# Configuring tools with run data

## DUNE Software Architecture

David Adams

BNL

July 28, 2021

# Introduction

## Why are we having these meetings?

- I hope we go beyond the narrow scope of splitting dunetpc
- Instead look for general patterns for writing SW
  - Make it easier for users for find and use existing SW

## This talk follows from my need to access conditions data

- Specifically, TPC run data:
  - Amplifier gain and shaping
  - Pulser source and DAC setting
- Used in art event loop, Root scripts, Python and shell scripts
- Applications include
  - Labels for plots generated in Root scripts and art jobs
  - Calibration scripts
  - Configuration of dataprep tools
    - Like to move away from different tool instances for each configuration
    - To configuration params that depend on conditions values, e.g.

      mytool.Threshold: “12.5*[gain]*[shaping]”

# History

I gave talks at the Conditions DB workshop July 2

- Offline requirements:
  - Interface for accessing conditions data
  - How conditions data is stored
  - What conditions data should be stored
- I also gave talk on existing ProtoDUNE access to conditions data

I talked about retrieval of conditions data at last (7/14) SWA

- Model is conditions data objects are retrieved through tool interface
- Retrieved object and tool interface are specific to each conditions category (run data, HV, e lifetimes, …)
  - Or should we have a generic interface for multiple categories?
- Tool interface implies we can have multiple implementations for a given category, e.g. for different storage mechanisms
  - Storage can evolve with time (text file today, Oracle DB tomorrow)
  - Or can be site specific including caching

# Recent progress

I have been working on a specific problem

- Configuring dataprep tools with formulas based on run data parameters
- To avoid the plethora of tool configurations needed to process Iceberg data taken with many preamp gains and shaping times
- Goal is to solve the problem and learn lessons about retrieval of conditions data
- Work is documented in dunetpc ticket 26064
- Following slides outline some work so far

# Run data retrieval

## Run data object was already existing

- dunetpc/dune/DuneInterface/Data/RunData.h
- Holds run, event preamp gain, shaping and much more with methods to get, set and check each field
  - Would we be better with generic method taking field names:
  - get(“gain”) instead of getGain()?

## Tool interface to retrieve these objects also already existing

- dunetpc/dune/DuneInterface/Tool/RunDataTool
- Method returns the run data for a given run number
  - For many or most conditions categories, a time stamp (or range of time stamps) will be more appropriate

## Run data Tool implementations

- Fcl-based implementation has existed for many years
  - Tedious to make a fcl file for each run
- Recently added an implementation that uses sam to find a file for the run number and then return the run data associated with that file

# Run data formulas

For me, the primary user of run data are dataprep tools

- I want to configure a tool with the run data, e.g.

  mytool.Threshold = “15.0*[gain]”

First solution added code to one dataprep tool

- Use TFormula to interpret a configuration string
- Retrieve run data, use parameter names to figure our which methods to call to retrieve values from run data, fetch those values, set them in formula and then evaluate the formula
  - Name of run data tool (“runDataTool”) is hardwired into dataprep tool
- Fine for one tool and one or two run data fields
- But a lot of code would have to be duplicated and maintained for every other tool (or other client) with formulas
- Improvements that relocate much of this code are described on the following pages

# ParFormula

## ParFormula interface added

- dunetpc/dune/DuneInterface/Utility/ParFormula.h
- Subclasses hold a formula and provide methods to
  - Indicate which parameter names appear in the formula
  - Set the parameters
  - Evaluate the formula

## Implementation based on TFormula was added

- dunetpc/dune/DuneCommon/Utility/RootParFormula.h
- Dataprep tools read config strings and constructs tools of this type
  - Alternative would be to make ParFormula a tool interface and configure the dataprep tool with that tool name or configuration

# Set formula parameters in RunData

## Method to evaluate parameters added to RunData

- Dataprep tool passes ParFormula object to the RunData object
- Latter does the work of mapping formula parameter names to run data fields

## Lesson for conditions data retrieval

- The mapping would be easy and require less maintenance if the RunData class held name:value pairs for each of its fields
- Might want to use the same association and mapping code for other conditions categories
- Maybe even common class or base class for many or all categories?

# Second (current) solution

Formula-based dataprep tools use the preceding enhancements

- See Redmine ticket for the current list of such tools
- Construct RootParFormula for each formula parameter
- Fetch tool to retrieve run data if any formulas have parameters
- Dataprep tool knows when run has (or may have) changed
  - Uses its run data tool to retrieve the new run data object
  - Passes the formula to that object to update parameters
- When needed, evaluates the formula to get the new value
  - E.g. threshold

# Further enhancement

There are many other possible enhancements

- Helper class containing some of code and state information that is being repeated in the formula-based tools
- Expand to other conditions categories
  - Dataprep tool (or helper) passes each formula to a series of conditions data objects which update their parameters in turn
  - Might want dynamic configuration of the conditions categories and tool names
    - E.g. mytool.Threshold: “12.5*[runData.gain]”
- Configure dataprep tools with with formula tools instead of strings
  - By name or with configuration string (could support both)
  - Instead of constructing RootParFormula in dataprep tool
  - Dataprep tool calls formula.eval() any time it needs the value
    - Must first pass keys (event info: run, event, time, ..) to the formula
- Add support for labels using conditions data
  - E.g. mytool.gainLabel: “Preamp gain = [gain,%.1f] mV/fC”

# Backward compatibility

## Tool configuration is changing

- Fcl params that once were numbers, e.g.
  - mytool.Threshold: 12.0
- Now are configured with strings
  - mytool.Threshold: “12.0” or
  - mytool.Threshold: “12.0*[gain]/14.0”
- I see the numeric configuration also works with strings
  - I.e. fcl parser does not require the quotes for simple expressions
  - OK not to update all fcl when parameter is converted to a formula
    - Can we rely on this?
    - Probably preferable to make change so users looking at fcl files know when a parameter is a formula

# Comments/conclusions

To summarize:

- I am modifying selected dataprep tools so they can be configured with values that depend on run data
  - Preamp gain, shaping time and charge injection info
  - So one configuration can work for many settings
- Use this experience to learn lessons for DUNE SW architecture
  - Interface for conditions data retrieval
  - Propagate conditions data to formulas and labels

Thank you

# Extras

Conditions categories

# Categories

First pass at list of categories (~DB tables)

- Run configuration
- Event metadata
- Geometry
- Event data file metadata (now in SAM)
- Datasets
- Good run (event?) lists
- Electron lifetime
- LAr temperature
- Cathode and anode voltages and currents
- Detector status (APAs, FEMBs, …)
- Channel status
- TPC CE channel calibration
- TPC dQ/dx correction
- Beam status