



Automating Component Test Insertion into DØRunII CTBUILD Software Packages

Mariano A. Zimmler
New York University

Supervisor
Dr. David J. Ritchie
Computing Division

Fermi National Accelerator Laboratory
Summer Internships in Science and Technology

Summer 2002

DØ Code Releases

- ◆ DØ makes weekly **test releases** of all its software packages to IRIX and Linux.
- ◆ Test releases are used by developers to introduce **new functionality**.
- ◆ Every three months, a **production release** is started based on a test release. Its purpose is to **fix bugs** and introduce minor changes.
- ◆ In the past, the web-page containing the status of these releases had had to be maintained manually but this turned out to be too time consuming.

CTEST/CTBUILD

CTEST is DØ's standard code management environment. It defines a convention for the physical organization of both the code and the instructions for building C++ software. CTBUILD is its implementation.

- ◆ The fundamental unit of software is the **component**. Components are grouped into **packages** and packages into **subsystems**.
- ◆ Each component consist of a **header** file, an **implementation** file and a **test** file. The implementation file contains the source, which is compiled into a bare object or a library. **The test file is a main program which tests the header interface and implementation source and returns zero if the test is successful.**
- ◆ Build actions are controlled by a set of “**instruction files**” located in a package’s top directory or subdirectories.
- ◆ Each of the subdirectories listed in the instruction file **SUBDIRS** is processed after carrying out the action in the current directory.

Dependencies and Testing

- ◆ **Each package builds at most one library.**
- ◆ The library will generally depend on other libraries which are used by the objects contained in the package library. These libraries are listed in **LIBDEPS**.
- ◆ The list of components that are to make up the package library is taken from the instruction files **COMPONENTS**.
- ◆ Two types of testing are supported by CTEST:
 - **Component tests, which apply to all library source files (components): a test file defining a main program must be provided for each component.**
 - **Integrated tests**, which are stand alone tests not restricted to dependence on lower level objects.
- ◆ Both kinds of tests are invoked by a script.

The project

My summer project involved doing an analysis of each of the 585 DØRunII software packages on the following aspects:

- Determine which ones are **CTBUILD** and which ones are **SRT** (SoftRelTools).
- For each CTBUILD package evaluate every component test to determine whether it is
 - **0th order**: a main program that returns zero and does not even include its own header file,
 - **1st order**: a 0th order test which includes its own header file,
 - **2nd order**: a 1st order test which has a constructor for each class.
- Write a 1st order component test.
- Write a 2nd order component test if time permits.

It was estimated that 50% of the packages had some form of 0th order test in them.

First approach to test insertion

- ◆ The first subsystem to be analyzed was Muon. This subsystem had **196** component tests in **23** packages. Of these, **127** were **0th order** tests, many of the form

```
main{return 0}
```

- ◆ A `gmake alltest` revealed errors in two packages due to an incomplete **LIBDEPS** file. These errors were fixed and all the changes were checked back into the CVS repository.
- ◆ Vertexing Subsystem was then analyzed. Of the **176** component tests in **27** packages, **55** were also found to be empty. Compiling the code revealed a broken header file that was referencing an undefined class.
- ◆ The project was producing good results. However, two weeks were needed to process just these two subsystems with this approach by hand. **To complete the remaining 24 subsystems of the DØRunII program suite would have required almost a half a year! It was necessary to automate the process.**

A new approach to test insertion

I proposed my supervisor automating component test insertion by writing a Python script to do it.

The main difficulty for its implementation was posed by the lack of standardization in the structure of the packages.

After having examined several other subsystems, the following observations were made:

- The **COMPONENTS** file was found mainly in three different places:
 - A directory called `src`
 - The top-level package directory
 - Other subdirectories named in arbitrary ways.
- The component tests were found to be in most cases together with the **COMPONENTS** file. There were also cases where they were placed in a separate directory, typically called `test`.
- Header files were found to be mainly in two locations:
 - In a directory with the same name as the package
 - Together with the **COMPONENTS** file.

The script

- ◆ **Three different versions** of the script were implemented to account for all of these options.
- ◆ This implementation still required a **preliminary analysis** of each package structure and the corresponding code changes that applied.
- ◆ Eventually, enough experience was acquired to implement a single script that would merge the other three versions and which would require no preliminary analysis.
- ◆ The script was implemented in a **procedural** way. Its skeleton is shown:

```
def getSubdirs(package directory)
def getComponents(componentsDirectory)
def analyzeComponentTests(componentsList, subdir, package, packageDirectory)
def generateFirstOrderTest(testFilesDirectory, component, includeString)
def main()
```

Summary of the results of adding component tests into every DØRunII CTBUILD package

Subsystems processed: 23

Packages processed: 495

Component tests processed: 3301

Added tests: 1559

Errors found and fixed due to the inclusion of these tests: 44

Conclusions

- ◆ The project was extremely successful. The technique developed for solving this problem proved that it is possible to process certain aspects of the release in an automatic way. Other immediate applications of this technique have already been proposed.
- ◆ This technique also proved that the features of CTEST can be fully exploited for accomplishing tasks unrelated to its original purpose.
- ◆ The estimate that 50% of the components in DØRunII did not have proper component tests proved to be correct: **47.2% of the component tests did not test anything.**



Automating DØ Code Releases Status Generation



Mariano A. Zimmler
New York University

Supervisor
Dr. David J. Ritchie
Computing Division

Fermi National Accelerator Laboratory
Summer Internships in Science and Technology

Summer 2002



Code Releases Status

Overview

this page would display a list of all the releases, stating whether or not they were on disk, and the date and time of the last change made to the inventory file of the release.

Build Status

this page would display the current build number in each of the three build machines (d0mino.fnal.gov, d0lxbld4.fnal.gov and d0lomite.fnal.gov) and each of the two possible compilation setups (debug or maxopt).

Freeze Status

a third page would display the status of the compression of the release into tar files in the same six categories specified in the previous page.

Error Status

this page would display the number of broken packages for each the release builds specified in the Build Status Page.

The Script

```
class Tools:
    def exists(self, objective, path, input)
class Command:
    def __init__(self, commandName)
    def execute(self)
class Build:
    def __init__(self, hostName, version, release)
    def getCurrentBuildNumber(self, buildDir, buildList)
    def getBrokenPackages(self, buildDir, buildList)
    def getBuildData(self, resultsDir, results)
class Release:
    def __init__(self, releaseName)
    def getDateLastModified(self)
    def getBuildStatus(self)
    def getFreezeStatus(self)
    def generateAnnotationsFile(self)
class Database:
    def __init__(self, databaseName)
    def retrieveReleaseNames(self)
    def retrieveRelease(self)
    def update(self, listOfReleases)
class StatusPages:
    def __init__(self)
    .
    .
    def getReleaseStatus(self)
    def generate(self)
```

Script main features

The script has an object oriented design in its entirety.

- ◆ In this case, the **object oriented approach** was adopted due to the nature of the problem. This allows for other scripts to reuse code from it by just **importing** it.
- ◆ It uses an existing module called HTMLgen to generate formatted HTML web-pages.
- ◆ It contains a mechanism for generating a **persistent database** of older releases using Python's `shelve` module.
- ◆ It contains a system for **interacting with the hosting UNIX operating system** using the command `popen`.
- ◆ It implements a method for **sending commands across a network** by means of the UNIX command `rsh`.
- ◆ It implements the notion of a **“Trojan Horse”** script.

Release Status Page (Version: 0.4.0)**Status Overview****Releases On Disk**

Release	On Disk	D0rel1db/inventory Date Last Modified
t226-gcc [A]	yes	Tue Jul 23 01:29:59 2002
t225-gcc [A]	no	Tue Jul 23 13:26:25 2002
t224-gcc [A]	no	Wed Jul 3 16:18:18 2002
t02.27.00 [A]	yes	Sat Aug 3 23:03:40 2002
t02.26.00 [A]	yes	Tue Jul 30 15:09:24 2002
t02.25.00 [A]	yes	Tue Jul 23 00:18:07 2002
t02.24.00 [A]	no	Wed Jul 3 13:51:48 2002
t02.23.00 [A]	no	Mon Jun 24 17:18:45 2002
p12.01.00 [A]	yes	Wed Jul 31 15:19:11 2002
p12.00.00 [A]	yes	Wed Jul 24 15:03:05 2002
p11.10.01 [A]	yes	Fri Aug 2 14:18:44 2002
p11.10.00 [A]	yes	Mon Jul 29 13:33:06 2002
p11.09.00 [A]	yes	Wed Jul 3 12:03:02 2002
p11.08.01 [A]	no	Mon Jul 1 09:33:14 2002
p11.08.00 [A]	no	Thu Jun 13 12:34:17 2002
p10.15.03 [A]	yes	Fri Jun 7 15:40:02 2002
p09.10.00 [A]	yes	Sat Sep 1 22:33:57 2001
onl01.67.00 [A]	no	Wed Nov 21 14:55:42 2001

The '[A]' hyperlink brings up free form information about the release that has been added by the release managers.



Acknowledgements

I would like to thank Alan Jonckheere, Paul Russo and David Ritchie for their help and support. They were key to making these projects a success.