# SimTK.org Build System

Requirements for SimTK 1.0

Bryan Keller

*Version 1.0, August 19, 2005*

## 1. Introduction

As a service to SimTK.org users, and to encourage high-quality code on SimTK.org, we want to provide a standardized build system for projects hosted on SimTK.org. We will put in the time and effort in setting up such a system so we, and others, can benefit. Many developers do not have the skills or the time to set up such a system. Eliminating redundant work on infrastructure like this will allow developers to spend more time working on their projects.

We benefit from having others use the system as well. It will encourage developers to have better tested and documented code on SimTK.org, which in turn makes the SimTK.org repository more valuable. Also, if we want to integrate a project that uses the build system with the SimTK framework in the future, there will be less complexity in porting the build system to the one that the framework uses, if both are using the same system.

Generating a standard build report will make it easier for those browsing SimTK.org projects to understand build reports. Once a user figures out how to check if tests are passing or if code is compiling for one project, he or she will be able to check other projects that use the system quickly and easily. If every project had a different look and feel to the reports, results could become difficult to decipher across many projects.

SimTK.org will eventually have a peer review mechanism for projects to help separate the wheat from the chaff. The build system may become a part of this. Projects that have good tests with adequate code coverage, and whose builds are compiling and tests are passing, could be ranked higher than those without any tests.

As with most services on SimTK.org, the build system will be an optional, but recommended, service to developers.

## 2. Components of the 1.0 build system

- Build binaries and executables
- Build documentation from code comments
- Run tests
- Generate build reports
- Publish reports to SimTK.org
- Email alerts, such as build failures

- The build system for 1.0 will be targeted at SimTK internal projects. Later versions will target external projects.

## 3. Building

### *3.1. Multiple platforms and languages*

The 1.0 build system will support three target platforms: Mac, Windows, and Linux/Unix. A project itself does not necessarily have to build on all three platforms; maybe it will build on one or two of the three. A project might, for instance, have a dependency on a Windows-specific library and thus cannot be built on Linux or Mac. SimTK core projects will build on all three platforms.

The SimTK.org web site will provide a user interface for defining on what platforms a project can be built. This will fall under the "Dashboard Admin" section of a project's developer page.

The project home page will display what platforms are supported by the build system in an easy-to-understand manner. A likely implementation will be to display an OS logo graphic for each supported OS.

The build system will support compilation of multiple languages using various compilers. We will support three compilers at a minimum for 1.0: GCC, javac, and Microsoft's C/C++ compiler (cl). Language that can be compiled using these compilers will be supported by the build system. For 1.0, we will support C/C++, Java, and Fortran. We plan on supporting other languages and compilers post-1.0, so no assumptions will be made in the design that will preclude doing this.

### *3.2. Build script*

The build system will require some type of script or definition file to drive the build. There are many approaches to this out there. Autoconf, cmake, ant, straight makefiles, or scripting languages can all do the job. Whichever system is chosen for 1.0 must run on all three of the target platforms mentioned above and be able to use all of the compilers mentioned above.

We will create a guide on how to set up a project to be built using the selected build mechanism. For 1.0, this guide will be targeted at internal project developers. A longer, more descriptive guide meant for external developers will be created post-1.0.

### *3.3. Building documentation*

Documentation generated from metadata comments in the code will be supported by the build system. We will support two such metadata formats in 1.0, Doxygen for non-Java code, and Javadoc for Java code.

### *3.4. User builds*

Users will have the ability to trigger their own builds on their own hardware.

### *3.5. Nightly builds*

Automated nightly builds will be performed for projects that use the build system. We will host build machines for this purpose. Initially, nightly builds will be performed on the SimTK.org Linux box (or the Linux dev box). When we want to support nightly Windows and Mac builds, we will need to get the appropriate hardware, so this cannot be guaranteed in 1.0.

Nightly builds will be performed only on the platforms supported by the project. Platform support will be set by the project administrator on the SimTK.org web site (see 3.1). Also, a project administrator will be able to opt-out of doing nightly builds for his or her project. There will be an option to do this on the same "Dashboard Admin" section of a project's developer page.

Nightly builds will be triggered via cron jobs on Linux and Mac, or "Scheduled Tasks" on Windows. A nightly build will tag the code in the code repository, then do the build against this tag, so that the build results are reproducible.

### *3.6. Build results*

Warnings, errors, and messages resulting from the build process will be recorded in a standard format so the results can be interpreted in a generic fashion. These results will be uploaded to SimTK.org, and then transformed into a user-friendly report. The documentation generated by the build will also be uploaded and published. Both results from nightly builds, as well as user-triggered builds, can be uploaded to the server.

### *3.7. Email alerts*

The 1.0 build system will send an email alert to the project administrators if a nightly build fails.

## 4. Testing

### *4.1. Framework*

The build system will have a mechanism to execute tests. The build script will define what the tests are and which ones to execute. The test framework will be flexible enough to support any type of test, and thus be as unobtrusive as possible. The framework will support unit and functional tests, and code coverage tests. For 1.0, code coverage testing will only be supported by projects that use GCC. Memory leak testing will be a post-1.0 feature.

We will create a guide on how to write tests and include them in the build script. For 1.0, this guide will be targeted at internal project developers. A longer, more descriptive guide meant for external developers will be created post-1.0.

### 4.2. Multiple platforms

As with compiling, the build system will have the ability to run tests on all three of the target platforms.

### 4.3. Test results

Results from the test execution will be recorded in a standard format, again so that the results can be interpreted in a generic fashion by the build system. The results will be uploaded to SimTK.org, and then transformed into user-friendly reports viewable on SimTK.org.

## 5. Reporting

### 5.1. Gathering results

The build system will gather the results from compilation, document generation, and testing on all of the supported platforms. A client-server approach, where a build is done on one machine and the results are published to a server is the most logical approach for dealing with multi-platform support. Results from a build will be uploaded to the SimTK.org server via FTP.

### 5.2. Generating reports

The results from the build, as mentioned before, will be in a standard format. XML is the logical choice here as it is easy to transform into a variety of other formats. On SimTK.org, the XML will be transformed into a user-friendly HTML report that will be viewable under the “Dashboard” section of a project’s developer page.

One report will be generated every day. Each report will have a summary, or “dashboard” page, which has an overview of all of the build results submitted for that day. A user browsing the report on SimTK.org will have the ability to drill down into one particular build to get details on that build, e.g. to view the test results or build errors and warnings.

There is no way to ensure that a build triggered by a user was done on the code that was checked into the code repository. Tests might be passing for that build, but not for anyone who checks out the code from the repository. Thus builds triggered by users will be clearly indicated in the report with an appropriate disclaimer. Nightly builds will always be done from the code checked into the repository, and thus do not need to be flagged in this way.

### *5.3. Archiving*

We will keep a month's worth of daily build reports on SimTK.org for viewing. Once a build report becomes older than a month, it will be archived and then removed from SimTK.org.

## 6. Future objectives not in the 1.0 build system

- Longer, more descriptive guides on the build system, targeted at external users
- Non-GCC code coverage testing
- Memory leak testing
- Support for additional compilers, languages, and platforms
- Ability for the project administrators to trigger a build on SimTK.org
- Email alert for when code coverage falls below a certain level
- Subprojects, i.e. for 1.0, the build system will build from the root of a project's source tree. A user can create a build script in the root directory that directs which subprojects to compile, but the subprojects in which to compile will not be user-definable from the SimTK.org UI.

## 7. CMake

CMake at this juncture looks like a good option. It supports the three platforms that the SimTK framework is targeting: Windows, Mac, and Linux/Unix. It also integrates nicely with Dart, a client-server build report system (more on that later). CMake is fairly easy to learn. For projects with straightforward build needs, a CMake build file (CMakeLists.txt) may only be a few lines long.

On the downside, CMake is not widely used, so this may be a barrier to entry to some recalcitrant types who are not inclined to use something that is not familiar.

CMake includes ctest, a driver for running tests and generating test results. Ctest can run user-defined tests, and also has built-in support for running code coverage and memory leak tests. The code coverage facilities in ctest work only with GCC, so we will need to add support for other compilers and languages if desired. There are tools out there, for instance, that can perform code coverage for Java programs. We could integrate such tools into the build system. Ctest can use either Purify or valgrind for memory leak testing.

Ctest can work in conjunction with other testing frameworks such as CPPUnit or JUnit, because ctest merely runs a test executable defined in the build script. A developer could build a test executable that links with one of these other test frameworks The SimTK.org build system should not require such a framework be used, for maximum flexibility, but we can provide pointers on how to use such frameworks if using such is desired.

Test frameworks such as JUnit and CPPUnit on their own have the capability to generate pretty test reports like CMake/Dart. Where CMake/Dart differs is in its ability to create a unified report of both the build and the tests, as well as the ability to rollup build results from several sources into a “dashboard”. Given our desire to target multiple platforms, this capability is a huge bonus and alleviates us from having to create such a rollup system ourselves.

Using CMake for performing the build and running the tests has the additional advantage in that build and test results are in a single data format, which in CMake’s case is XML. Having all data from all builds in one format makes transforming the data into nice reports much easier. Dart actually takes care of generating the reports through XSL transformation, so we would not have to worry about the format much, but it does give us flexibility down the road in moving to some other reporting format. As it stands, Dart’s XSL style sheets for generating the reports can be tweaked to our liking.

## 8. References

https://simtk.org

http://cmake.org

http://public.kitware.com/Dart

## 9. Acknowledgements

This work was funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 GM072970. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics.