



Installing CSIM 20 for UNIX / Linux / Mac OS X / Cygwin Systems

Mesquite Software, Inc.
PO Box 26306
Austin, TX 78755-0306
(800) 538-9153 or (512) 338-9153
info@mesquite.com
www.mesquite.com

CSIM is copyrighted by Microelectronics and Computer Technology Corporation, 1985-1994. Its use is covered by the CSIM Software License Agreement included with the software. This manual is copyrighted by Mesquite Software, Inc. It may not be copied without written permission from Mesquite Software, Inc.

Installing CSIM 20 for UNIX / LINUX / MAC OS X / Cygwin Systems

Introduction

This document describes how to install CSIM 20 on a UNIX, Linux, Mac OS X or Windows/Cygwin system for use with most of the compilers on such systems. It then describes how to compile and execute a CSIM 20 program on this class of platforms.

Installation

The CSIM 20 distribution package includes a set of directories and files that are either downloaded from the Internet or shipped on a compact disk (CD). Both distribution methods contain the same CSIM 20 files and documentation. If you downloaded CSIM 20 from the Mesquite website, you should have received a .zip archive file. You will need a program capable of extracting ZIP archives.

If you obtained a CD copy of CSIM 20, simply copy the contents of the CD to a location of your choice on your system's hard drive. For Internet download packages, simply extract the contents of the downloaded archive file to a location of your choice on your system's hard disk. For the purposes of this document, we will assume the CSIM 20 files have been placed in `/home/myusername/csim20`.

The CSIM 20 distribution contains a hierarchy of directories as follows:

```
Documentation
<platform name>
  32-bit
    <compiler name>
      <csim build files>
      lib
        <csim library>
  64-bit
    <compiler name>
      <csim build files>
      lib
        <csim library>
```

As an example, for a Linux 32-bit system using gcc as the compiler, the directory structure appears like this:

```
Documentation
Linux
  32-bit
    gcc
      lib
        csim.gcc.a
        csim.h
        <various build scripts and source files>
```

Similarly, for a Linux 32-bit system using g++ as the compiler, the directory structure appears like this:

```
Documentation
Linux
    32-bit
        g++
            lib
                csim.gpp.a
                csim.h
                cpp.h
                <various build scripts and source files>
```

Note that the name of the directory (`gcc / g++`) denotes the C vs. the C++ versions.

Compiling and Executing CSIM 20 Programs

The usual procedure is to compile and execute the first test case (`ex1.c` in the CSIM 20 directory). To do this, execute the following UNIX commands in the CSIM 20 directory:

```
./csim.gcc ex1.c
./a.out
```

Note: In Cygwin, the command to initiate execution of the model is “`./a.exe`”.

Here, `csim.gcc` is a simple script that is included in the CSIM 20 directory as a shortcut for building CSIM programs. While this script should work on most systems, it may sometimes be necessary to modify the script to fit your particular environment. For example, if your desired compiler is not in your shell path, you will need to add the full path to your compiler to the command in the script. You may also need to modify the include paths to reflect the complete path to the CSIM 20 directory if you wish to copy the script elsewhere on your system.

To compile CSIM 20 programs of your own in other directories, you need to make a copy of the script file “`csim`” and modify it to give valid path names for the include directory and for the `csim.a` file. For example, if the complete path name for the CSIM 20 directory is “`/home/john/csim20`”, then the CSIM script file should look like:

```
"cc -I/home/john/csim20/lib $* /home/john/csim20/lib/csim.a -lm"
```

The Gnu C/C++ compiler (`gcc` or `g++`) can be used in developing simulation models of systems. Note that for the C++ version of CSIM, it is necessary to define 'CPP' in the preprocessor definitions by adding '-DCPP' to the command line.

A Note to Windows Users:

CSIM 20 is supported on Windows systems with the GCC compilers under the Cygwin environment, and this is the only configuration that has been properly tested. Other configurations of GCC on Windows will probably work as well, such as the MinGW compiler, the MSYS POSIX environment, or the Bloodshed Dev-C++ IDE. However, these configurations have not been tested by Mesquite Software, so your results with these setups may vary. For information on using CSIM 20 with Microsoft's Visual Studio compilers, see the document Installing CSIM 20 for Microsoft Windows and Visual Studio, also included with this distribution.

32-bit vs. 64-bit Code

CSIM 20 now supports 64-bit platforms in the professional version. The primary advantage of building CSIM programs with 64-bit support is that it allows for a much larger address space, which, in turn, allows much larger and more complex models to be built and run. Most use cases will not encounter the limits of 32-bit mode, but for very large models, 64-bit compilation may be required. In addition to allowing larger models, compiling CSIM programs with 64-bit support will often yield better performance, but this result is not always true. For example, on the x86 architecture, x86-64 offers many more improvements and optimizations than just 64-bit mode. Thus, on x86-64 platforms, compiling for 64-bit will usually result in faster performance of CSIM models. However, this performance improvement may not occur on all platforms. For detailed help determining the need, capability, and advantages of using 64-bit mode on your specific platform with your specific usage needs, please feel free to contact Mesquite Software.

Writing CSIM 20 Programs

The major objects in a CSIM 20 simulation model (program) are the resources of the model and the processes (entities) that will “use” these resources. Thus, designing a CSIM 20 model usually proceeds as follows:

1. Determine the resources needed by the model. In CSIM 20, these resources can be of the following types:

- Facilities – single server, multiserver-server or sets of facilities
- Storage blocks – either a single block or a set of blocks

These resources are usually “global;” that is, declared outside of the context of a procedure. In addition, these resources must be initialized prior to their use.

2. Determine the processes of the model:

- The first process is normally named *sim*. If this is not done, then the programmer must provide a *main* procedure, which is used instead of the *main* provided in the CSIM 20 library. *Sim* is called with the arguments *argc* and *argv*, just as *main* is called in a normal C program.
- A CSIM 20 process is a C procedure that executes a *create* statement. This *create* statement is executed at the beginning of the procedure. There is no predefined relationship between CSIM 20’s processes; any process can “invoke” any other process (except *sim* should only be invoked once).
- A CSIM 20 process is not a UNIX process. It is like a thread or lightweight process. A CSIM 20 process cannot return a value (it is not a function).
- A CSIM 20 process terminates by doing a normal procedure exit: it either executes a “return” statement or it flows out of the procedure.
- A CSIM 20 process can synchronize its interactions with other processes by using events, mailboxes, facilities and storage blocks.

The *CSIM 20 User’s Guide* goes into more detail on how to write processes.

Testing CSIM 20 Programs

Testing and debugging CSIM 20 programs can proceed in several ways:

- A CSIM 20 program is a real C or C++ program, so *printf* statements, etc., can be used to give the state of various variables and procedures.
- A CSIM 20 event trace gives a detailed account of the actions of each process in the model. It can be turned on in two different ways:
 - The `-T` command line argument, which causes the program to start with the trace on. To specify this on a UNIX command line, specify “`-T`”
 - The *trace-on* and *trace-off* statements in a CSIM program can selectively activate and deactivate the event trace.
- The *dump_status* statement prints the current status of all resources and processes in the model. There are separate status routines for facilities (*status_facilities*), storage blocks (*status_storages*), etc.

The CSIM 20 library routines all send their output to one of three files. The files all default to `stdout`, but can be redirected by using one of the following statements:

set_output_file – redirects reports and status information

set_trace_file – redirects the event trace

set_error_file – redirects error messages

Note that all CSIM documentation is available on the Mesquite website at <http://mesquite.com/documentation/index.htm>

For Further Information

If you have difficulties installing the CSIM 20 library or with writing, testing and executing CSIM 20 programs, please contact:

Customer Support

Mesquite Software, Inc.

PO Box 26306

Austin, TX 78755, USA

Tel: 1-800-538-9153 (for US customers), 1-512-338-9153 (for all others)

Fax: 1-512-338-4966

E-mail: info@mesquite.com

www.mesquite.com