

How to build Simbody 3.0 from source on Windows

Michael Sherman, 13 Oct 2012

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

Some preliminaries

As of this writing there is not yet a Simbody 3.0 binary release posted, so you have to build from source if you want to use Simbody 3.0. The source can be obtained from our Subversion repository per the instructions below. These instructions cover building a 32-bit (x86) Windows version of Simbody, that can be used on 32 or 64 bit Windows machines. If you want to attempt to build a 64-bit (x64) version of Simbody, you will have to build from source and resolve a few other issues discussed on page 12.

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'll assume you want to build from within Visual Studio, in which case CMake will produce Visual Studio solution files for an assortment of different Visual Studio versions. So the first step in any build will be to run CMake once to configure the build environment. You will not have to understand CMake; we'll provide step-by-step instructions below which are not complicated.

You can also use CMake to create other build systems on Windows: nmake, mingw, msys. When you run CMake you can see what else it offers. We haven't tried those options and would be interested to hear about success or problems with those – please post to the Simbody forum.

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

- You are running Windows XP, Vista, or 7 and you have enough disk space.
- You will build from within Visual Studio. The screen images are from VS10 but VS9 (2008) builds are nearly identical.
- You are building a 32-bit Simbody library. This will work fine on either 32 or 64 bit Windows systems; it just means that you must call the Simbody API from a 32-bit main program.

Installation

You can work with Simbody from within its Visual Studio 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 projects. By

default the installation goes to *installDir*=%ProgramFiles%\SimTK. On English systems %ProgramFiles% is typically “C:\Program Files” or “C:\Program Files (x86)”. You can install it elsewhere if you like. So that executables can find the DLLs, your path needs to include *installDir*\bin. (Earlier version of Simbody required *installDir*\lib as well; if you still have that in your path take it out now.)

What you’re building

The Simbody build creates three libraries: SimTKsimbody.lib, SimTKmath.lib, and SimTKcommon.lib and the corresponding DLLs. Static libraries are also created (with “_static” appended to the names). We do not recommend you use static libraries at all; they are present only because in some cases it is easier to debug using them. Finally, all libraries will be created in debug form also, by appending a final “_d” to the library name.

There are also two prerequisite dependencies: lapack and pthreads. We provide pre-built 32 bit (x86) binaries for these that are automatically installed. They are SimTKlapack.lib and pthreadVC2.lib. There are no debug or static forms of these libraries.

The libraries are strictly ordered by dependency:

SimTKsimbody→SimTKmath→SimTKcommon→(lapack,pthreads) and of course everything depends on the Visual Studio and Microsoft runtimes.

If you build the Simbody Visualizer which you probably will, you will get a VisualizerGUI.exe installed along with the above DLLs. The Visualizer is dependent on GLUT, and we provide the necessary 32 bit glut32.lib and glut32.dll for Windows.

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 Public Forums).

Step 1: acquire needed goodies

If you don’t have them already, find and install these tools:

- Visual Studio C++ 9 (2008), 10 (2010), or a corresponding Visual Studio Express edition. You can probably build using VS8 (2005) also; we haven’t tried that recently.
- CMake 2.8.3 or later (cmake.org). Go to <http://cmake.org/cmake/resources/software.html> and look for the Win32 installer named something like cmake-2.8.4-win32-x86.exe.
- Doxygen 1.7.2 or later (doxygen.org). Click the Downloads link and look for the Win32 installer named something like doxygen-1.7.4-setup.exe.
- A command line Subversion client so CMake can use it to extract version number information; we have used the CollabNet command line client that you can get here: <http://www.open.collab.net/downloads/subversion/>; be sure to download “CollabNet Subversion Command-Line Client (for Windows)” not one of the other options there. You can build without this but CMake will (harmlessly) complain SVNVERSION_NOTFOUND.

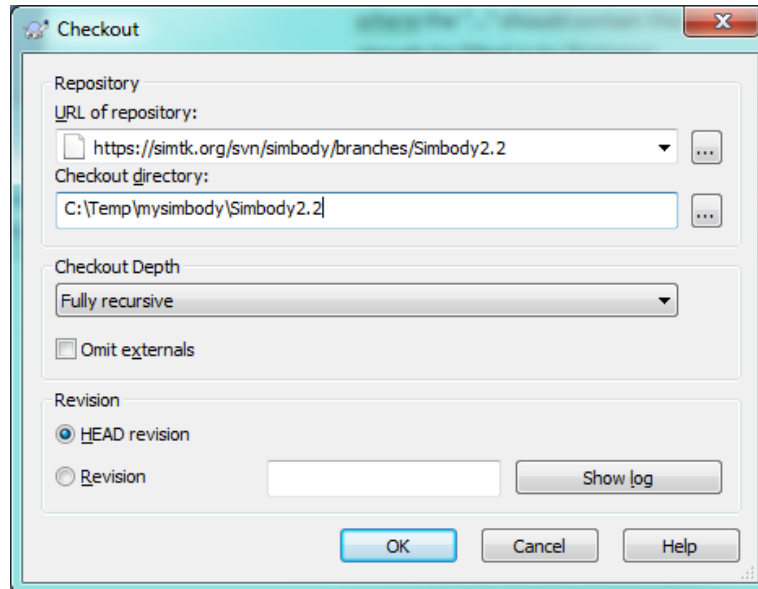
- For convenient access to the source from our Subversion repository you will probably want *also* to get the Subversion client Tortoise SVN (tortoisesvn.tigris.org) which is a very nice GUI-based client that integrates itself into Windows Explorer. (CMake still wants the command line client.)

Step 2: Checking out source from Subversion

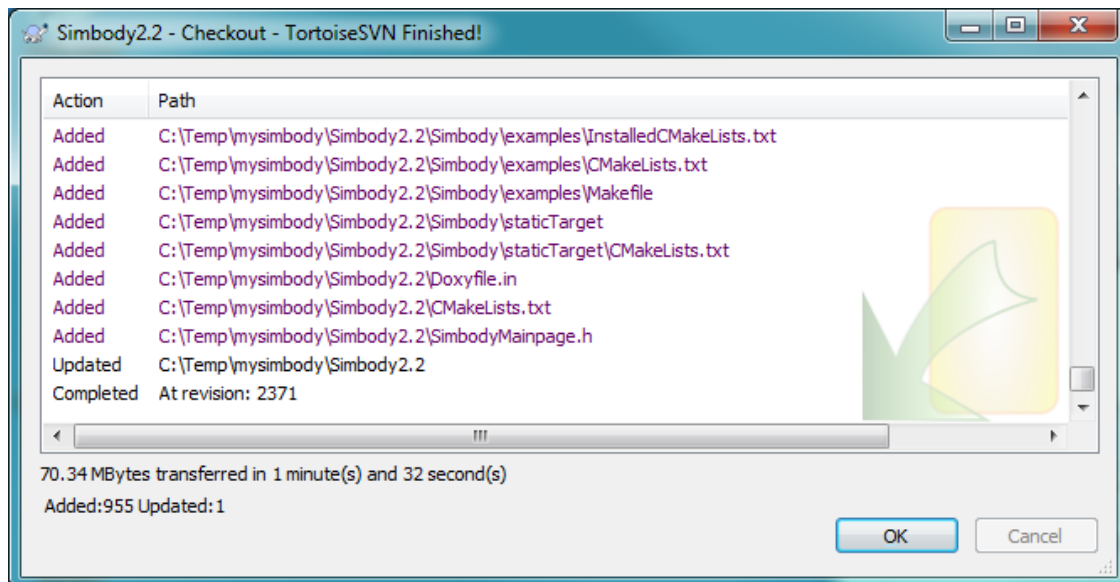
No permissions are required to get read-only access to the Subversion source. If you have a fix or feature you'd like to check in, please post to the Simbody developer's forum to arrange something.

1. Go get the Tortoise SVN Subversion client if you don't have it already (<http://tortoisesvn.tigris.org>) . (You can use a command line SVN client if you want but the instructions here assume you have installed Tortoise SVN.)
2. With Windows Explorer, go to the (empty) source folder (we're assuming it is named "mysimbody" here as above). Right-click and select "SVN Checkout...".
3. In the Tortoise SVN dialog window that appears, set the following values for the text fields:
URL of repository: `https://simtk.org/svn/simbody/branches/Simbody3.0`
Checkout directory: `...\mysimbody\Simbody3.0`
where the "..." should contain the location at which you created mysimbody (should already be filled in by Tortoise)

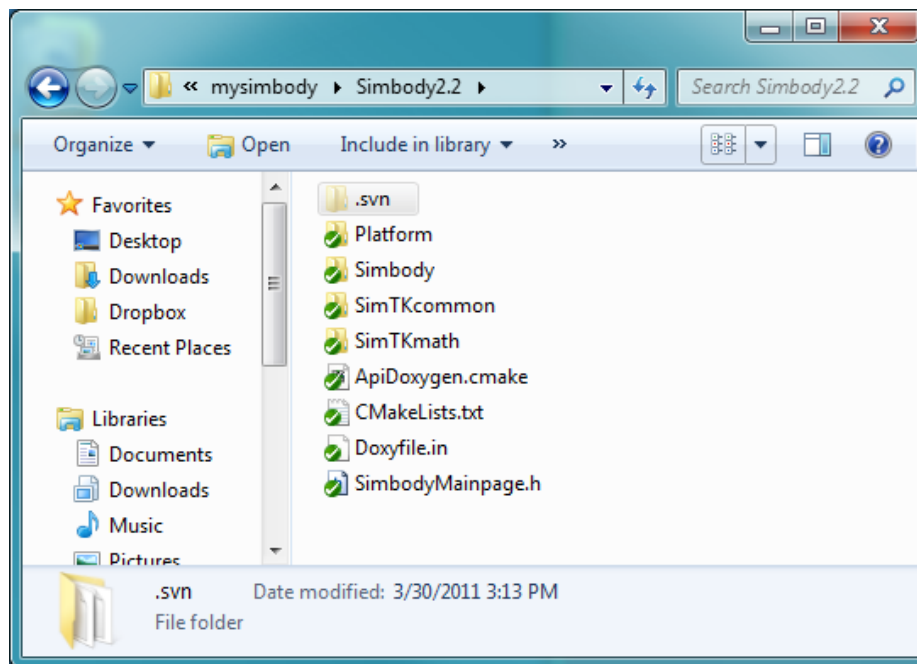
You will see something like this (except 3.0):



Click OK. Now you should see the files being checked out, with the end looking something like this (though 3.0):



Now your mysimbody/Simbody3.0 directory should look something like this:



(Tortoise SVN puts the green checks there, meaning that the local source is unchanged from the repository.)

Now you have the source you can follow the rest of the build instructions starting with Step 3 ("run CMake") below (except using "Simbody3.0" in place of "Simbody2.2-Source" as the source directory name).

Step 3: run CMake

CMake is used to create a Visual Studio “solution” (.sln) file that can be used to compile and build the code.

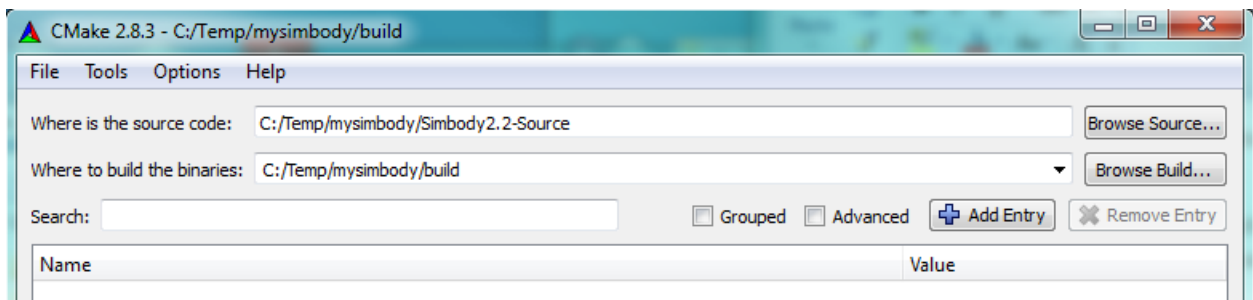
1. Run the CMake GUI. You may have a shortcut on your Desktop, or you can run it from the Start button. Look in the upper-left corner and make sure you have CMake 2.8.1 or later.
2. Set the following fields:

Where is the source code: .../mysimbody/Simbody3.0

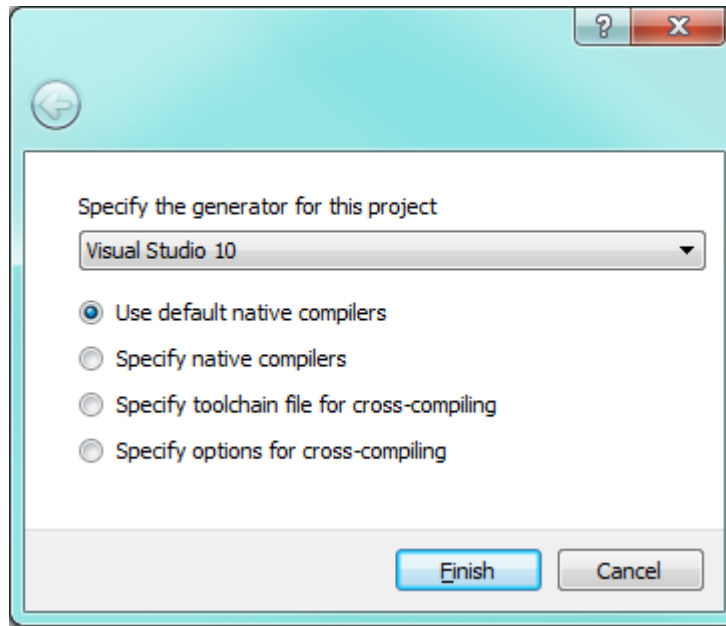
Where to build the binaries: .../mysimbody/build

The “...” should be the location at which you created mysimbody, for me it was C:/Temp. Note that 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 a “build” directory under mysimbody, next to the source tree but not inside it.

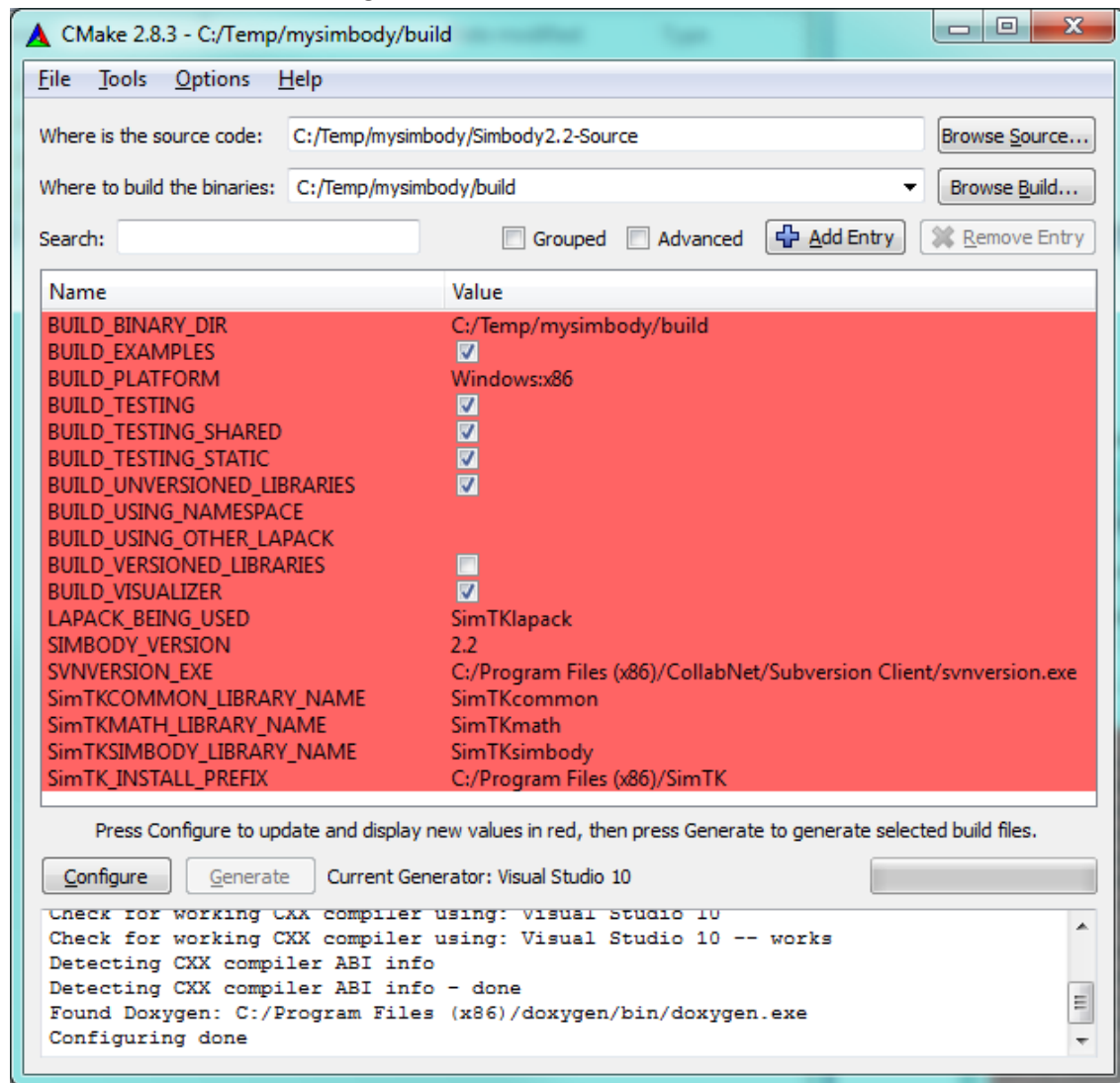
The top of the CMake GUI screen should look something like this:



3. Press the Configure button (near the bottom). Let CMake create the build directory. Choose a “generator” that best matches the compiler you are using, for example “Visual Studio 10 2010.” The form below will be displayed. Select “Use default native compilers” and click Finish. Don’t choose a 64 bit build unless you have installed a 64 bit Lapack and read the discussion on page 12.



You should now see something like this:



CMake highlights in red anything that has changed; since this is new, everything is red.

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`). Note that the default shown above is `%ProgramFiles%/SimTK` where `ProgramFiles` is a special Windows environment variable; for a 32 bit build on 64 bit Windows that is "C:/Program Files (x86)". On a 32 bit Windows the default is just "C:/Program Files". 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 here), then as a final step we'll do an install from the build directory into the installation direction. The contents and organization of the installation directory are different than those of the build directory. (For the example below, I created a directory `mysimbody/install` and set `SimTK_INSTALL_PREFIX` to `C:/Temp/mysimbody/install` in CMake, although that's not shown in the image above.)

Note: by default, we are building all the regression test cases, and each test will be linked twice, once with the shared 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.

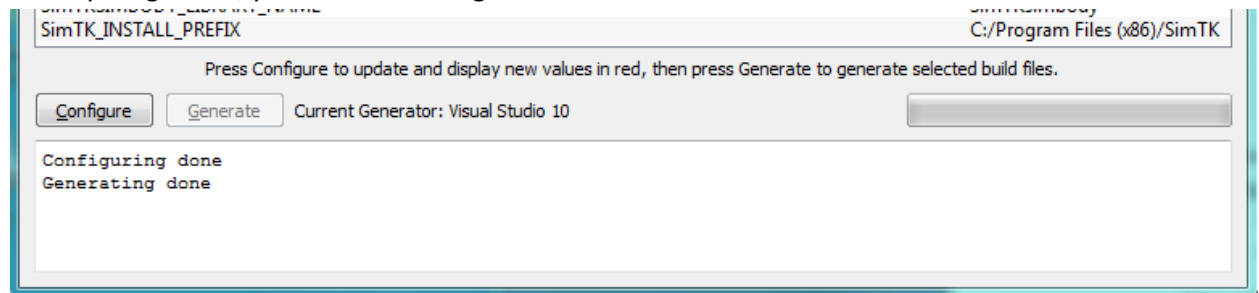
[There are a few other build options that might prove useful. In particular you can generate the libraries with a specified prefix, and you can have the names include the version. The instructions here assume you haven't changed those. You can also use a different Lapack library (see 64 bit instructions on page 12).]

Hit Configure again. This time there should be nothing in red and the Generate button will be available. Now hit the Generate button. The bottom panel should say "Generating done" and not have error messages. If you don't have a command-line Subversion client, you'll see a warning like:

```
Could not find 'svnversion' executable; 'about' will be
wrong. (Cygwin provides one on Windows.)
```

This will not prevent you from building but see Step 1 above for how to get a suitable Subversion client.

If everything works, you will see messages like these below:



That last step should have created the Visual Studio solution file for you. It will be `mysimbody/build/Simbody.sln`.

That's it for CMake. Exit the CMake GUI; we won't be needing it any more for the rest of the build.

Step 4: build, test, and install Simbody

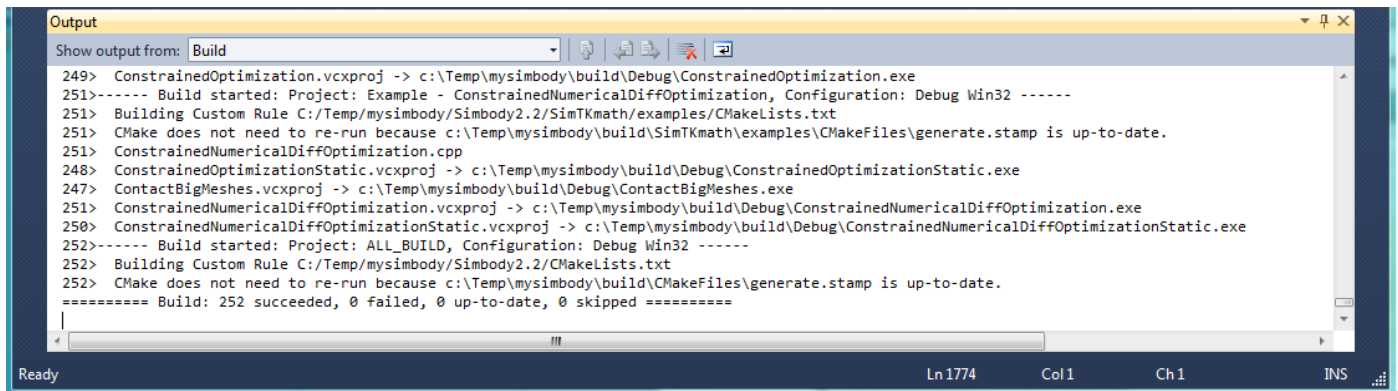
Simbody can be built using a number of different "configurations": Debug, RelWithDebInfo (Release with Debug Information), and Release. The default configuration is Debug. This configuration is very useful for testing and debugging, but too slow to use for most purposes. The production configuration

we recommend is RelWithDebInfo. That runs just as fast as Release, but keeps extra debugging information around that can be helpful if you encounter a problem. This makes the libraries bigger, but not slower. If you want the smallest possible libraries, use the Release configuration. One set of Debug libraries and one set of Release (or RelWithDebInfo) libraries can coexist in the installation directory; all Debug libraries are suffixed with “_d”. In all cases, both shared (DLL) and static libraries are built; static libraries have “_static” in their names. In the instructions below, we will first build Simbody in Debug mode and then build it in RelWithDebInfo mode. If you are going to build both sets be sure to do the release build last (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 RelWithDebInfo or 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. **Start up Visual Studio:** Use Windows Explorer to go to mysimbody/build. Click or double-click on the file Simbody.sln there. This brings up Visual Studio. Set configuration to Debug if it isn’t already.
2. **Build debug version:** Look for the ALL_BUILD target in the Solution Explorer panel. Right click on ALL_BUILD, and select “Build” from the drop down. Since we’re using the Debug configuration, this will build SimTKname_d and SimTKname_static_d libraries, where name={common,math,simbody}. The Simbody Visualizer and a complete set of dynamic and static test cases and example programs is also built unless you unchecked those options in the CMake GUI. The end of the “Output” window should look something like this:



```

Output
Show output from: Build
249> ConstrainedOptimization.vcxproj -> c:\Temp\mysimbody\build\Debug\ConstrainedOptimization.exe
251>----- Build started: Project: Example - ConstrainedNumericalDiffOptimization, Configuration: Debug Win32 -----
251> Building Custom Rule C:/Temp/mysimbody/Simbody2.2/SimTKmath/examples/CMakeLists.txt
251> CMake does not need to re-run because c:\Temp\mysimbody\build\SimTKmath\examples\CMakeFiles\generate.stamp is up-to-date.
251> ConstrainedNumericalDiffOptimization.cpp
248> ConstrainedOptimizationStatic.vcxproj -> c:\Temp\mysimbody\build\Debug\ConstrainedOptimizationStatic.exe
247> ContactBigMeshes.vcxproj -> c:\Temp\mysimbody\build\Debug\ContactBigMeshes.exe
251> ConstrainedNumericalDiffOptimization.vcxproj -> c:\Temp\mysimbody\build\Debug\ConstrainedNumericalDiffOptimization.exe
250> ConstrainedNumericalDiffOptimizationStatic.vcxproj -> c:\Temp\mysimbody\build\Debug\ConstrainedNumericalDiffOptimizationStatic.exe
252>----- Build started: Project: ALL_BUILD, Configuration: Debug Win32 -----
252> Building Custom Rule C:/Temp/mysimbody/Simbody2.2/CMakeLists.txt
252> CMake does not need to re-run because c:\Temp\mysimbody\build\CMakeFiles\generate.stamp is up-to-date.
===== Build: 252 succeeded, 0 failed, 0 up-to-date, 0 skipped =====

```

On my Windows i7 laptop with Visual Studio 10 the complete Debug build of Simbody and all the static and dynamic tests and examples takes about 15 minutes.

3. **Run regression tests:** Find the RUN_TESTS target in the Solution Explorer panel. Right click on RUN_TESTS and select “Build”. This will run all the tests (slowly since we’re in Debug). Every

test will run twice, once for the dynamically linked version and once for the statically linked version. The end of the “Output” window should look like this:

```

1> Start 155: TestReverseMobilizers
1> 155/158 Test #155: TestReverseMobilizers ..... Passed    0.14 sec
1> Start 156: TestReverseMobilizersStatic
1> 156/158 Test #156: TestReverseMobilizersStatic ..... Passed    0.41 sec
1> Start 157: TestTriangleMesh
1> 157/158 Test #157: TestTriangleMesh ..... Passed    2.92 sec
1> Start 158: TestTriangleMeshStatic
1> 158/158 Test #158: TestTriangleMeshStatic ..... Passed    3.09 sec
1>
1> 100% tests passed, 0 tests failed out of 158
1>
1> Total Test time (real) = 576.44 sec
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====

```

Build succeeded

The complete set of 158 dynamic and static regression tests using debug libraries ran in about 10 minutes on my i7 machine (closer to 1 minute for release libraries).

4. **Build the Doxygen documentation:** Find the `doxygen` target in the Solution Explorer panel. Right click on it and select “Build”. (That’s the plain “doxygen” target, not one of the subsidiary ones like `doxygen_Simbody`; ignore those.) The 3.0 doxygen documentation is also available pre-built here: https://simtk.org/api_docs/simbody/api_docs30/Simbody/html/index.html.
5. **Install debug version:** Find the `INSTALL` target. Right click on it to install the debug libraries and doxygen docs. Here is a screenshot of the Output window at the end of install:

```

1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/ExampleEventHandler.cpp
1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/ExampleEventReporter.cpp
1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/ExampleGears.cpp
1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/ExampleKneeJoint.cpp
1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/ExamplePendulum.cpp
1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/JaredsDude.cpp
1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/Pendulum.cpp
1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/README.txt
1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/README_Makefile.txt
1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/README_VisualStudio.txt
1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/CMakeLists.txt
1> -- Up-to-date: C:/Temp/mysimbody/install/examples/simbody/Makefile
===== Build: 1 succeeded, 0 failed, 252 up-to-date, 0 skipped =====

```

Build succeeded

6. **Switch to release configuration:** Now change the configuration in Visual Studio to “RelWithDebInfo”. This is the production configuration we recommend when you want to build “release” (full speed) libraries; see beginning of this step for discussion. We’ll refer to these fast libraries below as “release” libraries regardless of which way you built them.

7. **Build, test, install the release configuration:** Build the `ALL_BUILD` target. Check that the build was successful. Build the `RUN_TESTS` target (158 tests take about 80 seconds on my i7). Check that all tests passed. Build the `INSTALL` target. This installs all the full-speed release libraries. (You don't need to rebuild the Doxygen documentation; that is the same for release and debug.)

WARNING: if you build both the debug and release libraries, be sure to build the release ones last. That's because both builds will create a `VisualizerGUI.exe` with the same name and 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.exe` and only used it if you want to debug the Visualizer itself.

Step 5: Play with Simbody

While you are in Visual Studio with the Simbody solution file, you will see a bunch of “Examples” in the Solution Explorer window. These use the Simbody Visualizer 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 have the configuration set to “RelWithDebInfo” (or “Release”) not “Debug” – everything will run *very* slowly in Debug.
 - Right click on one of the targets whose name begins with “Example –”. Select “Set as Startup Project” from the drop down list.
 - Hit Ctrl-F5 (short for Debug/Start Without Debugging) to start the program. For most of them you will get a console window and a Visualizer display window. Some require input at the console before they start the simulation.
 - Repeat for other examples.
 - If that's not enough, you can play with some of the `Test_Adhoc` or `Test_Regr` (regression) projects.
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 download the examples from the Simbody project downloads page on Simtk.org (<http://simtk.org/home/simbody>, click on “Downloads”).
3. Try working through the first tutorial from the Documents page of the Simbody project (<https://simtk.org/home/simbody>).

NOTE: when using Simbody from outside of Visual Studio you must set your PATH environment variable to include the “bin” subdirectory of the installation directory, e.g. “C:\Program Files (x86)\SimTK\bin” or for the example we used above, “C:\Temp\mysimbody\install\bin”.

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 Simbody/doc subdirectory has tutorials. You can build your own doxygen API documentation, or use the posted documentation at https://simtk.org/api_docs/simbody/api_docs30/Simbody/html/index.html. There are lots of examples, and the source code (especially the header files) has lots of comments, some possibly helpful.

Appendix: Notes for building 64 bit (x64) Simbody

Simbody depends on Lapack and Blas for numerical linear algebra computations. The source release includes a pre-built 32-bit (x86) binary called SimTKlapack.{lib,dll} that includes both Lapack and Blas for Windows. This is a fairly high-quality implementation built with Atlas but assumes only lowest-common-denominator capability from the processor so that it works on almost any x86 platform.

For most systems Simbody’s performance is not limited by Lapack so you will probably not gain much by replacing the default SimTKlapack with a faster one. However, since we don’t currently provide a 64 bit (x64) Lapack, if you want to build 64 bit Simbody you’ll have to make or find a 64 bit Lapack. For this you might try Intel’s Math Kernal Library (MKL) or AMD’s Core Math Library (ACML). In the CMake GUI, look for the field `BUILD_USING_OTHER_LAPACK`, which is blank by default. There you can put a semicolon-separated list of libraries that should be used in place of SimTKlapack. For example, if you had a `mylapack.lib` and `myblas.lib` you would set `BUILD_USING_OTHER_LAPACK` to “`mylapack;myblas`”.

In addition to Lapack, Simbody depends on the pthreads library and the Visualizer depends on glut. The source distribution includes 32 bit binaries for those libraries, called `pthreadVC2.{lib,dll}` and `glut32.dll`. For a 64 bit build you have to get a 64 bit version of `pthreadVC2`. To build a 64 bit Visualizer you would also need a 64 bit `glut32`, but you could alternatively use the 32 bit Visualizer with your 64 bit libraries, since the Visualizer is a separate executable. Post to the Simbody help forum if you want some help getting 64 bit versions of these libraries, or even better post to offer them if you have already obtained them!

We have not yet built our own 64 bit Windows version of Simbody so would be interested to hear if anyone attempts this. We do build 64 bit Mac and Linux versions so the source code *should* compile OK. Please keep us up to date with your trials and successes by posting to the Simbody help forum.