Sourcery G++ Lite

MIPS GNU/Linux

Sourcery G++ Lite 2011.03-53

Getting Started

Sourcery G++ Lite: MIPS GNU/Linux: Sourcery G++ Lite 2011.03-53: Getting Started

CodeSourcery, Inc.

Abstract

This guide explains how to install and build applications with Sourcery G++ Lite, CodeSourcery's customized and validated 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 GNU/Linux"	This chapter contains information about using Sourcery G++ Lite that is specific to MIPS GNU/Linux 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, "Sourcery G++ Debug Sprite"	This chapter describes the use of the Sourcery G++ Debug Sprite for remote debugging. The Sprite is provided for debugging of the Linux kernel on the target board. This chapter includes information about the debugging devices and boards supported by the Sprite for MIPS GNU/Linux.
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 GNU/Linux. 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 GNU/Linux is intended for developers working on embedded GNU/Linux applications. It may also be used for Linux kernel development and debugging, or to build a GNU/Linux distribution.

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. Note that this checklist is also oriented towards application debugging rather than kernel debugging.

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 GNU/Linux”.

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. When the installation is complete, it offers 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”.

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

Identify your target libraries. Sourcery G++ Lite supports libraries optimized for different targets. Libraries are selected automatically by the linker, depending on the processor and other options you have specified. Refer to Section 3.2, “Library Configurations” for details.

Install runtime libraries on your target machine. In order to run programs built with the Sourcery G++ runtime libraries on target hardware, you must install these libraries, the Sourcery G++ dynamic linker, and other runtime support files -- collectively referred to as the *sysroot* -- on your GNU/Linux target system. Typically, this involves either using third-party tools to build a complete root filesystem including the Sourcery G++ sysroot, or using special commands when linking or running your program so it can find the sysroot installed in another location on the target. Refer to Section 3.5, “Using Sourcery G++ Lite on GNU/Linux Targets” for full discussion of these options.

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.

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.

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

Run your program on the MIPS GNU/Linux target. To run a program using the included Sourcery G++ libraries, you must install the sysroot on the target, as previously discussed. Copy the executable for your program to the target system. The method you use for launching your program depends on how you have installed the libraries and built your program. In some cases, you may need to invoke the Sourcery G++ dynamic linker explicitly. Refer to Section 3.5, “Using Sourcery G++ Lite on GNU/Linux Targets” for details.

Debug your program on the target using GDB server. You can use GDB server on a remote target to debug your program. When debugging a program that uses the included Sourcery G++ libraries, you must use the `gdbserver` executable included in the sysroot, and similar issues with respect to the dynamic linker as discussed previously apply. See Section 3.6, “Using GDB Server for Debugging” for detailed instructions. Once you have started GDB server on the target, you can connect to it from the debugger on your host system. Refer to Section 4.3, “Running Applications from GDB” for instructions on remote debugging from command-line 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-linux-gnu`.

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 2000, Windows XP, Windows Vista, and Windows 7 systems using IA32, AMD64, and Intel 64 processors.
- GNU/Linux systems using IA32, AMD64, or Intel 64 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 512MB 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. When you start the graphical installer, it checks whether there is sufficient disk space before beginning to install. Note that the graphical installer also requires additional temporary disk 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 GNU/Linux” 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].

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

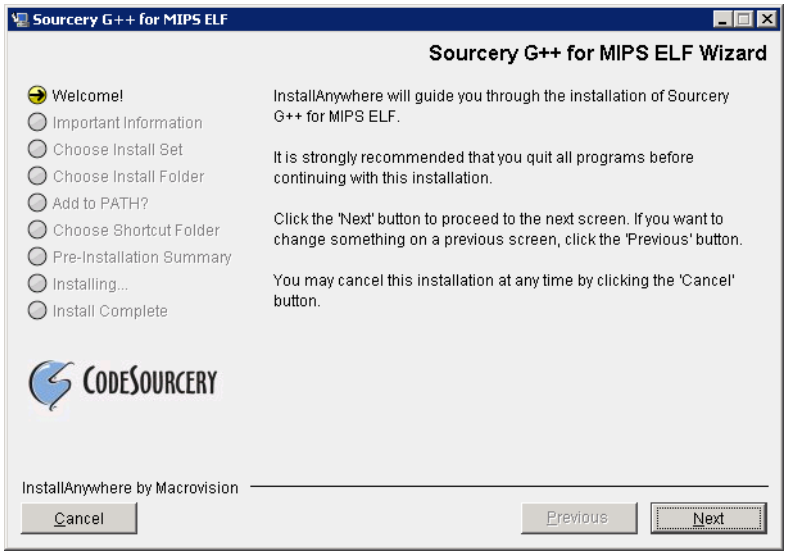
The method used to install Sourcery G++ Lite depends on your host system and the kind of installation package you have downloaded.

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.

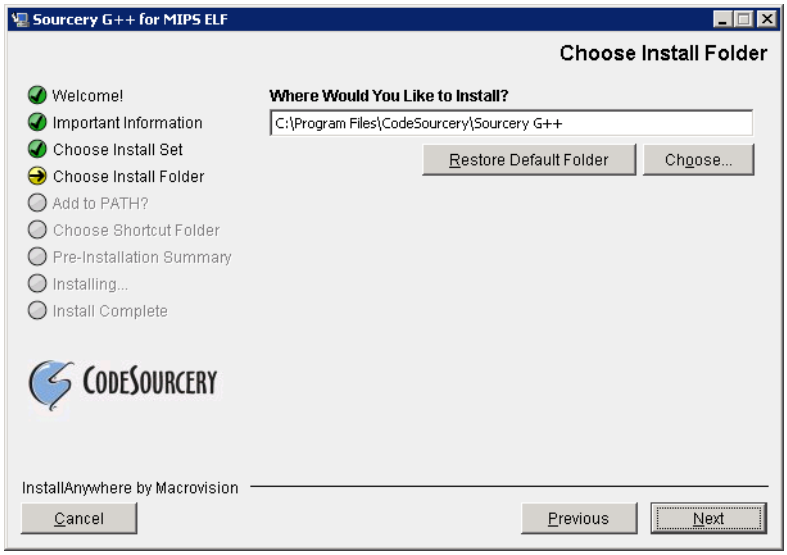
After the installer starts, follow the on-screen dialogs to install Sourcery G++ Lite. The installer is intended to be self-explanatory and on most pages the defaults are appropriate.

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

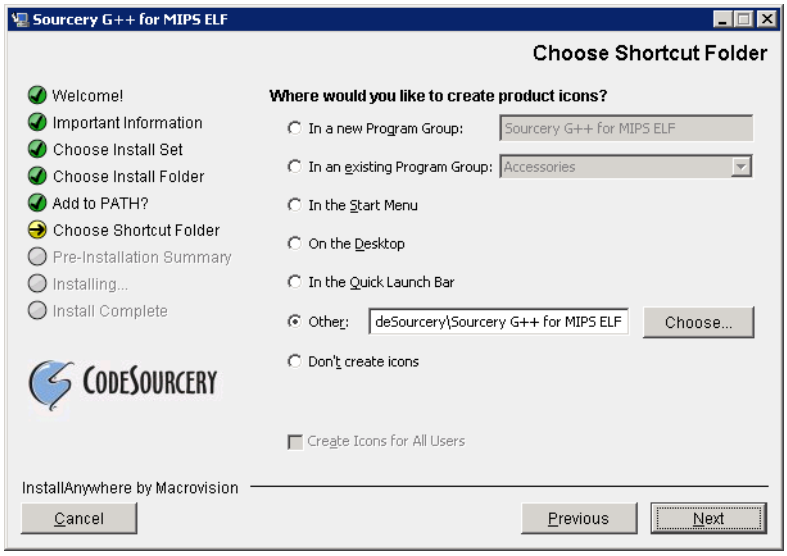


Running the Installer. The graphical installer guides you through the steps to install Sourcery G++ Lite.

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

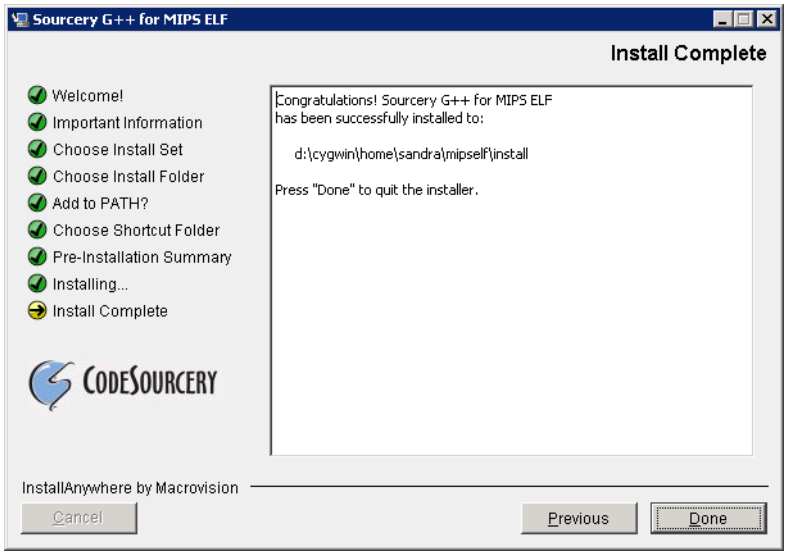


Choose Install Folder. Select the pathname to your install directory.



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++-2011.03`.

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 GNU/Linux 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 GNU/Linux target all begin with `mips-linux-gnu`. 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 Microsoft Windows 7, from the desktop bring up the Start menu and right click on Computer. Select Properties and click on Advanced system settings. 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. Be sure to adjust the pathname to reflect your actual installation directory.

To set the PATH on older versions of Microsoft Windows, 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-linux-gnu-g++ -v
```

Verify that the last line of the output contains: Sourcery G++ Lite 2011.03-53.

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-linux-gnu/man`.

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

```
> mips-linux-gnu-g++ -v
```

Verify that the last line of the output contains: `Sourcery G++ Lite 2011.03-53`.

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. In particular, the `mips-linux-gnu` directory located in the install directory will be removed entirely by the uninstaller.

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

You should use the provided uninstaller to remove a Sourcery G++ Lite installation originally created by the graphical installer. Start the graphical uninstaller by invoking the Uninstall executable located in your installation directory, or use the uninstall shortcut created during installation. 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 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 `Uninstall a program` (Vista and newer) or `Add or Remove Programs` (older versions of Windows) 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. 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 GNU/Linux

This chapter contains information about features of Sourcery G++ Lite that are specific to MIPS GNU/Linux 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 GNU/Linux, and tells you where you may find further information about these features.

Component	Version	Notes
GNU programming tools		
GNU Compiler Collection	4.5.2	Separate manual included.
GNU Binary Utilities	2.20.51	Includes assembler, linker, and other utilities. Separate manuals included.
Debugging support and simulators		
GNU Debugger	7.2.50	Separate manual included.
Sourcery G++ Debug Sprite for MIPS	2011.03-53	Provided for kernel debugging only. See Chapter 5, “Sourcery G++ Debug Sprite”.
GDB Server	N/A	Included with GDB. See Section 3.6, “Using GDB Server for Debugging”.
Target libraries		
GNU C Library	2.13	Separate manual included.
uClibc C Library	0.9.30	
Linux Kernel Headers	2.6.38	
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++ Lite for MIPS GNU/Linux includes the following library configuration.

MIPS32 revision 2 - Big-Endian, O32	
Command-line option(s):	default
Sysroot subdirectory:	`./`
Dynamic linker:	`lib/ld.so.1`

MIPS32 revision 2 - Little-Endian, O32	
Command-line option(s):	`-EL`
Sysroot subdirectory:	`el/`
Dynamic linker:	`lib/ld.so.1`

MIPS32 revision 2 - Big-Endian, Soft-Float, O32	
Command-line option(s):	`-msoft-float`
Sysroot subdirectory:	`soft-float/`
Dynamic linker:	`lib/ld.so.1`

MIPS32 revision 2 - Little-Endian, Soft-Float, O32	
Command-line option(s):	`-EL -msoft-float`
Sysroot subdirectory:	`soft-float/el/`
Dynamic linker:	`lib/ld.so.1`

MIPS32 revision 2 - Big-Endian, O32, microMIPS	
Command-line option(s):	`-mmicromips`
Sysroot subdirectory:	`micromips/`
Dynamic linker:	`lib/ld.so.1`

MIPS32 revision 2 - Little-Endian, O32, microMIPS	
Command-line option(s):	`-mmicromips -EL`
Sysroot subdirectory:	`micromips/el/`
Dynamic linker:	`lib/ld.so.1`

MIPS32 revision 2 - Big-Endian, Soft-Float, O32, microMIPS	
Command-line option(s):	`-mmicromips -msoft-float`
Sysroot subdirectory:	`micromips/soft-float/`
Dynamic linker:	`lib/ld.so.1`

MIPS32 revision 2 - Little-Endian, Soft-Float, O32, microMIPS	
Command-line option(s):	`-mmicromips -EL -msoft-float`
Sysroot subdirectory:	`micromips/soft-float/el/`
Dynamic linker:	`lib/ld.so.1`

MIPS32 revision 2 - uClibc, Big-Endian, O32	
Command-line option(s):	`-muclibc`
Sysroot subdirectory:	`uclibc/`
Dynamic linker:	`lib/ld-uClibc.so.0`

MIPS32 revision 2 - uClibc, Little-Endian, O32	
Command-line option(s):	`-muclibc -EL`
Sysroot subdirectory:	`uclibc/el/`
Dynamic linker:	`lib/ld-uClibc.so.0`

MIPS32 revision 2 - uClibc, Big-Endian, Soft-Float, O32	
Command-line option(s):	`-muclibc -msoft-float`
Sysroot subdirectory:	`uclibc/soft-float/`
Dynamic linker:	`lib/ld-uClibc.so.0`

MIPS32 revision 2 - uClibc, Little-Endian, Soft-Float, O32	
Command-line option(s):	`-muclibc -EL -msoft-float`
Sysroot subdirectory:	`uclibc/soft-float/el/`
Dynamic linker:	`lib/ld-uClibc.so.0`

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.

Each multilib corresponds to a *sysroot* directory which contains the files that should be installed on the target system. The sysroot contains the dynamic linker used to run your applications on the target as well as the libraries. Refer to Section 3.5, "Using Sourcery G++ Lite on GNU/Linux Targets" for instructions on how to install and use these support files on your target GNU/Linux system. You can find the sysroot directories provided with Sourcery G++ in the `mips-linux-gnu/libc` directory of your installation. In the tables below, the dynamic linker pathname is given relative to the corresponding sysroot.

3.3. Target Architectures

By default, Sourcery G++ Lite for MIPS GNU/Linux generates code for MIPS32r2 processors. If you wish to generate code for another MIPS processor, you must use an appropriate `-march` option when you build your application. Refer to the GCC manual for additional information about supported targets.

3.4. Target Kernel Requirements

The GNU C library supplied with this version of Sourcery G++ Lite requires that Linux kernel version 2.6.12 or later be installed on the target in order to run applications.

To support hardware watchpoints with `gdbserver`, Linux 2.6.28 or later is required.

3.5. Using Sourcery G++ Lite on GNU/Linux Targets

In order to run and debug programs produced by Sourcery G++ on a GNU/Linux target, you must install runtime support files on the target. You may also need to set appropriate build options so that your executables can find the correct dynamic linker and libraries at runtime.

The runtime support files, referred to as the *sysroot*, are found in the `mips-linux-gnu/libc` directory of your Sourcery G++ Lite installation. The sysroot consists of the contents of the `etc`, `lib`, `sbin`, and `usr` directories. There may be other directories in `mips-linux-gnu/libc` that contain additional sysroots customized for particular combinations of command-line compiler flags, or *multilibs*. Refer to Section 3.2, "Library Configurations" for a list of the included multilibs in this version of Sourcery G++ Lite, and the corresponding sysroot directory pathnames.

Note for Windows Host Users

The sysroots provided in Windows host packages for Sourcery G++ are not directly usable on the GNU/Linux target because of differences between the Windows and GNU/Linux file systems. Some files that are hard links, or copies, in the sysroot as installed on the

Windows file system should be symbolic links on the GNU/Linux target. Additionally, some files in the sysroot that should be marked executable on the GNU/Linux target are not marked executable on Windows. If you intend to use the sysroot provided with Sourcery G++ on a Windows host system as the basis for your GNU/Linux target filesystem, you must correct these issues after copying the sysroot to the target.

You have these choices for installing the sysroot on the target:

- You can install the files in the filesystem root on the target (that is, installing the files directly in `/etc/`, `/lib/`, and so on). All applications on the target then automatically use the Sourcery G++ libraries. This method is primarily useful when you are building a GNU/Linux root filesystem from scratch. If your target board already has a GNU/Linux filesystem installed, overwriting the existing C library files is not recommended, as this may break other applications on your system, or cause it to fail to boot.

- You can install the sysroot in an alternate location and build your application with the `-rpath` and `--dynamic-linker` linker options to specify the sysroot location.

- You can install the sysroot in an alternate location and explicitly invoke your application through the dynamic linker to specify the sysroot location. If you are just getting started with Sourcery G++ Lite, this may be the easiest way to get your application running, but this method does not support use of the debugger. In addition, this method only works with programs linked with the GNU C Library (glibc), not uClibc.

Setting the environment variable `LD_LIBRARY_PATH` on the target is not sufficient, since executables produced by Sourcery G++ depend on the Sourcery G++ dynamic linker included in the sysroot as well as the Sourcery G++ runtime libraries.

3.5.1. Installing the Sysroot

If you are modifying an existing system, rather than creating a new system from scratch, you should place the sysroot files in a new directory, rather than in the root directory of your target system.

If you choose to overwrite your existing C library, you may not be able to boot your system. You should back up your existing system before overwriting the C library and ensure that you can restore the backup even with your system offline.

The next step is to identify the correct sysroot subdirectory in the Sourcery G++ Lite install directory on your host system. The sysroot you copy to the target must be the one that corresponds to the linker options you are using to build your applications. The tables in Section 3.2, “Library Configurations” tell you which sysroot subdirectories correspond to which sets of command-line options. From the command line, you can identify the appropriate sysroot for your program by invoking the compiler with `-print-sysroot` added to your other build options. This causes GCC to print the host sysroot pathname and exit.

The mechanism you use for copying the sysroot to your target board depends on its hardware and software configuration. You may be able to use FTP or SSH with a server already running on your target. If your target board does not have networking configured, you may be able to copy files using an SD card or USB memory stick, or via a file transfer utility over a serial line. The instructions that come with your board may include specific suggestions.

When running Sourcery G++ on a GNU/Linux host, as an alternative to copying files to the target system, you may be able to NFS-mount the Sourcery G++ Lite installation directory from your host system on the target system. It is especially convenient for debugging if you can make the sysroot

pathname on the target system be identical to that on the GNU/Linux host system; refer to Section 3.6.3, “Setting the Sysroot in the Debugger” for further discussion of this issue.

Otherwise, you must copy files from the appropriate sysroot subdirectory in the `mips-linux-gnu/libc` directory of your Sourcery G++ Lite install to the target system. In many cases, you do not need to copy all of the files in the sysroot. For example, the `usr/include` subdirectory contains files that are only needed if you will actually be running the compiler on your target system. You do not need these files for non-native compilers. You also do not need any `.o` or `.a` files; these are used by the compiler when linking programs, but are not needed to run programs. You should definitely copy all `.so` files and the executable files in `usr/bin` and `sbin`.

3.5.2. Using Linker Options to Specify the Sysroot Location

If you have installed the sysroot on the target in a location other than the file system root, you can use the `-rpath` and `--dynamic-linker` linker options to specify the sysroot location.

If you are using Sourcery G++ from the command line, follow these steps:

1. First find the correct sysroot directory, dynamic linker, and library subdirectory for your selected multilib. Refer to Section 3.2, “Library Configurations”. In the following steps, *`sysroot`* is the absolute path to the sysroot directory on the target corresponding to your selected multilib. For the default multilib, the dynamic linker path relative to the sysroot is `lib/ld.so.1`, and the library subdirectory is `lib`. This is used in the example below.

2. When invoking `mips-linux-gnu-gcc` to link your executable, include the command-line options:

   ```
   -Wl,-rpath=sysroot/lib:sysroot/usr/lib \
   -Wl,--dynamic-linker=sysroot/lib/ld.so.1
   ```

   where *`sysroot`* is the absolute path to the sysroot directory on the target corresponding to your selected multilib.

3. Copy the executable to the target and execute it normally.

Note that if you specify an incorrect path for `--dynamic-linker`, the common failure mode seen when running your application on the target is similar to

```
> ./factorial
./factorial: No such file or directory
```

or

```
> ./factorial
./factorial: bad ELF interpreter: No such file or directory
```

This can be quite confusing since it appears from the error message as if it is the `./factorial` executable that is missing rather than the dynamic linker it references.

3.5.3. Specifying the Sysroot Location at Runtime

You can invoke the Sourcery G++ dynamic linker on the target to run your application without having to compile it with specific linker options. Note that this method of specifying the sysroot is specific to the GNU C Library (glibc) and does not work if you have linked your application with uClibc instead.

To do this, follow these steps:

1. Build your application on the host, without any additional linker options, and copy the executable to your target system.

2. Find the correct sysroot directory, dynamic linker, and library subdirectory for your selected multilib. Refer to Section 3.2, “Library Configurations”. In the following steps, *`sysroot`* is the absolute path to the sysroot directory on the target corresponding to your selected multilib. For the default multilib, the dynamic linker is `lib/ld.so.1`, and the library subdirectory is `lib`. This is used in the example below.

3. On the target system, invoke the dynamic linker with your executable as:

   ```
   > sysroot/lib/ld.so.1 \
     --library-path sysroot/lib:sysroot/usr/lib \
     /path/to/your-executable
   ```

   where *`sysroot`* is the absolute path to the sysroot directory on the target corresponding to your selected multilib.

   Invoking the linker in this manner requires that you provide either an absolute pathname to your executable, or a relative pathname prefixed with `./`. Specifying only the name of a file in the current directory does not work.

3.6. Using GDB Server for Debugging

The GDB server utility provided with Sourcery G++ Lite can be used to debug a GNU/Linux application. While Sourcery G++ runs on your host system, `gdbserver` and the target application run on your target system. Even though Sourcery G++ and your application run on different systems, the debugging experience when using `gdbserver` is very similar to debugging a native application.

3.6.1. Running GDB Server

The GDB server executables are included in the sysroot in ABI-specific subdirectories of *`sysroot`*`/usr`. Use the executable from the sysroot and library subdirectory that match your program. See Section 3.2, “Library Configurations” for details.

You must copy the sysroot to your target system as described in Section 3.5.1, “Installing the Sysroot”. You must also copy the executable you want to debug to your target system.

If you have installed the sysroot in the root directory of the filesystem on the target, you can invoke `gdbserver` as:

```
> gdbserver :10000 program arg1 arg2 ...
```

where *`program`* is the path to the program you want to debug and *`arg1 arg2 ...`* are the arguments you want to pass to it. The `:10000` argument indicates that `gdbserver` should listen for connections from GDB on port 10000. You can use a different port, if you prefer.

If you have installed the sysroot in an alternate directory, invoking `gdbserver` becomes more complicated. You must build your application using the link-time options to specify the location of the sysroot, as described in Section 3.5.2, “Using Linker Options to Specify the Sysroot Location”. You must also invoke `gdbserver` itself using the dynamic linker provided in the Sourcery G++ sysroot, as described in Section 3.5.3, “Specifying the Sysroot Location at Runtime”. In other words, the command to invoke `gdbserver` in this case would be similar to:

```
> sysroot/lib/ld.so.1 \
  --library-path sysroot/lib:sysroot/usr/lib \
  sysroot/usr/lib/bin/gdbserver :10000 program arg1 arg2 ...
```

3.6.2. Connecting to GDB Server from the Debugger

You can connect to GDB server by using the following command from within GDB:

```
(gdb) target remote target:10000
```

where *target* is the host name or IP address of your target system.

When your program exits, `gdbserver` exits too. If you want to debug the program again, you must restart `gdbserver` on the target. Then, in GDB, reissue the `target` command shown above.

3.6.3. Setting the Sysroot in the Debugger

In order to debug shared libraries, GDB needs to map the pathnames of shared libraries on the target to the pathnames of equivalent files on the host system. Debugging of multi-threaded applications also depends on correctly locating copies of the libraries provided in the sysroot on the host system.

In some situations, the target pathnames are valid on the host system. Otherwise, you must tell GDB how to map target pathnames onto the equivalent host pathnames.

In the general case, there are two GDB commands required to set up the mapping:

```
(gdb) set sysroot-on-target target-pathname
(gdb) set sysroot host-pathname
```

This causes GDB to replace all instances of the *target-pathname* prefix in shared library pathnames reported by the target with *host-pathname* to get the location of the equivalent library on the host.

If you have installed the sysroot in the root filesystem on the target, you can omit the `set sysroot-on-target` command, and use only `set sysroot` to specify the location on the host system.

Refer to Section 3.5.1, “Installing the Sysroot” for more information about installing the sysroot on the target. Note that if you have installed a stripped copy of the provided libraries on the target, you should give GDB the location of an unstripped copy on the host.

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-linux-gnu, as indicated by the `mips-linux-gnu` 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-linux-gnu-gcc -o factorial main.c
```

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

4.2. Running Applications on the Target System

You may need to install the Sourcery G++ runtime libraries and dynamic linker on the target system before you can run your application. Refer to Chapter 3, “Sourcery G++ Lite for MIPS GNU/Linux” for specific instructions.

To run your program on a GNU/Linux target system, use the command:

```
> factorial
```

You should see:

```
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
```

4.3. 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.

When starting GDB, give it the pathname to the program you want to debug as a command-line argument. For example, if you have built the factorial program as described in Section 4.1, “Building an Application”, enter:

```
> mips-linux-gnu-gdb factorial
```

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.

4.3.1. Connecting to the Sourcery G++ Debug Sprite

The Sourcery G++ Debug Sprite is a program that runs on the host system to support hardware debugging devices. You can use the Debug Sprite to run and debug programs on a target board without an operating system, or to debug an operating system kernel. See Chapter 5, “Sourcery G++ Debug Sprite” for detailed information about the supported devices.

You can start the Sprite directly from within GDB:

```
(gdb) target remote | mips-linux-gnu-sprite arguments
```

Refer to Section 5.2, “Invoking Sourcery G++ Debug Sprite” for a full description of the Sprite arguments.

4.3.2. Connecting to an External GDB Server

Sourcery G++ Lite includes a program called `gdbserver` that can be used to debug a program running on a remote MIPS GNU/Linux target. Follow the instructions in Chapter 3, “Sourcery G++ Lite for MIPS GNU/Linux” to install and run `gdbserver` on your target system.

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.

Chapter 5
Sourcery G++ Debug Sprite

This chapter describes the use of the Sourcery G++ Debug Sprite for remote debugging. The Sprite is provided for debugging of the Linux kernel on the target board. This chapter includes information about the debugging devices and boards supported by the Sprite for MIPS GNU/Linux.

Sourcery G++ Lite contains the Sourcery G++ Debug Sprite for MIPS GNU/Linux. This Sprite is provided to allow debugging of programs running on a bare board. You can use the Sprite to debug a program when there is no operating system on the board, or for debugging the operating system itself. If the board is running an operating system, and you wish to debug a program running on that OS, you should use the facilities provided by the OS itself (for instance, using `gdbserver`).

The Sprite acts as an interface between GDB and external debug devices and libraries. Refer to Section 5.2, “Invoking Sourcery G++ Debug Sprite” for information about the specific devices supported by this version of Sourcery G++ Lite.

Note for Linux users

The Debug Sprite provided with Sourcery G++ Lite allows remote debugging of the Linux kernel running on the target. For remote debugging of application programs, you should use `gdbserver` instead. See Chapter 3, “Sourcery G++ Lite for MIPS GNU/Linux” for details about how to install and run `gdbserver` on the target.

Important

The Sourcery G++ Debug Sprite is not part of the GNU Debugger and is not free or open-source software. You may use the Sourcery G++ Debug Sprite only with the GNU Debugger. You may not distribute the Sourcery G++ Debug Sprite to any third party.

5.1. Probing for Debug Devices

Before running the Sourcery G++ Debug Sprite for the first time, or when attaching new debug devices to your host system, it is helpful to verify that the Sourcery G++ Debug Sprite recognizes your debug hardware. From the command line, invoke the Sprite with the `-i` option:

```
> mips-linux-gnu-sprite -i
```

This prints out a list of supported device types. For devices that can be autodetected, it additionally probes for and prints out a list of attached devices. For instance:

```
Sourcery CodeBench Debug Sprite for MIPS (Sourcery G++ Lite \
2011.03-53)
mdi: [lib=<file>&cfg=<file>&rst=<n>] MDI device
  mdi:/23/1 - 24KE     (Instruction)/24KE LE
  mdi:/23/2 - 24KE     (Instruction)/24KE BE
  mdi:/24/1 - 24KE     (Cycle)/24KE LE
  mdi:/24/2 - 24KE     (Cycle)/24KE BE
  mdi:/$Target/$Device - Generic MDI target/device
```

This shows that MDI (Microprocessor Debug Interface) devices are supported. Four MIPSsim devices have been autodetected. Note that additional configuration steps for the MDI library are required to allow the Sprite to autodetect devices; see Section 5.4, “MDI Devices”.

5.2. Invoking Sourcery G++ Debug Sprite

The Debug Sprite is invoked as follows:

```
> mips-linux-gnu-sprite [options] device-url board-file
```

The *device-url* specifies the debug device to use to communicate with the board. It follows the standard format:

```
scheme:scheme-specific-part[?device-options]
```

Most device URL schemes also follow the regular format:

```
scheme:[//hostname:[port]]/path[?device-options]
```

The meanings of *hostname*, *port*, *path* and *device-options* parts depend on the *scheme* and are described below. The following schemes are supported in Sourcery G++ Lite for MIPS GNU/Linux:

`mdi` Use a Microprocessor Debug Interface (MDI) debugging device. Refer to Section 5.4, “MDI Devices”.

The optional `?`*device-options* portion is allowed in all schemes. These allow additional device-specific options of the form *name=value*. Multiple options are concatenated using `&`.

The *board-file* specifies an XML file that describes how to initialize the target board, as well as other properties of the board used by the debugger. If *board-file* refers to a file (via a relative or absolute pathname), it is read. Otherwise, *board-file* can be a board name, and the toolchain's board directory is searched for a matching file. See Section 5.6, “Supported Board Files” for the list of supported boards, or invoke the Sprite with the `-b` option to list the available board files. You can also write a custom board file; see Section 5.7, “Board File Syntax” for more information about the file format.

Both the *device-url* and *board-file* command-line arguments are required to correctly connect the Sprite to a target board.

5.3. Sourcery G++ Debug Sprite Options

The following command-line options are supported by the Sourcery G++ Debug Sprite:

`-b` Print a list of *board-file* files in the board config directory.

`-h` Print a list of options and their meanings. A list of *device-url* syntaxes is also shown.

`-i` Print a list of the accessible devices. If a *device-url* is also specified, only devices for that device type are scanned. Each supported device type is listed along with the options that can be appended to the *device-url*. For each discovered device, the *device-url* is printed along with a description of that device.

`-l` *[host]:port* Specify the host address and port number to listen for a GDB connection. If this option is not given, the Debug Sprite communicates with GDB using stdin and stdout. If you start the Sprite from within GDB using the `target remote | mips-linux-gnu-sprite ...` command, you do not need this option.

`-m` Listen for multiple sequential connections. Normally the Debug Sprite terminates after the first connection from GDB terminates. This option instead makes it listen for a subsequent connection. To terminate the Sprite, open a connection and send the string `END\n`.

`-q` Do not print any messages.

`-v` Print additional messages.

If any of `-b`, `-i` or `-h` are given, the Debug Sprite terminates after providing the information rather than waiting for a debugger connection.

5.4. MDI Devices

The Sourcery G++ Debug Sprite for MIPS supports MDI (Microprocessor Debug Interface) devices. Each MDI device is identified by a target number and device number; these form the *`path`* part of the device URL, and the *`hostname`* and *`port`* must be empty or omitted. Thus, the *`device-url`* has the form:

```
mdi:///targetnum/devicenum[?device-options]
```

You can also use the environment variables `GDBMDITARGET` and `GDBMDIDEVICE` to provide defaults for the *`targetnum`* and *`devicenum`*.

The following *`device-options`* are permitted:

`lib=`*`filename`*
: This option specifies the MDI library to load. It is equivalent to setting the `GDBMDILIB` environment variable.

`cfg=`*`filename`*
: Some MDI target libraries, such as MIPSsim, require a configuration file. (This is distinct from the Sprite's own *`board-file`*.) You can use this option to specify the file. It is equivalent to setting the `GDBMIPSSIMCONFIG` environment variable.

`rst=`*`seconds`*
: This option can be used to specify a delay after the target is reset by the Sprite. If the value of *`seconds`* is greater than zero, then execution is resumed for the specified number of seconds; this can be used to allow power-on firmware to initialize the memory controller and peripherals. Then the target is halted again and queried for configuration.

  If the value of *`seconds`* is `-1`, then the target is queried immediately without reset. This is the same effect as passing the `-a` command-line option to the Sprite, which allows the Sprite to attach to a running program.

  This option is equivalent to setting the `GDBMDICONNRST` environment variable. If neither the option nor the environment variable are provided, the default is to reset the target and query it immediately unless the `-a` option is specified.

`group=/`*`targetn/devicen`*
: This option may be specified multiple times and is cumulative. Each of the specified devices is opened and queried and they are all treated as threads of execution, subject to being enabled or active; if a device is disabled or has no active thread contexts associated with it, it is not visible to GDB but is still under control of the Sprite in case its state changes. This option cannot be used in combination with the `team=` option.

`team=`*`/targetn/devicen`* This option may be specified multiple times and is cumulative. The specified devices are not opened, but are associated with the base device by means of the MDI team mechanism for the purpose of synchronization. The specified devices may still be opened and controlled by another debugger (such as another instance of the Debug Sprite) independently. This option cannot be used in combination with the `group=` option.

Before you can connect to a target using the MDI API, you must tell the Debug Sprite which shared library or DLL to load for your simulator or device. 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 the Debug Sprite, or use the `lib=` device option to specify the library to load.

Similarly, the `-i` command-line option can only probe for devices if you have set the `PATH` or `LD_LIBRARY_PATH` environment variable appropriately, and specify an MDI library using either the `GDBMDILIB` environment variable or the `lib=` device option. Otherwise, it reports only the generic *`device-url`* syntax.

For example, to use an FS2 probe on a Windows host to debug a MIPS Malta board, first add the directory containing the MDI DLLs to your `PATH`. Then you can invoke the Sprite from GDB using a command line similar to:

```
(gdb) target remote | mips-linux-gnu-sprite \
'mdi:/2/2?lib=jnetfs2mdilib.dll&rst=7' malta
```

The quotes are required to prevent special characters in the *`device-url`* from being interpreted by the shell.

In the above command, the `rst=7` option provides for a sufficient delay for the board's reset code to execute on connection. Since this takes several seconds, GDB may time out waiting for the Sprite to respond. You can prevent this by issuing this command before you connect to the Sprite:

```
(gdb) set remotetimeout 10
```

To use the Sprite with MIPSsim, a configuration file is required. The configuration files provided with the MIPSsim distribution are intended for use with standalone execution from the command line, rather than running the program from the debugger. So, make a copy and comment out the `APP_FILE` setting. It is also recommended that you comment out `TRACE_FILE` as well, since the trace files can be very large.

To connect to MIPSsim using the Sprite on a Linux host, first set your `LD_LIBRARY_PATH` and `GDBMDILIB` as described above. You can run the Sprite from the shell to probe for devices to verify that your setup is correct:

```
> mips-linux-gnu-sprite -i
```

Then, from GDB, use a command similar to:

```
(gdb) target remote | mips-linux-gnu-sprite \
'mdi:/23/2?cfg=24KE.cfg&rst=-1' mipssim
```

Fill in your target and device numbers as reported by the probe output, and the full pathname to your configuration file. The `rst=-1` option is required, as MIPSsim does not support reset.

This section describes only 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 license in addition to the steps given here.

5.5. Debugging a Remote Board

You can run the Sourcery G++ Debug Sprite on a different machine from the one on which GDB is running. For example, if your board is connected to a machine in your lab, you can run the debugger on your laptop and connect to the remote board. The Sourcery G++ Debug Sprite must run on the machine that is connected to the target board. You must have Sourcery G++ installed on both machines.

To use this mode, you must start the Sprite with the `-l` option and specify the port on which you want it to listen. For example:

```
> mips-linux-gnu-sprite -l :10000 device-url board-file
```

starts the Sprite listening on port 10000.

When running GDB from the command line, use the following command to connect GDB to the remote Sprite:

```
(gdb) target remote host:10000
```

where *`host`* is the name of the remote machine. After this, debugging is just as if you are debugging a target board connected to your host machine.

For more detailed instructions on using the Sourcery G++ Debug Sprite in this way, please refer to the Sourcery G++ Knowledge Base[1].

5.6. Supported Board Files

The Sourcery G++ Debug Sprite for MIPS GNU/Linux includes support for the following target boards. Specify the appropriate *`board-file`* as an argument when invoking the Sprite from the command line.

Board	**Config**
MIPS Malta	`malta`
MIPS SEAD-3 LX110	`sead3-lx110`
MIPS SEAD-3 LX50	`sead3-lx50`
MIPSsim	`mipssim`

5.7. Board File Syntax

The *`board-file`* can be a user-written XML file to describe a non-standard board. The Sourcery G++ Debug Sprite searches for board files in the `mips-linux-gnu/lib/boards` directory in the installation. Refer to the files in that directory for examples.

The file's DTD is:

[1] https://support.codesourcery.com/GNUToolchain/kbentry132

```
<!-- Board description files

     Copyright (c) 2007-2009 CodeSourcery, Inc.

     THIS FILE CONTAINS PROPRIETARY, CONFIDENTIAL, AND TRADE
     SECRET INFORMATION OF CODESOURCERY AND/OR ITS LICENSORS.

     You may not use or distribute this file without the express
     written permission of CodeSourcery or its authorized
     distributor.  This file is licensed only for use with
     Sourcery G++.  No other use is permitted.
     -->

<!ELEMENT board
 (properties?, feature?, initialize?, memory-map?)>

<!ELEMENT properties
 (description?, property*)>

<!ELEMENT initialize
 (write-register | write-memory | delay
  | wait-until-memory-equal | wait-until-memory-not-equal)* >
<!ELEMENT write-register EMPTY>
<!ATTLIST write-register
          address CDATA   #REQUIRED
                        value   CDATA   #REQUIRED
                        bits    CDATA   #IMPLIED>
<!ELEMENT write-memory EMPTY>
<!ATTLIST write-memory
          address CDATA   #REQUIRED
                        value   CDATA   #REQUIRED
                        bits    CDATA   #IMPLIED>
<!ELEMENT delay EMPTY>
<!ATTLIST delay
          time CDATA   #REQUIRED>
<!ELEMENT wait-until-memory-equal EMPTY>
<!ATTLIST wait-until-memory-equal
          address CDATA   #REQUIRED
                        value   CDATA   #REQUIRED
                        timeout CDATA   #IMPLIED
                        bits    CDATA   #IMPLIED>
<!ELEMENT wait-until-memory-not-equal EMPTY>
<!ATTLIST wait-until-memory-not-equal
          address CDATA   #REQUIRED
                        value   CDATA   #REQUIRED
                        timeout CDATA   #IMPLIED
                        bits    CDATA   #IMPLIED>

<!ELEMENT memory-map (memory-device)*>
<!ELEMENT memory-device (property*, description?, sectors*)>
<!ATTLIST memory-device
                        address CDATA   #REQUIRED
          size    CDATA   #REQUIRED
          type    CDATA   #REQUIRED
```

```
                        device  CDATA   #IMPLIED>

<!ELEMENT description (#PCDATA)>
<!ELEMENT property (#PCDATA)>
<!ATTLIST property name CDATA #REQUIRED>
<!ELEMENT sectors EMPTY>
<!ATTLIST sectors
 size CDATA #REQUIRED
 count CDATA #REQUIRED>

<!ENTITY % gdbtarget SYSTEM "gdb-target.dtd">
%gdbtarget;
```

All values can be provided in decimal, hex (with a `0x` prefix) or octal (with a `0` prefix). Addresses and memory sizes can use a `K`, `KB`, `M`, `MB`, `G` or `GB` suffix to denote a unit of memory. Times must use a `ms` or `us` suffix.

The following elements are available:

`<board>` — This top-level element encapsulates the entire description of the board. It can contain `<properties>`, `<feature>`, `<initialize>` and `<memory-map>` elements.

`<properties>` — The `<properties>` element specifies specific properties of the target system. This element can occur at most once. It can contain a `<description>` element.

`<initialize>` — The `<initialize>` element defines an initialization sequence for the board, which the Sprite performs before downloading a program. It can contain `<write-register>`, `<write-memory>` and `<delay>` elements.

`<feature>` — This element is used to inform GDB about additional registers and peripherals available on the board. It is passed directly to GDB; see the GDB manual for further details.

`<memory-map>` — This element describes the memory map of the target board. It is used by GDB to determine where software breakpoints may be used and when flash programming sequences must be used. This element can occur at most once. It can contain `<memory-device>` elements.

`<memory-device>` — This element specifies a region of memory. It has four attributes: `address`, `size`, `type` and `device`. The `address` and `size` attributes specify the location of the memory device. The `type` attribute specifies that device as `ram`, `rom` or `flash`. The `device` attribute is required for `flash` regions; it specifies the flash device type. The `<memory-device>` element can contain a `<description>` element.

`<write-register>` — This element writes a value to a control register. It has three attributes: `address`, `value` and `bits`. The `bits` attribute, specifying the bit width of the write operation, is optional; it defaults to 32.

`<write-memory>` — This element writes a value to a memory location. It has three attributes: `address`, `value` and `bits`. The `bits` attribute is optional and defaults

	to 32. Bit widths of 8, 16 and 32 bits are supported. The address written to must be naturally aligned for the size of the write being done.
`<delay>`	This element introduces a delay. It has one attribute, `time`, which specifies the number of milliseconds, or microseconds to delay by.
`<description>`	This element encapsulates a human-readable description of its enclosing element.
`<property>`	The `<property>` element allows additional name/value pairs to be specified. The property name is specified in a `name` attribute. The property value is the body of the `<property>` element.

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++ Knowledge Base

The Sourcery G++ Knowledge Base is available to registered users at the Sourcery G++ Portal[1]. 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.

6.2. Example Programs

Sourcery G++ Lite includes some bundled example programs. You can find the source code for these examples in the `share/sourceryg++-mips-linux-gnu-examples` directory of your Sourcery G++ installation.

The subdirectories contain a number of small, target-independent test programs. You may find these programs useful as self-contained test cases when experimenting with configuring the correct compiler and debugger settings for your target, or when learning how to use the debugger or other features of the Sourcery G++ toolchain.

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-linux-gnu/` 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-linux-gnu/man/man1
```

Then you can invoke `man` as:

```
> man ./mips-linux-gnu-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.

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.

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

Appendix A
Sourcery G++ Lite Release Notes

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

A.1. Changes in Sourcery G++ Lite for MIPS GNU/Linux

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

A.1.1. Changes in Sourcery G++ Lite 2011.03-53

Improved support for 32-bit symbols when using 64-bit ABI. Sourcery G++ Lite has been improved to handle Procedure Linkage Table (PLT) entries for 32-bit function symbols when compiling for N64 ABI. Effectively, the toolchain now supports building applications with `-mabi=64 -msym32 -mplt` compiler options, which may improve performance of certain applications.

Interprocedural stack optimization. The compiler has a new experimental optimization that generates code that claims less stack space. To switch this optimization on, use `-mframe-header-opt`.

MIPS32r2 floating-point multiply-accumulate restrictions lifted. Following clarifications to the MIPS architecture specification, GCC has been updated to allow floating-point multiply-accumulate instructions on MIPS32r2 processors in all configurations.

Interprocedural register optimization. The compiler has a new experimental optimization that generates better code for functions that only call functions in the same object. To switch this optimization on, use `-fuse-caller-save`.

Incorrect C++ warning fixed. A bug in GCC has been fixed that caused spurious warnings about lambda expressions in C++ code that does not use them.

C++ constructor bug fix. A compiler bug has been fixed that caused incorrect code for C++ constructors for some class hierarchies that use virtual inheritance and include empty classes. At runtime, the incorrect constructors resulted in memory corruption or other errors.

microMIPS branches. A compiler bug that caused sub-optimal branch instructions to be generated in microMIPS mode has been fixed.

Addr2line bug fix. A bug has been fixed that caused Addr2line to print `??:0` for any address instead of file and line number.

24K Errata. The `-mfix-24k` assembler option has been extended to work around Errata E48: Lost Data on Stores During Refill. In addition, GCC now accepts this command-line option and passes it through to the assembler.

EGLIBC version 2.13. Sourcery G++ Lite for MIPS GNU/Linux now includes EGLIBC version 2.13 library which is based on GNU C Library version 2.13. For more information about changes, see `http://www.eglibc.org/news#eglibc_2_13`.

A.1.2. Changes in Sourcery G++ Lite 2011.03-11

Compiler optimization improvements. The compiler has been enhanced with a number of optimization improvements, including:

- Smaller and faster code for compound conditionals.
- Improved filling of branch delay slots.

- Removal of superfluous sign and zero extensions.

GCC version 4.5.2. Sourcery G++ Lite for MIPS GNU/Linux is now based on GCC version 4.5.2.

New `-fstrict-volatile-bitfields` option. The compiler has a new option, `-fstrict-volatile-bitfields`, which forces access to a volatile structure member using the width that conforms to its type. Refer to the GCC manual for details.

GCC code generation bug for casts to `volatile` types. A compiler bug has been fixed that sometimes caused incorrect code for references to pointers to types with `volatile` casts.

Improvements to synchronization primitives. GCC's `__sync` built-in functions have been improved to make better use of MIPS atomic instructions. In addition, some of GLIBC's atomic macros have been improved to make use of the `__sync` built-in functions.

Incorrect optimization fix. An optimizer bug that in rare cases caused incorrect code to be generated for complex AND and OR expressions containing redundant subexpressions has been fixed.

GCC microMIPS code size improvement. GCC has been improved to generate smaller code around function calls.

GCC bug where accesses to volatile structure fields are optimized away. A bug has been fixed where accesses to volatile fields of a structure were sometimes incorrectly optimized away if the structure instance was defined as non-volatile.

Improved MIPS16 and microMIPS library call support. The toolchain now supports the use of MIPS16 and microMIPS code in the Procedure Linkage Table (PLT). The table is used to make shared library function calls from programs. The new feature allows smaller and faster code to be generated.

Linker debug information fix. A bug in linker processing of debug information has been fixed. The bug sometimes prevented the Sourcery G++ debugger from displaying source code if the executable was linked with the `--gc-sections` option.

microMIPS linker code size optimization bug fix. A bug has been fixed that sometimes caused the linker to generate incorrect code when the `--relax` option is used to enable certain code size optimizations.

MCU ASE instructions assembler bug fixes. Bugs have been fixed that caused the assembler to sometimes generate incorrect code for the MCU ASE `ACLR`, `ASET` and `IRET` instructions.

`popen` bug fix. GLIBC's `popen` function no longer causes a deadlock situation when invoked from more than one thread.

`strstr` and `strcasestr` bug fixes. A problem has been fixed that caused GLIBC's `strstr` and `strcasestr` functions to return wrong results on certain inputs.

Linux kernel headers update. Linux kernel header files have been updated to version 2.6.38.

`gdbserver` support for watchpoints. `gdbserver` now supports hardware watchpoints on MIPS GNU/Linux targets. The Linux kernel version 2.6.28 or later is required.

GDB microMIPS disassembler bug fix. A bug in GDB has been fixed that caused some microMIPS instructions to disassemble incorrectly.

Improved GDB startup times when debugging remote targets . GDB has been enhanced to reduce the startup times when working with remote targets via GDBServer, especially when the target uses a large number of shared libraries.

A.1.3. Changes in Sourcery G++ Lite 2010.09-24

Changes to Sourcery G++ version numbering. Sourcery G++ product and Lite toolchains now uniformly use a version numbering scheme of the form 2011.03-53. The major and minor parts of the version number, in this case 2011.03, identify the release branch, while the final component is a build number within the branch. There are also new preprocessor macros defined by the compiler for the version number components so that you may conditionalize code for Sourcery G++ or particular Sourcery G++ versions. Details are available in the Sourcery G++ Knowledge Base[1].

Alignment attributes. A bug has been fixed that caused the compiler to ignore alignment attributes of C++ static member variables where the attribute was present on the definition, but not the declaration.

Compiler optimization improvements. The compiler has been enhanced with a number of optimization improvements, including:

- More efficient assignment for structures containing bitfields.
- Better code for initializing C++ arrays with explicit element initializers.
- Improved logic for eliminating/combining redundant comparisons in code with nested conditionals.
- Better selection of loop variables, resulting in fewer temporaries and more efficient register usage.
- Better code when constant addresses are used as arguments to inline assembly statements.
- Better code for copying small constant strings.

GCC version 4.5.1. Sourcery G++ Lite for MIPS GNU/Linux is now based on GCC version 4.5.1. For more information about changes from GCC version 4.4 that was included in previous releases, see `http://gcc.gnu.org/gcc-4.5/changes.html`.

C++ locale support. The C++ standard library now includes locale support. This feature is not enabled for uClibc multilibs.

`COP2` instruction bug fix. A bug in handling of the `COP2` instruction's argument has been fixed in the assembler and disassembler.

Archiver bug fix. A bug has been fixed in the `ar` utility, which sometimes caused it to produce unrecognizable 64-bit files. The bug also caused similar problems in the `strip` and `objcopy` utilities when processing 64-bit archives.

Linker bug fix. A bug in the linker's handling of some multiply-defined symbols from shared libraries has been fixed.

Fix for incorrect MIPS16 and microMIPS relocations. An assembler bug has been fixed that caused incorrect relocation information to be produced for MIPS16 and microMIPS code, diagnosed by a `Can't find matching LO16 reloc` linker warning.

[1] https://support.codesourcery.com/GNUToolchain/kbentry1

Additional validation in the assembler. The assembler now diagnoses an error, instead of producing an invalid object file, when directives such as `.hidden` are missing operands.

Binutils update. The binutils package has been updated to version 2.20.51.20100809 from the FSF trunk. This update includes numerous bug fixes.

`memcpy` and `memmove` optimizations. The `memcpy` and `memmove` functions have been tuned for MIPS targets and to better handle unaligned inputs.

More efficient process creation functions. The `system` and `popen` functions provided by GLIBC have been improved to require less memory when memory overcommit is disabled in the Linux kernel.

Optimized string and memory functions. The performance of GLIBC's string and memory functions, including `strstr` and `memmem`, have been significantly improved for large inputs.

Linux kernel headers update. Linux kernel header files have been updated to version 2.6.35.2.

GDB update. The included version of GDB has been updated to 7.2.50.20100908. This update adds numerous bug fixes and new features, including improved C++ language support, a new command to save breakpoints to a file, a new convenience variable `$_thread` that holds the number of the current thread, among many other improvements.

GDB crash fix. A bug has been fixed that caused GDB to crash on launch if the environment variable `CYGPATH` is set to a program that does not exist or cannot be executed.

A.1.4. Changes in Older Releases

For information about changes in older releases of Sourcery G++ Lite for MIPS GNU/Linux, 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 GNU/Linux 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 http://www.gnu.org/licenses/gpl.html
GNU Binary Utilities	GNU General Public License 3.0 http://www.gnu.org/licenses/gpl.html
GNU Debugger	GNU General Public License 3.0 http://www.gnu.org/licenses/gpl.html
Sourcery G++ Debug Sprite for MIPS	CodeSourcery License
GNU C Library	GNU Lesser General Public License 2.1 http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html
uClibc C Library	GNU Lesser General Public License 2.1 http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html
Linux Kernel Headers	GNU General Public License 2.0 http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
GNU Make	GNU General Public License 2.0 http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
GNU Core Utilities	GNU General Public License 2.0 http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

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-linux-gnu-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.

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, the CodeSourcery C Library (CSLIBC), 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, CSLIBC 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, and, for components subject to the Eclipse Public License, contains information on how to obtain the source code.

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.

B.3. Attribution

This version of Sourcery G++ Lite may include code based on work under the following copyright and permission notices:

B.3.1. Android Open Source Project

```
/*
 * Copyright (C) 2008 The Android Open Source Project
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *  * Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  * Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
```

```
* COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
* BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
* OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
* AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*/
```