DAQ User Guide

for the Muon g−2 Experiment

R. Chislett,, University College London
S. Ganguly, University of Illinois
W. Gohn, University of Kentucky
T. Gorringe, University of Kentucky
F. Hahn, University of Kentucky
T. Stuttard, University College London

Contents

***

Introduction

***

The data acquisition system (DAQ) for the Muon g-2 experiment is based on the MIDAS software package. The DAQ is designed to process data from Calorimeters, Trackers, Quads, and Fiber Harps, which has been digitized in a μTCA crate. The Calorimeter, Quad, and Fiber Harp signals are processed by ”Riders”, a custom 800 MSPS 12 bit waveform digitizer that has been developed at Cornell University. The Tracker data comes from an FC7, which is another type of advanced mezzanine card that resides in a μTCA crate.

The DAQ software is modular, and has several parts working together. MIDAS provides the mserver, mhttpd, and mlogger programs that run and organize communication between the frontends, hosts a web interface, and writes the data to disk. An event builder assembles data fragments from multiple buffers, which have been produced by the frontends.

The 24 calorimeters, fiber harps, and quads are read by a frontend named CaloReadoutAMC13. This frontend included code written using CUDA to process the data in the systems GPU, which in our case are Nvidia Tesla K40Cs. The frontend writes data in raw, histogrammed, and T and Q method formats.

A frontend called MasterGM2 uses RPC calls to send begin and end of run signals to the other frontends, as well as triggers to any synchronous frontends.

Additional frontends read data from the Tracker detectors.

The ROME software is used to create online displays of data in real time.

Installation

***

2.1 Installation

2.1.1 Prerequisites

The DAQ for muon $g - 2$ is based on MIDAS, so it is necessary to first install MIDAS, as well as other prerequisites, before installing the DAQ software. We will also need to install ROOT, which is a prerequisite for ROME, our data quality monitoring software.

First, you will want to use yum to install some of the basic prerequisite packages. Two packages that are required by our DAQ software can be installed by:

```
yum install mysql-devel readline-devel zlib-devel
```

Next, install all the prerequisites for ROOT:

```
yum install libX11-devel libXpm-devel libXft-devel libXext-devel
```

, and you might as well install the optional ROOT packages as well:

```
yum install gcc-gfortran openssl-devel pcre-devel \
        mesa-libGL-devel glew-devel ftgl-devel mysql-devel \
        fftw-devel cfitsio-devel graphviz-devel \
        avahi-compat-libdns_sd-devel libldap-dev python-devel \
        libxml2-devel gsl-static
```

Now, you can install ROOT using your favorite method. I prefer to use git, so execute

```
git clone http://root.cern.ch/git/root.git
cd root
git tag -l
git checkout -b v5-34-08 v5-34-08
./configure
make
```

If you are using the calorimeter readout code that processes data on the GPU, you must also install CUDA. To install CUDA, follow the instructions at https://developer.nvidia.com/cuda-downloads.

Now, you are ready to install MIDAS and ROME.

2.1.2 MIDAS Installation

To install MIDAS, first obtain our gm2midas git repository from Redmine.

```
git clone ssh://p-gm2midas@cdcvs.fnal.gov/cvs/projects/gm2midas
```

You will need the environment variable MIDASSYS set to the give the path to your midas installation. For example, if your username is daq and you checked out gm2midas in your home directory, you would

```
export MIDASSYS=/home/daq/gm2midas/midas
```

Now you go to the directory where your MIDASSYS has been defined, and install midas using

```
cd $MIDASSYS
gmake
sudo gmake install
```

The last command, ”gmake install”, is optional and will copy the MIDAS installation to your system (not good on a shared computer, for instance).

2.1.3 ROME Installation

Since you have already checked out gm2midas, you already have ROME, and it just needs to be compiled (if you did not already checkout gm2midas, refer back to the previous section for instructions on how to do so). You need to set the ROMESYS environment variable to the subdirectory gm2midas/rome, navigate there, and type ”make”.

```
export ROMESYS=/home/daq/gm2daq/rome
cd $ROMESYS
make
```

2.1.4 Checking out gm2daq

Now, to obtain our DAQ code, checkout our git repository

```
git clone ssh://p-gm2daq@cdcvs.fnal.gov/cvs/projects/gm2daq
```

If you are a DAQ user, you should be on the master branch, and you only need to compile. If you are a DAQ developer, or you would like to test out the newest ”experimental” version of the DAQ softwre, you should checkout the develop branch.

```
cd gm2daq
git checkout develop
```

We are currently using Makefiles for compilation. To compile any portion of the DAQ software, navigate to the desired subdirectory and type ”make”. The directories requiring compilation for normal *g*-2 running with AMC13 readout are

```
gm2daq/eventbuilderNew/
gm2daq/amc13/amc13StandaloneMAN_2014-05-12/
gm2daq/frontends/CaloReadoutAMC13/
gm2daq/frontends/MasterGM2/
```

If you also need to simulate data (i.e. you do not have an AMC13 as an input source), also compile

```
gm2daq/frontends/AMC13Simulator/
```

2.1.5 Environment Setup

Our environment is most easily setup using a script called setup.sh that is located in the main gm2daq repository. It is recommended to source that script from your own .bash_profile file. You may need to edit the values of some of the necessary environment variables if you are running on your own machine. The variables that this script sets up are:

- GM2DAQ_DIR: The location of your gm2daq repository.
- ROOTSYS: The location of your root installation.
- MIDASSYS: The location of your MIDAS installation.
- ROMESYS: The location of your ROME installation.
- CACTUS_ROOT: Where your IPBUS is installed (usually /opt/cactus).
- AMC13_STANDALONE_ROOT: The location of your AMC13 software.
- CUDASYS: Where your CUDA is installed (usually /usr/local/cuda).
- PATH: Adds the bin directories corresponding to the variables defined above to your PATH.
- LD_LIBRARY_PATH: Add the lib directories corresponding to the variables defined above to your LD_LIBRARY_PATH.

Running the DAQ

***

Before starting MIDAS, you must first define an experiment using the exptab file. By default, MIDAS will look for this file at /etc/exptab, but it is possible to put it somewhere else and tell MIDAS how to find it with an environment variable. You can define up to ten experiments in the exptab, with the format ”NAME directory user”, where NAME is the name of the experiment, the directory is where MIDAS will write the files necessary to define the experiment, and user is the username of the user who can run this experiment. As an example, an exptab file to define three experiments for users bernie, hillary, and donald , would look like.

```
EXP1 /home/bernie/exp bernie
EXP2 /home/hillary/exp hillary
EXP3 /home/donald/exp donald
```

It is then necessary that each user creates the directory ”exp” in their home directory.

To fully run our DAQ software, you need to run three MIDAS programs, and at least two of the programs we compiled in the previous section.

The first program to start is the MIDAS mserver. This is the main server that communicates between MIDAS programs running on different machines. If you are the only user, you can run the mserver under your own username, but we have learned that it works best if run under root, in particular when you have multiple users sharing a machine. One instance of mserver will handle all running experiments. The standard usage is

```
mserver -D
```

Next you have to start the MIDAS webserver, mhttpd, which allows the user to control an experiment via a MIDAS web interface. The standard usage is

```
mhttpd -e GM2 --mg 8080 -D
```

where the experiment name here is GM2 and you are using port 8080. You must run one instance of mhttpd for each experiment running on a system, and each must use a unique port. Since we typically run behind a firewall, we do not use all of the mhttpd securiy features, but it is possible to replace the mg flag with ”https 8444”, which will allow you to set a password for your webserver if desired.

The MIDAS program mlogger writes the data to disk. It requires only that you specify the experiment name, and the rest of it's settings are controlled via the ODB.

```
mlogger -e GM2
```

It is also necessary to run the event builder, which assembles data fragments from all specified frontends and combines them into a single event. This program is located in gm2daq/eventbuilderNew. After compiling, start the event builder using

```
./mevb -e GM2 -b BUF
```

where again GM2 is the experiment name and BUF is the prefix used to identify each unique event buffer. The event buffers must be defined for each frontend in it's ODB Common block, and they must each start with BUF. For instance, we generally define the buffers as BUF01, BUF02, BUF03, etc.

Finally, start the frontend code for each frontend you are running. As a minimum, you should start MasterGM2 and one other fronend. The usage for these will be discussed in subsequent sections.

Event Builder

***

4.1 /gm2daq/eventBuilderNew/

4.1.1 Running Event Builder

It is also useful to run the event builder, which assembles data fragments from all specified frontends and combines them into a single MIDAS event. This program is located in gm2daq/eventbuilderNew. After compiling, start the event builder using

```
./mevb -e GM2 -b BUF
```

where again GM2 is the experiment name and BUF is the prefix used to identify each unique event buffer. The event buffers must be defined for each frontend in it's ODB Common block, and they must each start with BUF. For instance, we generally define the buffers as BUF01, BUF02, BUF03, etc.

For each enabled frontend, the user may specify whether or not to send it's data to the event builder by setting an ODB flag under the Global settings for that frontend called ”Send to Event Builder”, i.e.

```
"/Equipment/AMC1301/Settings/Globals/Send to Event Builder" y
```

4.1.2 ODB Options

For normal running, you should not need to change the ODB options. There are settings that can be used to enable or disable the building of data from individual frontends, but they have been made redundant by the 'send to event builder' flag in the globals settings for each frontend.

MasterGM2 Frontend

***

5.1 /gm2daq/frontends/MasterGM2/

5.1.1 Functionality

The MasterGM2 frontend controls other frontends in the experiment via RPC calls, sending begin and end of run signals, as well as triggers to synchronous frontends. The triggering mechanism has multiple options, but the typical running mode will use a trigger input to a Meinberg GPS device that will time stamp each fill, and the MasterGM2 frontend will write these timestamps into the TRIG bank.

If the appropriate option is selected in the ODB, this frontend also sends begin of run and end of run commands via IPBus to the FC7 in the CCC μTCA crate.

5.1.2 ODB Options

The MasterGM2 ODB options are under /Equipment/MasterGM2/Settings/Globals/. The configurable parameters are:

- **Trigger source** : The type of trigger used, taken as a string. Possible options are:
  - GPS: Takes triggers via an input on a connected Meinberg GPS unit.
  - PP: Takes triggers via a parallel port input.
  - Socket: Takes triggers via a tcp socket, configurable via the ODB.
  - Fake: Generates triggers at a given rate.
- **Socket trigger IP addr** : IP address if using socket triggers.
- **Socket trigger port** : Port number if using socket trigger.
- **Rate** : Data rate if using Fake trigger.

- **Readout name** : Base name of the readout frontend. For normal operation, use "AMC13". Do not include index.
- **Simulator name**: Base name of simulator (if using). For normal simulator operation, use "AMC13Simulator". Do not include index.
- **Send to event builder** : Boolean to choose whether to include MasterGM2 (TRIG) data in built events.
- **Verbose** : Boolean to turn on extra print messages.
- **CCC** : Boolean to turn on IPBus BOR/EOR signals to the CCC system.

5.1.3 Bank Output

If data is written from the MasterGM2 frontend, it is written in the TRIG bank as DWORDs.

Calorimeter Frontends

***

6.1 /gm2daq/frontends/CaloReadoutAMC13

6.1.1 Functionality

The calorimeter readout frontend is located in the directory gm2daqfrontendsCaloReadoutAMC13, and will be displayed with the frontend name "AMC13XX", where XX is an integer corresponding to the index of the μTCA crate, 01-24 for the calorimeters.

Usage is:

```
./frontend -e GM2 -h <backend_name> -i <i>
```

Where the backend_name is the hostname of the computer where your MIDAS mserver is running, usually g2be or g2be1, and "i" is a frontend index, 1-24 for the calorimeter systems.

6.1.2 ODB Options

The ODB settings for the CaloReadoutAMC13 frontend are subdivided into Global setting, which control the basic frontend bahaviour, AMC13 settings, used to configure the AMC13 itself, Link01 settings, used to configure the DAQ link between the readout and source, Calorimeter settings for slow control parameters, and settings for each of the 12 Rider slots in the crate.

The global settings are used to set up the frontend for data taking, allowing the user to choose the type of data that will be saved, and how it will be processed. The available options are:

Globals

- **sync** : Boolean to define if the frontend run synchronously or asynchronously with the trigger.

- **TQ methods** : Boolean to turn on T and Q methods in the GPU processing.
- **GPU waveform length** : Length of digitized waveform in adc samples.
- **GPU island presamples** : Number of adc samples to prepend to each island.
- **GPU island postsamples** : Number of adc samples to add to end of each island.
- **Calo sum decimation factor** : Factor by which to decimate the Q-method sum.
- **raw data store** : Boolean to store raw data.
- **raw data prescale** : Prescale for raw data (data will be saved for every Nth event)
- **raw data prescale offset** : Number of events to wait before writing raw event..
- **histogram data store** : Boolean to turn on histogramming Q-method.
- **histogram data flush** : Number of events taken to build histogram before writing to data file.
- **histogram data flush offset** : Number of events to wait before writing histogrammed data.
- **use AMC13 simulator** : Boolean to turn on simulated data mode. Default is 'n' for running with Rider readout.
- **GPU Device ID** : Specifies which internal GPU to access for data processing.
- **island_option** : Option for T method threshold cut to identify islands. Options are: 0. Periodic island condition, 1. Sum of waveforms for calorimeter, 2. Individual crystal leading edge trigger, 3. Pulse shape weighted cut.
- **T method threshold value** : Threshold for T method. It should be negative for negative pulses.
- $+ve(-ve)$ **crossing-thres Y(N)** : Polarity of pulse. Y is positive and N is negative.
- **pedestal_option** : 1 computes pedestals in the GPU, and 0 uses a global pedestal.
- **T method global pedestal** : Value for pedestal if pedestal_option is 0.
- **Send to Event Builder** : Boolean to send data to event builder.
- **Shelf configuration** : Set to rider or ccc.

Link01

- **enabled** : Boolean to enable the DAQ link.
- **port nr** : Port number on the readout side.
- **readout ip** : IP address of the 10 Gb link on the readout computer.

- **source port** : Port number on the client side. For the AMC13 the port must be 0x1234 or 4660.
- **source ip** : IP address of your client, *i.e.* the DAQ link on the AMC13.

AMC13

- **header size (bytes)** : Size of AMC13 header; default is 0x1000.
- **amc_block_size** : Size of payload block; default is 0x8000.
- **tail_size** : Size of tail data; default is 0x8.
- **serialNo** : Serial number of AMC13.
- **slot** : Slot in μTCA crate where AMC13 is located (should be 13).
- **usingControlHub** : Boolean to define whether or not Control Hub is used for the AMC13.
- **T2ip** : IP address of the T2 FPGA.
- **T1ip** : IP address of the T1 FPGA.
- **addrTab1** : Path to address table for the Spartan FPGA.
- **addrTab2** : Path to address table for the Kintex FPGA.
- **enableSoftwareTrigger** : Boolean to enable software trigger. This should only be set if the AMC13 is generating internal triggers.

Rider

There are 12 Rider subdirectories for each of the 12 AMC slots in the μTCA crate. Each subdirectory has a Board subdirectory and five Channel subdirectories.

First, under board are the options:

- **rider_enabled** : Boolean to enable the AMC.
- **burst_count** : Sets the burst count on the board. The waveform length will be 8 times this burst count. The default burst count is 70000 to give a 700 μs waveform.
- **ip_addr**: Currently not used.

Second, under each of the five channel directories:

- **enabled** : enable the channel.
- **Gain** : Gain for each SiPM.
- **Segment index** : unique index for each Rider.

Bank Name	Function
CBxx	Header information from the AMC13 and Riders for each fill.
CZxx	Trailer information from the AMC13 and Riders for each fill.
CRxx	Raw data, which is often prescaled, so not written for every event.
CTxx	T-method processed islands.
CQxx	Q-method decimated sums.
CHxx	Q-method event-by event summed histograms.
CLxx	Information on timing of the data processing.

Table 6.1: The list of banks that can be printed by the CaloReadoutAMC13 frontend. The xx denotes the frontend index of 01-24, which is written into the bank name.

– **EEprom** : EEprom for each channel

– **Temperature** : SiPM Temperature

6.1.3 CalorimeterSettings

This directory contains slow control settings for the calorimeter. Currently the settings are only stored here, and must be manually modified and passed to the device.

– Filter wheel

– Bias Voltage1

– Bias Voltage2

– Bias Voltage3

– Bias Voltage4

CaloMap

The CaloMap directory contains 54 subdirectories, one for each segment. Under each subdirectory there are two entries, which map the segment to a particular rider and channel. The *Rider module* entry should be a number 1-12, and the *Rider channel* entry should be from 1-5.

6.1.4 Bank Output

The CaloReadoutAMC13 frontend outputs MIDAS banks with a variety of information, including Raw waveforms, T and Q method derived datasets, and header and trailer information. All of the included banks are summarized in Table. 6.1.

The data will come in three fill types, which the CaloReadoutAMC13 will determine from the AMC13 headers. The fill types are muon, laser, or pedestal. For muon fills, the full

Fill Type	Characteristics	CB,CZ	CT, CQ, CH	CR	CL
muon	Single, continuous waveform.	yes	yes	pre-scaled	yes
laser	One or more WFD-chopped waveforms.	yes	no	yes	yes
pedestal	Raw pedestals	yes	no	yes	yes

Table 6.2: The data saved for each fill type by the CaloReadoutAMC13 frontend.

processing in the GPU is turned on for each event (if the appropriate ODB flag is checked), but for the laser and pedestal types, only the raw data is written. Details of which banks are saved for each fill are given in Table 6.2.

ROME

***

How to use ROME.

Troubleshooting Tips

***

8.1 Troubleshooting

This section will provide some general tips for troubleshooting the DAQ. We will make an effor to collect some common failure modes here, and provide solutions.

8.1.1 Run does not stop

In the case when the user has clicked the stop button, and the experiment hangs for several minutes, the run transition can be expediated by using the ”stop now” command in odbedit. To execute, run

```
odbedit -e <expname> -c "stop now"
```

One should be judicial in using this solution. Sometimes a run stop can be delayed while the event builder and data logger are processing the events remaining in the data buffers, and a ”stop now” will throw away that data.

8.1.2 Cannot remove frontend

Sometimes MIDAS will give an error that a particular frontend cannot be removed. This will often happen for instance if you try to stop a frontend that is actively working, or something else causes it to lose it’s connection to the mserver. In this case, manually stop the frontend if it is running (go to the terminal or screen and type Ctrl+C). Then do a clean command in odbedit.

```
odbedit -e <expname> -c clean
```

8.1.3 ODB recovery

If your ODB becomes corrupt, you can save it by loading the last good ODB, which is saved in the last.xml file located in your data directory. To recover the ODB, use odbedit with the command

```
odbedit -e <expname> -c "load last.xml"
```

You may also find that you want to load the ODB settings from a previous run. You can do this in a similar way, but instead of the last.xml file you should use the .odb file created for the run from which you want to load the settings.

```
odbedit -e <expname> -c "load runXXXX.odb"
```

In this case, the old run number will also be loaded into your odb, so you must be careful to manually change the run number when starting your next run. You can change it to any number that has not already been used. If you try running with a previously used run number, that can create problems.

8.1.4 Restarting midas

There is a script under gm2daq/scripts called midas-restart, that should be in your path. To run it, first insure that the script has the correct experiment name defined, and then just type ”midas-restart” in any terminal. It will kill all midas processes, remove semaphores, and then restart mserver and mhttpd. You might want to run it twice to be safe.

8.1.5 Permission problems

Sometimes some of the shm files will be owned by root, and you may need to change the ownership to the user who is running the experiment. This is done using the standard chown command, but must be done by a user with root access. The typical usage is

```
chown user.gm2 .FILE.SHM
```

8.1.6 EXPERTS ONLY: Removing SHM files

If your midas-restart does not solve your problem, it may be necessary ro remove your SHM files. To do so, first go to your experiment directory. There you can view the SHM files using ”ls -a”. Depending on the problem, it may be fixed by removing just the .BUFXX.SHM files, and then doing a midas-restart.

If that does not fix the problem, you may need to also remove the .ODB.SHM file, but that also will clear out your odb. If you need to do this, follow the following steps.

1. rm .ODB.SHM.

2. Use odbedit to load your last.xml file into the odb.
3. Do a midas-restart

If these steps still do not solve the problem, it may be necessary to remove the files in /dev/shm/ that correspond to your experiment. This will require root access to the computer you are using. If you are going to do this, make sure you know what you are doing!

AMC13 Setup

***

To set up the AMC13, it must be installed in the upper slot 13 of the μTCA crate, with the MCH in the lower slot, as shown in Fig. A. All communication with the AMC13 is performed via the network connection on the MCH. Data is read out of the AMC13 via a 10 Gb fiber that should be connected to the top (DAQ0) SFP+ port on the AMC13.

For general use, all configuration of the AMC13 should be handled automatically by the DAQ code. For expert users, it may be necessary to configure and run the AMC13 manually using the program amc13tool2.exe, which is provided in the gm2daq repository.

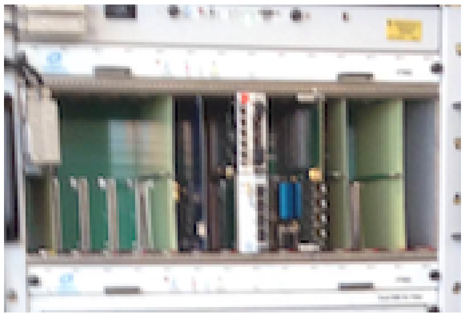

Figure A.1: An AMC13 in a μTCA crate.

TCPIP Configuration

***

B.1 TCPIP Optimization

We have found the following settings to be useful for 10 Gbps fiber readout.

Turn off 'irqbalance'.

```
root# service irqbalance stop
root# chkconfig irqbalance off
```

Setup jumbo frames

```
root# ifconfig p6p1 mtu 9000 txqueuelen 1000 up
```

Increase the TCP max buffer size settable using setsockopt() to 256 Mbytes

```
root# sysctl net.core.rmem_max
net.core.rmem_max = 124928
root# sysctl net.core.rmem_max=268435456
net.core.rmem_max = 268435456

root# sysctl net.core.wmem_max
net.core.wmem_max = 124928
root# sysctl net.core.wmem_max=268435456
net.core.wmem_max = 268435456
```

Increase 'netdev max backlog' to 250000. This is the maximum size of the receive queue.

```
root# sysctl net.core.netdev_max_backlog
net.core.netdev_max_backlog = 1000
root# sysctl net.core.netdev_max_backlog=250000
net.core.netdev_max_backlog = 250000
```

Enable 'tcp mtu probing'

```
root# sysctl net.ipv4.tcp_mtu_probing
net.ipv4.tcp_mtu_probing = 0
```

```
root# sysctl net.ipv4.tcp_mtu_probing=1
net.ipv4.tcp_mtu_probing = 1
```