

Sourcery G++

ARM uClinux

Sourcery G++ 4.4-61

Getting Started



Sourcery G++: ARM uClinux: Sourcery G++ 4.4-61: Getting Started

CodeSourcery, Inc.

Copyright © 2005, 2006, 2007, 2008, 2009 CodeSourcery, Inc.

All rights reserved.

Abstract

This guide explains how to install and build applications with Sourcery G++, CodeSourcery's customized, validated, and supported version of the GNU Toolchain. Sourcery G++ 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 both the IDE and from the command line.

Table of Contents

Preface	v
1. Intended Audience	vi
2. Organization	vi
3. Typographical Conventions	vii
1. Quick Start	1
1.1. Installation and Set-Up	2
1.2. Configuring Sourcery G++ for the Target System	2
1.3. Building Your Program	2
1.4. Running and Debugging Your Program	3
2. Installation and Configuration	4
2.1. Terminology	5
2.2. System Requirements	5
2.3. Registering with the Sourcery G++ Portal	6
2.4. Downloading an Installer	6
2.5. Installing Sourcery G++	7
2.6. Installing Sourcery G++ Updates	10
2.7. Setting up the Environment	10
2.8. License Keys	12
2.9. Installing Add-Ons	16
2.10. Uninstalling Sourcery G++	18
3. Sourcery G++ for ARM uClinux	20
3.1. Included Components and Features	21
3.2. Library Configurations	21
3.3. Using VFP Floating Point	22
3.4. ABI Compatibility	24
3.5. Building uClinux Applications	24
3.6. GDB Server	24
4. Using the Sourcery G++ IDE	26
4.1. Overview	27
4.2. Building Applications	27
4.3. Debugging Applications	34
4.4. Advanced IDE Features	46
5. Using Sourcery G++ from the Command Line	56
5.1. Building an Application	57
5.2. Running Applications on the Target System	57
5.3. Running Applications from GDB	58
6. Sourcery G++ Debug Sprite	59
6.1. Probing for Debug Devices	60
6.2. Invoking Sourcery G++ Debug Sprite	60
6.3. Sourcery G++ Debug Sprite Options	61
6.4. Remote Debug Interface Devices	62
6.5. Actel FlashPro Devices	62
6.6. Keil ULINK2 Devices	63
6.7. SEGGER J-Link Devices	65
6.8. Debugging a Remote Board	65
6.9. Supported Board Files	66
6.10. Board File Syntax	66
7. Next Steps with Sourcery G++	70
7.1. Sourcery G++ Subscriptions	71
7.2. Sourcery G++ Knowledge Base	71
7.3. Manuals for GNU Toolchain Components	71

7.4. Help for the Sourcery G++ IDE	72
A. Sourcery G++ Release Notes	74
A.1. Changes in Sourcery G++ for ARM uClinux	75
B. Sourcery G++ Licenses	93
B.1. Licenses for Sourcery G++ Components	94
B.2. Sourcery G++ Software License Agreement	95

Preface

This preface introduces the Sourcery G++ 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++. This guide provides a step-by-step guide to installing Sourcery G++ 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++ 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++. 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++ for ARM uClinux”	This chapter contains information about using Sourcery G++ that is specific to ARM uClinux targets. You should read this chapter to learn how to best use Sourcery G++ on your target system.
Chapter 4, “Using the Sourcery G++ IDE”	This chapter explains how to use the Sourcery G++ IDE, which is based on Eclipse. The IDE provides a fully-integrated programming environment that allows you to edit, build, and debug programs.
Chapter 5, “Using Sourcery G++ from the Command Line”	This chapter explains how to build applications with Sourcery G++ 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 6, “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 or uClinux kernel on the target board. This chapter includes information about the debugging devices and boards supported by the Sprite for ARM uClinux.
Chapter 7, “Next Steps with Sourcery G++”	This chapter describes where you can find additional documentation and information about using Sourcery G++ 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++ Release Notes”	This appendix contains information about changes in this release of Sourcery G++ for ARM uClinux. You should read through these notes to learn about new features and bug fixes.
Appendix B, “Sourcery G++ Licenses”	This appendix provides information about the software licenses that apply to Sourcery G++. Read this appendix to

understand your legal rights and obligations as a user of Sourcery G++.

3. Typographical Conventions

The following typographical conventions are used in this guide:

<code>> command arg ...</code>	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.
<i>literal</i>	Text provided to or received from a computer program.
<i>placeholder</i>	Text that should be replaced with an appropriate value when typing a command.
<code>\</code>	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++ for the first time. You may use this chapter as an abbreviated guide to the rest of this manual.

Sourcery G++ for ARM uClinux is intended for developers working on embedded uClinux applications. It may also be used for uClinux kernel development and debugging, or to build a uClinux distribution.

Follow the steps given in this chapter to install Sourcery G++ 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++ in Chapter 3, “Sourcery G++ for ARM uClinux”.

1.1. Installation and Set-Up

Register with the Sourcery G++ Support Portal. You must register with the Sourcery G++ Portal¹ before you can use this subscription version of Sourcery G++. Free evaluations are also available from the Portal. Registering with the Portal also gives you access to the latest software updates, the Sourcery G++ Knowledge Base, and support for subscription customers.

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

Install a license key. License keys are managed from the Sourcery G++ IDE. When you start the IDE for the first time, it pops up a dialog box to guide you through the steps to download and/or install your license key. For detailed help, refer to Section 2.8, “License Keys”.

1.2. Configuring Sourcery G++ for the Target System

Identify your target libraries. Sourcery G++ 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. You will need to identify the multilib appropriate for your target in order to find the correct **gdbserver** executable to use for debugging your applications, as described in Section 3.6, “GDB Server”.

1.3. Building Your Program

Build your program using the Sourcery G++ IDE. In the Sourcery G++ IDE, create a new managed build C project. Use the dialog in the New Project Wizard to configure the target properties for building your project. Using the editor in the IDE, create a file containing a simple test program; save the file and build the project. For a tutorial introduction to the Sourcery G++ IDE that covers these steps, refer to Section 4.2, “Building Applications”.

¹ <https://support.codesourcery.com/GNUToolchain/>

Build your program with Sourcery G++ command-line tools. If you prefer, you can build your program from the command line instead of using the IDE. Create a simple test program, and follow the directions in Chapter 5, “Using Sourcery G++ from the Command Line” to compile and link it using Sourcery G++.

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. For general information and a tutorial on using the Sourcery G++ IDE for debugging, see Section 4.3, “Debugging Applications”.

Run your program on the target system. Copy your program to the target system and run it from the command line.

Debug your program on the target using GDB server. You can debug a program on a remote ARM uClinux target using GDB server. Copy your program to the target system. Follow the instructions in Section 3.6, “GDB Server” to install and run **gdbserver** on your target system. Then, you can connect to GDB server from the debugger running on your host system. In the Sourcery G++ IDE, use the `External Server` debugger, as described in Section 4.3.2.1, “Sourcery G++ External Server”. Refer to Section 5.3, “Running Applications from GDB” for instructions on connecting to the target from command-line GDB.

Chapter 2

Installation and Configuration

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

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

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 `arm-uclinuxeabi`.

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

2.2. System Requirements

2.2.1. Host Operating System Requirements

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

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

Sourcery G++ is built as a 32-bit application. Therefore, even when running on a 64-bit host system, Sourcery G++ requires 32-bit host libraries. If these libraries are not already installed on your system, you must install them before installing and using Sourcery G++. 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++ 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++ toolchain.

2.2.2. Host Hardware Requirements

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

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

In addition, the graphical installer requires a similar amount of temporary space during the installation process. On Microsoft Windows hosts, the installer uses the location specified by the `TEMP` environment variable for these temporary files. If there is not enough free space on that volume, the installer

prompts for an alternate location. On Linux hosts, the installer puts temporary files in the directory specified by the `IATEMPDIR` environment variable, or `/tmp` if that is not set.

2.2.3. Target System Requirements

See Chapter 3, “Sourcery G++ for ARM uClinux” for requirements that apply to the target system.

2.3. Registering with the Sourcery G++ Portal

If you do not already have a Sourcery G++ Portal account, you must register for one now. You must have an active Sourcery G++ subscription to download an installer or generate a license key. Evaluation subscriptions are available at no charge and also give you access to support from CodeSourcery.

If you purchased Sourcery G++ directly from CodeSourcery, you already have an account, and you may skip ahead to the next section. However, if you received Sourcery G++ with a hardware development kit or from a distributor, you probably do not have an account.

To register for an account, visit the Sourcery G++ Portal¹. Click on the link to register for an evaluation subscription. Follow the instructions on the web site to create your account. Then, once your account is active, click the button to request an evaluation subscription.

You should request an evaluation version of Sourcery G++ that matches the version you received with your development kit. Select the host system where you will install Sourcery G++, and ARM uClinux as the target system where you will run applications. Then click the `Request Evaluation` button.

If there are newer versions of Sourcery G++ than the one provided with your development kit, they will be visible through the Sourcery G++ Portal once your evaluation subscription is active. CodeSourcery recommends that you first work with the version of Sourcery G++ that came with your development kit, since CodeSourcery and the manufacturer have tested that particular combination of hardware and software. However, you may also wish to experiment with newer versions.

2.4. Downloading an Installer

If you have received Sourcery G++ on a CD, or other physical media, then you do not need to download an installer. You may skip ahead to Section 2.5, “Installing Sourcery G++”.

Log into the Sourcery G++ Portal² to download your Sourcery G++ toolchain(s). This version of Sourcery G++ requires a valid subscription or evaluation. CodeSourcery also makes some toolchains available to the general public from the Sourcery G++ web site³. These publicly available toolchains do not include all the functionality of CodeSourcery’s product releases.

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++ is provided as an executable installer package with the `.bin` extension.

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

¹ <https://support.codesourcery.com/GNUToolchain/>

² <https://support.codesourcery.com/GNUToolchain/>

³ http://www.codesourcery.com/gnu_toolchains/

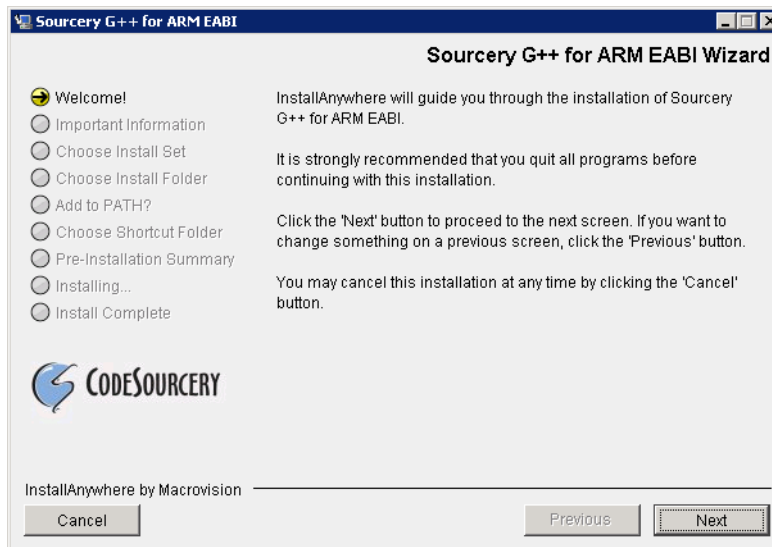
2.5. Installing Sourcery G++

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

2.5.1. Using the Sourcery G++ Installer on Microsoft Windows

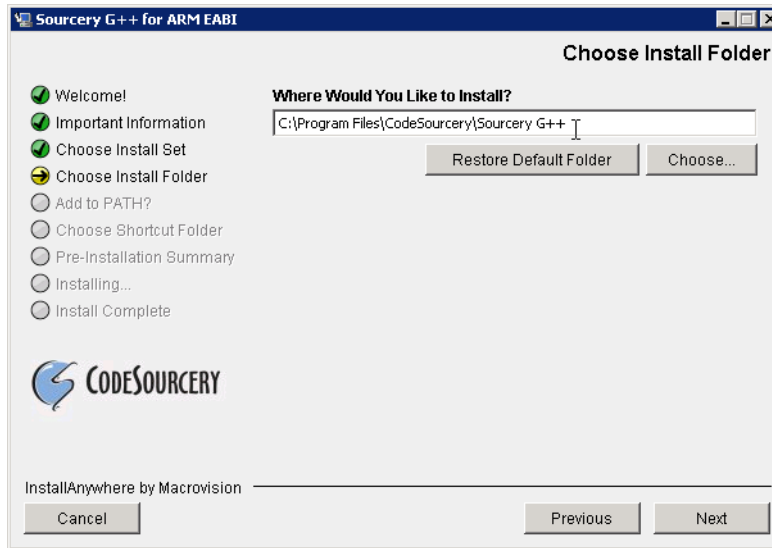
If you have received Sourcery G++ on CD, insert the CD in your computer. On most computers, the installer then starts automatically. If your computer has been configured not to automatically run CDs, open `My Computer`, and double click on the CD. If you downloaded Sourcery G++, double-click on the installer.

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

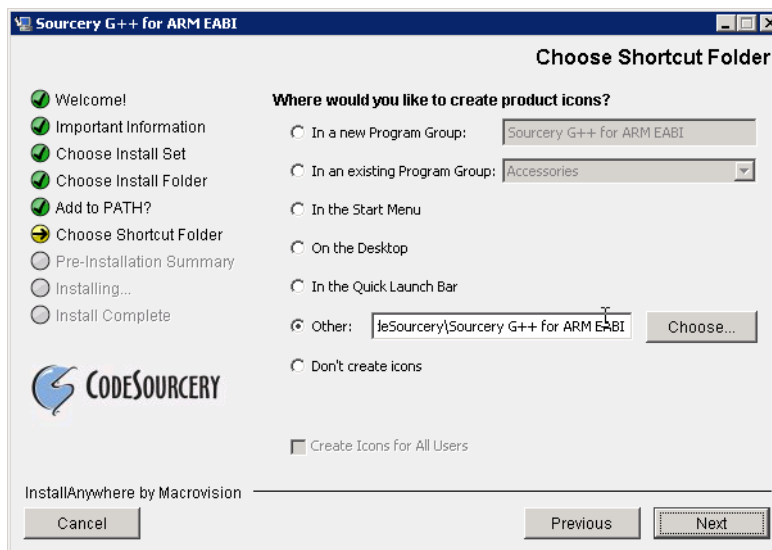


Running the Installer. The graphical installer guides you through the steps to install Sourcery G++.

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

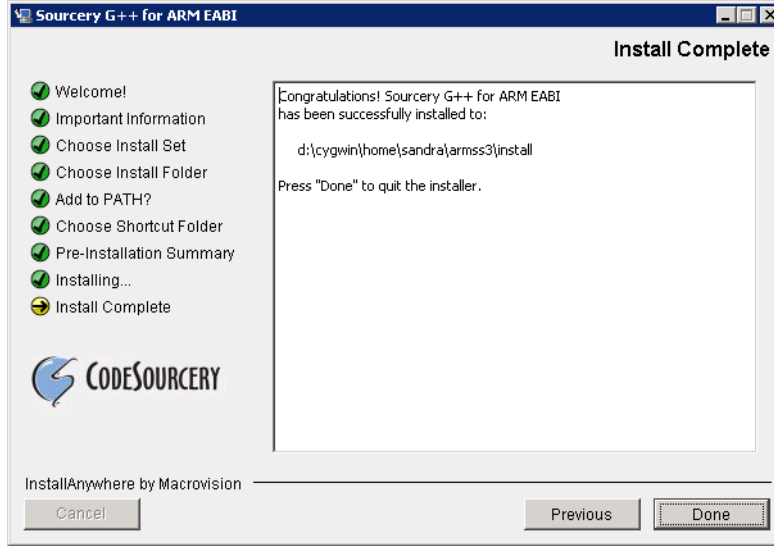


Choose Install Folder. Select the pathname to your install directory.



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

When the installer has finished, it asks if you want to launch the Sourcery G++ IDE and 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
```

2.5.2. Using the Sourcery G++ 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++. 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
```

2.5.3. Installing the Java Runtime Environment

Sourcery G++ for ARM uClinux includes the Sourcery G++ IDE, based on the Eclipse Integrated Development Environment. Eclipse is a Java application and requires the Java Runtime Environment (JRE).

If you use the graphical installer, the JRE is included when you install the Sourcery G++ IDE. Otherwise you must install the JRE separately if you wish to use the Sourcery G++ IDE. The Java Runtime Environment is available at no charge from Sun Microsystems Java website⁴. You may download either the Java Runtime Environment (JRE) or the Java Development Kit (JDK). (The JDK includes the JRE.)

⁴ <http://java.sun.com/j2se/>

2.6. Installing Sourcery G++ Updates

If you have already installed an earlier version of Sourcery G++ for ARM uClinux 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.

Note that the names of the Sourcery G++ commands for the ARM uClinux target all begin with **arm-uclinuxeabi**. This means that you can install Sourcery G++ for multiple target systems in the same directory without conflicts.

2.7. Setting up the Environment

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

2.7.1. Setting up the Environment on Microsoft Windows Hosts

2.7.1.1. Setting the PATH

In order to use the Sourcery G++ tools from the command line, you should add them to your PATH. If you plan to use only the Sourcery G++ IDE, it is not necessary to adjust your PATH, and you may skip this step. You may also 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++ installation.

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

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

```
> arm-uclinuxeabi-g++ -v
```

Verify that the last line of the output contains: `Sourcery G++ 4.4-61`.

2.7.1.2. Working with Cygwin

Sourcery G++ does not require Cygwin or any other UNIX emulation environment. You can use Sourcery G++ directly from the Eclipse IDE or 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++ 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++ 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.

When you run GDB from a Cygwin shell instead of a bare Windows console, Cygwin's terminal input handling conflicts with some features used by GDB, such as `Ctrl+C`, arrow key support, and automatic page breaks. To work around these issues, Sourcery G++ includes a Cygwin wrapper for GDB. When you use a Cygwin console, xterm, or SSH session, run `arm-uclinuxeabi-cyggdb` instead of `arm-uclinuxeabi-gdb`.

2.7.2. Setting up the Environment on GNU/Linux Hosts

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

Before using Sourcery G++ 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++ in an alternate location, you must replace the directory above with `bin` subdirectory of the directory in which you installed Sourcery G++.

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++-arm-uclinuxeabi/man`.

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

```
> arm-uclinuxeabi-g++ -v
```

Verify that the last line of the output contains: `Sourcery G++ 4.4-61`.

2.8. License Keys

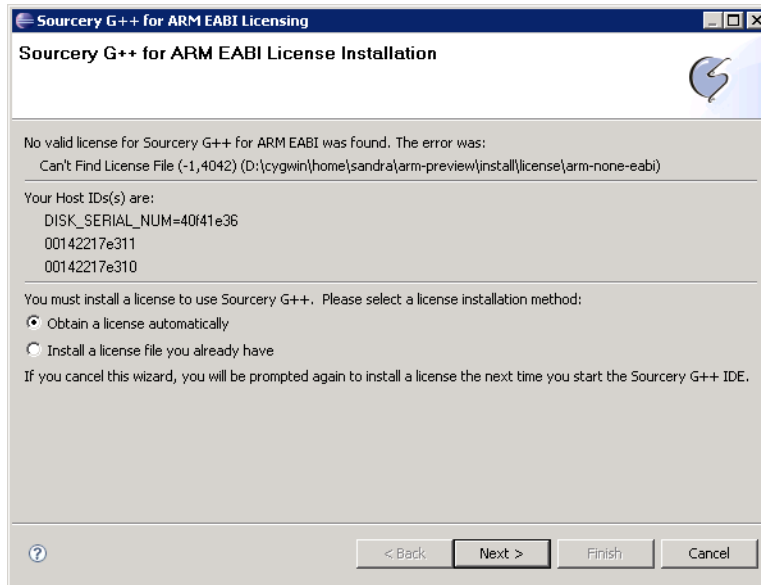
Sourcery G++ requires a license key. Each license key is associated with a particular computer, and is valid for the term of your subscription. Therefore, when you renew your subscription, you must download and install a new license key.

License Keys and the GPL

A license key is required to use the version of the GNU Compiler Collection included in Sourcery G++. Because GCC is made available under the terms of the GPL, all of the code linked into GCC is also covered by the GPL, including the code that checks for a license key. The GPL permits you to recompile the source code to remove the requirement that a license key be present. However, CodeSourcery's support covers only the original, validated GCC binaries provided with Sourcery G++.

2.8.1. Using the Licensing Wizard

When you start the Sourcery G++ IDE for the first time, it launches the Sourcery G++ Licensing wizard to guide you through the steps to install your license key.



Licensing Wizard. Use the Licensing wizard to install your license key.

The wizard displays the host ID (or host IDs) for the computer on which you have installed Sourcery G++. If your computer has more than one host ID, you must choose one of them for generating your license key.

Host IDs are either 12-digit hexadecimal numbers based on network interface addresses, or hard drive serial numbers prefixed with the text `DISK_SERIAL_NUM=`. You should choose a host ID associated with a device that is unlikely to be removed from your computer. If Sourcery G++ can determine your hard drive serial number, this is usually the best choice for your host ID, since network interface addresses may change if you change your networking configuration, as on a laptop that you use with a dock or removable network card.

Computers without a Host ID

If Sourcery G++ cannot determine your network interface address or hard drive serial number, it reports that there is no host ID for your computer.

On GNU/Linux systems, the network interface address is determined by looking for the `eth0` interface. Therefore, if your network interface has a different name, the wizard is unable to determine your host ID.

If the Sourcery G++ Licensing wizard cannot determine your host ID, please contact support@codesourcery.com for assistance.

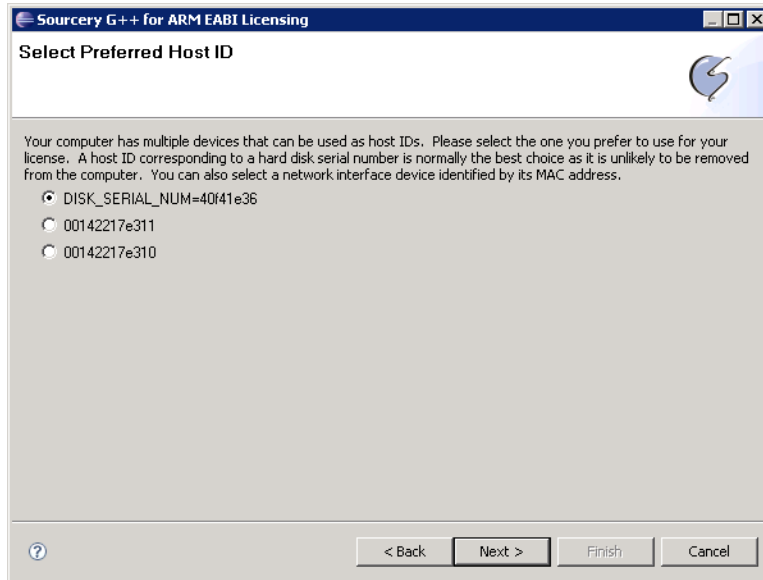
To continue with the Licensing wizard, you must select a license installation method.

Obtain a license automatically. If you select this option, the Licensing wizard contacts the Sourcery G++ Portal to download the license key for your host. This is the easiest way to install the license for a new Sourcery G++ subscription. Refer to Section 2.8.2, “Obtaining a License Automatically” for instructions on how to continue.

Install a license file you already have. You can use this method if you already have a valid license key file for your Sourcery G++ product, or if you prefer to download your license key manually from the Sourcery G++ Portal. You must also use this method to install a Sourcery G++ license on a computer that is not connected to the Internet. If you choose this installation method, refer to Section 2.8.4, “Manually Downloading Your License Key” and Section 2.8.5, “Installing a License File”.

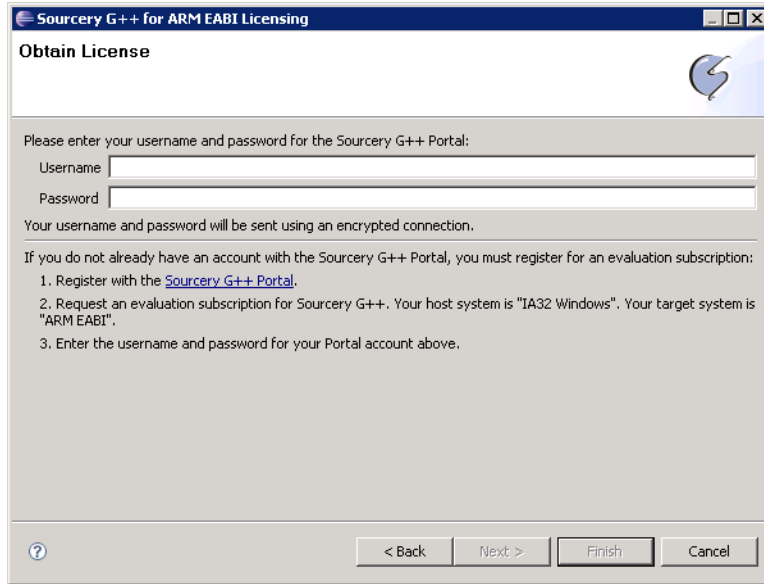
2.8.2. Obtaining a License Automatically

As mentioned in Section 2.8.1, “Using the Licensing Wizard”, if more than one host ID was found for your host computer, you must choose one to associate with your license. If the Licensing wizard has found only one host ID, it uses that ID and skips the selection screen.



Select Preferred Host ID. Select the host ID to associate with your license.

The next step is to supply your login information for the Sourcery G++ Portal. When you click on **Finish**, the Licensing wizard downloads and installs your license.



Obtain License. Supply your username and password for the Sourcery G++ Portal to obtain your license.

2.8.3. Configuring a Proxy Server

If the Licensing wizard is unable to connect to the Sourcery G++ Portal, this may mean access to the Portal is blocked by a firewall or VPN that requires the use of a proxy server for access to external web sites.

To set up your proxy configuration, first cancel the Licensing wizard. Then select **Window** → **Preferences** from the top menu, and then the **General** → **Network Connections** panel of the dialog. Contact your local system administrator for assistance with the proxy settings; they are typically the same as required for your regular web browser.

After you save your proxy settings, restart the Licensing wizard by selecting **Help** → **Sourcery G++ for ARM uClinux Licensing** from the top menu bar.

If you cannot use a proxy server to fetch your license key, you may still be able to download and install a license key manually, as described in the following sections. Otherwise, contact support@codesourcery.com for assistance.

2.8.4. Manually Downloading Your License Key

If you choose the option to have the Licensing wizard install your license key from a file, but do not already have a file containing your license key, you must visit CodeSourcery's web site to generate your license key. Log in to the Sourcery G++ Portal⁵. If you do not yet have an account, you must register for one at this time.

⁵ <https://support.codesourcery.com/GNUToolchain/>

After you log in, click on the ID of the Sourcery G++ product configuration for which you need a license key. On the next screen, click on the `Generate Key` button. The Sourcery G++ Portal then displays a dialog box:

Host IDs
Fill in the host IDs for the machine(s) on which you will use Sourcery G++ below.

The easiest way to determine your host ID is to run the Sourcery G++ IDE. If you do not have a valid license, the Sourcery G++ IDE will display your host ID.

The host ID is not the IP address or machine name for your host system.

Once you add a host ID to your license, you will not be able to remove it. Please check host IDs carefully before submitting this form. If you need to change one or more Host IDs associated with an existing license, please send mail to support@codesourcery.com.

Host ID #1

License Creation. Provide your Host ID to the Sourcery G++ Portal to generate a license.

Caution

You must use one of the host ID values displayed by the Sourcery G++ IDE's Licensing wizard. If you enter an incorrect host ID, you will not be able to correct the value. Instead, you must contact CodeSourcery for assistance in regenerating your license.

Your host ID appears in the dialog box. After checking the value, click the `Generate License` button.

After generating your key, the Sourcery G++ Portal provides a link that you can use to download your license file:

Download License File
Place the license file in the `azm-none-eabi` subdirectory of your Sourcery G++ installation.

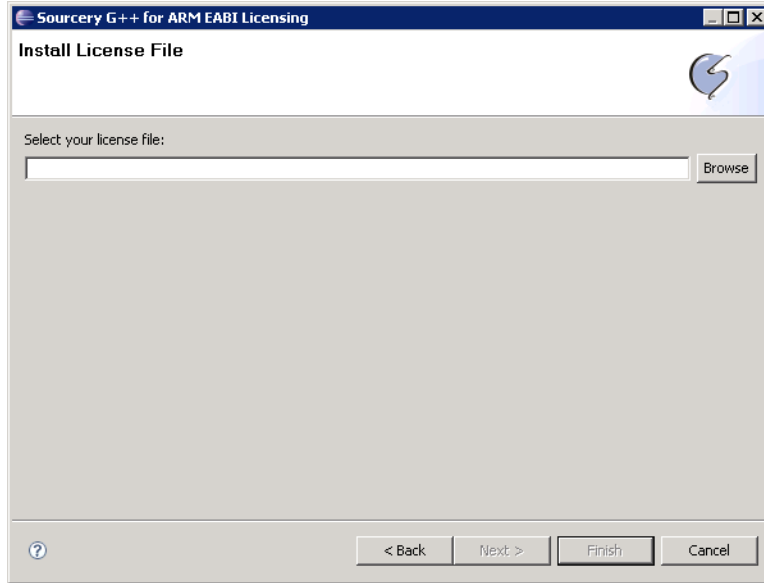
[Download the License File](#)

License Download. Click on the link to download your license file.

Click on the `Download the License File` link and save the file on your desktop or in another convenient location.

2.8.5. Installing a License File

If you choose the `Install a license file you already have` option in the Sourcery G++ Licensing wizard, the next screen allows you to select the file containing your license key. Click `Finish` to install your license and exit the wizard.



Install License File. Enter the filename to install a license from a file you already have.

2.8.6. Viewing or Reinstalling Your License Key

The Sourcery G++ Licensing wizard is normally started automatically when you run the IDE without a valid license key installed. You may also start the wizard explicitly by selecting `Help → Sourcery G++ for ARM uClinux Licensing` from the top menu bar.

If you start the Licensing wizard when you already have a valid license key installed, the wizard displays information about your current license on the first page. You can also request or install a new license key by invoking the Licensing wizard in this way. For example, this allows you to replace a temporary evaluation license key with a permanent one after you purchase a Sourcery G++ subscription.

2.9. Installing Add-Ons

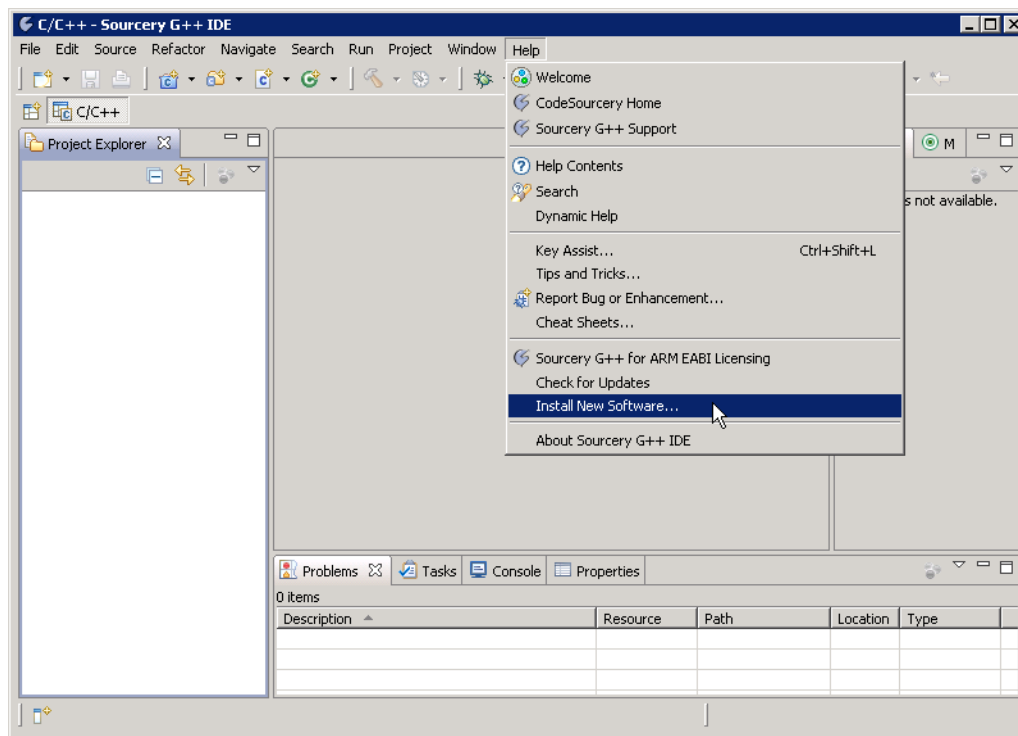
Professional Edition subscribers may download and install add-ons that provide additional features and functionality for Sourcery G++, including:

- Source code for the run-time libraries packaged for the debugger. While library source code is included in the freely available Sourcery G++ source package as well, the add-on package arranges the files for the debugger to find them and includes additional source files generated automatically during the build process.

Add-ons are managed from the Sourcery G++ IDE. To browse for and install available add-ons for your version of Sourcery G++, start the IDE and select `Help → Install New Software...`

Note about Screen Shots

The screen shots included in this section are provided for illustrative purposes only. The list of available add-ons for your version of Sourcery G++ will be different than those shown in these examples.

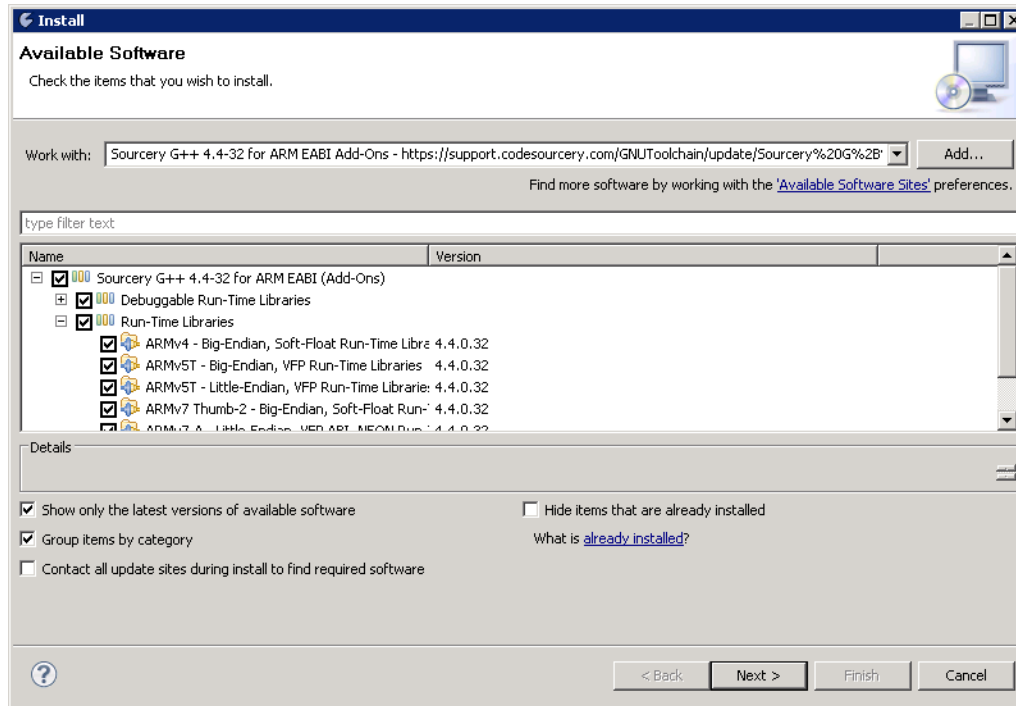


Install New Software. Add-on configurations are managed from the Sourcery G++ IDE.

In the `Work with:` drop-down, choose the Sourcery G++ update site for your target.

At this point, the IDE prompts you to enter your username and password for the Sourcery G++ Portal, as you did when installing your license key. See Section 2.3, “Registering with the Sourcery G++ Portal”. If your computer is behind a firewall and you have problems accessing the Portal from the Sourcery G++ IDE, you may need to check your proxy settings. Refer to Section 2.8.3, “Configuring a Proxy Server” for instructions.

Scanning the update site populates the list of available add-ons. Click the checkboxes for the ones you want to install; selecting the top-level list entry selects all available Sourcery G++ add-ons.



Available Software. Choose the Sourcery G++ update site, then check the boxes for the add-ons you want to install.

Click **Next** to continue and follow the prompts in the remaining pages of the wizard to download the add-ons. Then restart the Sourcery G++ IDE when prompted by the dialog box. At that point, installation of the add-ons is complete.

2.10. Uninstalling Sourcery G++

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

2.10.1. Using the Sourcery G++ Uninstaller on Microsoft Windows

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

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

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

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

2.10.2. Using the Sourcery G++ Uninstaller on GNU/Linux

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

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

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.

Chapter 3

Sourcery G++ for ARM uClinux

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

3.1. Included Components and Features

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

Component	Version	Notes
GNU programming tools		
GNU Compiler Collection	4.4.1	Separate manual included.
GNU Binary Utilities	2.19.51	Includes assembler, linker, and other utilities. Separate manuals included.
Sourcery G++ IDE		
Eclipse IDE	Galileo	See Chapter 4, “Using the Sourcery G++ IDE”. Additional documentation is available through the Eclipse online help facility.
Eclipse C/C++ Development Tools	6.0	
Sourcery G++ Eclipse Plugin(s)	4.4-61	
Sourcery G++ IDE Launcher	4.4-61	
Debugging support and simulators		
GNU Debugger	6.8.50	Separate manual included.
Sourcery G++ Cygwin GDB Wrapper	4.4-61	See Section 2.7.1.2, “Working with Cygwin”.
Sourcery G++ Debug Sprite for ARM	4.4-61	Provided for kernel debugging only. See Chapter 6, “Sourcery G++ Debug Sprite”.
GDB Server	N/A	Included with GDB. See Section 3.6, “GDB Server”.
Target libraries		
uClibc C Library	0.9.30	
Linux Kernel Headers	2.6.30	
Library Debug Information	N/A	Available as an add-on for Professional Edition subscribers. See Section 2.9, “Installing Add-Ons”.
Other utilities		
ELF-to-FLT Conversion Utility	N/A	
GNU Make	N/A	Build support on Windows hosts.
GNU Core Utilities	N/A	Build support on Windows hosts.

3.2. Library Configurations

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

Each multilib corresponds to a *sysroot* directory that contains the files that should be installed on the target system. You can find the *sysroot* directories provided with Sourcery G++ in the `arm-uclinuxeabi/libc` directory of your installation.

3.2.1. Included Libraries

The following library configurations are available in Sourcery G++ for ARM uClinux.

ARMv4T - Little-Endian, Soft-Float	
Command-line option(s):	default
Sysroot subdirectory:	./

ARMv6-M Thumb - Little-Endian, Soft-Float	
Command-line option(s):	-mthumb -march=armv6-m
Sysroot subdirectory:	armv6-m/

ARMv7 Thumb-2 - Little-Endian, Soft-Float	
Command-line option(s):	-mthumb -march=armv7 -mfix-cortex-m3-ldrd
Sysroot subdirectory:	thumb2/

3.2.2. Library Selection

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

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

```
> arm-uclinuxeabi-gcc -print-multi-directory options...
```

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

When you use the Sourcery G++ IDE to build your application, the target options that you set from the New Project Wizard or the `Target` tab of the project properties dialog apply to both compilation and linking. The specific options for ARM uClinux targets are documented in Section 4.2.1, “Setting Up an Example Project”. If you need to adjust the compiler and linker command-line options separately, you can do this from the project properties dialog, as described in Section 4.2.5, “Customizing Build Actions”. To check the exact command-line options that are passed to the linker, look at the command log in the `Console` tab after building your project.

3.3. Using VFP Floating Point

3.3.1. Enabling Hardware Floating Point

GCC provides three basic options for compiling floating-point code:

- Software floating point emulation, which is the default. In this case, the compiler implements floating-point arithmetic by means of library calls.
- VFP hardware floating-point support using the soft-float ABI. This is selected by the `-mfloat-abi=softfp` option. When you select this variant, the compiler generates VFP floating-point instructions, but the resulting code uses the same call and return conventions as code compiled with software floating point.
- VFP hardware floating-point support using the VFP ABI, which is the VFP variant of the Procedure Call Standard for the ARM® Architecture (AAPCS). This ABI uses VFP registers to pass function arguments and return values, resulting in faster floating-point code. To use this variant, compile with `-mfloat-abi=hard`.

You can freely mix code compiled with either of the first two variants in the same program, as they both use the same soft-float ABI. However, code compiled with the VFP ABI is not link-compatible with either of the other two options. If you use the VFP ABI, you must use this option to compile your entire program, and link with libraries that have also been compiled with the VFP ABI. For example, you may need to use the VFP ABI in order to link your program with other code compiled by the ARM RealView® compiler, which uses this ABI.

Sourcery G++ for ARM uClinux includes libraries built with software floating point, which are compatible with VFP code compiled using the soft-float ABI. While the compiler is capable of generating code using the VFP ABI, no compatible runtime libraries are provided for uClinux targets.

Note that, in addition to selecting hard/soft float and the ABI via the `-mfloat-abi` option, you can also compile for a particular FPU using the `-mfpu` option. For example, `-mfpu=neon` selects VFPv3 with NEON coprocessor extensions.

3.3.2. NEON SIMD Code

Sourcery G++ includes support for automatic generation of NEON SIMD vector code. Autovectorization is a compiler optimization in which loops involving normal integer or floating-point code are transformed to use NEON SIMD instructions to process several data elements at once.

To enable generation of NEON vector code, use the command-line options `-ftree-vectorize -mfpu=neon -mfloat-abi=softfp`. The `-mfpu=neon` option also enables generation of VFPv3 scalar floating-point code.

Sourcery G++ also includes support for manual generation of NEON SIMD code using C intrinsic functions. These intrinsics, the same as those supported by the ARM RealView® compiler, are defined in the `arm_neon.h` header and are documented in the 'ARM NEON Intrinsics' section of the GCC manual. The command-line options `-mfpu=neon -mfloat-abi=softfp` must be specified to use these intrinsics; `-ftree-vectorize` is not required.

3.3.3. Half-Precision Floating Point

Sourcery G++ for ARM uClinux includes support for half-precision (16-bit) floating point, including the new `__fp16` data type in C and C++, support for generating conversion instructions when compiling for processors that support them, and library functions for use in other cases.

To use half-precision floating point, you must explicitly enable it via the `-mfp16-format` command-line option to the compiler. For more information about `__fp16` representations and usage from C and C++, refer to the GCC manual.

3.4. ABI Compatibility

The Application Binary Interface (ABI) for the ARM Architecture is a collection of standards, published by ARM Ltd. and other organizations. The ABI makes it possible to combine tools from different vendors, including Sourcery G++ and ARM RealView®.

Sourcery G++ implements the ABI as described in these documents, which are available from the ARM Information Center¹:

- BSABI - ARM IHI 0036B (10 October 2008)
- BPABI - ARM IHI 0037B (10 October 2008)
- EHABI - ARM IHI 0038A (10 October 2008)
- CLIBABI - ARM IHI 0039A (10 October 2008)
- AADWARF - ARM IHI 0040A (10 October 2008)
- CPPABI - ARM IHI 0041B (10 October 2008)
- AAPCS - ARM IHI 0042C (10 October 2008)
- RTABI - ARM IHI 0043B (10 October 2008)
- AAELF - ARM IHI 0044C (10 October 2008)
- ABI Addenda - ARM IHI 0045B (10 October 2008)

Sourcery G++ currently produces DWARF version 2, rather than DWARF version 3 as specified in AADWARF.

3.5. Building uClinux Applications

When you use GCC to link a uClinux application, it creates two output files. The executable file, as specified by the `-o` command-line option, is a uClinux FLAT format binary (bFLT) file. This is the file you should copy to and run on your uClinux target. The second output file is an ELF-format file containing additional debug and symbol table information to allow you to debug your program with GDB, as described in Section 3.6, “GDB Server”. This file has a `.gdb` extension.

For example, if you specify the command

```
arm-uclinuxeabi-gcc foo.c -o bar
```

then `bar` is the FLAT-format executable and `bar.gdb` is the ELF-format file.

3.6. GDB Server

Sourcery G++ contains a **gdbserver** for running on the target. The server executable is located in the `sysroot/usr/bin` directory of your installation, where `sysroot` is the pathname to the `sysroot`, as documented in Section 3.2, “Library Configurations”. You need to copy the appropriate **gdbserver** executable to your target system and then invoke it as

¹ <http://infocenter.arm.com>

```
# gdbserver :port program
```

port can be any available TCP port; 5000 is a common choice. **gdbserver** waits for a connection from **gdb** and then commences serving requests for it. To connect to **gdbserver** from your host system, start **gdb**, but specify the special `.gdb` version of your program.

```
> arm-uclinuxeabi-gdb program.gdb
```

Then connect to the target system:

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

At this point you are able to debug as usual.

To connect to **gdbserver** from the Sourcery G++ IDE, follow the instructions in Section 4.3, “Debugging Applications” to connect to the Sourcery G++ External Server for ARM uClinux. Again, you must select the `.gdb` file as the application to debug.

Chapter 4

Using the Sourcery G++ IDE

This chapter explains how to use the Sourcery G++ IDE to build a C or C++ application. This chapter assumes you have installed Sourcery G++ as described in Chapter 2, “Installation and Configuration”. If you prefer to use the command line to build your applications, you may refer to Chapter 5, “Using Sourcery G++ from the Command Line” instead.

4.1. Overview

This chapter explains how to create, modify, and debug a program using the Sourcery G++ IDE. After working through the example program in this chapter, you can use the same techniques to create your own programs.

To start the Sourcery G++ IDE, use the launcher program which can be found in the `bin/` subdirectory of your Sourcery G++ installation. On Windows hosts, the launcher is called `sourcerygxx-ide.exe`. On Linux hosts, the launcher is called `sourcerygxx-ide`. Alternatively, if you installed Sourcery G++ with the graphical installer and specified a shortcut location, you can run the IDE using that shortcut.

When you start the IDE for the first time, it prompts you to select a *workspace* directory; this is where your program source files, compiled binaries, and other files managed by the IDE will be stored. Next, it starts the Sourcery G++ Licensing wizard to guide you through installation of your license key. Refer to Section 2.8, “License Keys” for more information. The IDE also displays a welcome screen with links you can click on to get more information about using Sourcery G++. When you are ready to begin using Sourcery G++, first close the welcome screen tab. You can return to the welcome screen again later, if you wish, from the `Help` menu.

Learning More About Eclipse

The Sourcery G++ IDE is based on Eclipse. While this chapter explains how to accomplish basic tasks using the Sourcery G++ IDE, it is not a comprehensive reference manual. If you want to learn more about Eclipse, use the online help from the welcome screen or as described in Chapter 7, “Next Steps with Sourcery G++”. You can also visit the Eclipse web site¹ for additional tutorials and documentation.

Note About Screen Shots

The screen shots included in this chapter are provided for illustrative purposes only. You may see slightly different sets of menu options when you run the IDE, corresponding to the specific features included in your version of Sourcery G++. Refer to the text for information about the particular targets and debugging options supported by Sourcery G++ for ARM uClinux.

The remainder of this chapter is divided into three sections. The first guides you through the process of creating and building an example program; the second section shows how to debug and run the program once it has been built. The final section covers advanced features of the Sourcery G++ IDE.

4.2. Building Applications

In the Sourcery G++ IDE, every program is a *project*. The project contains all of the source files required to build the program. So, the first step is to create a project.

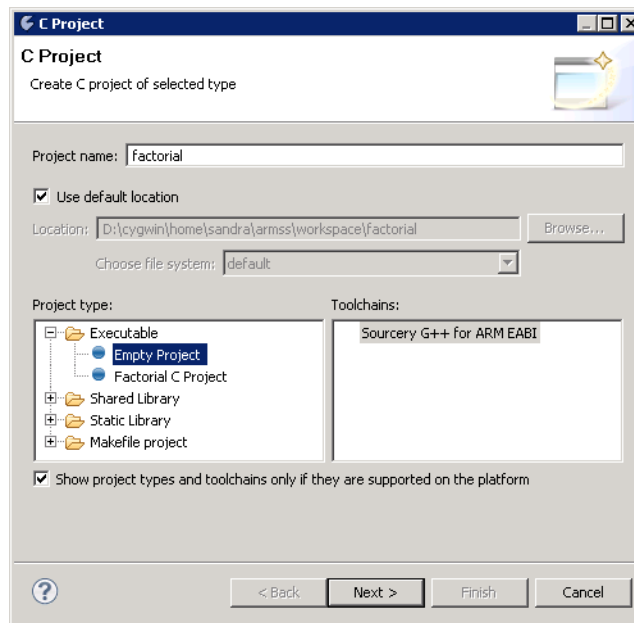
Normally, the IDE manages building your project for you. This is convenient if you intend to do all of your development from within the IDE. However, if you are working with code that has previously been built with **make**, you may wish to use a Makefile project instead. The following several sections explain how to create and work with a project using the IDE's managed build support. If you wish to use a Makefile project instead, refer to Section 4.4.1, “Makefile Projects”.

¹ <http://www.eclipse.org>

4.2.1. Setting Up an Example Project

Create a new project by selecting `File` → `New` → `C Project`. This opens the New Project Wizard. (If you want to build a C++ application, select `C++ Project` instead.) Then click `Next`.

Give the project the name `factorial`. From the `Project types` menu, expand `Executable`. Select `Empty Project` to begin a new empty project. (Selecting `Factorial C Project` creates a project pre-populated with the example used in this tutorial.) On the `Toolchain` menu ensure that `Sourcery G++ for ARM uClinux` is selected. Then click the `Next` button.

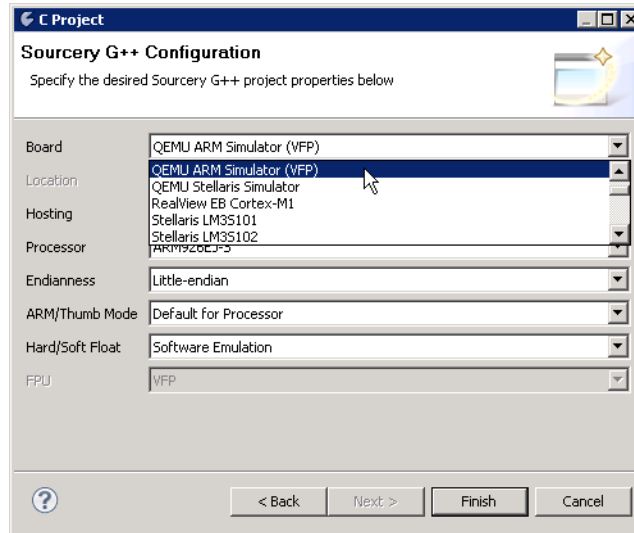


Creating an Executable Project. Use the New Project Wizard to create a new empty Executable project using the Sourcery G++ toolchain for your target.

The next page of the New Project Wizard allows you to customize the project build settings to match your target processor.

The following properties can be set on this page:

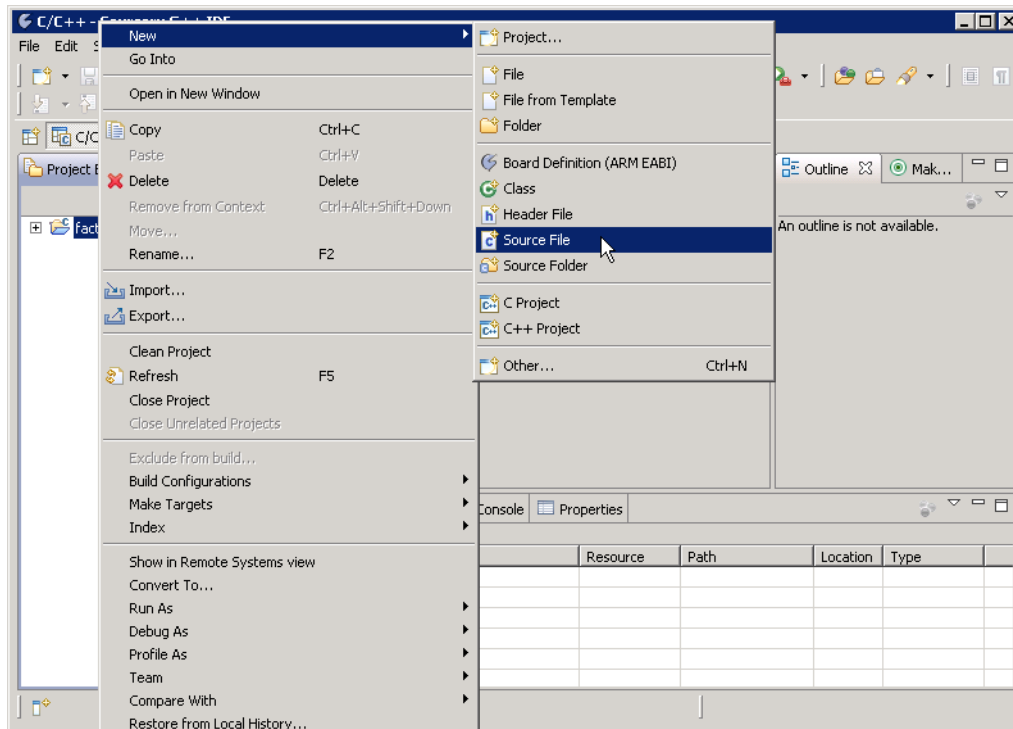
- `Processor`: Select the target processor. This controls code generation and library selection.
- `ARM/Thumb Mode`: Select whether to generate ARM or Thumb code.
- `Hard/Soft Float`: Select hardware or software floating-point support.
- `FPU`: Choose floating-point unit for code generation.



Setting Properties in the New Project Wizard. You can choose common build properties for your project from the New Project Wizard. The exact set of properties available depends on the target system.

Select **Finish** to create the project. If you are asked whether to open a new perspective, click the **Yes** button.

At this point, the project exists, but there is no associated source code. So, the next step is to create the main program. Right-click on the `factorial` project, and select **New** → **Source File**. Give the new file the name `main.c` and click the **Finish** button.



Adding a Source File. Right-click on the project name to add a new source file.

4.2.2. Writing Source Code

The Sourcery G++ IDE now displays an editing window for you to use to create the program. Type (or cut-and-paste) the following program into the editor:

```
#include <stdio.h>

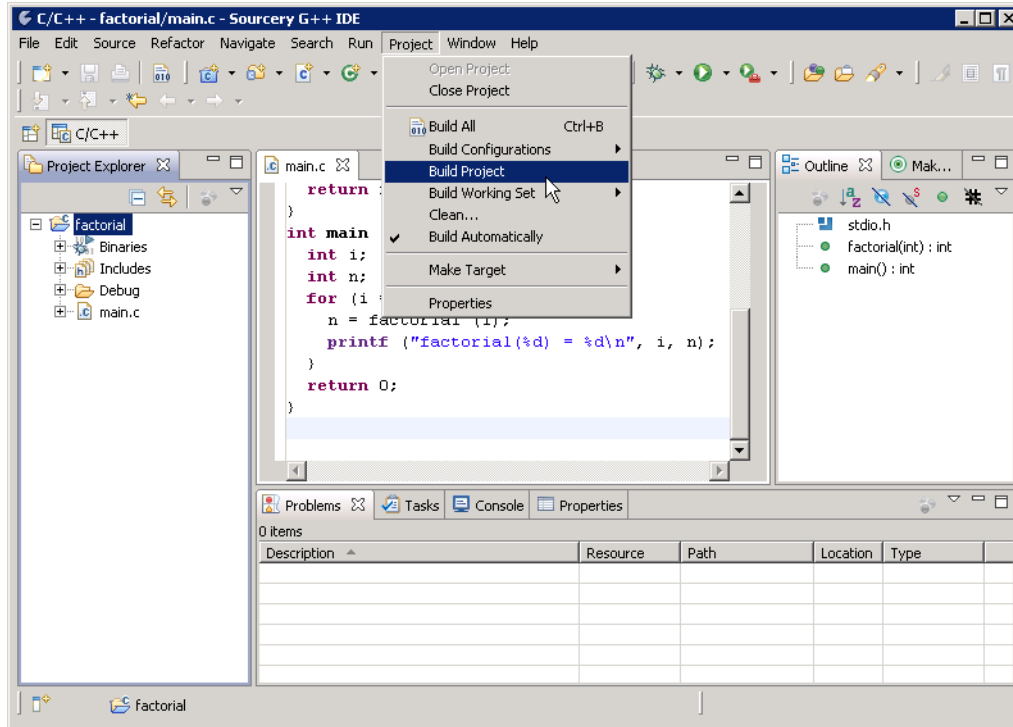
int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial (n - 1);
}

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

When you are done, save the file with **File** → **Save (Ctrl+S)**.

After you save the file, build your project by selecting it in the **Project Explorer** pane on the left, then going to the **Project** menu and selecting **Build Project**. The output of the commands run by the IDE is displayed in the **Console** tab in the lower pane. You should see the following output at the bottom of the console:

```
Finished building target: factorial
```



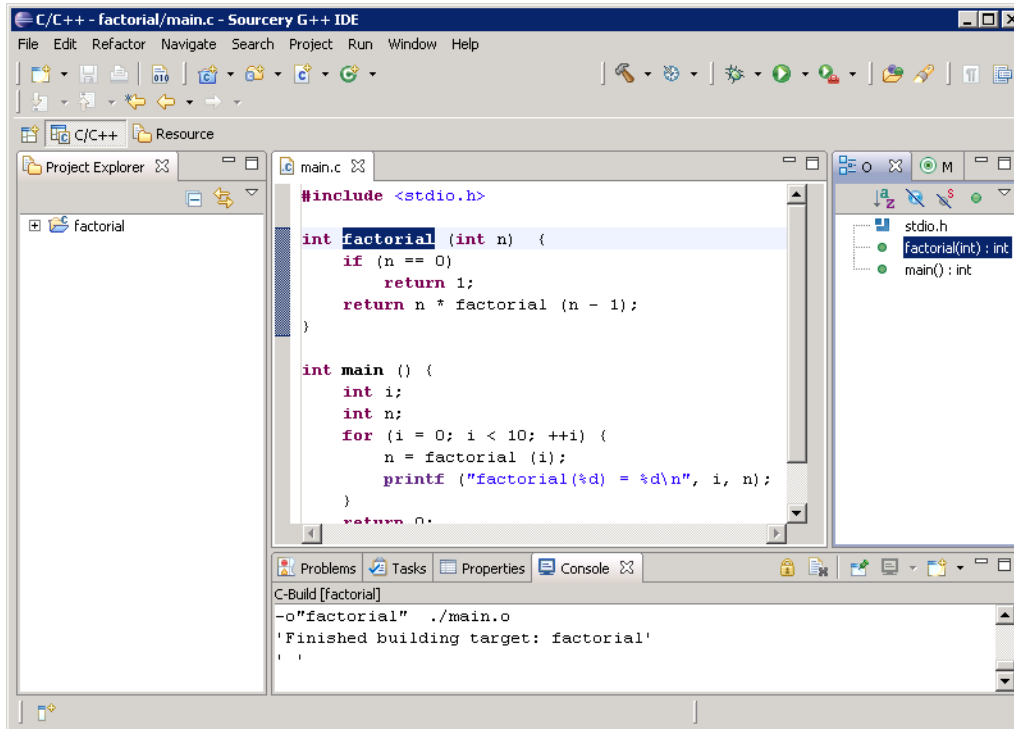
Building the Project. Use `Build Project` to build the project after saving it.

Building Automatically

If you want the IDE to build your project to automatically when you add or save files, in addition to selecting `Build Automatically` from the `Project` menu, you also need to enable the setting on a per-project basis. From the `Project` menu, select `Properties`, then `C/C++ Build`. Open the `Behaviour` tab and check `Build on resource save (Auto build)`.

4.2.3. Using Cross-Reference Information

Whenever it rebuilds your project, the Sourcery G++ IDE also computes cross-reference information. You can see some of this information in the `Outline` pane. In particular, each of the two functions in the program (`factorial` and `main`) are shown in the `Outline` pane. When you click on name of a function or variable in the `Outline` pane, the IDE repositions the cursor to show you that entity.





Using the Outline. Click a function name in the Outline pane to jump to it in the editor.

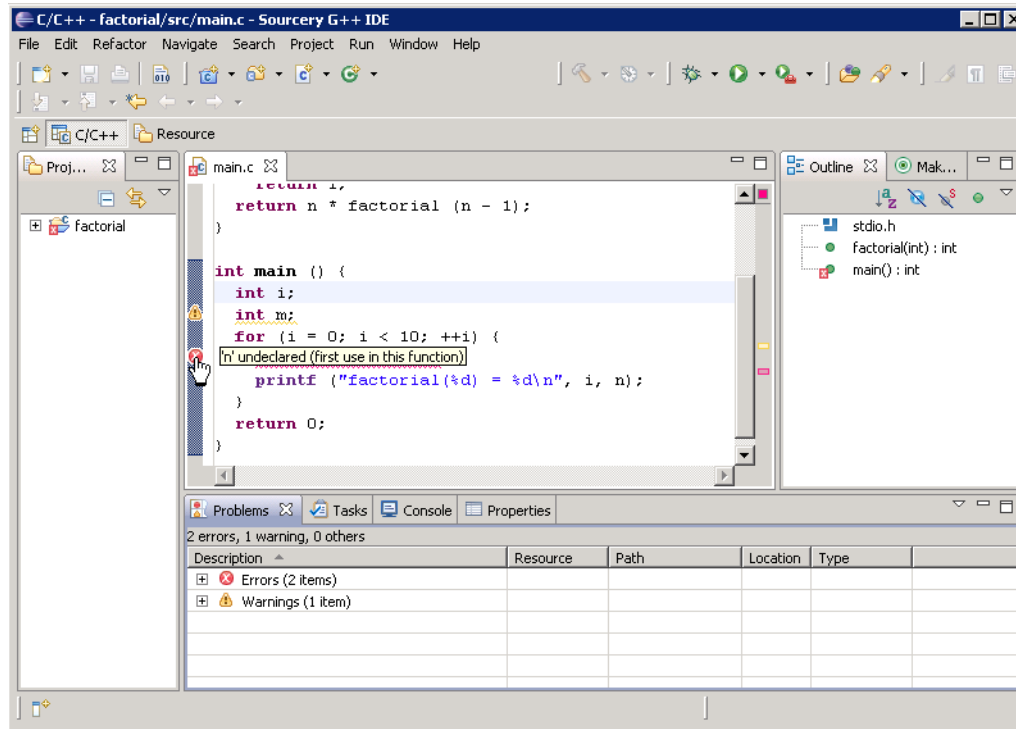
You can also use the cross-reference information to jump from a reference to a function or variable to its definition. For example, find the line in `main` that calls `factorial` and place the cursor over the name `factorial`. This pops up a small box showing the definition of the function. You can right-click and select `Open Declaration (F3)` to jump to the definition of `factorial` in the editor. The cross-reference functionality works even if the function call is in a different file from the declaration of the function.

4.2.4. Dealing with Errors

If you pasted the sample application into the IDE, the program probably compiled correctly the first time. But, of course, that rarely happens when writing a large program from scratch. To see how the Sourcery G++ IDE deals with errors, you can intentionally introduce an error.

Change the declaration of `n` in `main` to declare `m`, instead of `n`, and save the file. This change makes the program invalid because there are references to `n` in the function, but no declaration. In addition, the new variable `m` is not serving any useful purpose (since there are no references to it). Sourcery G++ informs you of both issues by flagging the problematic lines of source code.

The IDE places a circular red symbol  next to lines that cause errors and a triangular yellow symbol  on lines that cause warnings. There are several ways to get more detailed information about the problems. One way is to click on the `Problems` pane at the bottom of the IDE. This pane shows the error and/or warning messages issued by the compiler. Also, when you place the cursor over the error indicators, the IDE displays the error message.



Viewing Errors. Place the cursor over a warning or error indicator to see the cause of the problem.

Correct the error by changing `m` back to `n`, and then rebuild the project. The IDE removes the error indicators and the Problems pane is cleared, indicating a successful build.

4.2.5. Customizing Build Actions

If you wish, you can customize the actions the Sourcery G++ IDE uses to build your project. To do this, pull up the project properties dialog by right-clicking on your project name in the Project Explorer pane, and selecting Properties. Then expand C/C++ Build and select Settings.

You can adjust the options for invoking the compiler, assembler, and linker from the Tool Settings tab. The project properties initially set in the New Project Wizard (Section 4.2.1, “Setting Up an Example Project”) can be adjusted by selecting the Target category. The other categories allow you to configure other options specific to each tool. There are dialog boxes for the most common types of options, or you can specify arbitrary command-line options by selecting the Miscellaneous category for each tool.

For example, if you wish to build for a processor that is supported by the compiler but not listed in the Processor option in the Target category, you can select Other as the Processor, and then enter the appropriate command-line options directly in the Miscellaneous categories for the compiler, assembler, and linker.

You can also specify additional pre-build and post-build steps from the Build steps tab. For example, to automatically produce a disassembly listing of the executable after linking, enter the command

```

${cs_target}-objdump -ldr ${BuildArtifactFileName}.gdb \
> ${BuildArtifactFileName}-asm.txt

```


in the Command field for Post-build steps.

In this example, `cs_target` and `BuildArtifactFileName` are predefined *build macros* that expand to the name of the target prefix for Sourcery G++ commands (`arm-uclinuxeabi`, in the case of ARM uClinux), and the output filename from the build process, respectively. You can browse the list of available build macros and define new ones by selecting **Variables** in the left-hand pane of the project properties dialog.

4.3. Debugging Applications

4.3.1. Starting the Debugger

After you build your application, select it in the left-hand **Project Explorer** pane. Then choose **Run → Debug Configurations...** from the menu bar. This opens the dialog for creating and editing debug launch configurations.


The pane on the left lists the available debug launch configuration types. Sourcery G++ for ARM uClinux provides these custom launch types:

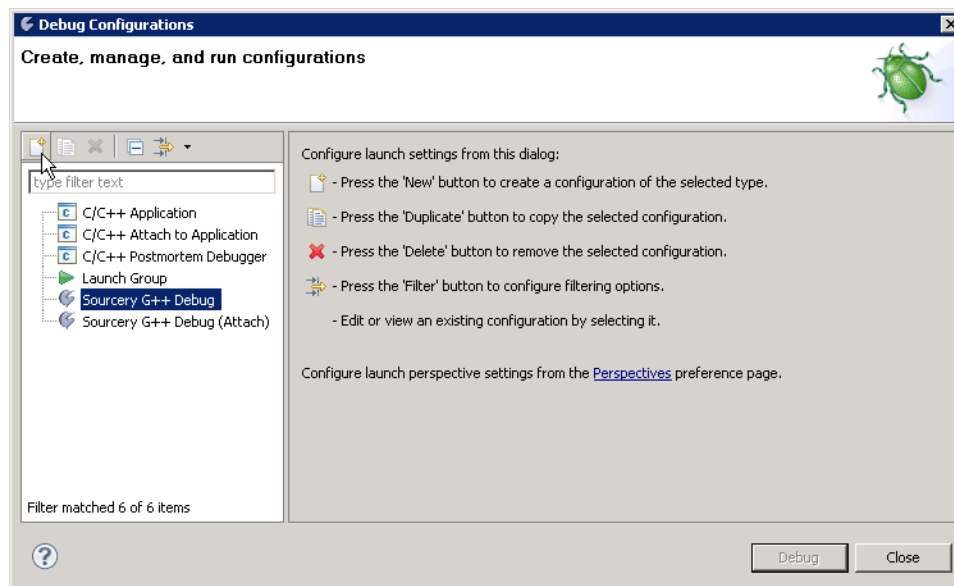
Sourcery G++ Debug. Use this launch configuration type to run an application on the target. This is the most typical launch type for application program debugging.

Sourcery G++ Debug (Attach). Use this launch configuration type to attach the debugger to an application which has already been started on the target.

Sourcery G++ Debug (Postmortem). Use this launch configuration type for debugging an application that has terminated, leaving a core file.

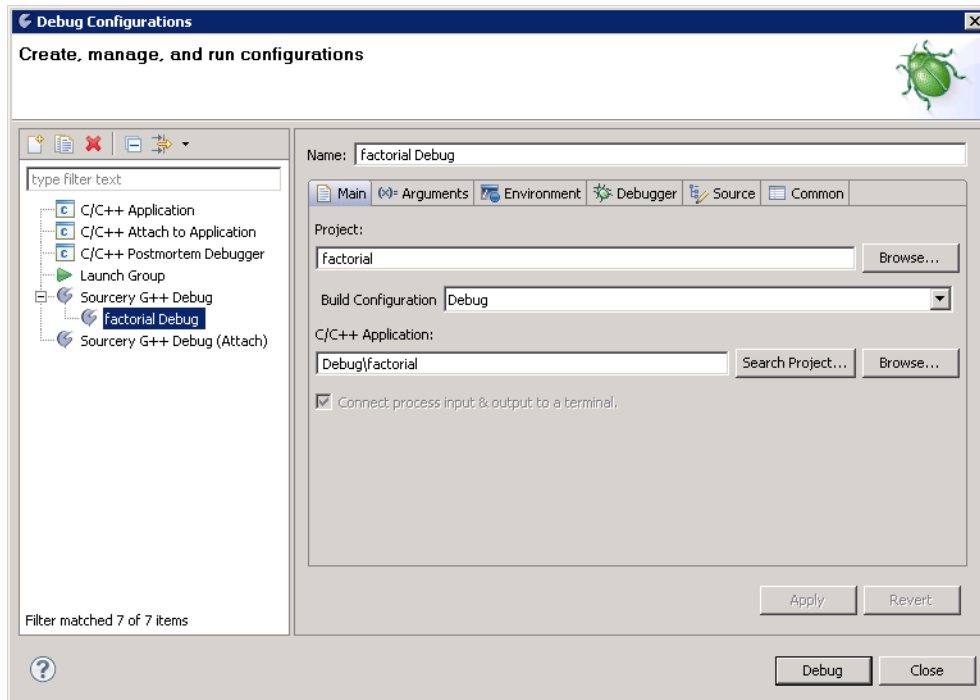
Sourcery G++ Kernel Debug (Attach). Use this launch configuration type for debugging a running uClinux kernel on the target using a hardware debug device. You cannot use this launch type for debugging ordinary user-space applications.

Select the **Sourcery G++ Debug** launch type in the left-hand pane. Then, click the **New** icon  positioned towards the upper left of the window.



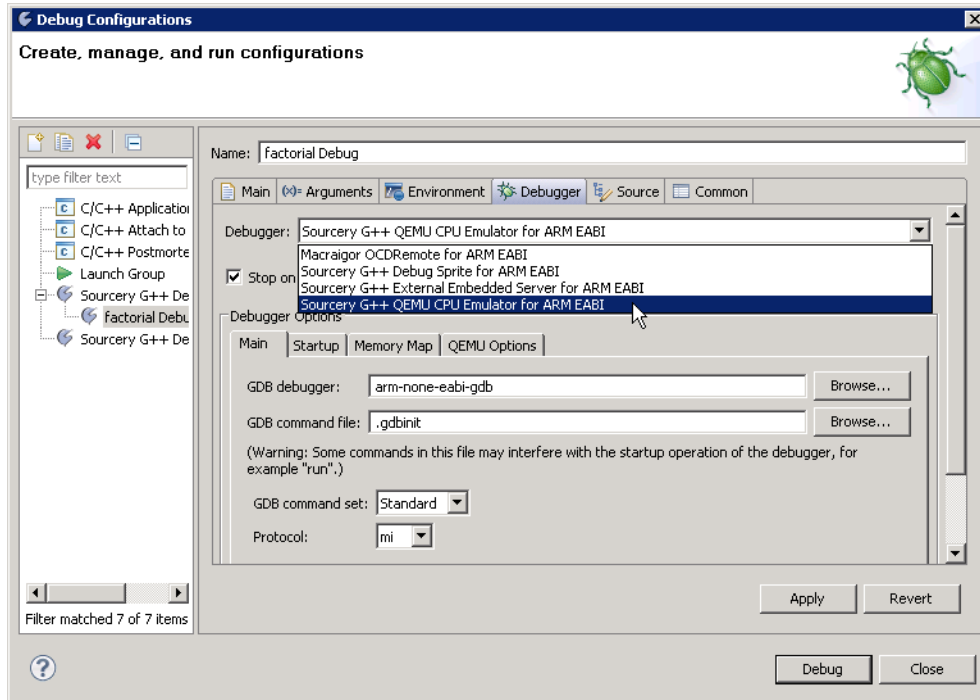
Creating a Debug Configuration. Select Sourcery G++ Debug and click the New icon to create to create a new debug configuration.

When you create the launch configuration, a new pane appears on the right. On the Main tab, use the Browse... button to select your project, if it is not already selected. Then, use the Search Project... or Browse... buttons to select your application. As discussed in Section 3.6, “GDB Server”, when debugging a uClinux application program you must select the .gdb file as the C/C++ Application.



Selecting a Program. Use the Search Project... button to locate your program.

Next, switch to the Debugger tab and select the debugging mode you want to use. The different debugging modes are discussed in detail below; the choices depend on which debug launch configuration type you have selected. Some debugging modes require you to configure additional options. When you have made any necessary adjustments, click the Debug button to start the debugger.



Selecting a Debugger. Use the drop-down menu to pick the debugger that you want to use.

You do not need to repeat the debugger selection process the next time you launch the debugger. Instead, you can select Run → Debug to start the debugger using the settings you have selected.

4.3.2. Application Debugging Modes

Sourcery G++ provides the following debugging modes for ARM uClinux applications.

Sourcery G++ External Server. Sourcery G++ includes a program called **gdbserver** that can be used to debug a program running on a remote ARM uClinux system. The External Server mode allows you to connect to an already-executing **gdbserver** or other debugging stub that uses the GDB remote protocol. You can use this mode with both Sourcery G++ Debug and Sourcery G++ Debug (Attach) launch configurations.

Sourcery G++ Postmortem Debugger for ARM uClinux. This debugging mode can be used only with the Sourcery G++ Debug (Postmortem) launch configuration. It allows you to use the debugger to examine the state of a program that has terminated, leaving a core file.

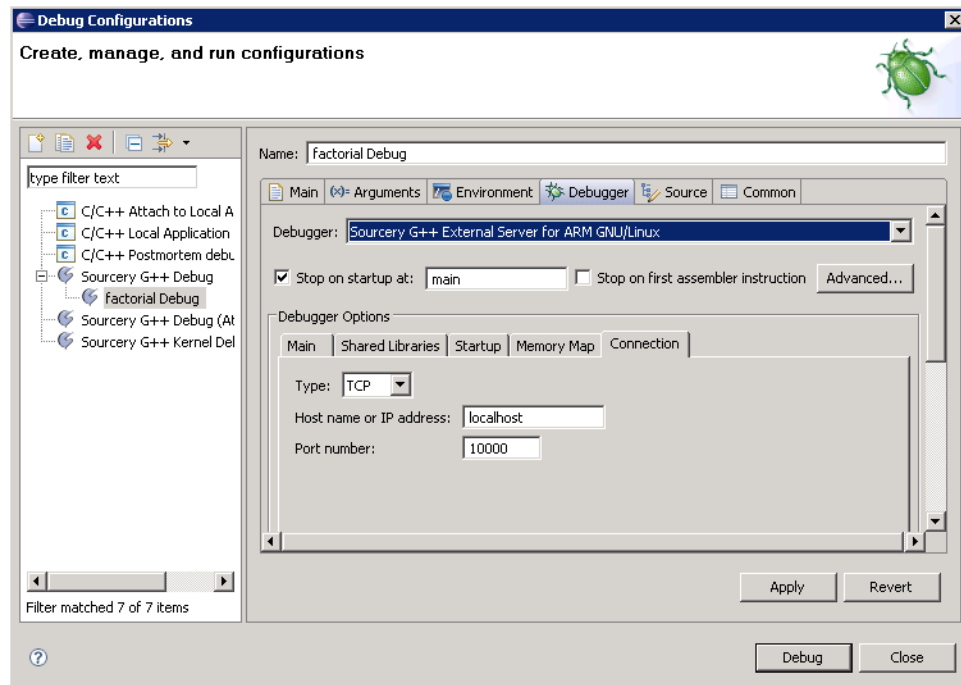
Detailed information about using each of these debugging modes is provided below.

4.3.2.1. Sourcery G++ External Server

The External Server mode allows you to connect to an already-executing **gdbserver** or other debugging stub that uses the GDB remote protocol.

To use External Server mode with GDB server, first follow the instructions in Chapter 3, “Sourcery G++ for ARM uClinux” to install and run **gdbserver** on your target system. You must use the **gdbserver** provided with Sourcery G++ rather than any **gdbserver** already installed on your target.

Then, use the `Connection` subtab of the `Debugger` tab to specify how the debugger connects to the target. Fill in the host name or IP address of your target, and the same TCP port number you use when starting `gdbserver` on the target. You can also use a serial connection instead of a TCP connection.



External Server Connection. Use the `Connection` subtab to configure the External Server debugger.

You must copy your application's executable to the target before you can run it with GDB server. You must restart GDB server on the target each time you run your program in the debugger. Console output from your application appears in the remote terminal where you run GDB server.

The Sourcery G++ IDE includes a built-in terminal emulator which you can use to open a command shell on your target. This allows you to run `gdbserver` on the target and interact with your application within the IDE, without having to switch to a separate terminal window. Refer to Section 4.4.5, “Using the Terminal Emulator” for instructions. You can also use the Remote System Explorer (RSE) file browser to transfer files between your host and target systems. For information about setting up and using RSE, see Section 4.4.6, “Using the Remote System Explorer”.

4.3.2.2. Sourcery G++ Postmortem Debugger for ARM uClinux

This debugging mode can be used only with the Sourcery G++ Debug (Postmortem) launch configuration. It allows you to use the debugger to examine the state of a program that has terminated, leaving a core file.

You do not need to set any additional configuration options to use the Postmortem debugging mode. When you launch the debug configuration, the Sourcery G++ IDE pops up a dialog box to allow you to select the core file to debug.

Debugger functionality is restricted in this debugging mode. Because the application has terminated, you cannot run or step it. However, you can use the debugger to examine the program state at the point where it terminated, such as the stack backtrace and contents of memory and registers.

4.3.3. Kernel Debugging Modes

The following debugging modes are available in Sourcery G++ Kernel Debug (Attach) launch configurations. They are provided for debugging a running uClinux kernel only, using a hardware debug device. To debug ARM uClinux applications, you must use one of the debugging modes listed in Section 4.3.2, “Application Debugging Modes” instead.

Sourcery G++ Debug Sprite for ARM. The Sourcery G++ Debug Sprite for ARM is designed to debug ARM hardware connected to your host system using a supported debugging device, including ARMUSB, ULINK2, J-Link, and RDI devices. You can select the device you are connecting with, the type of target board you have, and any device-specific options using the ARM `Settings` subtab of the `Debugger` tab.

Macraigor OCDRemote. OCDRemote is a utility provided by Macraigor Systems that supports various JTAG/BDM debugging devices. The Sourcery G++ IDE launches OCDRemote automatically when you begin to debug, using the options you provide when you select this debugging mode.

Sourcery G++ External Embedded Server. On uClinux targets, the External Embedded Server mode is provided to support kernel debugging via third-party debug devices. In the External Embedded Server mode, Sourcery G++ connects to a “GDB stub” that controls execution on the target system. You must start the stub manually.

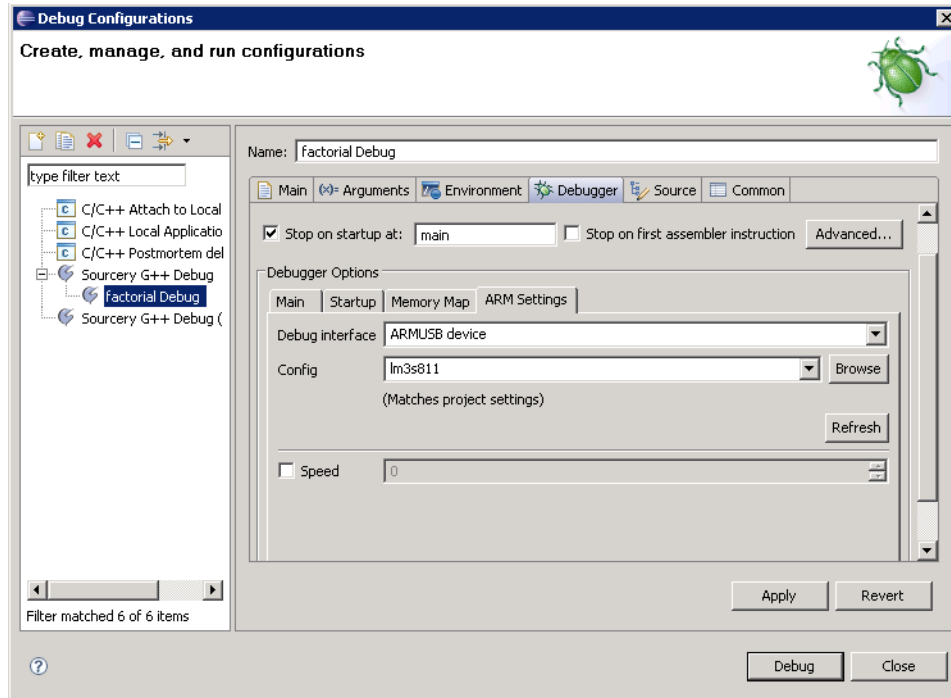
Detailed information about using each of these debugging modes is provided below.

4.3.3.1. Sourcery G++ Debug Sprite for ARM

Note

The Debug Sprite is provided for uClinux kernel debugging only, and requires that you select a Sourcery G++ Kernel Debug (Attach) launch configuration. You cannot use this debugging mode to debug user-space uClinux application programs.

Settings specific to the Debug Sprite for ARM can be found on the ARM `Settings` subtab of the `Debugger` tab. On that subtab, you can specify which device to use, which device initialization file to use, and, depending on the device, additional options relevant to that device.



ARM Settings. Use the ARM Settings subtab to configure the Sourcery G++ Debug Sprite for ARM uClinux.

The Device field allows you to select which debug device to use. When you select the Sprite debugger in the IDE, the Sprite probes for supported devices connected to your system. (This may take a few seconds.) You can then select one of the devices found on your system. Refer to Chapter 6, “Sourcery G++ Debug Sprite” for more information about the devices supported by this version of Sourcery G++, and for troubleshooting help if a device you have connected is not listed by the IDE.

The Config field allows you to specify a device initialization file that is used when connecting to the device. This initialization file should always be specified. You can either select from a list of predefined initialization files, or specify a custom initialization file by clicking the Browse button and selecting a file. By default, the IDE uses the initialization file for the board you previously selected in the project properties.

A debug device may have additional parameters that can be specified on the ARM Settings subtab. The parameters that apply to specific debug devices are documented in Chapter 6, “Sourcery G++ Debug Sprite”. Some parameters are always necessary and others are optional. To set the value for an optional parameter, first enable the parameter by selecting the corresponding checkbox, and then set its value.

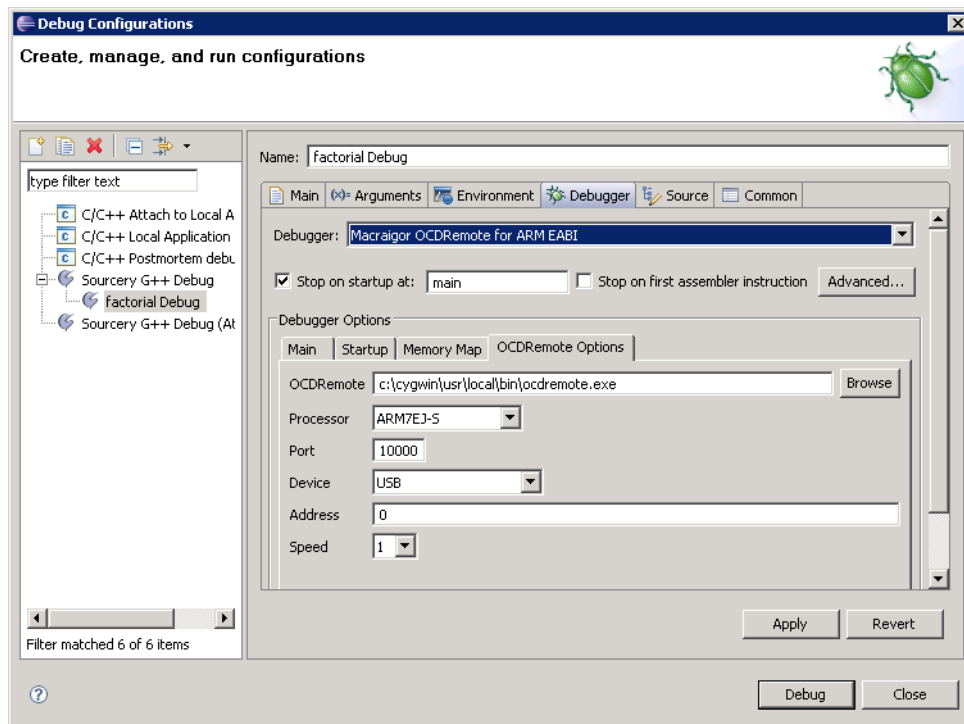
Refer to Section 4.3.4, “Tuning Debugger Behavior” for additional options you can set for this debugging mode.

4.3.3.2. Macraigor OCDRemote

Note

The OCDRemote debugger is provided for uClinux kernel debugging only, and requires that you select a Sourcery G++ Kernel Debug (Attach) launch configuration. You cannot use this debugging mode to debug user-space uClinux application programs.

When configuring the OCDRemote debugger option, you can click on the `OCDRemote Options` subtab to specify options that are passed to Macraigor OCDRemote. Refer to Macraigor's documentation for more information on the OCDRemote configuration parameters.



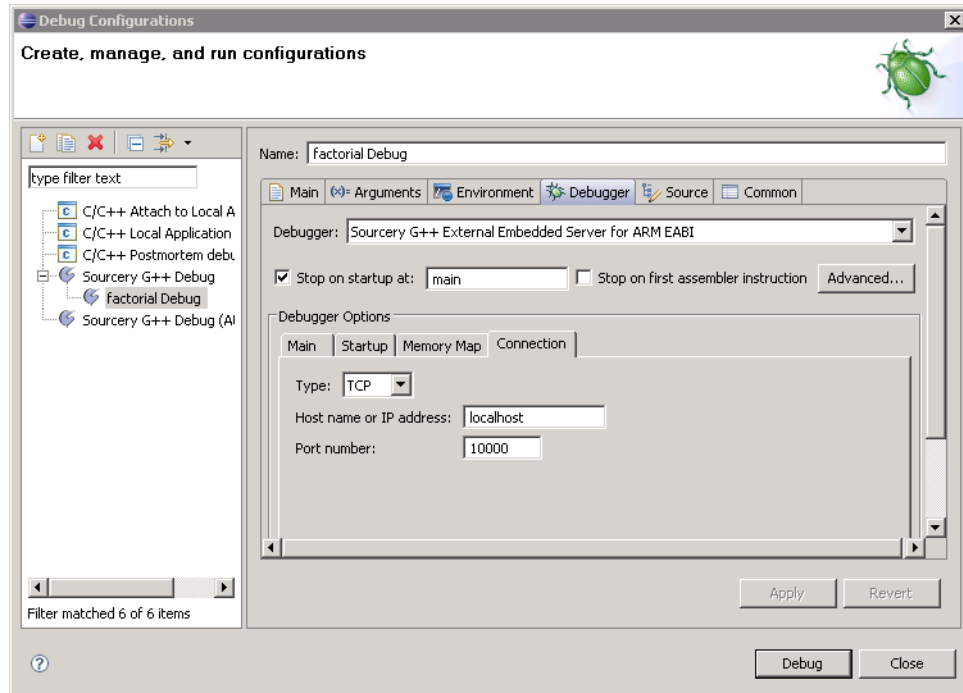
OCDRemote Options. Use the `OCDRemote Options` subtab to configure OCDRemote.

4.3.3.3. Sourcery G++ External Embedded Server

Note

External Embedded Server debugging mode is provided for uClinux kernel debugging only, and requires that you select a Sourcery G++ Kernel Debug (Attach) launch configuration. You cannot use this debugging mode to debug user-space uClinux application programs. Use the Sourcery G++ External Server debugger, described in Section 4.3.2.1, “Sourcery G++ External Server”, for application-mode debugging.

When using the External Embedded Server mode, you specify how the debugger connects to the target on the `Connection` subtab of the `Debugger` tab. The default connection is set to TCP connection to localhost, port 10000. You can adjust the host and port number, or you can select a serial line connection.



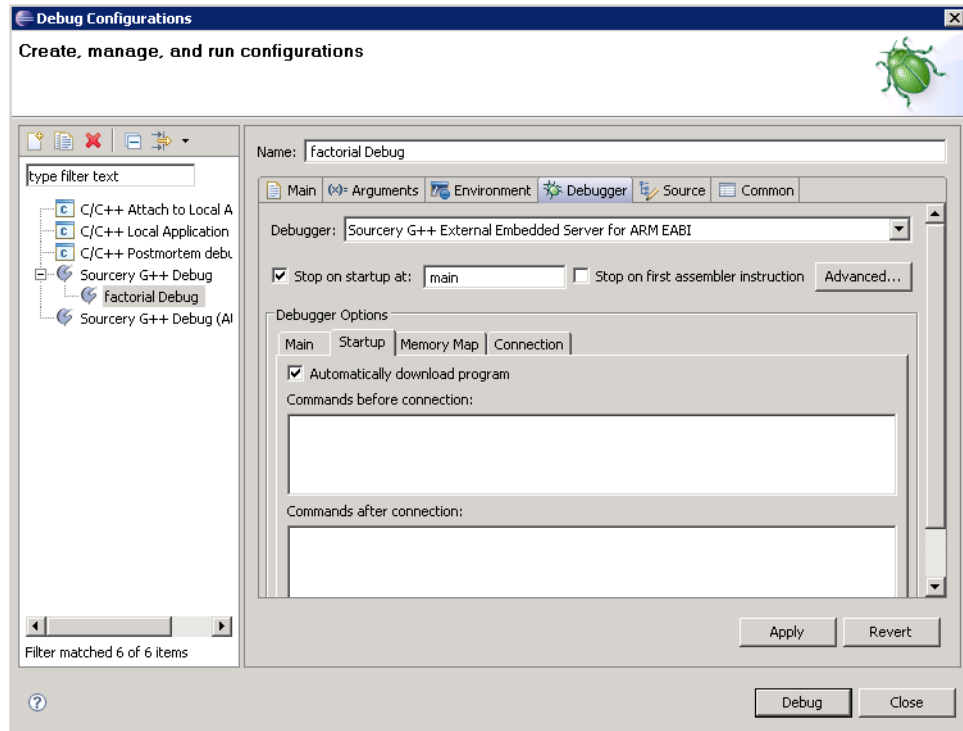
Embedded Server Connection. Use the `Connection` subtab to configure the External Embedded Server debugger.

Refer to Section 4.3.4, “Tuning Debugger Behavior” for additional options you can set for this debugging mode.

4.3.4. Tuning Debugger Behavior

4.3.4.1. Debugger Startup

Debugger startup consists of initialization or connection to a target, optional loading of the application (for non-native targets) and running the program. You can customize the startup process on the `Startup` subtab of the debugger dialog.



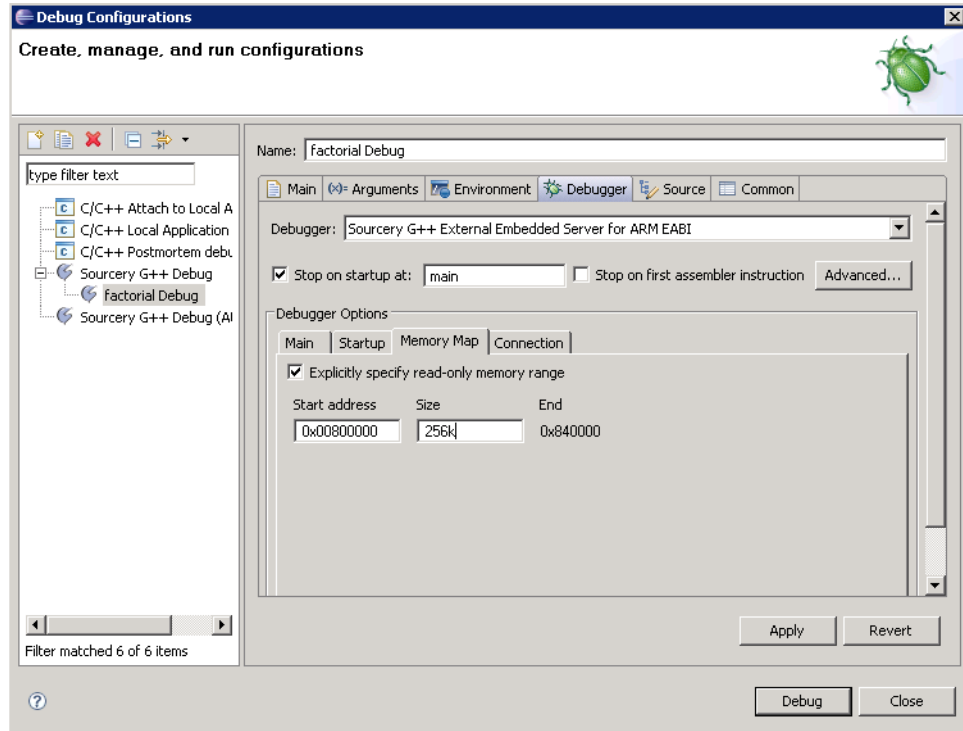
Customizing Debugger Startup. Use the Startup subtab to customize debugger startup actions.

The `Automatically download program` checkbox controls whether the IDE loads the program to the target during startup. If you are using GDB server on the target to run your program with the Sourcery G++ External Server debugger in the IDE, you should leave this option disabled since GDB server already loads the program specified on its command line.

The `Commands before connection` and `Commands after connection` text boxes allow you to specify debugger commands to be sent before and after connection. Each line in these text boxes is interpreted as a single GDB command.

4.3.4.2. Configuring the Memory Map

If your target has a read-only memory region, such as flash memory, you should supply a memory map. GDB uses this information to determine where it must use hardware breakpoints. GDB also flags writes to memory in the read-only region as errors.



Setting a Read-Only Memory Region. Use the `Memory Map` subtab to set a read-only memory region.

To specify a read-only memory range from the Sourcery G++ IDE, you can use the `Memory Map` subtab in the `Debugger` dialog. The `Explicitly specify read-only memory range` checkbox enables this option. The memory range is specified using the `Start address` and `Size` input fields. Both fields accept either decimal or hexadecimal (with the `0x` prefix) values. When entering decimal values, you can use the `K` and `M` suffixes to specify values in kilobytes (1024 bytes) and megabytes (1024 kilobytes). It is not possible to specify more than one memory range, or designate a memory range as a specific flash chip.

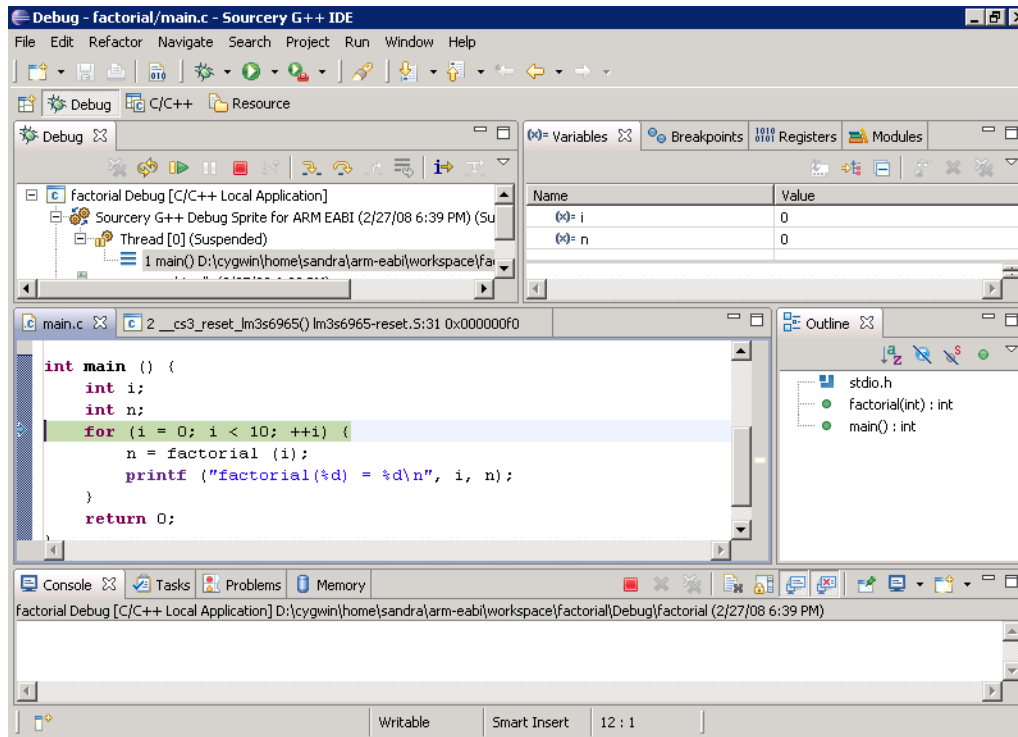
Once you have identified a read-only memory region, the debugger uses hardware breakpoints in this region automatically, and you can debug your program just as you would if it were in RAM.

4.3.4.3. Troubleshooting

When your application is large, or the debugging device is relatively slow, you may encounter timeout errors when starting debugging. In that case, you should increase the timeout settings. Select the `Preferences` item in the `Window` menu, and in the dialog that appears select `C/C++`, `Debug`, `GDB MI`. Increase the values in the `Debugger Timeout` and the `Launch Timeout` fields until your application starts without errors.

4.3.5. Controlling Execution

When you start the debugger, if you are asked whether to switch from the `C/C++` perspective to the debug perspective, click the `Yes` button. Instead of showing panes that help you to develop your application, the IDE now shows panes that help you to debug your application.

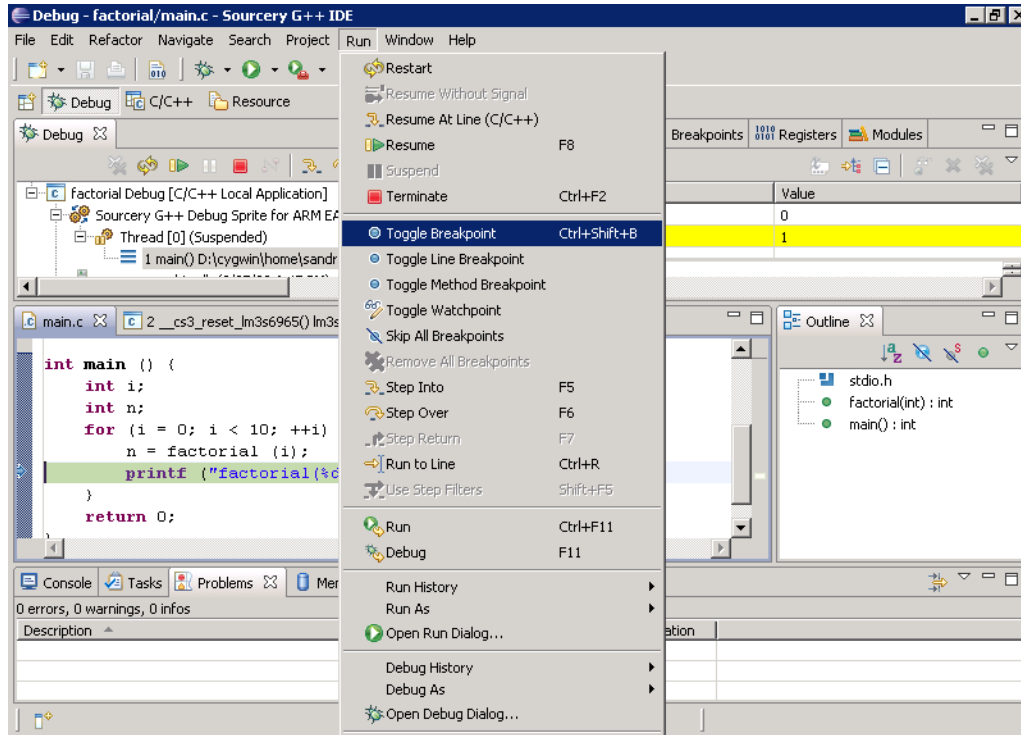


Debug Perspective. The debug perspective displays the stack, local variables, and the current location.

The debugger automatically stops on the first line of `main`. The currently active source line is highlighted. The pane at the upper left shows the application threads and the stack associated with each thread. The pane at the upper right shows the values of local variables. (At this point, `i` and `n` have not yet been initialized, so their values are indeterminate.)

Use `Run` → `Step Over (F6)` to advance by a single line. Because the program has changed the value of `i`, the IDE highlights the value in the variable pane.

By looking at the code, you can see that the program calls `factorial` and then calls `printf` to print out the resulting value. You can set a breakpoint right before the call to `printf` by clicking anywhere on that line, and then using `Run` → `Toggle Line Breakpoint (Ctrl+Shift+B)`.



Setting a Breakpoint. Set a breakpoint by highlighting the line where you want to stop and then using the Run menu.

After setting the breakpoint, use Run → Step Into (F5) to step into the body of `factorial`.

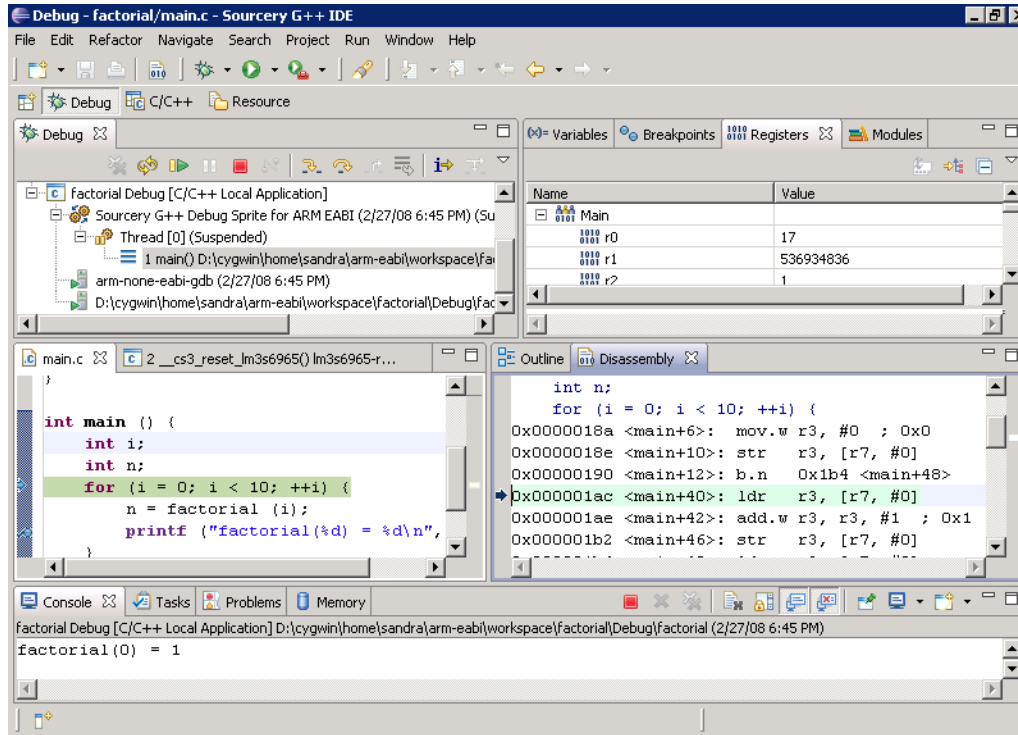
The IDE no longer displays the value of `i` because there is no local variable `i` within `factorial`. If you wish to see the value of `i` (from `main`), select the stack frame for `main` in the pane at the upper left. The IDE displays the variables for whichever frame is presently selected.

Now, proceed to the breakpoint by using Run → Resume (F8). The variable `n` now has the value 1 because the factorial of zero is one. Step over the call to `printf` to print the value in the console.

4.3.6. Low-Level Debugging

You may sometimes need to debug at the machine level, rather than at the source code level. For example, if you are working with an assembly code device driver, you may wish to see the values stored in machine registers and step through the code instruction by instruction.

To view machine registers, click on the Registers tab, and expand the Main register group. To see the instructions being executed, use Window → Show View → Disassembly.



Low-Level Debugging. The Sourcery G++ IDE can display machine registers and assembly code.

When the focus is on the disassembly window, the `Step Over` and `Step Into` commands operate at the assembly level, rather than at the source code level. So, a `Step Over` command advances by a single machine instruction. When the values of registers change, the registers are highlighted in the IDE. You can set breakpoints on particular machine instructions in the same way that you can set breakpoints on source code.

4.4. Advanced IDE Features

This section covers several advanced features of the Sourcery G++ IDE, including creating Makefile projects, building managed build projects from the command line, importing source code and already-compiled programs into the IDE, and using other programming tools provided by the IDE.

4.4.1. Makefile Projects

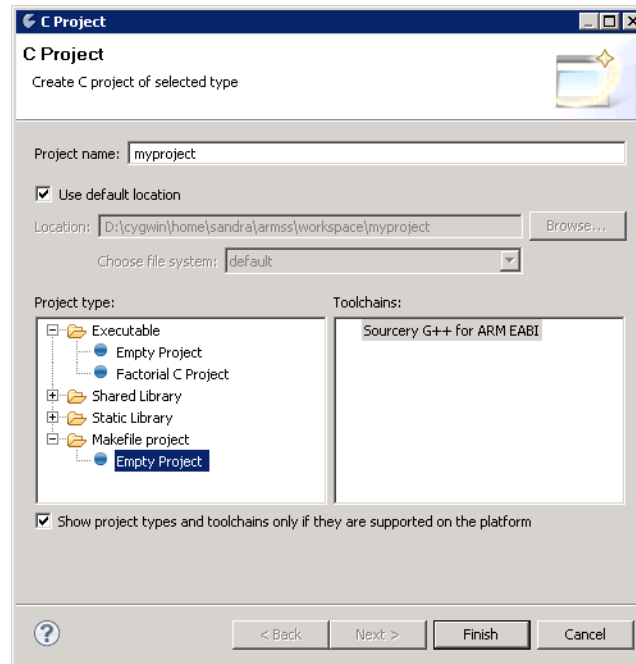
This section explains how to use the advanced Makefile project mode, instead of the simpler managed build mode described in Section 4.2, “Building Applications”.

Caution

Building a Makefile project requires that you manually maintain information about how your program is built. If you use this mode, you need to be familiar with the **make** utility.

If you want to import an existing project for use with the Sourcery G++ IDE, and that project uses **make**, or some similar command-line tool to manage the build process, you should use a Makefile project, instead of the IDE's built-in project types. In Makefile project mode, the IDE invokes **make** (or an alternative program that you specify) to build your program. If you add new files to your project, you have to manually update the `Makefile` for your project.

To create a new Makefile project in the Sourcery G++ IDE, open the New Project Wizard as described in Section 4.2.1, “Setting Up an Example Project”. This time, however, expand `Makefile project` under `Project type:`, and choose `Empty Project`. Under `Toolchain`, select `Sourcery G++ for ARM uClinux`. Then click `Finish`.



Creating a Makefile Project. Use the New Project Wizard to create a new empty Makefile project.

Before you can build your project, you must add a `Makefile` to it as well as your `C` or `C++` source files. You can create new files by right-clicking on the project name in the left-hand pane and selecting `New`. To import existing files into your project, right-click on the project name and select `Import...` In the `Import Wizard`, expand `General` and select `File System`. Then continue to the next page to select the files to import.

You may also have to adjust your `Makefile` to use Sourcery G++. For example, you might need to set the `CC` variable in your `Makefile` to `arm-uclinuxeabi-gcc`.

4.4.2. Building IDE Projects from the Command Line

If you need to be able to do batch-mode builds of your project — for example, for nightly builds or automated product packaging scripts — you can invoke the Sourcery G++ IDE builder from the command line. This is an alternative to converting your managed build project to a `Makefile` project so that it can be built outside the IDE.

The basic recipe is

```
> sourcerygxx-ide -data workspace -cleanBuild project
```

where `workspace` is the pathname of your workspace and `project` is the name of the project to build. You can also specify the literal value `all` to build all the projects in the workspace.

The above command does a full rebuild of the project, the equivalent of cleaning the project and then building it. To do an incremental build without cleaning, use `-build` rather than `-cleanBuild`:

```
> sourcerygxx-ide -data workspace -build project
```

You can also import projects into your workspace using the `-import` option:

```
> sourcerygxx-ide -data workspace -import project-uri
```

Here `project-uri` is a full pathname or URI of the project to import. The effect of this option is similar to using the Import wizard's Existing Projects into Workspace feature.

The `-import` option can be repeated and used in combination with `-build` or `-cleanBuild`. For example, you can import multiple projects into a new temporary workspace and use `-cleanBuild all` to build all of them with a single command.

Output from the command-line builder, which would be directed to a console window when building in the interactive IDE, is sent to standard output. If there are errors in the build command itself (such as specifying a project that doesn't exist), they are logged to standard error.

Command-line managed build support is a relatively new feature, and there are still some limitations in the underlying Eclipse support. In particular:

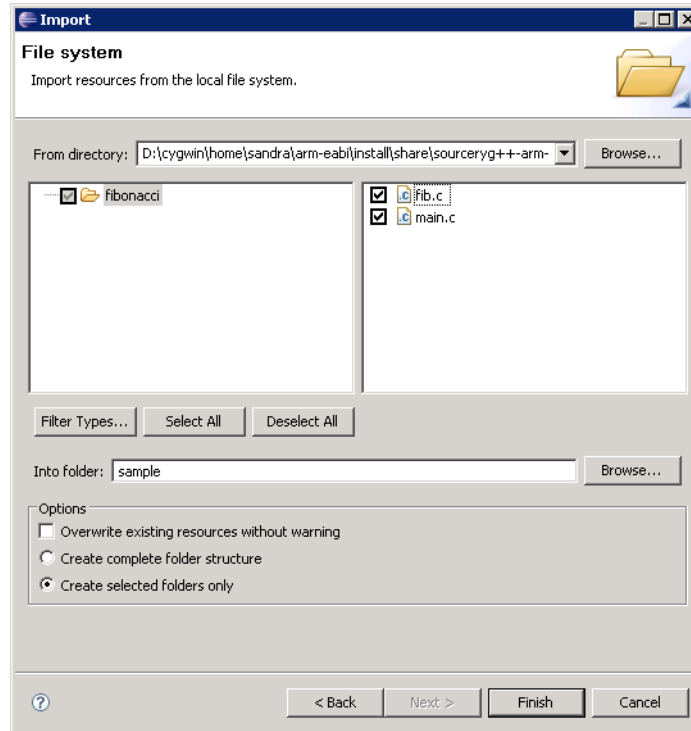
- Failed builds do not return an appropriate error status value to the shell.
- Only the active configuration of the selected projects (e.g., either the Debug or Release configuration) is built. It is not possible to select the configuration to build on the command line.

4.4.3. Importing Example Programs into the IDE

Sourcery G++ comes with some simple applications. You can import the source code for the sample programs into an IDE project.

Start by following the steps described in Section 4.2, “Building Applications” to create a new Executable project. Call the project `sample`, and do not create a new source file.

At this point, the `sample` project exists, but there is no associated source code. The next step is to import a file from the example directory. Right-click on the `sample` project in the Project Explorer tab, and select `Import...` Select `General` → `File System`, and click `Next`. Click on `Browse...` beside the `From directory:` edit box. Navigate to the Sourcery G++ install directory and then to `share/sourceryg++-arm-uclinuxeabi-examples/fibonacci`, and click `Ok`. Click the checkbox beside `main.c` and `fib.c`, then click `Finish`.



Importing a Source File. Click `Finish` to import the selected file.

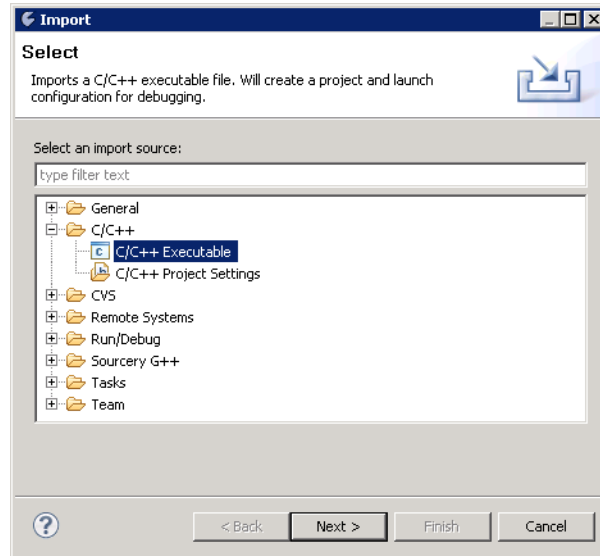
Now follow the instructions in Section 4.2, “Building Applications” to build the project. Your sample program is now ready to execute or debug. Please refer to Section 4.3, “Debugging Applications” for instructions on how to debug the target application.

4.4.4. Importing an Executable into the Sourcery G++ IDE

You can use the Sourcery G++ IDE to debug an executable program you have built outside the IDE. You may need to do this to debug a program that requires a build procedure that cannot be simply expressed in a `Makefile`, a program built with a non-Sourcery G++ compiler, or a binary provided by a third party for which you do not have complete source code. For example, you can use this procedure to import a uClinux kernel into the IDE.

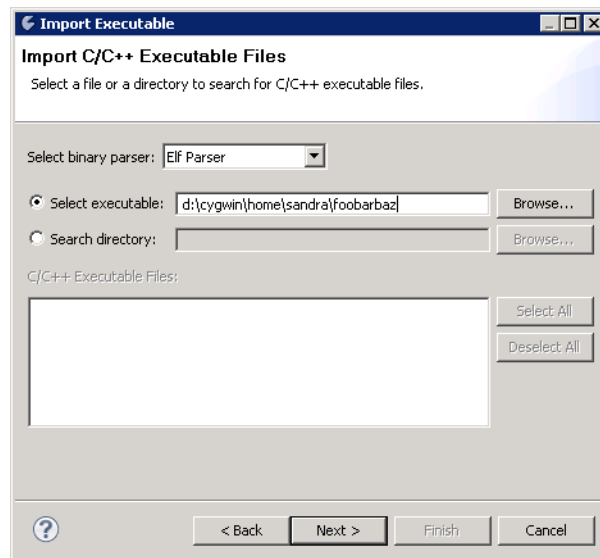
For best results in the debugger, you should build your application with debugging enabled (via the `-g` option) if possible prior to importing it. The IDE can use the debugging information in the imported executable to find and display the source code for the program as you debug it, even though the source files are not part of the project.

To import a program, select `File` → `Import...` This opens the Import Wizard. Expand `C/C++` and select `C/C++ Executable`. Then click `Next`.



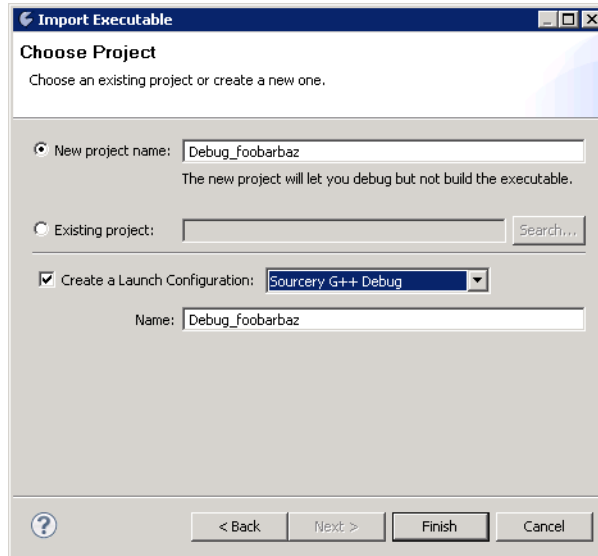
Import Wizard. Select C/C++ Executable.

On the next screen, use the **Browse** button to select the program you wish to debug. Make sure the binary parser is set to **Elf Parser**, which is the default. Then click **Next**.



Choose Executable to Import. Import the executable file you wish to debug.

Finally, choose a name for the project, and click **Finish**. At this point, you can continue to set up a debug launch configuration for your program as described in Section 4.3, “Debugging Applications”.

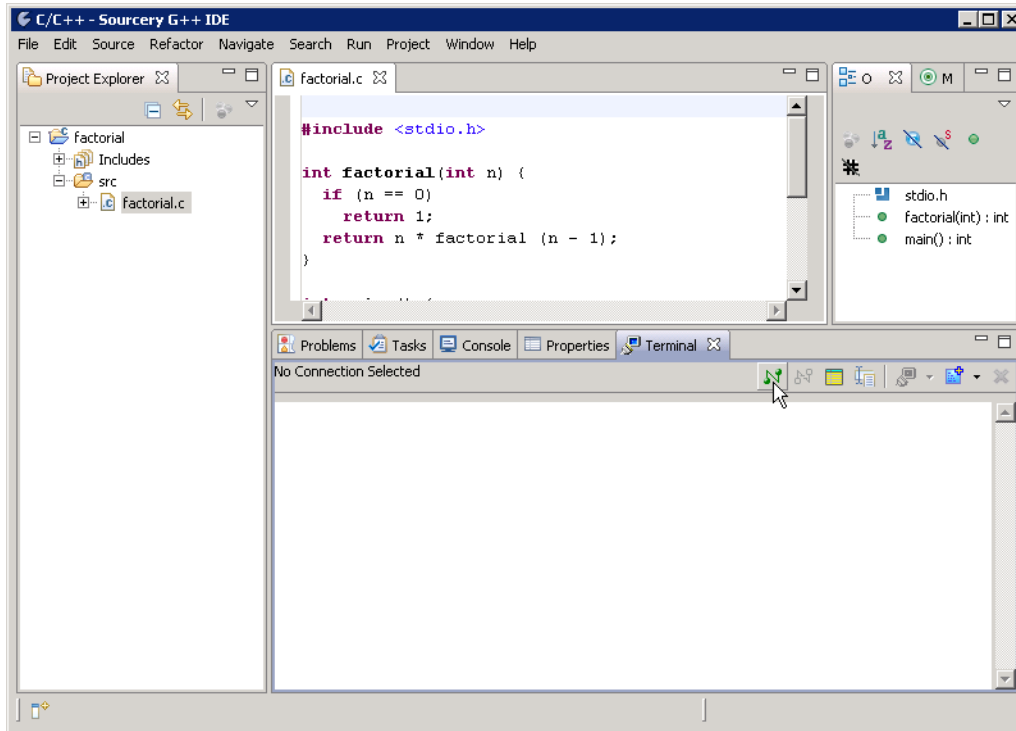


Choose Project for Import. Give the new project a name.

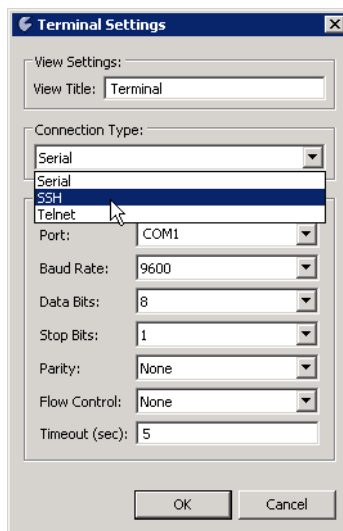
4.4.5. Using the Terminal Emulator

The Sourcery G++ IDE includes a built-in terminal emulator which supports serial, Telnet, and SSH connections. For example, you can use the terminal to connect to a serial console or UART driver on your target board. The Remote System Explorer, discussed below, also includes an SSH terminal feature, but for serial connections you must use the basic terminal as described in this section.

To open a terminal connection, first select `Window → Show View → Other...` from the top menu. Then choose `Terminal → Terminal` in the dialog box. This opens a new tab for the Terminal view. Click on the new connection icon in the toolbar for this tab, and use the pop-up dialog to configure the connection.



Opening a Terminal. Click on the new connection icon in the Terminal tab.



Terminal Settings. Configure the terminal properties in the pop-up dialog.

4.4.6. Using the Remote System Explorer

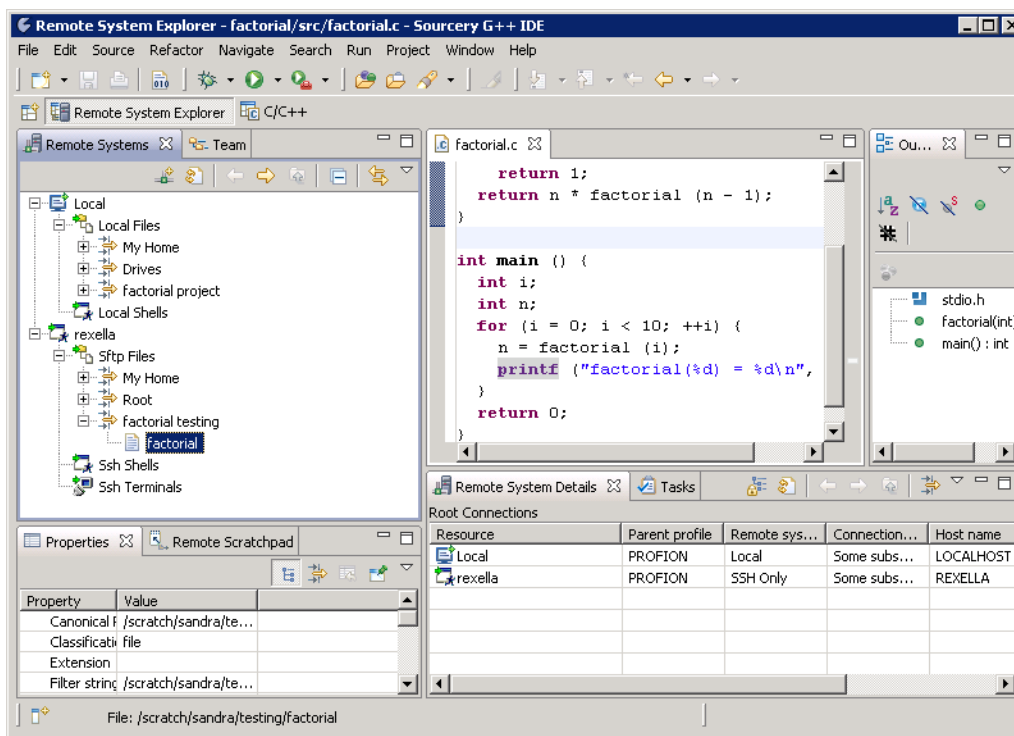
You can use the Remote System Explorer (RSE) to browse files on your remote ARM uClinux target from within the Sourcery G++ IDE. A typical use of RSE is to copy your application to the target so that you can run or debug it using **gdbserver** in a terminal window.

To use RSE, first set up the connection to the remote host. The discussion here assumes you have configured your target to accept incoming SSH connections via TCP/IP. RSE supports other connection protocols as well; refer to the online documentation for additional details.

Open the RSE perspective by choosing `Window → Open Perspective → Other...` from the top menu and then selecting `Remote System Explorer` in the dialog box. Click on the new connection icon in the toolbar in the `Remote Systems` tab. This starts the `New Connection` wizard. Choose `SSH Only` as the system type, and fill in the remote system's host name on the next screen. Then click `Finish`.

The remote system is added to the `Remote Systems` browser tab. Within this tab, you can perform the usual file browser operations. For example, you can copy files from your local system to the remote target system using a copy/paste sequence, open files on the remote system in an editor window, or view their properties.

To keep the lists of files to a more manageable size, create a filter for your working directories on the local and remote systems by right-clicking on the `Local Files` or `Sftp Files` item and choosing `New → Filter...`



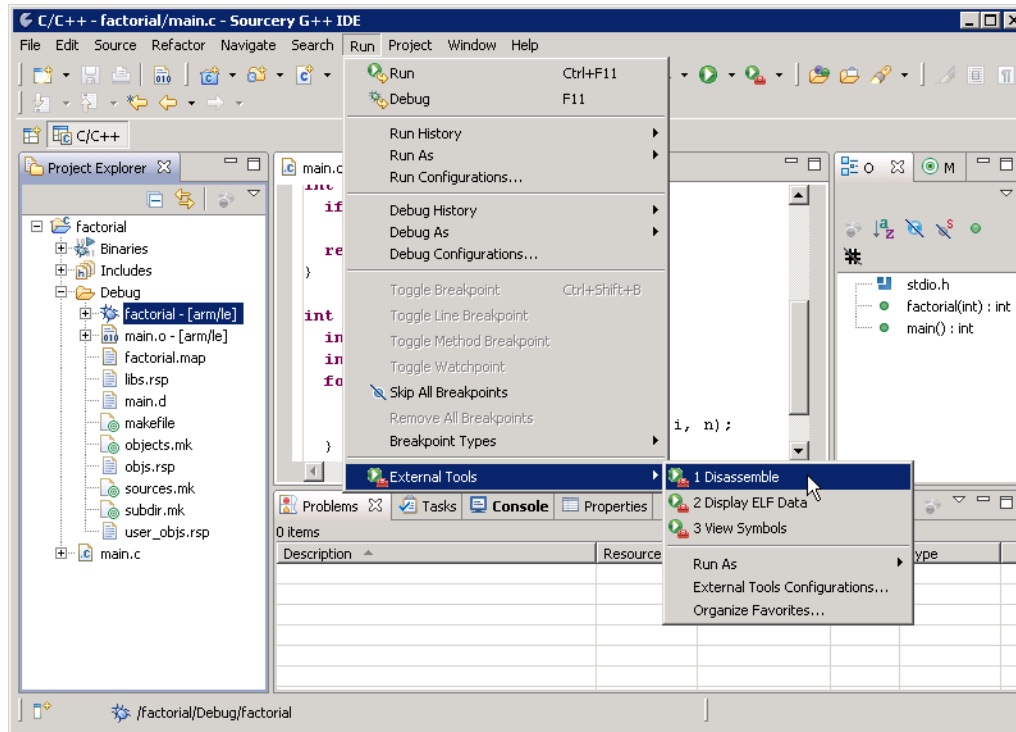
Remote System Explorer File Browser. Use RSE to browse files on your remote target system.

You can also open a terminal to your remote system using an RSE connection. Right-click on the `Ssh Terminals` item and choose `Launch Terminal`. The terminal opens in a new tab.

4.4.7. Using External Tools

The Sourcery G++ IDE includes some predefined External Tool launches to run common informational queries using the GNU Binary Utilities. To use these tools, first select the object file or executable in the `Project Explorer` tab. Then, choose `Run → External Tools` from the top

menu. The list of available tools includes Disassemble (which runs the **objdump -d** command), Display ELF Data (**readelf**), and View Symbols (**nm**).



External Tools. The Sourcery G++ IDE provides built-in tools for displaying properties of compiled files.

You can modify the command-line arguments passed to these tools or add your own custom External Tool launches by selecting `External Tools Configurations...` from this submenu. To create a new launch, select `Program` in the left-hand pane of the dialog and click the `New` icon. Use the predefined External Tool launches as a guide to filling in the properties of your new launch.

4.4.8. Using Eclipse Plugins in the Sourcery G++ IDE

Eclipse plugins provide many additional tools and utilities to extend the functionality of the Sourcery G++ IDE. Plugins that are bundled with the IDE include:

CVS. This plugin provides direct integration with the CVS version control system. You can check out a project from a CVS repository into your workspace by choosing `File` → `Import...` from the top menu and then `CVS` → `Projects from CVS` in the Import Wizard. Additional CVS operations are available by right-clicking on the project in the `Project Explorer` tab and selecting the `Team` submenu. For additional documentation, visit the CVS plugin web site².

Mylyn. This plugin provides task management features for Eclipse, including integration with external bug trackers as well as personal to-do lists. Mylyn also provides a task-focused way to organize your workspace, so that you can easily switch an entire set of active views and resources when you switch tasks. Visit the Mylyn project web site³ for tutorials and documentation.

² <http://www.eclipse.org/eclipse/platform-cvs/>

³ <http://www.eclipse.org/mylyn/>

In addition to these pre-installed plugins, there are many others available that you can install into the Sourcery G++ IDE yourself. You can find a directory of available plugins, sorted by category, on the Eclipse Plugin Central⁴ web site.

⁴ <http://www.eclipseplugincentral.com/>

Chapter 5

Using Sourcery G++ from the Command Line

This chapter demonstrates the use of Sourcery G++ from the command line. If you prefer to use an integrated development environment to build your applications, you may refer to Chapter 4, “Using the Sourcery G++ IDE” instead.

5.1. Building an Application

This chapter explains how to build an application with Sourcery G++ using the command line. As elsewhere in this manual, this section assumes that your target system is `arm-uclinuxeabi`, as indicated by the **`arm-uclinuxeabi`** 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:

```
> arm-uclinuxeabi-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 **`arm-uclinuxeabi-gcc`** with **`arm-uclinuxeabi-g++`**.)

5.2. Running Applications on the Target System

To run your program on a uClinux 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
```


5.3. 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.

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.

5.3.1. 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 6, “Sourcery G++ Debug Sprite” for detailed information about the supported devices.

You can start the Sprite directly from within GDB:

```
(gdb) target remote | arm-uclinuxabi-sprite arguments
```

Refer to Section 6.2, “Invoking Sourcery G++ Debug Sprite” for a full description of the Sprite arguments.

5.3.2. Connecting to an External GDB Server

Sourcery G++ includes a program called **gdbserver** that can be used to debug a program running on a remote ARM uClinux target. Follow the instructions in Chapter 3, “Sourcery G++ for ARM uClinux” 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 6

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 or uClinux kernel on the target board. This chapter includes information about the debugging devices and boards supported by the Sprite for ARM uClinux.

Sourcery G++ contains the Sourcery G++ Debug Sprite for ARM uClinux. 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 6.2, “Invoking Sourcery G++ Debug Sprite” for information about the specific devices supported by this version of Sourcery G++.

The Sourcery G++ Debug Sprite also supports programming of flash memory on the target. When used with an appropriate linker script and board configuration, flash programming is automatic when you load your program in the debugger.

Note for Linux/uClinux users

The Debug Sprite provided with Sourcery G++ allows remote debugging of the Linux or uClinux kernel running on the target. For remote debugging of application programs, you should use **gdbserver** instead. See Chapter 3, “Sourcery G++ for ARM uClinux” 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.

6.1. 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:

```
> arm-uclinuxeabi-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:

```
CodeSourcery ARM Debug Sprite
(Sourcery G++ Sourcery G++ 4.4-61)
armusb: [speed=<n:0-7>] ARMUSB device
armusb:///0B01000C - Stellaris Evaluation Board (0B01000C)
rdi: (rdi-library=<file>&rdi-config=<file>) RDI Device
rdi:/// - RDI Device
```

This shows that ARMUSB and RDI devices are supported. The exact set of supported devices depends on your host system and the version of Sourcery G++ you have installed; refer to Section 6.2, “Invoking Sourcery G++ Debug Sprite” for complete information.

Note that it may take several seconds for the Debug Sprite to probe for all types of supported devices.

6.2. Invoking Sourcery G++ Debug Sprite

The Debug Sprite is invoked as follows:

```
> arm-uclinuxeabi-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++ for ARM uClinux:

- rdi** Use an RDI debugging device. Refer to Section 6.4, “Remote Debug Interface Devices”.
- flashpro** Use a FlashPro debugging device. Refer to Section 6.5, “Actel FlashPro Devices”.
- ulink** Use a Keil ULINK2 debugging device. Refer to Section 6.6, “Keil ULINK2 Devices”.
- jlink** Use a SEGGER J-Link. Refer to Section 6.7, “SEGGER J-Link 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 6.9, “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 6.10, “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.

6.3. 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 | arm-uclinuxeabi-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.

6.4. Remote Debug Interface Devices

Remote Debug Interface (RDI) devices are supported. The RDI device URL accepts no hostname, port or path components, so the *device-url* is specified as follows:

```
rdi:[:///][?device-options]
```

The following *device-options* are required:

- `rdi-library=library` Specify the library (DLL or shared object) implementing the RDI target you wish to use.
- `rdi-config=configfile` Specify a file containing configuration information for *library*. The format of this file is specific to the RDI library you are using, but tends to constitute a list of *key=value* pairs. Consult the documentation of your RDI library for details.

6.5. Actel FlashPro Devices

On Windows hosts, Sourcery G++ supports FlashPro devices used with Actel Cortex-M1 development kits.

For FlashPro devices, the *device-url* has the following form:

```
flashpro:[//usb12345/][?jtagclock=rate]
```

The optional *usb12345* part indicates the ID of the FlashPro device to connect to, which is useful if you have more than one such device attached to your computer. If the ID is omitted, the Debug Sprite connects automatically to the first detected FlashPro device. You can enumerate the connected FlashPro devices by invoking the Sprite with the -i switch, as follows:

```
> arm-uclinuxeabi-sprite -i flashpro:
```

The *jtagclock* option allows the communication speed with the target board to be altered. The *rate* is specified in Hz and may range between 93750 and 4000000. The default is 93750, the slowest speed supported by the FlashPro device. Depending on your target board, you may be able to increase this rate, but beware that communication errors may occur above a certain threshold. If you encounter communication errors with a higher-than-default speed selected, try reducing the speed.

6.5.1. Installing FlashPro Windows drivers

Windows drivers for the FlashPro device are included with the FlashPro software provided by Actel. Refer to Actel's documentation for details on installing this software. You must use the Actel FlashPro software to configure the FPGA on your Cortex-M1 board, but it does not need to be running when using the Debug Sprite.

Once you have set up your board using the FlashPro software, you can check that it is recognized by the Sourcery G++ Debug Sprite by running the following command:

```
> arm-uclinuxeabi-sprite -i
flashpro: [jtagclock=<n:93750-400000>] FlashPro device
flashpro://usb12345/ - FlashPro Device
...
```

If output similar to the above does not appear, your FlashPro device is not working correctly. Contact CodeSourcery for further guidance in that case.

6.6. Keil ULINK2 Devices

Keil ULINK2 devices are supported on Windows hosts. For Cortex-M targets (e.g. Cortex-M1, Cortex-M3) the ULINK2 device partitions the *device-url* as follows:

```
ulink://cm/?opts=file
```

For older ARM targets (e.g. ARM7TDMI, ARM9), use the following instead:

```
ulink://arm/?opts=file
```

The *opts* option is mandatory, and is used to specify an options file. See Section 6.6.1, “Configuring ULINK2 Options” for instructions on creating this file.

6.6.1. Configuring ULINK2 Options

Before you can use the ULINK2 device for debugging, you must configure various JTAG and flash properties for your board. A graphical user interface is provided to assist you with this configuration.

When invoking the Debug Sprite from the command line, you must perform this configuration step first, and save the settings in an options file that you pass to the Sprite for debugging.

If you are using the Sourcery G++ IDE, there are buttons on the ARM *Settings* subtab that bring up the configuration dialogs described in this section. You do not need to run the Debug Sprite from the command line to create an options file, or specify an options file explicitly in the IDE.

Start by setting the JTAG properties. In the Sourcery G++ IDE, click on the corresponding button. From the command line, this dialog is opened by invoking the Sourcery G++ Debug Sprite with a *config* option. For Cortex-M targets, use:

```
> arm-uclinuxeabi-sprite "ulink://cm/?opts=file&config=debug"
```

For older ARM targets, use:

```
> arm-uclinuxeabi-sprite "ulink://arm/?opts=file&config=debug"
```

Replace *file* with the name of the options file to create. Note that you need the double quotes to prevent the shell from treating & as a special character.

This opens a dialog box on your desktop, which allows you to configure various debugging options.

- You can change the communication speed with your target board. Beware that rates above 1MHz may not work reliably.
- The `Cache Code` and `Cache Memory` options in the `Cache Options` group may speed up successive memory read operations from your board. It is safe to leave these two options checked.
- The `Verify Code Download` and `Download to Flash` options in the `Download Options` group should be left checked.
- The `Use Reset at Startup` option should be left checked.
- The `JTAG Device Chain` group can be used to override the default processor core detected by ULINK2. Leaving it set to `Automatic Detection` is usually the right thing to do, unless you have special requirements.

Clicking OK saves the configured options and closes the dialog.

To use flash programming support with ULINK2, you must configure a separate set of options for this. In the Sourcery G++ IDE, use the `ULINK Flash properties` button. From the command line, invoke the Sprite again with the appropriate `config` option. For Cortex-M targets, use:

```
> arm-uclinuxeabi-sprite "ulink://cm/?opts=file&config=flash"
```

For older ARM targets, use:

```
> arm-uclinuxeabi-sprite "ulink://arm/?opts=file&config=flash"
```

Use the same *file* that you previously used to store the JTAG debug configuration information.

This opens another dialog box on your desktop. There are several options which you can configure:

- In the `Download Function` group, leave the radio buttons for erase behavior set to `Erase Sectors`. Check the `Program` and `Verify` options, but leave the `Reset` and `Run` option unchecked.
- Add the flash device suitable for your target board by clicking `Add`, then choosing your device from the pop-up list.
- In the `RAM for Algorithm` group, choose a suitable RAM area for use as scratch space during flash programming.

Click OK to save your flash programming options.

6.6.2. ULINK2 Target Boards

ULINK2 supports many target boards and processors; refer to Keil documentation for a full list. However, only a limited number of boards are currently supported out-of-the-box by Sourcery G++. See Section 6.9, “Supported Board Files”. If you would like to use a particular development board which is supported by ULINK2 but not by Sourcery G++, please contact CodeSourcery for further information.

6.6.3. Installing ULINK2 Windows Drivers

No special driver is needed for ULINK2 devices.

6.7. SEGGER J-Link Devices

The Debug Sprite currently supports ARM7TDMI, ARM9 and Cortex-M3 cores via SEGGER J-Link on Microsoft Windows hosts. The *device-url* is:

```
jlink://
```

You must install drivers for the SEGGER J-Link device before you can use it to debug your program. These drivers are bundled with Sourcery G++ for your convenience. You can find them at `libexec/arm-uclinuxeabi-post-install/sprite-drivers/Setup_JLinkARM_version.exe` in your Sourcery G++ installation directory. Simply execute the file and follow instructions from the installer.

Flash programming is supported on CFI-compliant devices, Stellaris LM3Sxxx, NXP LPC21xx, STMicroelectronics STM32F10xxx, and STMicroelectronics STR91xFA.

The Debug Sprite does not require or utilize additional software components from SEGGER, such as J-Link ARM FlashBP, J-Link ARM FlashDL, or J-Link GDB Server.

6.8. 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++ and a valid license 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:

```
> arm-uclinuxeabi-sprite -l :10000 device-url board-file
```

starts the Sprite listening on port 10000.

If you are using the Sourcery G++ IDE, you can use the External Embedded Server debugger to connect to the remote Sprite; Section 4.3.3.3, “Sourcery G++ External Embedded Server”. 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 Base¹.

¹ <https://support.codesourcery.com/GNUToolchain/kbentry132>

6.9. Supported Board Files

The Sourcery G++ Debug Sprite for ARM uClinux 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
Actel CoreMP7 Cortex-M1	coremp7-cm1
Altera Cyclone III Cortex-M1	cycloneiii-cm1
ARMulator (RDI)	armulator
Freescale i.MX31 ADS	imx31
Keil MCBSTM32	mcbstm32
RealView EB Cortex-M1	realview-cm1

6.10. 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 `arm-uclinuxeabi/lib/boards` directory in the installation. Refer to the files in that directory for examples.

The file's DTD is:

```
<!-- 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.

It can also contain **<property>** elements with the following names:

<code>banked-regs</code>	The <code>banked-regs</code> property specifies that the CPU of the target board has banked registers for different processor modes (supervisor, IRQ, etc.).
<code>has-vfp</code>	The <code>has-vfp</code> property specifies that the CPU of the target board has VFP registers.
<code>system-v6-m</code>	The <code>system-v6-m</code> property specifies that the CPU of the target board has ARMv6-M architecture system registers.
<code>system-v7-m</code>	The <code>system-v7-m</code> property specifies that the CPU of the target board has ARMv7-M architecture system registers.
<code>core-family</code>	The <code>core-family</code> property specifies the ARM family of the target. The body of the <code><property></code> element may be one of <code>arm7</code> , <code>arm9</code> , <code>arm11</code> , and <code>cortex</code> .
<code><initialize></code>	The <code><initialize></code> element defines an initialization sequence for the board, which the Sprite performs before downloading a program. It can contain <code><write-register></code> , <code><write-memory></code> and <code><delay></code> elements.
<code><feature></code>	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.
<code><memory-map></code>	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 <code><memory-device></code> elements.
<code><memory-device></code>	This element specifies a region of memory. It has four attributes: <code>address</code> , <code>size</code> , <code>type</code> and <code>device</code> . The <code>address</code> and <code>size</code> attributes specify the location of the memory device. The <code>type</code> attribute specifies that device as <code>ram</code> , <code>rom</code> or <code>flash</code> . The <code>device</code> attribute is required for flash regions; it specifies the flash device type. Supported flash device types include <code>at91sam7sxxx</code> , <code>cfi</code> , <code>lpc21xx</code> , <code>stellaris</code> , <code>stm32f10xxx</code> , <code>str91xfa</code> , and <code>ulink</code> ; not all flash devices are supported by all debugging devices. The <code><memory-device></code> element can contain a <code><description></code> element.
	It can also contain the following named <code><property></code> element for additional flash-specific information:
<code>programaddress</code>	This numeric property is used for <code>ulink</code> flash devices. It specifies an alias for the flash region in the memory map that should be used for programming the flash device, independently of the address the CPU uses.
<code>system-clock</code>	This numeric property is used for <code>lpc21xx</code> and <code>stellaris</code> flash devices. It specifies the target

frequency, and is used to determine the flash frequency divider value.

- `<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 7

Next Steps with Sourcery G++

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

7.1. Sourcery G++ Subscriptions

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

If you have a Sourcery G++ subscription, you may access your account by visiting the Sourcery G++ Portal¹. If you have a support account, but are unable to log in, send email to <support@codesourcery.com>.

7.2. Sourcery G++ Knowledge Base

The Sourcery G++ Knowledge Base is available to registered users at the Sourcery G++ Portal². 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.

7.3. Manuals for GNU Toolchain Components

Sourcery G++ 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++, 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++, the documentation can be found in the `share/doc/sourceryg++-arm-uclinuxeabi/` subdirectory of your installation directory.

You can also access the manuals from the `Help` menu in the Sourcery G++ IDE. See Section 7.4, “Help for the Sourcery G++ IDE”, below.

In addition to the detailed reference manuals, Sourcery G++ 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++-arm-uclinuxeabi/man/man1
```

Then you can invoke `man` as:

```
> man ./arm-uclinuxeabi-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

¹ <https://support.codesourcery.com/GNUToolchain/>

² <https://support.codesourcery.com/GNUToolchain/>

your `.profile` or equivalent shell startup file; see Section 2.7, “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++ 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.

7.4. Help for the Sourcery G++ IDE

The Sourcery G++ IDE, which is based on Eclipse and its C/C++ Development Toolkit, includes an extensive online help facility. To access this information, select `Help Contents` from the `Help` menu in the IDE toolbar.

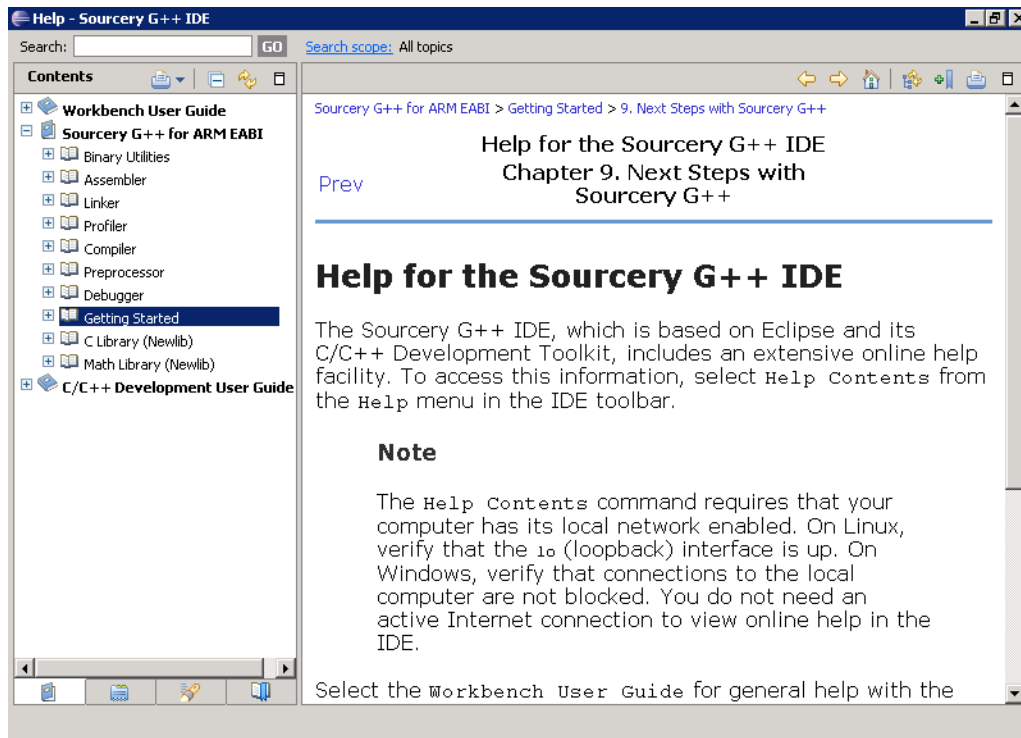
Note

The `Help Contents` command requires that your computer has its local network enabled. On Linux, verify that the `lo` (loopback) interface is up. On Windows, verify that connections to the local computer are not blocked. You do not need an active Internet connection to view online help in the IDE.

Select the `Workbench User Guide` for general help with the Eclipse IDE. Topics discussed in this manual include using the editor, file operations, and managing views and bookmarks.

Select the `C/C++ Development Toolkit User Guide` for assistance with using project templates, importing existing projects into the IDE, using Makefile mode, using and customizing features of the editor specific to C and C++ code, and using the debugger GUI.

Select `Sourcery G++ for ARM uClinux` to view the Getting Started Guide and manuals for GNU toolchain components included with this version of Sourcery G++.



IDE Help. Sourcery G++ manuals are available in the IDE from the `Help Contents` menu command.

Appendix A

Sourcery G++ Release Notes

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

A.1. Changes in Sourcery G++ for ARM uClinux

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

A.1.1. Changes in Sourcery G++ 4.4-61

GDB crash fix. A GDB bug has been fixed that caused GDB to crash when unloading shared libraries or switching executables.

IDE terminal emulator and remote file system browser . The Sourcery G++ IDE now includes a bundled terminal emulator. You can use the terminal to connect to a serial console or UART on your target device. The terminal also supports SSH and Telnet connections. In addition, the IDE now includes the Remote System Explorer (RSE). You can use the RSE file browser to transfer files between your host and target systems, or edit files that reside on the remote target from within the IDE. For more information, refer to Section 4.4.5, “Using the Terminal Emulator” and Section 4.4.6, “Using the Remote System Explorer”.

Project build errors on Windows hosts. A bug in the Sourcery G++ IDE that caused project builds to fail with a `CreateProcess` error on Windows hosts has been fixed.

Extraneous map files. A bug in Sourcery G++ IDE that caused a file named `_${BuildArtifactFileName}.map` to be sometimes created in the current directory has been fixed.

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

Preprocessor error handling. The preprocessor now treats failing to find a file referenced via `#include` as a fatal error. Missing include files are now correctly reported in the Sourcery G++ IDE.

NEON improvements. The compiler now generates improved NEON vector code when copying memory or storing constants to memory using the NEON coprocessor. The compiler also generates better code for accessing data arrays that are not known to have 64-bit alignment. In addition, a bug that caused internal compiler errors when compiling for Thumb-2 with NEON enabled has been fixed, as has another bug that caused some vector shift NEON operations to be wrongly rejected.

External Tools for binary file properties. The Sourcery G++ IDE now includes predefined External Tool launches for some common invocations of the GNU Binary Utilities. Refer to Section 4.4.7, “Using External Tools” for more information.

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

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

Compiler errors with `float32_t`. A bug has been fixed that caused compiler errors when using the `float32_t` type from `arm_neon.h`.

Support for ARM Cortex-A5 cores. Sourcery G++ now includes basic support for ARM Cortex-A5 cores. Use the `-mcpu=cortex-a5` command-line option.

Streamlined add-on installation. Installing add-ons is simpler than in previous Sourcery G++ releases. See Section 2.9, “Installing Add-Ons” for the new installation procedure.

Static variables and `asm` statements bug fix. A bug in GCC that caused functions containing static variables and `asm` statements to be miscompiled at `-O2` or above has been fixed. The bug also occurred at other optimization levels when the `-fremove-local-statics` command-line option was used.

Warnings for naked functions. A compiler bug that resulted in incorrect warnings about missing return statements in non-void functions declared with the `naked` attribute has been fixed.

Optimizer bug fix. A bug in GCC that caused functions with complex loop nests to be miscompiled at `-O2` or above has been fixed. The bug also occurred at other optimization levels when the `-fpromote-loop-indices` command-line option was used.

Improved IDE memory display. The Debug perspective in the Sourcery G++ IDE now displays hexadecimal and ASCII data side by side in the Memory view. A new Memory Browser view is also included, which provides a more streamlined user interface than the older Memory view. The Memory Browser is available via Window → Show View.

Improved command-line build of IDE projects. When building an IDE project from the command line, only the active configuration is built now. Previously, all configurations were built.

VFPv4 support. Sourcery G++ now includes support for VFPv4, VFPv4-D16 and NEON-VFPv4 coprocessors. Use the `-mfpu=vfpv4`, `-mfpu=vfpv4-d16` or `-mfpu=neon-vfpv4` options, respectively.

Bundled Eclipse plugins. The Sourcery G++ IDE now includes the Eclipse CVS and Mylyn plugins. For additional information, see Section 4.4.8, “Using Eclipse Plugins in the Sourcery G++ IDE”.

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

Flash programming support on Atmel AT91SAM7Sxxx. The Sourcery G++ Debug Sprite now supports flash programming on Atmel AT91SAM7Sxxx when using SEGGER J-Link devices.

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

A.1.2. Changes in Sourcery G++ 4.4-34

Building IDE projects from the command line. The Sourcery G++ IDE now supports batch-mode project builds from the command line. This feature provides an alternative to converting managed build projects to Makefile projects so that they can be built outside the IDE. For more information, refer to Section 4.4.2, “Building IDE Projects from the Command Line”.

VFP assembly mnemonics. The assembler now accepts unified assembly mnemonics for VFP instructions (e.g. `VADD.f32 s0, s0`) in legacy syntax mode.

ARM Cortex-R4F assembler bug fix. The assembler now correctly recognizes the `-mcpu=cortex-r4f` command-line option to select the Cortex-R4F processor.

VFP half-precision extensions. Sourcery G++ now includes support for VFP coprocessors with half-precision floating-point extensions. This can be enabled with the `-mfpu=vfpv3-d16-fp16` or `-mfpu=vfpv3-fp16` command-line options.

Linux kernel headers update. Linux kernel header files have been updated to version 2.6.30.

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

Improved optimization for Thumb-2. GCC now supports instruction scheduling for Thumb-2 code. This optimization is enabled when compiling with `-O2`, `-O3`, or `-Os`, and can improve performance substantially.

ARM VFP assembler bug fix. The assembler now correctly assembles the `vmls`, `vnmla` and `vnmls` mnemonics. Previously these were incorrectly assembled to different instructions.

Default IDE perspective. The Sourcery G++ IDE now uses the C/C++ perspective by default for all new workspaces.

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

Mixed PIC and non-PIC code. The `elf2flt` utility, automatically run by Sourcery G++ when linking uClinux applications, now gives an error when an attempt is made to link mixed PIC and non-PIC code, which is not supported on uClinux targets. Formerly, it silently produced an invalid executable in such cases.

GDB scripting support. The included version of GDB now supports Python scripting of GDB operations and the `python` GDB command. The Python interpreter is bundled with Sourcery G++. For more information, see the GDB manual.

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

Sourcery G++ IDE update. The Sourcery G++ IDE has been updated to use version 3.5 (Galileo) of the Eclipse Platform and version 6.0 of the Eclipse C/C++ Development Tools (CDT). This update includes several notable changes:

- The content assist feature works much better with C++ templates.
- The C++ indexer supports overloaded operators.
- The Extract Local Variable refactoring is implemented.
- The Rename refactoring can be performed interactively in the editor.

For the detailed list of changes, see the CDT documentation¹.

New assembler directive `.inst`. The assembler now accepts the new `.inst` directive to generate an instruction from its integer encoding.

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

IDE Memory view bug fix. In the Sourcery G++ IDE, the Memory view now prints values for unavailable memory as `?`. Formerly, attempting to view an unavailable memory region resulted in

¹ <http://wiki.eclipse.org/CDT/User/NewIn60>

a `Target request failed` error. This could happen when scrolling the memory view near the boundary of a valid memory region.

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

Debug information for assembler files. The Sourcery G++ IDE now compiles assembler files with debug information when using a debug configuration for project build. Previously, source-level debugging of assembler files was not possible.

Internal error in assembler. An assembler bug that caused an internal error when `.thumb` or `.arm` appears after an invalid instruction has been fixed.

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

Postmortem debugging support in IDE. The Sourcery G++ IDE now offers a `Sourcery G++ Debug (Postmortem)` debug configuration type to allow debugging from a core file.

Project rebuild after property changes. When a project is rebuilt in the Sourcery G++ IDE after modifying any target properties, all files are rebuilt. Previously, object files were not recompiled.

Improved bit counting operation. The `__builtin_ctz` built-in function, which returns the number of trailing zero bits in a value, has been improved to use a shorter instruction sequence for ARMv6T2 and later.

Out-of-range branch errors. A Thumb-2 code generation defect in the compiler that caused `branch out of range` errors from the assembler has been eliminated.

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

Assembler validation improvements. The assembler now issues a warning when a section finishes with an unclosed IT instruction block at the end of the input file. It also now rejects unwinding directives that appear outside of a `.fnstart/.fnend` pair. Additionally, 32-bit Thumb instructions are now correctly rejected when assembling for cores that do not support these instructions.

Assembler validations fix. A bug in the assembler that caused some `addw` and `subw` instructions with `SP` or `PC` as operand to be wrongly rejected has been fixed.

-mauto-it assembler option replaced with -mimplicit-it. The `-mauto-it` command-line option to the assembler has been replaced with a more general `-mimplicit-it` option to control the behavior of the assembler when conditional instructions appear outside an IT instruction block. If you were previously using `-mauto-it`, you should now use `-mimplicit-it=always`. Other `-mimplicit-it` modes allow you to separately control implicit IT instruction insertion behavior in ARM and Thumb-2 code. For more information, refer to the assembler manual. In addition to renaming the option, a number of bugs in the implicit IT generation have been fixed.

Sourcery G++ IDE launcher. The script formerly used to launch the Sourcery G++ IDE has been replaced with a binary program. On Windows hosts, the launcher has also been renamed from `sourcerygxx-ide.bat` to `sourcerygxx-ide.exe`. Refer to Chapter 4, “Using the Sourcery G++ IDE” for information on starting the Sourcery G++ IDE.

Memory monitor bug fix. A bug in the Sourcery G++ IDE has been fixed that sometimes caused Memory Monitors to give an `Unable to read memory` error. The error was caused by the

Monitor attempting to read memory by aligned blocks, which could result in accesses to memory before the requested address.

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

IDE Run and Debug bug fix. A bug in the Sourcery G++ IDE that caused the Run and Debug menu commands to sometimes fail with the error `The selection cannot be launched` has been fixed.

Debug information for variadic functions. A compiler bug that resulted in incorrect debug information for functions with variable arguments has been fixed.

Launch configuration naming for Makefile projects. The Sourcery G++ IDE now names launch configurations for Makefile projects in similar manner as for managed build projects. This fixes a bug that formerly caused launch configurations to be unintentionally overwritten due to inconsistent naming conventions.

Post-clean commands in the IDE. Sourcery G++ IDE now permits you to specify additional commands that are run during clean operation, using the `C/C++ Build → Settings → Build Steps` page of the project properties dialog.

Status Register display in the IDE. A bug in the Sourcery G++ IDE that made it impossible to change the display format of the Status Register in the `Registers` window has been fixed.

Overlay sections. `arm-uclinuxabi-readelf` now correctly recognizes section headers for `ARM_DEBUGOVERLAY` and `ARM_OVERLAYSECTION` sections.

Build errors linking many object files in the IDE. Sourcery G++ now uses response files to overcome the command line length limit of the Windows host platform. Previously, when a project contained many object files, the command line could become too long to pass to the linker, resulting in build errors.

Code generation improvements. The compiler has been changed to make better use of VFP registers in mixed integer and floating-point code, resulting in faster code.

IDE hang during import. A bug that sometimes caused the Sourcery G++ IDE to become unresponsive during project import has been fixed.

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

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

Assembler fix for mixed Thumb and ARM mode. A bug in the assembler has been fixed where mapping symbols were sometimes incorrectly placed at section boundaries. This could lead to incorrect disassembly in some cases.

C++ exception matching. A C++ conformance defect has been fixed. According to clause 15.3 of the standard, given a derived class `D` with base `B`, a thrown `D *` object is not caught by a handler with type `B *&` (that is, a reference to pointer `B`). The compiler formerly treated this case incorrectly as if the handler had type `B *`, which does catch `D *`.

IDE-generated dependency files and pathnames with spaces. A bug in the Sourcery G++ IDE that caused build failures due to incorrect processing of generated dependency files with pathnames containing spaces has been fixed.

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

IDE Welcome page. The Sourcery G++ IDE now starts with a Welcome page, which provides useful information about Sourcery G++ for new users.

Project-dependent templates for new files. The Sourcery G++ IDE now selects the default template for new source and header files based on the project type; C templates are used for C projects and C++ templates are used for C++ projects.

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

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

Hardware breakpoints. A bug has been fixed that caused GDB to report spurious breakpoints when using the multiple sequential connection feature (enabled via the `-m` command-line option).

Linker bug fix. A bug that caused the linker to crash when `.ARM.exidx` sections were discarded by a linker script has been fixed.

Proxy server support in IDE. The Sourcery G++ IDE now supports the use of a proxy server to download license keys from the Support Portal. This is useful when installing Sourcery G++ on a computer protected by a firewall or VPN. The proxy server configuration is also used when downloading add-on packages from the Sourcery G++ Portal. For more information, refer to Section 2.8.3, “Configuring a Proxy Server”.

Configuration file required for Debug Sprite. When invoking the Sourcery G++ Debug Sprite from the command line, it is now required to specify a board configuration file argument. (This is done automatically when the Sprite is run from the Sourcery G++ IDE.) This change eliminates a source of confusion and errors resulting from accidental omission of the configuration file argument, since recent improvements to debugger functionality depend on properties specified in the configuration file. Refer to Chapter 6, “Sourcery G++ Debug Sprite” for more details on invoking the Sourcery G++ Debug Sprite from the command line.

uClibc upgrade. uClibc has been updated to version 0.9.30 plus additional updates from the `uClibc.org` repository as of June 2009. Programs linked with uClibc shared libraries from a previous version of Sourcery G++ must be recompiled to run with the new version of uClibc.

Default debugger in the IDE. When a new debug configuration is created in the Sourcery G++ IDE, a debugger most appropriate to the current project is automatically selected.

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

Watchpoint support. The Sourcery G++ Debug Sprite now implements watchpoints on all currently-supported debugging devices.

Initialization sequence bug fix. A bug has been fixed that caused the Sourcery G++ Debug Sprite to skip the board initialization sequence when using SEGGER J-Link devices.

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

J-Link driver update. The J-Link driver included in Sourcery G++ has been updated to version 4.04a.

License manager upgrade. Sourcery G++ now uses FLEXnet Publisher Licensing Toolkit 11.6.1.

A.1.3. Changes in Sourcery G++ 4.3-160

Incorrect placement of linker-generated functions. A bug that caused some linker-generated functions (including stubs to support interworking from ARM mode to Thumb mode and stubs to avoid processor errata) to be placed in data sections has been fixed.

J-Link improvements. The Sourcery G++ Debug Sprite now supports the GDB **detach** command when using SEGGER J-Link devices.

New option for automatically generating IT blocks. The assembler now allows use of conditional Thumb-2 instructions without requiring explicit IT instructions. Use the `-mauto-it` command-line option to enable this automatic generation of IT instructions.

Assembler bug fix. A bug in the assembler that caused duplicate and missing mapping symbols has been fixed. The bug caused incorrect **objdump** output and incorrect byte-swapping for BE8 configurations.

Hexadecimal register display. The Sourcery G++ IDE now defaults to hexadecimal format when displaying registers in the `Registers` window. This change does not affect floating-point registers, which are displayed in decimal float format.

ULINK2 improvements. The Sourcery G++ Debug Sprite now supports the GDB **detach** command when using ULINK2 devices. Also, a bug that caused the target to be reset when using the Sprite to attach to an already-running program has been fixed.

Stack backtracing and C++ exception handling. Improvements have been made to the linker in support of C++ runtime exception handling and stack backtracing. A problem that caused crashes during the backtrace of C routines that were not compiled with the `-fexceptions` option has been fixed. In addition, the linker generates more compact stack unwinding tables which can lead to smaller executables.

Installing documentation for IDE. The help browser in the Sourcery G++ IDE now uses manuals that are bundled with the documentation component when using the graphical installer, rather than a redundant copy. If you choose a custom Sourcery G++ install without documentation, the manuals will not be available in the IDE. This change reduces the disk space requirements for the installed toolchain, as well as making installation faster.

Build artifact name with spaces. The Sourcery G++ IDE now properly handles space characters in the name of the project build artifact. Previously, this caused the linker to be invoked several times when building the project.

Stellaris flash programming with J-Link. The Sourcery G++ Debug Sprite now supports flash programming for Luminary Micro Stellaris devices with J-Link.

Disassembler bug fix. A bug has been fixed that caused incorrect disassembly of some object files with multiple sections whose symbol tables included symbols in the middle of functions. These typically resulted from hand-written assembly.

IDE project properties bug fix. A bug has been fixed that caused a variety of problems in the Sourcery G++ IDE related to project properties settings. Symptoms included failure to preserve some processor settings from projects created in older versions of Sourcery G++ after an upgrade, and other options that depend on these settings being incorrectly disabled or having incorrect default values.

Linker crash with very large applications. A linker bug that caused a crash when linking very large applications with the `--fix-cortex-a8` command-line option has been fixed.

IDE checkbox for `-fno-common` compiler option. The Sourcery G++ IDE now has a checkbox in the project properties dialog to control the setting of the `-fno-common` compiler option. The checkbox is located in the Miscellaneous group under Sourcery G++ C Compiler or Sourcery G++ C++ Compiler, as appropriate. This option controls how the compiler treats multiple definitions of uninitialized global variables; refer to the GCC manual for details.

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

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

Cortex-A8 erratum workaround enabled for ARMv7-A. The workaround for the erratum in Cortex-A8 processors mentioned below is now enabled by default if you are targeting the ARMv7-A architecture profile. The workaround can be disabled by passing the `--no-fix-cortex-a8` option to the linker.

Display of `\n` in IDE Debug Console. A bug in the Sourcery G++ IDE that caused literal `\n` sequences in debugger output to be incorrectly displayed as newlines in the Debug Console has been fixed. For example, this bug formerly caused some Windows pathnames to print incorrectly.

Preprocessor symbol definitions in IDE. In the Sourcery G++ IDE, when defining preprocessor symbols passed to the C/C++ build via `Project` → `Properties` → `C/C++ General` → `Paths and Symbols` → `Symbols`, symbols with quoted string values formerly caused a build error. This bug has been fixed, and symbols defined using this dialog are now correctly expanded by the preprocessor to exactly the values you provide.

Improved optimization for ARM. GCC now automatically enables loop unrolling and `-fpromote-loop-indices` when `-O2` or `-O3` is specified. Loop unrolling is limited at `-O2` to control code growth. These changes improve performance by more than 5%.

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

Maximum code alignment increased. The maximum allowed code alignment has been increased from 32 to 64 bytes. This change affects the `.p2align` and `.align` directives in GAS and the `-falign-functions` GCC option.

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

A.1.4. Changes in Sourcery G++ 4.3-135

Alternative extension for assembler files. The Sourcery G++ IDE now treats files with the `.sx` extension as assembler files that need preprocessing. This extension can be used as an alternative to the `.S` extension on case-insensitive file systems. This is consistent with the way GCC treats files with this extension.

Improvements and bug fixes in J-Link support. The J-Link support in the Sourcery G++ Debug Sprite has been improved in several ways.

- On ARM9 targets, both of the two breakpoint resources are now available for debugging; previously, one was reserved for semihosting.
- A bug has been fixed that caused flash programming failures when using the multiple sequential connection feature (enabled via the `-m` command-line option).
- Flash programming for STMicroelectronics STR91xFA devices is now supported.
- A bug that caused incorrect software breakpoint behavior when debugging Thumb code has been fixed.

J-Link support. The Sourcery G++ Debug Sprite now supports SEGGER J-Link on Windows hosts.

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

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

Vector register display. A bug in the `Registers` window of the Sourcery G++ IDE that resulted in an error dialog when examining values of some vector registers has been fixed.

Incorrect linker-generated functions. A bug that caused some linker-generated functions (such as stubs to support interworking from ARM mode to Thumb mode) to contain only `nop` instructions instead of correct code sequences has been fixed.

Assembler diagnostics for invalid instructions. The assembler now issues diagnostics for invalid `ADR` and `ADRL` instructions. Formerly, these invalid instructions were silently mis-assembled. This assembler bug did not affect correct code.

Sprite's failure to reset the target. A bug has been fixed that sometimes caused the Sourcery G++ Debug Sprite to fail to reset the target when using the multiple sequential connection feature (enabled via the `-m` command-line option). This problem was specific to running the Debug Sprite on Microsoft Windows hosts.

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

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

Import Executable launch configurations. The `Import Executable` wizard in the Sourcery G++ IDE now permits creating Sourcery G++ launch configurations. Previously, only standard launch configuration types were available.

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

Project properties dialog. A number of usability improvements have been made to the project properties configuration dialogs in the Sourcery G++ IDE, accessible from the `New Project Wizard` and from the `Project` → `Properties` menu item.

- All options on this dialog now have hover help descriptions.
- The entries on the drop-down `Processor` option are now properly sorted, making it easier to find a specific processor.
- The `Processor` option now has an `Other` choice, intended for use with arbitrary command-line options.
- The `FPU` option is disabled unless hardware floating-point has been selected on the `Hard/Soft Float` option.

DMB, DSB, and ISB instructions on ARMv6-M. The assembler now accepts the `DMB`, `DSB`, and `ISB` instructions on ARMv6-M CPUs, including `Cortex-M0` and `Cortex-M1`. These instructions were incorrectly rejected on these CPUs in previous releases.

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

Assembler marking of data. Data generated using the assembler directives `.ascii`, `.asciz`, `.dc.d`, `.dc.s`, `.dc.x`, `.dcb`, `.dcb.b`, `.dcb.d`, `.dcb.l`, `.dcb.s`, `.dcb.w`, `.dcb.x`, `.ds`, `.ds.b`, `.ds.d`, `.ds.l`, `.ds.p`, `.ds.s`, `.ds.w`, `.ds.x`, `.double`, `.fill`, `.float`, `.incbin`, `.single`, `.space`, `.skip`, `.string`, `.string8`, `.string16`, `.string32`, `.string64`, and `.zero` is now correctly marked by the assembler as data rather than code. This fixes incorrect byte-swapping of such data when linking for BE8 configurations.

Improved IDE debugging performance. The Sourcery G++ IDE now takes less time to update the `Modules` and `Debug` windows after step operations. This can considerably improve overall stepping performance for multithreaded programs.

Improved vectorization. Automatic vectorization for NEON now uses the fused multiply-add (VMLA) and fused multiply-subtract (VMLS) instructions. These fused instructions are faster than the equivalent two-instruction sequence consisting of a multiply followed by an add or subtract.

Internal compiler error with large NEON types. A bug has been fixed that caused internal compiler errors when compiling code using NEON types at least 32 bytes wide.

Linker map file. Sourcery G++ now creates a map file by default when linking managed build projects from the IDE. The linker writes the `.map` file to the same directory as the executable. You can explicitly enable or disable the map file generation using the checkbox in the `General` linker option category in the project properties dialog.

Spurious output in IDE application console. A bug in GDB has been fixed that caused the Sourcery G++ IDE to display internal informational messages in the application output console.

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

Out-of-bounds accesses to stack arrays. A bug has been fixed that caused internal compiler errors when some code involving out-of-bounds accesses to stack-allocated arrays was compiled with the `-mthumb` option. Such code is not valid C; although it is now accepted by the compiler and no diagnostic is issued, it has undefined behavior if executed.

Erratum workaround for Cortex-A8 processors. The linker now implements a workaround for an erratum in Cortex-A8 processors. If you are targeting an affected part and wish to use the workaround, pass the `--fix-cortex-a8` option to the linker. Please contact ARM for further details of the erratum.

A.1.5. Changes in Sourcery G++ 4.3-110

Debugger configuration simplified. In the Sourcery G++ IDE, some debugger configuration options that do not apply to ARM uClinux targets have been disabled.

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

Sourcery G++ IDE update. The Sourcery G++ IDE has been updated to use version 3.4.1 (Ganymede) of the Eclipse Platform and version 5.0.1 of the Eclipse C/C++ Development Toolkit (CDT). This update includes several user-visible changes from the version included with previous releases of Sourcery G++. These changes include:

- Additional C++ refactorings, including "Extract Function", "Implement Method" and "Generate Getters and Setters", were added.
- The C++ editor now can fold compound statements and highlight occurrences of the selected element.
- C++ indexer accuracy and performance were improved.
- Code and file templates were implemented.

For the detailed list of changes, see the CDT documentation².

Improved NOP generation for Thumb-2 cores. The assembler now generates Thumb-2/ARMv6K architectural NOP instructions when alignment padding is required in code sections.

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

CS3 board and processor support. CS3 board and processor support has been cleaned up to remove entries that are not appropriate for or supported by Sourcery G++ on ARM uClinux targets. This includes processors for which Sourcery G++ does not include appropriate run-time libraries. In addition, CS3 support for boards based on Cortex-M3 processors has been removed as these boards are not sufficiently powerful to run uClinux. These changes are intended to simplify processor and board selection.

² <http://wiki.eclipse.org/CDT/User/NewIn50>

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

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

Project properties configuration. The Project Properties dialog in the Sourcery G++ IDE has been enhanced to include drop-downs for configuring additional common build options, such as selecting ARM or Thumb code generation and floating-point support. These properties can also be set from the New Project Wizard.

Linker errors on non-ELF input. A bug has been fixed that caused internal errors from the linker when linking non-ELF input files (with the `-b` or `--format` linker options).

Undefined weak references in shared libraries. A linker bug has been fixed affecting calls from Thumb code in shared libraries to functions that are undefined weak references when the shared library is linked. Such calls executed as nops whether or not the functions were defined at run time.

uClibc splice, vmsplice and tee functions. uClibc now provides the functions `splice`, `vmsplice` and `tee`.

Improved code generation. The compiler has been improved to generate better code for an integer multiplication whose result feeds into an addition.

Restart in IDE debugger. In the Debug perspective in the Sourcery G++ IDE, the `Restart` command is now supported. However, note that the External Server or External Embedded Server debugger cannot be restarted from the IDE in this way, since you must explicitly restart the external server.

IDE `cs_target` build variable. The Sourcery G++ IDE now defines the `cs_target` build variable for Makefile projects as well as managed make projects. Refer to Section 4.2.5, “Customizing Build Actions” for more information.

Performance improvements. Tuning parameters for ARM code generation have been adjusted to improve performance of the generated code.

Per-file compilation options in IDE. A bug in the Sourcery G++ IDE that caused an exception when attempting to configure build settings on individual source files has been fixed.

Kernel debugging support in IDE. The Sourcery G++ IDE now supports a Kernel Debug launch configuration which can be used to debug a running uClinux kernel via a supported hardware debug device. The new launch configuration includes IDE support for the Sourcery G++ Debug Sprite. IDE support for Macraigor OCDRemote has also been enabled for kernel debugging. For details, see Section 4.3, “Debugging Applications”.

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

CMP Thumb-2 instruction. The assembler no longer issues an error about `CMP` instructions in which the second argument is the stack pointer (`r13`), as these are valid instructions. However, use

of the stack pointer in this context is deprecated in the current ARM architecture specification and the assembler now warns about the deprecated use.

Thumb half-precision floating point bug fix. A compiler bug has been fixed that formerly caused incorrect code to be generated in Thumb mode for functions using half-precision floating-point constants. The bug did not affect Thumb-2 code.

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

Improved code generation. The compiler has been improved to generate better code for integer multiplication by certain constants.

Debugging imported executables. A bug that caused errors in the Sourcery G++ IDE when configuring the debugger for imported executable projects has been fixed.

Thumb-2 switch code generation bug fix. A bug has been fixed that caused incorrect Thumb-2 code to be generated for some `switch` statements.

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

Debug launch configurations in IDE. Debug launch configuration types specific to Sourcery G++ have been added to the `Debug Configurations` dialog. Existing Local Application debug launches remain available and continue to work as before. However, the new launch configuration types are recommended as they support additional functionality specific to Sourcery G++.

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

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

Vectorized shift fix. A bug has been fixed that caused incorrect code for loops containing a right shift by a constant. The bug affected code compiled with `-mcpu=neon` and loop vectorization enabled with `-O3` or `-ftree-vectorize`.

Managed make project build errors. Managed make projects in the Sourcery G++ IDE containing source files with spaces in their names formerly failed to build on Windows hosts, due to an error in formatting the linker command line. This bug has been fixed.

Incorrect code for nested functions. A bug in GCC that caused the compiler to generate incorrect code for nested functions has been fixed. The bug resulted in incorrect stack alignments in the affected functions.

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

ARM build attributes conformance improvements. Several ARM EABI 2.07 conformance issues relating to the handling of build attributes in the assembler and linker have been fixed. All build attribute types are now recognized, and can now be declared by name, in addition to by number.

Support for merging attributes in the linker has been improved, and the linking of incompatible objects is now detected and rejected in more cases.

Debugging program startup in the IDE. The Sourcery G++ IDE now has a `Stop on first assembler instruction` option to allow debugging of program startup code executed before `main`.

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

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

Linker crash on incompatible input files. Some third-party compilers, including ARM RealView® 4.0, produce a build attribute marking output files that are not compatible with the ABI for the ARM Architecture. This attribute sometimes caused the linker to crash. The linker now correctly issues an error message.

A.1.6. Changes in Sourcery G++ 4.3-64

Optimizer bug fix. A bug that caused an unrecognizable `insn` internal compiler error when compiling at optimization levels above `-O0` has been fixed.

Makefile project defaults. Bugs in the IDE have been fixed that formerly caused Makefile projects to incorrectly default to using the native system tools rather than those provided by Sourcery G++. This affected the compiler used for scanner discovery, the default debugger used for launch configurations, and (on Windows hosts) the `make` utility used for building the project.

A.1.7. Changes in Sourcery G++ 4.3-57

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

VFP compiler fix. A compiler bug that resulted in `internal compiler error: output_operand: invalid expression as operand` when generating VFP code has been fixed.

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

Workaround for Cortex-M3 CPU errata. Errata present in some Cortex-M3 cores can cause data corruption when overlapping registers are used in `LDRD` instructions. The compiler avoids generating these problematic instructions when the `-mfix-cortex-m3-ldrd` or `-mcpu=cortex-m3` command-line options are used. The Sourcery G++ runtime libraries have also been updated to include this workaround.

Misaligned NEON memory accesses. A bug has been fixed that caused the compiler to use aligned NEON load/store instructions to access misaligned data when autovectorizing certain loops. The bug affected code compiled with `-mfpu=neon` and loop vectorization enabled with `-O3` or `-ftree-vectorize`.

IDE debugger timeout error. A bug in GDB has been fixed that sometimes caused the IDE to report a `Target is not responding (timed out)` error while the program being debugged was running.

Sprite crash on error. A bug has been fixed which sometimes caused the Sourcery G++ Debug Sprite to crash when it attempted to send an error message to GDB.

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

A.1.8. Changes in Sourcery G++ 4.3-34

Thumb-2 assembler fixes. The Thumb-2 encodings of `QADD`, `QDADD`, `QSUB`, and `QDSUB` have been corrected. Previous versions of the assembler generated incorrect object files for these instructions. The assembler now accepts the `ORN`, `QASX`, `QSAX`, `RRX`, `SHASX`, `SHSAX`, `SSAX`, `USAX`, `UHASX`, `UQSAX`, and `USAX` mnemonics. The assembler now detects and issues errors for invalid uses of register 13 (the stack pointer) and register 15 (the program counter) in many instructions.

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

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

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

Debugging improvements with ULINK2. The ULINK2 drivers and firmware have been updated. The Sourcery G++ Debug Sprite now supports using ULINK2 to debug Cortex-M3 devices from STMicroelectronics and Luminary Micro. More flash programming algorithms are included. Several bugs have been fixed, including timeouts during flash programming and errors stepping after reset.

Connecting to the target using a pipe. A bug in GDB's `target remote | program` command has been fixed. When launching the specified `program` failed, the bug caused GDB to crash, hang, or give a message `Error: No Error`. The bug could also be triggered by failures in launching the Sourcery G++ Debug Sprite from the IDE.

Mixed-case NEON register aliases. An assembler bug that prevented NEON register aliases from being created with mixed-case names using the `.dn` and `.qn` directives has been fixed. Previously only aliases created with all-lowercase or all-uppercase names worked correctly.

Memory map support in IDE debugger. The Sourcery G++ IDE now allows you to specify a read-only memory region (such as flash memory) in all debugging modes. See Section 4.3.4.2, “Configuring the Memory Map”, for more information.

Makefile project creation in IDE. A bug has been fixed that caused the New Project Wizard to fail when creating a Makefile project. This bug was present only in some early 4.3 toolchain versions.

IDE project import. A bug that prevented projects created by the generic Eclipse IDE and some early versions of the Sourcery G++ IDE from being opened by the current Sourcery G++ IDE has been fixed. Such projects are now opened, although some of the added Sourcery G++ functionality may not be available.

ARM exception handling bug fix. A bug in the runtime library has been fixed that formerly caused throwing an unexpected exception in C++ to crash instead of calling the unexpected exception handler. The bug only affected C++ code compiled by non-GNU compilers such as ARM RealView®.

LinuxThreads support. The included uClibc now supports the LinuxThreads implementation of POSIX threads for the ARMv4T multilib. Please note that this feature is not yet supported by the ARMv6-M Thumb and ARMv7 Thumb-2 multilibs.

New Project Wizard improvements. In the Sourcery G++ IDE, you can now specify global properties of the project, such as the target processor, directly from the New Project Wizard, as well as from the Project Settings dialog. See Section 4.2.1, “Setting Up an Example Project” for more information.

Customizing debugger startup. You can now specify additional GDB commands to be executed immediately before and after connecting to the target from within the Sourcery G++ IDE. You can use these commands to perform custom configuration actions for the target connection. For more information, see Section 4.3.4.1, “Debugger Startup”.

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

Half-precision floating point. Sourcery G++ now includes support for half-precision floating point via the `__fp16` type in C and C++. The compiler can generate code using either hardware support or library routines. For more information, see Section 3.3.3, “Half-Precision Floating Point”.

A.1.9. Changes in Sourcery G++ 4.3-31

Watch expression IDE bug. A bug in the IDE debugger interface has been fixed that formerly caused an error message when selecting an expression in the Expressions window. Now, the detail pane successfully displays the details for the selected expression.

A.1.10. Changes in Sourcery G++ 4.3-16

Improved support for debugging RealView® objects . GDB support for programs compiled by the ARM RealView® compiler has been improved.

Definition of `va_list`. In order to conform to the ABI for the ARM Architecture, the definition of the type of `va_list` (defined in `stdarg.h`) has been changed. This change impacts only the mangled names of C++ entities. For example, the mangled name of a C++ function taking an argument of type `va_list`, or `va_list *`, or another type involving `va_list` has changed. Since this is an incompatible change, you must recompile and relink any modules defining or using affected `va_list`-typed entities.

License manager upgrade. Sourcery G++ now uses FLEXnet Publisher Licensing Toolkit 11.6. This upgrade corrects a problem that affected GNU/Linux users with Ethernet interfaces other than `eth0`. The license manager is now aware of all Ethernet interfaces in the range `eth0` through `eth9`.

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

Uppercase operands to IT instructions. The assembler now accepts both uppercase and lowercase operands for the IT family of instructions.

NEON improvements. Several improvements and bug fixes have been made to the NEON Advanced SIMD Extension support in GCC. A problem that caused the autovectorizer to fail in some circumstances has been fixed. Also, many of the intrinsics available via the `arm_neon.h` header

file now have improved error checking for out-of-bounds arguments, and the `vget_lane` intrinsics that return signed values now produce improved code.

NEON compiler fix. A compiler bug that resulted in incorrect NEON code being generated has been fixed. Typically the incorrect code occurred when NEON intrinsics were used inside small `if` statements.

Remote debugging improvements. The `gdbserver` utility now supports a more efficient communications protocol that can reduce latency during remote debugging. The protocol optimizations are enabled automatically when `gdbserver` operates over a TCP connection. Refer to the GDB manual for more information.

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

IDE project builds on Windows. On Windows hosts, the Sourcery G++ IDE's automatic build process no longer assumes that any program named `sh.exe` found in the `PATH` is a compatible shell program. Instead, the DOS shell is now consistently used to execute build commands.

Thumb-2 MUL encoding. In Thumb-2 mode, the assembler now encodes `MUL` as a 16-bit instruction (rather than as a 32-bit instruction) when possible. This fix results in smaller code, with no loss of performance.

ARM C++ ABI utility functions. Vector utility functions required by the ARM C++ ABI no longer crash when passed null pointers. The affected functions are `__aeabi_vec_dtor_cookie`, `__aeabi_vec_delete`, `__aeabi_vec_delete3`, and `__aeabi_vec_delete3_nodtor`. These functions are not intended for use by application programmers; they are only called by compiler-generated code. They are not presently used by the GNU C++ compiler, but are used by some other compilers, including ARM's RealView® compiler.

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

Smaller Thumb-2 code. When optimizing for size (i.e., when `-Os` is in use), GCC now generates the 16-bit `MULS` Thumb-2 multiply instruction instead of the 32-bit `MUL` instruction.

Improved flash programming speed. Flash programming is now faster when using the Sourcery G++ Debug Sprite with Altera and ARMUSB devices.

Improvements to elf2flt utility. The `elf2flt` utility, automatically run by Sourcery G++ when linking uClinux applications, is now compatible with the linker option `--gc-sections`. Previously, applications linked with `--gc-sections` terminated at startup with an illegal instruction error.

Memory monitor improvements. The IDE memory monitor functionality has been made more robust in the case of memory read errors. If a memory address in the requested range cannot be read, all the memory until that address is displayed, and the following memory is marked as unavailable. Previously, the monitor failed to display any memory contents if it encountered a read error.

Janus 2CC support. GCC now includes a work-around for a hardware bug in Avalent Janus 2CC cores. To compile and link for these cores, use the `-mfix-janus-2cc` compiler option. If you are using the linker directly use the `--fix-janus-2cc` linker option.

IDE integration with manuals. The manuals shipped with Sourcery G++ can now be directly accessed from the IDE, via the `Help → Help Contents` menu item.

Mangling of NEON type names. A bug in the algorithm used by the C++ compiler for mangling the names of NEON types, such as `int8x16_t`, has been fixed. These mangled names are used internally in object files to encode type information in addition to the programmer-visible names of the C++ variables and functions. The new mangled name encoding is more compact and conforms to the ARM C++ ABI.

License file format changes. The license file format for Sourcery G++ has been simplified so that you no longer need a separate `cs.lic` file. If you installed Sourcery G++ before February, 2008, you must re-download your license key when you upgrade to a newer version. The Sourcery G++ IDE Licensing wizard detects this situation automatically, or if you prefer you can manually download an updated license key from the Sourcery G++ Portal³. For more information, see Section 2.8, “License Keys”.

Sprite communication improvements. The Sourcery G++ Debug Sprite now uses a more efficient protocol for communicating with GDB. This can result in less latency when debugging, especially when running the Sprite on a remote machine over a network connection.

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

IDE defines `cs_target` build macro. The Sourcery G++ IDE now defines the build macro `cs_target` to the target prefix (`arm-uclinuxeabi` for ARM uClinux). You can use this macro in custom build steps. Refer to Section 4.2.5, “Customizing Build Actions” for more information.

A.1.11. Changes in Older Releases

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

³ <https://support.codesourcery.com/GNUToolchain/>

Appendix B

Sourcery G++ Licenses

Sourcery G++ 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++. You should read this appendix to understand your legal rights and obligations as a user of Sourcery G++.

B.1. Licenses for Sourcery G++ Components

The table below lists the major components of Sourcery G++ for ARM uClinux 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++. Sourcery G++ 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++.

Component	License
GNU Compiler Collection	GNU General Public License 3.0 ¹
GNU Binary Utilities	GNU General Public License 3.0 ²
Eclipse IDE	Eclipse Public License 1.0 ³
Eclipse C/C++ Development Tools	Eclipse Public License 1.0 ⁴
Sourcery G++ Eclipse Plugin(s)	CodeSourcery License
Sourcery G++ IDE Launcher	CodeSourcery License
GNU Debugger	GNU General Public License 3.0 ⁵
Sourcery G++ Cygwin GDB Wrapper	GNU General Public License 2.0 ⁶
Sourcery G++ Debug Sprite for ARM	CodeSourcery License
uClibc C Library	GNU Lesser General Public License 2.1 ⁷
Linux Kernel Headers	GNU General Public License 2.0 ⁸
ELF-to-FLT Conversion Utility	GNU General Public License 2.0 ⁹
GNU Make	GNU General Public License 2.0 ¹⁰
GNU Core Utilities	GNU General Public License 2.0 ¹¹

The CodeSourcery License refers to the license agreement under which you licensed Sourcery G++ from CodeSourcery, Inc. or its authorized distributor or reseller, including without limitation the Sourcery G++™ Software License Agreement (see Section B.2, “Sourcery G++ Software License Agreement” below).

Important

Although some of the licenses that apply to Sourcery G++ 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++. You can develop proprietary applications and libraries with Sourcery G++.

¹ <http://www.gnu.org/licenses/gpl.html>

² <http://www.gnu.org/licenses/gpl.html>

³ <http://www.eclipse.org/org/documents/epl-v10.php>

⁴ <http://www.eclipse.org/org/documents/epl-v10.php>

⁵ <http://www.gnu.org/licenses/gpl.html>

⁶ <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

⁷ <http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html>

⁸ <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

⁹ <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

¹⁰ <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

¹¹ <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

Sourcery G++ may include some third party example programs and libraries in the `share/sourceryg++-arm-uclinuxeabi-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++™ Professional Edition, Sourcery G++™ Personal Edition, or Sourcery G++™ for Evaluation (the “Software”) provided to You, including any Updates thereto.
3. **Definitions.**
 - 3.1. **Effective Date.** The date on which CodeSourcery gives You access to CodeSourcery's electronic support system.
 - 3.2. **CodeSourcery Proprietary Components.** The components of the Software that are owned and/or licensed by CodeSourcery and are not subject to a “free software” or “open source” license, such as the GNU Public License. The CodeSourcery Proprietary Components of the Software include, without limitation, the Sourcery G++ Installer, any Sourcery G++ Eclipse plug-ins, and any Sourcery G++ Debug Sprite. For a complete list, refer to the *Getting Started Guide* included with the distribution.
 - 3.3. **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.
 - 3.4. **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.
 - 3.5. **Redistributable Components.** The CodeSourcery Proprietary Components that are intended to be incorporated or linked into Licensee object code developed with the Software. The Redistributable Components of the Software include, without limitation, the CSLIBC run-time library and the CodeSourcery Common Startup Code Sequence (CS3). For a complete list, refer to the *Getting Started Guide* included with the distribution.
4. **License Grant to Proprietary Components of the Software.**
 - 4.1. **Sourcery G++ Professional and Personal Edition.** Subject to the terms and conditions hereof, CodeSourcery hereby grants to Licensee of Sourcery G++ Professional or Personal Edition a perpetual, non-exclusive license under the Proprietary Rights of CodeSourcery and its licensors (a) to install and use the CodeSourcery Proprietary Components of the Software by a single user who uses the Software on one machine, and (b) to distribute the Redistributable Component(s) of the Software 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).

- 4.2. **Sourcery G++ for Evaluation.** Subject to the terms and conditions hereof, CodeSourcery hereby grants to Licensee of Sourcery G++ for Evaluation a limited, non-exclusive license under the Proprietary Rights of CodeSourcery and its licensors to install and use the CodeSourcery Proprietary Components of the Software, for evaluation purposes only, by a single user who uses the Software on one machine during the term of this Agreement.
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.
- 5.1. **ARM Keil ULINK2 Drivers.** Sourcery G++ may include ULINK2 drivers from ARM, Ltd. If these drivers are included, the following additional terms and conditions apply:
- a. You may use the ULINK2 drivers only in conjunction with a compatible ARM Keil ULINK2 hardware unit manufactured by or under license from ARM and purchased from CodeSourcery, ARM, or a distributor authorized by ARM.
 - b. You may use the ULINK2 drivers only to connect to the GNU Debugger included in Sourcery G++.
 - c. The ULINK2 drivers are not supported by ARM, Ltd.; You should contact CodeSourcery for any support regarding the ULINK2 drivers.
 - d. You may not redistribute or transfer the ULINK2 drivers.
 - e. You may not translate, adapt, arrange or otherwise alter the object code of the ULINK2 drivers (including without limitation copying, adapting or reverse compiling the object code of the ULINK2 drivers for the purpose of error correction) except as allowed by applicable law.
 - f. You may not remove or obstruct any notice or marker incorporated into the ULINK2 drivers to protect ARM's or third parties' intellectual property or Proprietary Rights.
 - g. The ULINK2 drivers are licensed, not sold; all right, title and interest therein is reserved to CodeSourcery or its licensors, and You acquire no right, title or interest therein.
- 5.2. **SEGGER J-Link™ Devices.** Sourcery G++ includes proprietary software from SEGGER Microcontroller GmbH & Co.KG that allows the use of SEGGER J-Link debug devices with the Sourcery G++ Debug Sprite. You may use software from SEGGER only under the SEGGER J-Link software terms of use¹² and license agreement¹³.
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*

¹² http://segger.com/jlink_arm_software_terms_of_use.html

¹³ http://segger.com/pub/jlink/license_agreement.txt

provides an overview of which license applies to different components. Definitive licensing information for each “free software” or “open source” component is available in the relevant source file.

7. **CodeSourcery Trademarks.** Notwithstanding any provision in a “free software” or “open source” license agreement applicable to a component of the Software that permits You to distribute such component to a third party in source or binary form, You may not use any CodeSourcery trademark, whether registered or unregistered, including without limitation, CodeSourcery™, Sourcery G++™, the CodeSourcery crystal ball logo, or the Sourcery G++ splash screen, or any confusingly similar mark, in connection with such distribution, and You may not recompile the Open Source Software Components with the `--with-pkgversion` or `--with-bugurl` configuration options that embed CodeSourcery trademarks in the resulting binary.
8. **Term.**
 - 8.1. **Sourcery G++ Professional and Personal Edition Subscription.** For Licensee of Sourcery G++ Professional or Personal Edition, this Agreement shall have a subscription term of one (1) year unless this Agreement is terminated pursuant to Section 12.
 - 8.2. **Sourcery G++ for Evaluation.** For Licensee of Sourcery G++ for Evaluation, this Agreement shall have an evaluation term of thirty (30) days unless this Agreement is terminated pursuant to Section 12.
9. **Updates.** During the term of this Agreement, You may download, free of charge, any new version(s), update(s), or upgrade(s) ("Updates") to the Software that CodeSourcery makes available at such times as may be determined by CodeSourcery in its sole discretion.
10. **Technical Support.** CodeSourcery shall provide technical support only to Licensee of Sourcery G++ Professional Edition.
 - 10.1. **Support Term.** CodeSourcery shall provide technical support to Licensee pursuant to this Section 10 during the subscription term of this Agreement.
 - 10.2. **Scope of Support.** CodeSourcery shall assist Licensee in installing and using the Software in binary form. CodeSourcery shall correct defects in the Software reported by Licensee, subject to the limitations set forth below. Licensee may submit support requests only on its own behalf and not on behalf of any other party, including a licensee of any other edition of Sourcery G++, including without limitation Sourcery G++ Personal, Standard or Lite Edition. If Licensee is a member of a development team, all members of the team must be Licensees of Sourcery G++ Professional Edition. CodeSourcery shall impose no limit on the number of support requests made by Licensee. Although CodeSourcery will make source code for the Open Source Components available to Licensee, CodeSourcery's support does not cover rebuilding the toolchains from the source packages, correcting defects in any such rebuilt toolchains, or answering any questions arising from Licensee's use of source packages.
 - 10.3. **Electronic Support System.** Licensee shall make all support requests via CodeSourcery's electronic support system, and CodeSourcery shall respond via the same electronic support system. CodeSourcery will not accept support requests by telephone or other means.
 - 10.4. **Response Time.** CodeSourcery's electronic support system will provide Licensee with an immediate acknowledgment of the support request (including a unique tracking number) by electronic mail. CodeSourcery shall respond to all support requests within

one business day, except in extraordinary circumstances. CodeSourcery shall attempt to resolve all support requests within three business days.

- 10.5. **No Guarantee of Resolution.** CodeSourcery does not guarantee that it will be able to resolve all support requests. Without limitation, CodeSourcery may, in its sole discretion, determine that a defect in the Software is too difficult to correct, or that any correction would likely risk the introduction of additional defects, or that the defect is not likely to be encountered often enough to be worthy of correction, or that the defect is insufficiently severe to be worthy of correction.
- 10.6. **Support for Previous Versions.** After the release of an Update, CodeSourcery shall provide support for previous version(s) of the Software for a period of six (6) months. CodeSourcery will have no obligation to provide support after this period.
- 10.7. **Test Cases.** In many cases, CodeSourcery will require access to Licensee's source code in order to resolve Licensee's support request. CodeSourcery may, in its sole discretion, create regression tests distilled from Licensee's source code for use in testing changes to the Software. CodeSourcery shall use commercially reasonable efforts to disguise the origin of the source code, to eliminate non-essential aspects of the source code, and to otherwise protect the confidentiality of Licensee's source code. These regression tests may be made available to other CodeSourcery licensees or to the general public.

11. Fees.

- 11.1. **Annual Subscription Fee.** The licenses and rights granted in this Agreement are subject to Licensee's payment of all fees owed to CodeSourcery.
- 11.2. **Taxes.** All fees are exclusive of sales or use taxes and any levy imposed on the transportation or use of the Software. Licensee shall pay all such charges either as levied by taxing authorities or as invoiced by CodeSourcery.
- 11.3. **Non-refundability.** All payments made to CodeSourcery are non-refundable.

12. Termination.

12.1. Grounds for Termination.

- 12.1.1. **Termination for Material Breach.** 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.
- 12.1.2. **Termination of Updates and Technical Support for Unauthorized Redistribution of Binaries .** CodeSourcery shall provide Updates and Technical Support to Licensee, pursuant to Sections 9 and 10, respectively, only on the condition that Licensee uses the Software for internal use and/or distributes the Software in binary form pursuant to a separate redistribution agreement with CodeSourcery. Any other distribution by Licensee of the Software in binary form, including distribution permitted by the applicable "free software" or "open source" license, shall automatically terminate this Agreement and shall void the remainder of Licensee's subscription term.

12.2. Effect of Termination.

- 12.2.1. **Surviving Obligations.** The following obligations shall survive the termination or expiration of this Agreement: (i) any and all warranty disclaimers or limitations of liability herein, and (ii) any covenant granted herein for the purpose of determining ownership of, or protecting, the Proprietary Rights of either party, including the CodeSourcery trademarks as set forth in Section 7, or any remedy for breach thereof.
- 12.2.2. **Surviving Rights.** After the expiration of this Agreement, Licensee of Sourcery G++ Professional Edition or Sourcery G++ Personal Edition may continue to use the Software, including the CodeSourcery Proprietary Components, pursuant to Section 4 and 5, but CodeSourcery shall provide no further Updates or Technical Support to Licensee. However, if this agreement is terminated pursuant to Section 12.1, Licensee's rights pursuant to Section 4 shall terminate immediately and Licensee shall destroy all copies of the CodeSourcery Proprietary Components of the Software.
13. **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.
14. **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.
15. **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.
16. **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.
17. **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, INCIDENTAL, 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.

18. **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.
19. **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.
20. **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 s'y 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.
21. **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.
22. **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.

23. **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.
24. **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.
25. **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.
26. **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.