The sPHENIX Run Control Program

Martin L. Purschke

March 11, 2023

Contents

1 Introduction

The sPHENIX experiment uses a large number of instances of the RCDAQ data acquisition program to acquire the detector data. Each RCDAQ instance typically reads out a particular section of the detector. For example, there are 24 "sectors" on the TPC pad planes (12 in the north and in the 12 south), so that the entire TPC is read out with 24 RCDAQ instances on as many servers.

In total, the entire experiment will have about 60 RCDAQ instances.

In order to coordinate those RCDAQ instances, a program is needed that controls them all from one place. This program is called *Run Control*, *rc* for short.

Much like the RCDAQ servers themselves, run control consists of a server program that the user, script, and GUIs interact with b way of a client program.

Users familiar with RCDAQ will recognize the similarities – RCDAQ has a server, called *rcdaq_server* that is the one that interact with the hardware, and a client, called *rcdaq_client* (Fig. 1).

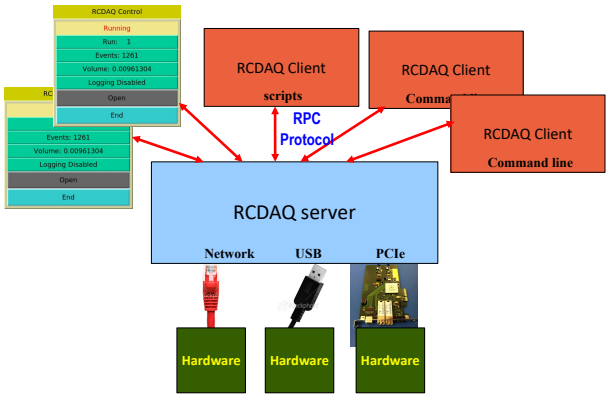

Figure 1: The schematic overview of how the RCDAQ client/server setup works. An arbitrary instance of clients, such as interactive commands, scripts and GUIs, can interact with the RCDAQ system concurrently.

Similarly, run control consists of a server, *rc_server* that is itself controlled by its client, *rc_client*. The rc_server acts like a client towards the RCDAQ instances, and instructs all of them to execute the same function or action.

In this way one can control a large number of instances from one control process.

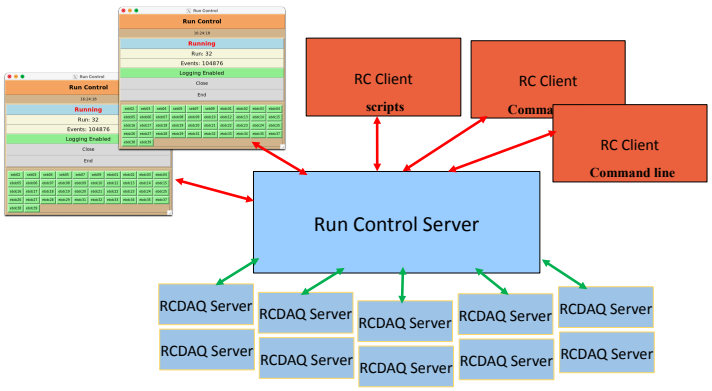

Figure 2: The schematic overview of how the Run Control client/server setup works, which is similar to the once of RCDAQ.

2 How to run Run Control

In order to function, the rc_server process needs to know the identity of the RCDAQ instances it needs to control. It also needs to know the identity of the GTM unit, and whether the GTMs are running in *local* or *global* mode.

The way the server operates on a begin-run action is the following:

- it goes through the list of RCDAQ instances and issues the programmatic equivalent of the *daq_begin -i* command;
- after that, it makes another pass through the RCDAQ instances issuing a *daq_sync*
- after that, all RCDAQ instances are "primed" and wait for triggers to acquire data.
- it issues the (again programmatic equivalent) of the *gtm_startrun* command.

The *daq_begin -i* instructs RCDAQ to return immediately from the call. In this way, all instances receive the instruction to begin data taking, but the issuer (Run Control in this case) does not need to wait for its completion. The subsequent *daq_sync* command will then wait for RCDAQ to complete the earlier command.

The effect is that all instances execute the begin-run sequences in parallel. The first daq_sync instruction will typically wait a bit for the first RCDAQ instance to complete the action, but the next calls will return right away because they have executed the action in parallel.

If we assume that the process takes 10 seconds per instance, and we would execute the command for each of the 60 instances one after the other, the begin-run action would take 600 seconds, or 10 minutes. Conversely, the parallel execution will start taking events in less that 15 seconds.

3 Starting Run Control

The simplest (although not really scalable) way to start run control is to list all RCDAQ hosts on the command line, such as (assuming we run just the sPHENIX Hadronic Calorimeter)

```
rc_server seb16 seb17 &
```

From this point on, all rc_client commands will act on those 2 instances.

One usually creates a script to start the server in a reproducible fashion, redirecting the output to a log file:

```
#! /bin/bash
rc_server seb16 seb17 > $HOME/rc.log 2>&1 &
```

Listing all hosts on the command line can get unwieldy for a larger number of hosts. For that reason, the `-f` option allows one to specify a file that contains the list of the RCDAQ hosts.

So `HCal.list` can contain the (short) list of hosts:

```
seb16
seb17
```

and then we can specify

```
#! /bin/bash
rc_server -f HCal.list  > $HOME/rc.log 2>&1 &
```

With the same effect as the command above.

Multiple `-f` options accumulate, providing a simple way to specify multiple detector system together while preserving the ability to run them individually during commissioning and debugging:

```
#! /bin/bash
rc_server -f HCal.list -f EmCal.list -f INTT.list -f MVTX.list -f TPC.list > $HOME/rc.log 2>&1 &
```

4 GTM Interaction

As shown, the above commands implicitly interact with our GTM unit in *global* mode, meaning that all enabled GTMs start together, enabling data taking from the same beam crossings among the detectors.

If one wishes to run just one detector and in local mode (e.g. the EmCal that is connected to GTM 5), one would issue

```
#! /bin/bash
rc_server -n 5  -f EmCal.list  > $HOME/rc.log 2>&1 &
```

This has the effect of starting all 8 EmCal instances first and then starting GTM 5.

The -n can take multiple GTM numbers, such as `-n 5,10,13` to interact with GTMs 5 (Emcal), 10 (MBD), and 13 (LL1).

The actual GTM unit is typically consistently identified by the (always centrally defined) env. variable `$GTMHOST`:

```
$ echo $GTMHOST
gtm.sphenix.bnl.gov
```

However, one can specify (and so override) the definition with the `-g` option:

```
#! /bin/bash
rc_server -n 5 -g gtm1.sphenix.bnl.gov -f EmCal.list  > $HOME/rc.log 2>&1 &
```

Finally, there are situation where one wants to run without any GTM interaction. This is accomplished with the `-x` option, which instructs the server not to interact with the GTM at all.

```
#! /bin/bash
rc_server -x -f EmCal.list  > $HOME/rc.log 2>&1 &
```

Here is the list of all supported options:

```
  $ rc_server -h

 rc_server <options> daq_host1 daq_host2 ...
     example: rc_server daq00 daq01
   options:
      -h            help this help text
      -v            verbose, multiple options accumulate
      -x            run without GL1/GTM support
      -g hostname   GL1/GTM host
      -n i,j...     vGTM numbers if in local mode
      -f filename   read hostlist from file
```

5 RC Commands

The rc commands are a subset of the well-know RCDAQ commands. Generally speaking, the commands replace "daq" with "rc", so that the single-instance "daq_begin" becomes "rc_begin" for Run Control. Over time we will add more commands as needed. Here is the current list:

```
$ rc_client
   help                              show this help text
   rc_status [-s] [-l]               display status [short] [long]
   rc_open                           enable logging
   rc_hostlist                       show the active hostlist
```

```
rc_begin [run-number]              start taking data for run-number, or auto-increment
rc_end                             end the run
rc_close                           disable logging
rc_setrunnumberfile file           define a file to maintain the current run number

rc_set_runtype type                activate a predefined run type
rc_get_runtype [-l]                list the active runtype (if any)
rc_set_maxevents nevt              set automatic end at so many events
rc_shutdown                        terminate the rcdaq backend
```

Each of the above commands is the exact equivalent of the corresponding rcdaq_client commands.

6 The Run Control GUI

The Run Control GUI is a graphical interface to the rc_server. It provides a simple and intuitive interface for the shift crew to run the DAQ.

Execute

```
RunControlGUI.py &
```

Figure 3: A screenshot of the RunControl GUI in action.

As with the GUIs of RCDAQ, this one is *stateless*, that is, multiple instances of the GUI can exist, and all show the same information.

7 Restrictions and Enhancements

There are currently some restrictions that we are working on. Planned enhancements include:

- a more comprehensive set of commands to bring more (albeit not all) RCDAQ commands to RC
- it is currently not possible to adjust the participating servers dynamically. In order to remove a server, one needs to shut down and the restart the server without the removed server name. We will implement a more dynamic RCDAQ instances management.
- like the server itself, the RunControl GUI has no ability to dynamically adjust the button matrix at the bottom (see Fig. 3 when a new instance of the rc_server restarts with a different list. We will make this dynamic as much as possible.