Concept for the Interface between the $\overline{\text{P}}$ANDA Detector Control System and the Front-End Electronics

Tobias Triffterer,
Institut für Experimentalphysik I,
Ruhr-Universität Bochum

Git revision: 68bc99a449387d88df6c367f1d1eb5b337ff5235
Revision date: 2018-05-23 13:32:47 +0200
Git branch: master

This document is still a draft and therefore subject to change. Do not base your developments on its content until it has been approved!

1 Introduction

The operation of the $\overline{\text{P}}$ANDA detector requires the close cooperation of several common systems, among them the Detector Control System (DCS), the Front-End Electronics (FEE), and the Data Acquisition (DAQ). This concept describes the interfaces between these systems and their shared data structures.

The basic idea contains the following two principles:

1. The DCS gives a command via a dedicated EPICS server (called Input/Output Controller or IOC in the context of EPICS) that instructs DAQ and FEE to load a certain configuration that is identified by a unique identifier which can be an integer, a hash, a

GUID, or something equally suited. After the configuration loading process is complete, the DAQ and FEE report a status back to the DCS, but the DCS is not involved in the data transfers necessary on SODAnet and similar infrastructures to execute the loading of the configuration values.

2. The DCS provides a central configuration database where DCS-related values like voltages or temperature setpoints are stored together with DAQ/FEE-related values like ADC pedestals. The DCS provides an API which any DAQ or FEE application can use to query data from this database. The configuration database will as well be used by analysis clients to retrieve the status of the $\overline{\text{P}}$ANDA detector as it was at the point in time when the data currently under analysis was taken.

The aforementioned unique identifier is intended to be used by the various applications involved in this process to unambiguously identify a configuration dataset as defined in section 4. These datasets will also get human-readable names, so that the users (except eventually the database administrators) do not have to deal with the unique identifier themselves.

2 Configuration Namespaces

The groups responsible for a $\overline{\text{P}}$ANDA sub-detector are also responsible for maintaining the configuration of this sub-detector in the central configuration database. The reason for this responsibility assignment is that the knowledge of the experts for said sub-detector is required to maintain its configuration properly.

To make sure that no conflicts with e. g. parameter names occur, the configuration data will be split into namespaces with each sub-detector having its own namespace. Names given to parameters (see section 4) only have to be unique within one namespace, but not within all of $\overline{\text{P}}$ANDA.

Each sub-detector will also get its own list of configuration datasets and the experts of each sub-detector can freely decide which condfiguration dataset shall be used for which run and operation mode (see section 4).

This means that the DCS will issue one "load configuration" command per sub-detector (that take part in the run), this requires one EPICS record for such commands per sub-detector. These records may all be hosted by a single dedicated EPICS server (called Input/Output Controller or IOC in the context of EPICS) or reside on different IOCs, eventually one per sub-detector. Due to the distributed nature of EPICS, this does not make any difference from the perspective of the operator in the control room.

3 Initiating Configuration Loading

The EPICS record that is used to initiate the configuration loading process will be equipped with a custom device support. Upon a write access on this record, the device support code

takes the necessary steps to instruct DAQ and FEE to start the configuration loading process. The value written to the value field ("VAL") of the record is the unique identifier (see section 1) of the configuration dataset to be loaded for the specific subsystem.

The method of communication used between the device support code and the DAQ/FEE code still has to be defined and needs input from the DAQ and FEE experts. The following methods are possible:

- Method A: The DAQ/FEE group provides a shared library that the device support can use. This library handles all internals of the DAQ/FEE system.
- Method B: The DAQ/FEE groups create a DAQ control daemon that is running on the same PC or another one. The device support communicates with this daemon via network (if it is on a different PC) or via inter-process communication like D-Bus, Sockets, etc. (if it is on the same PC).

The EPICS device support code itself will be written by the DCS group, the library or daemon will be provided by DAQ/FEE experts. The Application Programming Interface (API) between these components has to defined and agreed upon by both DCS and DAQ/FEE developers, but this can happen at a later date.

The device support code will forward the configuration dataset unique identifier to the DAQ/FEE code. The DAQ/FEE code then has to use this identifier to query the required parameters from the central configuration database (see section 4). EPICS and the rest of the DCS are not involved in the steps necessary to program the front-ends appropriately. This sequence is shown as a block diagram in figure 1.

After this programming process is finished, the DAQ/FEE code has to report back to the DCS with a status. This status can be either "Successful" or it can be an error code. There should be a list of defined error codes for this interface, but this list can be compiled while the actual programming work is done.

In addition, the EPICS device support controlling the configuration loading process will also trigger DCS-internal processes to load configuration parameters for which the DCS is responsible like voltage or temperature settings. To accomplish this, the DCS code for each sub-detector will also make requests to the central configuration database (see section 4). This process is not explained further in this document as it focuses on the interface between DCS and DAQ/FEE. The DCS-internal processes will also report a status back to the configuration device support.

If there is an error reported (from DAQ/FEE or DCS), the device support code will set an error status on the record, this will cause an alarm in the control room. The run control software can query this alarm status and will only proceed with the start of data taking if all $\overline{\text{P}}$ANDA sub-detectors report their initialization as "successful".

If there is no response regarding the status after a certain amount of time, the DCS will assume that there is a problem and report a timeout error to the run control, this means that the process of starting a run will be aborted. The maximum time to finish the initialization process has to be agreed upon by the involved $\overline{\text{P}}$ANDA groups. In case of long initialization

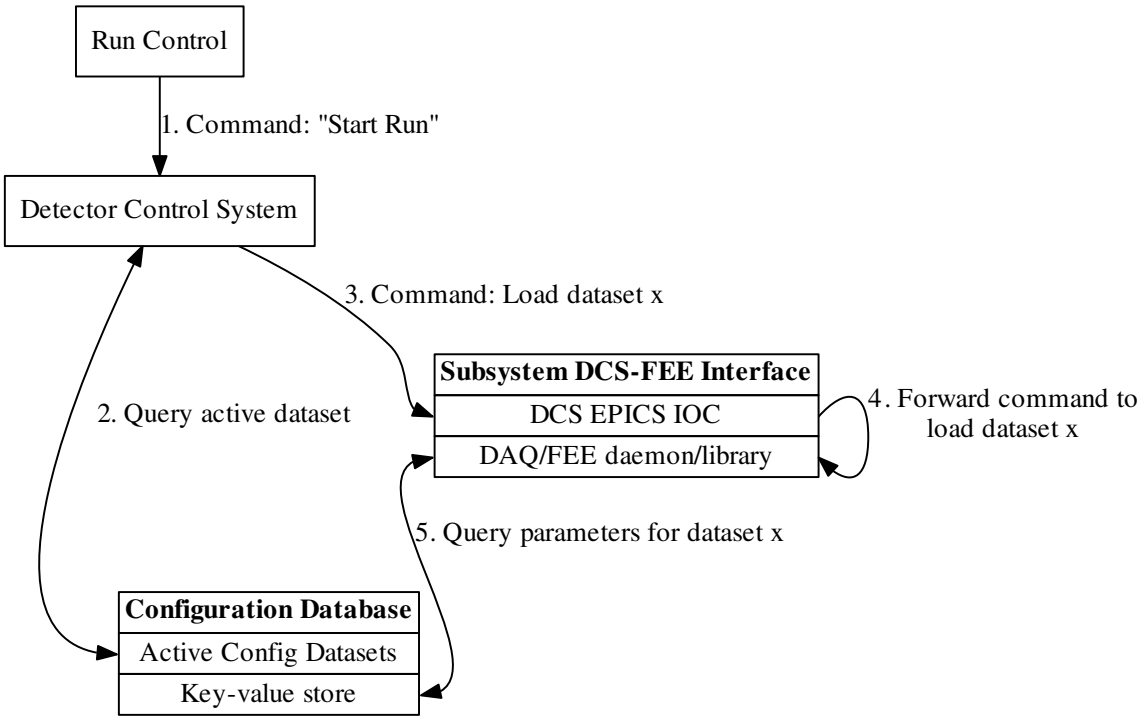

Figure 1: Visual representation of the steps required to load the DAQ/FEE parameters according to this concept. (The feedback from the DAQ/FEE to the DCS has been omitted to keep the graph clear.)

processes, one may implement a "still running" status message, so that the DCS can see that the processes on the DAQ/FEE side did not crash.

During a run, the DAQ/FEE applications should keep a copy of all the configuration parameters for that run in memory, in some cases the data may also be stored in the memory of dedicated hardware like a data concentrator or compute node. If a front-end device needs a reset and/or reprogramming during a run, the data is available immediately without having to make any requests to the DCS or the central configuration database. This ensures that the downtime of that particular front-end device is kept as short as possible.

Nevertheless, the information that reprogramming was necessary should be recorded – on the one hand to explain artifacts of a partial detector downtime during analysis, on the other hand to see how frequently such reprogrammings are necessary. The implementation of this recording should be discussed. It may be part of the (meta) data recorded during the run or the frontend device may notify the DCS so that this enters the DCS archive database. A third option would be to have a dedicated API for reporting such issues and a dedicated section in

the central database to keep these recordings.

4 Configuration Database

The configuration database is the central storage location for any information needed to configure the parts of the $\overline{\text{P}}$ANDA detector to make it ready for data-taking. It has two main usage scenarios:

1. The DCS, the DAQ, and the run control use it in the process of starting a run to configure the $\overline{\text{P}}$ANDA detector for a given operation mode. After starting a run, the software records the configuration used in the database.

2. The analysis clients use the database to learn which configuration parameters were used during the run they are currently analyzing, so they can act accordingly and e. g. ignore anomalies due to a disabled readout channel.

The configuration data itself will be organized in the form of a key-value store. As explained in section 2, each $\overline{\text{P}}$ANDA subsystem gets its own namespace and therefore every detector group can freely assign its key names without having to worry about possibly breaking the configuration of another system. For the content of the key-value store, a list of data types like integer, double and string will be available. Further data types may be added upon request from any $\overline{\text{P}}$ANDA group if the basic data types are not sufficient for them.

A defined collection of key-value pairs shall be known as a **configuration dataset**. A configuration dataset also exists in the namespace of its $\overline{\text{P}}$ANDA subsystem, so the $\overline{\text{P}}$ANDA groups can create datasets as they like without harming any others. A configuration dataset must be complete in the sense that it contains all the information required to make the subsystem it belongs to ready for data-taking. Each configuration dataset will be assigned an unique identifier (integer, hash, GUID, etc.) by the database management system or the DCS database operation software.

In addition to the unique identifier, each dataset shall be given a human-readable name. Alongside with its creation date and possibly other data, this name will be used when a user has to select a dataset from a list. The unique identifier is used internally by the database system and the DCS and DAQ/FEE applications, but usually not shown to the user. In particular, the user will not be required to memorize some numbers or hashes to operate the $\overline{\text{P}}$ANDA detector properly.

To be able to reconstruct the state of the $\overline{\text{P}}$ANDA detector during analysis (both online and offline), all configuration datasets are by definition read-only. Once a configuration dataset has been created and stored in the database, nobody may be allowed to change it. If a change is needed, it has to happen in a way that the modified version is added as a new configuration dataset and gets a new unique identifier.

Configuration datasets can then be assigned to an **operation mode**. Operation modes are global, i. e. they are not bound to subsystem namespace but they are the same for everyone.

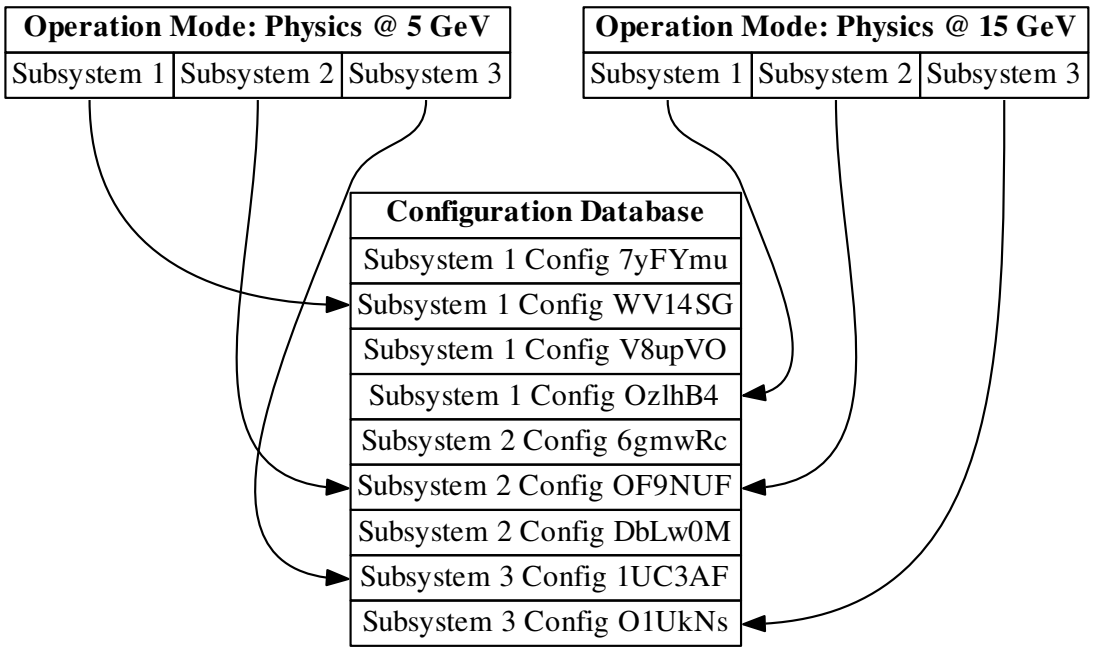

Figure 2: Block diagram showing the assignment of configuration datasets to operation modes.

The operation mode describes types of data-taking runs. There can be test runs and physics runs and maybe more. For the physcis runs, it would be smart to differentiate by beam energy range. If we run at e. g. 5 GeV/c, there will not be a 12 GeV photon so one can increase the gain of the front-end electronics to use the dynamic range of the ADCs properly and thus get a better resolution.

In addition, it is worth to consider having operation modes like "HESR down for time x". If the accelerator is not working, we cannot do physics. But we may use the time in a somewhat meaningful manner by doing some automated maintenance tasks. The EMC for example could take LED pulser runs to check the crystals for radiation damage. If there is a long downtime, one may also consider to activate the annealing LEDs to heal some radiation damage in the EMC scintillator crystals.

Each detector group can assign one of its configuration datasets to each operation mode and they can use the same dataset on different operation modes if they so like. Also, they can change this assignment at any point in time. An example for two operation modes and three subsystems in shown in figure 2.

When a run is about to be started, the DCS software checks the operation mode of the run that shall begin. It will then query the database to see which configuration dataset is assigned

to this operation mode at that very point in time for each subsystem. After this, the DCS will instruct the DAQ and FEE software of each subsystem via the procedure outlined in section 3 to load that specific configuration dataset by specifying its unique identifier. In addition, the DCS itself will take care that DCS-related values like voltages are set accordingly.

The DAQ and FEE software can then use the dataset unique identifier to query the individual parameter it needs from the key-value store in the configuration database and program the devices according to this parameters.

The DCS (or the run control) software makes an entry into the run database (which may be a part of the configuration database) and specifies which configuration dataset was used for each subsystem for this run. The run will here be identified by the run number or any other unique property. This information is also by definition read-only once it has been entered into the database.

During analysis, the user can query the list of configuration datasets used via the run number and then query any needed configuration parameter based on the unique identifiers of the configuration datasets. As the datasets are read-only, the information is guaranteed to be the same as it was during taking the run and writing the data to disk.