MIDAS INTRODUCTION

Gennaro Tortone - INFN Napoli

Introduzione alle Tecniche di Trigger e Data Acquisition in Esperimenti di Fisica

Napoli 9-12 ottobre 2023

TABLE OF CONTENTS

INTRODUCTION

WHAT IS MIDAS

MIDAS is an acronym for Maximum Integrated Data Acquisition System.

MIDAS is a general-purpose system for event-based data acquisition in small and medium scale Physics experiments. It is an on-going development at the Paul Scherrer Institute (Switzerland) and at TRIUMF (Canada), since 1993.

MIDAS is based on a modular networking capability and a central database system.

MIDAS consists of a C/C++ library and several applications, which can run on many different operating systems (Linux, Windows, MAC OS)

WHAT MIDAS CAN DO FOR YOU

- collect data from local and/or remote clients.
- provides a mean to configure the hardware managed by any of the client (online database)
- manages the run transitions functions during run operation (start, stop, etc.)
- provides a set of essential applications to control and monitor a data acquisition sequence
- records the collected data to various storage media
- provides programming interfaces of the data stream to different analysis packages (ROOTANA, manalyzer)
- provides a web interface and tools to build custom display for run monitoring and control

MIDAS COMPONENTS

COMPONENTS LIST

Buffer Manager

handles the experimental data transfer from the frontend to the backend

Message System

dedicated buffer system to handle MIDAS internal messages

Online Database (ODB)

database holding all user information related to a given experiment

Frontend Acquisition code

user code defining what is to be acquired/control over time during an active experiment

COMPONENTS LIST

RPC Server

interface connecting remote MIDAS client to your local experiment

Data Logger

MIDAS client handling the recording of the collected data to physical storage media

Data Analyzer

MIDAS client able to connect to a MIDAS data stream (or to a saved Midas data file) for data analysis

Run Control

data flow control

COMPONENTS LIST

History System

event history storage and retrieval

Alarm Systems

overall system and user alarm

Electronic Logbook

online experiment logbook

Run Sequencer

run manager for parametric runs

BUFFER MANAGER

The buffer manager consists of a set of library functions for event collection and distribution.

A buffer is a shared memory region in RAM, which can be accessed by several processes, called clients.

Processes sending events to a buffer are called producers. Processes reading events from the buffer are called consumers.

A buffer is organized as a FIFO (First-In-First-Out) memory. Consumers can specify which type of events they want to receive from a buffer. For this purpose each event in the data buffer contains a MIDAS header with an event ID and other pertinent information.

Buffers can be accessed locally through the shared memory or remotely via the MIDAS server mserver acting as an interface to that same shared memory.

MESSAGE SYSTEM

Any client can produce status or error messages with a single call using the MIDAS library.

These messages are then forwarded to any other clients who may be available to receive these messages, as well as to a central log file system.

The Message System is based on the buffer manager scheme, but with a dedicated header to identify the type of message.

ONLINE DATABASE (ODB)

All relevant data for a given experiment are stored in a central database called Online DataBase (ODB).

This database contains run parameters, logging channel information, condition parameters for front-ends and analyzers, slow control values, status and performance data and any information defined by the user.

The access to such a database can be remote, the connection is performed through an RPC layer.

The ODB is hierarchically structured, similar to a file system, with directories and sub-directories.

The data are stored in key/data pairs and data associated with a key can be of different types such as: byte, words, double words, float, strings, or arrays of any of those.

FRONTEND ACQUISITION CODE

The frontend program refers to a task running on a particular computer which has access to hardware equipment.

Each frontend can be composed of multiple equipments.

Several frontends can be attached simultaneously to a given experiment.

The frontend program is composed of a general framework which is experiment-independent, and a set of template routines for the user to fill in:

- register the given equipment(s) list to a specific MIDAS experiment.
- provide the means of collecting data from hardware sources defined by each equipment 'read' function.
- gather these data in a known event format (e.g. MIDAS) for each equipment.
- send these data to the buffer manager either locally or remotely.
- periodically collect statistics of the acquisition task, and send them to the ODB.

RPC SERVER

For remote access to a MIDAS experiment, a remote procedure call (RPC) server is available: mserver.

For each incoming connection it creates a new sub-process which serves this connection over a TCP link.

The MIDAS server not only serves client connections to a given experiment, but takes the experiment's name as a parameter meaning that only one MIDAS server is necessary to manage several experiments on the same node.

DATA LOGGER

The data logger mlogger is a client running on the backend computer receiving events from the buffer manager and saving them onto disk, tape or via FTP to a remote computer.

It supports several parallel logging channels with individual event selection criteria. Data can currently be written in different formats: MIDAS binary, ASCII, ROOT and DUMP.

Basic functionality of the logger includes:

- events number limit.
- run size limit.
- logging selection of particular events based on event identifier.
- recording of ODB values to a MIDAS History System
- dump the ODB at the begin-of-run and end-of-run states, as well as to a separate disk file in XML or ASCII format.

DATA ANALYZER

The analyzer takes care of receiving events, initializing the ROOT system and automatically booking N-tuples/TTree for all events.

Interface to user routines for event analysis is provided.

The same analyzer executable can be used to run online (where events are received from the buffer manager) and off-line (where events are read from file).

When running online, generated N-tuples/TTree are stored in a ring-buffer in shared memory. They can be analysed with ROOT without stopping the run.

RUN CONTROL

A basic program supplied in the package called odbedit provides a simple and safe means of interacting with the ODB for run control.

However, to access all the MIDAS capabilities, the MIDAS web-based run control utility mhttpd should be used.

HISTORY SYSTEM

The MIDAS History System is a recording function embedded in the MIDAS logger mlogger.

Parallel to its main data logging function of defined channels, the MIDAS logger can store slow control data and/or periodic events on disk file.

Each history entry consists of the timestamp at which the event has occurred, and the value(s) of the parameter to be recorded.

At any given time, history plots can be displayed through the web with the History Page of the MIDAS web-based Run Control utility mhttpd, or queried from the disk file through the MIDAS mhist utility.

ALARM SYSTEM

The MIDAS Alarm System is a built-in feature of the MIDAS server. It acts upon the description of the required alarm defined in the ODB.

The action triggered by the alarm is left to the user through the means of a detached script.

C/C++ API allow development of user defined alarms.

ELECTRONIC LOGBOOK

The electronic logbook is a feature which provides the experimenter an alternative way of logging his/her own information related to the current experiment.

The internal electronic logbook is a built-in feature of MIDAS, and the electronic logbook information is accessible from any web browser as long as the MIDAS web server.

An external electronic log ELOG can also be used.

RUN SEQUENCER

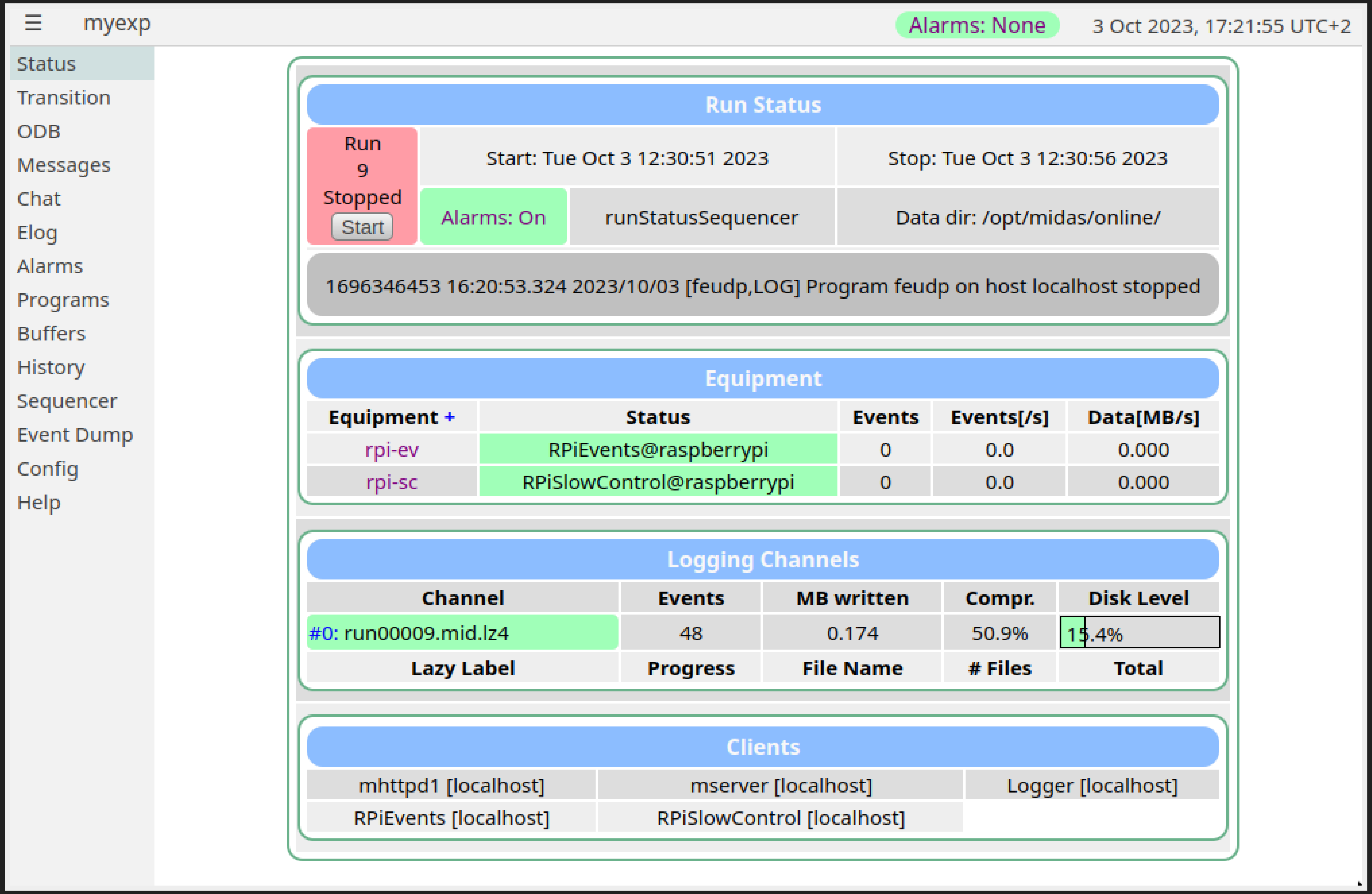
A sequencer for starting and stopping runs is available.

This allows the user to program a set of runs to be performed automatically.

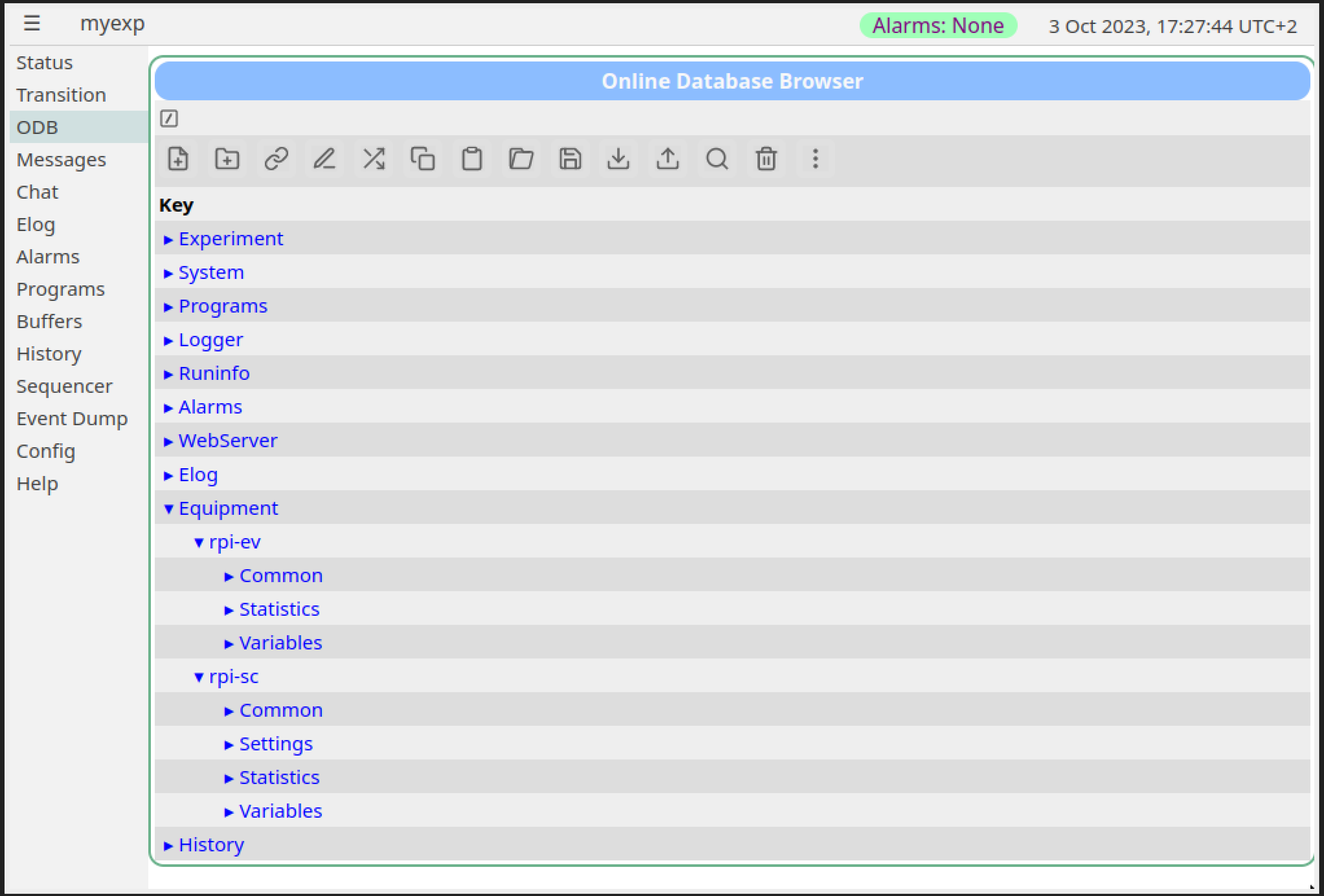
Conditions may be changed between runs, and each run may be stopped after a time or when a certain condition is reached.

MIDAS WEB INTERFACE

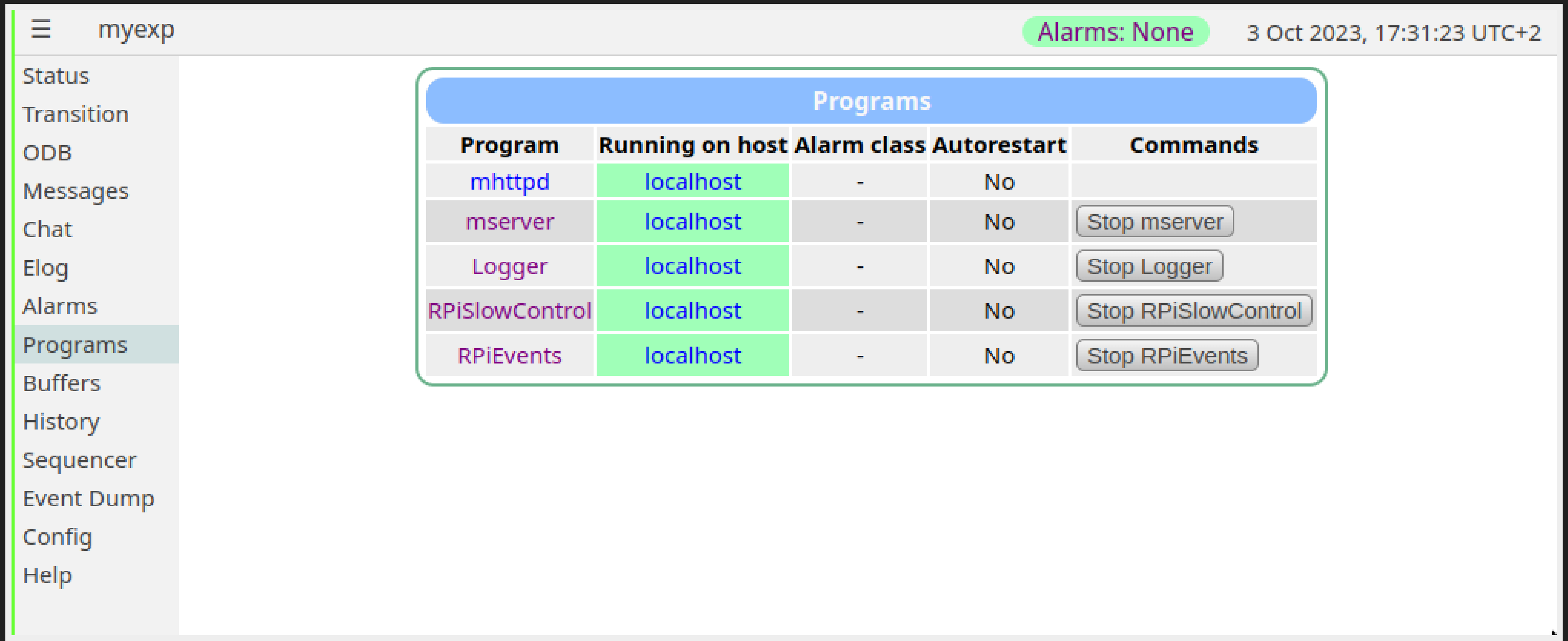
MAIN PAGE


myexp
Alarms: None
3 Oct 2023, 17:21:55 UTC+2
Status
Transition
ODB
Messages
Chat
Elog
Alarms
Programs
Buffers
History
Sequencer
Event Dump
Config
Help
Run Status
Run 9 Stopped
Start
Start: Tue Oct 3 12:30:51 2023
Stop: Tue Oct 3 12:30:56 2023
Alarms: On
runStatusSequencer
Data dir: /opt/midas/online/
1696346453 16:20:53.324 2023/10/03 [feudp,LOG] Program feudp on host localhost stopped
Equipment
Equipment +
Status
Events
Events[/s]
Data[MB/s]
rpi-ev
RPiEvents@raspberrypi
0
0.0
0.000
rpi-sc
RPiSlowControl@raspberrypi
0
0.0
0.000
Logging Channels
Channel
Events
MB written
Compr.
Disk Level
#0: run00009.mid.lz4
48
0.174
50.9%
15.4%
Lazy Label
Progress
File Name
Files
Total
Clients
mhttpd1 [localhost]
mserver [localhost]
Logger [localhost]
RPiEvents [localhost]
RPiSlowControl [localhost]

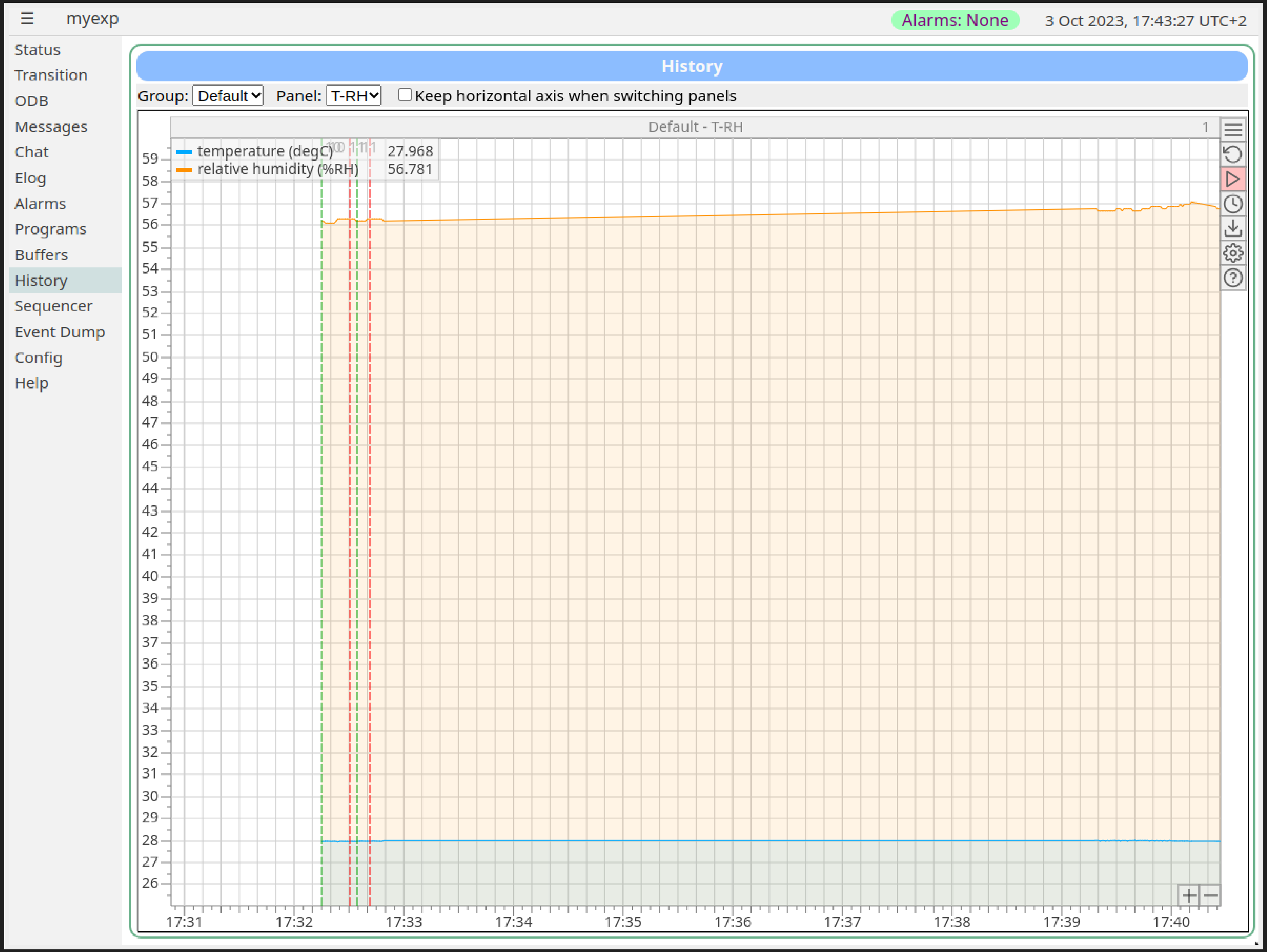
ODB


myexp
Alarms: None
3 Oct 2023, 17:27:44 UTC+2
Status
Transition
ODB
Messages
Chat
Elog
Alarms
Programs
Buffers
History
Sequencer
Event Dump
Config
Help
Online Database Browser
Key
Experiment
System
Programs
Logger
Runinfo
Alarms
WebServer
Elog
Equipment
rpi-ev
Common
Statistics
Variables
rpi-sc
Common
Settings
Statistics
Variables
History

PROGRAMS


myexp
Alarms: None
3 Oct 2023, 17:31:23 UTC+2
Status
Transition
ODB
Messages
Chat
Elog
Alarms
Programs
Buffers
History
Sequencer
Event Dump
Config
Help
Programs
Program
Running on host
Alarm class
Autorestart
Commands
mhttpd
localhost
-
No
mserver
localhost
-
No
Stop mserver
Logger
localhost
-
No
Stop Logger
RPiSlowControl
localhost
-
No
Stop RPiSlowControl
RPiEvents
localhost
-
No
Stop RPiEvents

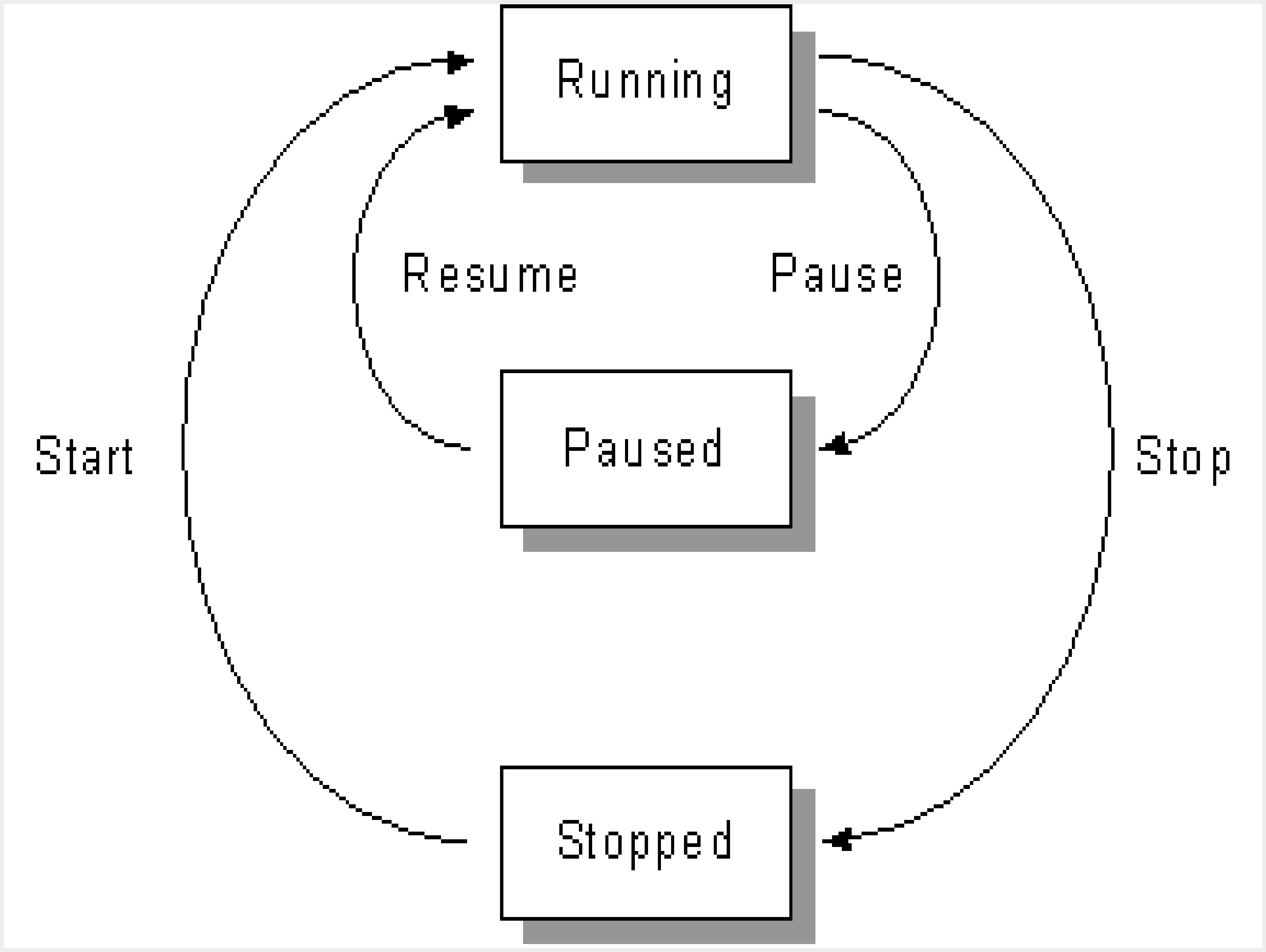
HISTORY



myexp
Alarms: None
3 Oct 2023, 17:43:27 UTC+2
Status
Transition
ODB
Messages
Chat
Elog
Alarms
Programs
Buffers
History
Sequencer
Event Dump
Config
Help
History
Group: Default
Panel: T-RH
Keep horizontal axis when switching panels
Default - T-RH
temperature (degC) 27.968
relative humidity (%RH) 56.781
17:31
17:32
17:33
17:34
17:35
17:36
17:37
17:38
17:39
17:40

MIDAS FRONTEND

RUN STATE MACHINE



MIDAS FRONTEND

The term frontend usually refers to a "frontend task" or program running on a particular computer which has access to hardware equipment in use by the experiment

An experiment may run several frontends, each performing different functions.

MIDAS FRONTEND

A frontend application consists of:

- a fixed experiment-independent system framework (i.e. mfe.c) handling the data flow control, data transmission and run control operation.
- a user part (e.g. frontend.c) written by the user describing the sequence of actions to acquire the hardware data

MIDAS EQUIPMENT

A single frontend may contain several equipments.

For example, an experiment may have a frontend to service crates of ADC, TDC and Scaler modules. The ADC and TDC modules may be grouped together in one equipment, and the scalers in a second equipment.

FRONTEND FEATURES

A typical frontend will:

- register the given equipment list(s) to a specific MIDAS experiment.
- provide the mean of collecting data from the hardware source defined by each Equipment read function.
- gather these data in one of the supported formats (i.e. FIXED format or in MIDAS data bank(s)) for each equipment.
- send these data banks to the buffer manager either locally or remotely.
- periodically collect statistics of the acquisition task, and send them to the ODB.

EQUIPMENT PARAMETERS

Each equipment has a predefined set of parameters (common parameters)

Equipment name

Each equipment name must be unique.
The name will be the reference name of the equipment generating the event.

Event ID

Each equipment has to be associated with a unique event ID.
The event ID will be part of the event header of that particular equipment.

Trigger Mask

When in use, each equipment is associated with a unique Trigger Mask. The Trigger Mask can be modified dynamically by the Readout Routine e.g. to define a sub-event type on an event-by-event basis.
This can be used to mix "physics events" and "calibration events" in one run and identify them later.
Trigger Mask is declared as 16-bit values.

Buffer

This field specifies the name of the buffer to which the event will be sent (usually SYSTEM buffer).

EQUIPMENT TYPE

EQ_PERIODIC

In this equipment type, no hardware requirement is necessary to trigger the readout function. Instead, the readout routine associated with this equipment is called periodically.

The Period field in the equipment declaration is used in this case to specify the time interval between calls to the readout function.

EQ_POLLED

In this equipment Type, the name of the routine polling on a trigger source is poll_event()".

This routine must be provided in the Frontend user code by the user.

The EQ_POLLED equipment type is mainly used for data acquisition based on a hardware condition becoming TRUE, at which time the readout routine associated with the equipment is called.

EQUIPMENT TYPE

EQ_INTERRUPT

This flag is similar to the EQ_POLLED mode, except a hardware interrupt is used to trigger the event rather than a polling loop.

Instead of passing a pointer to the polling routine, in EQ_INTERRUPT mode a pointer to the interrupt configuration routine is passed to the system.

EQ_MULTITHREAD

This flag implements the multi-threading capability within the frontend code.

The polling is performed within a separate thread and uses the MIDAS Ring Buffer Functions (rb_xxx in midas.c) for inter-thread communication.

EQ_MULTITHREAD is similar to EQ_POLLED mode, except for the polling function which in the case of EQ_MULTITHREAD resides in a separate thread.

This new type has been added to take advantage of the multi-core processor to free up CPU for tasks other than polling.

EQUIPMENT PARAMETERS

Format

This field specifies the data format used for generating the event. Only MIDAS and "FIXED formats are valid in the frontend.
The format must agree with the way the event is composed in the equipment Readout Routine.

Enabled

This Equipment List Parameter is the enable switch (true/false) for the equipment.

ReadOn

This field specifies when the read-out of an event occurs or is enabled.
It is possible to combine multiple ReadOn flags.

READ_ON Flag name	Value	Readout Occurs
RO_RUNNING	1	While running
RO_STOPPED	2	Before stopping run
RO_PAUSED	4	When run is paused
RO_BOR	8	At the beginning of run
RO_EOR	16	At the end of run
RO_PAUSE	32	Before pausing the run
RO_RESUME	64	Before resuming the run
RO_TRANSITIONS	127	At all transitions
RO_ALWAYS	255	Always (independent of the run status)
RO_ODB	256	Copies the event to the */Equipment/<equipment name>/Variables* ODB tree. The ODB is updated with a new event approximately every second. Note that this feature is generally used only for testing or monitoring, as writing large amounts of data to the ODB takes time.

EQUIPMENT PARAMETERS

Period

This field specifies the time interval for `EQ_PERIODIC` equipment or time out value in the case of EQ_POLLED or EQ_MULTITHREAD equipments (units are milliseconds).

Event limit

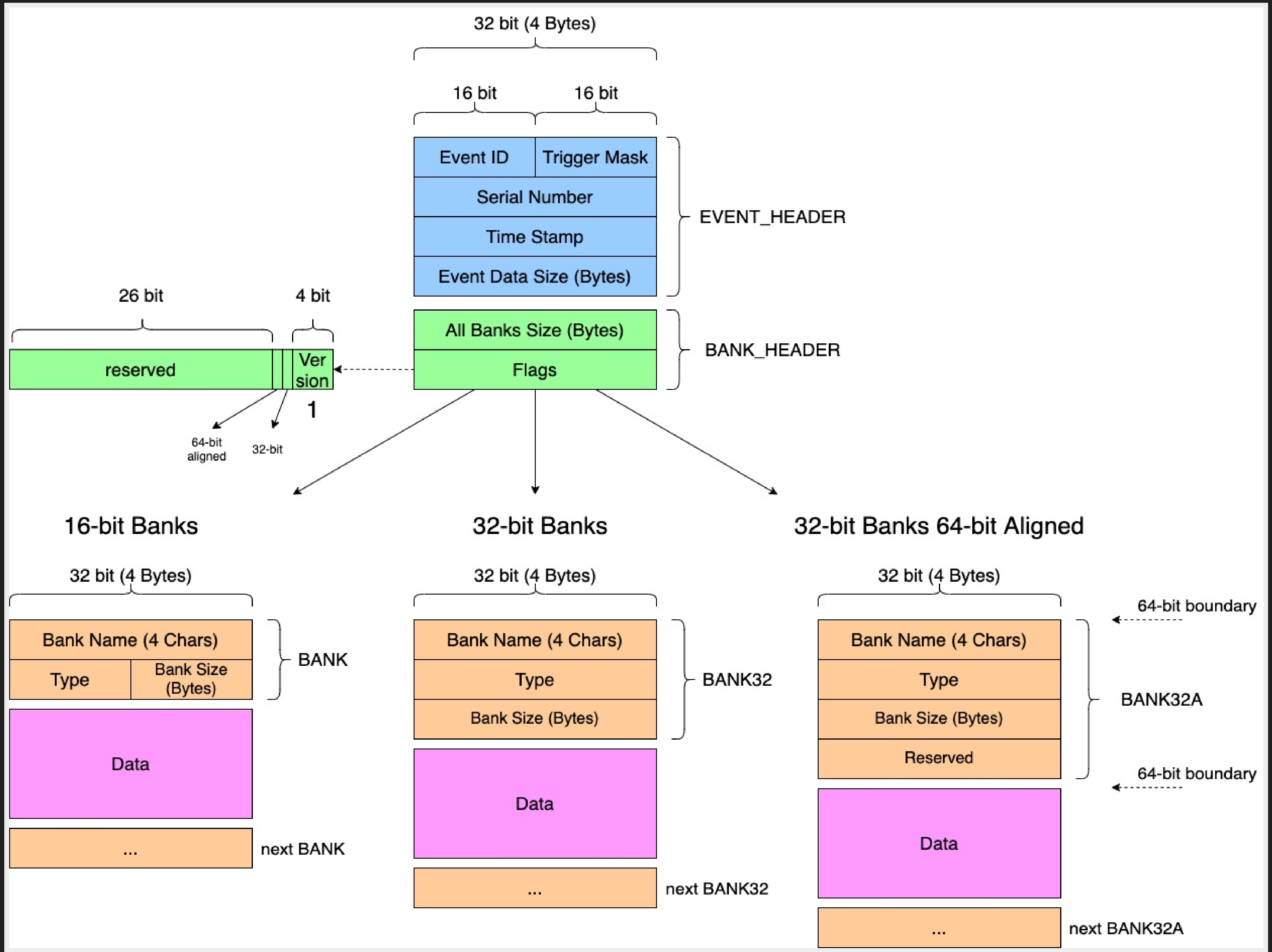
This Equipment List Parameter specifies the number of events to be taken prior to forcing an end-of-run transition. The value 0 disables this option.

Log history

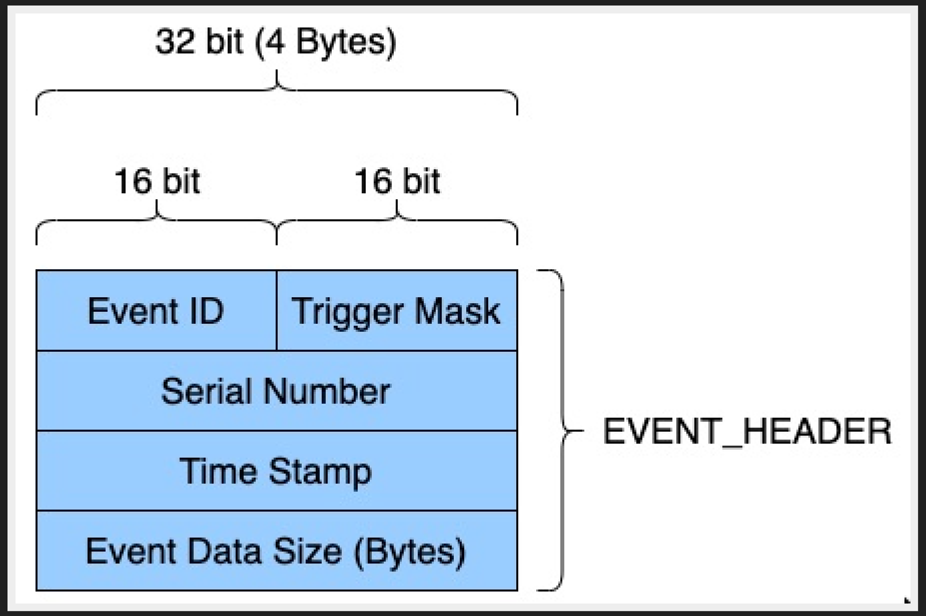
This parameter enable/disable the History System for that equipment. The value (positive in seconds) controls how frequently the history events are generated.
A positive value enables history logging, in which case the event data will also be sent automatically to the ODB in the `/Equipment/<equipment-name>/Variables` tree.

MIDAS EVENT FORMAT

MIDAS EVENT STRUCTURE



MIDAS EVENT HEADER



Event ID identifies the event by number.
Usually 1 is used for triggered events, 2 for scaler events, 3 for HV events etc.

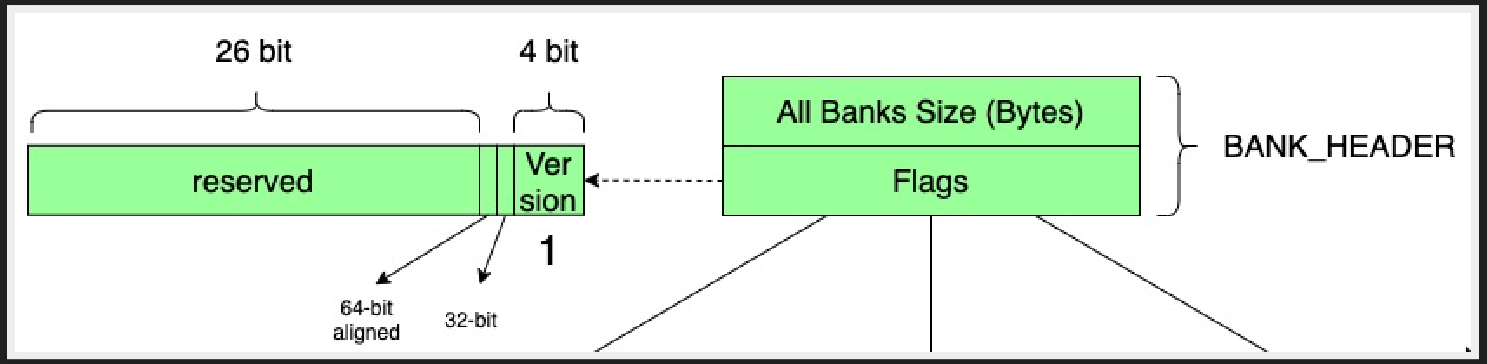
Trigger mask can be used to describe the sub-type of an event.
Consumers can request events with a specific triggering mask.

Serial number starts at 0 and is incremented by the front-end for each event.

Time stamp is written by the front-end before an event is read out.
It uses the time() function which returns the time in seconds since 1.1.1970 00:00:00 UTC.

Event data size contains the size of the event in bytes excluding the header.

MIDAS BANK HEADER



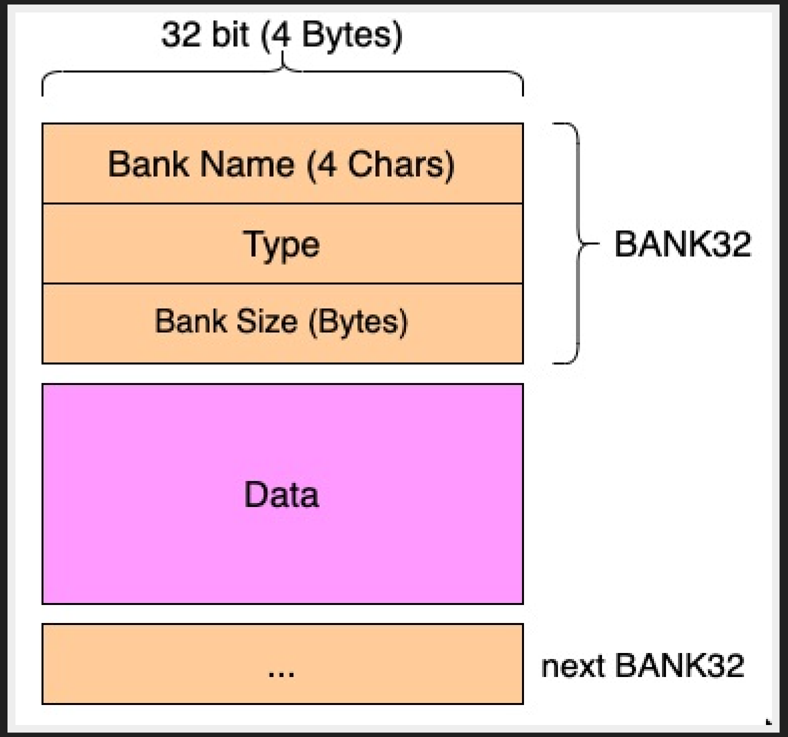
All banks size

Size in bytes of the following data banks including their bank names.

Flags

The four LSB 0:3 indicate the version of the bank structures.
This is currently "1" and used for endian detection (byte ordering).
The 5th bit indicates that the banks are 32-bit banks (Bank Size being 4 bytes long) and the 6th bit indicates that the banks are 64-bit aligned using the BANK32A bank header.

MIDAS BANK DATA



Bank name

Four characters for the name of each bank. Each bank in an event must have a unique name.

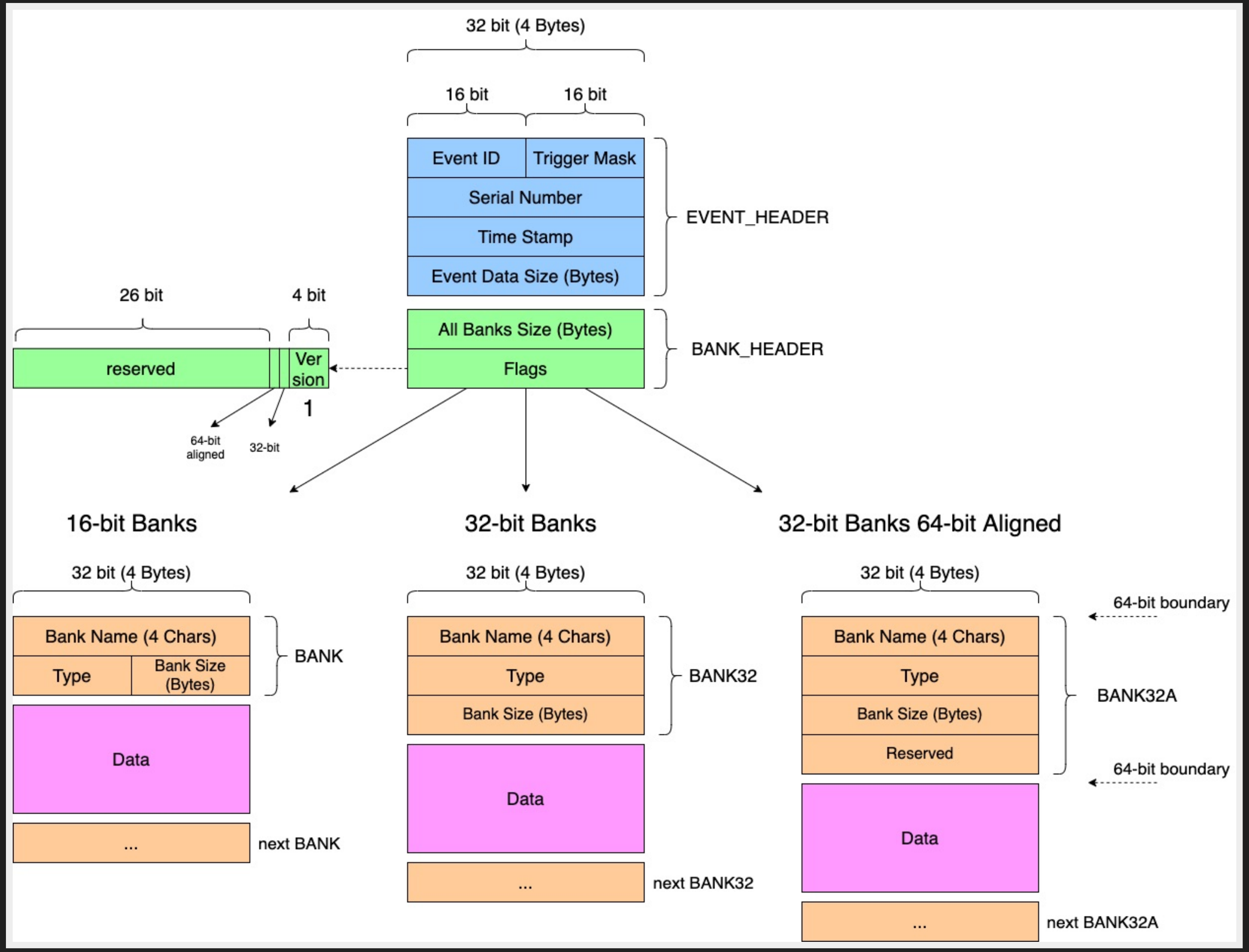
Bank type

One of the Midas Data Types TID_xxx values to encode the data type. A separate MIDAS bank must be created for each data type needed.

Bank size

Size in bytes of the following data.

EXAMPLE OF MIDAS EVENT




```
------------------------ Event# 2 --------------------------------
Evid:000d- Mask:0000- Serial:0- Time:0x4c7a6869- Dsize:48/0x30
\#banks:1 - Bank list:-SDAS-

Bank:SDAS Length: 32(I*1)/8(I*4)/8(Type)Type:Real*4 (FMT machine dependent)
  1-> 4.000e+00 1.000e+01 1.000e+00 3.400e+00 3.400e+00 3.400e+00 3.400e+00 3.400e+00

------------------------ Event# 3 --------------------------------
Evid:0001- Mask:0000- Serial:0- Time:0x4c7a686b- Dsize:344/0x158
\#banks:2 - Bank list:-MPETMCPP-

Bank:MPET Length: 304(I*1)/76(I*4)/76(Type) Type:Unsigned Integer*4
    1-> 0x80010000 0x00000002 0x10010000 0x00004e21 0x80020000 0x00000002 0x20020000 0x000015f4
  9-> 0x20020000 0x00001660 0x20020000 0x0000185f 0x20020000 0x0000191e 0x20020000 0x000019d6
 17-> 0x40020000 0x00001a37 0x20020000 0x00001a77 0x20020000 0x00001ba2 0x10020000 0x00004e22
 25-> 0x80030000 0x00000002 0x20030000 0x00001637 0x20030000 0x000018d1 0x20030000 0x000019bc
 33-> 0x20030000 0x00001b35 0x20030000 0x00001bb2 0x10030000 0x00004e21 0x80040000 0x00000002
 41-> 0x10040000 0x00004e22 0x80050000 0x00000002 0x20050000 0x000013c5 0x20050000 0x000017f2
 49-> 0x20050000 0x0000185f 0x20050000 0x00001976 0x20050000 0x00001aa8 0x10050000 0x00004e21
 57-> 0x80060000 0x00000002 0x20060000 0x000015c3 0x20060000 0x000018d8 0x20060000 0x0000198d
 65-> 0x20060000 0x00001ac4 0x10060000 0x00004e22 0x80070000 0x00000002 0x20070000 0x00001747
 73-> 0x20070000 0x000019ae 0x10070000 0x00004e21

Bank:MCPP Length: 16(I*1)/4(I*4)/4(Type) Type:Unsigned Integer*4
  1-> 0x00005e4c 0x0000352d 0x00006453 0x00006d5b
```

REFERENCES

- MIDAS wiki https://daq00.triumf.ca/MidasWiki/index.php/Main_Page
- MIDAS git repository https://bitbucket.com/tmidas/midas
- MIDAS forum https://midas.triumf.ca/forum