EUDET-Memo-2010-18

A test system for the iLCSoft framework

S.J. Aplin[*], J. Engels[*], F. Gaede[*]

January 5, 2011

Abstract

This memo presents the motivation, requirements and implementation of a new software testing system developed within the iLCSoft software framework. A brief introduction to the iLCSoft software framework is also given.

[*]Deutsches Elektronen-Synchrotron, Hamburg, Germany

1 Introduction

This memo presents the design and current implementation of a new test system currently being developed for the iLCSoft software framework [1]. The development has been performed partly within the context of the EUDET [2] project.

A short overview of iLCSoft is given in Section 2. This is followed in Section 3 by a discussion on the motivation and requirements for a test system in iLCSoft. Section 4 gives an overview of the CMake family of tools. Finally in Section 5 the design and current implementation of the new test system, iLCTest, is described.

2 Overview of iLCSoft

iLCSoft is a software framework that was originally developed for studying the Monte-Carlo simulation of a full detector design for the ILC. Within the EUDET project, iLCSoft has subsequently been extended and adapted for test beam data processing, and is now used by all three EUDET JRA's. The software framework provides the core software tools for the development of reconstruction and analysis software, and allows these to be used in conjunction with Mokka [3], a Geant4 based detector simulation application, through the provision of the LCIO [4] Event Data Model (EDM) API and the Gear [5] detector geometry API. Besides LCIO and GEAR, iLCSoft contains the application framework Marlin that can be used for all tasks which involve the processing of data stored using LCIO. It follows a modular design where every computing task is implemented as a Processor that analyses data in an LCIO Event (LCEvent) and produces additional LCIO Collections that are subsequently added to the event.

Several reconstruction and analysis software packages which are based on the Marlin framework are included within iLCSoft. These include higher level reconstruction algorithms, such as the Pandora particle flow algorithm [6] and the LCFI vertexing package [7], as well as complete test beam reconstruction applications, such as MarlinTPC [8] and EUTelescope [9]. The software used within the CALICE collaboration [10] is also built upon the software contained within iLCSoft.

The use of the same core software framework for the EUDET JRA test beam experiments as that which is used for the large scale detector optimisation studies within the ILD detector concept study, provides synergies in both of these two related areas of study. Such synergy is readily evident, not only in the greater number of users contributing to the improved quality of the software overall, but also in the evolution of both the Event Data Model API [12] and the geometry description API [11].

More information on iLCSoft can be found at http://ilcsoft.desy.de.

3 Motivation and Requirements for a test system in iLCSoft

Since its inception, iLCSoft has grown in order to meet the increasing needs of the user community. As has been described above, iLCSoft comprises of both core tools, such as the LCIO EDM API and GEAR geometry API, as well as applications covering simulation, data taking, and event reconstruction, thus leading to a full and relatively complex software framework. It should also be noted that due to the relatively high turn-over of software developers within the field, at times it can prove difficult to retain detailed knowledge of individual software packages, some of which, while they may still be actively used, have ceased to be actively developed.
Consequently, an automated testing system that goes beyond the traditional, compilation based, “Nightly Build” system, could provide a early indication of potential integration issues, and possible unexpected consequences, allowing them to be rectified at an early stage in the development cycle.
A suitable system should provide an efficient mechanism for developers to create, include, and evaluate tests. With the scope of these tests ranging from class level unit tests through to integration tests of the full simulation and reconstruction framework. Moreover, it is important that tests should be available which go beyond ensuring the technical integrity of the software itself, and that these tests are able to assess the results of the algorithms themselves, based on some suitable metrics.
In addition, the running of the tests should be automated with prompt feed back provided to developers when tests fail. Finally, it would be useful, although not mandatory, to provide indicators of quality assurance over a period of time incorporating one or more development and release cycles.

4 CMake family of tools

4.1 CMake

CMake [13] is used as the primary build tool within the iLCSoft framework. CMake is able to generate native build environments for a number of different platforms and IDEs, and is presently used for major Open Source projects such as KDE [15]. The project’s build parameters are defined in an easy-to-learn CMake-specific language contained within platform independent text files named CMakeLists.txt. These files are used by CMake to produced the system-specific Makefiles. Built-in features of CMake provide for managing inter package dependencies and makes cross platform development easier to maintain, making it particularly suitable for the iLCSoft framework.

4.2 CTest

CTest is a testing client which forms part of the CMake family of tools. It allows the definition of simple software tests, as well as more advanced testing scenarios, such as those mentioned in Section 3. Simple software tests, such as unit tests, usually tend to live inside the CMakeLists of the package itself, however, more advanced tests which chain together a different set of packages, typically live in a separated package, such as iLCTest. In this context it is important to note that CTest can be used independently of CMake and therefore suits both these scenarios perfectly well.

4.3 CDash

CDash [14] is a web based testing server used to build quality assurance dashboards. Once a CDash server is running, a configuration file may be exported to any CTest client to enable it to publish test results directly to the dashboard. This setup is typically used in conjunction with a cronjob which executes a suite of tests on a nightly basis, and provide a continuous check of the quality and integrity of a software framework.
CDash is able to generate dynamic plots of various quantities pertinent to software quality, these include software build times, execution times, and the number success and failures and warnings over a given time interval. This feature provides valuable information for tracking down software bugs or performance penalties as modifications are added to the software code repository.
Upon submission of test results to CDash, developers are imediately notified by email if any test fails within packages to which they have previously subscribed. This is particularly appealing to a geographically dispersed international community, such as the ILC, where developers wish to be promptly notified as soon as any of their code contributions break the current development version of the common software framework.

5 iLCTest

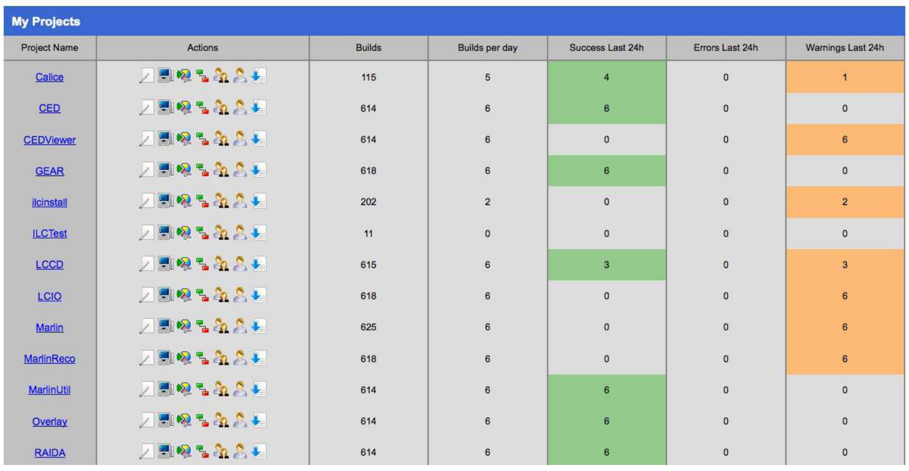
The iLCTest package was established in order to fullfill the requirements listed in Section 3. By making use of the CMake family of tools introduced in Section 4, this package aims at providing a central testing package for the iLCSoft framework.

5.1 Implementation

The iLCTest implementation can be divided into the following main components:

- CDash web interface for displaying and browsing quality dashboards
- C++ utility headers to simplify writing tests
- CMake utility macros to simplify adding new tests
- Example Tests directory

The CDash web interface is the most significant aspect as it gathers all information of the testing infrastructure in a single place, making it easily accessible to all iLCSoft developers. Fig. 1 shows the main overview of the iLCTest web interface. The remaining components contain infrastructure code to facilitate adding new tests in a straightforward manner with minimal overhead for software developers.

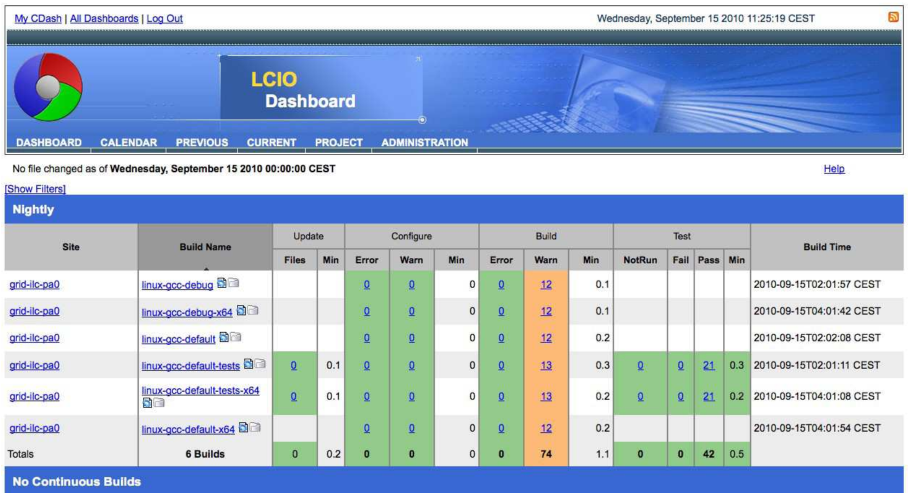

My Projects

Project Name	Actions	Builds	Builds per day	Success Last 24h	Errors Last 24h	Warnings Last 24h
Calice		115	5	4	0	1
CED		614	6	6	0	0
CEDViewer		614	6	0	0	6
GEAR		618	6	6	0	0
ilcinstall		202	2	0	0	2
ILCTest		11	0	0	0	0
LCCD		615	6	3	0	3
LCIO		618	6	0	0	6
Marlin		625	6	0	0	6
MarlinReco		618	6	0	0	6
MarlinUtil		614	6	6	0	0
Overlay		614	6	6	0	0
RAIDA		614	6	6	0	0

Figure 1: CDash main overview of the iLCTest web component.

5.2 CDash Web interface

All packages in the iLCSoft framework have their individual web component where all major steps of the project: update of the local svn working copy, configuration, build, and testing, are all mapped column-wise into a table, where each row stands for a different platform used for executing the tests. An example of such a table is shown in Fig. 2.



No file changed as of **Wednesday, September 15 2010 00:00:00 CEST**

Nightly

Site	Build Name	Update		Configure			Build			Test				Build Time
		Files	Min	Error	Warn	Min	Error	Warn	Min	NotRun	Fail	Pass	Min	
grid-ilc-pa0	linux-gcc-debug			0	0	0	0	12	0.1					2010-09-15T02:01:57 CEST
grid-ilc-pa0	linux-gcc-debug-x64			0	0	0	0	12	0.1					2010-09-15T04:01:42 CEST
grid-ilc-pa0	linux-gcc-default			0	0	0	0	12	0.2					2010-09-15T02:02:08 CEST
grid-ilc-pa0	linux-gcc-default-tests	0	0.1	0	0	0	0	13	0.3	0	0	21	0.3	2010-09-15T02:01:11 CEST
grid-ilc-pa0	linux-gcc-default-tests-x64	0	0.1	0	0	0	0	13	0.2	0	0	21	0.2	2010-09-15T04:01:08 CEST
grid-ilc-pa0	linux-gcc-default-x64			0	0	0	0	12	0.2					2010-09-15T04:01:54 CEST
Totals	**6 Builds**	0	0.2	**0**	**0**	0	**0**	**74**	1.1	**0**	**0**	**42**	0.5	

No Continuous Builds

Figure 2: LCIO Nightly Tests overview.

Furthermore, the web interface generates useful plots containing test execution times (Fig. 3) and a history of previous builds (Fig. 4), allowing developers to continuously keep track of the overall framework performance and easily detect changes that may lead to penalties in the total execution or build time of the software framework.

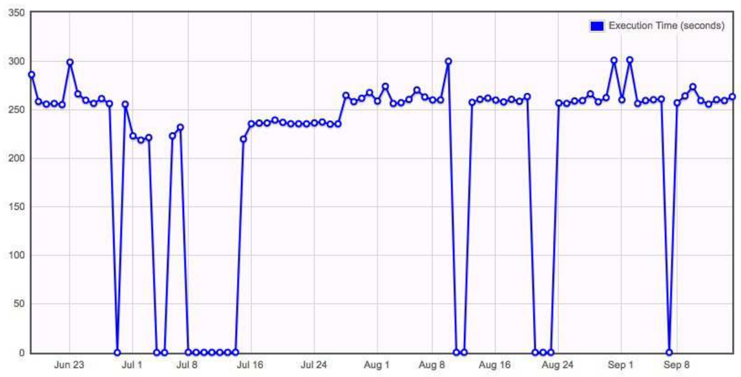


Figure 3: Marlin execution time history.

The web interface includes a substantial amount of links allowing the user to browse through the individual packages either by their test results or history and to easily identify possible sources of errors or compiler warnings.

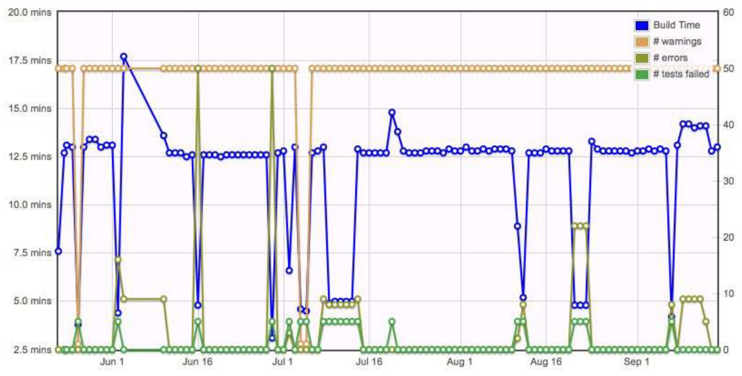


Figure 4: iLCSoft 32bit build history.

5.3 C++ utility headers

The C++ utility headers provide functionality to facilitate writing of new C++ test applications and to simplify their evaluation in CTest using regular expressions. This is done by using a simplified set of logging methods that can be easily checked in the CTest scripts.

5.4 CMake utility macros

CMake utility macros were implemented in order to facilitate the adding of new tests to the iLCTest package. These macros also provide functionality for managing the execution of tests within the tests directory decribed below.

5.5 Example Tests directory

The tests directory contains a set of simple examples to demonstrate the usage of the testing infrastructure. These examples are organised hierarchically, in a tree-like directory structure to facilitate activating or disabling individual branches of the tree as required. The examples in this directory include a simple 'hello world' test application written in C++, an example for writing and reading back a sample ROOT [16] histogram, and a more elaborated test that builds a Marlin plugin, loads it into the Marlin application, and check its output for a set of regular expressions which must be fullfilled in order for the test to pass.

6 Conclusion

The new software testing system, iLCTest, has now been established within the iLCSoft software framework and has already been successfully used during the development cycle of release v01-10 of iLCSoft. Now that the testing framework has been established and proof-of-principle demonstrated, testing coverage will be extended by the addition of further tests. This will involve the addition of tests to address existing functionality, as well as ensuring that new features are provided with suitable tests as they are added to the framework.

Acknowledgement

This work is supported by the Commission of the European Communities under the 6^{th} Framework Programme "Structuring the European Research Area", contract number RII3-026126.

References

[1] F. Gaede, J. Engels, "Marlin et al - A Software Framework for ILC detector R&D", EUDET-Report-2007-11.

[2] EUDET URL: `www.eudet.org`

[3] Mokka URL: `http://polzope.in2p3.fr:8081/MOKKA`

[4] LCIO URL: `http://lcio.desy.de`

[5] GEAR URL: `http://ilcsoft.desy.de/portal/software\_packages/gear/index\_eng.html`

[6] M. Thomson, "Particle Flow Calorimetry and the PandoraPFA Algorithm", NIMA 611 (2009) 25-40.

[7] S. Hillert for the LCFI Collaboration, "LCFI Vertex Package", arXiv:0811.4759v1 [physics.ins-det] 28 Nov 2008

[8] J. Abernathy, P. Conley, K. Dehmelt et al. "Update on the Status of MarlinTPC", Eudet-Memo-2009-16

[9] A. Bulgheroni, P. Rolof, J. Behr, A. F. Zarnecki, Y. Furletova ,"EUTelescope, the JRA1 tracking and reconstruction software: a status report (Milestone)", EUDET-Memo-2008-48

[10] R. Poeschl, "Calice Data Processing", EUDET-Memo-2007-57-1.

[11] M. Killenberg, S. Turnbull, "A Modular TPC Endplate Description for GEAR", EUDET-Memo-2008-31.

[12] V. Boudry, "Proposal for an LCIO format for the DHCALs", Presented at the Annual Meeting 2009 of the EUDET consortium

[13] Martin, K., Hoffman, B. 2005, *Mastering CMake*, Kitware, ISBN 1-930934-16-5

[14] CDash URL: `http://www.cdash.org`

[15] KDE URL: `http://www.kde.org`

[16] ROOT URL: `http://root.cern.ch`