

How to build Simbody 2.2 from source on Mac OS X and Linux platforms

Michael Sherman, May 2011

Simbody 2.2 has been re-engineered to be much easier to build from source than previous releases. One obvious change is that there is now a source distribution zip file posted on the Simbody downloads page. Also, the Simbody Visualizer no longer depends on VTK and other dependencies are minimal and easily handled. The basic procedure is now: install prerequisites, download the source, unzip, build, install.

Simbody is now distributed directly from the Simbody project page on simtk.org: <https://simtk.org/home/simbody>. (In earlier releases it was packaged with other software and distributed from the simtkcore project instead.)

This document covers the build-from-source procedure on Mac OSX and Linux platforms. The basic procedure is the same on Windows but the details are sufficiently different that there is a separate document for building on Windows.

Some preliminaries

The Simbody 2.2 downloads page already includes pre-built binaries for Mac and Linux. For many purposes you may be able to install one of those and skip the build-from-source step. However, if you want to build from source, the downloads page also includes a source distribution as a zip file. The release binaries were built directly from that source distribution, so you should be able to reproduce exactly the binaries we provide as well as build binaries for other compiler versions and operating systems. You can build 32-bit (i386 architecture) binaries for Mac, and either 32 or 64 bit binaries will be built for Linux systems, depending on whether you have a 32 or 64 bit Linux installation. (If you want to attempt to build a universal or 64-bit (x86-64 architecture) binary for Mac, please read the comments on page 13 first.)

We use CMake to produce the local platform-dependent build system. That allows us to maintain a single set of configuration files (CMake calls those CMakeLists.txt) and generate a variety of different ways to build Simbody. Here we're going to provide instructions for command-line build using make, which is common between Mac and Linux systems. So the first step in any build will be to run CMake once to configure the build environment. You do not have to be familiar with CMake; we'll provide step-by-step instructions below which are not complicated.

You can also use CMake to generate projects for various IDEs like XCode, Eclipse, and Netbeans but we won't cover that here. When you run CMake you can see what else it offers. We would be interested to

hear about success or problems with those – please post to the Simbody forum where you may be able to get help from other users who have successfully created different build environments.

Next we’re going to give step-by-step instructions for building from source under the following assumptions:

- You are running Mac OS X 10.5 (Leopard) or later, or a Linux system with command-line developer tools available, such as make and gcc. We will be working from a shell or terminal window. We have tested on Ubuntu and Centos but Simbody should work on other Linux platforms also.
- If you are running on a Linux machine you know how to use a package installer to get compatible software.
- You have sufficient administrator privileges to install packages and the Simbody libraries where you want them.
- You will start with the Simbody2.2 source distribution zip file downloaded from <https://simtk.org/home/simbody>, Downloads page. (Instructions for getting source from our Subversion repository are provided as an appendix.)

Installation

You can work with Simbody from its build environment if you want. But in most cases you’ll want to create a Simbody installation so that you can access it from other places. By default the installation goes to *installDir=/usr/local /SimTK*. You can install it elsewhere if you like. So that executables can find the dynamic libraries (shared objects), your library search path needs to include *installDir/lib*. On OS X the relevant path environment variable is DYLD_LIBRARY_PATH, on Linux systems it is LD_LIBRARY_PATH. Both debug and release libraries can coexist in the installation directory.

What you’re building

The Simbody build creates three libraries: libSimTKsimbody, libSimTKmath, and libSimTKcommon. Dynamic libraries are created with suffix “.dylib” on OSX and “.so” on Linux. Static libraries are also created (with “_static” appended to the names and a “.a” suffix). We do not recommend you use static libraries at all; they are present only because in some cases it may be easier to debug using them. Finally, you can also create all libraries debug form in which case the build system will append a final “_d” to the library name, just prior to the suffix.

There are also prerequisite dependencies: Lapack and Blas and a GLUT library and X support for the Visualizer. On OSX systems with developer tools installed, these libraries are already available. On most Linux systems they are available as prebuilt binaries that can be installed using the package installer to get Lapack (installs Blas as a dependency), Freeglut, Xi, Xmu. If you don’t build the Visualizer then the only dependencies are Lapack and Blas.

The libraries are strictly ordered by dependency:

SimTKsimbody→SimTKmath→SimTKcommon→Lapack→Blas and of course everything depends on various runtime libraries.

Getting help

If you have any trouble along the way, you can post to the “help” forum in the Simbody project (<http://simtk.org/home/simbody>, click on Advanced -> Public Forums).

Step 1: acquire needed goodies

Install build tools (Mac and Linux)

If you don’t have them already, find and install these tools. You aren’t going to have to learn to do much with them but our automated build system requires them.

- A developer environment including make, gcc, a text editor or IDE, and so on. Apple provides a free developer download for OSX that provides everything you need. Linux systems vary but all needed tools are available for free. We test our builds on Ubuntu and Centos.
- CMake 2.8.0 or later (cmake.org). On Linux systems you should be able to get this from the package installer. For Mac, go to <http://cmake.org/cmake/resources/software.html> and look for the OSX installer named something like cmake-2.8.4-Darwin-universal.dmg. You can also get a Linux binary there and can build from source if necessary, but you shouldn’t have to.
- Doxygen 1.7.2 or later (doxygen.org). On Linux systems you should be able to get this from the package installer. For Mac, go to <http://doxygen.org>. Click the Downloads link and look for the OSX installer named something like Doxygen-1.7.4.dmg. You can also get a Linux binary there and can build from source if necessary, but you shouldn’t have to.
- (Optional) A command line Subversion client so CMake can use it to extract version number information. This will already be installed with the Mac developer kit and you can get it from the Linux package installer if it isn’t there. You can check by typing “svn” in a terminal window.

The above tools should be installed so that they can run from a shell command line. You can test by making sure the commands “g++”, “make”, “ccmake”, and “doxygen” are all there.

Install prerequisite libraries (Linux only)

The Simbody libraries depend on Lapack and Blas for numerical linear algebra computations. Mac systems come with those; on Linux you will need to install them if you haven’t already. Generally you need only install a “lapack” package with a suitable package installer; “blas” will be installed automatically as a dependency since lapack depends on blas.

Building the Simbody Visualizer is optional, but you will probably want it and it is built by default. It depends on three libraries: freeglut, and the X libraries Xi and Xmu. OSX comes with everything preinstalled. But on a Linux platform, you will need to install them. Be sure to install the developer

version of these packages; for Ubuntu these are called “freeglut-dev” (or “freeglut3-dev”), “libxi-dev”, and “libxmu-dev”. You can install them (for example) like this:

```
sudo apt-get install freeglut-dev
sudo apt-get install libxi-dev
sudo apt-get install libxmu-dev
```

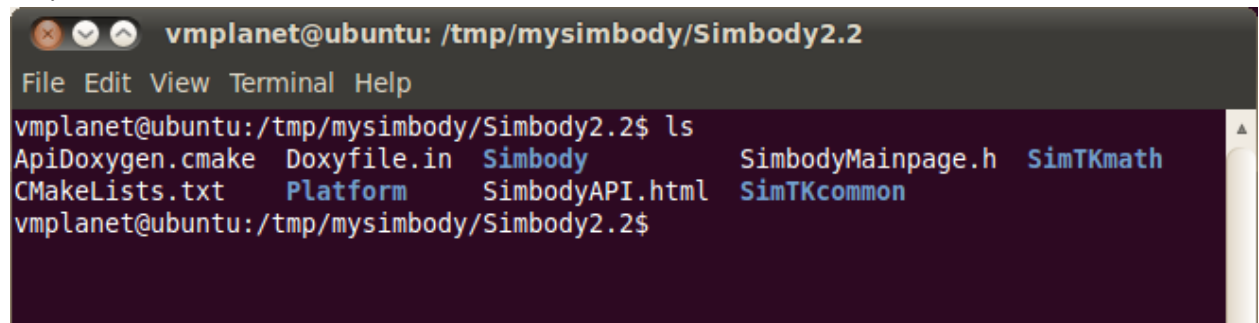
Depending on Linux version you may have a nice GUI software installer like the Ubuntu Software Center, or a command line installer like yum or apt.

Step 2: get the source code

1. Create a directory to contain the source and binaries while we’re building. This should *not* be in the installation directory. Here I’m going to call it “mysimbody”.
2. Go to <https://simtk.org/home/simbody>. Select the **Downloads** tab and look for the source distribution zip file for Simbody 2.2 (it’s named `Simbody2.2-Source.zip`). Download it and unzip it into mysimbody. It will unzip into a subdirectory `mysimbody/Simbody2.2`.

Warning: don’t use “src” in the name for your top-level source directory because our Doxygen configuration excludes path names matching “*/src/*” so that only include files are used to generate API docs; you’ll get blank Doxygen documentation if you use “src” in the path name.

You should now have a directory `.../mysimbody/Simbody2.2` that looks something like this, maybe with some extra files:

A terminal window with a dark background and light text. The title bar shows 'vmplanet@ubuntu: /tmp/mysimbody/Simbody2.2'. The menu bar includes 'File', 'Edit', 'View', 'Terminal', and 'Help'. The prompt is 'vmplanet@ubuntu:/tmp/mysimbody/Simbody2.2\$'. The command 'ls' has been executed, showing a directory listing: 'ApiDoxygen.cmake', 'Doxyfile.in', 'Simbody', 'SimbodyMainpage.h', 'SimTKmath', 'CMakeLists.txt', 'Platform', 'SimbodyAPI.html', and 'SimTKcommon'.

```
vmplanet@ubuntu:/tmp/mysimbody/Simbody2.2$ ls
ApiDoxygen.cmake  Doxyfile.in  Simbody      SimbodyMainpage.h  SimTKmath
CMakeLists.txt   Platform     SimbodyAPI.html  SimTKcommon
```

If instead you want to check out the source from Subversion rather than using the source distribution zip file, see the instructions on page 13. The source distribution zip file is a direct export of the Subversion branch `branches/Simbody2.2` in the Simbody source repository, so will provide the identical source code.

Step 3: run CMake

CMake is used to create a set of makefiles that can be used to compile and build the code. We are going to have CMake create two sets of these makefiles: one for building debug libraries, and one for building release (optimized) libraries. Both sets of libraries can coexist in the final installation directory, but not in the build directory.

1. **Create build directories.** Create two directories for the output of CMake and the binaries that will be produced when you build. One directory will hold the debug build system and one the release build system. You can put the generated binaries anywhere but you should not mix them in with the source code! For this example I put them in “build” (release) and “build_d” (debug) directories under mysimbody, next to the source tree but not inside it (i.e. mysimbody/build, mysimbody/build_d). So mysimbody looks like this on my machine:

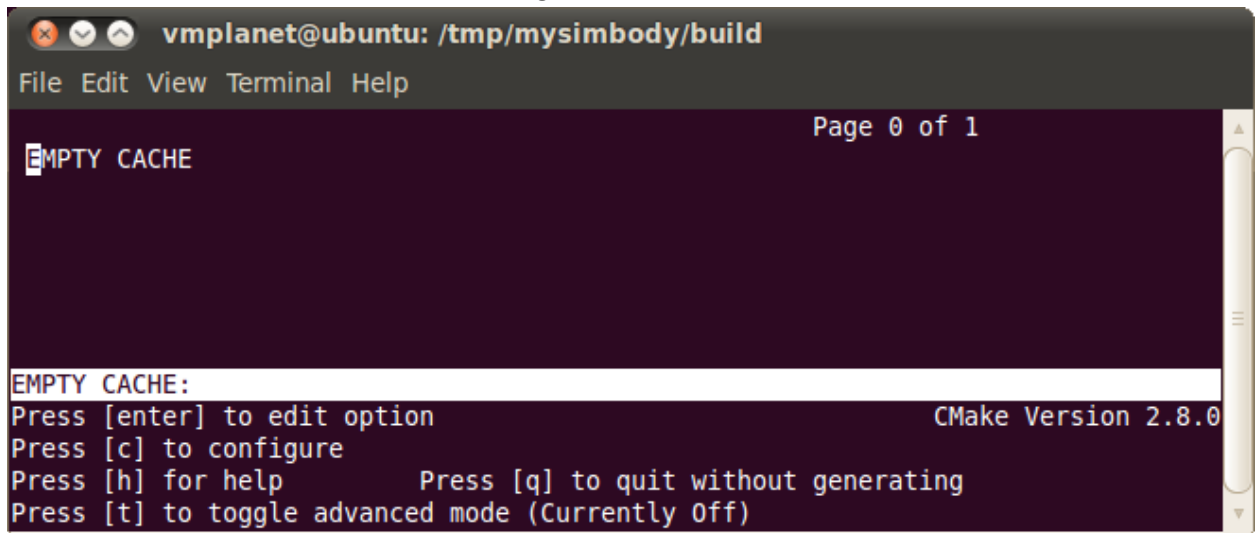
```
vmplanet@ubuntu:/tmp/mysimbody$ mkdir build_d
vmplanet@ubuntu:/tmp/mysimbody$ ls
build build_d Simbody2.2
vmplanet@ubuntu:/tmp/mysimbody$
```

2. **Create the release (optimized) build system.** cd to the build directory. Then run the “ccmake” CMake GUI from there, like this:

```
cd build
ccmake ../Simbody2.2
```

The ccmake program will look for ../Simbody2.2/CMakeLists.txt in this case as the top level of the build system. All ccmake output will go in build and its subdirectories. We are using the default “generator” for ccmake, which creates standard Unix Makefiles. Try “ccmake --help” to see what other generators are available, for example XCode, CodeBlocks, Eclipse CDT4, and KDevelop3. The instructions here assume the default Unix Makefiles are generated. We suggest you try this generator first to make sure you can build successfully before trying one of the other ones (for which we’re not providing instructions).

You should see a screen that looks something like this:



Look at the version number (lower right) and make sure you have CMake 2.8.0 or later. Press “c” in the lower window to configure. You should now see something like this (without the red squiggles, and maybe with the black/white color scheme reversed):

```
vmplanet@ubuntu: /tmp/mysimbody/build
File Edit View Terminal Help
Page 1 of 1
BUILD_BINARY_DIR      */tmp/mysimbody/build
BUILD_EXAMPLES        *ON
BUILD_PLATFORM        *Linux:x86
BUILD_TESTING         *ON
BUILD_TESTING_SHARED  *ON
BUILD_TESTING_STATIC  *ON
BUILD_UNVERSIONED_LIBRARIES *ON
BUILD_USING_NAMESPACE *
BUILD_USING_OTHER_LAPACK *
BUILD_VERSIONED_LIBRARIES *OFF
BUILD_VISUALIZER       *ON
CMAKE_BUILD_TYPE      *Release
DART_ROOT              *DART_ROOT-NOTFOUND
LAPACK_BEING_USED      *lapack;blas
SIMBODY_VERSION        *2.2
SVNVERSION              */usr/bin/svnversion
SVNVERSION_EXE         *SVNVERSION_EXE-NOTFOUND
SimTKCOMMON_LIBRARY_NAME *SimTKcommon
SimTKMATH_LIBRARY_NAME  *SimTKmath
SimTKSIMBODY_LIBRARY_NAME *SimTKsimbody
SimTK_INSTALL_PREFIX   */usr/local/SimTK

BUILD_BINARY_DIR: The Simbody build (not the install) puts all the libraries and
Press [enter] to edit option
Press [c] to configure
Press [h] for help
Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
CMake Version 2.8.0
```

CMake puts asterisks next to anything that changed during the configure operation. Since this is new, everything has an asterisk. To move around in the fields (shown in white above), click in the top window, then use up and down arrow in the white area. When you reach a field you want to change, hit ENTER. Boolean fields will switch between ON and OFF. Text fields can be typed in after you hit ENTER, then use arrow keys to move out of the field. When you're done, hit "c" (configure) again and repeat until there are no asterisks.

The field CMAKE_BUILD_TYPE is highlighted above with a red oval. This field can have one of two legitimate values "Release" and "Debug" (case sensitive; be careful). The default is Release which means fast Simbody libraries are created. If you want debug libraries, you must change this field to "Debug"; we're going to do that in the next section but here we want to build the optimized release libraries now so don't change that field.

Note that the last field SimTK_INSTALL_PREFIX circled above shows the default installation directory, /usr/local/SimTK. To use a different installation directory, set the value for the field SimTK_INSTALL_PREFIX to the desired directory (don't use CMAKE_INSTALL_PREFIX). You can put the installation anywhere you want. Note: the installation directory is not the same thing as the build directory. We will first build into the build directory (mysimbody/build or build_d here), then as a final step we'll do an install from the build directories into the

installation directory. The contents and organization of the installation directory are different than those of the build directory. For the example below, I set `SimTK_INSTALL_PREFIX` to `/tmp/mysimbody/install` in CMake. We're going to install both release and debug libraries into the same installation directory; they can coexist there but not in a build directory.

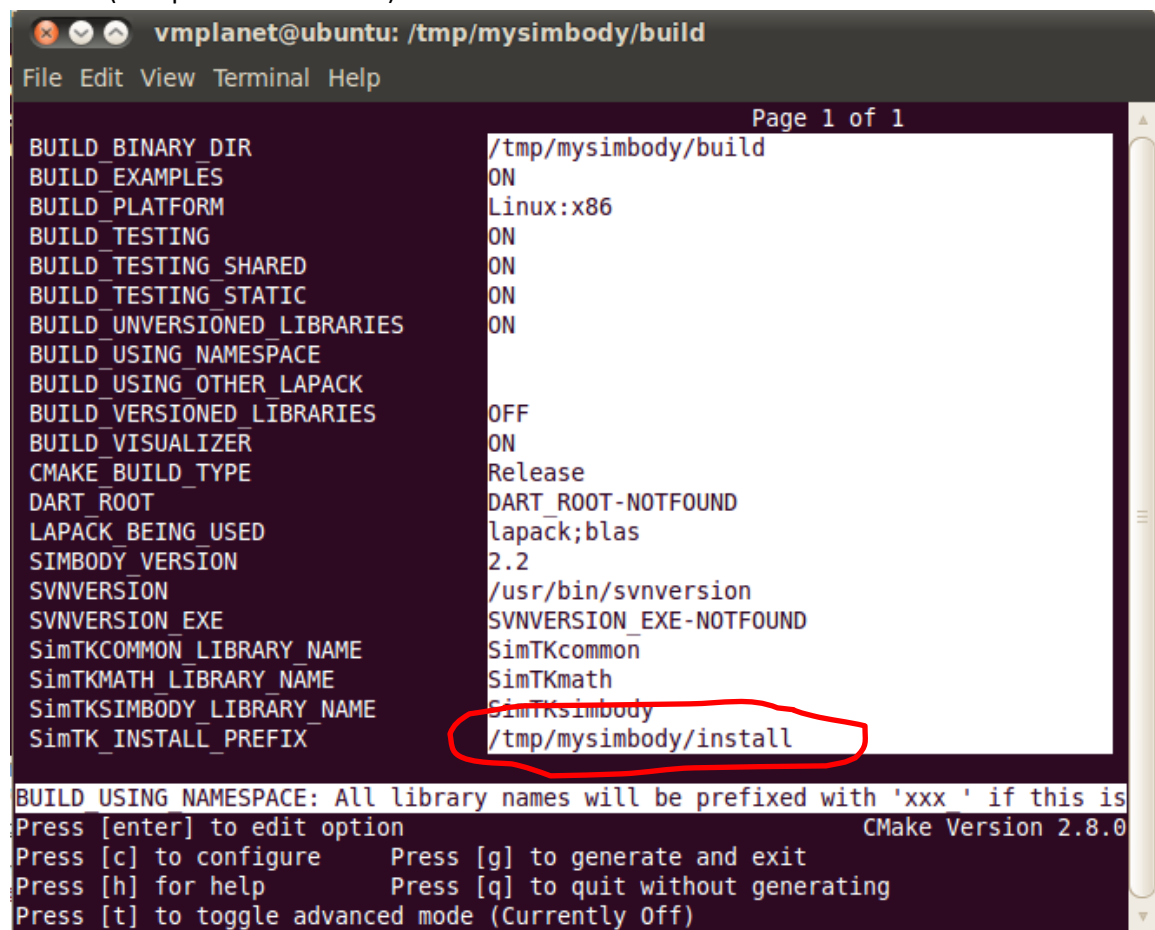
Mac only note: On the Mac there will be two additional fields shown in the CMake GUI that determine what OSX version we will target. With their default values shown, these are:

`CMAKE_OSX_DEPLOYMENT_TARGET=10.5`

`CMAKE_OSX_SYSROOT=/Developer/SDKs/MacOSX10.5.sdk`

Our default of 10.5 is the OSX "Leopard" release, meaning that the resulting Simbody libraries will work on Leopard, Snow Leopard (10.6) and later releases but *not* Tiger (10.4) or earlier. You can change that if you know what you're doing. There is also a `CMAKE_OSX_ARCHITECTURES` field that determines what goes into the universal binaries we produce. We create only 32 bit Intel-based binaries (that is, "i386" architecture); you shouldn't have to change that. (If you are running on a PowerPC-based Mac, you have our sympathy but not our support.)

After changing the install directory and hitting "c" to configure again, my CMake screen looked like this (except for the red oval):



```
vmplanet@ubuntu: /tmp/mysimbody/build
File Edit View Terminal Help
Page 1 of 1
BUILD_BINARY_DIR      /tmp/mysimbody/build
BUILD_EXAMPLES        ON
BUILD_PLATFORM        Linux:x86
BUILD_TESTING          ON
BUILD_TESTING_SHARED  ON
BUILD_TESTING_STATIC  ON
BUILD_UNVERSIONED_LIBRARIES ON
BUILD_USING_NAMESPACE OFF
BUILD_USING_OTHER_LAPACK ON
BUILD_VERSIONED_LIBRARIES OFF
BUILD_VISUALIZER       ON
CMAKE_BUILD_TYPE       Release
DART_ROOT              DART_ROOT-NOTFOUND
LAPACK_BEING_USED       lapack;blas
SIMBODY_VERSION        2.2
SVNVERSION             /usr/bin/svnversion
SVNVERSION_EXE         SVNVERSION_EXE-NOTFOUND
SimTKCOMMON_LIBRARY_NAME SimTKcommon
SimTKMATH_LIBRARY_NAME  SimTKmath
SimTKSIMBODY_LIBRARY_NAME SimTKsimbody
SimTK_INSTALL_PREFIX   /tmp/mysimbody/install

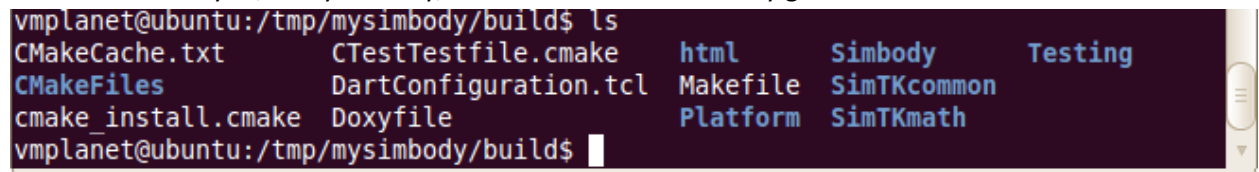
BUILD USING NAMESPACE: All library names will be prefixed with 'xxx ' if this is
Press [enter] to edit option
Press [c] to configure      Press [g] to generate and exit
Press [h] for help          Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
CMake Version 2.8.0
```


There are no more asterisks, which means we are ready to generate.

Note: by default, we are building all the regression test cases, and each test will be linked twice, once with the shared (dynamic) versions of the Simbody libraries, and once with the static versions. Static tests take a long time to link and use a lot of disk space. You can save some time and space by unchecking the BUILD_TESTING_STATIC option; normally only the shared library form of Simbody will be used anyway. However, at least the first time you build these libraries it can be useful to have all the various test cases around, so we suggest you build everything at least this once.

[There are a few other build options that might prove useful. In particular you can generate the libraries with a specified prefix, and you can have the names include the version number (2.2). The instructions here assume you haven't changed those. You can also use a different Lapack library.]

Now hit “g” for generate. If generation is successful, cmake will simply exit after it completes. Otherwise it will display error messages. That last step should have created the build environment for you, in mysimbody/build. An ls in that directory gives:



```
vmplanet@ubuntu:/tmp/mysimbody/build$ ls
CMakeCache.txt      CTestTestfile.cmake  html      Simbody      Testing
CMakeFiles          DartConfiguration.tcl Makefile   SimTKcommon
cmake_install.cmake Doxyfile             Platform  SimTKmath
```

Next we're going to repeat the above procedure to generate a build system that can produce debug libraries. The instructions here are abbreviated; see above for details.

3. **Create the debug (slow) build system.** cd to the `build_d` directory. Then run the “cmake” CMake GUI from there, as before:

```
cd build_d
cmake ../Simbody2.2
```

All cmake output will now go in `build_d` and its subdirectories. Hit “c” for configure. Change the CMAKE_BUILD_TYPE to “Debug” and the SimTK_INSTALL_PREFIX to your installation directory; I'm using `/tmp/mysimbody/install`, same as for the release libraries. After making those changes and hitting “c” again to configure, my CMake screen looked like this, with the changes circled:


```
vmplanet@ubuntu: /tmp/mysimbody/build_d
File Edit View Terminal Help

Page 1 of 1

BUILD_BINARY_DIR      /tmp/mysimbody/build_d
BUILD_EXAMPLES        ON
BUILD_PLATFORM        Linux:x86
BUILD_TESTING         ON
BUILD_TESTING_SHARED  ON
BUILD_TESTING_STATIC  ON
BUILD_UNVERSIONED_LIBRARIES ON
BUILD_USING_NAMESPACE
BUILD_USING_OTHER_LAPACK
BUILD_VERSIONED_LIBRARIES OFF
BUILD_VISUALIZER      ON
CMAKE_BUILD_TYPE      Debug
DART_ROOT             DART_ROOT-NOTFOUND
LAPACK_BEING_USED      lapack;blas
SIMBODY_VERSION        2.2
SVNVERSION            /usr/bin/svnversion
SVNVERSION_EXE        SVNVERSION_EXE-NOTFOUND
SimTKCOMMON_LIBRARY_NAME SimTKcommon
SimTKMATH_LIBRARY_NAME SimTKmath
SimTKSIMBODY_LIBRARY_NAME SimTKsimbody
SimTK_INSTALL_PREFIX  /tmp/mysimbody/install

BUILD_BINARY_DIR: The Simbody build (not the install) puts all the libraries and
Press [enter] to edit option                                CMake Version 2.8.0
Press [c] to configure      Press [g] to generate and exit
Press [h] for help          Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
```

Also note above that the BUILD_BINARY_DIR is now “build_d” rather than “build”. There are no more asterisks, which means we are ready to generate.

Now hit “g” for generate. If generation is successful, cmake will simply exit after it completes. Otherwise it will display error messages. That last step should have created the debug build environment for you, in mysimbody/build_d. An `ls` in that directory gives:

```
vmplanet@ubuntu:/tmp/mysimbody/build_d$ ls
CMakeCache.txt      CTestTestfile.cmake  html      Simbody      Testing
CMakeFiles          DartConfiguration.tcl Makefile   SimTKcommon
cmake_install.cmake Doxyfile             Platform  SimTKmath
vmplanet@ubuntu:/tmp/mysimbody/build_d$
```

That’s it for CMake. We won’t be needing it any more since we have used it to create the two build systems we’re going to need, for building debug and release versions of Simbody.

Step 4: build, test, and install Simbody

Above we used CMake to create two different build systems for Simbody: a release (optimized) build and a debug build. We put the release build system in directory `mysimbody/build` and the debug one in `mysimbody/build_d`. Now we'll build them both and then combine them in a single install directory, which we specified in CMake as `mysimbody/install` (the default is `/usr/local/SimTK`). One caveat: if you build both sets of libraries, be sure to do the debug install first, *then* the release install (see warning below for an explanation).

Note that while the debug version can be very helpful for you when you are debugging, it is *much* slower (more than 10X) than the release version, so you will definitely need to build the release version to do anything substantial.

Here is the procedure to build, test, and install Simbody. Timings are given for some steps but they will vary substantially depending on the compiler and machine – YMMV.

1. Build the debug libraries:

```
cd mysimbody/build_d
make
```

Tip: If you have multiple CPUs available, you can speed up the build substantially by telling make how many to use. For example, “`make -j2`” will run two parallel threads and run faster.

On one CPU this build, including all static and dynamically-linked test cases and examples, should take 15-30 minutes. There should be no errors and no warnings – please post to the Simbody help forum if you encounter any trouble.

2. Test debug libraries:

```
make test
```

This will run all the regression tests (twice since we built static and dynamic ones, and slowly since these are debug libraries). This took about six minutes on a MacBook Pro using one 2.4GHz Intel CPU.

These *should* all run successfully, but depending on the gcc version in use there may be a few failures. On Mac you may see a failure in `FactorLUtest`; that is a known problem with the test and harmless. On Ubuntu 10 with gcc 4.4.3 `TestNoseHooverThermostat` also fails harmlessly. If you see other failures, please post to the Simbody forum to see if they are unexpected; say what platform you are on and what version of gcc you used (“`gcc --version`” will tell you).

The regression tests are all numerical. You should make sure the Visualizer built properly by running one of the examples by hand (remember this is the debug version so don't expect good performance!). From within the `build_d` directory, type this to the shell prompt:

```
./ExamplePendulum
```

You should see a one-body pendulum hanging down from the ground plane and swinging; if so, the Visualizer built properly. Wait until you have built release libraries below to try more

examples. (Note: don't expect the Visualizer to perform well if you are running in a virtual machine – performance depends strongly on the quality of the OpenGL implementation and we have seen some very sorry VM OpenGLs. Turning off the ground & sky might help a little.)

3. **Create the API documentation:**

```
make doxygen
```

This will run Doxygen, which will read through all the code and extract API reference documentation from the comments to generate html files which together constitute our reference documentation for all Simbody classes. This will end up in the doc subdirectory of the installation directory. This will take less than a minute.

This step need be done only once; we aren't going to repeat it when we make the release libraries since the documentation is the same.

4. **Install debug libraries:**

```
sudo make install
```

This will ask for your password and then install the debug libraries and doxygen documentation into the install directory that you specified in CMake. You don't actually need the "sudo" unless you are installing into some protected place like /usr/local. In the example here we used /tmp/mysimbody/install which does not require root privileges to write on.

5. **Now build the release libraries:**

```
cd ../build          (that is, mysimbody/build rather than build_d)
make -j2             (assuming two CPUs)
```

Using two CPUs on a MacBook Pro with a 2.4GHz Intel core duo, building all the static and dynamic tests and universal binaries with two architectures, this took about 50 minutes . Time on a Linux platform will probably be closer to 30 minutes. There should be no errors and no warnings – please post to the Simbody help forum if you encounter any trouble.

6. **Test release libraries:**

```
make test
```

This will run all the regression tests (twice since we built static and dynamic ones). This took about one minute on a MacBook Pro using one 2.4GHz Intel CPU.

These *should* all run successfully, but depending on the gcc version in use there may be a few failures. On Mac you will see a failure in `FactorLUtest`; that is a known problem with the test and harmless. On Ubuntu 10 with gcc 4.4.3 we have seen `TestTriangleMesh` fail. If you see other failures, please post to the Simbody forum to see if they are unexpected; say what platform you are on and what version of gcc you used ("gcc --version" will tell you).

7. (No need to build doxygen documentation again)

8. Install release libraries:

```
sudo make install
```

This will ask for your password and then install the release libraries into the install directory that you specified in CMake. (You don't actually need the "sudo" unless you are installing into some protected place like /usr/local. In the example here we used /tmp/mysimbody/install which does not require root privileges to write on.) Note that we are *adding* the release libraries to the same installation directory in which we put the debug libraries.

WARNING: if you build both the debug and release libraries, be sure to install the release ones last. That's because both builds will create a `VisualizerGUI` program with the same name that gets installed in the bin directory. You want the release one to overwrite the debug one, not vice versa! This is a known bug in the build process – we should have created a `VisualizerGUI_d` and only used it if you want to debug the Visualizer itself.

Step 5: Play with Simbody

While you are in the build directory, try `ls Example*` to see some of the example programs you just built:

```
vmplanet@ubuntu:/tmp/xxx$ cd ../mysimbody/build
vmplanet@ubuntu:/tmp/mysimbody/build$ ls Example*
ExampleAmySIKProblem      ExampleContactPlayground  ExampleKneeJoint
ExampleAssemblerPlayground ExampleEventHandler        ExamplePendulum
ExampleBricardMechanism  ExampleEventReporter
ExampleChain              ExampleGears
vmplanet@ubuntu:/tmp/mysimbody/build$
```

These use the Simbody Visualizer (new in Simbody 2.2) to do some animation. At this point you should be able to run those and see the animation.

1. Experiment with Examples in the Simbody build environment.
 - Make sure you are running a release build, not debug – everything will run *very* slowly in debug.
 - Pick an example, say `ExampleChain`, and type `./ExampleChain` to start the program. You will get a console window and a Visualizer display window. Some require input at the console before they start the simulation; others take input in the Visualizer.
 - Repeat for other examples above.
 - Also try `ChainExample` and `JaredsDude` for fancier Visualizer examples.
2. From outside of the Simbody build environment, you can create projects that link with the installed libraries. A good place to start is to copy the examples from the Simbody installation subdirectory "examples" (don't build them in place). One option is to build them with the supplied Makefile – type just `make` to build `ExamplePendulum`, `make all` to get the rest. The Makefile looks for an environment variable `SimTK_INSTALL_DIR` that you should set to your installation direction (e.g. `/tmp/mysimbody/install` for the above example.) You will

also need to set your library search path

(LD_LIBRARY_PATH=\$SimTK_INSTALL_DIR/lib on Linux, DYLD_LIBRARY_PATH on Mac.)

3. Try working through the first tutorial from the Documents page of the Simbody project (<https://simtk.org/home/simbody>).

For more information

If you have any trouble along the way, you can post to the “help” forum in the Simbody project (<http://simtk.org/home/simbody>, click on Advanced -> Public Forums). There is also a Wiki although there isn’t much there yet as of this writing (please add to it!). The Documents page has tutorials and API reference documentation. There are lots of examples, and the source code (especially the header files) has lots of comments, some possibly helpful.

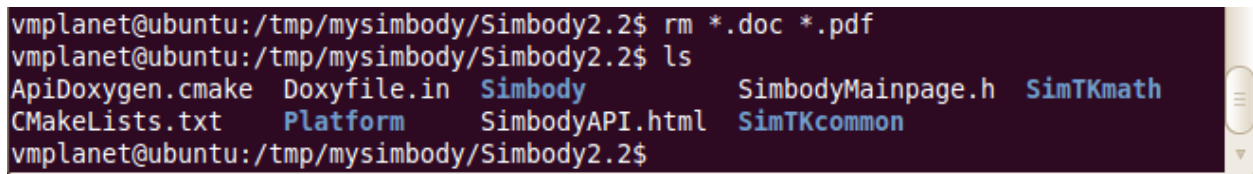
Appendix 1: Checking out source from Subversion

In case you want to access Simbody source from our Subversion repository directly, rather than from our source distribution zip file, follow the instructions below. Note that our source distribution is a direct export of the branches/Simbody2.2 branch of our Subversion repository. No permissions are required to get read-only access to the Subversion source. If you have a fix or feature you’d like to check in, please post to the Simbody developer’s forum to arrange something.

Use this command to check out the Simbody 2.2 SVN branch into directory `mysimbody/Simbody2.2`:

```
svn checkout https://simtk.org/svn/simbody/branches/Simbody2.2
               mysimbody/Simbody2.2
```

Now your `mysimbody/Simbody2.2` directory should look something like this, maybe with some extra files:

A terminal window screenshot showing the contents of the directory `mysimbody/Simbody2.2`. The prompt is `vmplanet@ubuntu:/tmp/mysimbody/Simbody2.2$`. The first command is `rm *.doc *.pdf`. The second command is `ls`, which lists the following files: `ApiDoxxygen.cmake`, `Doxyfile.in`, `Simbody`, `SimbodyMainpage.h`, `SimTKmath`, `CMakeLists.txt`, `Platform`, `SimbodyAPI.html`, and `SimTKcommon`. The prompt returns to `vmplanet@ubuntu:/tmp/mysimbody/Simbody2.2$`.

Now you have the source you can follow the rest of the build instructions starting with Step 3 (“run CMake”) on page 4.

Appendix 2: Universal binaries (or x86_64 architecture) for Mac

We have been unable to build a Simbody universal binary using the built-in Lapack and Blas on Mac OSX systems (part of the “Accelerate” framework), due to runtime errors that *appear* to be something wrong with the 64-bit builds of those libraries, but could be something else. We worked on this for a long time and got nowhere so decided to release Simbody 2.2 with just 32 bit (i386 architecture) binaries, and the

top-level CMakeLists.txt file has the 64 bit architecture commented out. If you use a different Lapack, you should be able to add the x86_64 architecture. Also, if you want to try using the built-in Mac Lapack and can figure out what we're doing wrong, please do! In any case if you try this please post to the Simbody forum and let us know what happens – we would like to be able to ship pre-build universal binaries with the next release.