

## dunetpc - Feature #19927

### Programmatic interface to run conditions

05/11/2018 09:35 AM - David Adams

<b>Status:</b>	Closed	<b>Start date:</b>	05/11/2018
<b>Priority:</b>	Normal	<b>Due date:</b>	
<b>Assignee:</b>	David Adams	<b>% Done:</b>	0%
<b>Category:</b>		<b>Estimated time:</b>	0.00 hour
<b>Target version:</b>			
<b>Description</b>			
As discussed at <a href="https://wiki.dunescience.org/wiki/ProtoDUNE_run_configuration">https://wiki.dunescience.org/wiki/ProtoDUNE_run_configuration</a> , it would be useful for dunetpc tools to have access to information about run conditions such a preamp gain and shaping time. Many of these values are implicit or already available through fcl-configured services but it would be preferable not to have to change the jobs fcl configuration based on run or to at least have means to check the values in that configuration.			

### History

#### #1 - 05/11/2018 09:36 AM - David Adams

- Subject changed from Programmatic interface to run conditions to Programmatic interface to run conditions

#### #2 - 05/11/2018 09:40 AM - David Adams

My proposal is to define a tool interface that provides the fields listed on the above link as a function of run number.

This would be followed by a simple implementation that reads a text file describing some recent runs of interest.

Future implementations could be more sophisticated and read XML files (from local or archived directories) or use a database.

#### #3 - 05/11/2018 10:17 AM - Tracy Usher

David Adams wrote:

My proposal is to define a tool interface that provides the fields listed on the above link as a function of run number.

This would be followed by a simple implementation that reads a text file describing some recent runs of interest.

Future implementations could be more sophisticated and read XML files (from local or archived directories) or use a database.

FWIW MicroBooNE is obtaining this information from databases, you can find the generic interfaces in the LArEvt package. It requires populating the database(s) but satisfies the requirement of not having to modify the fhicl for any given job.

#### #4 - 05/11/2018 10:28 AM - David Adams

I see two sensible approaches to the interface:

1. Fixed schema. The interface provides a dedicated method for each field, e.g.

```
int run = 123;
std::string scry = mytool.cryostat(run);
float gain = mytool.gain(run);
...
```

2. Arbitrary schema. The interface provides a [DataMap](#) holding the schema values mapped by name, e.g.

```
int run = 123;
DataMap rundat = mytool.data(run);
std::string scry = rundat.getString("cryostat");
float gain = rundat.getFloat("gain");
...
```

The former provides more type safety with less flexibility. If a field is added, all clients have to be recompiled in the first case.

Or we could have a tool that does both--i.e. has dedicated methods for fixed schema and also returns a DataMap with those and other values.

Or we might introduce a data schema type to hold the data, e.g.

```
int run = 123;
RunData rundat = mytool.data(run);
std::string scry = rundat.cryostat();
float gain = rundat.gain();
...
```

**#5 - 05/11/2018 10:53 AM - Brett Viren**

Orthogonal to issues of schema:

I think DAQ needs to be involved. Given the large number of settings changes needed to span (gain) x (shaping) x (DAC charge) x (DAC time offset) and the relatively quick time needed to collect data at each setting it may not be feasible to start a new run for each setting. If so, some interface is needed to pull the settings directly from each event - and some DAQ mechanism is needed to place the settings info in the data to start with.

**#6 - 05/11/2018 10:56 AM - David Adams**

Tracy:

I had a look at larevt. It is calibration rather than run conditions, e.g. a possibly time-dependent gain for each channel where I just want to know the gain setting for a run.

But thanks for pointing this out. There are some run variables of interest, e.g. temperature or purity, that we may want to derive from such conditions data before or instead of accessing it from a run table.

**#7 - 05/11/2018 11:13 AM - David Adams**

Brett:

I agree that the information had better be in the event if the value is going to change event to event. For now, I am inclined to have this tool return a valid value for fields that do not change during the run and leave clients to look elsewhere if an invalid value is returned.

I added the DAC charge to the schema but it is a good point that we probably also want the DAC time offset.

Can we start a new subrun when the values change or is that the same time as starting a new run?

I believe the DAQ time offset is w.r.t. a tick, i.e. the 2 MHz clock. Are we also able to specify this w.r.t. the trigger clock, i.e. a 2nd offset?

**#8 - 05/11/2018 02:02 PM - David Adams**

I have decided to follow my last option and add a class to describe the rune data. The class RunData is in `dunetpc/dune/DuneInterface/Data`.

**#9 - 05/15/2018 10:22 AM - David Adams**

- *Status changed from New to Closed*

A tool to read run data from fcl files in place: `dune/DuneCommon/Tool/FclRunDataTool` and fcl descriptions for a few selected protodune coldbox runs are in place in `dunetpc/fcl/protodune/fcldirs/rundata/protodune`. It is straightforward to add data for additional runs.