Conditions database - Updates

Ana Paula Vizcaya

15 - April - 2024

Colorado State University

Conditions database - Run conditions table

- Google docs with list of parameters, from meeting with interested people
- Will create new table with this list

https://docs.google.com/spreadsheets/d/1yI3T6cQEvWU7sisbjxIsViionl2k1EZPcu2mos0rhZ0/edit#gid=0

  - DAQ
    - ready
    - test table using this info
  - Slow control
    - Nilay is working on it
  - IFbeam
    - CERN was blocking data transfer to FNAL, Vladimir and Steve were on it
    - will use other sensors to indirectly get what we want
    - I will work on it, or Jake will include in file metadata

Run conditions table - DAQ time problem

- DAQ run registry start time and end time is incorrect!!
- They are addressing this problem, but not resolved yet.
- Old runs won't change

Interface with Metacat

- Marc Mengel and Alison Peisker are helping me with this
- Modified old filter (https://github.com/ivmfnal/metacat/blob/main/DUNE_specials/custom_filters/runsdb_incondb.py) to work with new version of run conditions table
- Already a dune_meta_prod instance on the test server
  - Need to try it out
- Hopefully have it ready by the end of the week
  - Working with a test table but just one parameter needs to change to try another table

Conditions database with Alma9

- The databases have moved without any problem
- The python API
  - Can be run on Alma9 but its harder to install
  - Before just install dunesw and all dependencies were installed
  - Now need to create python environment - Documentation on the way

- Todo
  - Finish documentation
  - Update de cron jobs that fill the run conditions table

Art and C++ interface with database

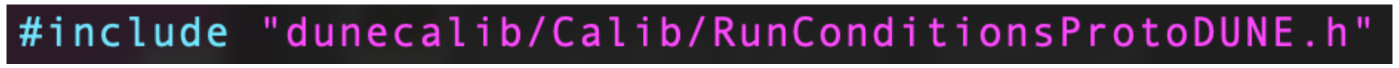
- New (updated) art an c++ interface with conditions database
- Based on: /nuevdb/IFDatabase and /dunecalib
    - More functions (upload data, load data from other databases)
    - Can't upload more that 3 columns or so (bug)
    - Can't specify new URL or interface with new conditions database
- New code
- On dunecalib - pull request needs review
  - Different options:
    - Provide URL
    - Provide table name and schema
  - Still uses some of the old code
    - Column
    - Row

Examples

- c++ example of how to access the data from the runs condition table
  - /dunecalib/ConInt/getRunConditionsPDUNE.cc

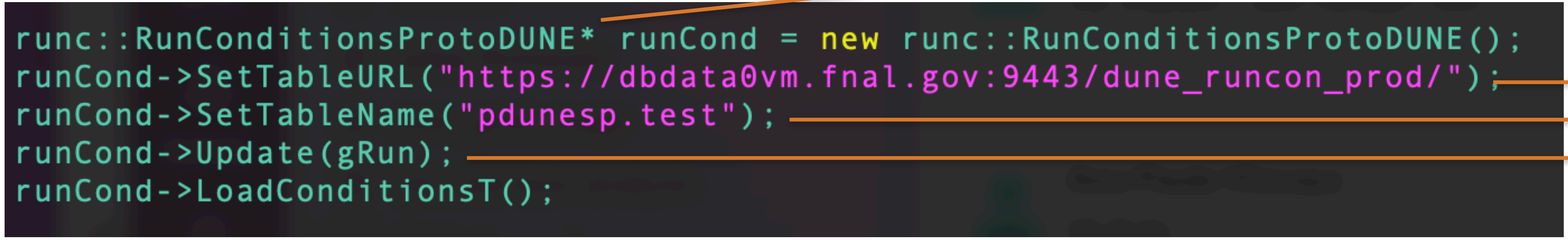
- For the art service
  - /dunecalib/ConIntServices/RunConditionsServicePDUNE_service.cc
  - /dunecalib/ConintServices/runconditions_pdune.fcl

C++ interface - Run Conditions table example



```
#include "dunecalib/Calib/RunConditionsProtoDUNE.h"
```

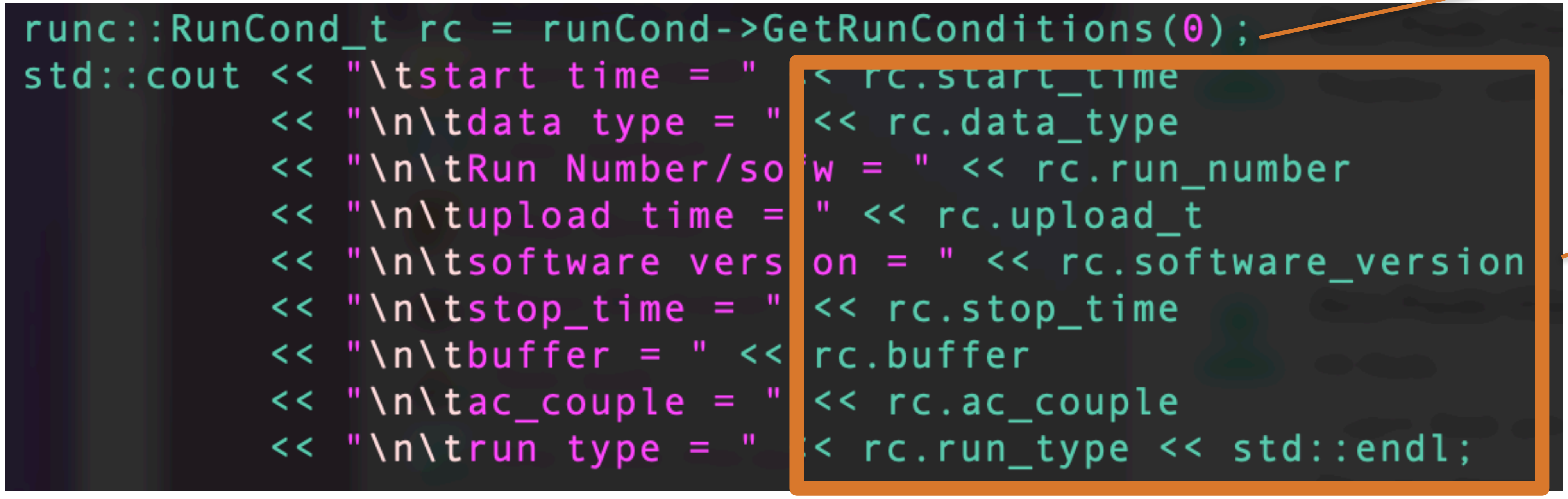
Include header file



```
runc::RunConditionsProtoDUNE* runCond = new runc::RunConditionsProtoDUNE();
runCond->SetTableURL("https://dbdata0vm.fnal.gov:9443/dune_runcon_prod/");
runCond->SetTableName("pdunesp.test");
runCond->Update(gRun);
runCond->LoadConditionsT();
```

Declare Run Condition table

Just needs 3 inputs:
URL
Table Name
Run/time



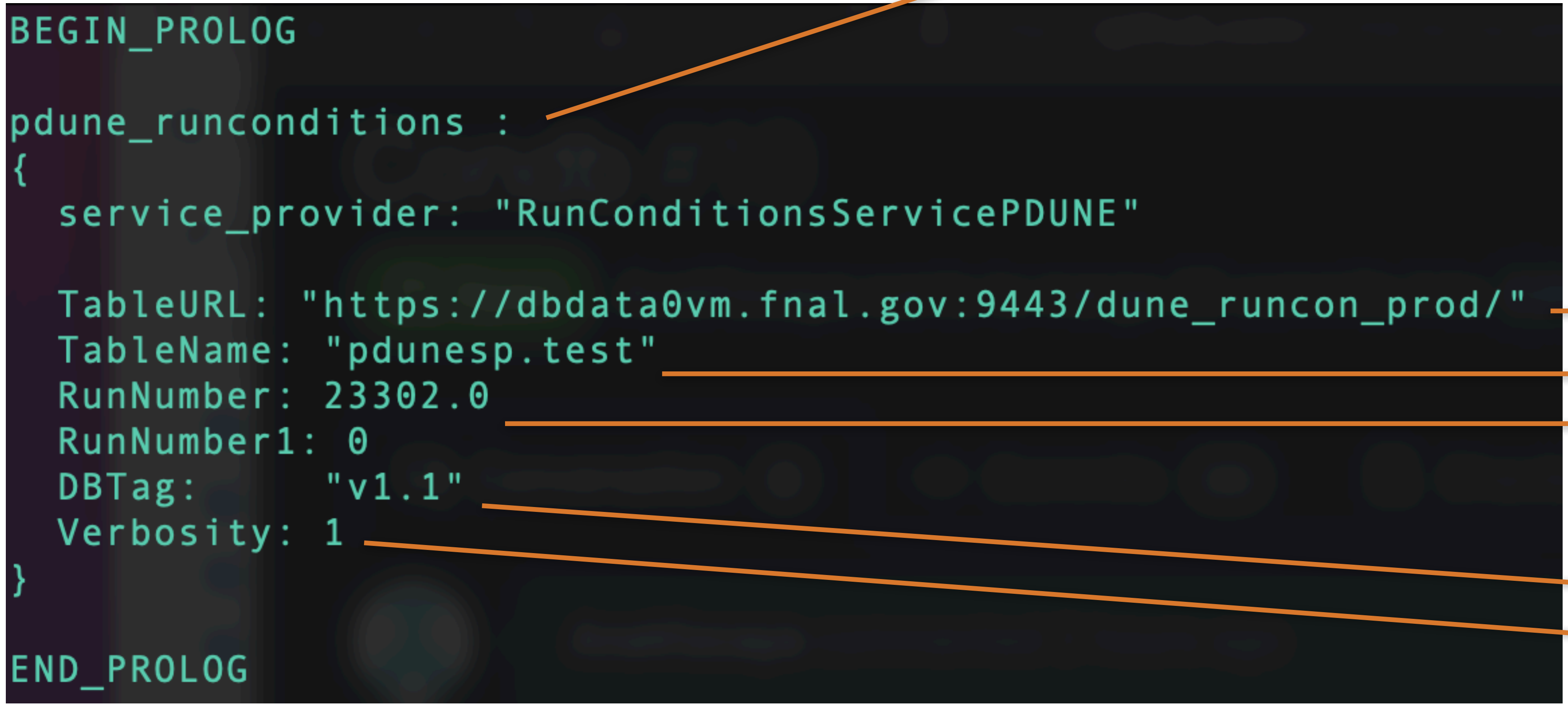
```
runc::RunCond_t rc = runCond->GetRunConditions(0);
std::cout << "\tstart time = " << rc.start_time
          << "\n\tdata type = " << rc.data_type
          << "\n\tRun Number/so w = " << rc.run_number
          << "\n\tupload time = " << rc.upload_t
          << "\n\tsoftware vers on = " << rc.software_version
          << "\n\tstop_time = " << rc.stop_time
          << "\n\tbuffer = " << rc.buffer
          << "\n\tac_couple = " << rc.ac_couple
          << "\n\trun type = " << rc.run_type << std::endl;
```

Declare row (one per run in this example, use run number value)

Variables ready to use, previously declared at the code specific to the table

ART service:

Service name

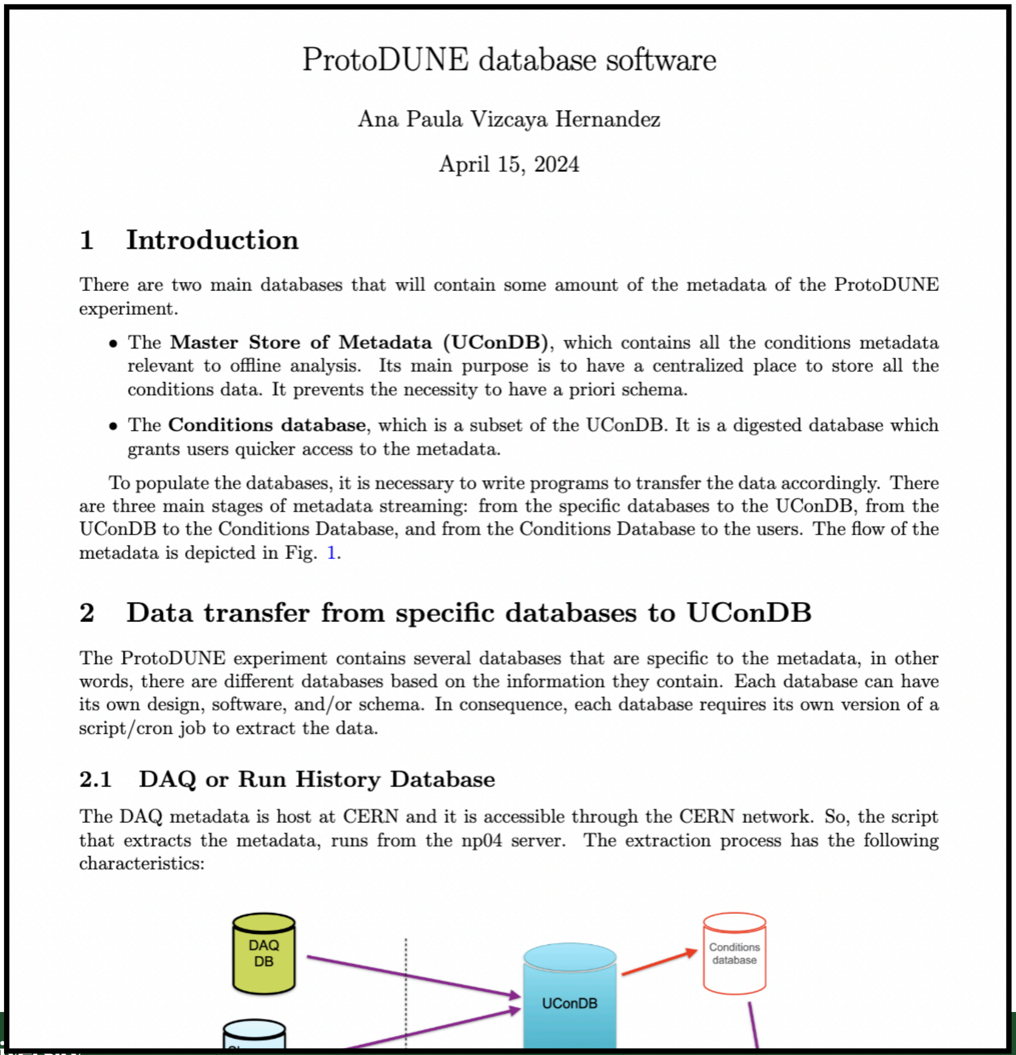


```
BEGIN_PROLOG

pdune_runconditions :
{
  service_provider: "RunConditionsServicePDUNE"

  TableURL: "https://dbdata0vm.fnal.gov:9443/dune_runcon_prod/"
  TableName: "pdunesp.test"
  RunNumber: 23302.0
  RunNumber1: 0
  DBTag:     "v1.1"
  Verbosity: 1
}

END_PROLOG
```

Inputs:
URL
Table Name
Run Number to get data from, if runNumber1 is also specified the in returns data from the interval
Table tag, returns newest if left blank
Verbosity is how much output you get

Documentation

- For admin users (Norm, Heidi)



ProtoDUNE database software

Ana Paula Vizcaya Hernandez

April 15, 2024

1 Introduction

There are two main databases that will contain some amount of the metadata of the ProtoDUNE experiment.

- The **Master Store of Metadata (UConDB)**, which contains all the conditions metadata relevant to offline analysis. Its main purpose is to have a centralized place to store all the conditions data. It prevents the necessity to have a priori schema.
- The **Conditions database**, which is a subset of the UConDB. It is a digested database which grants users quicker access to the metadata.

To populate the databases, it is necessary to write programs to transfer the data accordingly. There are three main stages of metadata streaming: from the specific databases to the UConDB, from the UConDB to the Conditions Database, and from the Conditions Database to the users. The flow of the metadata is depicted in Fig. 1.

2 Data transfer from specific databases to UConDB

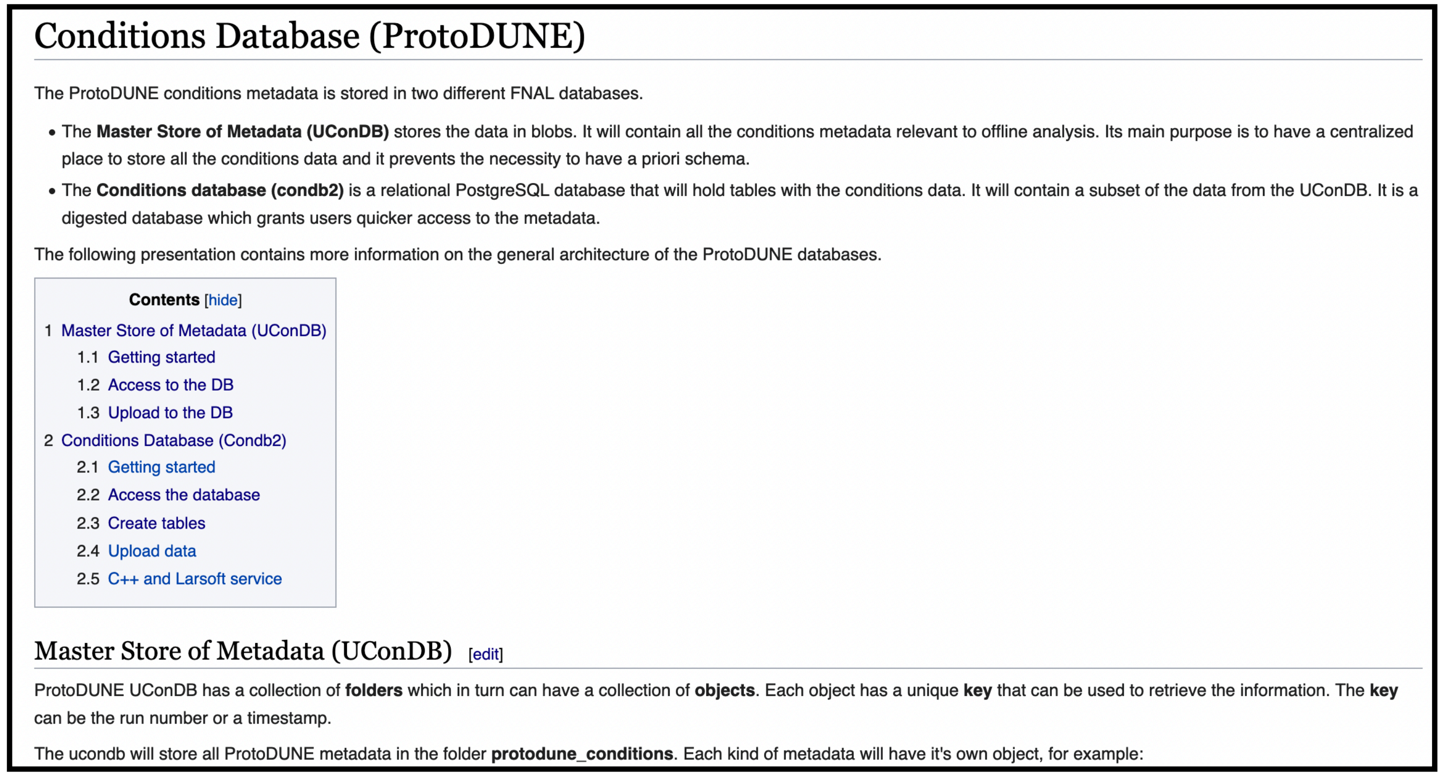
The ProtoDUNE experiment contains several databases that are specific to the metadata, in other words, there are different databases based on the information they contain. Each database can have its own design, software, and/or schema. In consequence, each database requires its own version of a script/cron job to extract the data.

2.1 DAQ or Run History Database

The DAQ metadata is host at CERN and it is accessible through the CERN network. So, the script that extracts the metadata, runs from the np04 server. The extraction process has the following characteristics:

- For normal users

https://wiki.dunescience.org/wiki/Conditions_Database_(ProtoDUNE)#Access_to_the_DB



Conditions Database (ProtoDUNE)

The ProtoDUNE conditions metadata is stored in two different FNAL databases.

- The **Master Store of Metadata (UConDB)** stores the data in blobs. It will contain all the conditions metadata relevant to offline analysis. Its main purpose is to have a centralized place to store all the conditions data and it prevents the necessity to have a priori schema.
- The **Conditions database (condb2)** is a relational PostgreSQL database that will hold tables with the conditions data. It will contain a subset of the data from the UConDB. It is a digested database which grants users quicker access to the metadata.

The following presentation contains more information on the general architecture of the ProtoDUNE databases.

Contents [hide]

Master Store of Metadata (UConDB) [edit]

ProtoDUNE UConDB has a collection of **folders** which in turn can have a collection of **objects**. Each object has a unique **key** that can be used to retrieve the information. The **key** can be the run number or a timestamp.

The ucondb will store all ProtoDUNE metadata in the folder **protodune_conditions**. Each kind of metadata will have it's own object, for example:

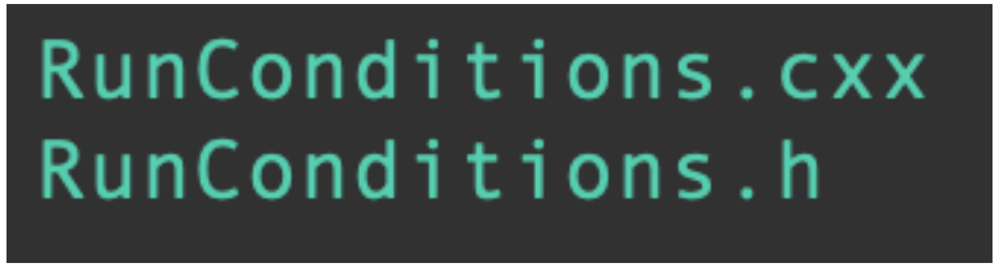
To-Do

- Write documentation on
  - How to create code for new table
    - Once per table
    - Follow template
  - Users
    - How to access data

- Integration with Metacat

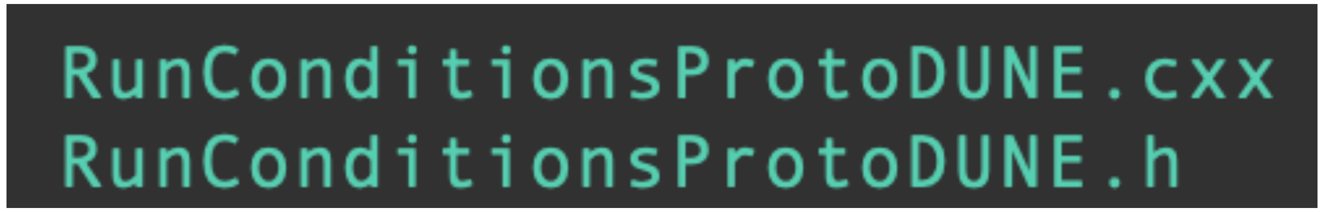
ART and C++ code

- The code
- General code to access the conditions tables in the conditions database



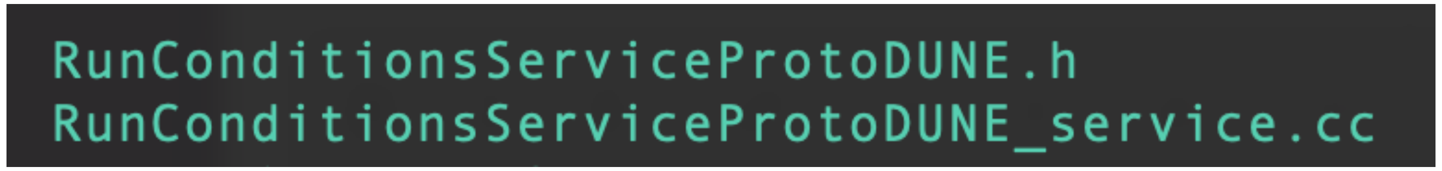
```
RunConditions.cxx
RunConditions.h
```

- Code specific to a table, where variables corresponding to table columns are defined



```
RunConditionsProtoDUNE.cxx
RunConditionsProtoDUNE.h
```

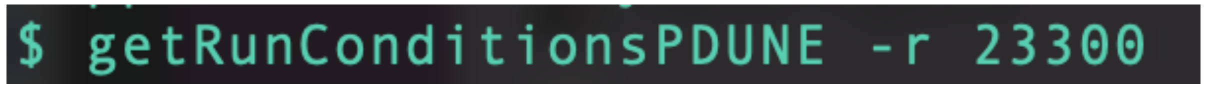
- ART service specific to the table



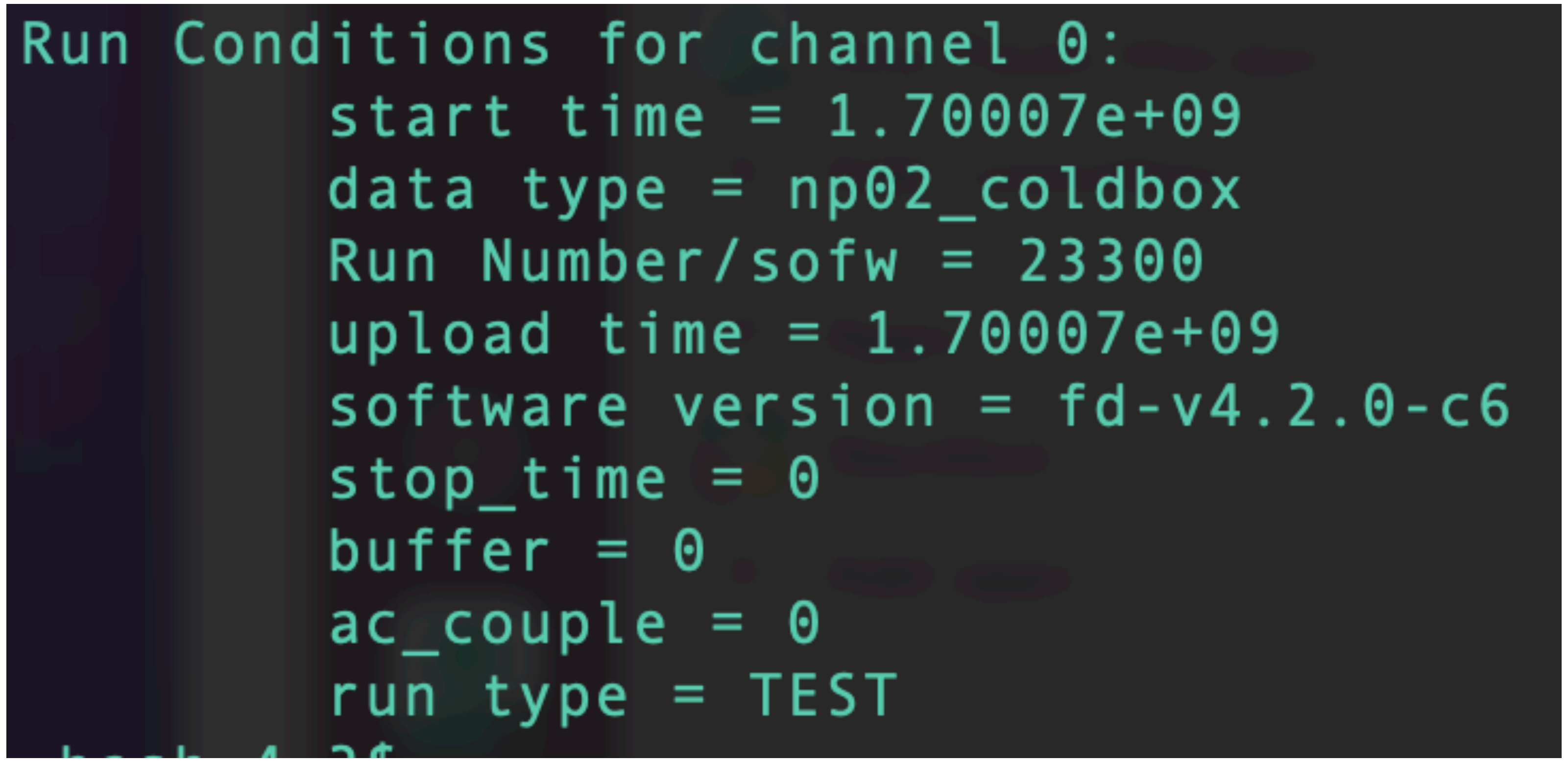
```
RunConditionsServiceProtoDUNE.h
RunConditionsServiceProtoDUNE_service.cc
```

C++ interface - Run Conditions table example

Output of c++ script



```
$ getRunConditionsPDUNE -r 23300
```



```
Run Conditions for channel 0:
        start time = 1.70007e+09
        data type = np02_coldbox
        Run Number/sofw = 23300
        upload time = 1.70007e+09
        software version = fd-v4.2.0-c6
        stop_time = 0
        buffer = 0
        ac_couple = 0
        run type = TEST
```