**RMD – HSFL Version FSW Test Commands**

This document will be a brief description of the commands to issue for collecting data with the Hawaii Instrument.

[1. Format Requirements 2](#_Toc10539626)

[2. Testing Commands 3](#_Toc10539627)

[2.1 System Parameter Loading 3](#_Toc10539628)

[2.2 Begin a DAQ Science Run 3](#_Toc10539629)

[2.3 End the DAQ Science Run 6](#_Toc10539630)

[2.4 Data Transfer Mode and Parameters 6](#_Toc10539631)

[2.5 Notes for Further DAQ Science Runs 7](#_Toc10539632)

[3. Test Command Set Table 8](#_Toc10539633)

# Format Requirements

Commands are sent to the instrument as ASCII statements. Bytes received by the RS422 are interpreted as ASCII characters.

All commands have parameters which follow the command. Those parameters can be either integers, floats, or strings, as documented below. All parameters are required, and will be parsed as the expected data type. Any input given to the command will be parsed as the data type given. If the number of parameters is too few or too many, the command will be rejected with a Command Failure packet (0x01).

* **The parsing relies on the “\_” (underscore) character.**

Parameters and naming sequences not separated by the underscore will not be recognized by the system.

The number which follows almost all the commands is the Detector ID (0 or 1) indicated as “#” (pound) in the command list below. The Mini-NS is comprised of two separate boards and commands need to be addressed to one or the other board.

Each board will parse, but silently reject commands which are addressed to the other board.

* **Commands must be terminated with a newline “\n” character.**

On the completion of a command, the return value will be written to the RS422 wrapped in a CCSDS packet; these return values are listed below each command. The success or failure of a command is indicated via the APID bytes (ie. the type of packet returned), see the appendix. All commanded functions return a CCSDS packet upon completion, but some output a packet other than SUCCESS if successful (eg. GETSTAT returns an SOH packet instead of a SUCCESS packet).

# Testing Commands

These commands assume that the board is powered and in Standby Mode.

## System Parameter Loading

Parameters to set and command strings to set those parameters:

Set the Trigger Threshold

MNS\_TRG\_0\_8850\n

Integration times

MNS\_INT\_0\_-52\_88\_472\_6000\n

These are the typical integration times

HV pot values

MNS\_ENABLE\_ACT\_0\n

MNS\_HV\_0\_3\_196\n

This sets the HV on the pot to -766.8 Volts

Energy calibration

MNS\_ECAL\_0\_1.0\_0.0\n

This gives us no energy calibration, which is acceptable for this test

Neutron cuts

To set the regular box cuts:

MNS\_NGATES\_0\_0\_0\_20000\_600000\_0.05\_0.4\n

To set the wide box cuts:

MNS\_NGATES\_0\_0\_1\_16000\_720000\_0.04\_0.48\n

There are two “boxes” which we can move around using these cuts to define an area which is “neutron sensitive”.

NOTE: The values for the neutron cut windows are wide and may catch more than just neutrons. This is intentional and will only affect the number of counts we see in the CPS files, which will be above what would normally be expected (due to a larger window).

Once these values are set, the system has an internal configuration file which will be updated. After receiving a command success from one of these commands, the values is now set within the system until is updated again. Thus, the user need only call these functions once each time that it needs to be set.

## Begin a DAQ Science Run

Move the system to DAQ mode:

MNS\_DAQ\_0\_1\n

This gives the DAQ run an ID number of “1” which we will keep track of

**NOTE**: an important note is that when DAQ is called, it loops trying to find a unique combination of ID and run numbers so that it can create a new directory to store the data products in. The system will return a Command Success packet containing the folder name for that run which follows the format shown in section 2.4 below. This is how we know what the run number is. The other way is to keep track of how many times MNS\_DAQ\_ is called. The run number increments each time DAQ is called.

Format of the Command Success Packet returned from the DAQ command:

|  |  |  |  |
| --- | --- | --- | --- |
| Byte | Description | Data Type | Group |
| 0-3 | Sync Marker | Byte | Primary CCSDS Header |
| 4 | Flags and Detector Number |
| 5 | APID |
| 6 | Group Flags and Sequence Count MSB |
| 7 | Sequence Count LSB |
| 8 | Packet Length MSB |
| 9 | Packet Length LSB |
| 10 | Reset Request Flag | Secondary CCSDS Header |
| 11-… | ASCII String: Commanded Input | ASCII String | Data |
| … | “\n” | Signed Char |
| … | ASCII String: Folder Name | ASCII String |
|  | Simple Checksum | Byte | RMD Data Checksums |
|  | Fletcher Checksum |
|  | CCSDS Checksum MSB | Checksum |
|  | CCSDS Checksum LSB |

Currently, what is returned is the command which was sent by the flight computer (the commanded input), a newline character, and the folder name for the data acquisition run. Because this was missed when changing over to fixed field format, the fields labeled as Commanded Input will grow with the input command. The ASCII string labeled Folder Name will always be the same size, and the format of that string is in section 2.4 below.

On the next page, Table 1 is an example of what a command success packet returned from a MNS\_DAQ command would look like. This assumes the following commanded input:

MNS\_DAQ\_0\_1

Table 1: Example Command Success Packet

|  |  |  |
| --- | --- | --- |
| Byte | Description | Group |
| 0 | 0x35 | Sync Marker |
| 1 | 0x2E |
| 2 | 0xF8 |
| 3 | 0x53 |
| 4 | 0x0A | Primary CCSDS Header |
| 5 | 0x00 |
| 6 | 0xC0 |
| 7 | 0x01 |
| 8 | 0x00 |
| 9 | 0x1E |
| 10 | 0x00 | Secondary CCSDS Header |
| 11 | 0x4D | Mini-NS Data Bytes |
| 12 | 0x4E |
| 13 | 0x53 |
| 14 | 0x5F |
| 15 | 0x44 |
| 16 | 0x41 |
| 17 | 0x51 |
| 18 | 0x5F |
| 19 | 0x30 |
| 20 | 0x5F |
| 21 | 0x31 |
| 22 | 0x0A |
| 23 | 0x30 |
| 24 | 0x3A |
| 25 | 0x2F |
| 26 | 0x49 |
| 27 | 0x30 |
| 28 | 0x30 |
| 29 | 0x30 |
| 30 | 0x31 |
| 31 | 0x5F |
| 32 | 0x52 |
| 33 | 0x30 |
| 34 | 0x30 |
| 35 | 0x30 |
| 36 | 0x31 |
| 37 | Cs1 | RMD Simple Checksum |
| 38 | Cs2 | RMD Fletcher Checksum |
| 39 | Cs3 | CCSDS Checksum MSB |
| 40 | Cs4 | CCSDS Checksum LSB |

Begin the run with:

MNS\_START\_0\_realTime\_1\n

RealTime should be a time from the flight computer. For now, we can use any unsigned long long value.

This gives the system a 1 minute timeout value. When we have looped for 60 seconds the system will gracefully quit DAQ.

## End the DAQ Science Run

There are three options for ending a DAQ run:

Timeout

Just wait the number of minutes specified to the detector via the START command.

Ending the DAQ run

MNS\_END\_0\_realTime\n

RealTime should be a time from the flight computer. For now, we can use any unsigned long long value.

Breaking out of the process:

MNS\_BREAK\_0

Choose one and send the command (or wait), the system will finalize the run and return to standby mode.

## Data Transfer Mode and Parameters

Using the values from the previous DAQ run, we first want to identify the folders/files that were created so we can properly request the files back. The folders are created via the following formula:

“0:/I0000\_R0000”

* The “0:/” specifies the root directory on the SD card.
* “I” is an uppercase letter i, which stands for ID Number. This is the unique identifier that we sent with the DAQ\_ command
* “0000” is the ASCII representation of the ID Number.
* “R” is an uppercase letter r, which stands for Run Number. This is an internal number that is tracked by the detector and incremented each time a DAQ run is started during a power cycle.
* “0000” is the ASCII representation of the Run Number

As an example of a folder name: if we choose and specify an ID Number of 12 and the this is the third time we have started a DAQ run during this particular power cycle we would have a folder name of:

“0:/I0012\_R0003”

With the knowledge of the file we want and its name, we can construct the parameters to give to transfer so we get the proper file back.

Choose a file type to request:

* 5 = counts per second file
* 6 = Waveforms
* 7 = event-by-event file
* 8 = 2D histogram from PMT 1
* 11 = 2D histogram from PMT 2
* 12 = 2D histogram from PMT 3
* 13 = 2D histogram from PMT 4

This file type goes into the file type parameter

Then place the ID and Run numbers into their respective fields

The values for the set number are only important for the event-by-event data product, for all other data products, these values are 0.

To specify the set number low, start at 0 and transfer files until a command failure is returned, this indicates there are no more files from that sequence. The number of files generated by event-by-event data can vary, but can be up to and more than 75 for a long run.

At the moment, set number high is not used, so always set this to 0

For example, to get the counts per second data:

MNS\_TX\_0\_5\_ID\_Run\_0\_0

Place the integer values for the ID and Run number in. The ID should be known, the run number is the number of times the DAQ command has been issued during the current power cycle. The set numbers should be 0.

To get the 2D histogram from PMT 1:

MNS\_TX\_0\_8\_ID\_Run\_0\_0

To get the first EVT data product set file:

MNS\_TX\_0\_7\_ID\_Run\_0\_0

And then…

MNS\_TX\_0\_7\_ID\_Run\_1\_0 and so forth.

## Notes for Further DAQ Science Runs

There is a “false” event generated the first time that DAQ is called each time the board is power cycled. Thus, each subsequent DAQ run that power cycle will not have a “false” event present in the data products.

The log and configuration files can’t be transferred with this release.

# Test Command Set Table

The following table shows a set of commands that can be issued to the system to set the system parameters, take data, and transfer the data out.

|  |  |
| --- | --- |
| Command | Description |
| MNS\_TRG\_0\_8850\n | Set trigger threshold to 8850 |
| MNS\_INT\_0\_-52\_88\_472\_6000\n | Set integration times |
| MNS\_ENABLE\_ACT\_0\n | Enable the 3.3 V |
| MNS\_HV\_0\_1\_196\n | Set the high voltage potentiometer to -766.8 Volts |
| MNS\_ECAL\_0\_1.0\_0.0\n | Set energy calibration parameters |
| MNS\_NGATES\_0\_0\_0\_20000\_600000\_0.05\_0.4\n | Set neutron cut gates |
| MNS\_NGATES\_0\_0\_1\_16000\_720000\_0.04\_0.48\n | Set wide neutron cut gates |
| MNS\_DAQ\_0\_1 | Change system to data acquisition mode |
| MNS\_START\_0\_123456\_5 | Begin data acquisition; take data for 5 minutes |
| MNS\_END\_0\_654321 | End data acquisition |
| MNS\_TX\_0\_5\_1\_1\_0\_0 | Transfer CPS data product file |
| MNS\_TX\_0\_7\_1\_1\_0\_0 | Transfer EVT data product file |
| MNS\_TX\_0\_8\_1\_1\_0\_0 | Transfer 2DH data product file for PMT 1 |
| MNS\_TX\_0\_11\_1\_1\_0\_0 | Transfer 2DH data product file for PMT 2 |
| MNS\_TX\_0\_12\_1\_1\_0\_0 | Transfer 2DH data product file for PMT 3 |
| MNS\_TX\_0\_13\_1\_1\_0\_0 | Transfer 2DH data product file for PMT 4 |

For the commands above, the following variables are identified:

* Detector Number = 0
* ID Number = 1
* Run Number = 1
* START real time = 123456
* END real time = 654321

For the file transfers the end result should be:

* CPS: 1 packet
* EVT: 528 packets
* 2DH: 8 packets

The files on the SD card should be the following sizes:

* CPS: 4,444 bytes – for a 5 minute run
* EVT: 1,048,608 bytes – for a set file
  + Each EVT data product is split into many set files, each file is 1 MB plus a header
  + The final EVT file will be smaller
* 2DH: 15,808 bytes – for each PMT