Outline for the *art* Users Guide

Rob Kutschke
June 2, 2016

Abstract:

This document is the working outline for the the Users Guide section of the *art* documentation suite. The Users Guide is Part III of the current large document. The chapter numbers in this document don't match the chapter numbers in the current large document. At this writing the outline in the large document is just a placeholder - this document is the working version of the outline.

The reader of this document is presumed to be familiar with the material in Parts I and II of the *art* documentation suite, the Introduction and the Workbook, respectively.

Contents

1 Overview

In this chapter we need to touch on a lot of ideas so that each of the following chapters can be reasonably standalone. Some of the most elementary material will be the same as in the Intro - maybe we want to structure the document that way, with
input ?

1. What is *art*?
2. Driving the Event loop
3. How does *art* see your experiment's code: Modules and Services
4. The Event Model
5. Input and Output
6. Scheduled reconstruction
7. Reconstruction on Demand
8. TFileService
9. Event Mixing - loss of provenance.
10. Random Number support
11. Interface with Geant4
12. Schedules
13. Concurrency
14. Interface with SAM
15. Philosophy: Modules may only communicate via the Event. Part of the mission of *art* is to document provenance.

2 Development and RunTime Environments

1. List of external products on which *art* depends. Short description of each and references to docs for them.
2. Describe UPS - hopefully can ref the new manual. Maybe ref the material in the Intro.
3. How to deal with build systems? Use system in the toyExperiment for definiteness?

3 User Response points in the Event Loop

1. This describes the full list of user callable points for both modules and services.

4 Run Time Configuration

1. Refer to the appendix for the FHCIL grammar guide.
2. Describe *art*'s view of a FHICL file; what words, in which scopes, have special meaning to *art*.x
3. Where does FHICL_FILE_PATH go - here or in the appendix?
4. For modules and services: relationship between class names, file name of the .so and the names that appear in the .fcl file.
5. Colission detection in the names of .so files for modules and services.
6. How to write a specialization of get_if_present¡T¿ for your own class T.

5 Modules

1. Overview of the 5 types of modules
2. What makes a class a module
3. Analyzer modules
4. Producer modules
5. Filter modules
6. Input modules
7. Output modules
8. The MACRO DEFINE_ART_MODULE
    (a) Factory Function
    (b) Registry and interaction with the registry
    (c) Violation of the ODR

6 Services

1. Writing your own Service
2. Legacy Services
3. Global Services
4. Schedule-Local Services
5. Service Interfaces
6. Internally *art* uses services to do some of its work. So of these have a default config and you do not need to supply a config in the FHiCL file. You may optionally provide a config - example is the scheduler.
7. *art* also supplies some services that are only turned on if a parameter set is provided in the FHiCL file. Examples: Tracer, Timer, SimpleMemoryCheck
8. When should you get your service handle? c'tor, each member function ...
9. Check Anne's notes from her discussion with Chris for more material.

7 Cartoon of art-Flow

1. Open FHCIL file
2. Create intermediate table
3. Apply Command line arguments
4. Create Parameter Sets in the Registry
5. Instantiate *art* internal services
6. Instantiate user services
7. Scan LD_LIBRARY_PATH - find all .so files.
8. Open and process and _dict.so and _map.so files
9. Parse the paths to figure out the order in which to instantiate modules.
10. Instantiate modules
11. Drive the event loop.
12. The concept of a schedule
13. trigger_path: skip to next module, skip to next trigger_path, next event
14. Shutdown as safely as possible

8 Event Data Model

1. Event/Run/SubRun
2. EventID
   (a) You must ensure the uniqueness of event IDs.
   (b) This has consequences on how to structure our MC jobs.
3. Transient
4. Persistent
5. Orthogonality of Transient and Peristent
6. art::ProductID
7. Provenance
8. Consistencey of input files - needs consistency of provenance as well as shape consistency
9. Merging two input files into one

9 Data Products

1. Focus on transient rep.
2. Refer to the chapter on ROOT IO for properties of persistent data products.
3. Facade Pattern

10 InterProduct References

1. art::Ptr
2. art::PtrVector
3. art::Assns
4. Discussion of when to prefer Assns or Ptr

11 RootInput and RootOutput

1. SelectEvents based on trigger path - is this in the RootOutput or in one of the base classes?
2. Drop on input and drop on output
3. fast cloning
4. art::Wrapper
5. Schema: classes.h and classes_def.xml
6. Automagic schema evolution
7. User written streamers
8. Include the guide to creating data products from the *art* wiki page.
9. Structure of an *art* root file
10. Size on disk of a data product.
11. Tips on how to read one of these files using native root (you loose inter-prodcut references but all else should be OK).
12. genreflex info should not be duplicated - structure your libraries accordingly.
13. *art* supplies the genreflex info for some basic data types. These are used for testing.

12 Exeption Handling

1. Orthogonality of throw and response.
2. Response is run time configurable (at the category level?? at other levels?)
3. Strong recommendation not to use exceptions for normal flow control. Use it to signal true errors. Except if an underlying package uses exceptions for flow control and you are forced to when using that package. Normally let *art* catch exceptions.
4. fpe
5. Do use exceptipon specifications and why.
6. `cet::exception` and `art::exception`
7. Defining exception (and log?) categories for your own experiment.
8. assert, cassert vs throw. Debugging stuff should compile to nothing with production code.

13 Message Logger

1. Crib CMS docs except for run time config; this may be a good stand-alone project for Anne?
2. Run time config
3. is there an equivalent of art::exception? I don't think so.

14 Art Supplied Services

1. TFileService - refer to its own full chapter.
2. Tracer
3. Timing Service
4. Simple Memory Check
5. RandomNumber Service - refer to its own full chapter.
6. Floating point control ?

15 Standards and Practices

1. Describe standards and practices used within *art*; this includes use of C++, use of the tool chain and interactions of pieces of *art* with other pieces of *art*.
2. We probably want a "short version" and a "long version" and ask that everyone be familiar with the short version.
3. Suggest standards and practices that might be adopted by the experiments.
4. "Coding standards" - types start upcase etc
5. Exception Saftey
6. Thread Saftey
7. Looking ahead towards concurrency
8. If it's bigger than a pointer, return by appropriate (safe) pointer type
9. What is the appropriate (safe) pointer type? Sometimes it's a const &.
10. About inlining.
    (a) it is always just a hint, never a firm directive
    (b) automatic if definition is together with declaration in the class declaration
    (c) if out of class declaration preceded by inline keyword.
    (d) always inline trivial accessors
    (e) never inline anything static
    (f) Guidelines for how to think about other cases.

16 Profiling your Code

17 Debugging and Troubleshooting

A Rules for names

1. Reserved identifiers in art .fcl files
2. Must not contain an underscore: process name, module label, instance name

B Code Guards

Describe what they are and why to use them.

C CLHEP

1. You are likely to encounter Vector, Random Engine, SystemOfUnits
2. CMS reports that we should prefer ROOT TMatrix to CLHEP Matrix classes.
3. For Vector, headers are the documentation.
4. There is a writeup for Random
5. .icc convention - no longer recommended but you need to know it.
6. Some things are nasty have a nasty: [] use 0 based indexing but () use 1-based indexing!
7. Hep3Vector and HepLorentzVector do not distinguish between a position, a displacent, a momentum and a velocity.
8. Proper use of SystemOfUnits - repeated multiplying is wrong!
   (a) double a = 123. * CLHEP::mm
   says that the numeric constant 123 is in mm and asks the code to convert it to the internal unit of length. It does NOT assert that a is in mm.
   (b) double b = a / CLHEP::mm
   If a is in CLHEP internal length units, then b will be in mm.
   (c) Never do: using CLHEP or using CLHEP::m; This is just asking for trouble.
   (d) Geant4 internally uses Hep3Vector, SystemOfUnits and HepRandom.

D C++ Ideas

We anticipate that there will be many places in the code that want to refer to certain C++ ideas. For example we will want discussions about inheritance, templates, exceptions, factory methods and other things. These are all examples of things that most people only need to know a little about and the standard texts have much more material than is needed. For each of these we need a superficial general discussion followed by an *art*-specific detailed discussion. We can put those pieces here and refer to them as needed.

Some of these may also be mentioned in Standards and Practices. I envisage the technical discussion here so that the Stanards and Practices discussion can presume that people know the technical bits already.

1. Inheritance
2. Templates
3. Exceptions
4. We need a section on when const in different positions means the same thing and when const in different positions means different things. See table.

The only real weirdness is that “const T” and “T const” mean exactly the same thing. When pointers are involved there are two notions of const-ness, the const-ness of the pointer and the const-ness of the pointee; all permutations are possible.

Declaration	Meaning
T t;	t is an object of type T
const T t;	t is an object of type const T
T const t;	same meaning as previous line
const T& t;	t is a const reference to an object of type T
T const& t;	same meaning as the previous line
T * t;	t is a pointer to an object of type T
T const * t;	t is a pointer to an object of type T
const T * t;	same meaning as previous line
T * const t;	t is a const pointer to a object of type T
T const * const t;	t is a const pointer to a object of type const T
const T * const t;	same meaning as the previous line

Table D.1: Meaning of const in the declaration of an object.

E CETLIB

1. Lots of stuff. Point to docs. The existing docs are a table of one-line descriptions plus what you find in the header code. Point out the string manips, like pad, trim, split,
2. Longer docs
3. `maybe_ref<T>` a simple handle with reference syntax
4. `map_vector<T>` should have been called `sparse_vector<T>`. Why not just use std::map<integral_type,T>?
   (a) no T const& operator[](integral_type) const;
   (b) read performance in ROOT IO.
5. `pow<T>`
6. `exempt_ptr<T>` Always a shallow copy.
7. `value_ptr<T>` Always a deep copy
8. `cet::exception`

F FHICL - grammar

G Enum-Matched-To-String

H Finite Precision Arithmetic and Floating Point Arithmetic

1. Dynamic range of int, long, double, float
2. Concept of machine epsilon for floating point types
3. Danger of using == for comparison of floating point types.
4. Danger of underflow if (e > p) m =sqrt(e*e-p*p). Instead: arg =(e-p)*(e+p); m = (arg >0) ? sqrt(arg): 0.; safesqrt. Much less likely to underflow. Or even just: m = sqrt((e-p)*(e+p)).
5. cet::sum_of_squares and cet::diff_of_squares.
6. NaN, signaling and non-signaling.
7. Care in using a floating point type as a loop index.
8. Rounding from float to int: ceil, floor.
9. don't do pow(x,n) when you really mean x*x. We have sqr(x) and pow¡n¿(x). Does it make sense to ask cetlib to provide by and inline powi¡n¿(x) and an non-inline?

I Anonymous Namespaces

What is an anonymous namespace, how does it work and when should I use one? When should I put a function in an anonymous namespace and when should it put it in a utility library.

J Dynamic Libraries

Discuss what a dynamic library file is. How does it differ from static library? How does one load them? Maybe point to my toy example?

K Checklist

This is a brainstorming list of topics for which I have not yet found a home.

1. Suppose that you write events to an *art* root file out of order. When you read it back, they will be in order! This is because the file contains a table that contains event ids with pointers to the data for that event;the table is sorted by event id.
2. If you are reading multiple input files you may encounter the same Run/-SubRun multiple times. By default *art* holds its Run and subRun objects in memory so that it is safe to encounter the same Run/SubRun multiple times. In a sparse skim this can be a memory pig. There is a switch to turn off this caching. What is it's name?
3. The if-combo-container has chapters on "Obtaining Credentials to Access Fermilab Computing Resources" and on "git". These don't really belong in the User's Guide but we should find a home for them.
4. Decide on "module type" vs "module_type" vs "module class name" and use it everywhere.
5. In the existing docs: scrub "module name" - it should be "module label" or "module type"
6. Where can we put http://mu2e.fnal.gov/atwork/computing/FrameworkIntro.shtml which has a list of C++ stuff you need to know - maybe not on day one but within the first few weeks.
7. What is a linkage loop. How do I diagnose it? How do I fix it? Chris Green's tool.
8. Is there a syntax to tell art to run on just one fully qualified eventId? On a list of them? On a range? On a list of ranges? These were in cmsrun.
9. Document the new TimeTracker and MemoryTracker Services and how to access the new databases. Provide code samples to do simple things and a link to SQL documentation.