art and *LArSoft* course schedule

August 3-7, 2015

Draft version 8

Contents

---

1 Introduction

This is a **draft** schedule for the *art*/*LArSoft* course for the summer of 2015. The course is scheduled to take place August 3-7, 2015, at Fermilab.

The goal of the course is to take physicists with basic C++ skills and start them on the path to be able to:

1. develop analysis, simulation, and reconstruction code of production quality in the *art* framework environment,
2. develop algorithms of sufficient quality to be able to extend the shared *LArSoft* and *art* products.

The course is aimed at relative newcomers to *art*, but not for complete newcomers to C++. A statement of prerequisites, along with some suggested references, is available at https://cdcvs.fnal.gov/redmine/projects/art-larsoft-course/wiki/Prerequisites. Much of the course is based on the **art Workbook**, available at https://web.fnal.gov/project/ArtDoc/Pages/workbook.aspx.

This document provides a brief introduction to the subject matter for each of the capsules in the course. Most capsules include a brief introductory talk and a longer period in which students work on exercises.

The final day is specific to *LArSoft*, and is intended for relative newcomers to *LArSoft*.

2 Monday

The goal of Monday morning is to get students up to speed on some of the critical parts of C++ upon which we will rely for the entire course. We will introduce some of the basics of good coding practice. Some registrants for the course might not need this material. The material is designed so that such students can skip this session. This first morning is **NOT** intended as an introduction to C++ for those who do not meet the prerequisites described at the web page above.

The goal of Monday afternoon is to introduce people to the parts of the framework, and to the environment in which framework programs are run.

2.1 Session 1: Basics of C++

1 hour: 20 minutes talk, 30 minutes working, 10 minutes wrap-up.

The meaning of pointers and references. Function calling and argument passing (by reference and value); function return values. Compiling and linking; creating dynamic libraries. Understanding the difference between compilation and link errors.

2.2 Session 2: Basics of objects

1 hour: 20 minutes talk, 30 minutes working, 10 minutes wrap-up.

Object lifetimes. Controlling resources (e.g. memory) using object lifetimes (RAII). Use of *shared_ptr* and *unique_ptr*. Avoiding use of *new* and *delete*; avoiding use of *static*.

2.3 Session 3: Basic data structures

1 hour: 20 minutes talk, 30 minutes working, 10 minutes wrap-up.

Introduction to correct use of *std* library components *array*, *vector*, *map*, and *unordered_map*. Performance characteristics; when to use each. Relying on move constructor for efficient use. Correct initialization.

2.4 Session 4: Framework introduction

30 minutes, talk.

Overview of the major features of the framework; what the framework does for you. How your code gets integrated into a framework program.

2.5 Session 5: Setup for using *art*

1.5 hours: 20 minutes talk, 45 minutes working, 25 minutes wrap-up.

Based on workbook exercise 1. Introduces the *art* runtime system. Basic understanding of UPS products and the *setup* command. Basic execution of the *art* executable. Creating output files. Basic introduction to FHiCL configuration file for *art*.

2.6 Session 6: Setting up for development of experiment code

2 hours: 20 minutes talk, 90 minutes work, 10 minutes wrap-up.

Based on workbook exercise 2; may update to use use *mrb* rather than *cetbuildtools* directly.

Goes all the way from cloning a *git* repository with source code, building what has been cloned, up to looking at a module, running *artmod* to create a module.

Exercises include:

1. Fixing a canned compilation failure
2. Fixing a canned link failure
3. Fixing a failure to find a module

3 Tuesday

The goal of Tuesday is to familiarize people with the development environment, in the context of starting to analyze data (i.e. making histograms) in the *art* setting.

3.1 Session 7: More module interface

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Introduce re-establishing a development environment after logging out of a previous shell session. Introduce the full *EDAnalyzer* interface; introduction to *Run* and *SubRun* objects (but not as containers of products yet). Understanding the module lifecycle. Introduce *Tracer* service.

Based on workbook exercise 3.

3.2 Session 8: Details of module configuration

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

How to correctly write a module constructor. How to use *ParameterSet* objects. How to handle errors in constructors. Revisit RAII here, and use of compiler-generated member functions when possible. Revisit *override* keyword. Introduce *art::Exception* class.

Based on workbook exercise 4.

3.3 Session 9: Multiple instances of a module

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Understanding module instances, and the concept of having more than one instance of the same module class in a workflow. Understanding the ordering guarantee (after producers and filters, but not ordered relative to each other) provided by *art* for analyzers (and output).

Based on workbook exercise 5.

3.4 Session 10: Using existing data products

30 minutes: 20 minutes talk, 10 minutes questions.

How to find data products, how to read headers to understand data products. Introduction to some good data product design practices.

3.5 Session 11: Making histograms.

2 hours: 20 minutes talk, 90 minutes work, 10 minutes questions.

Accessing event data products; introduction of details of module label and module type. Using *ValidHandle*; *Handle* is for special cases only. Simple configuration

of analyzer modules; understanding module execution order guarantees provided by *art*.

Using data product classes, iterating through sequences, filling histograms. Introduction to *TFileService*: how to configure, where it writes output.

Using Standard Library algorithms and lambda expressions to loop over data products. Reasons to prefer SL algorithms to explicit loops; avoiding fencepost errors, avoiding trivial inefficiencies, avoiding needless copies and conversions.

Combining examples 6 and 7 of the workbook, possibly modified to use *LArSoft* data products.

3.6 Session 12: Running multiple modules

1.5 hours: 20 minutes talk, 60 minutes work, 10 minutes questions.

Run a chain of reconstruction algorithms. Write an analyzer that compares data products from two different algorithms. Practice using *art::InputTag* to identify products; practice seeing how the configuration file determines how products are labeled. Introduction to paths, and the order in which producers are executed. How the framework avoids running the same configured module more than once.

4 Wednesday

The goal of Wednesday is to introduce people to writing producers, using good software development practices: iterative code development, writing modular code, and testing. We also introduce the creation of new data products.

4.1 Session 13: Creating a *Producer*

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Canonical form of a producer (get/do/put pattern). How *Event::put* works; how data products are labeled. How the FHiCL file controls each producer's module label and the process name. How to create an instance of a data product.

4.2 Session 14: Inventing a new data product

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Design guidelines for new data products following the *art* data product design guide.

Producing dictionaries for ROOT. We are currently using ROOT5. We may be in the process of moving to ROOT 6 around the time of the class. This may give us some problems to solve.

4.3 Session 15: Controlling output

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Introduce the output system's ability to do *event selection* and *product selection*. Detailed configuration of *RootOutput*. Keep/drop lists. Configuration and meaning of *drop on input*.

Controlling compression levels. Writing multiple output files.

4.4 Session 16: Introducing iterative algorithm development.

30 minutes: 20 minutes talk, 10 minutes questions.

Introduce the ideas behind iterative development of algorithms, and designing code for testability.

4.5 Session 17: Writing a new algorithm.

2 hours: 20 minutes talk, 1.5 hours work, 10 minutes wrap-up.

Implement in code an algorithm, suitable for coding in a single module, specified as part of the problem. Write unit tests for the algorithm. Use the build tools to build and run the tests. Illustrate how to work on the algorithm outside of the framework. Illustrate a clear separation between *unit testing* and *physics validation*.

4.6 Session 18: Using the algorithm in a producer

30 minutes: 5 minutes talk, 20 minutes work, 5 minutes wrap-up.

Use *artmod* to generate the skeleton of a producer. Insert the algorithm written in the previous session into the producer. Write the integration test that executes the producer.

5 Thursday

The goal of Thursday is to gain more experience in modularity and to introduce the use of some software development tools: a debugger and *Valgrind*.

This includes the development of a set of algorithms (only need two) that need to be split up between modules. This extends the previous day's work. Introduce people to some of the more advanced framework tools (association collections, *FindOne* and *FindMany* smart query objects, services), as well as the *art* command line. It also include some debugging exercises.

5.1 Session 19: Some additional *art* facilities

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Explanation of full set of *art* command-line options. Use of configuration dumping facilities. Introduction to the most important of the standard services: *TimeTracker*, *MemoryTracker*, and *RandomNumberGenerator*.

Examples will not include writing new code, but in using the *TimeTracker*, *MemoryTracker* and *Tracer* services with existing code.

5.2 Session 20: Using *Assns* and smart query objects

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Use a canned module that creates some product from the output of Wednesday's producer, and *Assns* between that product and Wednesday's product.

Write an analyzer that uses *FindOne* or *FindMany* to obtain information to fill histograms. This doesn't directly require that users see the *Assns* object.

5.3 Session 21: Creating *Assns*

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Write a producer that does something like what the "canned producer" above does. Understand how to create *Assns* objects.

5.4 Session 22: Good *art* workflow

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Using advanced features of FHiCL, and recommended practice for creating maintainable configuration files. Using FHiCL *PROLOG*, *@sequence*, *@table*, replacement syntax, @nil, and @erase.

Understanding how to organize paths, understanding how *art* deals with the same module label appearing in multiple paths.

This section will not have students writing C++ code; they will be using provided modules to create workflows using the various features of FHiCL.

5.5 Session 23: Debugging

1.5 hour: 10 minutes talk, 70 minutes work, 10 minutes wrap-up.

Introduction to using a debugger to understand code and to isolate a problem in code. In the work session, students are presented with code examples containing failures, and they need to find the failures.

5.6 Session 24: More debugging

1.5 hour: 10 minutes talk, 70 minutes work, 10 minutes wrap-up.

Introduction to using *Valgrind* to help find problems. Interpreting *Valgrind* output, using suppression files. In the work session, students are presented with code examples containing failures, and they need to find the failures.

6 Friday

Friday's schedule is specific to *LArSoft*.

The goal on Friday is to give an introduction into the design philosophy and content of the *LArSoft* products, and providing hands-on work sessions using it.

6.1 Session 25: Introduction to *LArSoft*

45 min: 35 minutes talk, 10 minutes work

This session will cover what is *LArSoft* and its specific data structures; we will describe the structure and working of a LArTPC detector, and translation of physics observables to data structures. Students will learn how to run a simple *LArSoft* example.

6.2 Session 26: Introduction to *LArSoft* code and work environment

30 min: 30 minutes talk

This session will describe *LArSoft* code organization and different repositories. The lecture will briefly cover mrb, cmake, gitflow, etc. Students will learn how to get/build/test *LArSoft* code using mrb.

6.3 Session 27: How to work with *LArSoft*

1 hr 15 min: 10 min talk, 55 min work, 10 min wrap-up

In this session, students will learn how to make changes in the code, and how to contribute new code. Students will follow instructions to work on an analysis task in *LArSoft* environment.

Hands-on working session: Setup the build environment. Checkout the analysis example code, modify, build, run tests.

6.4 Session 28: *LArSoft* Algorithms and Services

2 hr: 30 minutes talk, 80 minutes work, 10 minutes wrap-up.

In this session, we will discuss the design principles of *LArSoft*.
We will demonstrate application of *LArSoft* design principles to actual code. we will briefly describe *LArSoft* simulation, reconstruction and analysis algorithms. We will introduce the services available in *LArSoft* focusing on the services needed to complete the exercise.

Exercise to develop a simple new algorithm and module within *LArSoft*, build and run it.

6.5 Session 29: Using *LArSoft* for detector simulation and event generation

1 hr 30 min: 15 minutes talk, 60 minutes work, 15 minutes wrap-up.

How to configure/describe detector to *LArSoft*, how to change existing detector specific geometries and response functions. How do you make use of G4, GENIE in *LArSoft*? How to use event display in *LArSoft*?

Wrap up and concluding presentation: *Current directions and ideas for the future.*