

# DUNE/ProtoDUNE Databases

Norm Buchanan for DB Group

*(thanks to Ana Paula, Hajime, and Alex Rahe for slides)*

Colorado State University



# DUNE Database Systems

While I am referring to the DUNE database system, most of what I am discussing today pertains to the databases used for ProtoDUNE.

Certain parts of the system will be different for DUNE, specifically the systems feeding the IFBeam and Slow Control backend databases)

The DUNE database system comprises several backend databases (DAQ config, Slow Control, IFBeam, Calibration, and Data Quality).

All of these backend databases feed a Master Store (unstructured) database (UconDB), and a subset of information from this master store is then copied into a relational database (Conditions DB) that provides an interface for offline users. This approach was taken to enable the greatest degree of flexibility – the UconDB doesn't require a schema *a priori*.

Additionally, a hardware database is used for ProtoDune and DUNE to store information regarding hardware and related testing data.

# Metadata and conditions data

- **Metadata:** all information that describes the data
- **Conditions metadata:** non-event data required to correctly reconstruct or process detector event data, subset of metadata

Conditions data examples:

- Run configuration parameters
- Detector calibration and alignment data
- Monitoring information
- Some slow control parameters - high voltage

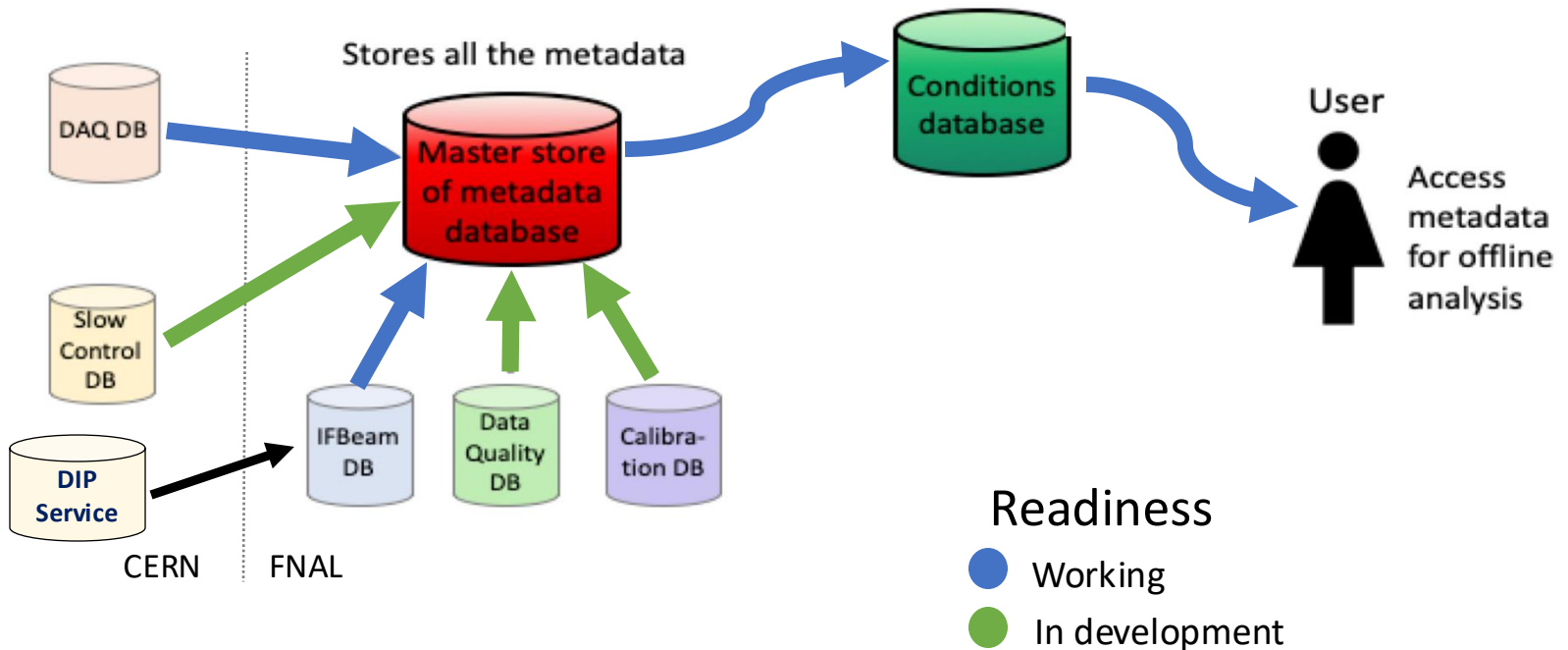
Databases to store metadata



# ProtoDUNE Database System

## Metadata stream

Metadata is sent from all databases to the Master store of metadata (UconDB)



# Current DB Projects

The following projects are currently under development (details given later in this presentation):

- Conditions DB (Ana Paula)
- DAQ Configuration (Ana Paula)
- Slow Control (Nilay Bostan)
- IFBeam (Ana Paula)
- Data Quality (Megan Wrobel, Norm)
- Caching (Alex Rahe – until Spring 2025)
- Hardware Database (Hajime and Alex Wagner)
- Documentation (all – using Software Carpentry with David Demuth's assistance)

All of the above have been implemented, in whole or in part, for the horizontal drift detector. We are currently gathering information to do the same for the vertical drift detector. We don't expect significant differences between the two.

# Uncovered DB Projects

The following projects we will need to have completed but haven't had the resources to tackle as of yet:

- Scaling of conditions DB
- Near Detector Databases
- Monitoring and Messaging/Alarms for all databases (partially uncovered)
- Calibration DB (Calib group using the FNAL conditions DB infrastructure)

Conditions DB scaling and ND databases needed for DUNE have a somewhat lower priority than the ProtoDUNE databases.

Given the current level of resources available to the DB group most of these tasks are considered medium priority and we expect that they will be undertaken in 2026.

We are cognizant of the fact that we will need to work DC planning into our thinking.

# Status of Conditions DB

High Priority

The conditions DB for the HD detector is largely complete – 80% level

Tasks that are not as advanced:

- Web interface (50%)
- Defining tables (including VD detector) (50%)
- Monitoring/Messaging (~20%)
- Scaling (0%)

Current effort level is appropriate to complete most tasks,

Monitoring: 25%-50% of a postdoc or advanced graduate student

Scaling: 25%-50% of a postdoc (potentially with student help)

# Master Store of Metadata (UconDB)

## Main characteristics

- Central place that can store all metadata
- Avoids a priori schema
- It store blobs of information

## Implementation

- PostgreSQL database
- It is an unstructured database
- Python API interface
- Command line interface
- Code now on FNAL git

```
cat of record
Run Number: 12000
Packed on Feb 08 03:57UTC

#####
2000/runMeta.json
#####
[{"RUN_NUMBER", "START_TIME", "STOP_TIME", "DETECTOR_ID", "RUN_TYPE", "SOFTWARE_V
RSION"}, [{"12000, "Thu, 04 Nov 2021 19:51:56 GMT", "Thu, 04 Nov 2021 19:53:32
MT", "np02_coldbox", "PROD", "dunedaq-v2.8.1"}]]

#####
2000/tmpmzhogsum/top_config.json
#####

  "np02_coldbox_daq": "/nfs/sw/dunedaq/dunedaq-v2.8.1/configurations/np02_
oldbox_h...",
  "np02_coldbox_wibs": "/nfs/sw/dunedaq/dunedaq-v2.8.1/configurations/np02_
coldbox_wibs"

#####
2000/tmpmzhogsum/np02_coldbox/np02_coldbox_wibs/boot.json
#####

  "apps": {
    "ctrl_wib401": {
      "exec": "daq_application",
      "host": "host_wibapp",
      "port": 3380
    },
    "ctrl_wib402": {
      "exec": "daq_application",
      "host": "host_wibapp",
      "port": 3381
    },
    "ctrl_wib403": {
      "exec": "daq_application",
      "host": "host_wibapp",
      "port": 3382
    },
    "ctrl_wib404": {
      "exec": "daq_application",
      "host": "host_wibapp",

```



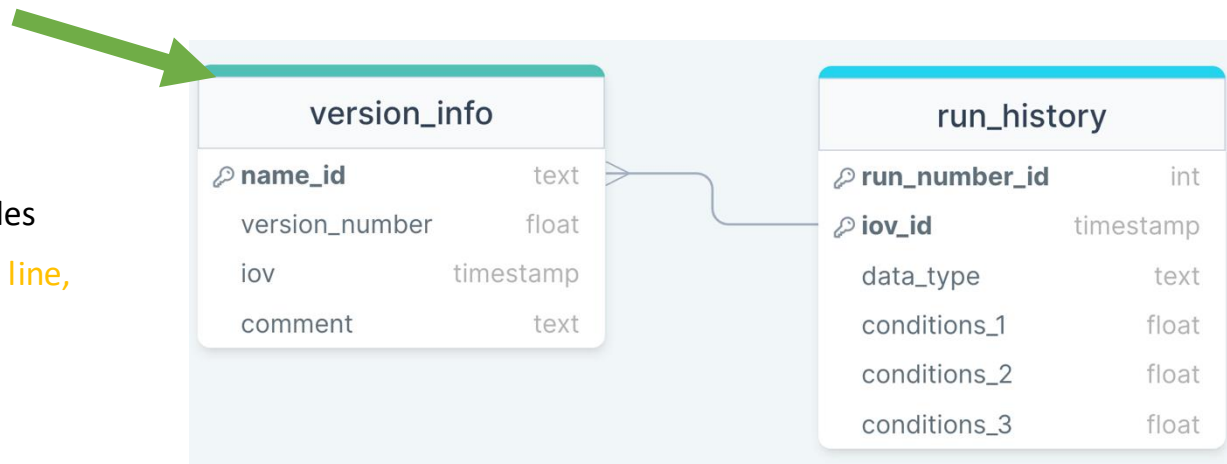
# Conditions database

## Main characteristics

- Store subset of metadata, conditions data of each run
- Needs to have a schema
- It store metadata in table

## Implementation

- PostgreSQL database
- Structured database with tables
- Interfaces: Python, command line, and Metacat
- Code now on FNAL git



Database schema, where the conditions\_# payloads represent all the conditions parameters in the database

# Status of Run Conditions table

High Priority

The DAQ config DB for the HD detector is complete within requested modifications (eg. changing format from JSON to XML) – 70% level.

Tasks needing effort:

- Modifications to table-specific scripts (90%)
- Define table parameters (70 %)
- Add specific info, like slow controls

Current effort level is appropriate to complete most task

# Run conditions table

Metadata	tv (run)	tr	data_type	upload_time	start_time	stop_time	run_type	detector_id	software_version	data_quality
Unit	N/A	Unix	N/A	Unix	Unix	Unix	N/A	N/A	N/A	N/A
Example	25034	1713497099.738875	np02_coldbox or np04_hd	1713497099.7388604	1713268519.0	1713269109.0	PROD	np02_coldbox or np04_hd or np02_hermes_WIB_conf	fddaq-v4.4.0-rc3-a9	good or bad
Comment		Used for versioning					data_stream			offline good runs

Run Conditions table - continued

ac_couple	baseline	buffering	enabled	gain	gain_match	leak	leak_10x	leak_f	peak_time	pulse_dac
N/A	N/A	N/A	N/A	mV/IC	N/A	pA	N/A	pA	us	N/A
dc_coupling or ac_coupling	2	0	True	14.0	True	500.0	False	None	2.0	0
	0 (900 mV), 1 (200 mV), 2 (200 mV collection, 900 mV induction)	0 (no buffer), 1 (se buffer), 2 (sedc buffer)	True of FEMB should be configured and read out by WIB	Options: 14, 25, 7.8, 4.7 mV/IC	Enable pulser DAC gain matching		Multiply leak current by 10 if true	final leak value	Channel peak time selector	Pulser DAC setting [0-63]

Run Conditions table - continued

strobe_delay	strobe_length	strobe_skip	test_cap	adc_test_pattern	cold	detector_type	pulser
N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
255	255	255	False	False	False	wib_default	False
64MHz periods to skip after 2MHz edge for strobe (pulser offset 0-255)	Length of strobe in 64MHz periods (pulser length 0-255)	2MHz periods to skip after strobe (pulser period 0-255)	Enable the test capacitor	True if the COLDADC test pattern should be enabled	True if the front end electronics are COLD (77k)	Options: WIB default, upper APA, lower APA, CRP	True if the calibration pulser should be enabled

Run Conditions table - continued and future data

beam_momentum	beam_polarity	detector_hv	wire_bias_g	wire_bias_u	wire_bias_x	lar_purity	lar_top_temp_mean	lar_bottom_temp_mean
GeV/c	N/A							
+5	positive or negative							
indirectly calculated using magnet MBPL.022.692 current	looking at magnet MBPL.022.692 current							

# Status of DAQ Configuration DB

High Priority

The DAQ config DB for the HD detector is complete within requested modifications (eg. changing format from JSON to XML) – 70% level.

Tasks needing effort:

- Modifications to API (85%)
- Working with DAQ group to obtain information not in DAQ online database (70 %)
- Monitoring/Messaging (~30%)

Current effort level is appropriate to complete most tasks, but a person working on monitoring would be helpful

Monitoring: 25%-50% of a postdoc or advanced graduate student

# Status of Hardware DB

High Priority

A production version of the HWDB is available and in use by some of the hardware consortia (90% complete) (planning underway for a person or two to facilitate interfacing between CS&C DB group and hardware teams).

Tasks that ongoing:

- Development based on feedback (80%)
- Training (75%)

Current effort level is appropriate to complete tasks – will need a modest level of support for maintaining the DB through the construction and commissioning of DUNE detectors (ND and FD) and somewhat less longer term.

# Status of the HWDB and helper applications

- The two PostgreSQL based databases, production version and development version, have been up&running stably.
- Various functionalities have been added/modified in the last ~two years. (see the next 2 pages)
- It is ready to accept massive inputs.  
Some groups have uploaded sizable data to the development version.
- However, not all consortia have tested against the dev. version, yet.  
And so, obviously, almost none have been uploaded to the pro. version.

# the HWDB - 1

- Offers the WEB-UI (intuitive) and the REST API (for massive uploads).
  - It can store;
    - ▶ each component spec
    - ▶ the corresponding location (and its received time)
    - ▶ links to its parent and/or daughter components, if any
    - ▶ the corresponding images, and csv files and spreadsheets.. etc.
      - The limit on sizes/types of these files are to be determined.
    - ▶ the corresponding (QC/AC) test results
- along with their histories and their searching functionalities.

# Plans going forward on HWDB - 1

- Just had the 2nd HWDB tutorials in this past July.

And we have the corresponding training site:

<https://dune.github.io/computing-HWDB/>

Most of the HWDB liaisons from each consortia attended.

Yet, do not see many users are even testing with the dev. version, yet.

- Recently created a “QuickStart” user guide page to use the Python-based app: <https://dune.github.io/computing-HWDB/quickstart/>

It provides an example spreadsheet (acts as a general purpose template), which can be downloaded and be used “as it is”.

Hopefully this would encourage users to start to test uploading.



## Plans going forward on HWDB - 2

- We are also planning to start to communicate with individual HWDB liaisons and find out what statuses of their preparations to upload their data and their needs, such as;
  - ▶ our helps on mapping their component (PID) hierarchy (system -> subsystem -> component types) and producing their DB schema,
  - ▶ requests to update the apps or to provide more specific examples/templates.
  - ▶ or possibly requests to modify/add the HWDB functionalities (hopefully we already have the DB ready for their needs by now)

# Plans going forward on the apps

While we will keep improving, updating and maintaining the existing apps, we will be also searching for;

1. an easier way to deploy the two iOS-based app,
2. possible needs/demands for Android-based app,
3. and also needs for GUI version of the Python-based app.

As for the item 1, we currently deploy the apps through Apple's TestFlight.

This requires to go through Apple's non-public app store.

It has caused issues previously, when we wanted to deploy our new release (e.g., bug fix) but the app store was very crowded (slow), which tends to happen when there is a new version of Xcode.

As for the items 2 and 3, we will do them only if there is a demand.

# Status of DB Caching

High Priority

Project is in its initial stages ( $\approx 10\%$ )

Tasks that need effort:

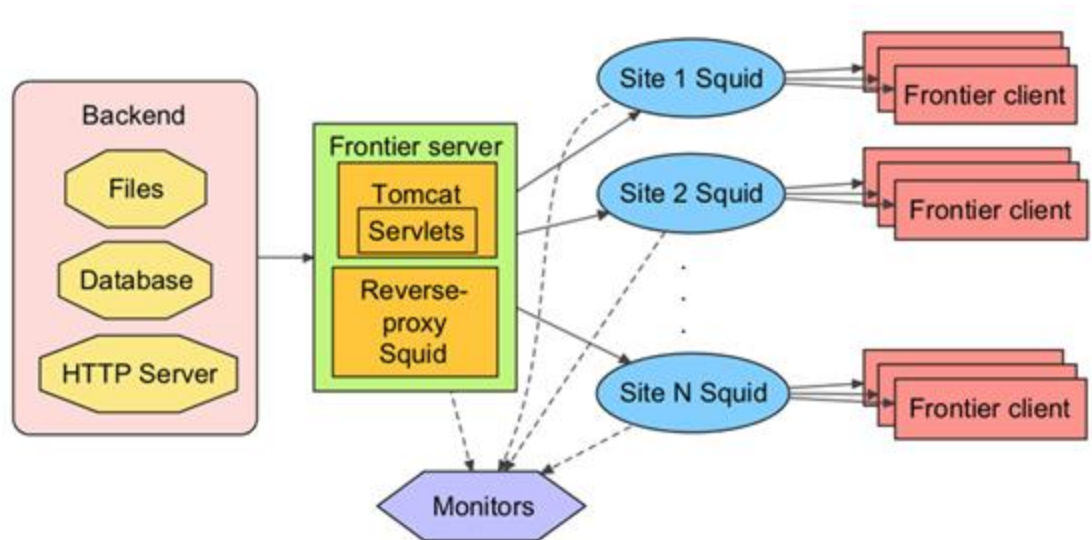
- Requirements (Manager)
- Implementation (Expert/Postdoc/GS)
- Initial Testing (Expert/Postdoc or GS)
- Scaling (Expert/Postdoc)

Current effort level is insufficient to accomplish this project – would like additional support starting in CY 2025 is needed

# Database Caching

## Frontier for DUNE

- Used as a caching system for ATLAS, CMS, and CDF
  - Scalable
- Customizable architecture
- Load balancing
- Data compression
- Open Source



# Database Caching

## Frontier for DUNE

- Frontier servers hosted on Metis cluster at NIU
  - 1 Squid server and 1 Tomcat server
  - Can use Frontier client to access calibration database
- DUNE calibration database access is overseen by a modified libcurl library in the ifdhc project
- Currently working to add the ability to resolve the Frontier client URL to this modified library
- Future Steps: Testing
  - Simulate many jobs accessing calibration data simultaneously

# Status of Slow Control DB

High Priority

Some development has been done building on work Lino did last year. Path from the backend system to the UconDB has been demonstrated with a single instrument. **Currently not working**

Tasks that need to be done:

- Fix backend system (Expert/Postdoc)
- Granularity Studies (Postdoc/GS)
- Complete list of instrument (50%)
- Monitoring/Messaging (~20%)
- Scaling (0%)

Current effort level is appropriate to complete most tasks,

Monitoring: 25%-50% of a postdoc or advanced graduate student

Scaling: 25%-50% of a postdoc (potentially with student help)

# Slow Control REST API

- Is deployed on a DUNE CERN VM
  - Ana Paula, and other DUNE members, have access to the VM
  - To get access we contact AP
- It's **not currently working**
  - Lino wrote it, and now Ana Paula in contact with David to fix it
  - A password has to be updated every ~6 month
  - Might need to be deployed again

# Status of Other Database Projects

IFBeam

High Priority

Mostly in place – tasks similar to those for Slow Control DB

Tasks that need to be done:

- Granularity Studies ( O(1%) Postdoc/GS)
- Complete list of desired instrument (50%)
- Monitoring/Messaging (50% Postdoc/GS)

Current effort level is appropriate to complete most tasks, but having some additional effort for monitoring and granularity studies.

DQ DB

Medium/High Priority

New effort so not much to say yet. This should not be a particularly large task. Current effort level is appropriate and we expect to have it in place early in CY 2025.



# Conclusions

Most of the projects are well under development with the effort needed in place.

## High Priority:

- Conditions database (~60%) – might need extra effort for scaling
- Run conditions table (~70%) – current effort level is appropriate
- DAQ configuration database (~70%) – current effort level is appropriate
- Hardware database (~90%) - current effort level is appropriate, modest level of support for maintainance
- Caching (~10%) – additional support is needed
- Slow controls (~30%) – might need extra effort for scaling
- IFBeam (~70%) - might need extra effort for scaling

## Medium / Low

- HSF conditions database (~30%) - additional support is needed or moved to a latter time
- Calibration tables
- Documentation and training

# Backup Slides

# Conditions database - interfaces

The documentation for the python API, command Line interface, and the REST API was recently updated <https://fermisda-condb2.readthedocs.io/en/stable/rest.html>

- URL: [https://dbdata0vm.fnal.gov:9443/dune\\_runcon\\_prod](https://dbdata0vm.fnal.gov:9443/dune_runcon_prod)
- Folder/Table name: pdunesp.run\_conditionstest
- User and password are not needed for getting the data, if you need to upload contact DB group

## ConDB2 documentation

[PREVIOUS](#) | [NEXT](#) | [INDEX](#)

### ConDB2 Client Installation

The recommended way to install the client application.

#### Installation from github

```
$ pip install git+https://github.com/fermisda/condb2.git
```

Once the client is installed, ConDB2 library module will be available as `condb2`:

### TABLE OF CONTENTS

Contents:

ConDB2 Client Installation

ConDB2 Command Line Interface

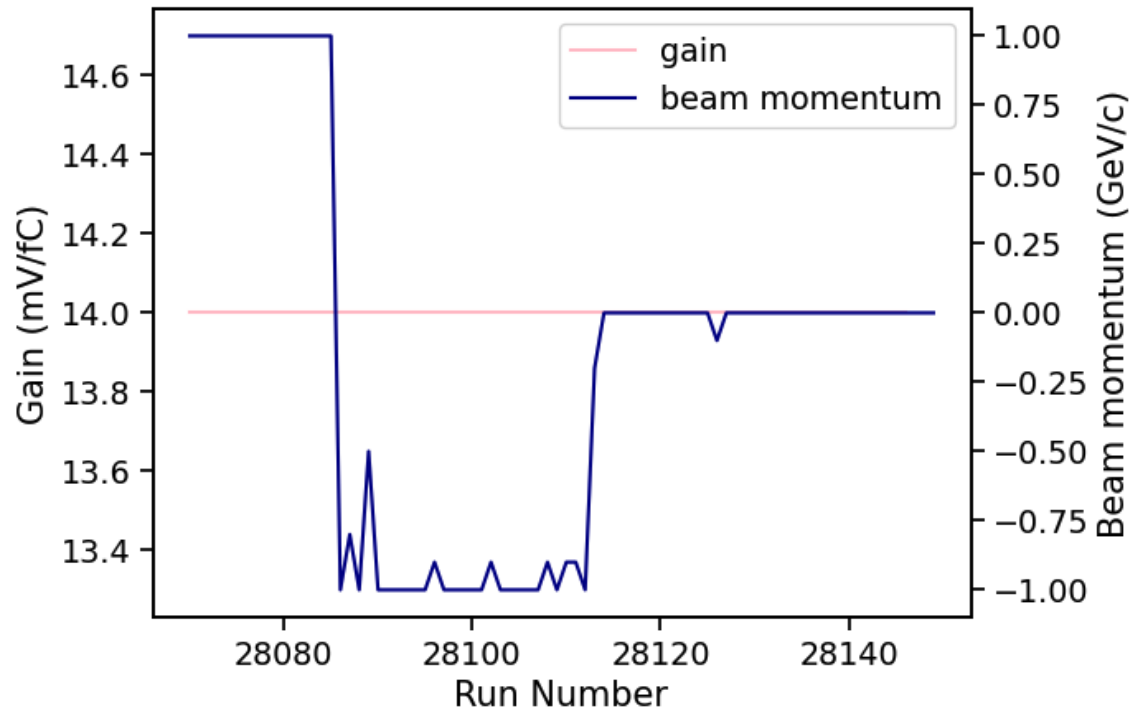
ConDB2 Web Access Python API

ConDB2 Direct Access Python API

ConDB2 REST API

# Conditions database – Outputs

- The python API and the rest API can **return the data** from the tables in **JSON** or **csv** format.
- With the python API it is possible to get a **selection of runs that comply with one or multiple conditions** on the table parameters, like:
  - Beam momentum = -1 GeV
  - Gain = 14



# Run conditions table – C++ interface

[https://wiki.dunescience.org/wiki/Run\\_Conditions\\_Table#Table\\_information](https://wiki.dunescience.org/wiki/Run_Conditions_Table#Table_information)

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

Include header file

Declare Run Condition table

```
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();
```

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/software = " << rc.run_number  
<< "\n\tupload time = " << rc.upload_t  
<< "\n\tsoftware version = " << 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

# Run conditions table – ART service

- Service name: `pdune_runconditions`
- Fhicl file inputs:
  - Table URL = [https://dbdata0vm.fnal.gov:9443/dune\\_runcon\\_prod](https://dbdata0vm.fnal.gov:9443/dune_runcon_prod)
  - Table Name = `pdunesp.run_conditionstest`
  - Run Number = 28000
  - Run Number 1 = 0
  - DB tab = 'v1.1'

## Examples and documentation

- Wiki  
[https://wiki.dunescience.org/wiki/Run\\_Conditions\\_Table#Table\\_information](https://wiki.dunescience.org/wiki/Run_Conditions_Table#Table_information)
- 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`

# Conditions database – Metacat interface

[https://wiki.dunescience.org/wiki/Run\\_Conditions\\_Table#Table\\_information](https://wiki.dunescience.org/wiki/Run_Conditions_Table#Table_information)

Goal: Look for data files using parameters from the run conditions table as filters

The following query is an example of how to write the queries:

```
filter dune_runshistdb() (files from hd-protodune:hd-protodune_25016)  
where runs_history.run_type = PROD
```

filter name

filter using the run condition parameter

dataset of files to look from

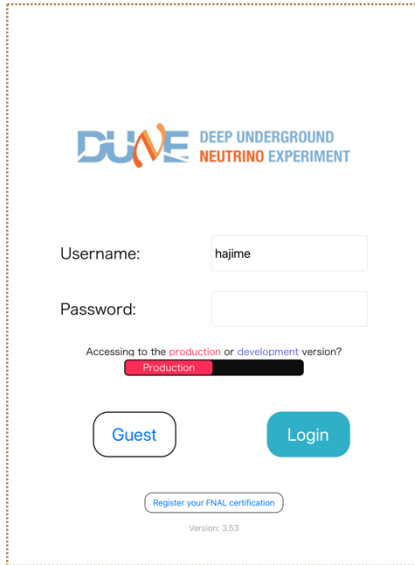
New filters can be made for other tables in the conditions database

## the HWDB - 2

- **Users do need to define DB schema for each types in YAML (e.g., JSON). However, defining an empty value, such as “DATA: {}”, would allow to have any JSON structure inside of its blob without defining any scheme. This feature could be useful when/if one needs to transfer data from a JSON-oriented 3rd database.**



# 3 helper applications (so far...)

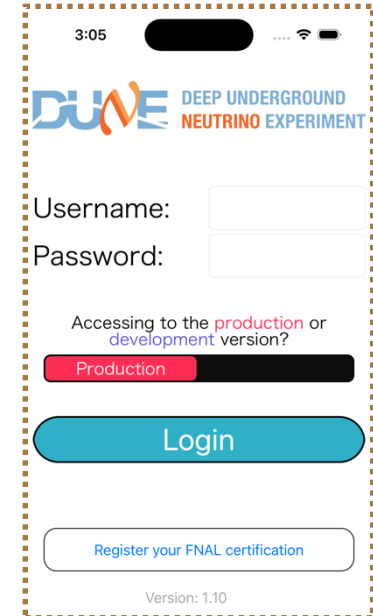


The screenshot shows the DUNE web interface on a desktop browser. At the top left is the DUNE logo (DEEP UNDERGROUND NEUTRINO EXPERIMENT). Below it are input fields for Username (containing 'hajime') and Password. A toggle switch is set to 'Production'. There are 'Guest' and 'Login' buttons, and a link to 'Register your FNAL certification'. The version number '3.53' is displayed at the bottom.



```
DUNE HWDB Utility v1.2.2.dev.2024.10.24b

Configuration Summary:
=====
profile:      default (default)
REST API:    dbwebapi2.fnal.gov:8443/cdbdev (development)
certificate:  pem
cert info:   Hajime Muramatsu (hajime3)
cert status: Expires in 24 days
server check: REST API 'whoami' returned Hajime Muramatsu (hajime3)
```



The screenshot shows the DUNE web interface on an iPhone. At the top is the status bar with the time '3:05' and battery level. Below is the DUNE logo. There are input fields for Username and Password. A toggle switch is set to 'Production'. There is a large 'Login' button and a link to 'Register your FNAL certification'. The version number '1.10' is displayed at the bottom.

- iPad version: Provides intuitive UI, access to drawings/assembly procedures, scans bar/QR-codes.
- Python-based app: (so far) runs at command-line. Takes spreadsheets as inputs and upload them.
- iPhone version: Useful to quickly scan bar/QR-codes and update location info in the HWDB. Also provides stored component info.