

New features in *art*

Kyle J. Knoepfel

Fermi National Accelerator Laboratory

February 10, 2016

art

- Developers can be reached at artists@fnal.gov.
- *art* is used by 9 experiments.
- Our development efforts are guided by:
 - Current and future experiment needs, as reported on the issue tracker or at stakeholder meetings.
 - Current and future software and hardware technology, in and outside of HEP.
 - Constraints imposed by dependent packages.
 - Feedback from individual users, and our own estimation of what features would make *art* simpler to use.

art/LArSoft course

- Last summer, we had a very successful *art*/LArSoft course:

<https://indico.fnal.gov/conferenceDisplay.py?confId=9928>

- Lot of material:
 - Basic *art* stuff
 - Creating products/dictionaries
 - Suggestions on workflow
 - Debugging tips
 - Improving your coding
 - etc.
- No course this year due to financial constraints.

General improvements to *art*

- Documentation (never-ending task)
- More informative diagnostic messages
 - Parsing problems with configuration file
 - Incorrectly specified parameters for a module
 - Source of final value for a configuration parameter
 - Detecting inconsistent input files
- Tighter restrictions on placing products onto the Event.
- Relaxed requirements on product declarations.
- Can process multiple output files if the first file has no events in it.

art from 2015-Present

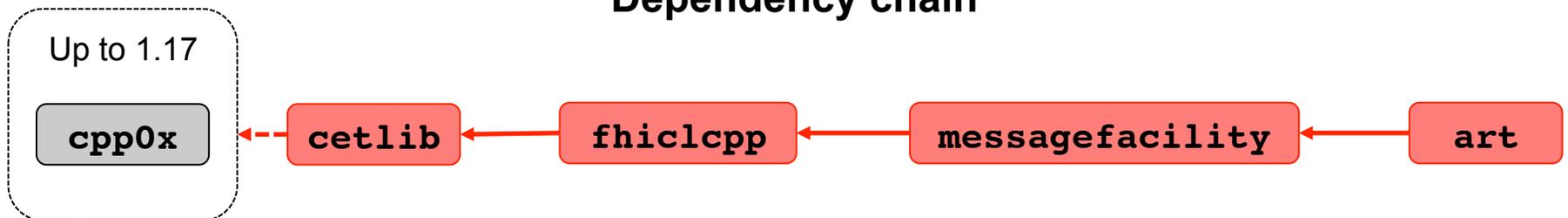
ROOT 5 ROOT 6

1.13	1.14	1.15	1.16	1.17	1.18
1.13.00	1.14.00	1.15.00	1.16.00	1.17.00	1.18.00
1.13.01	1.14.01	1.15.01	1.16.01	1.17.01	1.18.01
1.13.02	1.14.02	1.15.02	1.16.02	1.17.02	1.18.02
	1.14.03			1.17.03	
				1.17.04	
				1.17.05	1.18.03
				1.17.06	1.18.04
				1.17.07	1.18.05

art from 2015-Present

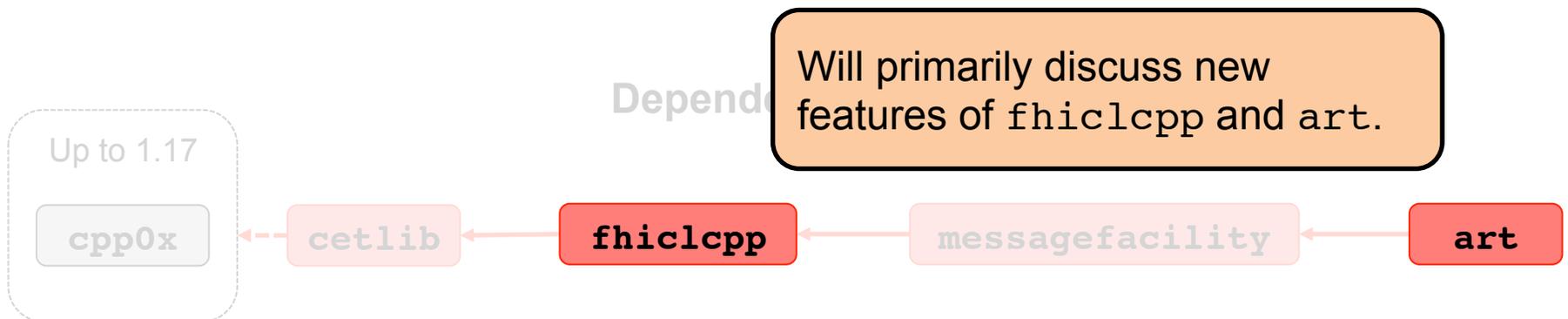
ROOT 5					ROOT 6
1.13	1.14	1.15	1.16	1.17	1.18
1.13.00	1.14.00	1.15.00	1.16.00	1.17.00	1.18.00
1.13.01	1.14.01	1.15.01	1.16.01	1.17.01	1.18.01
1.13.02	1.14.02	1.15.02	1.16.02	1.17.02	1.18.02
	1.14.03			1.17.03	
				1.17.04	
				1.17.05	1.18.03
				1.17.06	1.18.04
				1.17.07	1.18.05

Dependency chain



art from 2015-Present

ROOT 5					ROOT 6
1.13	1.14	1.15	1.16	1.17	1.18
1.13.00	1.14.00	1.15.00	1.16.00	1.17.00	1.18.00
1.13.01	1.14.01	1.15.01	1.16.01	1.17.01	1.18.01
1.13.02	1.14.02	1.15.02	1.16.02	1.17.02	1.18.02
	1.14.03			1.17.03	
				1.17.04	
				1.17.05	1.18.03
				1.17.06	1.18.04
				1.17.07	1.18.05



FHiCL and fhiclcpp

- The current configuration language is FHiCL 3:
 - See the quick-start guide at <https://cdcv.s.fnal.gov/redmine/projects/fhicl/wiki>
 - Recommendations for best practices – see session 22 from the *art/LArSoft* course.
- `fhiclcpp` is the C++ *binding* which provides a C++ interface to a parsed FHiCL document.

New features in FHiCL

- Main goal: reduce points of maintenance
- Splicing utilities (already in widespread use)
- **@sequence::**

```
BEGIN_PROLOG
numbers: {
  list: [1,2,3]
}
END_PROLOG
fullList: [@sequence::numbers.list, 4,5,6]
```

```
# Parsed FHiCL document
fullList: [1,2,3,4,5,6]
```

New features in FHiCL

- Main goal: reduce points of maintenance
- Splicing utilities (already in widespread use)
- **@table::**

```
BEGIN_PROLOG
experiment: {
  producers: {
    p1: {}
    p2: {}
    p3: {}
  }
}
END_PROLOG

myProducers: {
  @table::experiment.producers
}
```

```
# Parsed FHiCL document

myProducers: {
  p1: {}
  p2: {}
  p3: {}
}
```

New features in FHiCL

- **Modified binding operators**
 - Introduces protection levels so that subsequent attempts to override a value are either ignored or trigger an exception.

New features in FHiCL

- **Modified binding operators**
 - Introduces protection levels so that subsequent attempts to override a value are either ignored or trigger an exception.

```
#initial declaration  
  
fileNames @protect_ignore: ["a.root", "b.root"]  
  
# a bunch of configuration code ...  
# and then:  
  
fileNames : ["c.root", "d.root"] // silently ignored
```

New features in FHiCL

- **Modified binding operators**
 - Introduces protection levels so that subsequent attempts to override a value are either ignored or trigger an exception.

```
#initial declaration  
  
fileNames @protect_ignore: ["a.root", "b.root"]  
  
# a bunch of configuration code ...  
# and then:  
  
fileNames : ["c.root", "d.root"] // silently ignored
```

```
#initial declaration  
  
fileNames @protect_error: ["a.root", "b.root"]  
  
# a bunch of configuration code ...  
# and then:  
  
fileNames : ["c.root", "d.root"] // error - exception thrown
```

New features in fhiclcpp

- **Extended ParameterSet retrieval.**

- Supports more std containers

```
pset.get<std::tuple<Type1, Type2, Type3>> ("tuple");  
pset.get<std::pair <Type1, Type2>>      ("pair" );  
pset.get<std::array<Type1, 4u>>        ("array");
```

- Supports CLHEP objects

```
pset.get<CLHEP::Hep2Vector      > (...);  
pset.get<CLHEP::Hep3Vector      > (...);  
pset.get<CLHEP::HepLorentzVector> (...);
```

New features in fhiclcpp

- **Extended ParameterSet retrieval.**
 - Can specify sequence elements.
- Want Higgs mass from right.

```
pset: {  
    particles: [  
        {  
            name: Higgs  
            mass: 125  
        },  
        {  
            name: photon  
            mass: 0  
        }  
    ]  
}
```

New features in fhiclcpp

- **Extended ParameterSet retrieval.**
 - Can specify sequence elements.
- Want Higgs mass from right.

```
pset: {  
    particles: [  
        {  
            name: Higgs  
            mass: 125  
        },  
        {  
            name: photon  
            mass: 0  
        }  
    ]  
}
```

Before art 1.16

```
auto const& particles = pset.get<vector<ParameterSet>>("particles");  
auto const higgsMass = particles.at(0).get<double>("mass");
```

New features in fhiclcpp

- **Extended ParameterSet retrieval.**
 - Can specify sequence elements.
- Want Higgs mass from right.

```
pset: {  
    particles: [  
        {  
            name: Higgs  
            mass: 125  
        },  
        {  
            name: photon  
            mass: 0  
        }  
    ]  
}
```

Before art 1.16

```
auto const& particles = pset.get<vector<ParameterSet>>("particles");  
auto const higgsMass = particles.at(0).get<double>("mass");
```

art 1.16 -

```
auto const higgsMass = pset.get<double>("particles[0].mass");
```

New features in `fhiiclcpp`

- Helper utility: **`fhiicl-dump`**
 - Tool that prints out the fully processed FHiCL document.

New features in `fhiclcpp`

- Helper utility: **`fhicl-dump`**
 - Tool that prints out the fully processed FHiCL document.

```
[knoepfel@woof sample_code]$ fhicl-dump sample.fcl
# Produced from 'fhicl-dump' using:
#   Input   : sample.fcl
#   Policy  : cet::filepath_lookup
#   Path    : "FHICL_FILE_PATH"

pset: {
  particles: [
    {
      mass: 125
      name: "Higgs"
    },
    {
      mass: 0
      name: "photon"
    }
  ]
}
```

New features in fhiclcpp

- Helper utility: **fhicl-dump**
 - Tool that prints out the fully processed FHiCL document.

```
[knoepfel@woof sample_code]$ fhicl-dump sample.fcl
# Produced from 'fhicl-dump' using:
#   Input   : sample.fcl
#   Policy  : cet::file
#   Path    : "FHICL_FILE_PATH"

pset: {
  particles: [
    {
      mass: 125
      name: "Higgs"
    },
    {
      mass: 0
      name: "photon"
    }
  ]
}
```

```
[knoepfel@woof sample_code]$ fhicl-dump --annotated sample.fcl
# Produced from 'fhicl-dump' using:
#   Input   : sample.fcl
#   Policy  : cet::filepath_lookup
#   Path    : "FHICL_FILE_PATH"

pset: { # ./sample.fcl:1
  particles: [ # ./sample.fcl:3
    { # ./sample.fcl:4
      mass: 125 # ./sample.fcl:6
      name: "Higgs" # ./sample.fcl:5
    },
    { # ./sample.fcl:8
      mass: 0 # ./sample.fcl:10
      name: "photon" # ./sample.fcl:9
    }
  ]
}
```

New features in fhiclcpp

- Helper utility: **fhicl-dump**
 - Tool that prints out the fully processed FHiCL document.

```
[knoepfel@woof sample_code]$ fhicl-dump --help
fhicl-dump [-c] <file>
Options:
  -h [ --help ]                produce this help message
  -c [ --config ] arg         input file
  -o [ --output ] arg         output file (default is STDOUT)
  -a [ --annotated ]          include source location annotations
  --prefix-annotated          include source location annotations on
                              line preceding parameter assignment
                              (mutually exclusive with 'annotated'
                              option)
  -q [ --quiet ]              suppress output to STDOUT
  -l [ --lookup-policy ] arg (=1) lookup policy code:
                                0 => cet::filepath_maker
                                1 => cet::filepath_lookup
                                2 => cet::filepath_lookup_nonabsolute
                                3 => cet::filepath_lookup_after1
  -p [ --path ] arg (=FHICL_FILE_PATH) path or environment variable to be used
                                      by lookup-policy
```

New features in art

- Command-line options
- Product access through secondary files
- Results products
- Configuration validation & description
- ROOT6

art --help

Allowed options:

-c [--config] arg

-h [--help]

--process-name arg

--print-available arg

Configuration file.

produce help message

art process name.

List all available plugins with the provided suffix. Choose from:

'module'

'plugin'

'service'

'source'

--print-available-modules

List all available modules that can be invoked in a FHiCL file.

--print-available-services

List all available services that can be invoked in a FHiCL file.

--print-description arg

Print description of specified module, service, source, or other plugin (multiple OK).

art --print-available-modules

module_type	Type	Source location
AddIntsProducer	producer	/home/knoepfel/art/test/Integration/AddIntsProducer_module.cc
AssnsAnalyzer	analyzer	/home/knoepfel/art/test/Integration/AssnsAnalyzer_module.cc
AssnsProducer	producer	/home/knoepfel/art/test/Integration/AssnsProducer_module.cc
BareStringAnalyzer	analyzer	/home/knoepfel/art/test/Integration/BareStringAnalyzer_module.cc
BareStringProducer	producer	/home/knoepfel/art/test/Integration/BareStringProducer_module.cc
BlockingPrescaler	filter	/home/knoepfel/art/art/Framework/Modules/BlockingPrescaler_module.cc
CompressedIntProducer	producer	/home/knoepfel/art/test/Integration/CompressedIntProducer_module.cc
CompressedIntTestAnalyzer	analyzer	/home/knoepfel/art/test/Integration/CompressedIntTestAnalyzer_module.cc
DeferredIsReadyWithAssnsAnalyzer	analyzer	/home/knoepfel/art/test/Integration/DeferredIsReadyWithAssnsAnalyzer_module.cc
DeferredIsReadyWithAssnsProducer	producer	/home/knoepfel/art/test/Integration/DeferredIsReadyWithAssnsProducer_module.cc
DerivedPtrVectorProducer	producer	/home/knoepfel/art/test/Integration/DerivedPtrVectorProducer_module.cc
DoubleProducer	producer	/home/knoepfel/art/test/Integration/DoubleProducer_module.cc
FlushingGeneratorTestFilter	filter	/home/knoepfel/art/test/Integration/FlushingGeneratorTestFilter_module.cc
HMRunProdProducer	producer	/home/knoepfel/art/test/Integration/high-memory/HMRunProdProducer_module.cc
HMSubRunProdProducer	producer	/home/knoepfel/art/test/Integration/high-memory/HMSubRunProdProducer_module.cc
InputProducer	producer	/home/knoepfel/art/test/Integration/event-shape/InputProducer_module.cc
***InputProducerNoEvents	producer	/home/knoepfel/art/test/Integration/event-shape/InputProducerNoEvents_module.cc
***InputProducerNoEvents	producer	/home/knoepfel/art/test/Integration/run-subrun-shape/InputProducerNoEvents_module.cc

art --print-available-modules

module_type	Type	Source location
AddIntsProducer	producer	/home/knoepfel/art/test/Integration/AddIntsProducer_module.cc
AssnsAnalyzer	analyzer	/home/knoepfel/art/test/Integration/AssnsAnalyzer_module.cc
AssnsProducer	producer	/home/knoepfel/art/test/Integration/AssnsProducer_module.cc
BareStringAnalyzer	analyzer	/home/knoepfel/art/test/Integration/BareStringAnalyzer_module.cc
BareStringProducer	producer	/home/knoepfel/art/test/Integration/BareStringProducer_module.cc
BlockingPrescaler	filter	/home/knoepfel/art/art/Framework/Modules/BlockingPrescaler_module.cc
CompressedIntProducer	producer	/home/knoepfel/art/test/Integration/CompressedIntProducer_module.cc
CompressedIntTestAnalyzer	analyzer	/home/knoepfel/art/test/Integration/CompressedIntTestAnalyzer_module.cc
FlushingGeneratorTestFilter	filter	/home/knoepfel/art/test/Integration/FlushingGeneratorTestFilter_module.cc
HMRunProdProducer	producer	/home/knoepfel/art/test/Integration/HMRunProdProducer_module.cc
HMSubRunProdProducer	producer	/home/knoepfel/art/test/Integration/HMSubRunProdProducer_module.cc
InputProducer	producer	/home/knoepfel/art/test/Integration/event-shape/InputProducer_module.cc
***InputProducerNoEvents	producer	/home/knoepfel/art/test/Integration/event-shape/InputProducerNoEvents_module.cc
***InputProducerNoEvents	producer	/home/knoepfel/art/test/Integration/run-subrun-shape/InputProducerNoEvents_module.cc

Two modules that are specified in the same way.
How does art know which one to use?

art --print-description InputProducerNoEvents

module_type : **InputProducerNoEvents** (or "test/Integration/event-shape/InputProducerNoEvents")

provider: user

type : producer

source : /home/knoepfel/art/test/Integration/event-shape/InputProducerNoEvents_module.cc

library : /home/knoepfel/scratch/build-art-prof/lib/libtest_Integration_event-shape_InputProducerNoEvents_module.so

Allowed configuration

[None provided]

module_type : **InputProducerNoEvents** (or "test/Integration/run-subrun-shape/InputProducerNoEvents")

provider: user

type : producer

source : /home/knoepfel/art/test/Integration/run-subrun-shape/InputProducerNoEvents_module.cc

library : /home/knoepfel/scratch/build-art-prof/lib/libtest_Integration_run-subrun-shape_InputProducerNoEvents_module.so

Allowed configuration

[None provided]

art --print-description InputProducerNoEvents

```
-----  
module_type : InputProducerNoEvents (or "test/Integration/event-shape/InputProducerNoEvents")  
  provider: user  
  type    : producer  
  source  :  
  library : (or "test/Integration/event-shape/InputProducerNoEvents")  
  
Allowed configuration  
-----  
[ None provided ]  
  
-----  
module_type : InputProducerNoEvents (or "test/Integration/run-subrun-shape/InputProducerNoEvents")  
  provider: user  
  type    : producer  
  source  :  
  library : (or "test/Integration/run-subrun-shape/InputProducerNoEvents")  
  
Allowed configuration  
-----  
[ None provided ]  
-----
```

Using the long specification disambiguates between modules.

art --print-description BlockingPrescaler

```
=====
```

```
module_type : BlockingPrescaler (or "art/Framework/Modules/BlockingPrescaler")
```

```
provider: art
```

```
type    : filter
```

```
source  : /home/knoepfel/art/art/Framework/Modules/BlockingPrescaler_module.cc
```

```
library : /home/knoepfel/scratch/build-art-prof/lib/libart_Framework_Modules_BlockingPrescaler_module.so
```

```
Allowed configuration
```

```
-----
```

```
<module_label>: {
```

```
    module_type: BlockingPrescaler
```

```
    errorOnFailureToPut: true # default
```

```
    blockSize: 1 # default
```

```
    stepSize: <unsigned long>
```

```
    offset: 0 # default
```

```
}
```

```
=====
```

If configuration description enabled for the module/plugin, the allowed configuration is printed.

Secondary files

- One of the feature requests is to access products from the same event but in a file produced in an earlier process:
 - Smaller memory footprint
 - Access to unintentional dropping of products from earlier processes

Secondary files

- One of the feature requests is to access products from the same event but in a file produced in an earlier process:
 - Smaller memory footprint
 - Access to unintentional dropping of products from earlier processes

```
source: {  
  module_type: RootInput  
  fileNames: ["a1.root", "a2.root"]  
  secondaryFileNames: [  
    {  
      a: "a1.root"  
      b: ["b1.root", "c1.root"]  
    },  
    {  
      a: "a2.root"  
      b: ["b2.root", "c2.root"]  
    }  
  ]  
}
```

Secondary files

- One of the feature requests is to access products from the same event but in a file produced in an earlier process:
 - Smaller memory footprint
 - Access to unintentional dropping of products from earlier processes

```
source: {  
  module_type: RootInput  
  fileNames: ["a1.root", "a2.root"]  
  secondaryFileNames: [  
    {  
      a: "a1.root"  
      b: ["b1.root", "c1.root"]  
    },  
    {  
      a: "a2.root"  
      b: ["b2.root", "c2.root"]  
    }  
  ]  
}
```

Not obvious how this can work with SAM.

Results products

- Feature request from NOvA:
 - Need to be able to create products that do not correspond to an Event, SubRun, or Run, but correspond to a single result.
- New kind of product (`Results`), and a new kind of module (`ResultsProducer`).

ResultsProducer

```
class RPTest : public art::ResultsProducer {
public:

    explicit RPTest(fhicl::ParameterSet const& p); // required

    void beginJob() override { /*...*/ }

    void readResults (art::Results const& res) override { /*...*/ }

    void beginRun      (art::Run      const&) override { /*...*/ }
    void beginSubRun   (art::SubRun   const&) override { /*...*/ }
    void event         (art::Event    const&) override { /*...*/ }
    void endSubRun     (art::SubRun   const&) override { /*...*/ }
    void endRun        (art::Run      const&) override { /*...*/ }

    void writeResults(art::Results& res)  override; // required
    void clear()      override; // required

    void endJob() override { /*...*/ }
```

ResultsProducer

```
class RPTest : public art::ResultsProducer {
public:

    explicit RPTest(fhicl::ParameterSet const& p); // required

    void beginJob() override { /*...*/ }

    void beginRun      (art::Run      const&) override { /*...*/ }
    void beginSubRun   (art::SubRun   const&) override { /*...*/ }
    void event         (art::Event    const&) override { /*...*/ }
    void endSubRun     (art::SubRun   const&) override { /*...*/ }
    void endRun        (art::Run      const&) override { /*...*/ }

    void writeResults(art::Results& res) override; // required
    void clear() override; // required

    void endJob() override { /*...*/ }
```

Use the standard functions to build up your Results product, which is a member of the RPTest class.

ResultsProducer

```
class RPTest : public art::ResultsProducer {
public:

    explicit RPTest(fhicl::ParameterSet const& p); // required

    void beginJob() override { /*...*/ }

    void readResults (art::Results const& res) override { /*...*/ }

    void beginRun      (art::Run      const&) override { /*...*/ }

    void writeResults(art::Results& res)  override; // required
    void clear()      override; // required

    void endJob() override { /*...*/ }
```

Before the output file is closed, writeResults is called.

To write the result, it needs only:

```
res.put(std::make_unique<MyProduct>(…), myInstanceName);
```

ResultsProducer

```
class RPTest : public art::ResultsProducer {
public:

    explicit RPTest(fhicl::ParameterSet const& p); // required

    void beginJob() override { /*...*/ }

    void readResults (art::Results const& res) override { /*...*/ }

    void beginRun      (art::Run      const&) override { /*...*/ }
```

Before the output file is closed, `writeResults` is called.

To write the result, it needs only:

```
res.put(std::make_unique<MyProduct>(…), myInstanceName);
```

```
void writeResults(art::Results& res) override; // required
void clear() override; // required
```

The `clear` function is a place for users to reset their counters, clear their vectors, etc. It is called after `writeResults`.

ResultsProducer

```
class RPTest : public art::ResultsProducer {
public:

    explicit RPTest(fhicl::ParameterSet const& p); // required

    void beginJob() override { /*...*/ }

    void readResults (art::Results const& res) override { /*...*/ }

    void endSubRun (art::SubRun const&) override { /*...*/ }
    void endRun (art::Run const&) override { /*...*/ }

    void writeResults(art::Results& res) override; // required
    void clear() override; // required

    void endJob() override { /*...*/ }
```

Results products in the input file are not propagated to the output file. You must handle product propagation yourself. That is the purpose of `readResults`, which is called whenever an input file is opened.

ResultsProducer configuration

```
physics: {
  e1: [ o1 ]
}

outputs: {
  o1: {
    module_type: RootOutput
    fileName: "out_%r.root"
    results: {
      producers: {
        rpw: {
          plugin_type: RPTest
        }
        rpr: {
          plugin_type: RPTestReader
          intResultsLabel: "o1#rpw"
          nResultsExpected: 1
        }
      }
      rpath: [ rpw, rpr ]
    }
  }
}
```

ResultsProducer configuration

- ResultsProducers are tied to specific output streams.

```
physics: {  
  e1: [ o1 ]  
}  
outputs: {  
  o1: {  
    module_type: RootOutput  
    fileName: "out_%r.root"  
    results: {  
      producers: {  
        rpw: {  
          plugin_type: RPTest  
        }  
        rpr: {  
          plugin_type: RPTestReader  
          intResultsLabel: "o1#rpw"  
          nResultsExpected: 1  
        }  
      }  
      rpath: [ rpw, rpr ]  
    }  
  }  
}
```

ResultsProducer configuration

- ResultsProducers are tied to specific output streams.
- Module label for a ResultsProducer is the output-stream label, followed by '#', and then the producer module-label.

```
physics: {  
  e1: [ o1 ]  
}  
outputs: {  
  o1: {  
    module_type: RootOutput  
    fileName: "out_%r.root"  
    results: {  
      producers: {  
        rpw: {  
          plugin_type: RPTest  
        }  
        rpr: {  
          plugin_type: RPTestReader  
          intResultsLabel: "o1#rpw"  
          nResultsExpected: 1  
        }  
      }  
      rpath: [ rpw, rpr ]  
    }  
  }  
}
```

ResultsProducer configuration

- ResultsProducers are tied to specific output streams.
- Module label for a ResultsProducer is the output-stream label, followed by '#', and then the producer module-label.
- The order of processing is specified by the rpath parameter.

```
physics: {
  e1: [ o1 ]
}
outputs: {
  o1: {
    module_type: RootOutput
    fileName: "out_%r.root"
    results: {
      producers: {
        rpw: {
          plugin_type: RPTest
        }
        rpr: {
          plugin_type: RPTestReader
          intResultsLabel: "o1#rpw"
          nResultsExpected: 1
        }
      }
    }
  }
}
rpath: [ rpw, rpr ]
}
```

Configuration validation & description

- A common frustration:
What is the allowed configuration for a given module?
- One solution: look at the source code.
 1. *Okay...where is it?*
 2. *I'm a newcomer. Do I have to look at (potentially complicated) C++ to figure out how to configuration a presumably simple job?*
- Better solution:
Devise a system that documents itself, providing description and validation capabilities.

Configuration validation & description

- Need to find way to represent in C++ the three fundamental FHiCL values:

- **Atom:** no underlying structure

```
module_type: RootInput
```

- **Sequence:** list of unnamed objects

```
fileNames: ["a.root", "b.root"]
```

- **Table:** object with underlying name-value pairs

```
source: {  
  module_type: RootInput  
  fileNames: ["a.root", "b.root"]  
}
```

fhiclcpp type system

```
#include "fhiclcpp/types/Atom.h"  
#include "fhiclcpp/types/Sequence.h"  
#include "fhiclcpp/types/Table.h"  
  
#include "fhiclcpp/types/OptionalAtom.h"  
#include "fhiclcpp/types/OptionalSequence.h"  
#include "fhiclcpp/types/OptionalTuple.h"  
#include "fhiclcpp/types/OptionalTupleAs.h"  
#include "fhiclcpp/types/OptionalTable.h"  
#include "fhiclcpp/types/TableFragment.h"  
#include "fhiclcpp/types/Tuple.h"  
#include "fhiclcpp/types/TupleAs.h"
```

Simple analyzer

- Provide list of composers, and print out their names and birth/death dates
- Use the following configuration:

```
physics: {  
  analyzers: {  
    a1: {  
      module_type: ComposerAnalyzer  
      settings : {  
        printAtBeginJob: true  
        printAtEvent: false  
      }  
      composers: [Bach, Beethoven, Brahms]  
      births    : [1685, 1770, 1833]  
      deaths    : [1750, 1827, 1897]  
    }  
  }  
  e1: [a1]  
}
```

First look at the class

```
class ComposerAnalyzer : public art::EDAnalyzer {
public:

    ComposerAnalyzer(fhicl::ParameterSet const& pset)
        : EDAnalyzer{pset}
        , printAtBeginJob_{pset.get<bool>("settings.printAtBeginJob")}
        , printAtEvent_    {pset.get<bool>("settings.printAtEvent")    }
    {
        // Need to fill 'composers_' somehow
    }

    void beginJob() override
    {
        if (!printAtBeginJob_) return;
        for (auto const& c : composers_) cout << c;
    }

    void analyze(art::Event const&) override
    {
        if (!printAtEvent_) return;
        for (auto const& c : composers_) cout << c;
    }

private:
    bool printAtBeginJob_;
    bool printAtEvent_;
    vector<Composer> composers_;
};
```

First look at the class

```
class ComposerAnalyzer : public art::EDAnalyzer {
public:

    ComposerAnalyzer(fhicl::ParameterSet const& pset)
        : EDAnalyzer{pset}
        , printAtBeginJob_{pset.get<bool>("settings.printAtBeginJob")}
        , printAtEvent_    {pset.get<bool>("settings.printAtEvent")}
    {
        struct Composer {
        }
        Composer(string const& n, int const b, int const d)
            : name{n}, birth{b}, death{d}
        {}

        string name;
        int birth;
        int death;
    };

    ostream& operator<<(ostream& os, Composer const& c)
    {
        return os << c.name << " (" << c.birth << '-' << c.death << ")\n";
    }
private:
    bool printAtBeginJob_;
    bool printAtEvent_;
    vector<Composer> composers_;
};
```

Filling composers_

```
ComposerAnalyzer(fhicl::ParameterSet const& pset)
: EDAnalyzer{pset}
, printAtBeginJob_{pset.get<bool>("settings.printAtBeginJob")}
, printAtEvent_{pset.get<bool>("settings.printAtEvent")}
{
  auto const& composers = pset.get<vector<string>>("composers", {});
  auto const& births     = pset.get<vector<int>>("births", {});
  auto const& deaths     = pset.get<vector<int>>("death", {});

  for(size_t i{0}; i!=composers.size(); ++i)
    composers_.push_back( Composer{ composers[i],
                                     births[i],
                                     deaths[i]} );
}
```

- Ugly, but get's the job done.

Filling composers_

```
ComposerAnalyzer(fhicl::ParameterSet const& pset)
: EDAnalyzer{pset}
, printAtBeginJob_{pset.get<bool>("settings.printAtBeginJob")}
, printAtEvent_{pset.get<bool>("settings.printAtEvent")}
{
  auto const& composers = pset.get<vector<string>>("composers", {});
  auto const& births     = pset.get<vector<int>>("births", {});
  auto const& deaths    = pset.get<vector<int>>("death", {});

  for(size_t i{0}; i < composers.size(); ++i)
  {
    INFO: using default process_name, "DUMMY."
    %MSG-i MF_INIT_OK: mu2e 09-Feb-2016 12:38:05 CST JobSetup
    Messagelogs initialization complete.
    %MSG
    Segmentation fault
  }
}
```

- Ugly, but get's the job done...or not!

Filling composers_

```
ComposerAnalyzer(fhicl::ParameterSet const& pset)
: EDAnalyzer{pset}
, printAtBeginJob_{pset.get<bool>("settings.printAtBeginJob")}
, printAtEvent_{pset.get<bool>("settings.printAtEvent")}
{
  auto const& composers = pset.get<vector<string>>("composers", {});
  auto const& births     = pset.get<vector<int>>("births", {});
  auto const& deaths     = pset.get<vector<int>>("death", {});

  for(size_t i{0}; i < composers.size(); ++i)
  {
    INFO: using default process_name, "DUMMY."
    %MSG-i MF_INIT_OK: mu2e 09-Feb-2016 12:38:05 CST JobSetup
    Messagelogs initialization complete.
    %MSG
    Segmentation fault
  }
}
```

- Ugly, but get's the job done...or not!
- Oops...forgot the 's' at the end of "death".

Introducing the allowed configuration

```
physics: {  
  analyzers: {  
    a1: {  
      module_type: ComposerAnalyzer  
  
      composers: [Bach, Beethoven, Brahms]  
      births    : [1685, 1770, 1833]  
      deaths    : [1750, 1827, 1897]  
    }  
  }  
  e1: [a1]  
}
```

```
struct Config {  
  Sequence<string> composers { Name("composers"), vector<string>{} };  
  Sequence<int>    births    { Name("births")    , vector<int>{}    };  
  Sequence<int>    deaths    { Name("deaths")    , vector<int>{}    };  
};
```

Introducing the allowed configuration

```
struct Config {
    Sequence<string> composers { Name("composers"), vector<string>{} };
    Sequence<int>    births     { Name("births")    , vector<int>{}    };
    Sequence<int>    deaths     { Name("deaths")    , vector<int>{}    };
};

class ComposerAnalyzer : public art::EDAnalyzer {
public:

    using Parameters = art::EDAnalyzer::Table<Config>;
    ComposerAnalyzer(Parameters const& config)
        : EDAnalyzer{config}
        , printAtBeginJob_{pset.get<bool>("settings.printAtBeginJob")}
        , printAtEvent_   {pset.get<bool>("settings.printAtEvent")}
    {
        auto const& composers = config().composers(); // returns vector<string>
        auto const& births    = config().births();    // returns vector<int>
        auto const& deaths    = config().deaths();    // returns ""

        for(size_t i{0}; i!=composers.size(); ++i)
            composers_.push_back( Composer{ composers[i],
                                             births[i],
                                             deaths[i]} );
    }
};
```

Introducing the allowed configuration

```
struct Config {
    Sequence<string> composers { Name("composers"), vector<string>{} };
    Sequence<int>    births     { Name("births")    , vector<int>{}    };
    Sequence<int>    deaths    { Name("deaths")   , vector<int>{}   };
};

class ComposerAnalyzer : public art::EDAnalyzer {
public:

    using Parameters = art::EDAnalyzer::Table<Config>;
    ComposerAnalyzer(Parameters const& config)
        : EDAnalyzer{config}
        , printAtBeginJob_{pset.get<bool>("settings.printAtBeginJob")}
        , printAtEvent_   {pset.get<bool>("settings.printAtEvent")   }
    {
        auto const& composers = config().composers(); // returns vector<string>
        auto const& births    = config().births();   // returns vector<int>
        auto const& deaths    = config().deaths();   // returns ""

        for(size_t i{0}; i!=composers.size(); ++i)
            composers_.push_back( Composer{ composers[i],
                                             births[i],
                                             deaths[i]} );
    }
};
```

Introducing the allowed configuration

```
physics: {  
  analyzers: {  
    a1: {  
      module_type: ComposerAnalyzer  
      settings : {  
        printAtBeginJob: true  
        printAtEvent: false  
      }  
      composers: [Bach, Beethoven, Brahms]  
      births    : [1685, 1770, 1833]  
      deaths    : [1750, 1827, 1897]  
    }  
  }  
  e1: [a1]  
}
```

```
struct Settings {  
  Atom<bool> printAtBeginJob { Name("printAtBeginJob") };  
  Atom<bool> printAtEvent   { Name("printAtEvent") };  
};  
  
struct Config {  
  Table<Settings> settings { Name("settings") };  
  Sequence<string> composers { Name("composers"), vector<string>{} };  
  Sequence<int> births { Name("births"), vector<int>{} };  
  Sequence<int> deaths { Name("deaths"), vector<int>{} };  
};
```

Introducing the allowed configuration

```
using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
    : EDAnalyzer{config}
    , printAtBeginJob_{config().settings().printAtBeginJob()}
    , printAtEvent_{config().settings().printAtEvent()}
{
    auto const& composers = config().composers(); // returns vector<string>
    auto const& births     = config().births();   // returns vector<int>
    auto const& deaths     = config().deaths();   // returns ""

    for(size_t i{0}; i!=composers.size(); ++i)
        composers_.push_back( Composer{ composers[i],
                                         births[i],
                                         deaths[i]} );
}
```

Introducing the allowed configuration

```
using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
    : EDAnalyzer{config}
    , printAtBeginJob_{config().settings().printAtBeginJob()}
    , printAtEvent_{config().settings().printAtEvent()}
{
    auto const& composers = config().composers(); // returns vector<string>
    auto const& births    = config().births();   // returns vector<int>
    auto const& deaths    = config().deaths();   // returns ""

    for(size_t i{0}; i!=composers.size(); ++i)
        composers_.push_back( Composer{ composers[i],
                                         births[i],
                                         deaths[i]} );
}
```

Introducing the allowed configuration

```
using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
    : EDAnalyzer{config}
    , printAtBeginJob_{config().settings().printAtBeginJob()}
    , printAtEvent_{config().settings().printAtEvent()}
{
    auto const& composers = config().composers(); // returns vector<string>
    auto const& births    = config().births();   // returns vector<int>
    auto const& deaths    = config().deaths();   // returns ""

    for(size_t i{0}; i!=composers.size(); ++i)
        composers_.push_back( Composer{ composers[i],
                                         births[i],
                                         deaths[i] } );
}
```

```
Bach (1685-1750)
Beethoven (1770-1827)
Brahms (1833-1897)
```

What happens if you misspecify a parameter?

```
!! The following modules have been misconfigured: !!
```

```
Module label: a1  
module_type : ComposerAnalyzer
```

```
Missing parameters:
```

```
  settings: {  
-   printAtBeginJob: <bool>  
  }
```

```
Unsupported parameters:
```

```
+ settings.printAtBeginRun    [ ]
```

A better configuration

- But the configuration was icky.
- A better one:

```
physics: {  
  analyzers: {  
    a1: {  
      module_type: ComposerAnalyzer  
      settings : {  
        printAtBeginJob: true  
        printAtEvent: false  
      }  
      composers: [  
        [Bach      , 1685, 1750],  
        [Beethoven, 1770, 1827],  
        [Brahms   , 1833, 1897]  
      ]  
    }  
  }  
  e1: [a1]  
}
```

A better configuration

- But the configuration was icky.
- A better one:

```
physics: {  
  analyzers: {  
    a1: {  
      module_type: ComposerAnalyzer  
      settings : {  
        printAtBeginJob: true  
        printAtEvent: false  
      }  
      composers: [  
        [Bach      , 1685, 1750],  
        [Beethoven, 1770, 1827],  
        [Brahms   , 1833, 1897]  
      ]  
    }  
  }  
  e1: [a1]  
}
```

Heterogeneous sequences
Use `Tuple<string, int, int>`.

Introducing the allowed configuration

```
struct Settings {
    Atom<bool> printAtBeginJob { Name("printAtBeginJob") };
    Atom<bool> printAtEvent    { Name("printAtEvent") };
};

struct Config {
    Table<Settings> settings { Name("settings") };
    Sequence<Tuple<string,int,int>> composers { Name("composers") };
};

using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
    : EDAnalyzer{config}
    , printAtBeginJob_{config().settings().printAtBeginJob()}
    , printAtEvent_    {config().settings().printAtEvent()}
{
    auto const& composers = config().composers(); // returns vector<tuple<string,int,int>>
    for(size_t i{0}; i!=composers.size(); ++i)
        composers_.push_back( Composer{ std::get<0>(composers[i]),
                                         std::get<1>(composers[i]),
                                         std::get<2>(composers[i])} );
}
```

Introducing the allowed configuration

```
struct Settings {
    Atom<bool> printAtBeginJob { Name("printAtBeginJob") };
    Atom<bool> printAtEvent    { Name("printAtEvent") };
};

struct Config {
    Table<Settings> settings { Name("settings") };
    Sequence<Tuple<string,int,int>> composers { Name("composers") };
};

using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
: EDAnalyzer{config}
, printAtBeginJob_{config().settings().printAtBeginJob()}
, printAtEvent_    {config().settings().printAtEvent()}
{
    auto const& composers = config().composers(); // returns vector<tuple<string,int,int>>
    for(size_t i{0}; i!=composers.size(); ++i)
        composers_.push_back( Composer{ std::get<0>(composers[i]),
                                          std::get<1>(composers[i]),
                                          std::get<2>(composers[i])} );
}
```

Introducing the allowed configuration

```
struct Settings {
    Atom<bool> printAtBeginJob { Name("printAtBeginJob") };
    Atom<bool> printAtEvent    { Name("printAtEvent") };
};

struct Config {
    Table<Settings> settings { Name("settings") };
    Sequence<Tuple<string,int,int>> composers { Name("composers") };
};

using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
: EDAnalyzer{config}
, printAtBeginJob_{config.settings.printAtBeginJob}
, printAtEvent_{config.settings.printAtEvent}
{
    auto const& composers = config().composers(); // returns vector<tuple<string,int,int>>
    for(size_t i{0}; i!=composers.size(); ++i)
        composers_.push_back( Composer{ std::get<0>(composers[i]),
                                          std::get<1>(composers[i]),
                                          std::get<2>(composers[i])} );
}
```

Index 'i' used only to access elements.
Replace with range-for loop.

Introducing the allowed configuration

```
using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
    : EDAnalyzer{config}
    , printAtBeginJob_{config().settings().printAtBeginJob()}
    , printAtEvent_{config().settings().printAtEvent()}
{
    for(auto const& c : config().composers())
        composers_.push_back( Composer{ std::get<0>(c),
                                         std::get<1>(c),
                                         std::get<2>(c)} );
}
```

Introducing the allowed configuration

```
using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
    : EDAnalyzer{config}
    , printAtBeginJob_{config().settings().printAtBeginJob()}
    , printAtEvent_{config().settings().printAtEvent()}
{
    for(auto const& c : config().composers())
        composers_.push_back( Composer{ std::get<0>(c),
                                         std::get<1>(c),
                                         std::get<2>(c)} );
}
```

Shouldn't have to use `push_back(Composer{...})`.
The vector knows what type the element is. Replace
`push_back` with `emplace_back`.

Introducing the allowed configuration

```
using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
    : EDAnalyzer{config}
    , printAtBeginJob_{config().settings().printAtBeginJob()}
    , printAtEvent_{config().settings().printAtEvent()}
{
    for(auto const& c : config().composers())
        composers_.emplace_back(std::get<0>(c), std::get<1>(c), std::get<2>(c));
}
```

What does the description look like?

```
[knoepfel@woof build-art-prof]$ mu2e --print-description ComposerAnalyzer
```

```
module_type : ComposerAnalyzer (or "test/Integration/ComposerAnalyzer")

provider: user
type    : analyzer
source  : /home/knoepfel/art/test/Integration/ComposerAnalyzer_module.cc
library : /home/knoepfel/scratch/build-art-prof/lib/libtest_Integration_ComposerAnalyzer_module.so

Allowed configuration
-----

<module_label>: {
    module_type: ComposerAnalyzer

    ( SelectEvents: {
      (
      ( # The following parameter is a user-provided list
      ( # of filter paths. The default list is empty.
      (
      ( SelectEvents: [
      ( ]
      ( }
    )

    settings: {
      printAtBeginJob: <bool>

      printAtEvent: <bool>
    }

    composers: [
      [
        <string>,
        <int>,
        <int>
      ],
      ...
    ]
  }
}
```

What does the description look like?

```
[knoepfel@woof build-art-prof]$ mu2e --print-description ComposerAnalyzer
```

```
module_type : ComposerAnalyzer (or "test/Integration/ComposerAnalyzer")

provider: user
type    : analyzer
source  : /home/knoepfel/art/test/Integration/ComposerAnalyzer_module.cc
library : /home/knoepfel/scratch/build-art-prof/lib/libtest_Integration_ComposerAnalyzer_module.so

Allowed configuration
-----

<module_label>: {
  module_type: ComposerAnalyzer

  ( SelectEvent
  (
  ( # The
  ( # of f
  (
  ( Select
  ( ]
  ( }

  settings:
    printA
    printA
  }

  composers
  [
    <st
    <in
    <in
  ],
  ...
]
}
```

```
composers: [
  [
    <string>,
    <int>,
    <int>
  ],
  ...
]
```

What should I provide as the string and the integers?

Adding a comment

```
struct Settings {
  Atom<bool> printAtBeginJob { Name("printAtBeginJob") };
  Atom<bool> printAtEvent    { Name("printAtEvent") };
};

struct Config {
  Table<Settings> settings { Name("settings") };
  Sequence<Tuple<string,int,int>> composers { Name("composers"), Comment("This is a list of composers; for each entry,\n"
    "the first field is the name of the composer,\n"
    "and the second and third entries are the birth\n"
    "and death dates of the composer, respectively.\n") };
};
```

Adding a comment

```
struct Settings {
  Atom<bool> printAtBeginJob { Name("printAtBeginJob") };
  Atom<bool> printAtEvent   { Name("printAtEvent") };
};

struct Config {
  Table<Settings> settings { Name("settings") };
  Sequence<Tuple<string,int,int>> composers { Name("composers"), Comment("This is a list of composers; for each entry,\n"
    "the first field is the name of the composer,\n"
    "and the second and third entries are the birth\n"
    "and death dates of the composer, respectively.\n") };
};
```

```
composers: [
  [
    <string>,
    <int>,
    <int>
  ],
  ...
]
```

```
# This is a list of composers; for each entry,
# the first field is the name of the composer,
# and the second and third entries are the birth
# and death dates of the composer, respectively.
```

```
composers: [
  [
    <string>,
    <int>,
    <int>
  ],
  ...
]
```

One of the very few times to use a preprocessor macro

- *If you need to provide a lengthy string literal.*

```
#define COMPOSER_COMMENT \
    "This is a list of composers; for each entry,\n" \
    "the first field is the name of the composer,\n" \
    "and the second and third entries are the birth\n" \
    "and death dates of the composer, respectively.\n"
```

```
struct Config {
    Table<Settings> settings { Name("settings") };
    Sequence<Tuple<string,int,int>> composers { Name("composers"), Comment(COMPOSER_COMMENT) };
};
```

Where we were...

```
using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
    : EDAnalyzer{config}
    , printAtBeginJob_{config().settings().printAtBeginJob()}
    , printAtEvent_{config().settings().printAtEvent()}
{
    for(auto const& c : config().composers())
        composers_.emplace_back(std::get<0>(c), std::get<1>(c), std::get<2>(c));
}
```

Where we were...

```
using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
    : EDAnalyzer{config}
    , printAtBeginJob_{config().settings().printAtBeginJob()}
    , printAtEvent_{config().settings().printAtEvent()}
{
    for(auto const& c : config().composers())
        composers_.emplace_back(std::get<0>(c), std::get<1>(c), std::get<2>(c));
}
```

Would be nice to have a mechanism that automatically converts the `Tuple` sequence to `Composer`, without me having to do it by hand.

Introducing `TupleAs...`

Where we were...

```
using Parameters = art::EDAnalyzer::Table<Config>;
ComposerAnalyzer(Parameters const& config)
    : EDAnalyzer{config}
    , printAtBeginJob_{config().settings().printAtBeginJob()}
    , printAtEvent_{config().settings().printAtEvent()}
{
    for(auto const& c : config().composers())
        composers_.emplace_back(std::get<0>(c), std::get<1>(c), std::get<2>(c));
}
```

Would be nice to have a mechanism that automatically converts the Tuple sequence to Composer, without me having to do it by hand.

Introducing TupleAs...

For this case:

```
TupleAs< Composer(string, int, int) >
```

TupleAs

```
struct Config {  
    Table<Settings> settings { Name("settings") };  
    Sequence< TupleAs< Composer(string,int,int) > > composers { Name("composers"),  
                                                                Comment(COMPOSER_COMMENT) };  
};
```

TupleAs

```
struct Config {  
    Table<Settings> settings { Name("settings") };  
    Sequence< TupleAs< Composer(string,int,int) > > composers { Name("composers"),  
                                                                Comment(COMPOSER_COMMENT) };  
};
```

```
using Parameters = art::EDAnalyzer::Table<Config>;  
ComposerAnalyzer(Parameters const& config)  
    : EDAnalyzer{config}  
    , printAtBeginJob_{config().settings().printAtBeginJob()}  
    , printAtEvent_    {config().settings().printAtEvent()}  
    , composers_       {config().composers()} // returns vector<Composer>  
{}
```

Description with TupleAs

```
# This is a list of composers; for each entry,  
# the first field is the name of the composer,  
# and the second and third entries are the birth  
# and death dates of the composer, respectively.  
  
composers: [  
  
    # N.B. The following sequence is converted to type:  
    #      '(anonymous namespace)::Composer'  
    [  
        <string>,  
        <int>,  
        <int>  
    ],  
    ...  
]
```

Description with TupleAs

```
Sequence< TupleAs< Composer(string, int, int) > > composers { Name("composers"),  
                                                             Comment(COMPOSER_COMMENT),  
                                                             vector<Composer>{Composer{Mahler, 1860, 1911}} };
```

```
# This is a list of composers; for each entry,  
# the first field is the name of the composer,  
# and the second and third entries are the birth  
# and death dates of the composer, respectively.
```

```
composers: [
```

```
    # N.B. The following sequence is converted to type:  
    #       '(anonymous namespace)::Composer'  
    #       with a default value of:  
    #       Mahler (1860-1911)
```

```
    [  
        <string>,  
        <int>,  
        <int>
```

```
    ]
```

```
]
```

Quite a bit more ...

- Conditional configuration
- Optional parameters
- Supporting flat configurations
- See:

https://cdcvcs.fnal.gov/redmine/projects/art/wiki/Configuration_validation_and_description

ROOT5 vs. ROOT6

- **Practical considerations:**
 - All ROOT development is concentrated on ROOT6/7.
 - ROOT5 is quickly becoming (if not already) an unsupported product.
 - All LHC experiments have adopted ROOT6.
- **Technical considerations:**
 - ROOT6 is much more stringent on syntax. Your macros must contain valid C++.
 - ROOT6 uses more memory than ROOT5 does. Not grossly so, but a little bit.
 - All ROOT5 files can be read by ROOT6.
 - All ROOT6 files can be read by ROOT5.
 - C++11/14 constructs are typically allowed for data-product dictionaries.

ROOT5 vs. ROOT6

- **Practical considerations:**

- All ROOT development is concentrated on ROOT6/7.
 - ROOT5 is quickly becoming (if not already) an unsupported product.
- All LHC experiments have adopted ROOT6.

- **Technical considerations:**

- ROOT6 is much more stringent on syntax. Your macros must contain valid C++.

ROOT6-supported versions of *art* exist.

- Significant portion of last year went toward implementing them.

To our knowledge, no experiment has reported updating their builds to use ROOT6.

We would greatly benefit from hearing of experiments' experiences when upgrading to ROOT6.

duct

Soon to come ...

- More flexible output-file handling
- Specifying an arbitrary list of events to process
- etc.

Thank you.