

Using CTest to test CALICE software

Jack Hansom

University College London
Department of Physics and Astronomy

Supervised by:
Shaojun Lu

September 8, 2010

Abstract

In this paper I will present the work I have done in the FLC HCAL group during the 2010 summer student program. The aim of this project was to automate the testing of the CALICE software using CTest, a powerful testing client distributed with CMake. The software packages tested in this project were calice-reco, calice-sim, and calice-userlib.

1 Introduction to CTest

CTest is the testing driver distributed with CMake (an open-source, cross platform build system) which provides the ability to perform several different types of test on a software package. There are several reasons why CTest was used to perform the testing of the CALICE software.

The CALICE software was written using CMake, which already has knowledge of the testing process (as CTest is distributed with it). This makes it very easy to include tests as part of the software package. CMake will recognise the tests added in and even compile them if needed.

There are many different areas of software development that need to be tested: to see if the software compiles (smoke testing), if the binary directory builds properly, if CMake configures the build directory properly, or other more specific tests. Writing a script using CTest allows the developer to test all of these at the same time. CTest then collects the results from all of these tests and writes them as xml files, this is very useful as it tells us exactly where the errors lie.

CTest is also compatible with version control systems (such as Subversion or others) which allow the user to automatically update the local source directory before performing the tests. This is very useful for a project in which there are several developers situated in different areas of the world, making it difficult to keep track of different updates.

Another major benefit from using CTest is that it is compatible with the DART Dashboard system. This means that it is possible to send the results of the test to a server that will post these as HTML. This makes it very easy to detect where there are errors, as well as giving developers around the world easy access to these results.



Figure 1: CMake logo

2 How the CTest automation was implemented

In this project we used the "Nightly" model of CTest. This means that CTest will check to see whether the current version of the source code is the same as the version in the repository at a given time (we used CEST 00:00). If this is not the case it will checkout the newest version before performing the test.

CTest has several possible steps during a test: Start, Update, Configure, Build, Test, MemoryCheck, Coverage, and Submit. Five different CTest scripts were written for this project and different steps are used for each of these. One script was written for the calice-sim package, one for calice-

reco, one for calice-userlib, one to test the full CALICE software, and one to perform a memory check on the software. We will now go through how the most useful steps were implemented in these scripts. To see how CTest scripting works in more detail, see the CTest documentation [1] and the user manual.

2.1 The update step

In all the scripts we used subversion as a version control system. This compares the version numbers of the source code and the repository (at a given time, because we are using the Nightly model) and if they are different, subversion will checkout the version in the repository. This makes sure that all nightly dashboard entries are based on the same source code.



Figure 2: Subversion logo

2.2 The Configure and Build steps

In the scripts, there is a command to wipe the binary directory before the configure and build steps. This makes sure that we are testing a clean build every time the dashboard is run.

Then in the Configure step CTest runs CMake on the project to guarantee that the Make files are up to date. For that it uses a Cache file called user-pro-test.cmake . This creates the build directory.

In the Build step CTest runs gmake to build the project in the build directory.

2.3 The Test step

Tests are added to the source directory as either executable files or non compiled code. By default if the test returns 0, CTest will output that the test has passed. Any other value and CTest will say that the test has failed. Any extra output from the tests will be displayed in the xml file in the build directory or on the dashboard.

Recently, using an older version of the CALICE software (v 04-00), running caliceMarlin on the input file Reco350389.xml using steering file Run350389.000.slcio was known to crash due to a Segfault after approximately 10000 events. This was recreated in calice-test, and the same error was found (as shown in fig3). This indicates that the current setup for CTest would be able to detect similar errors in the future.

At the time of writing there is only one test in the calice-test source code.

```
[ VERBOSE "ProgressHandler"] Speed: last event took 40ms since last report 36.3ms/event rate: 27Hz
[ VERBOSE "ProgressHandler"] *****
[ VERBOSE "ProgressHandler"] Current run: 350389 (1 runs so far) processed events: 10578 (10578 total) processtime: 504s
[ VERBOSE "ProgressHandler"] Current event: 10577 time: 22.08.2007 01:34:44.522945000
[ VERBOSE "ProgressHandler"] Speed: last event took 30ms since last report 34.4ms/event rate: 29Hz
[ VERBOSE "Conditions"] Front end configuration: calib off.
[ VERBOSE "Conditions"] Front end configuration: calib off.
[ VERBOSE "Conditions"] Trigger configuration 3 enabled bits:
[ VERBOSE "Conditions"] Trigger configuration bits:Triggers: pure PEDESTAL;
[ VERBOSE "Conditions"] Individual Software trigger off
[ VERBOSE "Conditions"] Trigger configuration bits:Triggers: pure PEDESTAL;
[ VERBOSE "HcalPedestalProcessor"] RunTimeConditionsHandler::update: no default collection set for conditions handler: Ahc
[ VERBOSE "HcalPedestalProcessor"] , do nothing

*** Exception executing: Segmentation fault
```



Figure 3: CTest error message for CALICE v04-00

The caliceMarlin executable is run using an input file (Run580089.000.slcio) and a steering file (Reco580089.xml). This test uses the ILC-SOFT and CALICE-SOFT libraries so it is a good test to see if the CALICE software does what it is supposed to. However this does not mean that the output from caliceMarlin is reasonable, further tests will need to be implemented in order to do this.

If any further test is needed in the future it is very easy to include one in the source directory without changing the script. The procedure to do so is described in the "Using CTest for CALICE software" user manual written for this project.

2.4 The Memory Check step

In the calice-memcheck script, we use valgrind to perform the memory check of the same executable as in calice-test. Valgrind is a programming tool for memory debugging, memory leak detection, and profiling. In our script we use the memcheck tool with detailed memory leak detection, this means that valgrind gives us more details of each individual leak but this takes more time to perform. The memcheck tool can detect many common memory related errors in C or C++ programs.



Figure 4: Valgrind logo



Figure 5: View of CALICE Nightly Dashboard

2.5 The Submit step

CTest allows users to create Dashboards which are extremely useful for summarising the results of a nightly test on different platforms. Its hyperlinks allow for quick and easy access to additional levels of detail.

The CTest scripts written send the results to the CALICE dashboard [2]

Figure 5 shows what a developer would typically see the morning after a test.

2.6 The automation

the next step of the project was to get the five scripts running automatically at night. CTest itself doesn't offer a way to run the tests automatically at night, so we achieved it the following way.

First a shell script was written. This script contains all the commands needed to run CTest on the five scripts and submit the results to the dashboard.

Then Cron was used to run this script automatically at 00:30 at night (CEST). Cron is a time based scheduler for Unix-like systems. The nightly test uses the version in the repository at midnight so we decided to run the test shortly after midnight so that it is unlikely for someone to update the

software just before the test runs.

3 Conclusion

In this project, five CTest scripts were written to test the building process of calice-sim, calice-reco, and calice-userlib, as well as testing and memory checking the CALICE software. This testing is now completely automatic and runs at night. The results from these tests are easily accessible through the CALICE dashboard. This improves the efficiency of the CALICE software testing by providing easy global access to the results of nightly tests, allowing for quicker debugging.

The way CTest works also makes the testing procedure very modular, additional tests can easily be included in the nightly testing. A user manual was written to explain in more detail how these scripts work, and how to add to them.

There is one step that wasn't implemented, which is the Coverage step. However this step is only useful for white box testing, and there are so far no such tests in calice-test.

The test we have described earlier shows whether the software works properly (ie: doesn't crash or return exceptions), but this could be improved upon by writing a test root macro to check whether the output from caliceMarlin is reasonable. This would provide a more complete test of the CALICE software.

References

- [1] <http://www.cmake.org/cmake/help/documentation.html>
- [2] <http://zitxserve1.desy.de/~aplin/CDash/>