GEANT 4 NIGHTLY BUILDS SYSTEM

Victor Diez, Gunter Folger, Stefan Roiser

victor.diez.gonzalez@cern.ch, gunter.folger@cern.ch, stefan.roiser@cern.ch

Abstract. Geant4 is a toolkit to simulate the passage of particles through matter, and is widely used in HEP, in medical physics and for space applications. Ongoing developments and improvements require regular testing for new or modified code. Geant4 is a world-wide collaboration and it is developed by many different organizations and people, so integration of new and modified code needs to be tested regularly. Geant4 integration testing has been migrated to the LCG Applications Area nightly builds system, a system which unifies building and testing of the Applications Area projects.

1. Introduction

Geant4 [1] is a world-wide collaboration of scientists and software engineers whose goal is to develop, maintain and provide support for the Geant4 toolkit. Such a complex collaboration requires detailed tracking of software changes and their integration, through tests for each specific Geant4 component and for the source code compilation. The previous Geant4 testing system was based on a customized version of the Mozilla Bonsai tool [2], a tool to record CVS [3] tags, to collect and select tags for testing; a set of shell and Perl scripts to submit building of the software and running the tests to a set of Unix platforms; and the Mozilla Tinderbox tool [4] to collect and display test results. Mac OS and Windows were not integrated in this system.

The Geant4 testing required some improvements and the addition of several tests. This was difficult with the previous system and it needed extra manpower to understand and improve its code. The LCG Applications Area nightly build system [5] was already used by many projects, so Geant4 decided to migrate to that system.

2. The LCG Applications Area nightly build system

Geant4 testing has been integrated into the LCG applications area nightly build system. The LCG nightly builds system is based on CMT [6], a configuration management tool, and on python scripts. It supports testing on many different platforms, where we call a platform a unique combination of architecture, operating system and compiler (e.g. gcc3.4 compiler in an ia32 architecture running under slc4). The supported operating systems now include Windows and Mac OS.

The LCG build system organizes the projects into slots. A slot consists of interdependent projects, which are built and tested in order, making the binaries of the previous ones available for the next projects.

CMT is responsible for the configuration of the build and test environment and external dependencies in a structured and modular way, giving fine control of configuring options for the build and execution of tests.
For the testing itself, the LCG nightly build system provides QMTest [7], a test suite providing tools to test software and to present the test outcome in different formats.

The G4 collaboration requires a testing system to test integration of the ongoing developments and to give feedback to the individual developer to be able to fix bugs and errors. The CMT tool makes the configuration process very modular, and adding a new external requirement just requires adding a new line to the requirements file. The nightly builds are organized in five different steps: get, config, build, install and test.

2.1. Get

The get stage consists of getting the code and making it available for the nightly builds system. It can be via CVS, SVN, from a tar file, copying the files directly, etc…

The Geant4 source code is stored in a CVS repository. Due to the many developers collaborating in Geant4, a tag collection system was implemented to choose the tags to test instead of testing the head of the repository. The developers can propose a tag for testing once they have committed and tagged the code. The System Test Team (STT) will select the tags for testing and, depending on the build and test outcome, will accept or reject each tag. The tool to select the tags is based on Bonsai, which records CVS tags and allows us to interact with the repository and to manage the tags in a MySQL database (Figure 1). The nightlies will obtain the list of tags to check out from a web server which queries the database.

![Figure 1. The web interface of the Bonsai-based system](image)

Given the collaboration nature of Geant4, and the geographical dispersion of its collaborators, there is a possibility that a change made on one part of the code can affect the build process, making it fail and thus the rest of the build and testing process. This is the main reason Geant4 decided to use a management of tags instead testing the repository head directly.
The system also supports testing from officially distributed tar files, instead of checking out the code, so we can test the official releases using the same file that the users will get.

2.2. Config

The config stage consists in preparing the source files for the build process. It can be viewed as a wrapper equivalent to the configure command. In Geant4 it is not used, except if the chosen method of getting the code has been the tar file, in which case the tar file containing the code will be expanded.

2.3. Build

The build stage consists in running the project-specific scripts in order to compile and obtain the libraries and executables required for the testing.

CMT will set up the environment with all the environment variables needed for the Geant4 compilation process. The build system of Geant4 is based on GNUmake files and running make will create all the needed binaries. On Windows we use the Cygwin implementation of GNU make to be compatible with GNU makefiles, and the compiler used is VisualC++ 9.0. The compilers used on Linux (SLC 4/5) are gcc3.4 and gcc4.3, and in MacOS we use gcc4.01.

2.4. Install

The install stage consists in installing the binary files (libraries, data, executables…) to a shared folder to make the binary build products available to other parties.

Until now Geant4 does not make use of this feature, as none of the LCG Applications Area projects do have dependencies on Geant4.

2.5. Test

For the testing in Geant4 we use QMtest, a testing suite written in Python which simplifies the process of software testing. It standardizes testing by running all tests using a standard interface. It is easy to customize the output or give extra information, as well to add extra actions to the testing. Also it provides the ability of grouping several tests into “test suites”.

Geant4 testing currently consists of 85 individual tests. In Geant4 we understand as a test the build of the example as well as its execution, so every testing program for the Geant4 code will consist of at least two tests in QMtest. That way, the 85 tests are translated into 175 QMtest tests. We have a general suite containing a group of suites, so to change the tests that we want to run we just have to edit the corresponding “sub-suite”. QMtest provides dependencies from one test to another, so if the compilation of a given example has not been successful, its execution will not be launched. Also, some examples or test binaries are run more than once, using different input files to test different parts of the code. Some additions have also been made to keep the output of all the tests in files, an option not supported by the QMtest default test class. To make these additions, as well as to sort out some problems with testing under Windows platforms, we have created our own Geant4 QMtest class, expanding the default class with our own requirements.

3. Postprocessing

The main goal of testing is to provide a summary across platforms of the testing process to the Geant4 developer team. Once the tests are finished on an individual machine, the log files are copied to an AFS shared folder, concentrating the results on the web server disk space.

The Linux testing machines will copy the log files themselves to the web server shared disk space. For Windows and MacOS the central server copies the log files from the test machines, as these two platforms don’t have write access to AFS. On the central server this is implemented by acron jobs running when the machines are expected to have finished.
A parser process will analyze all the files related to the build and test processes to generate HTML output. The user will be able to access that HTML output, as well as to the log files, via a webpage (Figure 2). The site will show the results of the building and the testing by project and platform.

Each result will be represented in a colored box representing the state. A red box means that there were errors and it will indicate the number of errors in parentheses. Orange means warnings, also specifying the number. And green means that there were no errors or warnings.

This webpage also provides statistics about the evolution of failing tests for the last thirty days (Figure 3).

![Figure 2. Web with the daily result of the nightly builds](image)

![Figure 3. Statistics given by the web page](image)
That way, the testing cycle is closed, since the developer gets feedback on success or failure, and tags are accepted or rejected depending on the results.

4. Ongoing improvements

Currently, the current system overwrites the results every week, without keeping data for history and valuable statistics about long term evolution on testing. Also, the data is not accessible for another purpose, as it is in an html file, mixing data and presentation. We have decided to separate data and presentation into two layers, storing all data in a database. We chose MySQL because is already used within the project.

We chose to use a database because it is an organized and fast technology of storing data and it can be accessed easily from a web server. Also, it is easy to extend for future features and its speed is less sensitive to the increase of data. That way, the data mining to prepare better statistics and compare results in time will be much easier and more accessible. The possibilities include statistics by platform, project or test, giving different views in the webpage instead of a unique and static view.

The structure is designed to keep a similar organization as in the current web page, organizing the nightlies data in slots, using the projects as rows and the platforms as columns (Figure 4). Once is in production, the parser process will store the data in this database instead of creating a static html file.

We have a central table (nightlies) storing the results organized by date, platform and project tag. The interdependent tags are organized in slots. Given this structure, the global parent table is the slots table. The platforms on which every slot is executed are stored in another table (slot_columns), the same way with the tags the slot contains, keeping the order of the tag into the slot (slot_rows). From the tag, we will refer to which project the tag belongs to, stored in another table.

We will use another table to store the tests and a reference to which project they belong. Finally, a table (nightlies_tests) linked to the central table containing the results will keep information about which tests have been run every night and their results.

![MySQL database schema for the new storing system](image-url)
5. Summary

Geant4 has successfully migrated to the LCG Applications Area nightly builds. The migration to the LCG Applications Area nightly build system has allowed us to test regularly and automatically on MacOS and Windows, thus fully integrating these two platforms in Geant4 regular testing. In addition, in the new system we have added several new test cases while this was difficult with the previous Geant4 testing system.

With the ongoing addition of the database storing capabilities to the current system, the possibilities for offering statistics will increase considerably, and its implementation will be fast and easy.

Further ideas for improvements include 'on-the-fly' automatic tag testing, parallel execution of tests, improvements on the time use of the server, automatic patches testing and more efficiency improvements.

6. References

[1] S. Agostinelli et. al. (Geant4 collaboration), Geant4 – a simulation toolkit, *Nuclear Instruments and Methods in Physics Research A* 506 (2003) 250-303

J. Allison et. al. (Geant4 collaboration), Geant4 developments and applications, *IEEE Transactions on Nuclear Science* 53 No. 1 (2006) 270-278

[2] http://www.mozilla.org/projects/bonsai/

[3] http://ximbiot.com/cvs/

[4] http://www.mozilla.org/projects/tinderbox/

[5] http://lcgapp.cern.ch/

[6] http://www.cmtsite.org/

[7] http://www.codesourcery.com/qmtest