

Online Calibration System

T. Yasuda

Fermilab

April 12, 2002

1 Introduction

Periodic calibration of the sub-detector systems is one of the most important tasks during the data taking run. In the D Run II, most sub-detector systems will take special data during the quiet time between stores for calibration, collecting pedestals or measuring gains by injecting electronic charges to the front-end amplifiers. These data are analysed online in the front-end processors in the case of SMT, or in the Level 3 nodes in the case of Calorimeter, or analysed offline in the cases of CFT, CPS, and FPS, which uses a common readout system, and the muon systems. The calibration constants are inserted into the online ORACLE database. These constants will be transferred from the online database to the offline database for later use in the data analysis.

The online calibration runs will be coordinated by a set of software programs, each representing a different aspect of the task, with `coor` controlling the overall running of the DAQ system and the calibration manager (`Calib_Manager`) overseeing the data transfer from the calibration data processor to the calibration database. `Calib_Manager` and the database interface have been developed as common tools for all sub-detectors and run with the standard elements of the data acquisition (DAQ) system, such as `coor`, `comics`, `Collector/Router`. `comics` is the common tool for the download to elements of the DAQ system, such as the front-end crates. The `Collector/Router` selectively sends the special event data from the front-end processors or from the level 3 nodes to the `Calib_Manager`.

When a calibration run is about to begin, the `Calib_Manager` receives a series of commands from `coor` and prepares itself for the run. Once the run starts, the `Calib_Manager` takes over the process. After a front-end processor or a L3 node collects a sufficient number of events, it calculates the calibration parameters, such as pedestals and gains, and sends them to the `Calib_Manager` as a special `Event Message`. The calibration runs for individual crates take place in parallel. The `Calib_Manager` checks if it has received data for all of the front-end crates and therefore the calibration run is completed. After receiving data for all of the crates, the `Calib_Manager` sends a “`force_stop`” message to `coor`. Upon receiving this message, `coor` will stop the run and frees the resources allocated for the calibration run. The `Calib_Manager` forwards the data received from the calibration

data processor in a front-end or a level 3 node to the Validation process (**Validator**) for validation and for database insertion.

Communication between the processes is based on a mechanism implemented in the ITC client-server package. The **Calib_Manager** is an ITC server. The calibration data processors in the front-end, **Validator**, **coor** and the graphical user interface for the **Calib_Manager** (**Calib_Manager_GUI**) are ITC clients. The results of the calibration are transported as a special **Event Message** with a **stream id** assigned by **coor**.

The **Calib_Manager** is a c++ program and contains an ITC server, an ITC Message processor and an **Online_Calibration** object. The **Online_Calibration** object holds all the information about the subsystem calibration in the form of **Subsystem_Onl_Calibration** object as a data member. Since there are more than one detector subsystems, the object is stored as an **std vector**. Each **Subsystem_Onl_Calibration** object holds the status of data processing, the status of validation, and the status of database commit for each crate as **std vectors**. It also contains data on the results of calibration and a summary of the validation for each crate. The states of these objects are set directly or indirectly by receiving messages or data from the front-end processes or the validation process.

The **Validator** acts as an interface to the calibration database and is written in Python. It receives data from the **Calib_Manager** and compares them to the reference set obtained from the database and inserts them to the database. The **Validator** for a detector subsystem is started by the **Calib_Manager** after a “**start_run**” command is received from **coor**. The **Validator** is a client to the **Calib_Manager**. After establishing a connection to the **Calib_Manager**, it sends a request for a list of crates that are being calibrated so that the reference set can be retrieved from the calibration database while the calibration run is in progress. The **Calib_Manager** forwards data from the calibration data processor to the **Validator** one crate at a time, and receives the summary of the result of comparison from the **Validator**. The **Calib_Manager** keeps track of the crates that are calibrated. The **Validator** also performs database commit.

The **Calib_Manager** and the **Validator** are maintained as a **dcvs** package **onl_calib_system**. The low level database access is based on the **dcvs** package **onl_calDbAccess**.

2 Steps of Online Calibration

This section describes the sequence of a typical online calibration run to demonstrate how the process works.

1. Initialization of **Calib_Manager**

The **Calib_Manager** starts out by creating an ITC processor and an ITC server. An **Online_Calibration** object is created with the processor.

A connection to **Collector/Router** is made at this point so that data can be received from it with **stream ids** assigned dynamically by **coor**.

After this, the **Calib_Manager** goes into a wait loop.

2. `coor` communication for a calibration run

In most cases, `coor` has to connect to the `Calib_Manager` as a client to send the calibration run information. The `Calib_Manager` in turn connects to `coor` as a client to be able to send “`force_stop`” run message to `coor` when all the calibration data are collected.

`coor` sends a series of messages describing the calibration configuration to the `Calib_Manager` at the beginning of a calibration run. Upon receiving these messages, `Subsystem_Onl_Calibration` object is created in the `Online_Calibration` object to receive data from the front-end processes or the L3 nodes.

3. Initialization of `Subsystem_Onl_Calibration` object

The status of `Subsystem_Onl_Calibration` is set to `READY_FOR_RUN`, when it is created. The processing status is set to `C_READY_FOR_RUN` for each crate. The validation status is set to `C_READY_FOR_VALIDATION` for each crate if the validation is requested. If not, it is set to `C_UNKNOWN_VLDT_STATE`. The commit status is set to `C_UNKNOWN_COMMIT_STATE` for each crate.

Upon receiving a “`start_run`” command from `coor`, the status of `Subsystem_Onl_Calibration` is set to `RUN_IN_PROGRESS` and the processing status is set to `C_RUN_IN_PROGRESS` for each crate.

4. Receiving data

Upon receiving the results of calibration, the processing status is set to `C_RUN_FINISHED` for the crate. It is also checked if the data from all of the crates are received or not. If all of the crates have sent data, the status is set to `RUN_FINISHED`, `_end_calib` time is set and `Calib_Manager` sends a “`force_stop`” command to `coor` to stop the run.

5. Initialization of `Validator`

In the meantime, the `Validator` is started by `Calib_Manager`, after receiving a “`start_run`” command from `coor`.

The `Validator` creates a `Subsystem_Validation` object and connects to the `Calib_Manager` as a client.

The `Validator` asks the status of `Subsystem_Onl_Calibration` object to the `Calib_Manager` by sending a `VALIDATION_INQUIRY` message. It keeps sending this message until the `Subsystem_Onl_Calibration` object is ready for validation or database commit.

6. Preparation for validation in `Calib_Manager`

Upon receiving a `VALIDATION_INQUIRY` message, the status of the `Subsystem_Onl_Calibration` object is set to `READY_FOR_VALIDATION`, if validation is requested by a `coor` message and if the status is `RUN_FINISHED`. The list of crate is sent to the

validation process as a `START_VALIDATION` message, if the status is `RUN_IN_PROGRESS` or `RUN_FINISHED`.

If validation is not requested, the status of `Subsystem_Onl_Calibration` is set to `READY_FOR_COMMIT` and the status of each crate is set to `C_READY_FOR_COMMIT`.

7. Data request by Validator

The status of the `Subsystem_Validation` object in the `Validator` becomes `READY_FOR_VALIDATION`, when a `START_VALIDATION` message is received, regardless of the readiness of `Subsystem_Onl_Calibration` object in the `Calib_Manager`. When the status of the `Subsystem_Validation` object becomes `READY_FOR_VALIDATION` or `READY_FOR_COMMIT`, and it is not in the state of `VALIDATION_FINISHED`, nor `C_VALIDATION_IN_PROGRESS`, the `Validator` sends a `DATA_REQUEST` message to the `Calib_Manager`, if validation is requested.

If validation is not requested, the status of the `Subsystem_Validation` object is immediately set to `READY_FOR_COMMIT` and a `DATA_REQUEST` message is sent to the `Calib_Manager`, until the status of the `Subsystem_Validation` object becomes `COMMIT_FINISHED`.

8. Data transfer for validation

Upon receiving a `DATA_REQUEST` message, the `Online_Calibration` object sets the status of the `Subsystem_Onl_Calibration` object to `READY_FOR_VALIDATION` and sends back the processed data to the `Validator`, if the status is `RUN_FINISHED`.

If the status of the `Subsystem_Onl_Calibration` object is `READY_FOR_VALIDATION` or `VALIDATION_IN_PROGRESS`, it sends back the processed data to the `Validator`.

If the status of the `Subsystem_Onl_Calibration` object is `VALIDATION_FINISHED`, it sends back a status message with `VALIDATION_FINISHED`.

9. Data transfer for database commit

If the status of the `Subsystem_Onl_Calibration` object is `READY_FOR_COMMIT` or `COMMIT_IN_PROGRESS`, it sends back the processed data to the `Validator`.

If the status of the `Subsystem_Onl_Calibration` object is `COMMIT_FINISHED`, it sends back a status message with `COMMIT_FINISHED` and marks the object for deletion.

3 Use Cases

This section describes the use cases for the subprocesses that participate in the online calibration.

3.1 Taker

- Select calibration configuration

A DAQ shifter selects a sub-detector, a configuration of readout crates, and a mode of calibration and requests downloads to COMICS through `coor`. In this process, calibration data processors on Level 3 nodes or on the front-end processors should also be started.

- Start calibration run

When a calibration run request is processed without problems, the DAQ shifter starts the calibration run by requesting `coor` to start a run.

3.2 `coor`

- Connect to Calibration Manager

`coor` connects to `Calib_Manager`.

- Send the run configuration to Calibration Manager

`coor` sends a run configuration selected by the shifter to `Calib_Manager`, e.g., a sub-detector, a list of crates, and a calibration type, etc.

- Send the start run message to Calibration Manager

`coor` sends a start of run message to `Calib_Manager`.

- Stop calibration run

A DAQ shifter can stop (abort) a calibration run by sending a `force_stop` message to `coor` from `Calib_Manager`.

3.3 Calibration Manager

- Start Calibration Manager

`Calib_Manager` is started when the DAQ system is started. This is a server process and waits for connection requests from sub-processes.

- Accept the connection request from Calibration Manager GUI

`Calib_Manager` accepts connection requests from `Calib_Manager_GUI`.

- Accept the connection request from `coor`

`Calib_Manager` accepts a connection request from `coor`.

- Connect to `coor`

`Calib_Manager` connects to `coor` to be able to send “`force_stop`” run message.

- Connect to Collector/Router
Calib_Manager connects to Collector/Router, from which the results of Calibration are received in the form of an event message.
- Configure a Calibration process
Calib_Manager creates a **Subsystem_Onl_Calibration** object if it does not already exist or re-initialize the **Subsystem_Onl_Calibration** object if it does.
- Abort a Calibration process
Delete the **Validator** process and re-initialize **Subsystem_Onl_Calibration** object.
- Start Validation process
Calib_Manager starts a Validation process when a **start_run** command is received from **coord**. The Validation process in turn connects to **Calib_Manager**.
- Receive data from Calibration Data Processor
Calib_Manager receives data from Calibration Data Processors via Collector/Router in the form of the Event message.
- Send data to Validation process
Calib_Manager sends data to **Validator**, if it is ready to accept data.
- Receive the status report from Validation process
Calib_Manager receives status reports from **Validator**, e.g., whether it has read a reference set from the database and is ready to accept data, etc.
- Receive the request for the status from Calibration Manager GUI
Calib_Manager receives requests for the status report from **Calib_Manager_GUI**.
- Send the status report to Calibration Manager GUI
Calib_Manager sends status reports to **Calib_Manager_GUI**.
- Receive action commands for database transaction from Calibration Manager GUI
Calib_Manager receives action commands for database transaction from **Calib_Manager_GUI**.
- Forwards action commands for database transaction to Validation process
Calib_Manager forwards action commands for database transaction to Validation process.
- Receive summary reports from Validation process
Calib_Manager receives summary reports from Validation process.

- Save summary of calibration task
`Calib_Manager` saves the summary of the calibration task to a file (or sends it to a logger).

3.4 Calibration Manager GUI

- Connect to Calibration Manager
`Calib_Manager_GUI` connects to `Calib_Manager`.
- Request the status report to Calibration Manager
`Calib_Manager_GUI` requests the status report to `Calib_Manager`.
- Receive the status report from Calibration Manager
`Calib_Manager` sends the status report to `Calib_Manager_GUI` upon request.
- Send action command to Calibration Manager
A DAQ shifter makes a decision on whether to commit the new results to the database or not and sends a command to `Validator` thorough `Calib_Manager`.
- Stop calibration run
When the `Calib_Manager` receives the data from all of the crates, the `Calib_Manager` sends a request to stop the run to `coor`. Also the DAQ shifter can stop (abort) a calibration run by sending a message to `coor` via `Calib_Manager`.
- Abort calibration process
Send an Abort command to `Calib_Manager`.

3.5 Calibration Data Processor

- Start Calibration Data Processor
Calibration Data Processors are started by a `coor` command or a `COMICS` download to the front-end processors.
- Send data to Collector/Router
A Calibration Data Processor sends data that contains the results of the calibration data processing, e.g., pedestals and gains, to Collector/Router.

3.6 Validation Process

- Connects to Calibration Manager
`Validator` connects to `Calib_Manager`.

- Send status report to Calibration Manager
`Validator` sends status reports to `Calib_Manager`, e.g., reading the reference data set, not ready for data, ready for data, validation in progress, etc.
- Receive data from Calibration Manager
`Validator` receives data from `Calib_Manager`.
- Retrieve the reference set of the calibration constants from the database
`Validator` retrieves the reference set of the calibration constants from the database.
- Perform comparison between the measured and the reference calibration constants
`Validator` performs comparison between the measured and the reference calibration constants.
- Send summary report to Calibration Manager
`Validator` sends a summary report of the validation to `Calib_Manager`.
- Receive action command from Calibration Manager
`Validator` receives action commands for database transaction from `Calib_Manager`.
- Insert the new calibration results to database
`Validator` inserts the new calibration results to the database.

4 States of objects

This section describes possible states in which the objects used in the calibration process are in.

4.1 `Subsystem_Onl_Calibration` object

The `Subsystem_Onl_Calibration` object can be in the following states:

- `UNKOWN`
Not used.
- `READY_FOR_RUN`
Set in the constructor.
- `RUN_IN_PROGRESS`
Set in `COOR_Message_Handler` upon receiving a `coor` command “`start_run`”, or set in `add_subsystem` method when it is called by a `Calib_Manager_GUI` message.

- **RUN_FINISHED**
Set in `set_proc_data` method upon receiving a data (an event) message that completes the data set requested.
- **RUN_ABORTED**
Not used.
- **READY_FOR_VALIDATION**
Set in `DB_Interface_Message_Handler`, when a `DATA_REQUEST` command is received and the status is `RUN_FINISHED`, or set in `set_validation_status` method, when a `VALIDATION_STATUS` message is received, or set in `send_vldt_crates` method, when a `VALIDATION_INQUIRY` message is received and the status is `RUN_FINISHED`.
- **VALIDATION_IN_PROGRESS**
Set in `set_validation_status` method, when a `VALIDATION_STATUS` message is received.
- **VALIDATION_FINISHED**
Set in `set_validation_status` method, when a `VALIDATION_STATUS` message is received and all of the crates are validated.
- **VALIDATION_ABORTED**
Not used.
- **READY_FOR_COMMIT**
Set in `start_db_commit` method, when a `START_COMMIT` message is received, or set in `set_commit_status` method, when a `COMMIT_STATUS` message is received and all of the crates are in `C_READY_FOR_COMMIT`.
- **COMMIT_IN_PROGRESS**
Set in `set_commit_status` method, when a `COMMIT_STATUS` message is received and at least one of the crates is in `COMMIT_IN_PROGRESS`.
- **COMMIT_FINISHED**
Set in `set_commit_status` method, when a `COMMIT_STATUS` message is received and all of the crates are in `C_COMMIT_FINISHED`.
- **COMMIT_ABORTED**
Not used.

4.2 Subsystem_Proc_Status object

The Subsystem_Proc_Status object can be in the following states:

- C_UNKOWN_PROC_STATE
Not used.
- C_READY_FOR_RUN
Set to this, when a Subsystem_Proc_Status object is created.
- C_RUN_IN_PROGRESS
Currently not set. This should be set, when coor sends out a “start_run”.
- C_RUN_FINISHED
Set in set_proc_data.
- C_RUN_ABORTED
Not used.

4.3 Subsystem_Validation_Status object

The Subsystem_Validation_Status object can be in the following states:

- C_UNKNOWN_VLDT_STATE
Set to this, when a Subsystem_Validation_Status object is created with the validation option off.
- C_READY_FOR_VALIDATION
Set to this, when a Subsystem_Validation_Status object is created with the validation option on.
- C_VALIDATION_IN_PROGRESS
Set in send_processed_data.
- C_VALIDATION_FINISHED
Set in set_validation_status, upon receiving a Validation_Status message from the Validator.
- C_ABORTED_VALIDATION
Not used.

4.4 Subsystem_Commit_Status object

The Subsystem_Commit_Status object can be in the following states:

- C_UNKNOWN_COMMIT_STATE
Not used.
- C_READY_FOR_COMMIT
Set in `start_db_commit`.
- C_COMMIT_IN_PROGRESS
Set in `send_commit_data`.
- C_COMMIT_FINISHED
Set in `set_commit_status`, upon receiving a `Commit_Status` message from the Validator.
- C_ABORTED_COMMIT
Not used.

4.5 Subsystem_Validation object in Subsystem_Validator

The Subsystem_Validation object can be in the following states:

- UNKNOWN
- READY_FOR_VALIDATION
Set in `read_opaque_message` method, when a `START_VALIDATION` message is received.
- VALIDATION_IN_PROGRESS
- VALIDATION_FINISHED
- VALIDATION_ABORTED
- READY_FOR_COMMIT
Set in `read_opaque_message` method, when a `START_COMMIT` message is received.
- COMMIT_IN_PROGRESS
- COMMIT_FINISHED
- COMMIT_ABORTED

5 Messages and Message Handlers

This section describes messages that are exchanged between subprocesses and `Calib_Manager`.

5.1 String Messages

String messages are sent by `coor`, `Calib_Manager_GUI` and `Validator` to `Calib_Manager`.

5.1.1 `coor` Messages

- `init`
Do nothing at the moment
- `coor_addr`
`_coor_port` and `_coor_host` are set in `Online_Calibration` object.
- `set_client`
`add_subsystem(subsys_id, coor_cmd)` is called. `coor_cmd` is used to see if `set` or `clear` is required when `configure` is issued later.
- `calib_type`
set the calibration type.
- `crates`
set the list of crates.
- `stream`
set the calibration `stream id`.
- `configure`
configures `Subsystem_Onl_Calibration` object.
- `runinfo`
set `run_number`.
- `start_run`
set status to `RUN_IN_PROGRESS`.
- `pause_run`
- `resume_run`
- `stop_run`
- `clear_client`

5.1.2 Calib_Manager_GUI Messages

- STATUS_REQUEST
subsystem_status_msg method is called.

5.1.3 Validator Messages

- VALIDATION_INQUIRY
send back the list of crates requested for validation.
- DATA_REQUEST
if the status of this subsystem is READY_FOR_VALIDATION or VALIDATION_IN_PROGRESS, send back the subsystem calibration data to the validation process. if the validation for all of the crates are completed, send a VALIDATION_STATUS message with status=VALIDATION_FINISHED.

5.2 Opaque Messages

Opaque messages are used to convey mainly numerical values that are cumbersome to code/encode in the `String` message. The format of the message is the following:

1. Message identifier
An integer word is used to identify the message type.
2. Subsystem identifier
An integer word is used to identify the sub-detector system.
3. Number of integer words
An integer word to describe the number of integer values that are sent by this message.
4. Array of integer words
An array of integer numbers.
5. Number of float words
An integer word to describe the number of floating point values that are sent by this message.
6. Array of float words
An array of floating point numbers.

There are 10 message types described below.

- **START_CALIB**
Calib_Manager receives it from GUI.
- **STATUS_MSG**
Calib_Manager sends it to GUI, upon receiving a request.
- **PROC_STATUS**
Obsolete.
- **PROCESSED_DATA**
Calib_Manager sends it to the Validator.
- **START_VALIDATION**
Calib_Manager sends it to the Validator.
- **VALIDATION_STATUS**
Calib_Manager sends it to the Validator, when data for all of the crates have been sent to the Validator.
- **VALIDATION_SUMMARY**
Validator sends it to the Calib_Manager.
- **START_COMMIT**
GUI sends it to the Calib_Manager.
- **COMMIT_STATUS**
Validator sends it to the Calib_Manager.
- **COMMIT_SUMMARY**
Not used.

5.3 Event Messages

There are two types of **Event** messages for calibration runs. The **Event** message from a SDAQ calibration run is similar to **Opaque** messages except that they contain an event header for routing by the DAQ data mover processes. The format of the data part of the message is the following:

1. Crate identifier

One **Event** message contains data from only one crate. This crate identifier is also used to determine the sub-detector system that is sending this message. The data crate map is described in a note (see <http://niuhep.physics.niu.edu/fortner/d0/algo/unpack/crates.pdf>).

2. Message identifier

An integer word is used to define the calibration type.

3. Number of channels

An integer word is used to describe the number of channels that are calibrated.

4. Calibration record

If the pedestal values are sent by this message, the following four words are repeated: an integer word of channel id, a float word of the pedestal value, a float word of the error of pedestal value, and an integer word of error flag.

If the gain values are sent by this message, the following six words are repeated: an integer word of channel id, a float word of the pedestal value, a float word of the error of pedestal value, a float word of the gain value, a float word of the error of gain value, and an integer word of error flag.

The **Event** message from a Calorimeter calibration run that uses the L3 as the data processor is the standard **Event_Message** with an additional data chunk (**L3DebugChunk**) attached to it. The **L3DebugChunk** contains the results of a calibration run. This chunk can have pedestals from all 12 Calorimeter crates.