Google Summer of Code 2018

ROOT Package Manager

Organization : CERN-HSF

Sub Organization : CERN, UNL

Mentor(s) : Brian Bockelman, Oksana Shadura, Vassil Vassilev

Abstract

This document describes the proposal that the author submits for the project ROOT Package Manager, from CERN-HSF organization. ROOT is the data processing framework created at CERN - the heart of the research on high-energy physics. Every day, thousands of physicists use ROOT applications to analyze their data or to perform simulations. The ROOT software framework is foundational for the HEP ecosystem, providing capabilities such as IO, a C++ interpreter, GUI, and math libraries. It uses object-oriented concepts and build-time modules to layer between components.

This project aims to provide additional functionality using a package manager over the minimal base install of core features. It involves defining ROOT modules, packages and package manager, mainly to scale the large codebase of the project. The current development involves creating a modular version of ROOT that provides a minimal base install of core features, then later adding functionality using the package manager. This requires introducing new layering mechanisms and extending the functionality of the existing ROOT package manager prototype.

Benefit to Community

Numerous experiments and projects involving very large amounts of data are worked upon at CERN. As ROOT is the data processing framework for hundreds of petabytes of data, the role of this application is crucial. Scientific data handling with ROOT has been easier for physicists and other users too. Simplifying the build process and adding enhancements to it would avoid the task of building ROOT for the users, which serves to be an extra work for those working to integrate it with higher-level toolkits, such as the Python-based data and machine-learning ecosystem. The community would benefit from this project for a faster ROOT installation process leading to earlier and ‘dependency-handled’ , ready to use build

of ROOT. Packages and package management provide a mechanism for ROOT users to socialize and and reuse projects built in the context of ROOT and it aims to make ROOT more flexible and open for new customers. This allows ROOT to continue to serve as a community nexus. Having a package format can help define community standards and allows the community to go in a similar direction, focusing what effort we have and then in the same time to reduce the desire/ability for every experiment and analysis group to ‘invent their own’ set of packaging on top.[1]

Implementation details

Object oriented systems have the abstraction layers to provide separation between user of an object and implementer which helps the project to scale to very large codebases. Another way to scale projects is via modularization - providing explicit boundaries, minimal dependencies, and APIs between areas of the code base.

For small scale projects classes act as the unit of modularization, but for large scale projects with large number of classes, we consider grouping the classes together to create 'modules' and a set of modules and metadata to get a package which serves to be an atomic unit for this project.

ROOT Modularization

ROOT has a strong history of object oriented programming but lacks in the concept of ‘modules’ - set of interdependent classes implementing coherent functionalities and providing well-defined APIs. A good organization of classes into identifiable and collaborative packages eases understanding and maintenance. ROOT being a framework with very large number of classes and packages, it requires package organization.

Modularization is a way of grouping functionalities from a software product. Library dependencies alone result in a very complex relationship diagram. By introducing a

module layer, it would provide boundaries between components - allowing ROOT to scale as a project. By making boundaries and relationships more explicit through modules, we can better define and implement “minimal ROOT”, so as to increase its functionality in being embedded in other contexts, enabling ROOT users to interact with wider data science ecosystem. Module can be seen as a ROOT “component” - package being distributed and maintained through ‘standard’ ROOT - living in root.git. [1] Basically every facility provided by exactly one module.

ROOT contains various packages - each having its own specific functionality. Sometimes the experiments and projects in scientific community requires these packages in groups (to provide more than a single functionality). ROOT base would be a minimal version for any such projects/experiments involving use of ROOT packages. So it would contain only the must-have parts of the ROOT build.

ROOT package manager

ROOT package is a grouping of softwares for data analysis and associated resources, intended for its distribution. It acts to be a contract for code organization - to simplify build and deployment steps. The package is a distinct, self-describing resource.

Package manager locates and installs packages into a ROOT installation from a package reference which are recorded in package database along with transitive dependencies.

The ROOT package manager reduces coordination costs by automating process of downloading and building the dependencies for a project required for users. So, performing steps manually for getting dependencies, writing steps to generate compilation files and then compiling them could be automated. The process used by the package manager is written as a script in a yaml file.

Minimal requirement for the ROOT package manager is to be able to define dependencies and its version - could be done in manifest of package.

For the package manager in this idea, the work involves project/application dependency manager (PDM) - interactive system for managing source code dependencies of a single project in a particular language and Language package

manager(LPM) - interactive tool to retrieve and build specified packages of source code for a particular language. PDMs constantly work out through four separate states on the disk - project code, manifest file, lock file and dependency code.

Over the period of time, ROOT Minimal has migrated away from its “core-like” ROOT - providing just the basic requirements and not other packages. The base module would revive this motive and provide a module starting with three components : Core, RIO and Cling.

ROOT Base Improvement

This would involve the work on improvement of CMake macros and functions used for “root-base”.

Timeline

Community Bonding Period (April 23 - May 14)

- Learn about existing package managers (focusing on the language package managers). 1-2 week.
- Extend the continuous integration to root-base to be able to upload nightly the binaries on the GH releases page. 1 week.

First Coding Period (May 14 - June 10)

- Improve the cmake functions which are related to the workflow of external packages (Code still not in the master). 1 week.
- Implement an ‘externally-mountable’ map which defines how to split ROOT’s codebase into packages in cmake. 1 week.
- Define package format (logical and concrete format) and implement install routine for package manager. 1 week.
- Enhance the package metadata including information about the package dependency direct acyclic graph (DAG). 1 week.

Second Coding Period (June 16 - July 8)

- Teach root-get to show the list of available packages. 1 week.
- Teach root-get to show all installed modules together with it's versioning information. 1 week.
- Teach root-get to resolve dependencies. Investigate the common dependency conflict resolution algorithms. 1-2 weeks.
- Investigate the integration of root-get and the ROOT runtime. 1-2 weeks.

Third Coding Period (July 14 - August 5)

- Teach root-get to look for a package on a predefined set of locations. 1 week.
- Teach root-get to provide a lazy install based on missing package. 2 week.
- Teach root-get to provide an external package resolution. 1 week

Literature:

1. Brian Bockelman, Oksana Shadura, Vassil Vassilev. ROOT Modularization. ROOT-0001, ROOT - Evolution proposals. https://github.com/root-project/root-evolution/blob/master/proposals/0001-modularization.md

Related work (test task) :

I have been in constant contact with Oksana Shadura discussing the project information. In these emails, initially I was asked to work on an exercise. The task involved setting up Travis CI for root-base prototype - a minimal base install of core features.

After making a fork from the following branch
https://github.com/oshadura/root/tree/rootbase-cmake

I added the travis.yml in my own
https://github.com/kiryteo/root/tree/kiryteo-rootbase-cmake branch.

Following is the .travis.yml file for the kiryteo-rootbase-cmake branch

https://github.com/kiryteo/root/blob/kiryteo-rootbase-cmake/.travis.yml

After launching Travis CI for ROOT base, builds were still crashing and having timeout, mentor asked me to try to define CMAKE_CXX_STANDART CMake flag for ROOT base and look out for the result.

After going through the travis documentation for customizing the build I made the following changes :

1. Deleted cache from travis settings.
2. Increased build parallelism for recipe.

This caused the build to pass within 14 minutes 29 seconds. Whereas the previous build errored at 49 minutes and 58 seconds.

Student Information

Bio of student

I am Ashwin Samudre, an undergraduate third-year student at Pune Institute of Computer Technology, Pune, pursuing B.E. in Computer Engineering.

I recently completed my work in Season of KDE open source development program for Kdenlive. Project report -
https://github.com/kiryteo/kiryteo.github.io/blob/master/SoK-report.pdf

I am currently working as an intern for OpenEBS/scope open source project at Mayadata.

I have participated in other open source programmes like Kharagpur Winter of Code 2017 conducted by IIT-Kharagpur, NJACK Winter of Code 2017 conducted by IIT-Patna, Opencode 2018 conducted by IIIT-Allahabad.

I have developed an expense management android application. I have worked on MPI, OpenMP, BLAS libraries for dense matrix computations as a trainee in Centre for Development of Advanced Computing, Pune, India.

Contact Information

Name :	Ashwin Samudre
Phone :	+918605889371
Emails :	capnoi8@gmail.com
	ashwin.samudre@gmail.com
Telegram :	@ashwins13
Twitter :	@samudre_ashwin
Github :	@kiryteo