

**Voyager3-1Z Developers Firmware Guide**

*Version 0.3*

Created: 6/4/2020

Author: Sharkey, Tom

Last Modified: 16/8/2021

Modified by: McCarthy, Jack

# Overview

This guide serves as a companion document to the Voyager 1Z Firmware guide. It explains the operation of the Voyager 1Z in a greater level of detail than the original firmware guide, and is aimed towards helping developers who may seek to make changes to the operation of the mote and manager system, and to understand the system of communication between the two in a greater level of detail.

It is advised that the reader has already explored the basic documentation for the Voyager 1Z.

* WCBM-01 Firmware Guide
* CBM\_Setup

These documents are available on the [Voyager-1Z Wiki.](https://wiki.analog.com/resources/eval/user-guides/industrial_wireless_cbm_evaluation_tool_3axis#documentation)

Contents

[Overview 2](#_Toc80024015)

[1. High Level Overview of Operation 3](#_Toc80024016)

[2. Header 4](#_Toc80024017)

[2.1 Framing 4](#_Toc80024018)

[2.2 Format of Header 5](#_Toc80024019)

[2.2.1 Axis 6](#_Toc80024020)

[2.2.2 Version 6](#_Toc80024021)

[3. Downstream communication 7](#_Toc80024022)

[3.1 Sampling Frequency 8](#_Toc80024023)

[3.2 Number of ADC Samples 8](#_Toc80024024)

[3.3 Sleep Duration 8](#_Toc80024025)

[3.4 Alarm Triggered 8](#_Toc80024026)

[3.5 Axes Received 10](#_Toc80024027)

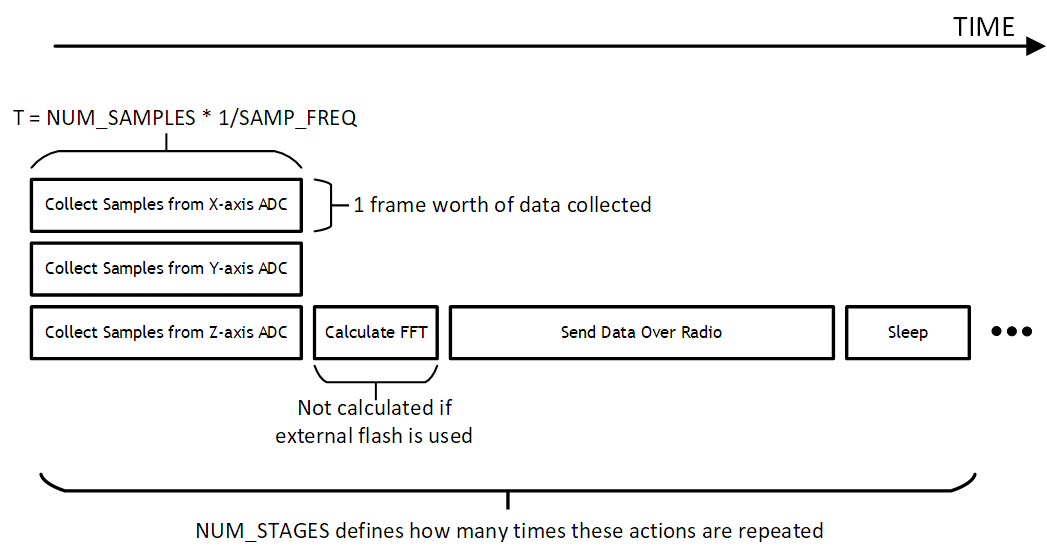
[3.6 Updating Parameters 12](#_Toc80024028)

# 1. High Level Overview of Operation

A summary of the high-level operation of Voyager 3 can be seen in the below figure. Note, the sequence of events shown in this figure are collectively known as a **Stage**.

When a mote is connected to the manager via the GUI, Stages are repeated until the GUI is disconnected.

If the mote is connected to the manager through the Terminal Mode, the user passes in the argument NUM\_STAGES to control how many times to repeat the sequence.



Voyager 3 Timeline of Events. This Collection of Events is Called a Stage

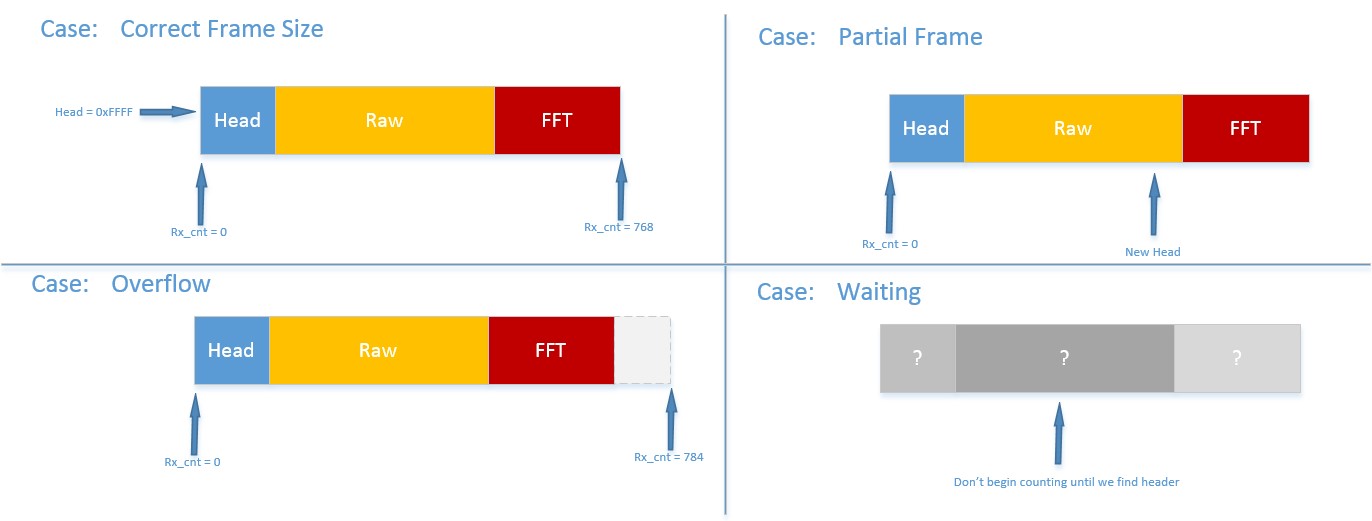
# 2. Header

## 2.1 Framing

Normal operation from mote to manager results in a constant stream of acceleration data, separated into raw data and FFT data. Note if the number of samples per frame exceeds 1024 then a flash chip external to the microcontroller’s is used and FFT calculations are no longer performed

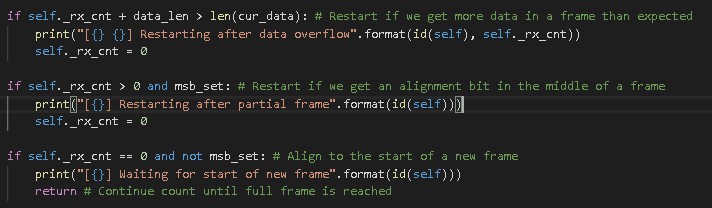
In order to create complete “frames”, such that streamed data can be easily be interpreted by the GUI, the data must be prepended with a header which contains information about the data contained within the frame. SmartMesh places a limit on the size of the data that can be sent to 90 byte packets. Each individual piece of raw data is a 16-bit integer, corresponding to a reading from the on-board ADC.

A header byte is attached to the beginning of a frame which contains information about the frame and creates a separation between frames, as demonstrated in Figure 1.



*Figure 1: Forming a correct Data Frame*

These checks are reflected in the code for the Python GUI, as shown in Figure 2 . Here, \_rx\_cnt is a variable which counts the number of bytes that have been received. This process takes place in the \_handle\_bytes function within the Mote class. If any of the conditions are met which indicate that the frame is not the correct size, \_rx\_cnt is reset to 0.



*Figure 2: Forming correct Data Frame (GUI)*

## 2.2 Format of Header

The header is broken into 3 parts: The pattern marking the beginning of the header; the pattern which marks the axis of the current data; and the pattern which marks the version of the firmware.



16

-

bit Header

0

xFF

92

1111 1111 1001 0011



1111 1111



1001



0011

Marks beginning of header

Marks axis of data

[

X

,

Y

,

Z

]

Indicates firmware version

(.

03

)

*Figure 3: Format of a Header*

*Sample headers*

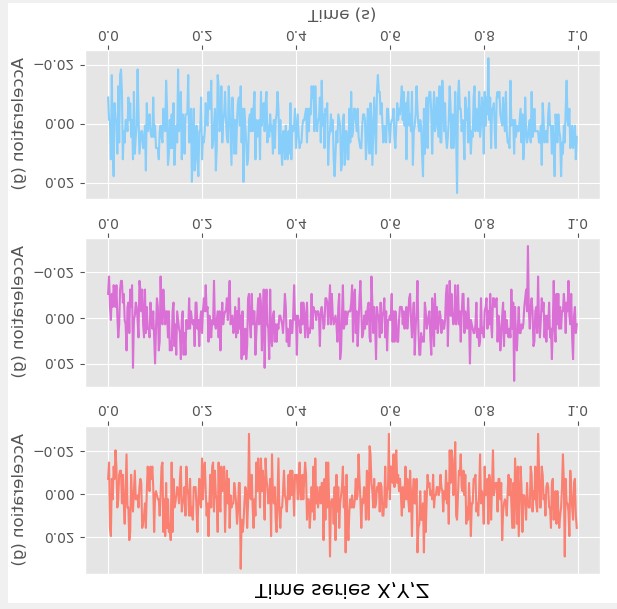
|  |  |  |
| --- | --- | --- |
| Header (Hex) | Binary | Interpretation |
| 0xFF93 | 0b 1111 1111 1001 0011 | Header frame, x-axis, version x.03 |
| 0xFFA4 | 0b 1111 1111 1010 0100 | Header frame, y-axis, version x.04 |

*Table 1: Sample Headers*

As the unique all-1’s pattern of the top 8 bits is used to signify a header, if this pattern occurs in a non-header packet, a partial frame error will occur. However, this value will only occur in the data if the accelerometer reads a value above 99.6% of its maximum value.

### 2.2.1 Axis

- 0x0090  0000 0000 1001 0000 - 0x00A0  0000 0000 1010 0000 - 0x00C0  0000 0000 1100 0000



1001

1010

1100

*Figure*

*4*

*:*

*Axis Selection to control Plotting*

The top 4-bits of the lower byte in the header signify the axis to which the data belongs, as the accelerometer records data in 3 axes. This information can be used by GUI to choose where to plot the relevant raw and FFT data. Importantly, the GUI allows for the sending of specific axes of data.

### 2.2.2 Version

*Examples*

* 0x0000 0000 0000 0002
* 0x0000 0000 0000 0003

The final 4 bits of the header are reserved for the version number of the currently running firmware. This is used as a means of confirming correct operation between firmware and GUI, which must be updated in tandem for the correct operation of the system. This version system is simplistic in order to communicate the version of the program in the least number of bits possible, and so keep the header overhead small. The version number sent is simply the two numbers after the decimal point for the version (1.03). As only four bits are available, this version number must not exceed .15.

N.B. The version of the firmware and the GUI must be updated as changes are made to the system. Both versions are constants defined at the top of the GUI and in the

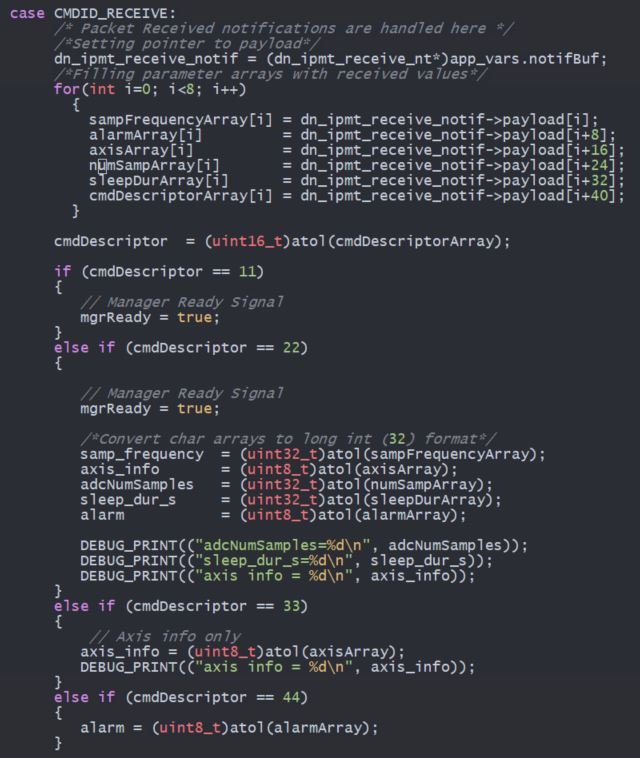
SmartMesh\_RF\_Cog.h file of the firmware respectively. If the versions of GUI and firmware do

not match, an error is issued to the user in the GUI command window.

# 3. Downstream communication

In order to control the operation of individual motes, a command packet must be sent to each mote from the GUI. This command packet is used whenever the sampling frequency must be updated for FFT calculation, when an alarm must be triggered, and to communicate which axis information the GUI wants to receive.

Any update to mote operation is sent in the form of a command message, which triggers a specific callback in the motes firmware, as shown in Figure 5.



*Figure 5: Mote Command Received*

## 3.1 Sampling Frequency

The sampling frequency is updated by the user and used to control the time between ADC acquisitions. The updated sampling frequency sent by the user is used to adjust a timer controlling ADC readings. The sampling frequency can be programmed with a minimum granularity of 1Hz.

*Examples*

Sampling Frequency (fS = 1000Hz)

## 3.2 Number of ADC Samples

The number of samples to be collected per enabled axis i.e., the number of ADC data samples per frame can be controlled by the user from version 0.5 onwards.

Note, if a number larger than 1024 is selected then the data is stored on a flash memory chip that is external to the microcontroller and therefore **FFT calculations will not be performed**.

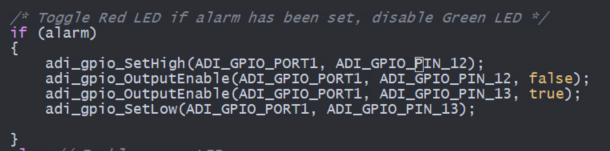
## 3.3 Sleep Duration

This controls the duration that the microcontroller will enter a low power sleep mode after it has finished transmitting all of the collected ADC data and FFT calculations i.e., after a Stage has been completed. This enables more realistic use-cases where the mote can wake up at user defined intervals and collect a user-defined number of samples before returning to a low power state.

The microcontroller’s real-time clock is used to wake the microcontroller up after the selected number of seconds. The minimum granularity of this clock is one second.

## 3.4 Alarm Triggered

The first check that occurs after a command has been received in the firmware is to see if the alarm variable has been set. An alarm command is sent from the GUI to the Mote that triggered it, if a Mote value such as “Peak” acceleration exceeds a user-set value. The LED used is the same green LED that blinks to indicate a successful network has been developed between a mote and a manager. The green LED is disabled on Port 1, Pin 12, and the red LED is enabled on Port 1, Pin 13, as shown in Figure 6.



*Figure 6: Setting Red LED (Smartmesh\_RF\_Cog.c)*



*Figure 7: Alarm Triggered*

If the alarm variable is not set, the green LED returns to normal operation, blinking if a network still exists between mote and manager.

## 3.5 Axes Received

Based on the axes that the user is interested in, only specific data is communicated by each mote. This can be any combination of the X, Y and Z axes. As normal operation for the mote is to send all 3 axes in order, changing which axis are sent simply adjusts the pointer to the data, as shown in Figure 8.



X



Y



Z

pByteBuf

Case

:

XYZ



X



Y



Z

pByteBuf

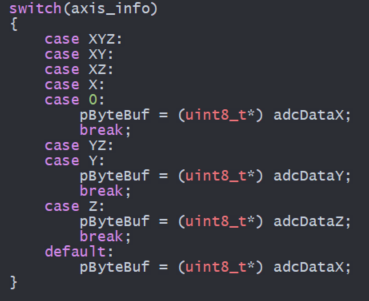
Case

:

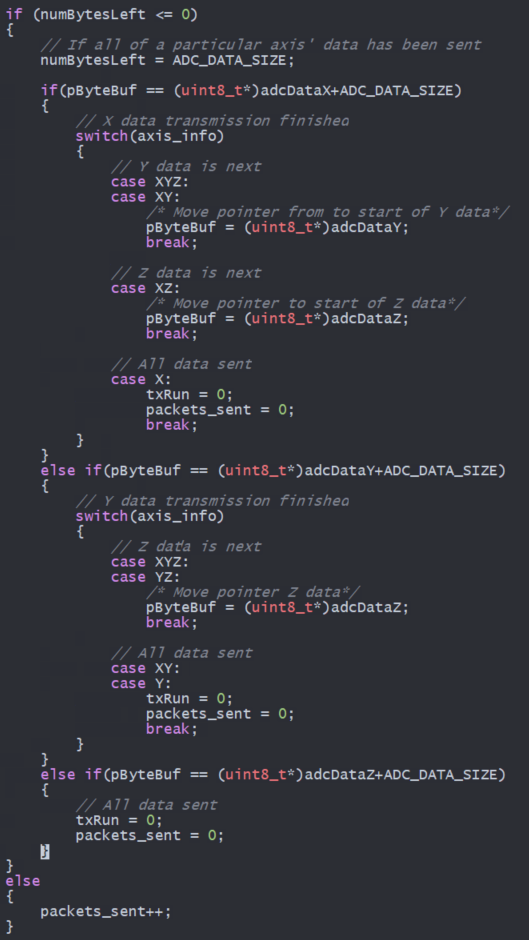
YZ

*Figure 8: Sending User Defined Axes*

As normal operation for the mote is to send all three axis in the order X  Y  Z, variations to this operation simply control the buffer to begin at, and the buffers to exclude, as shown in Figure 9. When all buffers have been sent, the variable txRun is set to 0 signalling the end of transmission to the data handling state machine.



*Figure 9: Select Axis Buffer to Begin Sending*



*Figure 10: Selecting Axis Buffers to Exclude*

## 3.6 Updating Parameters

Mote variables such as sampling frequency, number of samples per frame and sleep duration are updated within the “NEW\_PARAMS” state of the data handling state machine in main\_prog.c. This prevents errors that can occur when a firmware variable is updated instantly upon receiving a downstream command. By defining a specific state for updating these variables, the time at which these updates occur is known to the user, preventing timing-related errors.

As the red LED that signals an alarm (e.g. the mote exceeding a certain value peak acceleration) can be updated at any time, the check for this alarm flag occurs directly after the setting of downstream variables in SmartMesh\_RF\_cog.c.