Sourcery G++ Lite

ARM uClinux

2007q3-51

Getting Started

Sourcery G++ Lite: ARM uClinux: 2007q3-51: Getting Started

CodeSourcery, Inc.

Preface

This preface introduces *Getting Started With Sourcery G++ Lite*. It explains the structure of this guide and lists other sources of information that relate to Sourcery G++ Lite.

1 Intended Audience

This guide is written for people who will install and/or use Sourcery G++ Lite. This guide provides a step-by-step guide to installing Sourcery G++ Lite and to building simple applications. Parts of this document assume that you have some familiarity with using the command-line interface. If you are an administrator installing Sourcery G++ Lite on a UNIX-like system for all of your users to use, you should also be familiar with the package-management software (such as the Red Hat Package Manager) for your system.

2 Organization

This document is organized into the following chapters and appendices:

Chapter 1, *Sourcery G++ Lite Licenses*	This chapter provides information about the software licenses that apply to Sourcery G++ Lite. Read this chapter to understand your legal rights and obligations as a user of Sourcery G++ Lite.
Chapter 2, *Sourcery G++ Subscriptions*	This chapter provides information about Sourcery G++ Lite subscriptions. CodeSourcery customers with Sourcery G++ Lite subscriptions receive comprehensive support for Sourcery G++ Lite. Read this chapter to find out how to obtain and use a Sourcery G++ Lite subscription.
Chapter 3, *Sourcery G++ Lite for ARM uClinux*	This chapter provides information about this release of Sourcery G++ Lite including any special installation instructions, recent improvements, or other similar information. You should read this chapter before building applications with Sourcery G++ Lite.
Chapter 4, *Installation and Configuration*	This chapter describes how to download, install and configure Sourcery G++ Lite. This section describes the available installation options and explains how to set up your environment so that you can build applications.
Chapter 5, *Using the Sourcery G++ IDE*	This chapter explains how to use the Sourcery G++ IDE, which is based on Eclipse.
Chapter 6, *Using Sourcery G++ from the Command Line*	This chapter explains how to build applications with Sourcery G++ Lite using the command line. In the process of reading this chapter, you will build a simple application that you can use as a model for your own programs.

3 Typographical Conventions

The following typographical conventions are used in this guide:

`> command arg ...`	A command, typed by the user, and its output. The "`>`" character is the command prompt.

command	The name of a program, when used in a sentence, rather than in literal input or output.
`literal`	Text provided to or received from a computer program.
`placeholder`	Text that should be replaced with an appropriate value when typing a command.

Chapter 1
Sourcery G++ Lite Licenses

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

1.1 Licenses for Sourcery G++ Lite Components

The table below lists the major components of Sourcery G++ Lite 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++ Lite. Sourcery G++ Lite may contain free or open-source components not included in the list below; for a definitive list, consult the source package corresponding to this release of Sourcery G++ Lite.

Component	**License**
GNU Binary Utilities	GNU General Public License 3.0 [1]
GNU Compiler Collection	GNU General Public License 3.0 [2]
GNU Debugger	GNU General Public License 3.0 [3]
uClibc C Library	GNU Lesser General Public License 2.1 [4]
Linux Kernel	GNU General Public License 2.0 [5]
Sourcery G++ Lite Debug Sprite for ARM	CodeSourcery License
GNU Make	GNU General Public License 2.0 [6]
GNU Core Utilities	GNU General Public License 2.0 [7]

The CodeSourcery License is available in Section 1.2, “Sourcery G++™ Software License Agreement”.

Important

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

1.2 Sourcery G++™ Software License Agreement

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

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

[1] http://www.gnu.org/licenses/gpl.html
[2] http://www.gnu.org/licenses/gpl.html
[3] http://www.gnu.org/licenses/gpl.html
[4] http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html
[5] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
[6] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
[7] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

3. **Definitions.**

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

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

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

4. **License Grant to Proprietary Components of the Software.** You are granted a non-exclusive, royalty-free license to install and use the CodeSourcery Proprietary Components of the Software, transmit the CodeSourcery Proprietary Components over an internal computer network, and/or copy the CodeSourcery Proprietary Components for Your internal use only.

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; or (iii) reverse engineer, decompile, or disassemble the CodeSourcery Proprietary Components of the Software, except to the extent this restriction is expressly prohibited by applicable law.

6. **"Free Software" or "Open Source" License to Certain Components of the Software.** This Agreement does not limit Your rights under, or grant You rights that supersede, the license terms of any Open Source Software Component delivered to You by CodeSourcery. For a list of which license applies to each component, refer to the "Getting Started Guide" included with this distribution.

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

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

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

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

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

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

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

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

By downloading or using the Software, Licensee agrees to the foregoing and represents and warrants that it complies with these conditions.

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

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

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

18. **Arbitration.** Except for actions to protect intellectual property rights and to enforce an arbitrator's decision hereunder, all disputes, controversies, or claims arising out of or relating to this Agreement or a breach thereof shall be submitted to and finally resolved by arbitration under the rules of the American Arbitration Association ("AAA") then in effect. There shall be one arbitrator, and such arbitrator shall be chosen by mutual agreement of the parties in accordance with AAA rules. The arbitration shall take place in Granite Bay, California, and may be conducted by telephone or online. The arbitrator shall apply the laws of the State of California, USA to all issues in dispute. The controversy or claim shall be arbitrated on an individual basis, and shall not be consolidated in any arbitration with any claim or controversy of any other party. The findings of the arbitrator shall be final and binding on the parties, and may be entered in any court of competent jurisdiction for enforcement. Enforcements of any award or judgment shall be governed by the United Nations Convention on the Recognition and Enforcement of Foreign Arbitral Awards. Should either party file an action contrary to this provision, the other party may recover attorney's fees and costs up to $1000.00.

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

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

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

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

Chapter 2
Sourcery G++ Subscriptions

CodeSourcery provides support contracts for Sourcery G++. This chapter describes these contracts and explains how CodeSourcery customers can access their support accounts.

2.1 About Sourcery G++ Subscriptions

CodeSourcery offers Sourcery G++ subscriptions. Professional Edition subscriptions provide unlimited support, with no per-incident fees. CodeSourcery's support 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++ to resolve critical problems. Personal Edition subscriptions do not include support, but do include free upgrades as long as the subscription remains active.

CodeSourcery's support is provided by the same engineers who build Sourcery G++. A Sourcery G++ subscription is like having a team of compiler engineers and programming language experts available as consultants!

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

2.2 Accessing your Sourcery G++ Subscription Account

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

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

Chapter 3
Sourcery G++ Lite for ARM uClinux

This chapter contains information about using Sourcery G++ Lite on your target system. This chapter also contains information about changes in this release of Sourcery G++ Lite. You should read this chapter to learn how to best use Sourcery G++ Lite on your target system.

3.1 Debugging ARM uClinux

Sourcery G++ Lite provides two ways to debug a target board. A gdbserver can be used to debug user applications and a Sourcery G++ Lite Debug Sprite can be used to debug the kernel itself.

3.1.1 GDB Server

Sourcery G++ Lite contains a **gdbserver** for running on the target. The servers are located in the **arm-uclinuxeabi/libc/*cpu*/usr/bin** directories of your installation. Although there are different servers for each CPU variant, the default in **arm-uclinuxeabi/libc/usr/bin** works across all ARM architectures. You need to copy this to your target system and then invoke it as

```
# 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.

3.1.2 ARM Halting Debug Mode

Sourcery G++ Lite contains the Sourcery G++ Debug Sprite. This sprite uses Halting Debug Mode, and is provided for remote debugging of a bare board. You can use this 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**).

This section demonstrates execution and debugging of a simple application. Create a file named `hello.c`:

```
#include <stdio.h>

int
main (void)
{
  printf("Hello World!\n");
  return 0;
}
```

Compile and link the program for the target board. If it is a stand-alone program for a Stellaris LM3S800-series board use:

```
> arm-uclinuxeabi-gcc -mcpu=cortex-m3 -mthumb -Tlm3s8xx-rom.ld \
  hello.c -o hello -g
```

For other boards you must make appropriate substitutions in the preceding command. If your program is an operating system kernel such as uClinux or Linux, your usual build method should be adequate, as the kernel contains the necessary initialization code for interrupt handlers.

Verify that the Sourcery G++ Debug Sprite supports your debug hardware:

```
arm-uclinuxeabi-sprite -i
```

This prints out a list of supported device types. All ICE units supported by the Sourcery G++ Debug Sprite auto-detect the device connected to them, so nothing needs to be done to identify it explicitly. The command should output:

```
CodeSourcery ARM Debug Sprite
rdi: (rdi-library=<file>&rdi-config=<file>) RDI Device
  rdi:/// - RDI Device
armusb: [speed=<n:0-7>] ARMUSB device
  armusb:/// - ARMUSB Device
```

This shows that RDI and ARMUSB devices are supported.

Start the debugger on your host system:

```
> arm-uclinuxeabi-gdb hello
```

Connecting GDB to the board depends on the debug device you are using. If you are using an ARMUSB debug device, use:

```
(gdb) target remote | arm-uclinuxeabi-sprite \
armusb:///?speed=2 lm3s8xx
Remote debugging using | arm-uclinuxeabi-sprite \
armusb:///?speed=2 lm3s8xx
arm-uclinuxeabi-sprite:Target reset
start () at /buildpath/newlib-arm/libgloss/arm/crt0.S:50
50              ldr     r0, =__data_load
Current language:  auto; currently asm
```

If you are connecting via RDI, you must specify the full path to the RDI library file and configuration file for that library:

```
(gdb) target remote | \
arm-uclinuxeabi-sprite \
"rdi:///?rdi-library=library&rdi-config=config"
Remote debugging using | arm-uclinuxeabi-sprite \
"rdi:///?rdi-library=library&rdi-config=config"
ARMulator RVARMulatorISS1.4 [Build 297]
For support please contact support-sw@arm.com
Software supplied by: ARM Limited
ARM1136JF-S
ARM11 Instruction Set Simulator, May 24 2006
ARM Instruction Set Simulator for  [Build number 297]
, CP15, 8KB ICache, 8KB DCache 32KB DTCRam0 -Supports SmartCaching
32KB ITCRam0 -Supports SmartCaching , VFP11 (no support code), \
4GB, Pagetables, Mapfile, VIC - PL192
VIC: this is a RELEASE build
, Profiler, SIMRDI MemCallback, Tube, Millisecond [6666.67
cycles_per_millisecond], Tracer
Tracing: Instructions, Memory accesses, Events, Disassemble, \
Trace bus, Trace registers, Opcode Fetch Mask \
0x00000000-0x00000000, RDI Codesequences, Semihosting, \
```

```
CP14 Debug(6,2,2)
Little endian
arm-uclinuxeabi-sprite:Missing config file; this may not work
arm-uclinuxeabi-sprite:Target reset
0x00000000 in ?? ()
```

Refer to Section 3.1.2.1, “Invoking Sourcery G++ Debug Sprite” for a full description of the `armusb:`, `rdi:` and `lm3s8xx` arguments, or if you are using a different device to access your target board.

At this point you can use GDB to load your program onto the target board and control its execution as required:

```
(gdb) load
Loading section .text, size 0xaa0 lma 0x0
Loading section .ARM.exidx, size 0x8 lma 0xaa0
Loading section .data, size 0xfc lma 0xaa8
Start address 0x11, load size 2980
Transfer rate: 6231 bits/sec, 596 bytes/write.
```

Set a breakpoint so that the debugger stops when your program reaches `main`:

```
(gdb) break main
Breakpoint 1 at 0xf4: file hello.c, line 5.
```

If you continue (begin) execution of your program and you are using an ARMUSB device, GDB will initially stop at the `start` symbol of your program:

```
(gdb) continue
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
start () at /buildpath/newlib-stable/libgloss/arm/crt0.S:50
50      in /buildpath/newlib-stable/libgloss/arm/crt0.S
```

Then, allow the program to execute until it reaches `main`:

```
(gdb) continue
Continuing.

Breakpoint 1, main () at hello.c:5
5         printf ("Hello world\n");
Current language:  auto; currently c
(gdb) next
Hello world
6         return 0;
```

Permit the program to finish executing with:

```
(gdb) continue
Continuing.

Program exited normally.
```

3.1.2.1 Invoking Sourcery G++ Debug Sprite

The debug sprite is invoked as follows:

```
arm-uclinuxeabi-sprite [options] device-url config-file
```

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

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

format. Most device URL schemes also follow the regular

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

format. The meanings of *hostname*, *port*, *path* and *options* parts depend on the *scheme* and are described below. The following schemes are supported:

armusb Use an ARMUSB debugging device. Refer to Section 3.1.2.1.2, “ARMUSB Devices”.

rdi Use an RDI debugging device. Refer to Section 3.1.2.1.3, “Remote Debug Interface Devices”.

The optional *?options* portion is allowed in all schemes. This allows additional device-specific options of the form `name=value`. Multiple options are concatenated using `&`. Some options are required for proper operation of some devices, e.g. the RDI scheme must be given the `rdi-library` and `rdi-config` options.

The *config-file* specifies an XML configuration file that describes the memory map and features of the target board. If *config-file* refers to a file (via a relative or absolute pathname), it is read. Otherwise, *config-file* can be a board name, and the toolchain's board config directory is searched for a matching file. The `-b` option lists config files in the board config directory. See Section 3.1.2.4, “Config File Syntax” for the syntax of the configuration files.

3.1.2.1.1 Sourcery G++ Debug Sprite Options

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

-b Print a list of *config* 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. Note that no currently-implemented device types report scanned devices, as each may only connect to a single auto-detected 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.

3.1.2.1.2 ARMUSB Devices

ARMUSB debug devices are supported. There are no valid hostname, port or path settings for the *`device-url`*, so it is simply specified as:

```
armusb:[///][?options]
```

The following options are permitted:

speed=*`speed`* — Specify the speed of the connection, from `0` (fastest, default) to `7` (slowest). Depending on the CPU speed of the target board, lower values may lead to unreliable communication with the target. It is recommended to use slower speeds in that case.

3.1.2.1.3 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:[///][?options]
```

The following 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.

3.1.2.2 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 ARM board.

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
```

starts the sprite listening on port 10000. 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.

3.1.2.3 Semihosting

Semihosting is implemented in a way which depends on the device type. ARMUSB and RDI both provide semihosting support, with slightly different sets of provided functions. For RDI, semihosting is implemented using RDI vector entries. The supported functions are:

- dbgprint
- dbgpause
- writec
- readc
- write
- gets
- reset
- message

For ARMUSB and other device types, semihosting is implemented using the Angel interface. The following functions are present:

- open
- close
- writec
- write0
- write
- read
- istty
- seek
- flen
- remove
- rename
- system
- errno
- exit

If appropriate, the C library uses semihosting to provide the low-level operations required for file access and other system facilities.

3.1.2.4 Config File Syntax

The *config-file* can be a user-written XML file to describe a non-standard board. The Sourcery G++ Debug Sprite searches for config 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:

```
<!-- Target description files for ARM.  -->
<!ELEMENT target-description
    (target-init|target-memory|target-features)*>
<!ELEMENT target-init
    (write-control-register|write-memory|delay)*>
<!ELEMENT write-control-register EMPTY>
<!ATTLIST write-control-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 target-memory (memory-device)*>
<!ELEMENT memory-device (description?)>
<!ATTLIST memory-device
    address CDATA   #REQUIRED
    size    CDATA   #REQUIRED
    type    CDATA   #REQUIRED
    device  CDATA   #IMPLIED>
<!ELEMENT description (#PCDATA)>

<!ELEMENT target-features
  (banked-regs|has-vfp|system-v6-m|system-v7-m)*>
<!ELEMENT banked-regs EMPTY>
<!ELEMENT has-vfp EMPTY>
<!ELEMENT system-v6-m EMPTY>
<!ELEMENT system-v7-m EMPTY>
```

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:

`<target-description>` This top-level element encapsulates the entire description of the target. It can contain `<target-init>`, `<target-features>`, and `<target-memory>` elements.

`<target-init>`	The `<target-init>` element must be present, but is currently unused. Future versions of the Sourcery G++ Debug Sprite may use this element to record a board-specific initialization sequence.
`<target-features>`	The `<target-features>` element specifies features of the target system. This element can occur at most once. It can contain `<banked-regs>`, `<has-vfp>`, `<system-v6-m>` and `<system-v7-m>` elements.
`<target-memory>`	This element describes the memory map of the target board. It is used by GDB to determine where software breakpoints may be used and when flash programming sequences must be used. This element can occur at most once. It can contain `<memory-device>` elements.
`<banked-regs>`	The `<banked-regs>` element specifies that the CPU of the target board has banked registers for different processor modes (supervisor, IRQ, etc.).
`<has-vfp>`	The `<has-vfp>` element specifies that the CPU of the target board has VFP registers.
`<system-v6-m>`	The `<system-v6-m>` element specifies that the CPU of the target board has ARMv6-M architecture system registers.
`<system-v7-m>`	The `<system-v7-m>` element specifies that the CPU of the target board has ARMv7-M architecture system registers.
`<memory-device>`	This element specifies a region of memory. It has four attributes: `address`, `size`, `type` and `device`. The `address` and `size` attributes specify the location of the memory device. The `type` attribute specifies that device as `ram`, `rom` or `flash`. The `device` attribute is required for `flash` regions; it specifies the flash device type.
`<description>`	This element encapsulates a human-readable description of its enclosing element.

3.2 Sourcery G++ Lite Release Notes

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

3.2.1 Changes in Sourcery G++ Lite 2007q3-51

Volatile postincrement and postdecrement bug fix. A code generation bug that caused postincrement or postdecrement of a volatile object to reread the modified value from that object in some contexts has been fixed. The bug affected code performing a comparison of the postincrement or postdecrement expression with a constant, or that was optimized to comparison with a constant.

Support for debugging with FlashPro3. Support has been added for debugging with the Actel FlashPro3 JTAG device on Windows hosts. This works only with Actel Cortex-M1 FPGAs.

C++ class debug information. The flag `-femit-class-debug-always` is now disabled by default. The flag produces duplicate C++ class debug information as a work-around for older debuggers.

Improved breakpoints in constructors and template functions. GDB now supports breakpoints on source code locations that have several code addresses associated with them. Setting a breakpoint on a constructor automatically associates the breakpoint with all constructor bodies generated by GCC. If you set a breakpoint on a line of a templated function, GDB breaks at the indicated line in all instantiations of the templated function.

GDB printf %p. GDB's **printf** command now supports the `"%p"` format specifier.

GDB update. The included version of GDB has been updated to 6.6.20070821. This update includes numerous bug fixes.

Assembler code file name suffixes. GCC now recognizes `.sx` as well as `.S` as a file name suffix indicating assembler code which must be preprocessed. The alternate suffix may be useful in conjunction with other program development tools on Windows that do not distinguish case on filenames and treat `.S` the same as `.s`, which GCC uses to indicate assembler code without preprocessing.

3.2.2 Changes in Sourcery G++ Lite 2007q3-33

Preprocessing assembly code. The compiler driver passes `-I` options to the assembler, so that `#include` directives (processed by the preprocessor) and `.include` directives (processed by the assembler) use the same search path.

uClibc `memcpy` and `memmove` functions. A bug that caused the uClibc implementations of `memcpy` and `memmove` to return incorrect values has been fixed.

Dynamically-initialized `const` variables. Dynamically-initialized namespace-scope C++ variables are no longer placed in read-only data sections, even when marked `const`. These variables must be modified at startup, so they cannot be placed in ROM, even though their values cannot change once initialized.

Register allocation bug fix. A register allocation bug has been fixed. Under rare circumstances, the bug caused incorrect code generation.

iWMMXt bug fix. A GCC bug affecting code generation for iWMMXt processors has been fixed. The bug caused internal compiler errors when compiling some functions with large stack frames.

NEON coprocessor system registers. The assembler now accepts the `MVFR0` and `MVFR1` coprocessor registers in `fmrx` and `fmxr` instructions.

Disabling diagnostics for use of system header and library directories. The warnings for use of options such as `-I/usr/include` when cross compiling can be disabled with a new option `-Wno-poison-system-directories`. This option is intended for use in chroot environments when such directories contain the correct headers and libraries for the target system rather than the host.

Default linker script. GCC no longer uses the simulator linker script by default. To avoid a link failure, you must specify a linker script explicitly with the `-T` command-line option, or via the `Properties` item on the `Project` menu in the Sourcery G++ Lite IDE.

Thumb-2 doubleword writeback addressing modes. An assembler bug that caused writeback addressing modes for `ldrd` and `strd` to be incorrectly encoded has been fixed.

Stricter check for anonymous unions. G++ now issues an error about invalid code that uses the same name for a member of an anonymous union and an entity in the surrounding namespace. For example, you will now get an error about code like:

```
int i;
static union { int i; };
```

because both the global variable and the anonymous union member are named `i`. To make this code valid you must change one of the declarations to use a different name.

GCC update. The GCC package has been updated to version 4.2.1. This version includes numerous bug fixes since GCC 4.2.

Smaller code for C++ destructors. G++ now generates more compact code to handle the destruction of C++ objects declared at namespace scope or declared within a function scope using the `static` keyword.

Robustness on Microsoft Windows. Defects that sometimes caused GDB to become non-responsive on Microsoft Windows have been eliminated.

Binutils update. The binutils package has been updated to the 2007-08-19 version of the pre-2.18 FSF trunk. This contains many new improvements and bug fixes. For more information, refer to the manuals for the individual utilities, and to the binutils web site at `http://www.gnu.org/software/binutils/`.

Debugging information fix. GCC no longer generates invalid debugging information for sections with no contents. The invalid debugging information caused the GNU/Linux prelinker to crash.

Calls to undefined weak symbols. The linker now implements semantics that comply to the ARM EABI for `R_ARM_CALL` and `T_ARM_THM_CALL` relocations against undefined weak symbols. These now result in a jump to the next instruction.

Thumb-2 shift instruction aliases. The assembler now accepts `mov` with shifted operands as an alias for Thumb-2 shift instructions. For example `mov r0, r1, lsl r2` is encoded as `lsl r0, r1, r2`.

Inlined function debugging fix. GDB now backtraces correctly when stopped at the first instruction of an inlined function. Earlier versions would sometimes encounter internal errors in this situation.

Assembler skipping \ characters. A bug is fixed where the assembler would skip `\` characters when they appeared at certain positions in the input file. This bug primarily affected assembler macros.

Improved diagnostics for region overflow. The linker will now give more helpful diagnostics when the object files being linked are too big for one of the memory regions defined in the linker script.

EABI object attribute merging. The linker now properly merges EABI object attributes into its output file.

Thumb-2 exception return instructions. An assembler bug that caused `subs pc, lr, #const` and `movs pc, lr` to be incorrectly encoded has been fixed.

Tag_ABI_PCS_wchar_t object attributes. Objects generated with `-fshort-wchar` are now given the correct `Tag_ABI_PCS_wchar_t` EABI object attribute annotations.

Spurious compiler warnings eliminated. GCC no longer emits warnings when linker-specific command-line options are provided in combination with modes that do not perform linking, such as with the `-c` flag.

Debugging of inlined functions. GDB now supports inlined functions. GDB can include inlined functions in the stack trace; display inlined functions' arguments and local variables; and step into, over, and out of inlined functions.

Uppercase special register names. The assembler now accepts both uppercase and lowercase special register names when assembling `msr` and `mrs` instructions for the Microcontroller profile of the ARM Architecture.

Debugger access to out-of-bounds memory. GDB turns on `inaccessible-by-default` by default, disallowing access to memory outside the regions specified in a board configulation.

Call shortening bug fix. GCC no longer overrides `__attribute__((long_call))` on calls to locally-defined functions when the function is weak, or when it is in a different section from the caller.

Binutils update. The binutils package has been updated from version 2.17 to the pre-2.18 FSF trunk. This is a significant update with many improvements and bug fixes.

Changes to the assembler (**as**) include:

- On MIPS targets, support for additional processors and the SmartMIPS and DSP Release 2 extensions has been added.

New linker (**ld**) features include:

- A new command-line option `--default-script` has been added to give more precise control over linker script processing.
- There are new command-line options `-Bsymbolic-functions`, `--dynamic-list`, `--dynamic-list-cpp-new`, and `--dynamic-list-data` to control symbols that should be dynamically linked.
- The new `--print-gc-sections` option lists sections removed by garbage collection.

Other changes include:

- The **objcopy** utility has a new `--extract-symbol` option to extract only symbol table information from the input file.
- The **gprof** utility now allows input files to have histogram records for several memory ranges, provided those ranges are disjoint.

For more information, refer to the manuals for the individual utilities, and the binutils web site at `http://www.gnu.org/software/binutils/`.

GDB update. The included version of GDB has been updated to 6.6.50.20070620. This update includes numerous bug fixes.

Forced alignment of array variables. A new option `-falign-arrays` has been added to the compiler. Specifying this option sets the minimum alignment for array variables to be the largest power of two less than or equal to their total storage size, or the biggest alignment used on the machine, whichever is smaller. This option may be helpful when compiling legacy code that uses type punning on arrays that does not strictly conform to the C standard.

ARM EABI compliance. Objects produced by Sourcery G++ Lite are now marked as ARM ELF version 5 rather than ARM ELF version 4. This reflects compliance with recent revisions of the ARM EABI. Sourcery G++ Lite still accepts objects marked with version 4.

Smaller C++ applications. The C++ runtime library has been modified so that using namespace-scope objects with destructors does not pull in unnecessary support functions. Therefore, statically linked C++ applications compiled with `-fno-exceptions` are substantially smaller.

ARMv6-M floating-point bug fix. A bug affecting conversion of wider floating-point types to subnormal `float` values on ARMv6-M processors has been fixed.

3.2.3 Changes in Sourcery G++ Lite 2007q1-21

NEON coprocessor system registers. The assembler now accepts the `MVFR0` and `MVFR1` coprocessor registers in `fmrx` and `fmxr` instructions.

Disabling diagnostics for use of system header and library directories. The warnings for use of options such as `-I/usr/include` when cross compiling can be disabled with a new option `-Wno-poison-system-directories`. This option is intended for use in chroot environments when such directories contain the correct headers and libraries for the target system rather than the host.

Thumb-2 doubleword writeback addressing modes. An assembler bug that caused writeback addressing modes for `ldrd` and `strd` to be incorrectly encoded has been fixed.

Thumb-2 shift instruction aliases. The assembler now accepts `mov` with shifted operands as an alias for Thumb-2 shift instructions. For example `mov r0, r1, lsl r2` is encoded as `lsl r0, r1, r2`.

EABI object attribute merging. The linker now properly merges EABI object attributes into its output file.

Thumb-2 exception return instructions. An assembler bug that caused `subs pc, lr, #const` and `movs pc, lr` to be incorrectly encoded has been fixed.

Tag_ABI_PCS_wchar_t object attributes. Objects generated with `-fshort-wchar` are now given the correct `Tag_ABI_PCS_wchar_t` EABI object attribute annotations.

Uppercase special register names. The assembler now accepts both uppercase and lowercase special register names when assembling `msr` and `mrs` instructions for the Microcontroller profile of the ARM Architecture.

3.2.4 Changes in Sourcery G++ Lite 2007q1-10

Disassembly of overlapping sections. A bug in the disassembler that caused code to be displayed as data (and vice-versa) in files with overlapping sections has been fixed. This mainly affects the **objdump** utility.

Installer hangs while refreshing environment. The Sourcery G++ installer for Microsoft Windows now updates the `PATH` environment variable without waiting for open applications to acknowledge the update. This change prevents open applications from blocking the installer's progress.

Improved assembler diagnostics for 8-bit offsets. The assembler now correctly diagnoses out-of-range offsets to instructions such as `LDRD` as 8-bit rather than half-word offsets.

Less disk space required for installation. Sourcery G++ Lite packages are smaller because multiple copies of files have been replaced with hard and/or symbolic links when possible. Both the size of the installer images and the amount of disk space required for an installed package have been reduced.

Thumb register corruption fix. A bug in the compiler that could cause register corruption in Thumb mode has been fixed. The compiler was formerly emitting code to restore registers on function return that was not interrupt safe.

__aeabi_lcmp. An error in the libgcc implementation of `__aeabi_lcmp` that caused incorrect results to be returned has been fixed. This is a support routine defined by the ARM EABI. GCC does not normally use this routine directly, however it may be used by third-party code.

The \@ assembler pseudo-variable. A bug in the assembler that caused uses of the `\@` pseudo-variable to be mis-parsed as comments has been fixed.

Crash when generating vector code. A bug that sometimes caused the compiler to crash when invoked with the `-ftree-vectorize` option has been fixed.

Propagation of Thumb symbol attributes. Symbols referring to Thumb functions on ARM targets now have their Thumb attribute correctly propagated to any aliases defined with `.set` or `.symver`.

Linking of non-ELF images. A linker bug that could cause a crash when linking non-ELF objects for ARM targets has been fixed.

Invalid load instructions. A bug in the compiler which caused it to generate invalid assembly (e.g. `ldrd r0, [#0, r2]`) has been fixed.

VFPv3/NEON debug information. A bug in the compiler which caused it to generate incorrect debug information for code using VFPv3/NEON registers has been fixed. The debugger is now able unable to locate and display values held in these registers.

ARMv6-M system instructions. An assembler bug that caused some ARMv6-M system instructions to be incorrectly rejected has been fixed. The affected instructions are `msr`, `mrs`, `yield`, `wfi`, `wfe` and `sev`.

Assembly of Thumb-2 load/store multiple instructions. The Thumb-2 `ldm` and `stm` assembly mnemonics are now assembled to `ldr` and `str` instructions when a single register is transferred, as specified in the Thumb-2 Architecture Supplement.

Conditional Thumb-2 branch instructions. A linker bug that could cause objects involving conditional Thumb-2 branch instructions to be incorrectly rejected has been fixed.

Alignment bug fix. A bug has been fixed that formerly caused incorrect code to be generated in some situations for copying structure arguments being passed by value. The incorrect code caused alignment errors on stack accesses on some targets.

3.2.5 Changes in Sourcery G++ Lite 2007q1-3

Initial release. This is the initial release for ARM uClinux.

Chapter 4
Installation and Configuration

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

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

4.1 Terminology

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

4.2 System Requirements

4.2.1 Host Operating System Requirements

Sourcery G++ Lite supports the following host operating systems:

- Microsoft Windows NT 4, Windows 2000, Windows XP, and Windows Vista systems using IA32, AMD64, and EM64T processors.
- GNU/Linux systems using the IA32, AMD64, or EM64T processors, including Debian 3.0 (and later), Red Hat Enterprise Linux 3 (and later), SuSE Enterprise Linux 8 (and later).
- Solaris 2.8 (and later) systems using SPARC processors.

Not all combinations of host and target systems are available. Therefore, Sourcery G++ Lite for your target system may not be available on all of the above host systems.

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

4.2.2 Host Hardware Requirements

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

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

4.2.3 Target System Requirements

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

4.3 Downloading an Installer

If you have received Sourcery G++ Lite on a CD, or other physical media, then you do not need to download an installer. You may skip ahead to Section 4.4, “Installing Sourcery G++ Lite”.

If you have a Sourcery G++ Lite subscription (or evaluation), then you can log into the Sourcery G++ Portal[1] to download your Sourcery G++ Lite toolchain(s). CodeSourcery also makes some toolchains available to the general public from the Sourcery G++ web site[2]. 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++ Lite installer is provided as an executable, with the `.exe` extension. For GNU/Linux systems with an X Window System, Sourcery G++ Lite is provided as a graphical installer with the `.bin` extension. For Solaris, and GNU/Linux systems without an X Window System, Sourcery G++ Lite is provided as a compressed archive `.tar.bz2`. If installing on a RPM-based GNU/Linux system you may download the `.rpm` file.

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

4.4 Installing Sourcery G++ Lite

The method used to install Sourcery G++ Lite depends on your host system.

4.4.1 Installing Sourcery G++ Lite on Microsoft Windows

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

After the installer starts, follow the on-screen dialogs to install Sourcery G++ Lite. This package comes with a bundled Java Runtime Environment; you do not have to download any additional software.

4.4.2 Installing Sourcery G++ Lite on GNU/Linux systems with an X Window System

Start the graphical installer by invoking the executable shell script:

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

After the installer starts, follow the on-screen dialogs to install Sourcery G++ Lite. This package comes with a bundled Java Runtime Environment; you do not have to download any additional software.

4.4.3 Installing Sourcery G++ Lite on Solaris or GNU/Linux systems without an X Window System

You do not need to be a system administrator to install Sourcery G++ Lite on a GNU/Linux or Solaris system. You may install Sourcery G++ Lite using any user account and in any directory to which you have write access. This guide assumes that you have decided to install Sourcery G++ Lite in the `$HOME/CodeSourcery` subdirectory of your home directory and that the filename of the package

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

you have downloaded is `/path/to/package.tar.bz2`. After installation the toolchain will be in `$HOME/CodeSourcery/sourceryg++-4.1` or similar.

First, uncompress the package file:

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

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

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

Change to the installation directory:

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

Unpack the package:

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

If you are installing a native toolchain, it is then necessary to run a post-install script found in the `share` directory:

```
> /bin/sh sourceryg++-4.1/share/postinst-*
```

The `.tar.bz2` package is not bundled with a Java Runtime Environment.

4.4.4 Installing Sourcery G++ Lite on RPM-based GNU/Linux systems

On a RPM-based system you should use RPM to install the provided package. Execute the following command as root (administrator):

```
> rpm -ivh /path/to/package.rpm
```

The `.rpm` package is not bundled with a Java Runtime Environment.

4.4.5 Installing the Java Runtime Environment

Some versions of Sourcery G++ Lite include the Eclipse Integrated Development Environment. Because Eclipse is an optional component, the installer allows you to choose whether or not to install it. Eclipse is a Java application and requires the Java Runtime Environment (JRE). The Java Runtime Environment is available at no charge from Sun Microsystems Java website[3]. You may download either the Java Runtime Environment (JRE) or the Java Development Kit (JDK). (The JDK includes the JRE.)

4.5 Uninstalling Sourcery G++ Lite

The method used to uninstall Sourcery G++ Lite depends on your host system. 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.

[3] http://java.sun.com/j2se/

4.5.1 Uninstalling Sourcery G++ Lite on Microsoft Windows

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++ Lite.

4.5.2 Uninstalling Sourcery G++ Lite on Microsoft Windows Vista

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++ Lite.

4.5.3 Uninstalling Sourcery G++ Lite on GNU/Linux using the graphical uninstaller

If you installed on GNU/Linux using the graphical installer, then you must use the graphical uninstaller to remove Sourcery G++ Lite. 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 shell script:

```
> /bin/sh ./path/to/install/\
        Sourcery_G++/\
        Uninstall_Sourcery_GXX_for_ARM uClinux
```

After the uninstaller starts, follow the on-screen dialogs to uninstall Sourcery G++ Lite.

4.5.4 Uninstalling Sourcery G++ Lite on RPM-based GNU/Linux systems

On a RPM-based system you should use RPM to uninstall the installed package. Execute the following command as root (administrator):

```
> rpm -e sourceryg++-arm-uclinuxeabi
```

4.5.5 Uninstalling Sourcery G++ Lite on GNU/Linux

If you did not use the graphical installer or RPM, uninstall Sourcery G++ Lite by manually deleting the installation directory created in the install procedure.

4.6 Setting up the Environment

As with the installation process itself, the steps required to set up your environment depend on your host operating system. The name of the Sourcery G++ Lite commands all begin with **arm-uclinuxeabi** so that you can install Sourcery G++ Lite for multiple target systems in the same directory.

4.6.1 Setting up the Environment on Microsoft Windows

On a non-Vista Microsoft Windows system, the installer automatically adds Sourcery G++ Lite to your `PATH`. You can test that your `PATH` is set up correctly by using the following command:

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

and verifying that the last line of the output contains: `Sourcery G++ 2007q3-51`.

On a Microsoft Windows Vista system, the installer does not automatically add Sourcery G++ Lite to your `PATH`. The Sourcery G++ IDE does not need this step to function correctly. This step is only required if you wish to use the tools from the command line on a Microsoft Windows Vista system. To set up your `PATH` on Microsoft Windows Vista, use the following command in a `cmd.exe` shell:

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

where `C:\Program Files\Sourcery G++` should be changed to the path of your Sourcery G++ Lite installation. You can verify that the command worked by starting a second `cmd.exe` shell and running:

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

Verify that the last line of the output contains: `Sourcery G++ 2007q3-51`.

4.6.1.1 Working with Cygwin

Sourcery G++ Lite does not require Cygwin or any other UNIX emulation environment. You can use Sourcery G++ Lite directly from the Eclipse IDE or from the Windows command shell. You can also use Sourcery G++ Lite 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++ Lite is not a Cygwin application, it does not, by default, recognize Cygwin paths.

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

```
> export CYGPATH=cygpath
```

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

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

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

4.6.2 Setting up the Environment on GNU/Linux or Solaris

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

Before using Sourcery G++ Lite you should add Sourcery G++ Lite 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 $HOME/CodeSourcery/sourceryg++-4.1/bin:$PATH
```

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

```
> export PATH=$HOME/CodeSourcery/sourceryg++-4.1/bin:$PATH
```

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

You may also wish to set the `MANPATH` environment variable so that you can access the Sourcery G++ Lite manual pages, which provide additional information about using Sourcery G++ Lite. To set the `MANPATH` environment variable, follow the same steps shown above, replacing `PATH` with `MANPATH`, and `bin` with `share/doc/sourceryg++-2007q3-51-arm-uclinuxeabi/man`.

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

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

and verifying that you receive the message:

```
arm-uclinuxeabi-g++: no input files
```

Chapter 5
Using the Sourcery G++ IDE

This chapter explains how to use the Sourcery G++ IDE provided in the Personal and Professional editions of this product. The Sourcery G++ IDE is not included in Sourcery G++ Lite.

5.1 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.

This chapter is divided into two sections. The first explains how to create and build a program; the second section explains how to debug and run a program once it has been built.

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 visit the Eclipse web site[1].

5.2 Building Applications

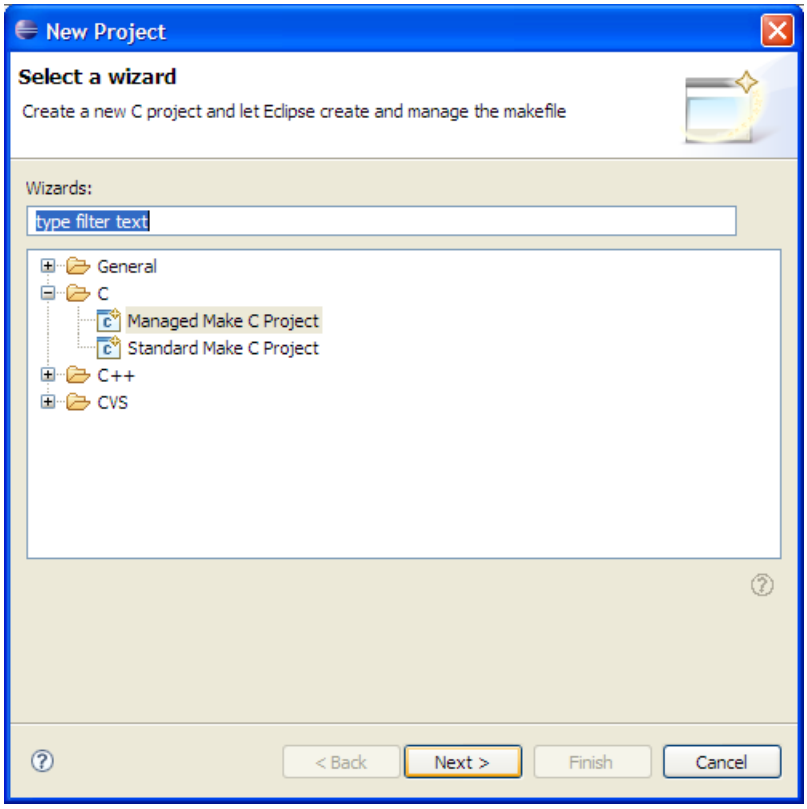
In the Sourcery G++ IDE, every program is a *project*. The project contains all of the source files required to build the program. So, the first step is to create a project.

There are two kinds of projects: "Managed Make" and "Standard Make" projects. In general, if you intend to do all of your development from within the IDE, you should use a Managed Make project. In this mode, the IDE automatically handles building your project for you. However, if you are working with code that has previously been built with **make**, you may wish to use a Standard Make project instead. The following several sections explain how to create and work with a Managed Make project. If you wish to use a Standard Make project instead, skip ahead to section Section 5.2.5, "Using Standard Make Mode".

5.2.1 Setting Up

Create a new project by selecting `File` → `New` → `Project`. Expand the `C` label and select `Managed Make C Project`. (If you want to build a C++ application, expand the `C++` label instead.) Click the `Next` button.

[1] http://www.eclipse.org

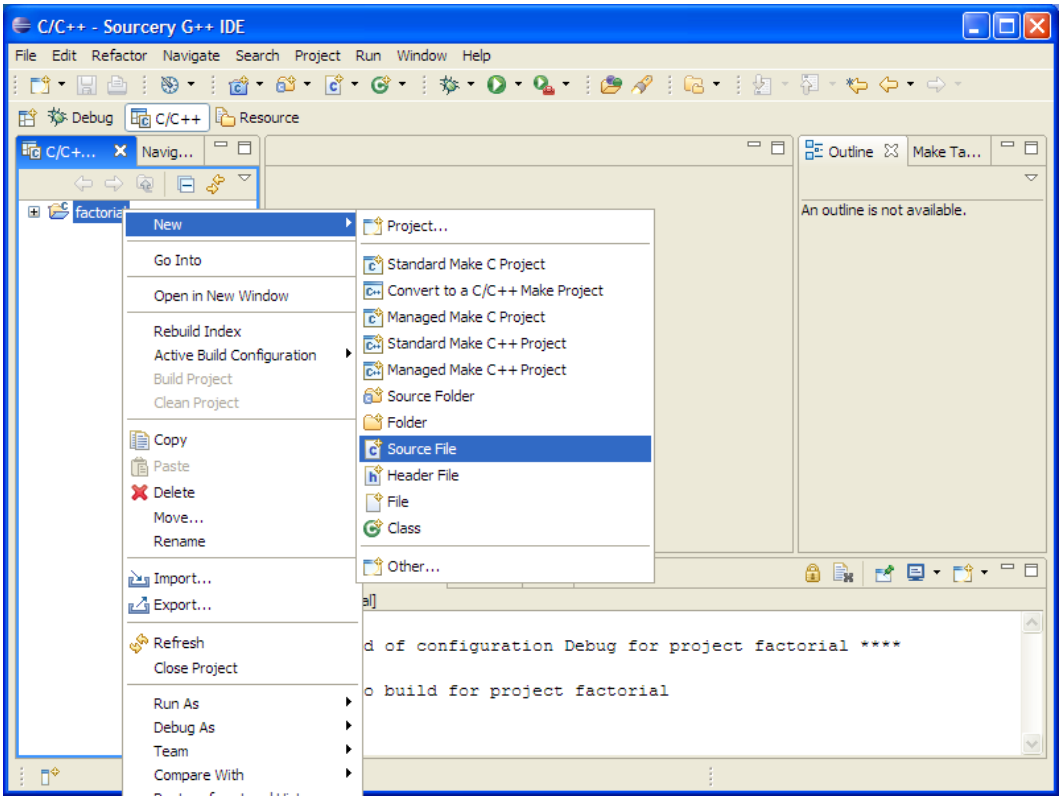


Expand the `C` folder and select the Managed Make C Project to create a new program.

Give the project the name `factorial` and click the `Next` button. From the `Project Type` menu select `Executable (Sourcery G++ for ARM uClinux)` and click `Finish`. If you are asked whether or not to open a new perspective, click the `Yes` button.

On ELF and EABI targets, you must choose a target board before you can link your application. On all targets, you may wish to choose a CPU other than the default so that Sourcery G++ can optimize for your processor. To set these options right-click on the `factorial` project, and select `Properties`. From the list on the left select `C/C++ Build`. From the `Configuration Settings` panel select the `Tool Settings` tab. Select the `Target` menu from the list and choose your target-specific options. If you have no target board use the `Simulator` board option, when available; otherwise choose the first board on the list.

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.



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

5.2.2 Writing Source Code

Whenever you create or save a file, the Sourcery G++ IDE attempts to rebuild the program. Because the program is empty at this point, the compilation does not succeed, and you may notice some messages in the `Console` tab indicating errors. Those errors will go away when the program is completed.

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;
}
```

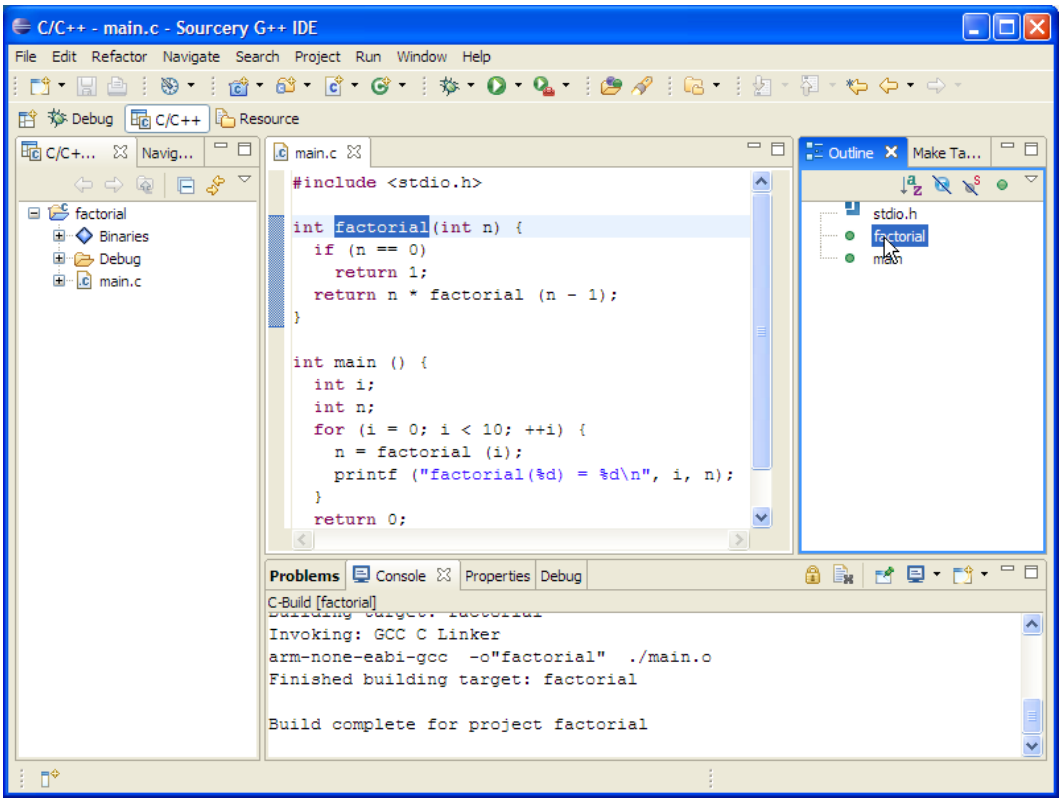
When you are done, save the file with `File` → `Save` (**Ctrl-S**).

When you save the file, the Sourcery G++ IDE rebuilds the project. The output of the commands run by the IDE is displayed in the `Console` tab. You should see the following output at the bottom of the console:

```
Build complete for project factorial
```

5.2.3 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.



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

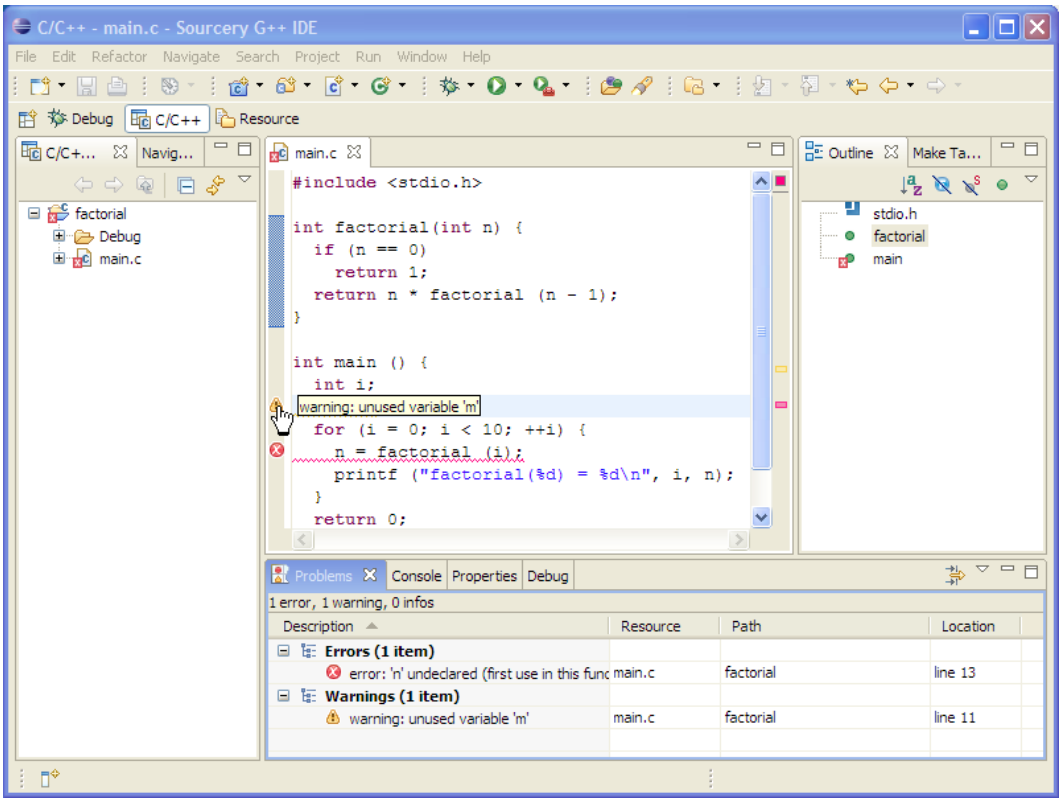
You can also use the cross-reference information to jump from the place where a function is called to the definition of the function. For example, find the line in `main` which calls `factorial` and place the cursor over the name `factorial`. Then, right-click and select `Open Declaration` (**F3**) to jump to the point at which `factorial` is declared. The cross-reference functionality works even if the function call is in a different file from the declaration of the function.

5.2.4 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.



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

Before proceeding, you must correct the error by changing `m` back to `n`.

5.2.5 Using Standard Make Mode

This section explains how to use the advanced Standard Make mode, instead of the simpler Managed Make mode described above. If you are just getting started with Sourcery G++, you should skip this section and proceed directly to Section 5.3, “Debugging Applications”.

Caution

Using Standard Make Mode 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 Standard Make project, instead of a Managed Make project. In Standard Make 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.

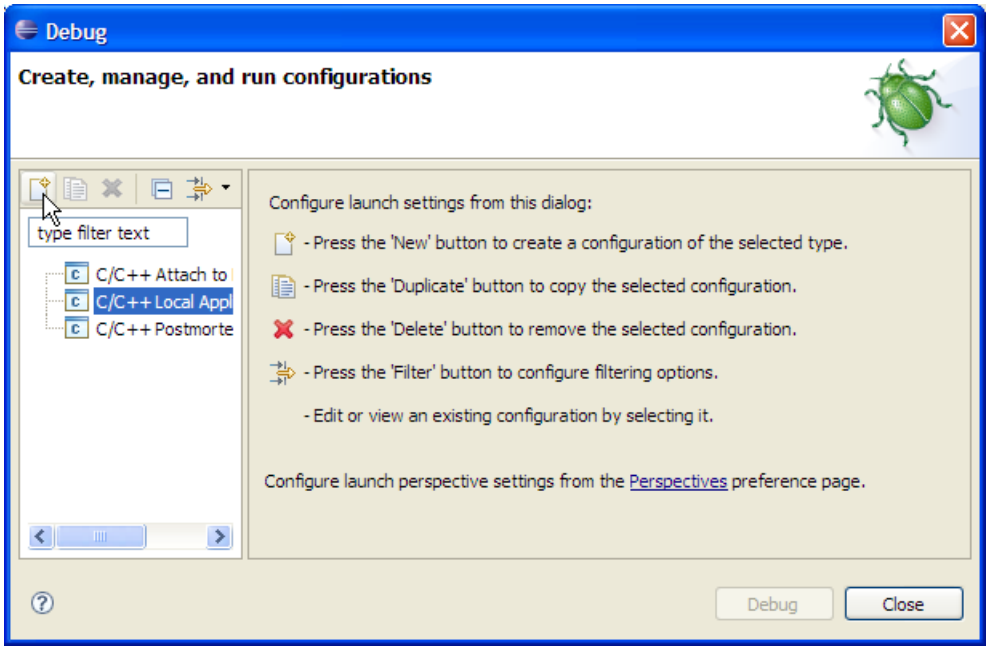
To set up the Standard Make mode to work with Sourcery G++, you have to make a few changes to the default project settings. When you create the project, the IDE displays a window that permits you to define the project settings.

Select the `Discovery Options` tab and set the `Compiler invocation command` to **arm-uclinuxeabi-gcc** instead of the default **gcc**. That change tells the IDE to use the Sourcery G++ compilers when scanning your program code to determine cross-reference information. 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**.

5.3 Debugging Applications

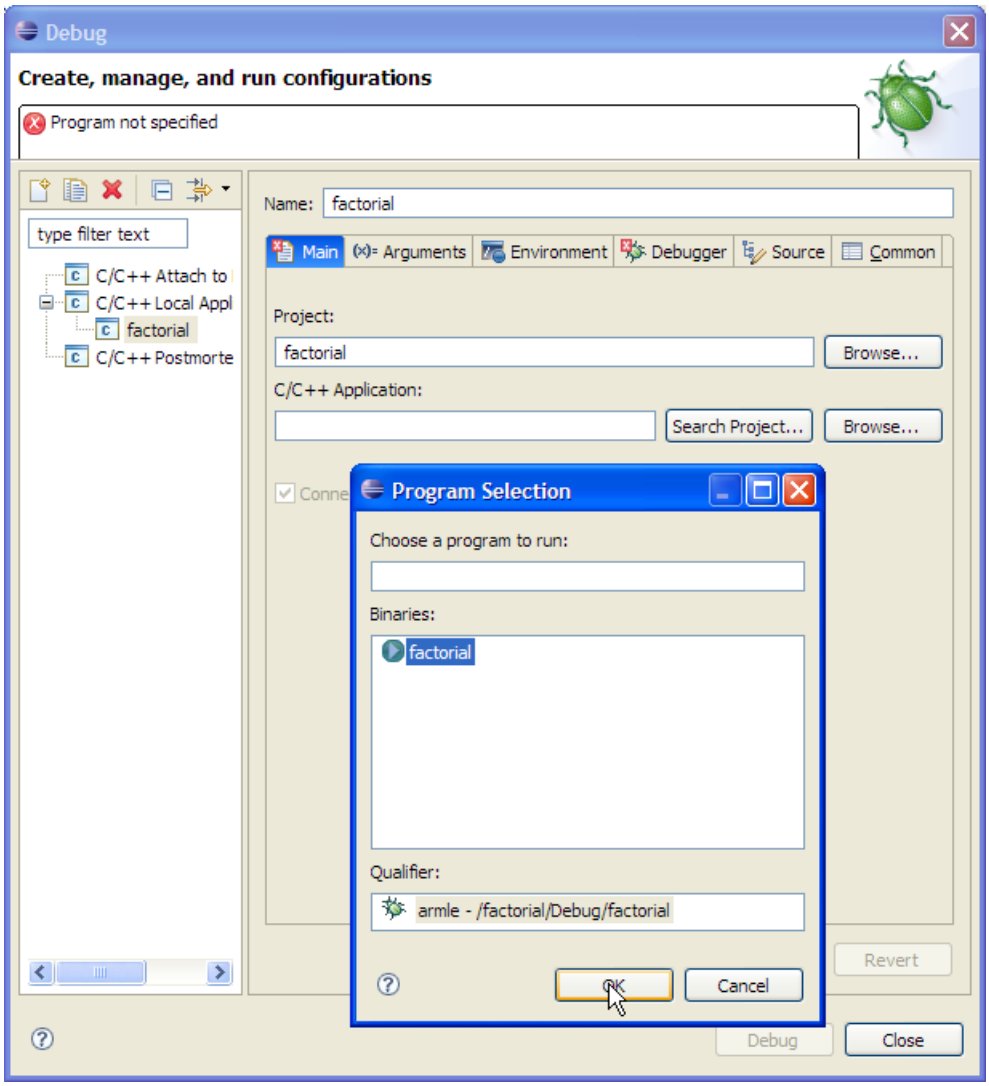
5.3.1 Starting the Debugger

After you build your application, choose `Run`. Select the `C/C++ Local Application` label in the `Configurations` pane. Then, click the `New` icon  positioned towards the upper left of the window.



Click the `New` icon to create to create a new debug configuration.

When you create the launch configuration, a new window appears. On the `Main` tab, use the `Browse...` button to select your project, if it is not already selected. Then, use the `Search Project..` button to select your application.



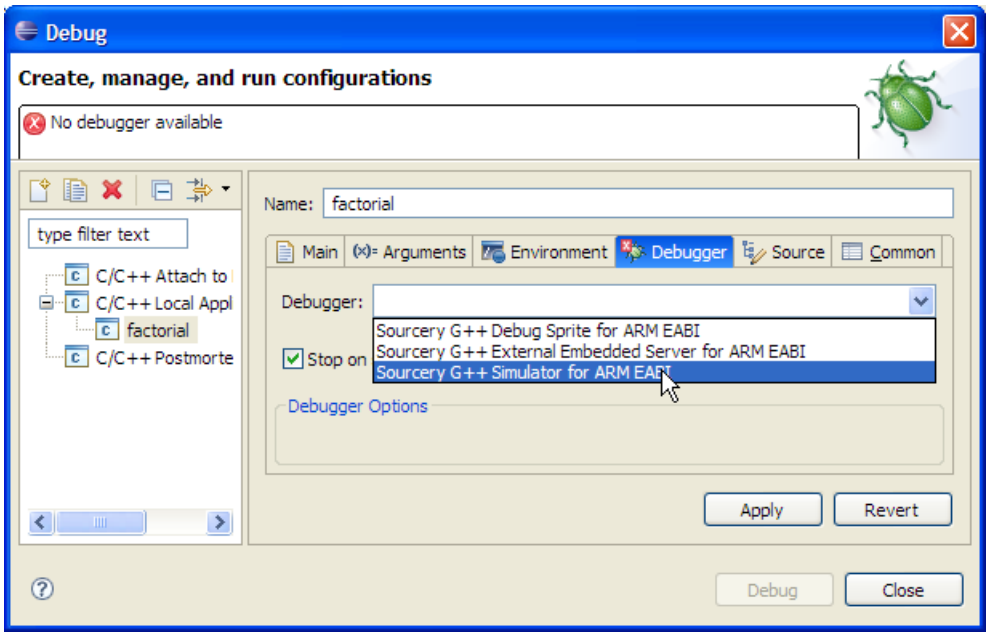
Use the `Search Project...` button to locate your program.

5.3.2 Choosing a Debugging Mode

Before you can use the Sourcery G++ IDE to debug your application, you must decide which debugging mode to use. Sourcery G++ supports several debugging modes, as described below.

5.3.2.1 Selecting a Debugger

Once you have decided which debugger to use, switch to the `Debugger` tab and select the appropriate Sourcery G++ option.



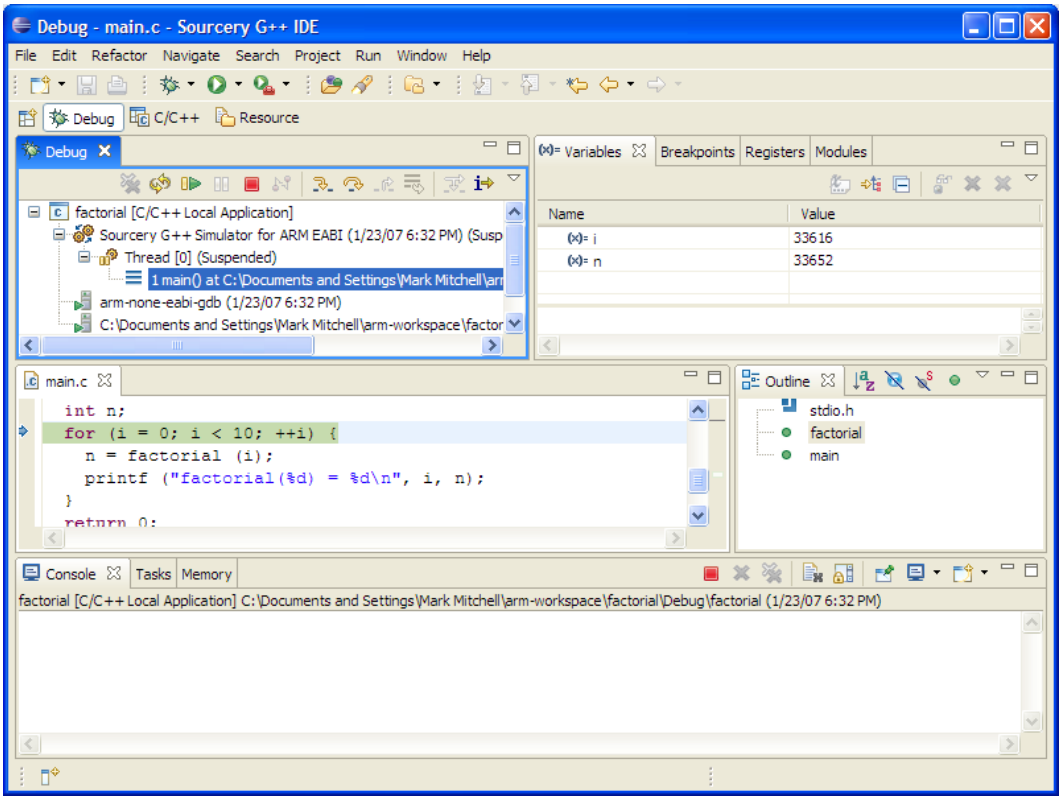
Pick the debugger that you want to use.

Once you have made any necessary adjustments, click the `Debug` button to start the debugger.

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

5.3.3 Controlling Execution

When you start the debugger, the IDE switches from the C/C++ perspective to the debug perspective. Instead of showing panes that help you to develop your application, the IDE now shows panes that help you to debug your application.

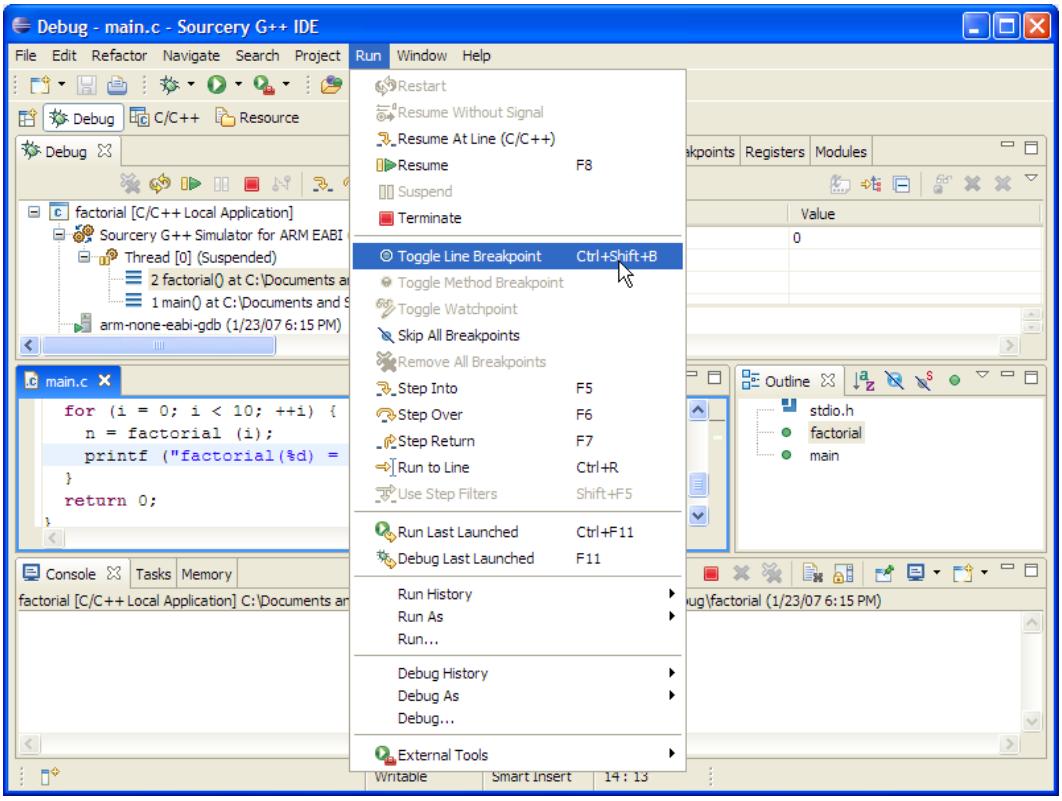


The debug perspective displays the stack, local variables, and the current location.

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` (**Ctrl-Shift-B**).



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`.

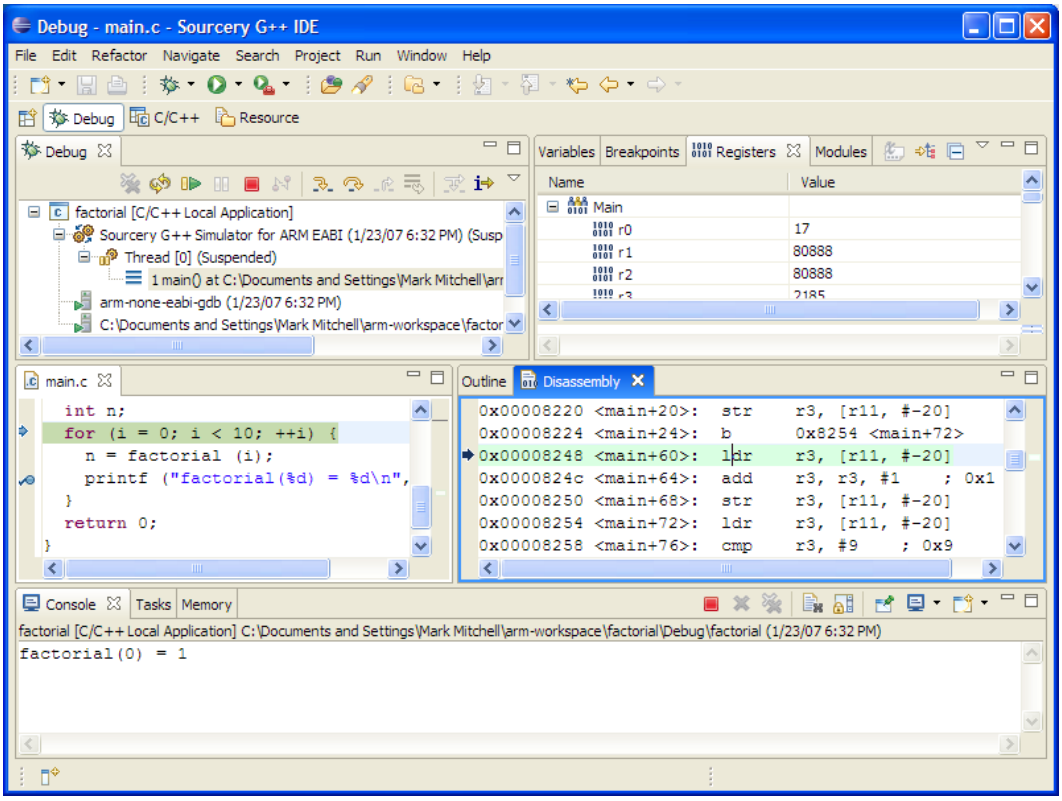
The IDE no longer displays the value of `i` because there is no local variable `i` within `factorial`. If you wish to see the value of `i` (from `main`), select the stack frame for `main` in the pane at the upper left. The IDE displays the variables for whichever frame is presently selected.

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.

5.3.4 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`.



The Sourcery G++ IDE can display machine registers and assembly code.

When the disassembly window is active, 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.

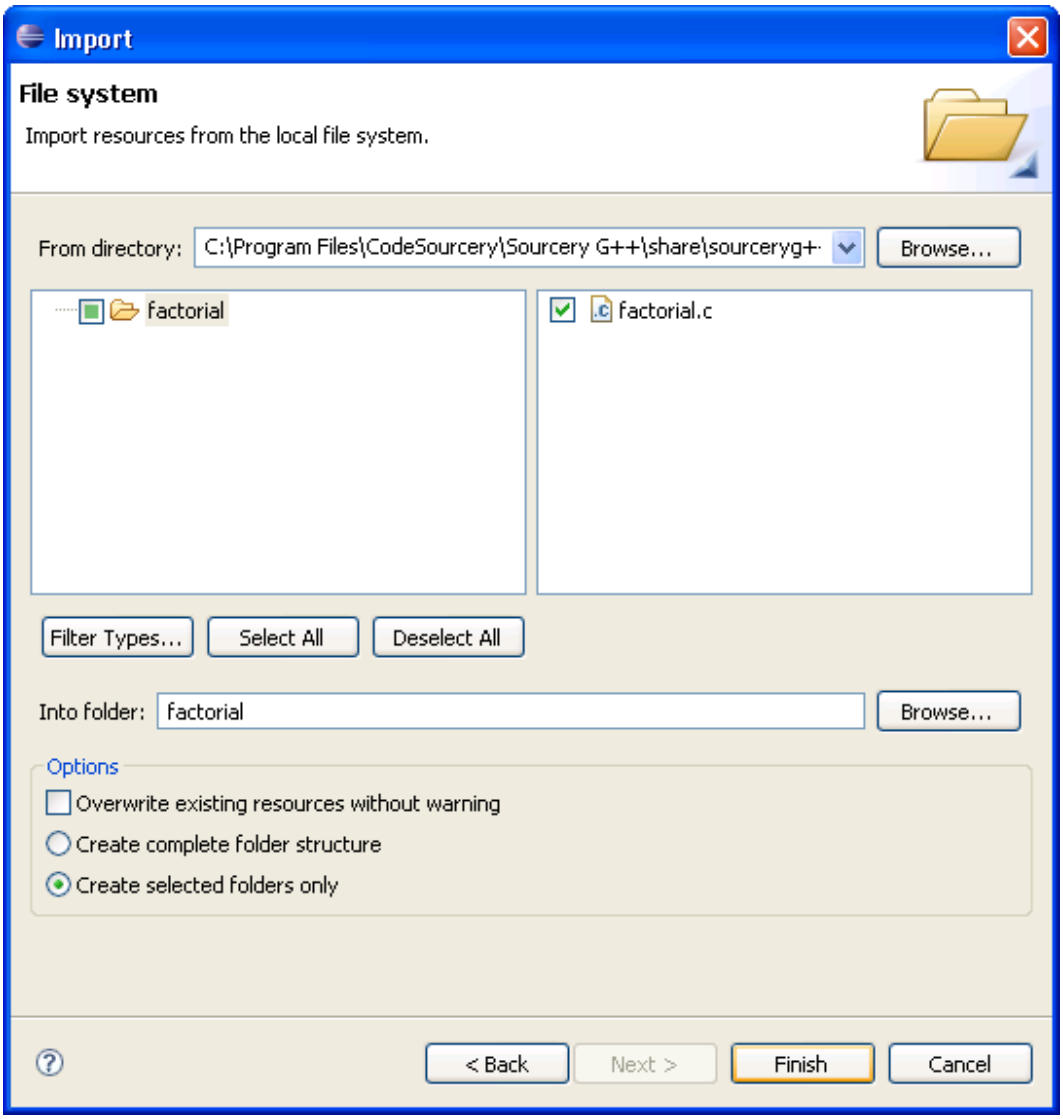
5.3.5 Troubleshooting

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.

5.4 Sample Programs

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

Start by following the steps described in Section 5.2, “Building Applications”. Call the project `sample`, and do not create a new source file.



Click `Finish` to import the selected 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, 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`.

The “Managed Make C project” compiles automatically. Your sample program is now ready to execute or debug. Please refer to Section 5.3, “Debugging Applications” for instructions on how to debug the target application.

Chapter 6
Using Sourcery G++ from the Command Line

This chapter demonstrates the use of Sourcery G++ Lite from the command line. This chapter assumes you have installed Sourcery G++ Lite as described in Chapter 4, *Installation and Configuration*. If you prefer to use an integrated development environment to build your applications, you may refer to Chapter 5, *Using the Sourcery G++ IDE* instead.

6.1 Building an Application

This chapter explains how to build an application with Sourcery G++ Lite using the command line. As elsewhere in this manual, this section assumes that your target system is arm-uclinuxeabi. If you are using a different target system, you must replace commands that begin with **arm-uclinuxeabi** with the name of your target system.

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

```
#include <stdio.h>

int
main (void)
{
  printf("Hello World!\n");
  return 0;
}
```

Compile and link this program using the command:

```
> arm-uclinuxeabi-gcc -o hello hello.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++**.)

Sourcery G++ Lite may require that you specify a linker script to build the application. If you receive linker errors like “undefined reference to `read'”, then you must select an appropriate linker script for your target system. Default linker scripts are provided in `arm-uclinuxeabi/lib`. You may use a linker script by adding the following `-T script` to the compiler command line.

6.2 Running an Application

If the target system is the same as the host system (e.g., if you are running Sourcery G++ Lite on IA32 GNU/Linux to build an application for IA32 GNU/Linux), then you can just run the resulting application. On a Microsoft Windows system, you may use the command:

```
> hello
```

On a GNU/Linux or Solaris system, use the slightly more complex:

```
> ./hello
```

command. In either case, you should see:

```
Hello world!
```

If the target system is not the same as the host system, then you cannot run the application directly. Instead, you must run the application on the target system. You should consult the manuals for your target system to determine the exact procedures required to run the application.

On some systems, Sourcery G++ Lite includes a simulator that can be used to run the program. To use the simulator run:

```
> arm-uclinuxeabi-run hello
```

The simulator is available if you see the expected output:

```
Hello, world!
```

There is no simulator for your target system if you see a message like:

```
'arm-uclinuxeabi-run' is not recognized \
as an internal or external command
```

or:

```
arm-uclinuxeabi-run: command not found
```