-elf`.

If you are developing a workstation or server application to run on the same system that you are using to run Sourcery G++, then the host and target systems are the same. On the other hand, if you are developing an application for an embedded system, then the host and target systems are probably different.

2.2. System Requirements

2.2.1. Host Operating System Requirements

This version of Sourcery G++ supports the following host operating systems and architectures:

- Microsoft Windows NT 4, Windows 2000, Windows XP, and Windows Vista systems using IA32, AMD64, and EM64T processors.
- GNU/Linux systems using IA32, AMD64, or EM64T processors, including Debian 3.1 (and later), Red Hat Enterprise Linux 3 (and later), and SuSE Enterprise Linux 8 (and later).

Sourcery G++ is built as a 32-bit application. Therefore, even when running on a 64-bit host system, Sourcery G++ requires 32-bit host libraries. If these libraries are not already installed on your system, you must install them before installing and using Sourcery G++ Lite. Consult your operating system documentation for more information about obtaining these libraries.

Installing on Ubuntu and Debian GNU/Linux Hosts

The Sourcery G++ graphical installer is incompatible with the **dash** shell, which is the default `/bin/sh` for recent releases of the Ubuntu and Debian GNU/Linux distributions. To install Sourcery G++ Lite on these systems, you must make `/bin/sh` a symbolic link to one of the supported shells: **bash**, **csh**, **tcsh**, **zsh**, or **ksh**.

For example, on Ubuntu systems, the recommended way to do this is:

```
> sudo dpkg-reconfigure -plow dash
Install as /bin/sh? No
```

This is a limitation of the installer and uninstaller only, not of the installed Sourcery G++ Lite toolchain.

2.2.2. Host Hardware Requirements

In order to install and use Sourcery G++ Lite, you must have at least 128MB of available memory.

The amount of disk space required for a complete Sourcery G++ Lite installation directory depends on the host operating system and the number of target libraries included. Typically, you should plan on at least 400MB.

In addition, the graphical installer requires a similar amount of temporary space during the installation process. On Microsoft Windows hosts, the installer uses the location specified by the `TEMP` environment variable for these temporary files. If there is not enough free space on that volume, the installer prompts for an alternate location. On Linux hosts, the installer puts temporary files in the directory specified by the `IATEMPDIR` environment variable, or `/tmp` if that is not set.

2.2.3. Target System Requirements

See Chapter 3, "Sourcery G++ Lite for MIPS ELF" for requirements that apply to the target system.

2.3. Downloading an Installer

If you have received Sourcery G++ Lite on a CD, or other physical media, then you do not need to download an installer. You may skip ahead to Section 2.4, "Installing Sourcery G++ Lite".

You can download Sourcery G++ Lite from the Sourcery G++ web site[1]. This free version of Sourcery G++, which is made available to the general public, does not include all the functionality of CodeSourcery's product releases. If you prefer, you may instead purchase or register for an evaluation of CodeSourcery's product toolchains at the Sourcery G++ Portal[2]. For more information about subscriptions for Sourcery G++ product releases, see Section 6.1, "Sourcery G++ Subscriptions".

Once you have navigated to the appropriate web site, download the installer that corresponds to your host operating system. For Microsoft Windows systems, the Sourcery G++ installer is provided as an executable with the `.exe` extension. For GNU/Linux systems Sourcery G++ Lite is provided as an executable installer package with the `.bin` extension. You may also install from a compressed archive with the `.tar.bz2` extension.

On Microsoft Windows systems, save the installer to the desktop. On GNU/Linux systems, save the download package in your home directory.

2.4. Installing Sourcery G++ Lite

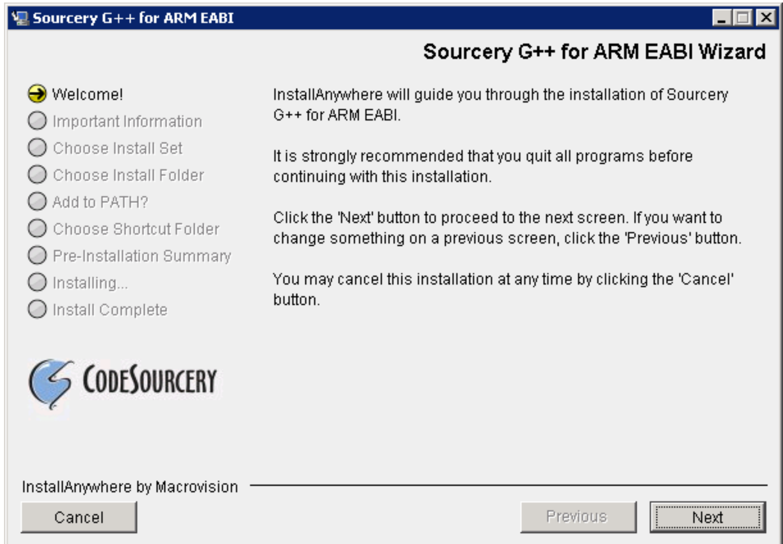
The method used to install Sourcery G++ Lite depends on your host system and the kind of installation package you have downloaded.

2.4.1. Using the Sourcery G++ Lite Installer on Microsoft Windows

If you have received Sourcery G++ Lite on CD, insert the CD in your computer. On most computers, the installer then starts automatically. If your computer has been configured not to automatically run CDs, open `My Computer`, and double click on the CD. If you downloaded Sourcery G++ Lite, double-click on the installer.

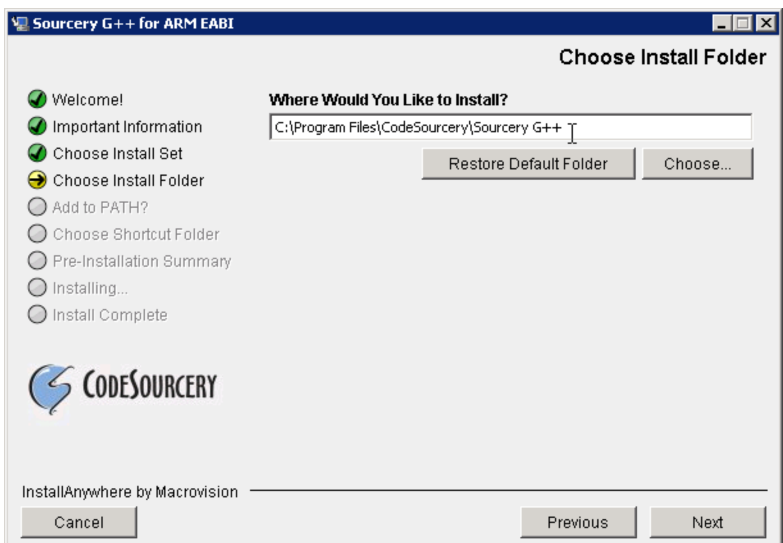
After the installer starts, follow the on-screen dialogs to install Sourcery G++ Lite. The installer is intended to be self-explanatory and on most pages the defaults are appropriate.

[1] http://www.codesourcery.com/gnu_toolchains/
[2] https://support.codesourcery.com/GNUToolchain/

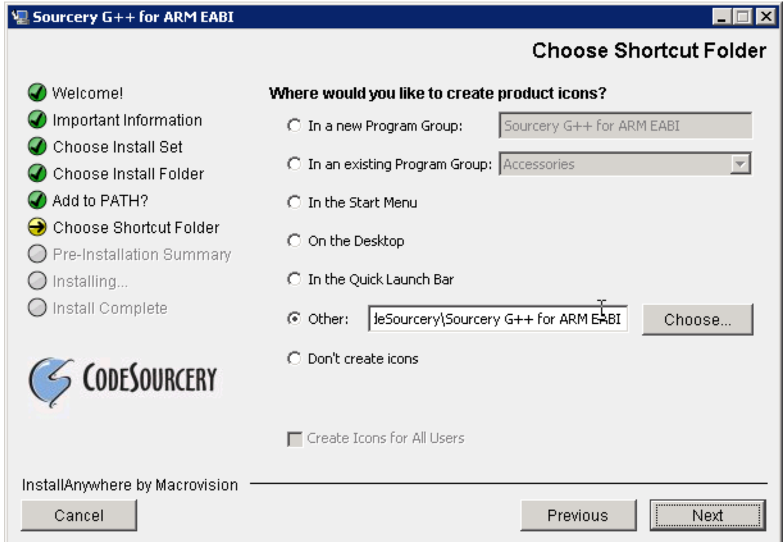


Running the Installer. The graphical installer guides you through the steps to install Sourcery G++ Lite.

You may want to change the install directory pathname and customize the shortcut installation.

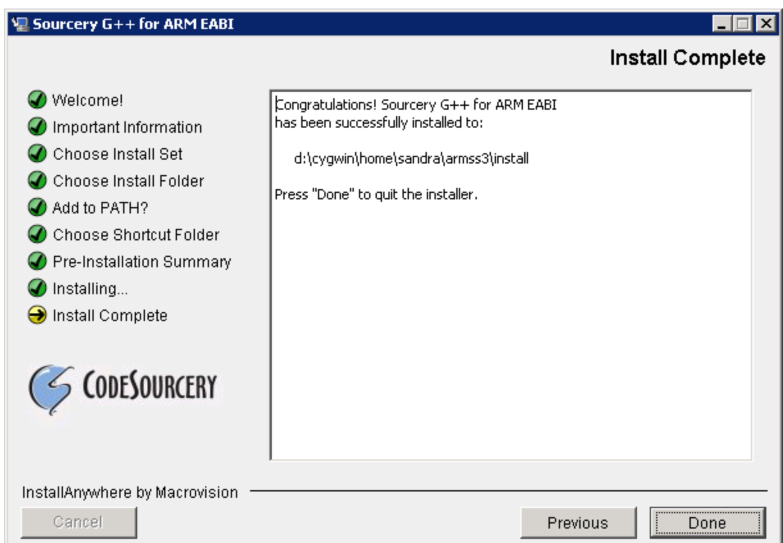


Choose Install Folder. Select the pathname to your install directory.



Choose Shortcut Folder. You can customize where the installer creates shortcuts for quick access to Sourcery G++ Lite.

When the installer has finished, it asks if you want to launch a viewer for the Getting Started guide. Finally, the installer displays a summary screen to confirm a successful install before it exits.



Install Complete. You should see a screen similar to this after a successful install.

If you prefer, you can run the installer in console mode rather than using the graphical interface. To do this, invoke the installer with the `-i console` command-line option. For example:

```
> /path/to/package.exe -i console
```

2.4.2. Using the Sourcery G++ Lite Installer on GNU/Linux Hosts

Start the graphical installer by invoking the executable shell script:

```
> /bin/sh ./path/to/package.bin
```

After the installer starts, follow the on-screen dialogs to install Sourcery G++ Lite. For additional details on running the installer, see the discussion and screen shots in the Microsoft Windows section above.

If you prefer, or if your host system does not run the X Window System, you can run the installer in console mode rather than using the graphical interface. To do this, invoke the installer with the `-i console` command-line option. For example:

```
> /bin/sh ./path/to/package.bin -i console
```

2.4.3. Installing Sourcery G++ Lite from a Compressed Archive

You do not need to be a system administrator to install Sourcery G++ Lite from a compressed archive. You may install Sourcery G++ Lite using any user account and in any directory to which you have write access. This guide assumes that you have decided to install Sourcery G++ Lite in the `$HOME/CodeSourcery` subdirectory of your home directory and that the filename of the package you have downloaded is `/path/to/package.tar.bz2`. After installation the toolchain will be in `$HOME/CodeSourcery/sourceryg++-4.4`.

First, uncompress the package file:

```
> bunzip2 /path/to/package.tar.bz2
```

Next, create the directory in which you wish to install the package:

```
> mkdir -p $HOME/CodeSourcery
```

Change to the installation directory:

```
> cd $HOME/CodeSourcery
```

Unpack the package:

```
> tar xf /path/to/package.tar
```

2.5. Installing Sourcery G++ Lite Updates

If you have already installed an earlier version of Sourcery G++ Lite for MIPS ELF on your system, it is not necessary to uninstall it before using the installer to unpack a new version in the same location. The installer detects that it is performing an update in that case.

If you are installing an update from a compressed archive, it is recommended that you remove any previous installation in the same location, or install in a different directory.

Note that the names of the Sourcery G++ commands for the MIPS ELF target all begin with **mips-sde-elf**. This means that you can install Sourcery G++ for multiple target systems in the same directory without conflicts.

2.6. Setting up the Environment

As with the installation process itself, the steps required to set up your environment depend on your host operating system.

2.6.1. Setting up the Environment on Microsoft Windows Hosts

2.6.1.1. Setting the PATH

In order to use the Sourcery G++ tools from the command line, you should add them to your PATH. You may skip this step if you used the graphical installer, since the installer automatically adds Sourcery G++ to your PATH.

To set the PATH on a Microsoft Windows Vista system, use the following command in a cmd.exe shell:

```
> setx PATH "%PATH%;C:\Program Files\Sourcery G++\bin"
```

where C:\Program Files\Sourcery G++ should be changed to the path of your Sourcery G++ Lite installation.

To set the PATH on a system running a Microsoft Windows version other than Vista, from the desktop bring up the Start menu and right click on My Computer. Select Properties, go to the Advanced tab, then click on the Environment Variables button. Select the PATH variable and click the Edit. Add the string ;C:\Program Files\Sourcery G++\bin to the end, and click OK. Again, you must adjust the pathname to reflect your installation directory.

You can verify that your PATH is set up correctly by starting a new cmd.exe shell and running:

```
> mips-sde-elf-g++ -v
```

Verify that the last line of the output contains: Sourcery G++ Lite 4.4-58.

2.6.1.2. Working with Cygwin

Sourcery G++ Lite does not require Cygwin or any other UNIX emulation environment. You can use Sourcery G++ directly from the Windows command shell. You can also use Sourcery G++ from within the Cygwin environment, if you prefer.

The Cygwin emulation environment translates Windows path names into UNIX path names. For example, the Cygwin path /home/user/hello.c corresponds to the Windows path c:\cygwin\home\user\hello.c. Because Sourcery G++ is not a Cygwin application, it does not, by default, recognize Cygwin paths.

If you are using Sourcery G++ from Cygwin, you should set the CYGPATH environment variable. If this environment variable is set, Sourcery G++ Lite automatically translates Cygwin path names into Windows path names. To set this environment variable, type the following command in a Cygwin shell:

```
> export CYGPATH=cygpath
```

To resolve Cygwin path names, Sourcery G++ relies on the **cygpath** utility provided with Cygwin. You must provide Sourcery G++ with the full path to cygpath if **cygpath** is not in your PATH. For example:

```
> export CYGPATH=c:/cygwin/bin/cygpath
```

directs Sourcery G++ Lite to use c:/cygwin/bin/cygpath as the path conversion utility. The value of CYGPATH must be an ordinary Windows path, not a Cygwin path.

2.6.2. Setting up the Environment on GNU/Linux Hosts

If you installed Sourcery G++ Lite using the graphical installer then you may skip this step. The installer does this setup for you.

Before using Sourcery G++ Lite you should add it to your `PATH`. The command you must use varies with the particular command shell that you are using. If you are using the C Shell (**csh** or **tcsh**), use the command:

```
> setenv PATH $HOME/CodeSourcery/Sourcery_G++/bin:$PATH
```

If you are using Bourne Shell (**sh**), the Korn Shell (**ksh**), or another shell, use:

```
> PATH=$HOME/CodeSourcery/Sourcery_G++/bin:$PATH
> export PATH
```

If you are not sure which shell you are using, try both commands. In both cases, if you have installed Sourcery G++ Lite in an alternate location, you must replace the directory above with `bin` subdirectory of the directory in which you installed Sourcery G++ Lite.

You may also wish to set the `MANPATH` environment variable so that you can access the Sourcery G++ manual pages, which provide additional information about using Sourcery G++. To set the `MANPATH` environment variable, follow the same steps shown above, replacing `PATH` with `MANPATH`, and `bin` with `share/doc/sourceryg++-mips-sde-elf/man`.

You can test that your `PATH` is set up correctly by running the following command:

```
> mips-sde-elf-g++ -v
```

Verify that the last line of the output contains: `Sourcery G++ Lite 4.4-58`.

2.7. Uninstalling Sourcery G++ Lite

The method used to uninstall Sourcery G++ Lite depends on the method you originally used to install it. If you have modified any files in the installation it is recommended that you back up these changes. The uninstall procedure may remove the files you have altered.

2.7.1. Using the Sourcery G++ Lite Uninstaller on Microsoft Windows

For Windows hosts other than Microsoft Windows Vista, select `Start`, then `Control Panel`. Select `Add or Remove Programs`. Scroll down and click on `Sourcery G++ for MIPS ELF`. Select `Change/Remove` and follow the on-screen dialogs to uninstall Sourcery G++ Lite.

On Microsoft Windows Vista hosts, select `Start`, then `Settings` and finally `Control Panel`. Select the `Uninstall a program` task. Scroll down and double click on `Sourcery G++ for MIPS ELF`. Follow the on-screen dialogs to uninstall Sourcery G++ Lite.

You can run the uninstaller in console mode, rather than using the graphical interface, by invoking the Uninstall executable found in your Sourcery G++ Lite installation directory with the `-i console` command-line option.

To uninstall third-party drivers bundled with Sourcery G++ Lite, first disconnect the associated hardware device. Then use `Add or Remove Programs` (non-Vista) or `Uninstall a program` (Vista) to remove the drivers separately. Depending on the device, you may need to reboot your computer to complete the driver uninstall.

2.7.2. Using the Sourcery G++ Lite Uninstaller on GNU/Linux

You should use the provided uninstaller to remove a Sourcery G++ Lite installation originally created by the executable installer script. The `mips-sde-elf` directory located in the install directory will be removed entirely by the uninstaller. Please back up any changes you have made to this directory, such as modified linker scripts.

Start the graphical uninstaller by invoking the executable Uninstall shell script located in your installation directory. After the uninstaller starts, follow the on-screen dialogs to uninstall Sourcery G++ Lite.

You can run the uninstaller in console mode, rather than using the graphical interface, by invoking the Uninstall script with the `-i console` command-line option.

2.7.3. Uninstalling a Compressed Archive Installation

If you installed Sourcery G++ Lite from a `.tar.bz2` file, you can uninstall it by manually deleting the installation directory created in the install procedure.

Chapter 3
Sourcery G++ Lite for MIPS ELF

This chapter contains information about features of Sourcery G++ Lite that are specific to MIPS ELF targets. You should read this chapter to learn how to best use Sourcery G++ Lite on your target system.

3.1. Included Components and Features

This section briefly lists the important components and features included in Sourcery G++ Lite for MIPS ELF, and tells you where you may find further information about these features.

Component	Version	Notes
GNU programming tools		
GNU Compiler Collection	4.4.1	Separate manual included.
GNU Binary Utilities	2.19.51	Includes assembler, linker, and other utilities. Separate manuals included.
Debugging support and simulators		
GNU Debugger	6.8.50	Separate manual included.
GDB Simulator	N/A	See Section 4.3, “Running Applications in the Simulator”.
MDI Debugging Support	N/A	Included with GDB. See Section 4.4.3, “Connecting with MDI”.
Target libraries		
CodeSourcery Common Startup Code Sequence	4.4-58	See Chapter 5, “CS3™: The CodeSourcery Common Startup Code Sequence”.
Newlib C Library	1.17.0	Separate manuals included.
Other utilities		
GNU Make	N/A	Build support on Windows hosts.
GNU Core Utilities	N/A	Build support on Windows hosts.

3.2. Library Configurations

Sourcery G++ includes copies of run-time libraries that have been built with optimizations for different target architecture variants or other sets of build options. Each such set of libraries is referred to as a *multilib*. When you link a target application, Sourcery G++ selects the multilib matching the build options you have selected.

Sourcery G++ Lite includes linker scripts as well as runtime libraries for each multilib. You can find these files in multilib-specific subdirectories of the `mips-sde-elf/lib` directory of your Sourcery G++ install.

3.2.1. Included Libraries

The following library configurations are available in Sourcery G++ Lite for MIPS ELF.

MIPS32 revision 2 - Big-Endian, O32	
Command-line option(s):	default
Library subdirectory:	`./`

MIPS32 revision 2 - Little-Endian, O32	
Command-line option(s):	`-EL`
Library subdirectory:	`el/`

MIPS32 revision 2 - Big-Endian, O32, mips16	
Command-line option(s):	`-mips16`
Library subdirectory:	`mips16/`

MIPS32 revision 2 - Big-Endian, O32, fp64	
Command-line option(s):	`-mfp64`
Library subdirectory:	`fp64/`

MIPS32 revision 2 - Soft-Float, O32	
Command-line option(s):	`-msoft-float`
Library subdirectory:	`sof/`

MIPS32 revision 2 - No-Float, O32	
Command-line option(s):	`-mno-float`
Library subdirectory:	`nof/`

MIPS32 revision 2 - Big-Endian, O32, mips16, fp64	
Command-line option(s):	`-mips16 -mfp64`
Library subdirectory:	`mips16/fp64/`

MIPS32 revision 2 - Big-Endian, O32, mips16, Soft-Float	
Command-line option(s):	`-mips16 -msoft-float`
Library subdirectory:	`mips16/sof/`

MIPS32 revision 2 - Big-Endian, O32, mips16, No-Float	
Command-line option(s):	`-mips16 -mno-float`
Library subdirectory:	`mips16/nof/`

MIPS32 revision 2 - Big-Endian, O32, mips16, code-readable=no	
Command-line option(s):	`-mips16 -mcode-readable=no`
Library subdirectory:	`mips16/spram/`

MIPS32 revision 2 - Big-Endian, O32, mips16, fp64, code-readable=no	
Command-line option(s):	`-mips16 -mfp64 -mcode-readable=no`
Library subdirectory:	`mips16/fp64/spram/`

MIPS32 revision 2 - Big-Endian, O32, mips16, Soft-Float, code-readable=no	
Command-line option(s):	`-mips16 -msoft-float -mcode-readable=no`
Library subdirectory:	`mips16/sof/spram/`

MIPS32 revision 2 - Big-Endian, O32, mips16, No-Float, code-readable=no	
Command-line option(s):	`-mips16 -mno-float -mcode-readable=no`
Library subdirectory:	`mips16/nof/spram/`

MIPS32 revision 2 - Little-Endian, O32, mips16	
Command-line option(s):	`-EL -mips16`
Library subdirectory:	`el/mips16/`

MIPS32 revision 2 - Little-Endian, O32, fp64	
Command-line option(s):	`-EL -mfp64`
Library subdirectory:	`el/fp64/`

MIPS32 revision 2 - Little-Endian, O32, Soft-Float	
Command-line option(s):	`-EL -msoft-float`
Library subdirectory:	`el/sof/`

MIPS32 revision 2 - Little-Endian, O32, No-Float	
Command-line option(s):	`-EL -mno-float`
Library subdirectory:	`el/nof/`

MIPS32 revision 2 - Little-Endian, O32, mips16, fp64	
Command-line option(s):	`-EL -mips16 -mfp64`
Library subdirectory:	`el/mips16/fp64/`

MIPS32 revision 2 - Little-Endian, O32, mips16, Soft-Float	
Command-line option(s):	`-EL -mips16 -msoft-float`
Library subdirectory:	`el/mips16/sof/`

MIPS32 revision 2 - Little-Endian, O32, mips16, No-Float	
Command-line option(s):	`-EL -mips16 -mno-float`
Library subdirectory:	`el/mips16/nof/`

MIPS32 revision 2 - Little-Endian, O32, mips16, code-readable=no	
Command-line option(s):	`-EL -mips16 -mcode-readable=no`
Library subdirectory:	`el/mips16/spram/`

MIPS32 revision 2 - Little-Endian, O32, mips16, fp64, code-readable=no	
Command-line option(s):	`-EL -mips16 -mfp64 -mcode-readable=no`
Library subdirectory:	`el/mips16/fp64/spram/`

MIPS32 revision 2 - Little-Endian, O32, mips16, Soft-Float, code-readable=no	
Command-line option(s):	`-EL -mips16 -msoft-float -mcode-readable=no`
Library subdirectory:	`el/mips16/sof/spram/`

MIPS32 revision 2 - Little-Endian, O32, mips16, No-Float, code-readable=no	
Command-line option(s):	`-EL -mips16 -mno-float -mcode-readable=no`
Library subdirectory:	`el/mips16/nof/spram/`

3.2.2. Library Selection

A given multilib may be compatible with additional processors and build options beyond those listed above. However, even if a particular set of command-line options produces code compatible with one of the provided multilibs, those options may not be sufficient to identify the intended library to the linker. For example, on some targets, specifying only a processor option on the command line may imply architecture features or floating-point support for compilation, but not for library selection. The details of the mapping from command-line options to multilibs are target-specific and quite complex. Therefore, it is recommended that your link command line include exactly the options listed in the tables above for your intended target multilib. In some cases, you may need to supply different options for linking than for compilation.

If you are uncertain which multilib is selected by a particular set of command-line options, GCC can tell you if you invoke it with the `-print-multi-directory` option in addition to your other build options. For example:

```
> mips-sde-elf-gcc -print-multi-directory options...
```

The output of this command is a directory name for the multilib, which you can look up in the tables given previously.

3.3. CS3 Support

Sourcery G++ Lite includes CS3 linker scripts and initialization code to support three different classes of target configurations:

- Simulator targets, such as MIPSsim, running under control of the debugger.
- Malta hardware targets running in a bare-metal configuration under control of the debugger.
- Malta hardware targets running under control of the YAMON boot monitor.

You must use the appropriate linker script to match your target, since the memory layouts and startup code sequences are different in each case. Refer to Chapter 5, “CS3™: The CodeSourcery Common Startup Code Sequence” for details on the supported boards for this version of Sourcery G++ Lite.

For simulator and bare-metal targets, CS3 provides semihosted I/O via the debugger console on the host. For instructions on loading and running code on the target from command-line GDB, see Section 4.4, “Running Applications from GDB”.

3.4. Using Sourcery G++ with MIPS Malta Boards

The provided CS3 linker scripts for MIPS Malta boards (both bare-metal and YAMON profiles) assume 128MB of RAM is available on the target. If your target board has less memory, you must adjust the memory layout used by the linker by specifying a custom linker script.

Find the linker script for your selected profile, such as `mips-sde-elf/lib/malta-ram-hosted.ld`, in your Sourcery G++ Lite installation and copy it to your project working directory. In your local copy, find the `MEMORY` directive and edit the `LENGTH` expression to match the amount of memory available on your board. Then, use the full absolute pathname of your modified linker script with the `-T` command-line option when linking your program.

3.5. Using Sourcery G++ with YAMON

For YAMON targets, CS3 provides basic I/O services via the YAMON console. This section briefly covers how to load and run programs using YAMON.

To prepare an application to run from YAMON, you must first convert the executable file to SREC format. You can do this from the command line on your host system using the objcopy utility provided with Sourcery G++ Lite.

```
> mips-sde-elf-objcopy -O srec prog prog.srec
```

Next, use YAMON to load the SREC image file into RAM. For example, to load via TFTP, use a command similar to:

```
YAMON> load tftp://host/path/prog.srec
```

Then, start the program from the YAMON prompt:

```
YAMON> go .
```

For more detailed information about YAMON usage and features, refer to the *YAMON User's Manual*.

3.6. Profiling Support

Sourcery G++ Lite includes CS3 support for code profiling on MIPS ELF targets using **gprof**. To enable profiling, compile your program with the `-pg` option. You must also build your program with a hosted linker script.

You can run a program built with profiling from the debugger the same as you would any other hosted application. While your program is running, profiling data is saved in buffers in the heap memory area on the target. This may affect the amount of memory available to your application, and it is also possible that the profiler itself may run out of memory. Profiling data is written to a file on the host (`gmon.out`) only when your application exits. Since many embedded applications are structured to run indefinitely rather than exit, you may need to add an explicit `exit` call in order to collect profiling data.

For instructions on using the **mips-sde-elf-gprof** utility to process the collected `gmon.out` data, refer to the GNU Profiler (gprof) manual included with Sourcery G++ Lite.

3.7. Using Flash Memory

Sourcery G++ Lite supports development and debugging of applications loaded into flash memory on MIPS ELF targets. There are three steps involved:

1. You must use an appropriate linker script that identifies the ROM memory region on your target board, and locates the program text within that region. Refer to Chapter 5, “CS3™: The CodeSourcery Common Startup Code Sequence” for information about the boards supported by Sourcery G++.

2. Next, load your program into the flash memory on your target board. You must use third-party tools to program the flash memory.

3. Finally, when debugging a program in flash memory, GDB must be told about the ROM region so that it knows where it must use hardware breakpoints to control program execution.

   When using GDB from the command line, you can mark the flash memory as read-only by using the command:

   ```
   (gdb) mem start end ro
   ```

   where *`start`* and *`end`* define the address range of the read-only memory region.

Although GDB automatically attempts to use hardware breakpoints on code locations in the read-only memory region, on many targets the number of available hardware breakpoints is very small. Furthermore, GDB also uses hardware breakpoints internally to implement commands such as **step**, **next**, and **finish**. Thus the number of breakpoints you can explicitly set in ROM may be fewer than the number supported by the target system.

3.8. Additional Documentation

A document that provides additional details on using Sourcery G++ Lite for MIPS ELF is provided. The document can be found at `share/doc/sourceryg++-mips-sde-elf/pdf/MIPS_TOOLCHAIN.pdf` within your installation directory.

Chapter 4
Using Sourcery G++ from the Command Line

This chapter demonstrates the use of Sourcery G++ Lite from the command line.

4.1. Building an Application

This chapter explains how to build an application with Sourcery G++ Lite using the command line. As elsewhere in this manual, this section assumes that your target system is mips-sde-elf, as indicated by the **mips-sde-elf** command prefix.

Using an editor (such as **notepad** on Microsoft Windows or **vi** on UNIX-like systems), create a file named `main.c` containing the following simple factorial program:

```
#include <stdio.h>

int factorial(int n) {
  if (n == 0)
    return 1;
  return n * factorial (n - 1);
}

int main () {
  int i;
  int n;
  for (i = 0; i < 10; ++i) {
    n = factorial (i);
    printf ("factorial(%d) = %d\n", i, n);
  }
  return 0;
}
```

Compile and link this program using the command:

```
> mips-sde-elf-gcc -o factorial main.c -T script
```

Sourcery G++ requires that you specify a linker script with the `-T` option to build applications for bare-board targets. Linker errors like `undefined reference to `read'` are a symptom of failing to use an appropriate linker script. Default linker scripts are provided in `mips-sde-elf/lib`. Refer to Chapter 5, “CS3™: The CodeSourcery Common Startup Code Sequence” for information about the boards and linker scripts supported by Sourcery G++ Lite.

There should be no output from the compiler. (If you are building a C++ application, instead of a C application, replace **mips-sde-elf-gcc** with **mips-sde-elf-g++**.)

4.2. Running Applications on the Target System

Consult your target board documentation for instructions on loading programs onto the target, and running them.

4.3. Running Applications in the Simulator

Sourcery G++ Lite includes a simulator that you can use on the host system to run programs compiled for the target system. Since you do not need target hardware, this is the easiest way to try out Sourcery G++.

To use the simulator run:

```
> mips-sde-elf-run factorial
```

You should see the expected output:

```
factorial(0) = 1
factorial(1) = 1
factorial(2) = 2
factorial(3) = 6
factorial(4) = 24
factorial(5) = 120
factorial(6) = 720
factorial(7) = 5040
factorial(8) = 40320
factorial(9) = 362880
```

You can also use the simulator to execute target programs when debugging with GDB. See Section 4.4, “Running Applications from GDB” for more information.

The simulator supports the MIPS32r2 instruction set, including the MIPS16e, MIPS DSP and DSP Revision 2, SmartMIPS, and MIPS-3D ASEs. It can also emulate earlier variants of the MIPS architecture.

4.4. Running Applications from GDB

You can run GDB, the GNU Debugger, on your host system to debug programs running remotely on a target board or system. You can also run and debug programs using the GDB simulator.

While this section explains the alternatives for using GDB to run and debug application programs, explaining the use of the GDB command-line interface is beyond the scope of this document. Please refer to the GDB manual for further instructions.

4.4.1. Connecting to the GDB Simulator

GDB includes a simulator that allows you to debug MIPS ELF applications without target hardware. To start and connect to the simulator from within GDB, use this command:

```
(gdb) target sim
```

4.4.2. Connecting to an External GDB Server

From within GDB, you can connect to a running **gdbserver** or other debugging stub that uses the GDB remote protocol using:

```
(gdb) target remote host:port
```

where *`host`* is the host name or IP address of the machine the stub is running on, and *`port`* is the port number it is listening on for TCP connections.

4.4.3. Connecting with MDI

Sourcery G++ Lite for MIPS ELF supports debugging with third-party simulators and hardware debug devices that implement the MDI (Microprocessor Debug Interface) API.

Before you can connect to a target using the MDI API, you must tell GDB which shared library or DLL to load for your simulator or device, and set up parameters to select your MDI target. This can be done either by means of environment variables or GDB commands. For example, you may want to put the GDB configuration commands in your `.gdbinit` file, which is loaded automatically when you start GDB.

This section describes the basic MDI usage; refer to the documentation for your MDI simulator or debug device for details specific to that target. Note, in particular, that some MDI targets may require you to set up a configuration file and/or license in addition to the steps given here.

In order to tell GDB which MDI library to load, on Linux hosts you should add the directory containing the shared library files to your `LD_LIBRARY_PATH` environment variable. On Windows hosts, add the directory containing the DLLs to your `PATH` environment variable. Then, either set the environment variable `GDBMDILIB` to the base name of the MDI library before starting GDB, or select the library within GDB using the command:

```
(gdb) set mdi library name
```

To verify that your shared library configuration is correct, you can query it from GDB for the devices it supports:

```
(gdb) show mdi devices
```

Each device is identified by a target number and device number. You can select the values you want to use with the following GDB commands:

```
(gdb) set mdi target targetnum
(gdb) set mdi device devicenum
```

Alternatively, you can set the environment variables `GDBMDITARGET` and `GDBMDIDEVICE`.

At this point, you can establish a connection to the selected MDI device using:

```
(gdb) target mdi
```

Then you can load and run or attach to your program on the target.

4.4.4. Loading and Running Applications

Connecting to a bare-metal target or simulator from GDB does not cause your program to be loaded into target memory. You must do this explicitly from GDB after you connect:

```
(gdb) load
```

Alternatively, you can use third-party tools to load your application into flash memory before starting GDB.

To begin execution of your application, you should generally use the **continue** command:

```
(gdb) continue
```

However, you should use **run** instead of **continue** to start your program if you used **target mdi** or **target sim** to connect:

```
(gdb) run
```

Chapter 5
CS3™: The CodeSourcery Common Startup Code Sequence

CS3 is CodeSourcery's low-level board support library. This chapter describes the organization of the system startup code and tells you how you can customize it, such as by defining your own interrupt handlers. This chapter also documents the boards supported by Sourcery G++ Lite and the compiler and linker options you need to use with them.

Many developers turn to the GNU toolchain for its cross-platform consistency: having a single system support so many different processors and boards helps to limit risk and keep learning curves gentle. Historically, however, the GNU toolchain has lacked a consistent set of conventions for processor- and board-level initialization, language run-time setup, and interrupt and trap handler definition.

The CodeSourcery Common Startup Code Sequence (CS3) addresses this problem. For each supported system, CS3 provides a set of linker scripts describing the system's memory map, and a board support library providing generic reset, startup, and interrupt handlers. These scripts and libraries all follow a standard set of conventions across a range of processors and boards.

5.1. Startup Sequence

CS3 divides the startup sequence into three phases:

- In the *hard reset phase*, we initialize the memory controller and set up the memory map.
- In the *assembly initialization phase*, we prepare the stack to run C code, and jump to the C initialization function.
- In the *C initialization phase*, we initialize the data areas, run constructors for statically-allocated objects, and call `main`.

The hard reset and assembly initialization phases are necessarily written in assembly language; at reset, there may not yet be stack to hold compiler temporaries, or perhaps even any RAM accessible to hold the stack. These phases do the minimum necessary to prepare the environment for running simple C code. Then, the code for the final phase may be written in C; CS3 leaves as much as possible to be done at this point.

The CodeSourcery board support library provides default code for all three phases. The hard reset phase is implemented by board-specific code. The assembly initialization phase is implemented by code specific to the processor family. The C initialization phase is implemented by generic code.

5.1.1. The Hard Reset Phase

This phase is responsible for initializing board-specific registers, such as memory base registers and DRAM controllers, or scanning memory to check the available size. It is written in assembler and ends with a jump to `_start`, which is where the assembly initialization phase begins.

The hard reset code is in a section named `.cs3.reset`. The section must define a symbol named `__cs3_reset_`*`sys`*, where *`sys`* is a name for the board being initialized; for example, the reset code for M5208EVB boards would be named `__cs3_reset_m5208evb`. The linker script defines the symbol `__cs3_reset` to refer to this reset address. If you need to refer to the reset address from generic code, you can use this non-specific `__cs3_reset` name.

Some simulators provide a supervisory operation to determine the amount of available memory. This operation is performed in the hard reset phase. Thus for simulators, execution always begins at `__cs3_reset_`*`sys`*.

The CodeSourcery board support library provides reasonable default reset code, but you may provide your own reset code by defining `__cs3_reset_`*`sys`* in an object file or library, in a `.cs3.reset` section.

5.1.2. The Assembly Initialization Phase

This phase is responsible for initializing the stack pointer and creating an initial stack frame. The symbol `_start` marks the entry point of the assembly initialization code; this name lacks the `__cs3` prefix because is the symbol traditionally used by debuggers and other integrated development environments for the address where program execution begins. The assembly initialization phase ends with a call or jump to `__cs3_start_c`.

Simulators typically initialize the stack pointer and initial stack frame automatically on startup. CS3 can also support targets running a boot monitor that likewise initializes the stack before starting user code. On these targets, CS3 does not perform the assembly initialization phase at all; instead, `_start` is aliased to `__cs3_reset_`*`sys`*, so that execution always starts with the hard reset phase. The hard reset phase then ends with a jump directly to `__cs3_start_c`.

On the other hand, on bare-board targets setting the stack pointer explicitly in the assembly initialization phase is required even if the processor itself initializes the stack pointer automatically on reset. This is to support restarting programs from `_start` in the debugger.

The value of the symbol `__cs3_stack` provides the initial value of the stack pointer. The CodeSourcery linker scripts provide a default value for this symbol, which you may override by defining `__cs3_stack` yourself.

The initial stack frame is created for the use of ordinary C and C++ calling conventions. The stack should be initialized so that backtraces stop cleanly at this point; this might entail zeroing a dynamic link pointer, or providing hand-written DWARF call frame information.

Finally, we call the C function `__cs3_start_c`. This function never returns, and `_start` need not be prepared to handle a return from it.

As with the hard reset code, the CodeSourcery board support library provides reasonable default assembly initialization code. However, you may provide your own code by providing a definition for `_start`, either in an object file or a library.

5.1.3. The C Initialization Phase

Finally, C code can be executed. The C startup function is declared as follows:

```
void __cs3_start_c (void) __attribute__ ((noreturn));
```

In this function we take the following steps:

- Initialize all `.data`-like sections by copying their contents.
- Clear all `.bss`-like sections.
- Run constructors for statically-allocated objects, recorded using whatever conventions are usual for C++ on the target architecture.

  CS3 reserves priorities from 0 to 100 for use by initialization code. You can handle tasks like enabling interrupts, initializing coprocessors, pointing control registers at interrupt vectors, and so on by defining constructors with appropriate priorities.
- Call `main` as appropriate.
- Call `exit`, if it is available.

As with the hard reset and assembly initialization code, the CodeSourcery board support library provides a reasonable definition for the `__cs3_start_c` function. You may override this by providing a definition for `__cs3_start_c`, either in an object file or in a library.

The CodeSourcery-provided definition of `__cs3_start_c` can pass command-line arguments to `main` using the normal C `argc` and `argv` mechanism if the board support package provides corresponding definitions for `__cs3_argc` and `__cs3_argv`. For example:

```
int __cs3_argc;
char **__cs3_argv;
```

These variables should be initialized using a constructor function, which is run by `__cs3_start_c` after it initializes the data segment. Use the `constructor` attribute on the function definition:

```
__attribute__((constructor))
static void __cs3_init_args (void) {
   __cs3_argc = ...;
   __cs3_argv = ...;
}
```

The constructor function may have an arbitrary name; `__cs3_init_args` is used only for illustrative purposes here.

If definitions of `__cs3_argc` and `__cs3_argv` are not provided, then the default `__cs3_start_c` function invokes `main` with zero as the `argc` argument and a null pointer as `argv`.

5.2. Exit and Embedded Systems

A program running on an embedded system is usually designed never to exit — it runs until the system is powered down. The C and C++ standards leave it unspecified as to whether `exit` is called at program termination. If the program never exits, then there is no reason to include `exit`, facilities to run functions registered with `atexit`, or global destructors. This code would never be run and would therefore just waste space in the application.

The CS3 startup code, by itself, does not cause `exit` to be present in the application. It dynamically checks whether `exit` is present, and only calls it if it is. If you require `exit` to be present, either refer to it within your application, or add `-Wl,-u,exit` to the linking command line.

Similarly, code to register global destructors is only invoked when `atexit` is already in the executable; CS3, by itself, does not cause `atexit` to be present. If you require `atexit`, either refer to it within your application, or add `-Wl,-u,atexit` to the linking command line.

5.3. Memory Layout

The header file `cs3.h` declares variables and types that describe the layout of memory on the system to C code. The variables are defined by the CS3 linker script or in the board support library.

The following variables describe the regions of memory to be initialized at startup:

```
/* The number of elements in __cs3_regions.  */
const size_t __cs3_region_num;

/* An untyped object, aligned on an eight-byte boundary.  */
typedef unsigned char __cs3_byte_align8
```

```
                      __attribute__ ((aligned (8)));

struct __cs3_region
{
  /* Flags for this region.  None defined yet.  */
  unsigned flags;

  __cs3_byte_align8 *init;  /* Region's initial contents.  */
  __cs3_byte_align8 *data;  /* Region's start address.  */

  /* These sizes are always a multiple of eight.  */
  size_t init_size;     /* Size of initial data.  */
  size_t zero_size;     /* Additional size to be zeroed.  */
};

/* An array of memory regions.  __cs3_regions[0] describes
   the region holding .data and .bss.  */
extern const struct __cs3_region __cs3_regions[];
```

The following variables describe the area of memory to be used for the dynamically-allocated heap:

```
/* The addresses of these objects are the start and end of
   free space for the heap, typically following .data and .bss.
   However, &__cs3_heap_end may be zero, meaning that we must
   determine the heap limit in some other way --- perhaps via a
   supervisory operation on a simulator, or simply by treating
   the stack pointer as the limit.  */
extern __cs3_byte_align8 __cs3_heap_start[];
extern __cs3_byte_align8 __cs3_heap_end[];

/* The end of free space for the heap, or zero if we haven't been
   able to determine it.  It usually points to __cs3_heap_end,
   but in some configurations, may be overridden by a supervisory
   call in the reset code.  */
extern void *__cs3_heap_limit;
```

For each region named *R*, `cs3.h` declares the following variables:

```
/* The start of the region.  */
extern unsigned char __cs3_region_start_R[]
                     __attribute__ ((aligned (8)));

/* The region's size, in bytes.  */
extern const size_t __cs3_region_size_R;
```

At the assembly level, the linker script also defines symbols with the same names and values.

If the region is initialized, then `cs3.h` also declares the following variables, corresponding to the region's element in `__cs3_regions`:

```
/* The data we initialize the region with.  */
extern const unsigned char __cs3_region_init_R[]
                           __attribute__ ((aligned (8)));

/* The size of the initialized portion of the region.  */
```

```
extern const size_t __cs3_region_init_size_R;

/* The size of the additional portion to be zeroed.  */
extern const size_t __cs3_region_zero_size_R;
```

Any of these identifiers may actually be defined as preprocessor macros that expand to expressions of the appropriate type and value.

Like the `struct __cs3_region` members, these regions are all aligned on eight-byte boundaries, and their sizes are multiples of eight bytes.

CS3 linker scripts place the contents of sections named `.cs3.region-head.R` at the start of each memory region. Note that CS3 itself may want to place items (like interrupt vector tables) at these locations; if there is a conflict, CS3 raises an error at link time.

5.4. Interrupt Vectors and Handlers

CS3 provides standard handlers for interrupts, exceptions and traps, but also allows you to easily define your own handlers as needed. In this section, we use the term *interrupt* as a generic term for this entire class of events.

Different processors handle interrupts in various ways, but there are two general approaches:

- Some processors fetch an address from an array indexed by the interrupt number, and jump to that address. We call these *address vector* processors; ColdFire systems are a typical example.
- Others multiply the interrupt number by some constant factor, add a base address, and jump directly to that address. Here, the interrupt vector consists of blocks of code, so we call these *code vector* processors; PowerPC systems are a typical example.

On address vector processors, the CS3 library provides an array of pointers to interrupt handlers named `__cs3_interrupt_vector_form`, occupying a section named `.cs3.interrupt_vector`, where *`form`* identifies the particular processor variant the vector is appropriate for. If the processor supports more than one variety of interrupt vector (for example, a full-length form and a shortened form), then *`form`* identifies the variety as well. Each entry in the vector holds a reference to a symbol named `__cs3_isr_int`, where *`int`* is the customary name of that interrupt on the processor, or a number if there is no consistently used name. The library further provides a reasonable default definition for each `__cs3_isr_int` handler routine.

To override an individual handler, provide your own definition for the appropriate `__cs3_isr_int` symbol. The definition need not be placed in any particular object file section.

Interrupt handlers typically require special call/return and register usage conventions that are target-specific and beyond the scope of this document. As an alternative to writing interrupt handlers in assembly language, on some targets they may be written in C using the `interrupt` attribute. For example, to override the `__cs3_isr_access_error` handler, use the following definition:

```
void __attribute__ ((interrupt)) __cs3_isr_access_error (void)
{
  ... custom handler code ...
}
```

To override the entire interrupt vector, you can define `__cs3_interrupt_vector_form`, placing the definition in a section named `.cs3.interrupt_vector`. The linker script reports an error if the `.cs3.interrupt_vector` section is empty, to ensure that the definition of `__cs3_interrupt_vector_`*`form`* occupies the proper section.

You may define the vector in C with an array of pointers using the `section` attribute to place it in the appropriate section. For example, to override the interrupt vector on M5208EVB boards, make the following definition:

```
typedef void handler(void);
handler *__attribute__((section (".cs3.interrupt_vector")))
  __cs3_interrupt_vector_coldfire[] =
{ ... };
```

On code vector processors, we follow the same conventions, with the following exceptions:

- In addition to being named `__cs3_isr_`*`int`*, each interrupt handler must also occupy a section named `.cs3.interrupt_`*`int`*. Naturally, each handler must fit within a single interrupt vector entry.
- Instead of providing a default definition for `__cs3_interrupt_vector_`*`form`* in the library, the linker script gathers the `.cs3.interrupt_`*`int`* sections together, in the proper order and on the necessary address boundaries, and defines the `__cs3_interrupt_vector_`*`form`* symbol to refer to its start.

To override an individual handler on a code vector processor, you provide your own definition for `__cs3_isr_`*`int`*, placed in an appropriate section. The linker script ensures that each `.cs3.interrupt_`*`int`* section is non-empty, so that placing a handler in the wrong section elicits an error at link time.

CS3 does not allow you to override the entire interrupt vector on code vector processors, because the code vector must be constructed by the linker script, and thus cannot come from a library or object file. However, the portion of the linker script that constructs the interrupt vector occupies its own file, which other linker scripts can incorporate using the **INCLUDE** linker script command, making it easier to replace the linker script entirely and still take advantage of CS3's other features.

Some processors, like the Innovasic fido, use more than one interrupt vector: the processor provides several interrupt vector pointer registers, each used in different circumstances. Each register may point to a different vector, or some or all may share vectors.

On these processors, CS3 provides only a single pre-constructed interrupt vector, but defines a separate symbol for each interrupt vector pointer register; all the symbols point to the pre-constructed vector by default. The CS3 startup code initializes each register from the corresponding symbol. You can provide your own vectors by defining the appropriate symbols.

For example, the fido processor has five contexts, each of which can use its own interrupt vector; on this architecture, CS3 defines the standard `__cs3_interrupt_vector_fido` symbol referring to the pre-constructed vector, and then goes on to define per-context symbols `__cs3_interrupt_vector_fido_ctx0`, `__cs3_interrupt_vector_fido_ctx1`, and so on, all referring to `__cs3_interrupt_vector_fido`. The CS3 startup code sets each context's vector register to the value of the corresponding symbol. By default, all the contexts share an interrupt vector, but if your code provides its own definition for `__cs3_interrupt_vector_fido_ctx1`, then the startup code initializes context one's register to point to that vector instead.

This arrangement requires you to use a different approach to specify a handler for a secondary context that differs from the corresponding handler in the primary context. For example, to handle division-by-zero exceptions in context 1 with the function `ctx1_divide_by_zero`, you should write the following:

```
typedef void (*handler_type) (void);
handler_type __cs3_interrupt_vector_fido_ctx1[256];
extern handler_type __cs3_interrupt_vector_fido[256];

__attribute__((interrupt))
void
ctx1_divide_by_zero (void)
{
  /* Your code here.  */
}

__attribute__((constructor))
void
initialize_vector_ctx1 (void)
{
  /* Initialize our custom vector from the
     pre-constructed CS3 vector.  */
  memcpy (__cs3_interrupt_vector_fido_ctx1,
          __cs3_interrupt_vector_fido,
          sizeof (__cs3_interrupt_vector_fido));

  /* Initialize custom interrupt handlers.  */
  __cs3_interrupt_vector_fido_ctx1[5] = ctx1_divide_by_zero;
}
```

With this code in place, when a division-by-zero exception occurs in context 1, the processor calls `ctx1_divide_by_zero` to handle it. Defining `initialize_vector_ctx1` with the `constructor` attribute arranges for CS3 to call it before calling your program's `main` function.

5.5. Linker Scripts

CS3 provides linker scripts for each supported board. Each board may be used in a number of different configurations, and these are reflected in the linker script names. The linker scripts are named *`board-profile-hosted`*`.ld`, where *`board`* is the name of the board, *`profile`* describes the memory arrangement used and *`-hosted`* indicates whether hosting or semihosting is provided.

> **Caution**
>
> Linker scripts are required to create executable programs for MIPS ELF targets. When invoking the Sourcery G++ linker from the command line, you must explicitly supply a linker script using the `-T` option; otherwise a link error results.

5.5.1. Program and Data Placement

Many boards have both RAM and ROM (flash) memory devices. CS3 provides distinct linker scripts to place the application either entirely in RAM, or in ROM where data is initialized during the C initialization phase.

Some boards have very small amounts of RAM memory. If you use large library functions (such as `printf` and `malloc`), you may overflow the available memory. You may need to use the ROM-based linker scripts for such programs, so that the program itself is stored in ROM. You may be able to reduce the total amount of memory used by your program by replacing portions of the Sourcery G++ runtime library and/or startup code.

5.5.2. Hosting and Semihosting

CS3 is designed to support boards without an operating system. To allow functions like `open` and `write` to work without operating system support, a *semihosting* feature is supported, in conjunction with the debugger.

With semihosting enabled, these system calls are translated into equivalent function calls on your host system. You can only use these function calls while connected to the debugger; if you try to use them when disconnected from the debugger, you will get a hardware exception.

Semihosting requires support from the remote GDB debugging stub or agent, as well as the debugger itself. Semihosting is supported by the GDB Simulator included with Sourcery G++ Lite. You can additionally use semihosting when connecting to the target using MDI. However, semihosting may not be supported by debugging stubs provided by third parties. If you are using a debug device that communicates with GDB using the GDB remote protocol, check the documentation for your device to see whether semihosting is supported.

A good use of semihosting is to display debugging messages. For example, this program prints a message on the standard error stream on the host:

```
#include <unistd.h>

int main () {
  write (STDERR_FILENO, "Hello, world!\n", 14);
  return 0;
}
```

The hosted CS3 linker scripts provide the semihosting support, and as such programs linked with them may only be run with the debugger. The unhosted CS3 linker scripts provide stub versions of the system calls, which return an appropriate error value in `errno`. If such a stub system call is required in the executable, the linker also produces a warning. Such a warning may indicate that you have left debugging code active, and that your executable is larger than it might need to be.

Some targets supported by CS3 can run a boot monitor that provides console I/O services and other basic system calls. CS3 can also provide hosting via these facilities; where a boot monitor is supported, this is noted in the board tables below. Unlike semihosting, hosting via the boot monitor can be used when running programs outside of the debugger.

5.5.3. Choosing a Linker Script

When using Sourcery G++ from the command line, you must add `-T` *`script`* to your linking command, where *`script`* is the appropriate linker script. For example, to target M5208EVB boards, you could link with `-T m5208evb-ram`.

5.6. Supported Boards for MIPS ELF

CS3 provides support for the following boards on MIPS ELF targets.

MIPS Malta		
Processor name:	unspecified	
Processor options:	none	
Memory regions:	`ram`	
Linker scripts:	RAM Hosted	`malta-ram-hosted.ld`
	RAM Unhosted	`malta-ram.ld`
	YAMON	`malta-yamon.ld`

MIPSsim		
Processor name:	unspecified	
Processor options:	none	
Memory regions:	`ram`	
Linker scripts:	Hosted	`mipssim-hosted.ld`
	Unhosted	`mipssim.ld`

5.7. Interrupt Vector Tables

5.8. Regions and Memory Sections

The following regions are defined for MIPS ELF.

Region	**Contents**
`ram`	`.data` and `.bss` sections. In ram-based profiles, also contains `.text` and other program-like sections.

Note that not all regions are provided in every linker script or profile; see the documentation of the individual linker scripts in Section 5.6, “Supported Boards for MIPS ELF”, above.

Regions documented as "Memory regions" correspond to similarly-named program sections. For example, the linker script assigns the `.ram` section to the `ram` region. You can explicitly locate data or code in these sections using section attributes in your source C or C++ code. Section attributes are especially useful on code compiled for boards that include special memory banks, such as a fast on-chip cache memory, in addition to the default `ram` and/or `rom` regions. CS3 arranges for additional data-like sections to be initialized in the same way as the default `.data` section.

As an example to illustrate the attribute syntax, you can put a variable `v` in the `.ram` section using:

```
int v __attribute__ ((section (".ram")));
```

To declare a function `f` in this section, use:

```
int f (void) __attribute__ ((section (".ram"))) {...}
```

For more information about attribute syntax, see the GCC manual.

Regions documented as "Other regions" do not have a corresponding program section. Typically, these regions correspond to memory-mapped control and I/O registers that cannot be used for general data or program storage. If you need to manipulate data in these regions, you can use the CS3 memory layout facilities declared in `cs3.h`, as described in Section 5.3, “Memory Layout”.

Memory maps for boards supported by Sourcery G++ Lite for MIPS ELF are documented in the linker scripts in the `mips-sde-elf/lib/` subdirectory of your Sourcery G++ installation directory.

Chapter 6
Next Steps with Sourcery G++

This chapter describes where you can find additional documentation and information about using Sourcery G++ Lite and its components.

6.1. Sourcery G++ Subscriptions

CodeSourcery offers two levels of Sourcery G++ subscriptions. Professional Edition subscriptions include unlimited support, with no per-incident fees. CodeSourcery's support is provided by the same engineers who build Sourcery G++, and covers questions about installing and using Sourcery G++, the C and C++ programming languages, and all other topics relating to Sourcery G++. CodeSourcery provides updated versions of Sourcery G++ on demand to resolve critical problems reported by Professional Edition subscribers. Personal Edition subscriptions do not include support, but do include access to updates as long as the subscription remains active.

Subscription editions of Sourcery G++ also include many additional features not included in the free Lite editions:

- **Sourcery G++ IDE.** The Sourcery G++ IDE, based on Eclipse, provides a fully visual environment for developing applications, including an automated project builder, syntax-highlighting editor, and a graphical debugging interface. The debugger provides features especially useful to embedded systems programmers, including the ability to step through code at both the source and assembly level, view registers, and examine stack traces. CodeSourcery's enhancements to Eclipse include improved support for hardware debugging via JTAG or ICE units and complete integration with the rest of Sourcery G++.
- **Debug Sprites.** Sourcery G++ Debug Sprites provide hardware debugging support using JTAG and ICE devices. On some systems, Sourcery G++ Sprites can automatically program flash memory and display control registers. Debug Sprites included in Lite editions of Sourcery G++ include only a subset of the functionality of the Sprites in the subscription editions.
- **CS3.** CS3 provides a uniform, cross-platform approach to board initialization and interrupt handling on bare-metal ELF and EABI platforms. Subscription versions of Sourcery G++ include CS3 support for an expanded set of boards. In addition, the Sourcery G++ Board Builder allows you to extend the power of CS3 to cover custom board definitions. The Board Builder is fully integrated with the Sourcery G++ IDE and Debug Sprites.
- **CodeSourcery C Library.** Subscription versions of Sourcery G++ for bare-metal targets include the CodeSourcery C Library, a proprietary library implementation that is optimized to be smaller and faster than the Newlib C library included with Lite editions of Sourcery G++.
- **QEMU Instruction Set Simulator.** The QEMU instruction set simulator can be used to run — and debug — programs even without target hardware. Most bare-metal configurations of Sourcery G++ include QEMU and linker scripts targeting the simulator. Configurations of Sourcery G++ for GNU/Linux targets include a user-space QEMU emulator that runs on Linux hosts.
- **Sysroot Utilities.** Subscription editions of Sourcery G++ include a set of sysroot utilities for GNU/Linux targets. These utilities simplify use of the Sourcery G++ dynamic linker and shared libraries on the target and also support remote debugging with **gdbserver**.
- **GNU/Linux Prelinker.** For select GNU/Linux target systems, Sourcery G++ includes the GNU/Linux prelinker. The prelinker is a postprocessor for GNU/Linux applications which can dramatically reduce application launch time. CodeSourcery has modified the prelinker to operate on non-GNU/Linux host systems, including Microsoft Windows.
- **Library Reduction Utility.** Sourcery G++ also includes a Library Reduction Utility for GNU/Linux targets. This utility allows the GNU C Library to be relinked to include only those functions used by a given collection of binaries.

- **Additional Libraries.** For some platforms, additional run-time libraries optimized for particular CPUs are available. Pre-built binary versions of the libraries with debug information are also available to subscribers.
- **Additional Documentation.** Subscription customers receive expanded access to the Sourcery G++ Knowledge Base, covering many more tips, howtos, and application notes to help you make the best use of Sourcery G++.

If you would like more information about Sourcery G++ subscriptions, including a price quote or information about evaluating Sourcery G++, please send email to `<sales@codesourcery.com>`.

If you have a Sourcery G++ subscription, you may access your account by visiting the Sourcery G++ Portal[1]. If you have a support account, but are unable to log in, send email to `<support@codesourcery.com>`.

6.2. Sourcery G++ Knowledge Base

The Sourcery G++ Knowledge Base is available to registered users at the Sourcery G++ Portal[2]. Here you can find solutions to common problems including installing Sourcery G++, making it work with specific targets, and interoperability with third-party libraries. There are also additional example programs and tips for making the most effective use of the toolchain and for solving problems commonly encountered during debugging. The Knowledge Base is updated frequently with additional entries based on inquiries and feedback from customers.

6.3. Manuals for GNU Toolchain Components

Sourcery G++ Lite includes the full user manuals for each of the GNU toolchain components, such as the compiler, linker, assembler, and debugger. Most of the manuals include tutorial material for new users as well as serving as a complete reference for command-line options, supported extensions, and the like.

When you install Sourcery G++ Lite, links to both the PDF and HTML versions of the manuals are created in the shortcuts folder you select. If you elected not to create shortcuts when installing Sourcery G++ Lite, the documentation can be found in the `share/doc/sourceryg++-mips-sde-elf/` subdirectory of your installation directory.

In addition to the detailed reference manuals, Sourcery G++ Lite includes a Unix-style manual page for each toolchain component. You can view these by invoking the **man** command with the pathname of the file you want to view. For example, you can first go to the directory containing the man pages:

```
> cd $INSTALL/share/doc/sourceryg++-mips-sde-elf/man/man1
```

Then you can invoke **man** as:

```
> man ./mips-sde-elf-gcc.1
```

Alternatively, if you use **man** regularly, you'll probably find it more convenient to add the directory containing the Sourcery G++ man pages to your `MANPATH` environment variable. This should go in your `.profile` or equivalent shell startup file; see Section 2.6, “Setting up the Environment” for instructions. Then you can invoke **man** with just the command name rather than a pathname.

[1] https://support.codesourcery.com/GNUToolchain/
[2] https://support.codesourcery.com/GNUToolchain/

Finally, note that every command-line utility program included with Sourcery G++ Lite can be invoked with a `--help` option. This prints a brief description of the arguments and options to the program and exits without doing further processing.

Appendix A
Sourcery G++ Lite Release Notes

This appendix contains information about changes in this release of Sourcery G++ Lite for MIPS ELF. You should read through these notes to learn about new features and bug fixes.

A.1. Changes in Sourcery G++ Lite for MIPS ELF

This section documents Sourcery G++ Lite changes for each released revision.

A.1.1. Changes in Sourcery G++ Lite 4.4-58

GDB finish internal error. A bug has been fixed that caused a GDB internal error when using the **finish** command. The bug occurred when debugging optimized code.

GDB update. The included version of GDB has been updated to 6.8.50.20090630. This update adds numerous bug fixes and new features, including support for multi-byte and wide character sets and improved C++ template support.

Arguments to `main`. A bug in CS3 YAMON support has been fixed that formerly caused command-line arguments provided on program startup to be ignored. In YAMON profiles, the arguments are now correctly passed to `main` via *`argc`* and *`argv`*.

GDB and third-party compilers. Some bugs that caused GDB to crash when debugging programs compiled with third-party tools have been fixed. These bugs did not affect programs built with Sourcery G++.

GDB internal warning fix. A GDB bug has been fixed that caused warnings of the form `warning: (Internal error: pc` *`address`* `in read in psymtab, but not in symtab.)`.

`@FILE` **fix.** A bug has been fixed in the processing of *`@FILE`* command-line options by GCC, GDB, and other tools. The bug caused any options in *`FILE`* following a blank line to be ignored.

Preprocessor error handling. The preprocessor now treats failing to find a file referenced via `#include` as a fatal error.

New header file. The header file `mips/m32cache.h` has been added to provide declarations for MIPS32 cache management functions. For more information, refer to the *MIPS® Toolchain Specifics* document.

Bug fix for `read`. A bug in CS3's console I/O support for YAMON has been fixed. The bug caused `read` to return immediately rather than waiting for input to become available.

ELF file corruption with strip. A bug that caused **strip** to corrupt unusual ELF files has been fixed.

GDB support for Cygwin pathnames. A bug in GDB's translation of Cygwin pathnames has been fixed.

Startup code debugging fixes. Two GDB bugs have been fixed that caused errors when debugging startup code. One bug caused an internal error message; the other caused the error `Cannot find bounds of current function`.

MIPS32 TLB support. Functions for initialization and maintenance of the CPU's memory management Translation Lookaside Buffer (TLB) have been added to CS3. For more information about TLB support on MIPS ELF targets, refer to the *MIPS® Toolchain Specifics* document.

Debugging programs built by Green Hills compilers. GDB has been extended to accommodate non-standard debug information produced by some Green Hills toolchains.

Linker script fixes. A bug in CS3 linker scripts for YAMON and simulator profiles has been fixed. The bug resulted in data memory being too small, which sometimes caused the stack to be overwritten during initialization, or reduced space for `malloc` to allocate.

GCC internal compiler error. A bug has been fixed that caused the compiler to crash when optimizing code that casts between structure types and the type of the first field.

ELF Program Headers. The linker now better diagnoses errors in the usage of `FILEHDR` and `PHDRS` keywords in `PHDRS` command of linker scripts. Refer to the linker manual for more information.

A.1.2. Changes in Sourcery G++ Lite 4.4-25

Remote debugging hardware watchpoint bug fix. A GDB bug has been fixed that caused hardware watchpoint hits to be incorrectly reported in some cases.

Optimizer improvements. When optimizing for speed, the compiler now uses improved heuristics to limit certain types of optimizations that may adversely affect both code size and speed. This change also makes it possible to produce better code when optimizing for space rather than speed.

Binutils update. The binutils package has been updated to version 2.19.51.20090709 from the FSF trunk. This update includes numerous bug fixes.

Destructor function bug fix. A bug in CS3 has been fixed that caused functions with the `destructor` attribute not to be run on program termination.

Support for MIPS 1004K cores. Sourcery G++ now includes basic compiler and assembler support for MIPS 1004K cores. Use the `-march=1004kc` (integer cores), `-march=1004kf2_1` (half-speed FPU), `-march=1004kf1_1` (full-speed FPU), or `-march=1004kf` (alias for `1004kf2_1`) command-line options.

Malta board support. The Malta 24Kc board definition has been removed from CS3. This board definition was made obsolete in a previous release by the addition of a new generic Malta board definition that is not restricted to 24Kc processors. You should use this generic Malta board definition in place of the deleted 24Kc-specific one. For example, if you were formerly using the `malta-24kc-yamon.ld` linker script, you should now use `malta-yamon.ld`. Refer to Chapter 5, “CS3™: The CodeSourcery Common Startup Code Sequence” for a full listing of supported boards and provided linker scripts in this version of Sourcery G++ Lite.

Register variable corruption. A compiler bug has been fixed that caused incorrect code to be generated when the frame pointer or other special-use registers are used as explicit local register variables, introduced via the `asm` keyword on their declarations.

Stack unwinding termination bug fix. A bug has been fixed that caused GDB not to detect the outermost frame correctly while doing stack unwinding. The bug sometimes caused the debugger to go into an infinite loop, or other unpredictable behavior.

`-fremove-local-statics` optimization. The `-fremove-local-statics` optimization is now enabled by default at `-O2` and higher optimization levels.

Elimination of spurious warnings about `NULL`. The C++ compiler no longer issues spurious warnings about comparisons between pointers to members and `NULL`.

Profiling support. Profiling is now supported for MIPS ELF targets. For more information on profiling with **gprof**, please see Section 3.6, “Profiling Support”.

Vectorizer improvements. The compiler now generates improved code for accesses to static nested array variables (e.g. `static int foo[8][8];`).

Function attributes to support interrupt handling. Support for the `interrupt` attribute has been implemented. `use_debug_exception`, `keep_interrupts_masked`, and `use_shadow_register_set` have also been implemented. These are attributes which can be used to modify the behavior of the interrupt handler. For more information on how to use these attributes, please refer to the GCC manual.

GCC version 4.4.1. Sourcery G++ Lite for MIPS ELF is now based on GCC version 4.4.1. For more information about changes from GCC version 4.3 that was included in previous releases, see `http://gcc.gnu.org/gcc-4.4/changes.html`.

Linker map address sorting. The map generated by the linker `-Map` option now lists symbols sorted by address.

Floating-point register initialization for YAMON. A bug that caused the floating-point registers to be initialized in the incorrect mode has been fixed. The reset code for YAMON applications now initializes the floating-point register mode to conform to the compilation mode of the application.

A.1.3. Changes in Sourcery G++ Lite 4.3-221

No significant changes. There are no significant changes for MIPS ELF in this release.

A.1.4. Changes in Sourcery G++ Lite 4.3-219

Malta board support memory map. Other recent changes to Sourcery G++ have required changing CS3's memory map for MIPS Malta boards to reserve an area of low memory for interrupt vectors. If you are using a CS3-provided Malta linker script for your program, you will pick up the changes automatically. If you are using a copied or custom linker script, you may need to adjust it for this change.

MIPSsim board support. CS3 now includes a board definition intended specifically for use with MIPSsim targets. If you were previously using a Malta linker script to build a program intended to run on MIPSsim, you must change to use the new MIPSsim linker script instead. When invoking the linker from the command line, use the `-T mipssim-hosted.ld` option to select the new linker script. Using the new MIPSsim-specific linker script for MIPSsim targets is now necessary because other recent changes to Sourcery G++ have required changes to the memory map used by CS3 that are incompatible between MIPSsim and Malta hardware targets.

A.1.5. Changes in Sourcery G++ Lite 4.3-199

MIPS Malta board support. Support for a generic Malta board has been added to CS3. This is similar to the existing Malta 24Kc board support, but is not specific to a particular processor, to reflect the fact that these boards can be configured with a number of different processors. If you were previously using the Malta 24Kc CS3 board support with a processor other than the 24Kc, you should switch to using the new generic Malta board instead. Refer to Chapter 5, "CS3™: The CodeSourcery Common Startup Code Sequence" for more information about CS3 support for these boards.

Incorrect linker diagnostic removed. The linker has been corrected to not emit an error message when the load address of an output section with no contents overlaps an output section with contents. This can occur in linker scripts that use `MEMORY` regions and `AT>` to place initialized contents into ROM.

GDB backwards compatibility fix. A bug has been fixed that caused GDB to crash when loading symbols from binaries built by very old versions of GCC.

Overloaded function resolution. The C++ compiler now correctly diagnoses an error when the second operand of a comma expression is an unresolved set of overloaded functions. Previously, it incorrectly used the context of the comma expression to resolve the function.

Pointer-to-member functions. A bug has been fixed that caused the C++ compiler to crash when compiling a pointer-to-member function reference without an explicit & operator. This syntax is allowed only when the `-fms-extensions` command-line option is used.

New assembler option: `-mfix-24k`. The assembler now accepts the `-mfix-24k` command-line option. The use of this option causes the assembler to work around hardware errata in the `eret` and `deret` instructions on 24K and 24KE cores.

A.1.6. Changes in Sourcery G++ Lite 4.3-152

Reduced compilation time. Compilation and build times when using Sourcery G++ Lite are now slightly faster. This performance improvement is the result of building the compilers and other host tools with a recent version of Sourcery G++, rather than an older GCC version.

Linker script load address processing. A bug in the linker has been fixed affecting linker scripts using `AT>region` to set the load address. This now follows the documented behavior of maintaining the virtual address to load address difference in output section statements. Refer to the "Output Section LMA" section of the linker manual for details of how to control the load address.

Hardware floating point emulation library. The hardware floating point emulation support which was formerly included with the SDE library is now available as part of CS3. This library provides trap handlers for unsupported floating-point instructions, which invoke the corresponding soft-float library routines. To use the library in your code, compile with `-mhard-float` and link with `-lcs3-mips-cp1 -lcs3-mips-fpemu -Wl,--defsym,__cs3_mips_float_type=2`. For more information about floating-point support on MIPS ELF targets, refer to the *MIPS® Toolchain Specifics* document.

mips-sde-elf-objcopy bug fix. A bug has been fixed that caused **mips-sde-elf-objcopy** to issue an error when generating output in the Intel HEX format and using `--change-section-lma` to change section addresses.

Linker script search path. The bug in the linker has been fixed that caused it not to follow its documented behavior for searching for linker scripts named with the `-T` option. Now scripts are looked up first in the current directory, then in library directories specified with `-L` command-line options, and finally in the default system linker script directory.

Internal compiler error when optimizing. A bug has been fixed that caused `internal compiler error: in build2_stat` when compiling.

Corruption of block-scope variables. A compiler optimization bug that sometimes caused corruption of stack-allocated variables has been fixed. The bug affected variables declared in a local block scope in functions containing multiple non-overlapping lexical block scopes, a technique commonly used by programmers to reduce stack frame size. In some rare cases, other optimizations performed by the compiler were ignoring the local extent of such block-scope variables.

A.1.7. Changes in Sourcery G++ Lite 4.3-147

Optimized math routines. The Newlib implementations of `rint`, `drem`, `sqrtf` and `sqrt` have been replaced with the versions of these functions that were formerly included with the SDE math library, and are optimized specifically for MIPS targets.

mips-sde-elf-objdump bug fix. A bug has been fixed that caused **mips-sde-elf-objdump** to enter an infinite loop.

Incorrect code when using `-falign-labels`. A bug that caused the compiler to generate incorrect code for `switch` statements when the `-falign-labels` option is used has been fixed.

Debug section placement. A linker script bug in CS3 has been fixed that caused `.debug_ranges` debug sections to be misplaced.

MDI semihosting. A bug in MDI semihosting that could result in a crash when making a system call (such as `read` or `write`) has been fixed.

Interrupting the target from the debugger. GDB has been improved to be more responsive to attempts to interrupt the target (as by a **Ctrl**+**C** when using GDB from the command line) during execution of programs using semihosting.

Loop optimization improvements. A new option, `-fpromote-loop-indices`, has been added to the compiler. Specifying this option enables an optimization that improves the performance of loops with index variables of integer types narrower than the target machine word size, such as `char` or `short`. This optimization also applies to `int` on 64-bit targets.

Optimized string and memory functions. The Newlib implementations of `memcpy`, `memcmp`, `bzero`, `strcmp`, `strcpy`, `strlen` and `memset` have been replaced with the versions of these functions that were formerly included with the SDE C library, and are optimized specifically for MIPS targets.

Remote debugging connection auto-retry. The **target remote** command within GDB now uses a configurable auto-retry timeout when establishing TCP connections. This is useful in avoiding race conditions when the remote GDB stub or GDB server is launched simultaneously with GDB. The auto-retry behavior is enabled by default; refer to the GDB manual for details.

Extraneous linker error messages. A linker bug that caused extraneous error messages of the form `Dwarf Error: Offset (507) greater than or equal to .debug_str size (421).` has been corrected. This bug did not affect the correctness of output binaries.

GDB quit error. A bug in GDB has been fixed that caused **quit** to report `Quitting: You can't do that without a process to debug.` when debugging a core dump file.

GDB update. The included version of GDB has been updated to 6.8.50.20081022. This update includes numerous bug fixes.

A.1.8. Changes in Sourcery G++ Lite 4.3-113

GCC version 4.3.3. Sourcery G++ Lite for MIPS ELF is now based on GCC version 4.3.3. This is a bug fix update to GCC. For more information about changes from GCC version 4.3.2 that was included in previous releases, see `http://gcc.gnu.org/gcc-4.3/changes.html`.

Arguments to `main`. A bug in CS3 hosting support has been fixed that formerly caused command-line arguments provided on program startup to be ignored. In hosted environments, the arguments are now correctly passed to `main` via *`argc`* and *`argv`*.

Internal compiler error with `-O3` or `-fpredictive-commoning`. A bug has been fixed that caused internal compiler errors when compiling some code with `-O3` or `-fpredictive-commoning`.

Debug information for anonymous structure types. A GCC bug in the generation of debug information for anonymous structure types in C++ code has been fixed. The bug caused printing the type information for such structures in the debugger (via the **ptype** command) to fail with an error message.

Out-of-range branches. A bug has been fixed that caused the compiler to emit incorrect branch code in some very large functions when generating position-independent code (`-fpic`) for O32 (`-mabi=32`) or O64 (`-mabi=o64`) ABIs.

Newlib update. The Newlib package has been updated to version 1.17.0, with additions from the community CVS trunk as of 2009-02-24. This update provides new C99 wide-character functions; POSIX regex functions; string-function performance improvements; an improved `sprintf` implementation that no longer requires I/O functions like `_open`, `_write`, and `_close`; and other bug fixes and improvements. For more information, refer to the Newlib C Library and Math Library manuals, and to the Newlib web site at `http://sourceware.org/newlib/`.

Installer fails during upgrade. The Sourcery G++ installer for Microsoft Windows hosts could fail during an upgrade while waiting for the previous version to be uninstalled. This bug has been fixed.

Uninstaller removed by upgrade. The uninstaller could be incorrectly deleted during an upgrade on Microsoft Windows hosts. This bug has been fixed.

Compile-time error for some `-march` options. A bug has been fixed that caused the error message: `mips-sde-elf-gcc: switch '|march=octeon' does not start with '-'` to be reported. The bug affected programs compiled with the options `-march=mips64`, `-march=5k`, `-march=20k`, `-march=sb1` and `-march=r71000`.

Internal compiler errors when optimizing. A defect that occasionally caused internal compiler errors when partial redundancy elimination (PRE) optimization was enabled has been corrected.

Install directory pathnames. Bugs in the install and uninstall scripts for Linux hosts that caused errors or incorrect behavior when the Sourcery G++ install directory pathname contains whitespace characters have been fixed.

Temporary files on Microsoft Windows. On Microsoft Windows hosts, Sourcery G++ Lite now uses the standard Windows algorithm to choose the directory in which to place temporary files. This change eliminates a crash that occurred if none of the `TEMP`, `TMP`, or `TMPDIR` variables were set to a suitable directory.

Binutils update. The binutils package has been updated to version 2.19.51.20090205 from the FSF trunk. This update includes numerous bug fixes.

CS3 board and processor support. CS3 board and processor support has been cleaned up to remove entries that are not appropriate for or supported by Sourcery G++ Lite on MIPS ELF targets. This includes processors for which Sourcery G++ Lite does not include appropriate run-time libraries. These changes are intended to simplify processor and board selection. For the full list of boards supported by CS3, refer to Chapter 5, “CS3™: The CodeSourcery Common Startup Code Sequence”.

Internal compiler error with `-fremove-local-statics`. An internal compiler error that occurred when using the `-fremove-local-statics` option has been fixed. The error occurred when compiling code with function-local `static` array or structure variables.

A.1.9. Changes in Sourcery G++ Lite 4.3-81

C++ named operators bug fix. A bug has been fixed that caused the compiler to crash in some cases when the C++ operators `and_eq`, `bitand`, `bitor`, `compl`, `not_eq`, `or_eq` and `xor_eq` were used in contexts where the preprocessor converts their names to strings.

GDB target extended-remote error. A bug in GDB has been fixed that caused **target extended-remote** to report `Remote failure reply: E01` if no remote program was running.

GDB segment warning. Some compilers produce binaries including uninitialized data regions, such as the stack and heap. GDB incorrectly displayed the warning `Loadable segment "`*`name`*`" outside of ELF segments` for such binaries; the warning has now been fixed.

A.1.10. Changes in Sourcery G++ Lite 4.3-59

Setting breakpoints on Windows. A bug in GDB on Microsoft Windows hosts has been fixed. The bug caused setting breakpoints on a source line by using the file's full path to fail with `No source file named` *`filename`*`.`

Handling of out-of-range values by `strtof`. The `strtof` function now sets `errno` to `ERANGE` when the input is not representable as a `float`, as required by the ISO C standard.

Printing casted values in GDB. A GDB bug that caused incorrect output for expressions containing casts, such as in the `print *(Type *)ptr` command, has been fixed.

Bug fix for objcopy/strip. An objcopy bug that corrupted COMDAT groups when creating new binaries has been fixed. This bug also affected **strip -g**.

Bug fix for assembly listing. A bug that caused the assembler to produce corrupted listings (via the `-a` option) on Windows hosts has been fixed.

DSP support. The compiler now automatically enables support for the DSP ASE when the `-march` option is used to specify a core in the 24KE, 34K, or 74K families. It is no longer necessary to provide the `-mdsp` option explicitly.

GDB update. The included version of GDB has been updated to 6.8.50.20080821. This update adds numerous bug fixes and new features, including support for decimal floating point, the new **find** command to search memory, the new `/m` (mixed source and assembly) option to the **disassemble** command, and the new **macro define** command to define C preprocessor macros interactively.

PIE linking fix. A bug in the GCC `-fpie` and `-fPIE` options has been fixed. The bug caused linker errors referring to `R_MIPS_HI16`.

Binutils support for DWARF Version 3. The **addr2line** command now supports binaries containing DWARF 3 debugging information. The **ld** command can display error messages with source locations for input files containing DWARF 3 debugging information.

GDB support for YAMON. GDB now supports debugging via the YAMON boot loader GDB stub. Consult YAMON documentation for details on enabling the GDB stub.

GDB Support for MIPSsim on Windows. A bug in the GDB support for MIPSsim on Microsoft Windows hosts has been fixed. The bug caused the `target mdi` command to fail with the error `Cannot find MIPSsim config file template: mipssim.cfg`.

CodeSourcery Common Startup Code Sequence. Support for CS3, CodeSourcery's unified startup scheme, has been added to this release. CS3 replaces the MIPS-provided MDI startup code and linker scripts included in previous releases. Refer to Chapter 5, "CS3™: The CodeSourcery Common Startup Code Sequence" for more information about CS3, including details about the boards and linker scripts supported by this release. Note that the Malta board configuration is usable by MIPSsim and the included GDB simulator as well as actual Malta hardware targets.

GDB display of source. A bug has been fixed that prevented GDB from locating debug information in some cases. The debugger failed to display source code for or step into the affected functions.

FPU defaults. The `-ffast-math` option now causes subnormal numbers to be immediately flushed to zero. It also sets the rounding mode to round-to-nearest.

Connecting to the target using a pipe. A bug in GDB's **target remote | *`program`*** command has been fixed. When launching the specified *`program`* failed, the bug caused GDB to crash, hang, or give a message `Error: No Error`.

Output files removed on error. When GCC encounters an error, it now consistently removes any incomplete output files that it may have created.

Placing bss-like regions in load regions. The linker no longer issues an incorrect error message when a bss-like section is placed at specific load region. The linker formerly incorrectly considered the section as taking up space in the load region.

`-mwarn-framesize=`*`size`* option. GCC has a new command-line option, `-mwarn-framesize=`*`size`*, which causes warnings if any function's stack frame exceeds the given *`size`*. This option is useful when generating code for environments with limited or absent stack, e.g., BIOS.

Newlib manuals. The documentation packaged with Sourcery G++ Lite now includes the Newlib C Library and Math Library manuals.

GCC version 4.3.2. Sourcery G++ Lite for MIPS ELF is now based on GCC version 4.3.2. For more information about changes from GCC version 4.2 that was included in previous releases, see `http://gcc.gnu.org/gcc-4.3/changes.html`.

Spurious GDB error message fixed. A spurious `Current thread went away!?` message is no longer generated when using GDB to debug programs running on cores that do not support hardware multi-threading. This problem was specific to the MDI target support in GDB.

Unnecessary section removed. A linker bug that caused an unnecessary `.rel.dyn` section to be placed in the executable has been fixed.

Linker bug fix for `--gc-sections`. A linker bug that caused certain linker-generated sections to be incorrectly omitted from the executable when the `--gc-sections` option is used has been fixed.

Errors after loading the debugged program. An intermittent GDB bug has been fixed. The bug could cause a GDB internal error after the **load** command.

Bug fix for objdump on Windows. An objdump bug that caused the `-S` option not to work on Windows in some cases has been fixed.

Persistent remote server connections. A GDB bug has been fixed that caused the **target extended-remote** command to fail to tell the remote server to make the connection persistent across program invocations.

A.1.11. Changes in Older Releases

For information about changes in older releases of Sourcery G++ Lite for MIPS ELF, please refer to the Getting Started guide packaged with those releases.

Appendix B
Sourcery G++ Lite Licenses

Sourcery G++ Lite contains software provided under a variety of licenses. Some components are “free” or “open source” software, while other components are proprietary. This appendix explains what licenses apply to your use of Sourcery G++ Lite. You should read this appendix to understand your legal rights and obligations as a user of Sourcery G++ Lite.

B.1. Licenses for Sourcery G++ Lite Components

The table below lists the major components of Sourcery G++ Lite for MIPS ELF and the license terms which apply to each of these components.

Some free or open-source components provide documentation or other files under terms different from those shown below. For definitive information about the license that applies to each component, consult the source package corresponding to this release of Sourcery G++ Lite. Sourcery G++ Lite may contain free or open-source components not included in the list below; for a definitive list, consult the source package corresponding to this release of Sourcery G++ Lite.

Component	License
GNU Compiler Collection	GNU General Public License 3.0 [1]
GNU Binary Utilities	GNU General Public License 3.0 [2]
GNU Debugger	GNU General Public License 3.0 [3]
CodeSourcery Common Startup Code Sequence	CodeSourcery License
Newlib C Library	BSD License. For the text of the license and a complete list of copyright holders, see COPYING.NEWLIB included in the source package.
GNU Make	GNU General Public License 2.0 [4]
GNU Core Utilities	GNU General Public License 2.0 [5]

The CodeSourcery License is available in Section B.2, “Sourcery G++ Software License Agreement”.

> **Important**
>
> Although some of the licenses that apply to Sourcery G++ Lite are “free software” or “open source software” licenses, none of these licenses impose any obligation on you to reveal the source code of applications you build with Sourcery G++ Lite. You can develop proprietary applications and libraries with Sourcery G++ Lite.

Sourcery G++ Lite may include some third party example programs and libraries in the `share/sourceryg++-mips-sde-elf-examples` subdirectory. These examples are not covered by the Sourcery G++ Software License Agreement. To the extent permitted by law, these examples are provided by CodeSourcery as is with no warranty of any kind, including implied warranties of merchantability or fitness for a particular purpose. Your use of each example is governed by the license notice (if any) it contains.

[1] http://www.gnu.org/licenses/gpl.html
[2] http://www.gnu.org/licenses/gpl.html
[3] http://www.gnu.org/licenses/gpl.html
[4] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
[5] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

B.2. Sourcery G++™ Software License Agreement

1. **Parties.** The parties to this Agreement are you, the licensee (“You” or “Licensee”) and CodeSourcery. If You are not acting on behalf of Yourself as an individual, then “You” means Your company or organization.

2. **The Software.** The Software licensed under this Agreement consists of computer programs and documentation referred to as Sourcery G++™ Lite Edition (the “Software”).

3. **Definitions.**

   3.1. **CodeSourcery Proprietary Components.** The components of the Software that are owned and/or licensed by CodeSourcery and are not subject to a “free software” or “open source” license, such as the GNU Public License. The CodeSourcery Proprietary Components of the Software include, without limitation, the Sourcery G++ Installer, any Sourcery G++ Eclipse plug-ins, and any Sourcery G++ Debug Sprite. For a complete list, refer to the *Getting Started Guide* included with the distribution.

   3.2. **Open Source Software Components.** The components of the Software that are subject to a “free software” or “open source” license, such as the GNU Public License.

   3.3. **Proprietary Rights.** All rights in and to copyrights, rights to register copyrights, trade secrets, inventions, patents, patent rights, trademarks, trademark rights, confidential and proprietary information protected under contract or otherwise under law, and other similar rights or interests in intellectual or industrial property.

   3.4. **Redistributable Components.** The CodeSourcery Proprietary Components that are intended to be incorporated or linked into Licensee object code developed with the Software. The Redistributable Components of the Software include, without limitation, the CSLIBC run-time library and the CodeSourcery Common Startup Code Sequence (CS3). For a complete list, refer to the *Getting Started Guide* included with the distribution.

4. **License Grant to Proprietary Components of the Software.** You are granted a non-exclusive, royalty-free license (a) to install and use the CodeSourcery Proprietary Components of the Software, (b) to transmit the CodeSourcery Proprietary Components over an internal computer network, (c) to copy the CodeSourcery Proprietary Components for Your internal use only, and (d) to distribute the Redistributable Component(s) in binary form only and only as part of Licensee object code developed with the Software that provides substantially different functionality than the Redistributable Component(s).

5. **Restrictions.** You may not: (i) copy or permit others to use the CodeSourcery Proprietary Components of the Software, except as expressly provided above; (ii) distribute the CodeSourcery Proprietary Components of the Software to any third party, except as expressly provided above; or (iii) reverse engineer, decompile, or disassemble the CodeSourcery Proprietary Components of the Software, except to the extent this restriction is expressly prohibited by applicable law.

6. **“Free Software” or “Open Source” License to Certain Components of the Software.** This Agreement does not limit Your rights under, or grant You rights that supersede, the license terms of any Open Source Software Component delivered to You by CodeSourcery. Sourcery G++ includes components provided under various different licenses. The *Getting Started Guide* provides an overview of which license applies to different components. Definitive licensing

information for each “free software” or “open source” component is available in the relevant source file.

7. **CodeSourcery Trademarks.** Notwithstanding any provision in a “free software” or “open source” license agreement applicable to a component of the Software that permits You to distribute such component to a third party in source or binary form, You may not use any CodeSourcery trademark, whether registered or unregistered, including without limitation, CodeSourcery™, Sourcery G++™, the CodeSourcery crystal ball logo, or the Sourcery G++ splash screen, or any confusingly similar mark, in connection with such distribution, and You may not recompile the Open Source Software Components with the `--with-pkgversion` or `--with-bugurl` configuration options that embed CodeSourcery trademarks in the resulting binary.

8. **Term and Termination.** This Agreement shall remain in effect unless terminated pursuant to this provision. CodeSourcery may terminate this Agreement upon seven (7) days written notice of a material breach of this Agreement if such breach is not cured; provided that the unauthorized use, copying, or distribution of the CodeSourcery Proprietary Components of the Software will be deemed a material breach that cannot be cured.

9. **Transfers.** You may not transfer any rights under this Agreement without the prior written consent of CodeSourcery, which consent shall not be unreasonably withheld. A condition to any transfer or assignment shall be that the recipient agrees to the terms of this Agreement. Any attempted transfer or assignment in violation of this provision shall be null and void.

10. **Ownership.** CodeSourcery owns and/or has licensed the CodeSourcery Proprietary Components of the Software and all intellectual property rights embodied therein, including copyrights and valuable trade secrets embodied in its design and coding methodology. The CodeSourcery Proprietary Components of the Software are protected by United States copyright laws and international treaty provisions. CodeSourcery also owns all rights, title and interest in and with respect to its trade names, domain names, trade dress, logos, trademarks, service marks, and other similar rights or interests in intellectual property. This Agreement provides You only a limited use license, and no ownership of any intellectual property.

11. **Warranty Disclaimer; Limitation of Liability.** CODESOURCERY AND ITS LICENSORS PROVIDE THE SOFTWARE “AS-IS” AND PROVIDED WITH ALL FAULTS. CODESOURCERY DOES NOT MAKE ANY WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. CODESOURCERY SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SYSTEM INTEGRATION, AND DATA ACCURACY. THERE IS NO WARRANTY OR GUARANTEE THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED, ERROR-FREE, OR VIRUS-FREE, OR THAT THE SOFTWARE WILL MEET ANY PARTICULAR CRITERIA OF PERFORMANCE, QUALITY, ACCURACY, PURPOSE, OR NEED. YOU ASSUME THE ENTIRE RISK OF SELECTION, INSTALLATION, AND USE OF THE SOFTWARE. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS AGREEMENT. NO USE OF THE SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

12. **Local Law.** If implied warranties may not be disclaimed under applicable law, then ANY IMPLIED WARRANTIES ARE LIMITED IN DURATION TO THE PERIOD REQUIRED BY APPLICABLE LAW.

13. **Limitation of Liability.** INDEPENDENT OF THE FORGOING PROVISIONS, IN NO EVENT AND UNDER NO LEGAL THEORY, INCLUDING WITHOUT LIMITATION, TORT, CONTRACT, OR STRICT PRODUCTS LIABILITY, SHALL CODESOURCERY BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCID-

ENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND, INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER MALFUNCTION, OR ANY OTHER KIND OF COMMERCIAL DAMAGE, EVEN IF CODESOURCERY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY TO THE EXTENT PROHIBITED BY APPLICABLE LAW. IN NO EVENT SHALL CODESOURCERY'S LIABILITY FOR ACTUAL DAMAGES FOR ANY CAUSE WHATSOEVER, AND REGARDLESS OF THE FORM OF ACTION, EXCEED THE AMOUNT PAID BY YOU IN FEES UNDER THIS AGREEMENT DURING THE PREVIOUS ONE YEAR PERIOD.

14. **Export Controls.** You agree to comply with all export laws and restrictions and regulations of the United States or foreign agencies or authorities, and not to export or re-export the Software or any direct product thereof in violation of any such restrictions, laws or regulations, or without all necessary approvals. As applicable, each party shall obtain and bear all expenses relating to any necessary licenses and/or exemptions with respect to its own export of the Software from the U.S. Neither the Software nor the underlying information or technology may be electronically transmitted or otherwise exported or re-exported (i) into Cuba, Iran, Iraq, Libya, North Korea, Sudan, Syria or any other country subject to U.S. trade sanctions covering the Software, to individuals or entities controlled by such countries, or to nationals or residents of such countries other than nationals who are lawfully admitted permanent residents of countries not subject to such sanctions; or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals and Blocked Persons or the U.S. Commerce Department's Table of Denial Orders. By downloading or using the Software, Licensee agrees to the foregoing and represents and warrants that it complies with these conditions.

15. **U.S. Government End-Users.** The Software is a “commercial item,” as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of “commercial computer software” and “commercial computer software documentation,” as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire the Software with only those rights set forth herein.

16. **Licensee Outside The U.S.** If You are located outside the U.S., then the following provisions shall apply: (i) Les parties aux presentes confirment leur volonte que cette convention de meme que tous les documents y compris tout avis qui siy rattache, soient rediges en langue anglaise (translation: “The parties confirm that this Agreement and all related documentation is and will be in the English language.”); and (ii) You are responsible for complying with any local laws in your jurisdiction which might impact your right to import, export or use the Software, and You represent that You have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

17. **Severability.** If any provision of this Agreement is declared invalid or unenforceable, such provision shall be deemed modified to the extent necessary and possible to render it valid and enforceable. In any event, the unenforceability or invalidity of any provision shall not affect any other provision of this Agreement, and this Agreement shall continue in full force and effect, and be construed and enforced, as if such provision had not been included, or had been modified as above provided, as the case may be.

18. **Arbitration.** Except for actions to protect intellectual property rights and to enforce an arbitrator's decision hereunder, all disputes, controversies, or claims arising out of or relating to this Agreement or a breach thereof shall be submitted to and finally resolved by arbitration under the rules of the American Arbitration Association (“AAA”) then in effect. There shall be one arbitrator, and such arbitrator shall be chosen by mutual agreement of the parties in accordance with AAA rules. The arbitration shall take place in Granite Bay, California, and may be conducted

by telephone or online. The arbitrator shall apply the laws of the State of California, USA to all issues in dispute. The controversy or claim shall be arbitrated on an individual basis, and shall not be consolidated in any arbitration with any claim or controversy of any other party. The findings of the arbitrator shall be final and binding on the parties, and may be entered in any court of competent jurisdiction for enforcement. Enforcements of any award or judgment shall be governed by the United Nations Convention on the Recognition and Enforcement of Foreign Arbitral Awards. Should either party file an action contrary to this provision, the other party may recover attorney's fees and costs up to $1000.00.

19. **Jurisdiction And Venue.** The courts of Placer County in the State of California, USA and the nearest U.S. District Court shall be the exclusive jurisdiction and venue for all legal proceedings that are not arbitrated under this Agreement.

20. **Independent Contractors.** The relationship of the parties is that of independent contractor, and nothing herein shall be construed to create a partnership, joint venture, franchise, employment, or agency relationship between the parties. Licensee shall have no authority to enter into agreements of any kind on behalf of CodeSourcery and shall not have the power or authority to bind or obligate CodeSourcery in any manner to any third party.

21. **Force Majeure.** Neither CodeSourcery nor Licensee shall be liable for damages for any delay or failure of delivery arising out of causes beyond their reasonable control and without their fault or negligence, including, but not limited to, Acts of God, acts of civil or military authority, fires, riots, wars, embargoes, or communications failure.

22. **Miscellaneous.** This Agreement constitutes the entire understanding of the parties with respect to the subject matter of this Agreement and merges all prior communications, representations, and agreements. This Agreement may be modified only by a written agreement signed by the parties. If any provision of this Agreement is held to be unenforceable for any reason, such provision shall be reformed only to the extent necessary to make it enforceable. This Agreement shall be construed under the laws of the State of California, USA, excluding rules regarding conflicts of law. The application of the United Nations Convention of Contracts for the International Sale of Goods is expressly excluded. This license is written in English, and English is its controlling language.