User Manual

CompassPoint Prime

3-Axis Electronic Compass Module

PNI SENSOR CORPORATION

Table of Contents

List of Tables

List of Figures

1 Copyright & Warranty Information

Revised July 2011. For most recent version visit our website at www.pnicorp.com

PNI Sensor Corporation
133 Aviation Blvd, Suite 101
Santa Rosa, CA 95403, USA
Tel: (707) 566-2260
Fax: (707) 566-2261

Warranty and Limitation of Liability. PNI Sensor Corporation ("PNI") manufactures its Prime products (“Products”) from parts and components that are new or equivalent to new in performance. PNI warrants that each Product to be delivered hereunder, if properly used, will, for one year following the date of shipment unless a different warranty time period for such Product is specified: (i) in PNI’s Price List in effect at time of order acceptance; or (ii) on PNI’s web site (www.pnicorp.com) at time of order acceptance, be free from defects in material and workmanship and will operate in accordance with PNI’s published specifications and documentation for the Product in effect at time of order. PNI will make no changes to the specifications or manufacturing processes that affect form, fit, or function of the Product without written notice to the OEM, however, PNI may at any time, without such notice, make minor changes to specifications or manufacturing processes that do not affect the form, fit, or function of the Product. This warranty will be void if the Products’ serial number, or other identification marks have been defaced, damaged, or removed. This warranty does not cover wear and tear due to normal use, or damage to the Product as the result of improper usage, neglect of care, alteration, accident, or unauthorized repair.

THE ABOVE WARRANTY IS IN LIEU OF ANY OTHER WARRANTY, WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE. PNI NEITHER ASSUMES NOR AUTHORIZES ANY PERSON TO ASSUME FOR IT ANY OTHER LIABILITY.

If any Product furnished hereunder fails to conform to the above warranty, OEM’s sole and exclusive remedy and PNI’s sole and exclusive liability will be, at PNI’s option, to repair, replace, or credit OEM’s account with an amount equal to the price paid for any such Product which fails during the applicable warranty period provided that (i) OEM promptly notifies PNI in writing that such Product is defective and furnishes an explanation of the deficiency; (ii) such Product is returned to PNI’s service facility at OEM’s risk and expense; and (iii) PNI is satisfied that claimed deficiencies exist and were not caused by accident, misuse, neglect, alteration, repair, improper installation, or improper testing. If a Product is defective, transportation charges for the return of the Product to OEM within the United States and Canada will be paid by PNI. For all other locations, the warranty excludes all costs of shipping, customs clearance, and other related charges. PNI will have a reasonable time to make repairs or to replace the Product or to credit OEM’s account. PNI warrants any such repaired or replacement Product to be free from defects in material and workmanship on the same terms as the Product originally purchased.

Except for the breach of warranty remedies set forth herein, or for personal injury, PNI shall have no liability for any indirect or speculative damages (including, but not limited to, consequential, incidental, punitive and special damages) relating to the use of or inability to use this Product, whether arising out of contract, negligence, tort, or under any warranty theory, or for infringement of any other party’s intellectual property rights, irrespective of whether PNI had advance notice of the possibility of any such damages, including, but not limited to, loss of use, revenue or profit. In no event shall PNI’s total liability for all claims regarding a Product exceed the price paid for the Product. PNI neither assumes nor authorizes any person to assume for it any other liabilities.

Some states and provinces do not allow limitations on how long an implied warranty lasts or the exclusion or limitation of incidental or consequential damages, so the above limitations or exclusions may not apply to you. This warranty gives you specific legal rights and you may have other rights that vary by state or province.

2 Introduction

Thank you for purchasing PNI's CompassPoint™ Prime 3-axis electronic compassing module. Incorporating 3-axis magnetic field sensing and 3-axis tilt sensing, the Prime provides accurate and precise tilt-compensated heading measurements at up to 45° of tilt. The Prime utilizes PNI's advanced magnetic distortion correction algorithms to provide accurate heading information when incorporated into a user's system, even when the compass is being tilted. With its small size, the Prime is capable of fitting into today's size sensitive systems. These advantages make PNI Sensor Corporation's Prime the choice for applications that require a low price with unmatched performance.

The Prime's advantages make it suitable for a variety of applications, including:

- Sonobuoys
- Seismic monitoring systems
- Acoustic Doppler current profilers (ADCPs)
- Robotic systems

With its many potential applications, the Prime provides a command set designed with flexibility and adaptability in mind. Many parameters are user-programmable, including reporting units, a wide range of sampling configurations, output damping, and more. We believe the Prime will help you achieve great performance from your system.

3 Specifications

3.1 Performance Specifications

Table 3-1: Specifications[1]

Parameter			Value
Heading	Range		360°
	Accuracy, tilt ≤45°		1° rms
	Resolution		0.1°
	Repeatability		0.05° rms
Tilt (Pitch & Roll)	Range	Pitch	±90°
		Roll	±180°
	Accuracy		1° rms
	Resolution		0.1°
	Repeatability		0.05° rms
Maximum Functional Dip Angle[2]			85°
Magnetometer	Usable Field Range		±100 µT
	Resolution		0.05 µT
	Repeatability		0.1 µT

Footnotes:

1. Specifications are subject to change. Assumes the Prime is motionless and the local magnetic field is clean relative to the calibration.
2. Performance will degrade somewhat as the maximum functional dip angle is approached.

3.2 Operating Characteristics

Table 3-2: I/O Characteristics

Parameter		Value
Communication Interface		Binary RS232
Communication Rate		300 to 115,200 baud
Maximum Sample Rate		10 samples/second
Maximum Receive Voltages		±30 V
Threshold Voltages	Receive Low	0.6 V maximum
	Receive High	2.4 V minimum
	Transmit Low[1]	-5.0V maximum
	Transmit High[1]	+5.0 V minimum
Time to Initial Good Data[2]	Initial power up	<180 ms
	Sleep mode recovery	<60 ms

Footnotes:

1. Transmit lines with ≥3k Ω load to ground.
2. FIR Taps set to 0.

Table 3-3: Power Requirements

Parameter		Value
DC Supply Voltage		3.6 - 5 V (unregulated)
Average Current Draw @ 10 Hz sample rate[1]		18 mA
Peak Current Draw[1]	During external power up	120 mA pk, 75 mA over 2 ms
	During logical power up/down	110 mA pk, 85 mA over 1 ms, 60 mA over 2 ms
Sleep Mode Current Draw		0.25 mA

Footnote:

1. Tested at 3.6 V.

Table 3-4: Environmental Requirements

Parameter	Value
Operating Temperature[1]	-40C to +85C
Storage Temperature	-40C to +85C

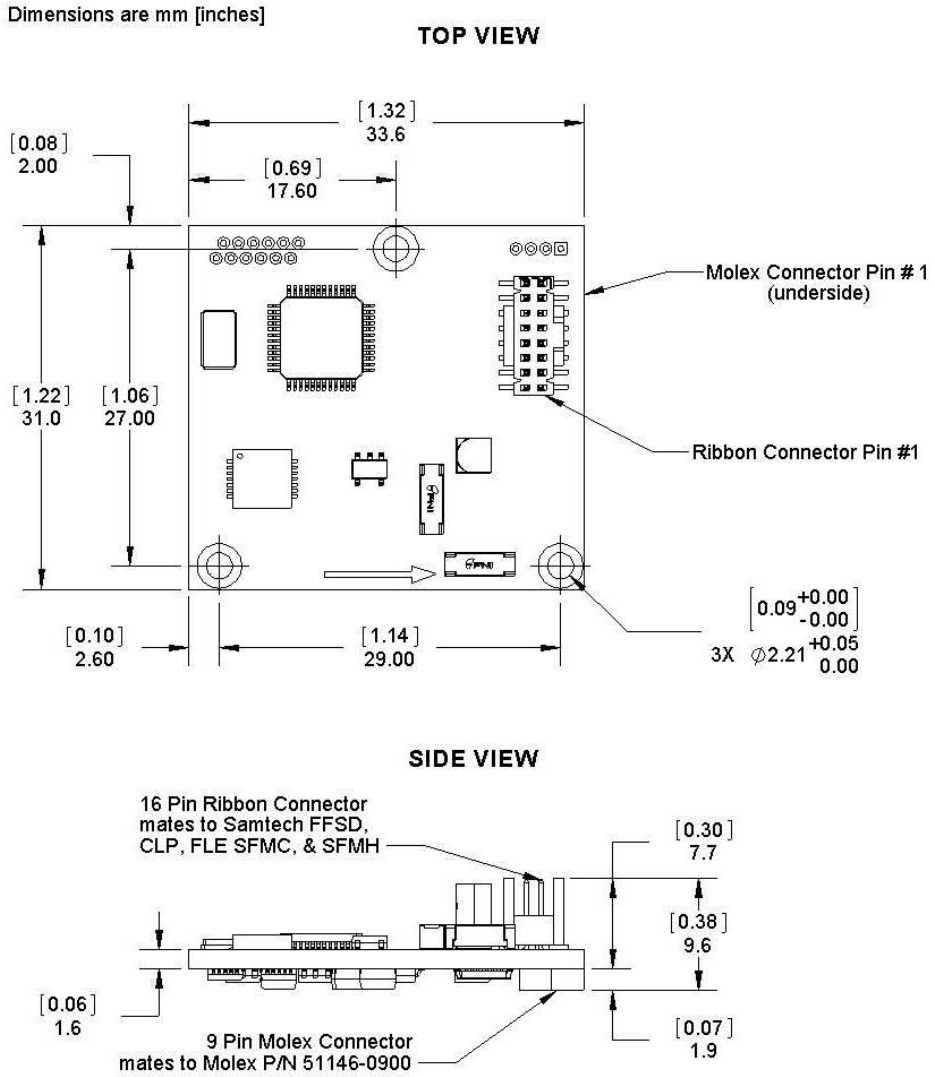
Footnote:

1. To meet performance specifications, recalibration may be necessary as temperature varies.

Table 3-5: Mechanical Characteristics

Parameter	Value
Dimensions (*l* x *w* x *h*)	3.3 x 3.1 x 1.3 cm (see Figure 3-1)
Weight	5 gm
Mounting Options	Screw mount / standoff, horizontal or vertical
Connector	16 pin ribbon or 9 pin Molex (Same functionality: use only one.)

3.3 Mechanical Drawing



Note: *The default orientation for the Prime is for the silk-screened arrow to point in the "forward" direction.*

Figure 3-1: Prime Mechanical Drawing

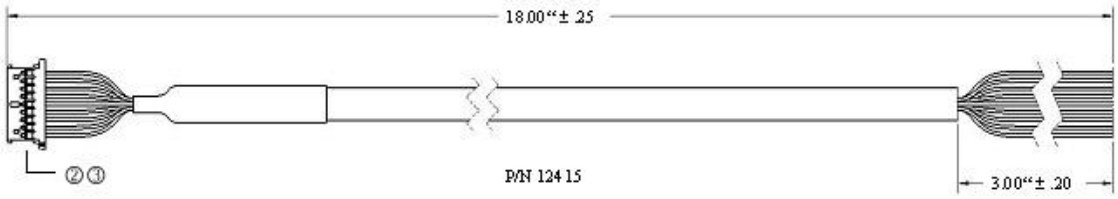


Figure 3-2: PNI Pigtailed Cable Drawing

4 Set-Up

This section describes how to configure the Prime in your host system. To install the Prime into your system, follow these steps:

- Make electrical connections to the Prime.
- Evaluate the Prime using the included StudioPrime program or terminal emulation software, such as Tera Term or RealTerm, to ensure the compass generally works correctly.
- Choose a mounting location in the host system.
- Mechanically mount the Prime in the host system.
- Perform a field calibration.

4.1 Electrical Connections

The Prime incorporates both a 16 pin ribbon connector (topside of PCB) and a 9 pin Molex connector (bottom side of PCB) for connecting the unit to the user's system. The user should decide which connector they want to use, and only use this connector. The Prime will not function properly if commands are sent on both connectors. The pin-out for both connectors is given below in Table 4-1. Pin #1 for both connectors is indicated in Figure 3-1.

Table 4-1: Prime Pin Descriptions

Pin #	16 pin Ribbon Connector	9 pin Molex Connector	PNI Cable Wire Color
1	NC	GND	Black
2	NC	NC	Gray
3	GND	GND	Green
4	~CS	NC	Orange
5	GND	NC	Violet
6	NC	NC	Brown
7	GND	RS232 TxD	Yellow
8	NC	RS232 RxD	Blue
9	GND	+5 VDC	Red
10	NC		
11	GND		
12	RS232 RxD		
13	GND		
14	RS232 TxD		
15	GND		
16	+5 VDC		

The Prime Interface Kit includes the PNI 45 cm (18") custom pigtailed cable (see Figure 3-2). One end of the cable mates with the Prime's 9 pin Molex connector while the other end is not connectorized and has 9 wires accessible. These wires are intended to mate with the user's system. The cable's wires are color coded as indicated in Table 4-1.

The Prime Evaluation Kit includes the same PNI pigtailed cable as provided in the Interface Kit, plus a 1.8 m (6') custom dual-connectorized cable. This cable incorporates the Molex 51146-0900 connector on one end that mates to the Prime and a 9-pin sub-D connector on the other end to mate with a computer's serial port. This cable primarily is intended for basic evaluation of the Prime with a computer.

4.2 Mechanical Installation

The Prime's wide dynamic range and sophisticated calibration algorithms allow it to operate in many environments. For optimal performance however, you should mount the Prime with the following considerations in mind:

4.2.1 Operate within sensors' useable regime

The Prime can be field calibrated to correct for large static magnetic fields created by the host system. However, each axis of the Prime has a maximum useable range of ±100 µT. If the total field exceeds this value for any axis the Prime may not give accurate heading information. When mounting the Prime, consider the effect of any sources of magnetic fields in the host environment that, when added to the earth's field, may take the sensors out of their linear regime. For example, distortions can be caused by large masses of ferrous metals such as transformers and vehicle chassis, large electric currents, permanent magnets contained within electric motors, and so on.

4.2.2 Locate away from changing magnetic fields

It is not possible to calibrate for changing magnetic anomalies. Thus, for greatest accuracy, keep the Prime away from sources of local magnetic distortion that will change with time, such as electrical equipment that will be turned on and off, or ferrous bodies that will move. Make sure the Prime is not mounted close to cargo or payload areas that may be loaded with large sources of local magnetic fields.

4.2.3 Mount in a physically stable location

Choose a location that is isolated from excessive shock, oscillation, and vibration. The Prime works best when stationary. Any non-gravitational acceleration results in a distorted reading of Earth's gravitational vector, which affects the heading measurement.

4.2.4 Location-verification testing

Location-verification testing should be performed at an early stage of development to understand and accommodate the magnetic distortion contributors in a host system. The data logger in StudioPrime (see Section 5.6) can be used to perform the following tests.

Determine the distance range of field distortion

Place the compass in a fixed position, then move or energize suspect components while observing the output to determine when they are an influence.

Determine if the maximum field is within the linear range of the compass

With the compass mounted, rotate and tilt the system in as many positions as possible. While doing so, monitor the magnetic sensor outputs, observing if the maximum useable range is exceeded.

4.3 Mounting

Refer to Figure 3-1 for dimensions, mounting holes, and reference frame orientation.

The Prime is pre-loaded with calibration coefficients so it nominally indicates north per the arrow on the PCB, assuming a standard orientation (STD 0°) and minimal local magnetic distortions. It must be aligned within the host system with respect to the mounting holes. Ensure any stand-offs or screws used to mount the module are non-magnetic.

The Prime can be mounted in 24 different orientations, as called out in Table 7-4 and depicted in Figure 7-2. However, StudioPrime only supports 6 mounting configurations, as depicted in Figure 5-1. All reference points are based on the white silk-screened arrow on the top side of the board. The orientation should be programmed in the Prime using the kSetConfig command and the kMountRef setting, as described in Section 7.3.6.

4.4 Pitch and Roll Convention

As shown in Figure 4-1, roll is defined as the angle rotated around the long axis of the module while pitch is rotation around shorter axis of the module. Positive pitch is when the front edge of the board is rotated upward and positive roll is when the right edge of the board is rotated downward. These two rotations are independent of each other.

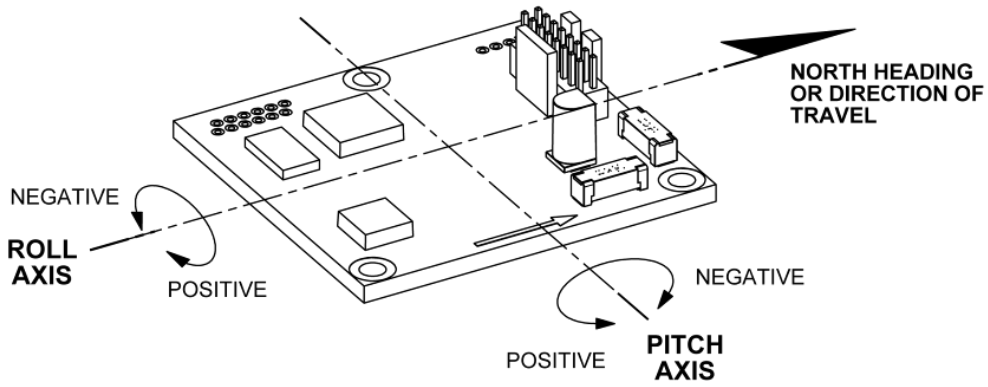


Figure 4-1: Positive & Negative Roll and Pitch Definition

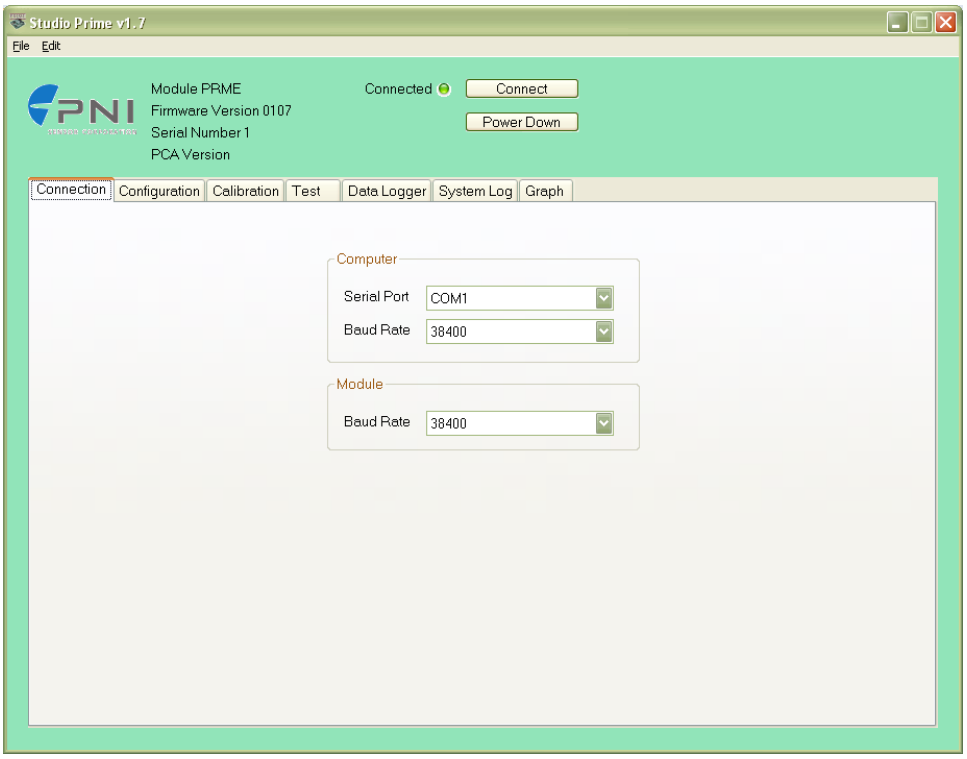
5 Operation with StudioPrime

The StudioPrime evaluation software communicates with the Prime through the RS232 serial port of your computer. It puts an easy-to-use, graphical-user interface (GUI) onto the binary command language used by the Prime. Instead of manually issuing command codes, the user can use buttons, check boxes, and dialog boxes to control the Prime and obtain data. It reads the binary responses of the Prime output and formats this into labeled and easy-to-read data fields. StudioPrime also includes the ability to log and save the outputs of the Prime to a file. This allows you to begin understanding the capabilities of the Prime while using StudioPrime's friendly interface. Anything that can be performed using StudioPrime can also be performed using the RS232 interface and associated protocol.

5.1 Installation onto a Windows system

StudioPrime is provided as an executable program which can be downloaded from PNI's website at www.pnicorp.com. It runs on Windows XP, Vista, and Windows 7 operating systems. To get started, download the StudioPrime.msi file onto your computer, then open the file and step through the Setup Wizard.

5.2 Connection Tab



5.2.1 Initial connection

When initially launching StudioPrime:

- If using the PNI dual-connecterized cable, ensure the batteries are well-charged and the cable is securely attached to the Prime and the PC's serial port.
- Select the serial port the module is plugged into, which is generally COM 1.
- Select 38400 as the baud rate.
- Click the <Connect> button if the connection is not automatically made.

Once a connection is made the "Connected" light will turn green and the module's firmware version and serial number will be displayed in the upper left.

5.2.2 Changing baud rate

To change the baud rate:

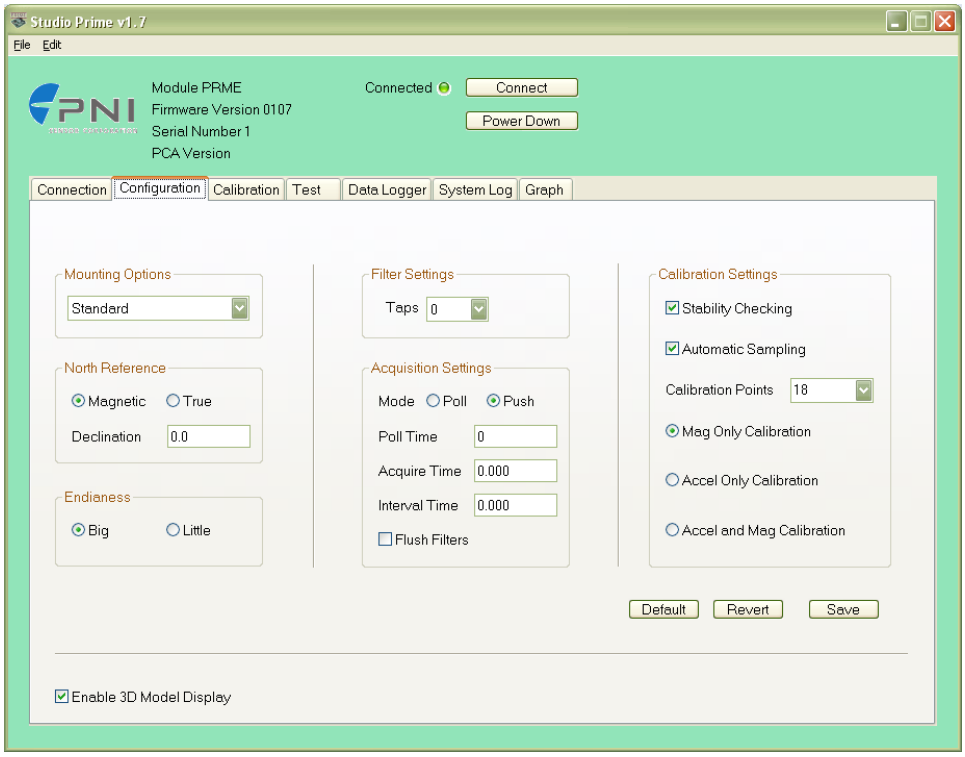
- In the Module box, select the new desired baud rate for the module.
- Click the <Power Down> button. The button will change to read <Power Up>.
- In the Computer box, select same baud rate for the computer.
- Click on the <Power Up> button. The button will revert back to <Power Down>.

Note: *While it is possible to select a baud rate of 230400, the serial port will not operate this fast.*

5.2.3 Changing modules

Once a connection has been made, StudioPrime will recall the last settings. If a different module is used, click the <Connect> button once the new module is attached. This will reestablish a connection assuming the baud rate is unchanged.

5.3 Configuration Tab



Note: *No settings will be changed in the unit until the <SAVE> button has been selected.*

5.3.1 Mounting Options

StudioPrime supports 6 mounting orientations, as detailed below in Figure 5-1. Note that PNI's binary protocol supports 18 more orientations (24 total), as shown in Figure 7-2.

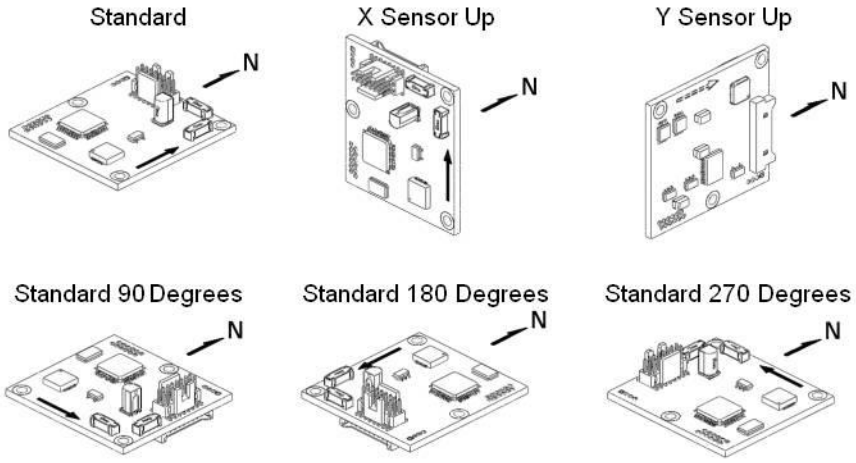


Figure 5-1: StudioPrime Mounting Orientations

5.3.2 North Reference

Magnetic

When the <Magnetic> button is selected, heading will be relative to magnetic north.

True

When the <True> button is selected, heading will be relative to geographic (true) north. In this case, the declination needs to be set in the "Declination" box. Refer to Section 6.3 for more information.

5.3.3 Endianess

Select either the <Big> or <Little> Endian button. The default is <Big>. See Section 7.2 for more information on appropriate data formats.

5.3.4 Filter Setting (Taps)

The Prime incorporates a finite impulse response (FIR) filter to effectively provide a more stable heading reading. The number of taps represents the number of measurement readings to be included in the filter. The user should select either 0, 4, 8, 16, or 32 taps, with zero taps representing no filtering. The default setting is 8 taps.

The Prime is configured to update the filter with each new measurement reading. When a new reading is added to the filter, the oldest reading in the filter is dropped. For example if 8 taps is selected, the 8 most recent readings will populate the filter.

Note: *Selecting a large number of taps will significantly slow the time for the initial heading reading, since the filter must be fully populated before the Prime will output a heading reading.*

5.3.5 Acquisition Settings

Mode

"Poll" mode should be selected when the host system will poll the Prime for data. StudioPrime allows the user to simulate this on their PC. In this case, StudioPrime requests data from the Prime module at a relatively fixed basis.

"Push" mode should be selected if the user will have the Prime output data at a relatively fixed rate to the host system. In this case the Prime module is pushing data out to StudioPrime at a relatively fixed rate.

Poll Delay

The Poll Delay is relevant when Poll Mode is selected, and is the time delay, in seconds, between the completion of StudioPrime receiving one set of sampled data and requesting the next sample set. If the time is set to "0" then StudioPrime requests

new data as soon as the previous request has been fulfilled. Note that the inverse of the Poll Delay is somewhat greater than the sample rate, since the Poll Delay does not include actual acquisition time.

Interval Delay

The Interval Delay is relevant when Push Mode is selected, and is the time delay, in seconds, between completion of the Prime module sending one set of sampled data and the start of sending the next sample set. If the time is set to 0 then the Prime will begin sending new data as soon as the previous data set has been sent. Note that the inverse of the Interval Delay is somewhat greater than the sample rate, since the Interval Delay does not include actual acquisition time.

Acquire Delay

The Acquire Delay sets the time between samples taken by the module, in seconds. This is an internal setting that is NOT tied to the time with which the module transmits data to StudioPrime or the host system. Generally speaking, the Acquire Delay is either set to 0, in which case the Prime is constantly sampling or set to equal either the Poll Delay or Interval Delay values. The advantage of running with an Acquire Delay of 0 is that the FIR filter can run with a relatively high Tap value to provide stable and timely data. The advantage of using a greater Acquire Delay is that power consumption can be reduced, assuming the Interval or Poll Delay are no less than the Acquire Delay.

Flush Filters

Selecting this box results in the FIR filters being flushed (voided out) after each measurement reading. This can be useful if a single, accurate reading is desired and there is sufficient time to re-populate the filters The default is not to flush the filters.

Note: *If "Flush Filters" is selected the rate at which the output is updated will be inversely proportional to the number of taps. For example, if 8 taps and flush filters are selected, the output will be updated at ~3.5 Hz, while it will be updated at ~1 Hz if 32 taps is selected.*

5.3.6 Calibration Settings

The Prime supports both magnetic sensor and accelerometer calibration. These calibrations can be performed independently or simultaneously. The relevant StudioPrime inputs for these calibrations are discussed below. See Section 6.2 for information on how to perform a calibration.

Stability Checking

By default when calibrating the module, a measurement must be stable for 3 consecutive readings prior to saving the sample for use in the calibration. This is why the module must be held steady between points during the field calibration. This stability helps to ensure a proper heading and allow for higher accuracy, but it also

takes more time. If this checkbox is deselected, then the module will NOT wait for a stable reading and will immediately take a reading once the minimum change-between-points threshold has been met.

Automatic Sampling

When this checkbox is selected, the module automatically takes a calibration point once the minimum change-between-points requirement and the stability check requirement (if selected) have been satisfied. If the user wants direct control over when a calibration point is taken, then Automatic Sampling should be deselected. In this case, the <Take Sample> button on the Calibration tab will be active. Clicking the <Take Sample> button indicates to the module to take a sample once the minimum requirements are met.

Calibration Points

The user can select the number of points to take during a calibration. A minimum of 12 sample points are needed for a successful magnetic sensor calibration, while 18 samples are recommended for either accelerometer-only calibration or simultaneous magnetic sensor and accelerometer calibration.

Mag Only Calibration

Select when only magnetic sensor calibration will be performed.

Accel Only Calibration

Select when only an accelerometer calibration will be performed.

Accel and Mag Calibration

Select when magnetic sensor and accelerometer calibrations will be performed simultaneously.

5.3.7 Default

Clicking this button reverts the StudioPrime program to the factory default settings.

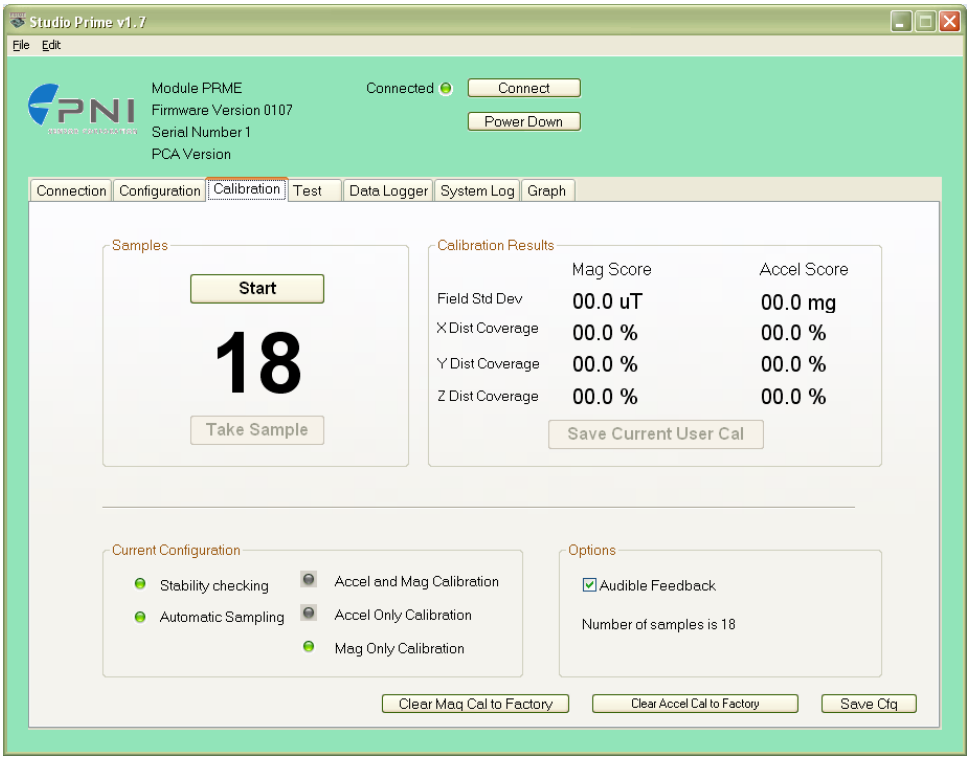
5.3.8 Revert

This button will have the StudioPrime program read the settings from the unit and display them on the screen.

5.3.9 Enable 3D Model

StudioPrime's Test tab includes a live-action 3-D rendering of a helicopter. Some computer systems may not have the graphics capability to render the 3D Model, and for this reason it may be desirable to turn off this feature.

5.4 Calibration Tab



5.4.1 Samples

Before proceeding, refer to Section 6.2 for the recommended calibration procedure corresponding to the calibration method selected on the Configuration tab.

Clicking the <Start> button begins the calibration process.

If "Automatic Sampling" is not checked on the Configuration tab, it is necessary to click the <Take Sample> button to take a calibration sample point. This should be repeated until the total number of samples (as set on the Configuration tab) is taken, changing the orientation of the module between samples as discussed in Section 6.2.

If "Automatic Sampling" is checked, the module will need to be held steady for a short time and then a sample automatically will be taken. Once the window indicates the next number, the module's orientation should be changed and held steady for the next sample. Once the pre-set number of samples has been taken (as set on the Configuration tab) the calibration is complete.

5.4.2 Calibration Results

Once calibration is complete the "Calibration Results" window will indicate the quality of the calibration. This applies to both magnetic sensor and accelerometer calibration. The X, Y, and Z "Dist Coverage" values show a percentage of each vector that has been covered during the calibration. The only way to get a Z value greater than 50% would be to take some points with the module upside-down. The "Field Std Dev" values for Mag Score and Accel Score indicate the overall quality of the calibration. The target value for the "Field Std Dev" Mag Score is ~0.1 and the Accel Score value should be ~2, where a lower score is better. Note that acceptable compass performance can be obtained with somewhat higher scores, and the obtainable score will be a function of the host system, the specific Prime module, and the execution of the calibration procedure. See Section 6.2 for additional information.

If a Mag Only Calibration is performed, the Mag Score will reflect the new magnetic sensor calibration, while the Accel Score will be grayed out and represent the last saved Accel Score values. Similarly, if an Accel Only Calibration is performed the Accel Score will reflect the new accelerometer calibration and the Mag Score will be grayed out and reflect the last saved Mag Score values.

If the <Stop> button is clicked during a calibration process prior to taking the minimum required samples, this will abort the calibration. (The <Start> button turns into the <Stop> button once the calibration process is started.) The scores associated with the calibration method that was aborted will be "-1", while scores unassociated with the aborted calibration method will be grayed out and represent their last saved values.

If the calibration is acceptable, click the <Save Current User Cal> button to save the calibration. If this button is not selected then the unit will need to be recalibrated after it is turned off.

Note: *The values in µT or mg refer to the quality of the calibration and NOT the accuracy of the heading. It is possible to have a good calibration but poor heading accuracy if the local magnetic field changes after calibration.*

5.4.3 Current Configuration

These indicators mimic the pertinent selections made on the Configuration tab.

5.4.4 Options

Audible Feedback:

If selected, StudioPrime gives an audible signal when a calibration point is taken. Note that an audible signal also will occur when the <Start> button is clicked, but no data will be taken.

5.4.5 Clear

Clear Mag Cal to Factory:

This button clears the user's calibration of the magnetic sensors. Once selected, the module reverts to its factory default values. To save this action in non-volatile memory, click the <Save Cfg > button. It is not necessary to clear the current calibration in order to perform a new calibration.

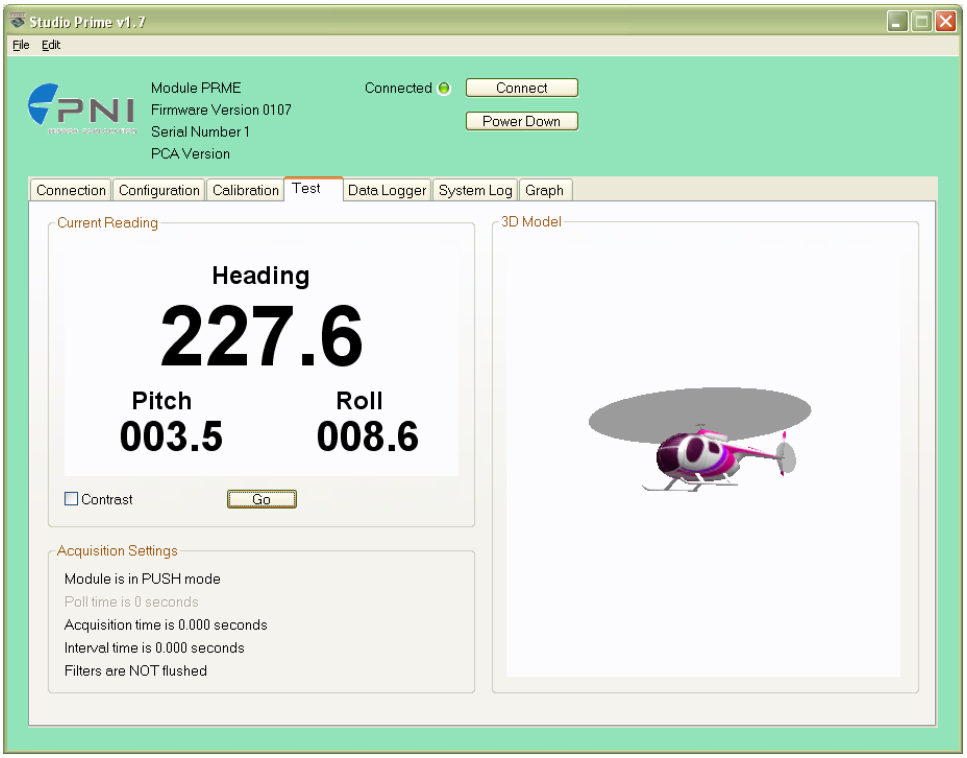
Clear Accel Cal to Factory:

This button clears the user's calibration of the accelerometers. Once selected, the module reverts to its factory default values. To save this action in non-volatile memory, click the <Save Cfg> button. It is not necessary to clear the current calibration in order to perform a new calibration.

Save Cfg:

This button is only used in conjunction with either of the two buttons detailed above.

5.5 Test Tab



5.5.1 Current Reading

Once the <GO> button is selected the unit will begin outputting heading, pitch and roll information. The <GO> button then turns to a <Stop> button. Selecting the <Stop> button or changing tabs will halt the output.

Contrast

Selecting this box sets the "Current Readings" window to have yellow lettering on a black background, rather than black lettering on a white background.

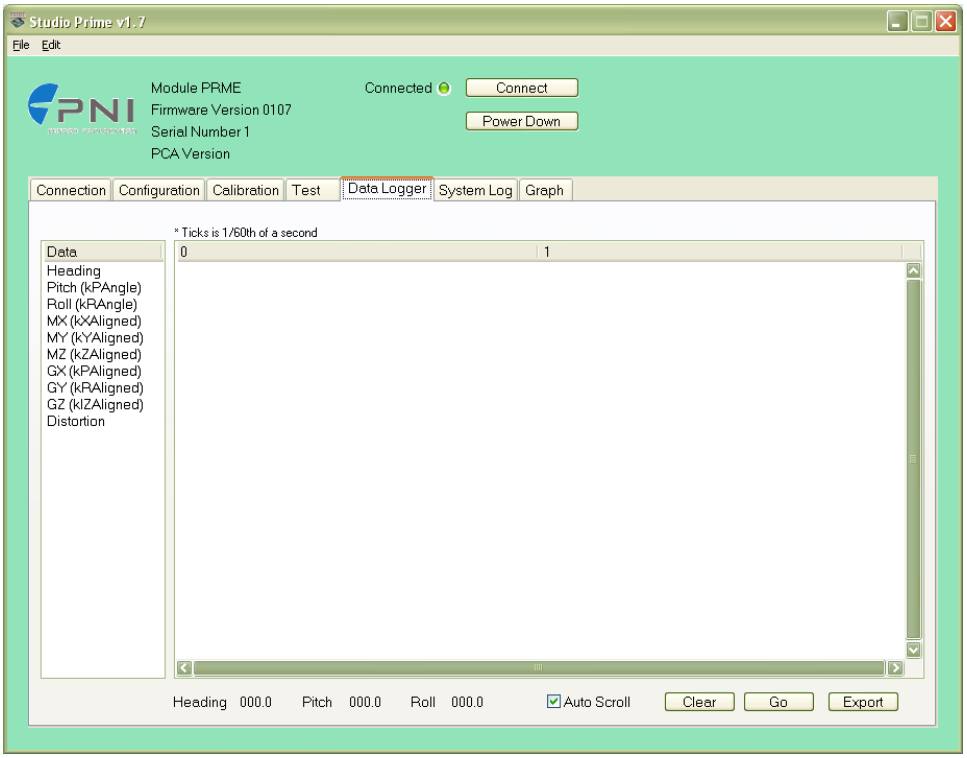
5.5.2 Acquisition Settings

These indicators mimic the pertinent selections made on the Configuration tab.

5.5.3 3D Model

The helicopter will follow the movement of the attached module and give a visual representation of the module's orientation, assuming the "Enable 3D Model Display" box is selected on the Configuration tab.

5.6 Data Logger Tab

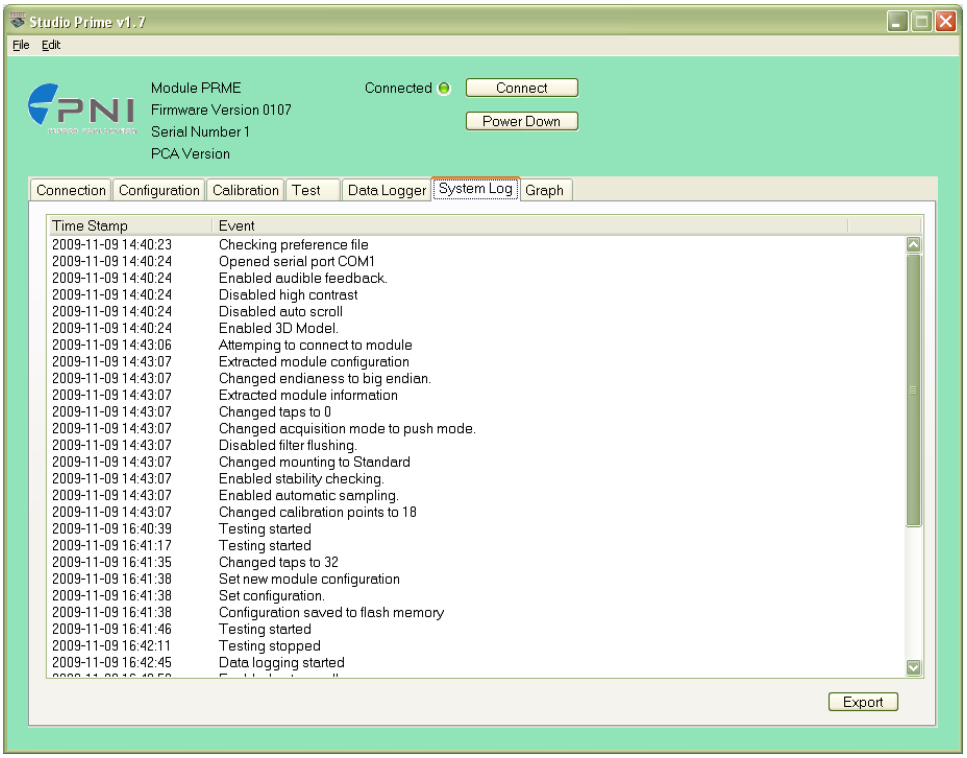


StudioPrime can capture measurement data and then export it to a text file. To acquire data and export it, follow the procedure below:

- Select the data to log in the "Data" window. Use Shift-Ctrl-Click and Ctrl-Click to select multiple items.
- Click on the <GO> button to start logging. The <Go> button changes to a <Stop> button after data logging begins.
- Click the <STOP> button to stop logging.
- Click the <Export> button to save the data to a text file.
- Click the <Clear> button to clear the data from the window.

Note: *The data logger use ticks for time reference. A tick is 1/60 second.*

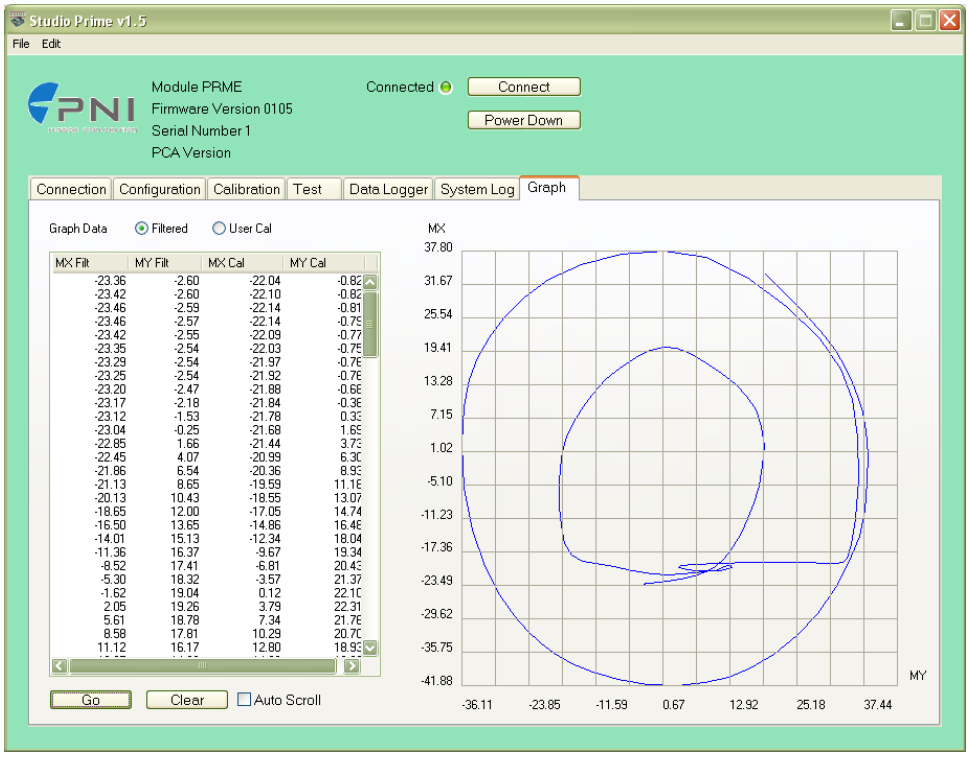
5.7 System Log Tab



The System Log tab shows all communication between StudioPrime and the Prime module since StudioPrime was opened. Closing StudioPrime will erase the system log.

Select the <Export> button, at the bottom right of the screen, to save the system log to a text file.

5.8 Graph Tab



The graph provides a 2-axis (X,Y) plot of the measured field strength. The graph can be used to visually see hard and soft iron effects within the environment measured by the Prime module as well as corrected output after a field calibration has been performed. (The screen shot shown shows the MX and MY readings as the module was held horizontally and rotated through 360º in the horizontal plane, then held in vertical orientation and rotated 360º in the vertical plane.)

6 Field Calibration

Sources of magnetic distortion that are positioned near the Prime in the user's system will distort Earth's local magnetic field and need to be compensated for before implementing the Prime in the host system. Examples of such sources include ferrous metals and alloys (ex. iron, nickel, magnetized or non-stainless steel, etc.), batteries, permanent magnets, wires, and electric motors. Compensation is accomplished by calibrating the Prime's magnetic sensors while mounted in the user's system. In the user's system it is expected the sources of magnetic distortion will remain fixed relative to the Prime's position. When performing a field calibration, the Prime identifies the local sources of magnetic distortion and subtracts these effects from the overall reading to provide an accurate compass heading.

Additionally, the Prime's MEMS accelerometers gradually may change over time and it may be desirable to recalibrate the accelerometers from time-to-time. The accelerometer calibration procedure corrects for changes in accelerometer gain and offset. Unlike the magnetic sensors, the accelerometers may be calibrated outside the host system. Accelerometer calibration is more sensitive to noise or hand jitter than magnetic sensor calibration, especially for subsequent use at high tilt angles. Because of this, a stabilized fixture is recommended for accelerometer calibration, although resting the unit against a stable surface often is sufficient. Alternatively, the Prime can be returned to PNI for recalibration. Since change in the accelerometers is gradual and relatively small, accelerometer calibration is optional.

Key Points

- Magnetic calibration requires incorporating the module in the user's system such that the magnetic components of the user's system can be compensated for. Full sphere coverage during calibration is ideal, but not necessary to obtain a good calibration.
- Even though heading is only specified to a tilt angle of ≤45°, calibrating at >45° of tilt generally will improve accuracy when operating near the specification limit of 45°.
- Accelerometer calibration requires that the module essentially be rotated through a full sphere of coverage. However, it does not require that the module be incorporated into the user's system to perform the calibration.
- Magnetic and accelerometer calibration can be performed simultaneously. However, it generally is easier to perform them separately since the requirements of each calibration are significantly different. (Magnetic calibration requires the module be incorporated in the user's system, while accelerometer calibration requires full sphere coverage.)
- Pay attention to the calibration scores. See Section 7.3.18 for the score meanings.

6.1 Magnetic Calibration Overview

The main objective of a magnetic field calibration is to compensate for distortions to the magnetic field caused by the host system. To that end, the Prime needs to be mounted within the host system and the entire system must be moved as a single unit during calibration.

6.1.1 Hard-Iron and Soft-Iron Effects

Hard-iron distortions are caused by permanent magnets and magnetized steel or iron objects within close proximity to the sensors. This type of distortion remains constant and in a fixed location relative to the sensors for all heading orientations. Hard-iron distortions add a constant magnitude field component along each axis of sensor output.

Soft-iron distortions are the result of interactions between the Earth's magnetic field and any magnetically "soft" material within close proximity to the sensors. In technical terms, soft materials have a high permeability. The permeability of a given material is a measure of how well it serves as a path for magnetic lines of force, relative to air, which has an assigned permeability of one. Unlike hard-iron distortion, soft-iron distortion changes as the host system's orientation changes, making it more difficult to compensate.

The Prime features both hard and soft-iron correction. For more information, see PNI's website for the white paper "Local Magnetic Distortion Effects on 3-Axis Compassing".

6.1.2 Magnetic Calibration Limitations

The Prime measures the total magnetic field within its vicinity, and this is a combination of Earth's magnetic field and local magnetic sources. The Prime can compensate for local static magnetic sources. However, a magnetic source which is not static (such as a motor which turns on/off) can create errors, and it is not possible to compensate for such a dynamic nature. In such cases, moving the Prime away from dynamic magnetic fields is recommended, or taking measurements only when the state of the magnetic field is known (ex. only take measurements when a nearby motor is turned off).

6.2 Field Calibration Procedures

The following sub-sections provide instructions for obtaining calibration points when performing either magnetic sensor or accelerometer field calibrations.

Before proceeding with a calibration, the Prime should be properly installed in the host system, as discussed in Section 4, and the software should be properly configured with respect to the mounting orientation, Endianness, magnetic vs true north, etc.

The calibration procedures can be executed using StudioPrime or the Prime's binary protocol. Sections 5.3 and 5.4 outline how to configure and perform a calibration in StudioPrime. The steps below provide an example sequence of commands to perform a calibration using the PNI binary protocol. (Refer to Section 7 for information on how to implement the binary protocol commands.)

- Using the kSetParam command, set the number of filter taps. (8 is typical.)
- Using the kSetConfig command, set kUserCalAutoSampling. FALSE allows for more control over the process, while TRUE may simplify the procedure.
- Using the kSetConfig command again, set kUserCalNumPoints to the appropriate number of calibration points. The recommended number of calibration points is at least 12 for Magnetic Sensor Only Calibration and at least 18 for Accel Only Calibration or Mag and Accel Calibration.
- Initiate a calibration using the kStartCal command. Note that this command requires indentifying the type of calibration procedure (i.e. Magnetic Sensor Only, etc.).
- Follow the appropriate calibration procedure discussed in Sections 6.2.1 to 6.2.3. If kUserCalAutoSampling is FALSE, then send a kTakeUserCalSample command when ready to take a calibration point. If kUserCalAutoSampling is TRUE, then look for kUserCalSampCount to confirm when a calibration point has been taken.
- When the final calibration point is taken, the module will present the calibration score using kUserCalScore.
- If the calibration is acceptable (see Section 6.2.4), save the calibration coefficients using kSave.

6.2.1 Magnetic Calibration Procedure

The following procedure is recommended for magnetic sensor calibration. Additional sample points with the module flipped upside down are also desirable if possible.

Move the module to the following positions, noting that these are not absolute heading directs, but rather relative headings referenced to your initial heading sample. (i.e. yaw is relative to the starting orientation, and does not need to be North.)

Note: *Once a calibration procedure is started with auto sampling enabled, pausing between desired calibration points can cause unintentional points to be taken.*

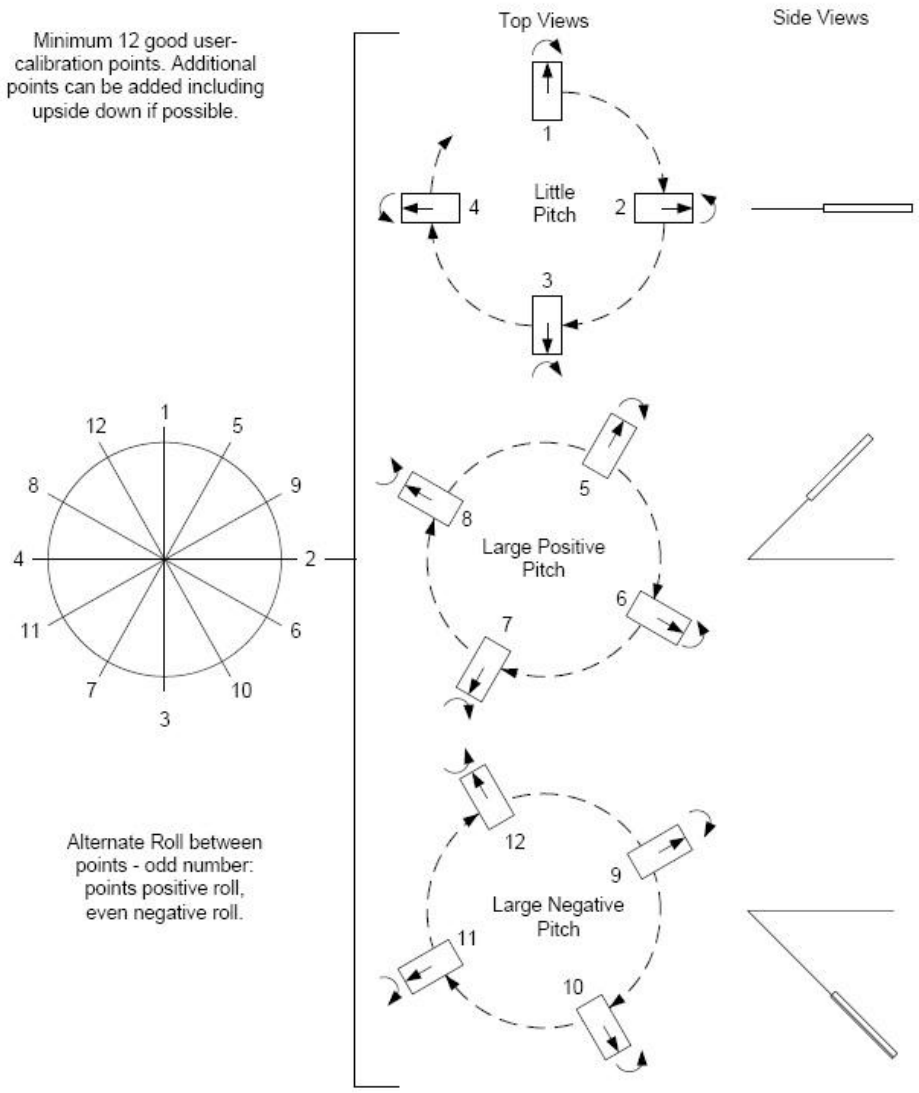


Figure 6-1: Magnetic Calibration Pattern

Table 6-1: Magnetic Calibration Pattern

Sample #		Yaw	Pitch	Roll
First Circle				
	1	0°	±5°	30° to 40°
	2	90°	±5°	-30° to -40°
	3	180°	±5°	30° to 40°
	4	270°	±5°	-30° to -40°
Second Circle				
	5	30°	> +45°	30° to 40°
	6	120°	> +45°	-30° to -40°
	7	210°	> +45°	30° to 40°
	8	300°	> +45°	-30° to -40°
Third Circle				
	9	60°	< -45°	30° to 40°
	10	150°	< -45°	-30° to -40°
	11	240°	< -45°	30° to 40°
	12	330°	< -45°	-30° to -40°

6.2.2 Accelerometer Calibration Procedure

The requirements for a good accelerometer calibration differ from the requirements for a good magnetic sensor calibration. For example, a level yaw sweep, no matter how many points are acquired, is effectively only 1 accelerometer calibration point. PNI recommends 18-32 calibration points for Accelerometer Only Calibration.

Figure 6-2 shows the two basic starting positions for Accelerometer Only Calibration. Calibration can occur within the user's system or with the module alone. It is not necessary to place the Prime on a hard surface as shown, but it must be held very still during calibration, and holding it against a hard surface is a way to help ensure this.

- Starting with the Prime as shown on the left in Figure 6-2, rotate the module such that it sits on each of its 6 faces. Take a calibration point on each face.
- Starting with the Prime as shown on the right, take a calibration point with it being vertical (0°). Tilt the module back 45° and take another calibration point (+45°), then tilt the module forward 45° and take another calibration point (–45°). Repeat this 3-point process by holding the module on each of its 4 corners.

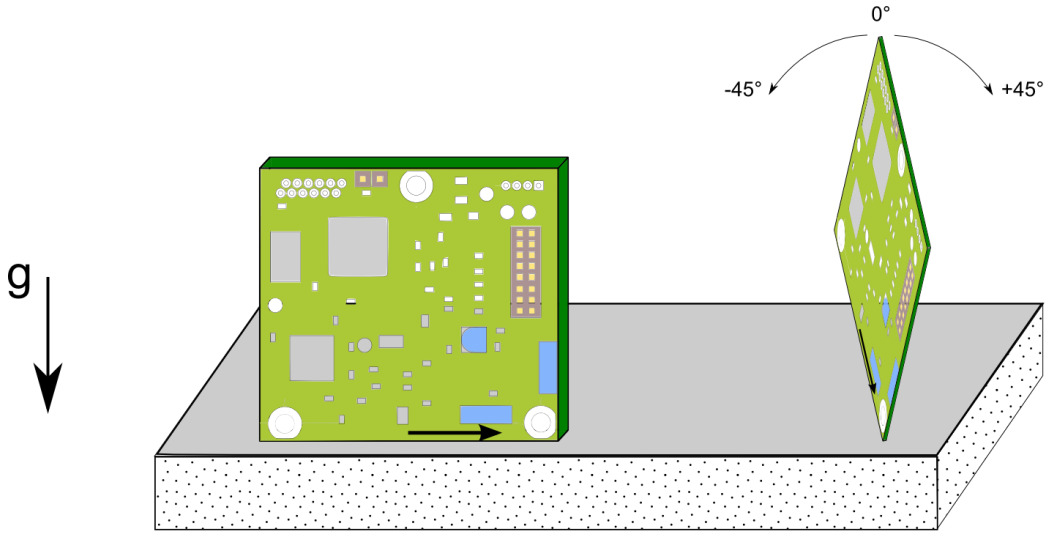


Figure 6-2: Accelerometer Calibration Starting Positions

Table 6-2: Accelerometer Calibration Pattern

Sample #	Yaw	Pitch	Roll
Sides			
1	0°	0°	90°
2	0°	90°	90°
3	180°	0°	-90°
4	0°	-90°	90°
5	0°	0°	0°
6	0°	0°	180°
First Corner			
7	0°	±5°	10° to 20°
8	90°	±5°	-10° to -20°
9	180°	±5°	10° to 20°
Second Corner			
10	270°	±5°	-10° to -20°
11	30°	> +45°	10° to 20°
12	120°	> +45°	-10° to -20°
Third Corner			
13	210°	> +45°	10° to 20°
14	300°	> +45°	-10° to -20°
15	60°	< -45°	10° to 20°
Fourth Corner			
16	150°	< -45°	-10° to -20°
17	240°	< -45°	10° to 20°
18	330°	< -45°	-10° to -20°

6.2.3 Simultaneous Mag and Accel Calibration

The Prime allows for a simultaneous magnetic sensor and accelerometer field calibration. This requires a good 18 point calibration pattern, stable measurements, and installation in the user's system. The Accelerometer Calibration Pattern discussed in Section 6.2.2 works well for a simultaneous calibration. Optimal performance is obtained when all rotations of the cube are performed towards magnetic north to achieve the widest possible magnetic field distribution.

Note that a Mag and Accel Calibration only makes sense if all the host system's magnetic distortions (steel structures or batteries, for instance) are present and fixed relative to the Prime when calibrating. If Accelerometer Only Calibration is performed, the user's system distortions are not relevant, which allows for the Prime to be removed from the system in order to perform the Accelerometer Only Calibration.

6.2.4 Calibration Scores

Once calibration is complete, the Prime provides calibration scores indicating the quality of the calibration. These are found in "Calibration Results" on the Calibration tab in StudioPrime or in the kUserCalScore payload. These apply to both magnetic sensor and accelerometer calibration. For magnetic sensor calibration, a score of ≥85% is desirable for the X and Y vectors. The only way to get a Z value greater than 50% is to take calibration points with the unit upside-down as well as right-side-up. For accelerometer calibration, a score of ≥95% is desirable for the X and Y vectors, and ≥90% for Z vector. The value for Mag Score should be ≤0.1 and the value for the Accel Score should be ≤2.

6.3 Declination

Declination is the angle between true north and magnetic north, where a positive value means true north is east of magnetic north. As such, declination depends on geographic location. It also changes very slowly over time. The Prime's default is to output magnetic heading. To output true heading, use the kSetConfig command to set the kTrueNorth parameter to TRUE and to store the correct declination angle, kDeclination. (See Section 7.3.6.)

For the greatest accuracy, PNI recommends checking the National Geophysical Data Center website (below) to get the declination angle based on your latitude and longitude:

http://www.ngdc.noaa.gov/geomagmodels/Declination.jsp

7 Operation with PNI Binary Protocol

The Prime utilizes a binary protocol that is transmitted over an RS232 interface. The configuration parameters should be set as follows:

Table 7-1: Configuration

Parameter	Value
Number of Data Bits	8
Start Bits	1
Stop Bits	1
Parity	none

7.1 Datagram Structure

The data structure is shown below:

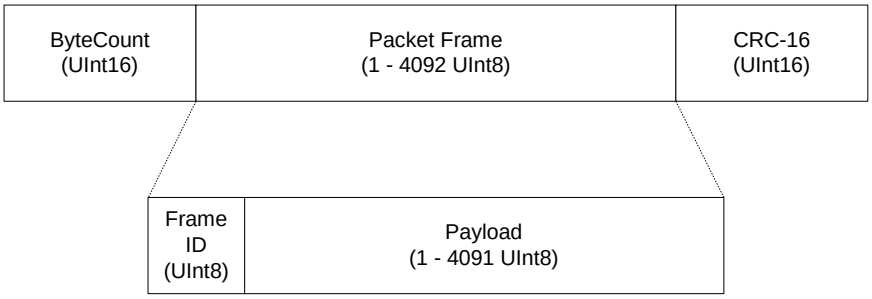


Figure 7-1: Datagram Structure

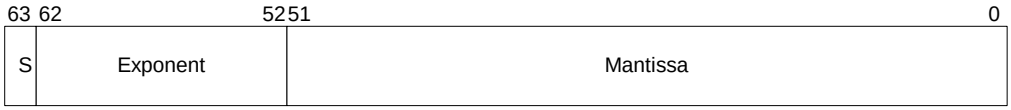
The ByteCount is the total number of bytes in the packet including the CRC-16 (checksum). CRC-16 is calculated starting from the ByteCount to the last byte of the Packet Frame (see included C function at end of document). The ByteCount and CRC-16 are always transmitted in Big Endian.

7.2 Parameter Formats

Note: *Floating-point based parameters conform to ANSI/IEEE Std 754-1985. Please refer to the Standard for more information. PNI also recommends the user refer to the compiler's instructions to understand how the compiler implements floating-point format.*

64-Bit floating point (Float64)

Below is the 64-bit float format (double precision) in Big Endian. In Little Endian the bytes are in reverse order in 4 byte groups. (e.g. Big Endian: ABCD EFGH; Little Endian: DCBA HGFE).

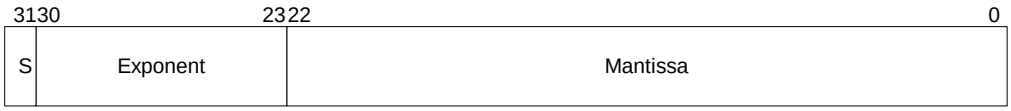


The value (v) is determined as shown below (if and only if 0 < Exponent < 2047):

$v = (-1)S * 2(Exponent-1023) * 1.Mantissa$

32-Bit floating point (Float32)

Shown below is the 32-bit float format (single precision) in Big Endian. In Little Endian format, all 4 bytes are in reverse order (LSB first).

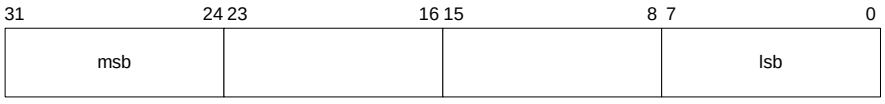


The value (v) is determined as shown below (if and only if 0 < Exponent < 255):

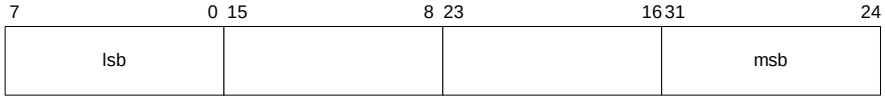
$v = (-1)S * 2(Exponent-127) * 1.Mantissa$

Signed 32-bit Integer (SInt32)

SInt32 based parameters are signed 32 bit numbers (2's compliment). Bit 31 represents the sign of the value (0=positive, 1=negative)



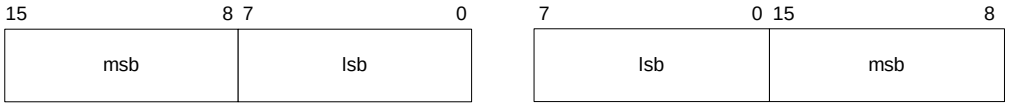
Big Endian



Little Endian

Signed 16-bit Integer (SInt16)

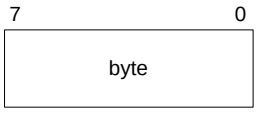
SInt16 based parameters are signed 16 bit numbers (2's compliment). Bit 15 represents the sign of the value (0=positive, 1=negative)



Big Endian Little Endian

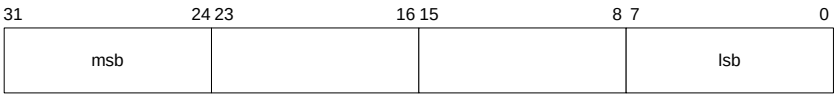
Signed 8-bit Integer (SInt8)

UInt8 based parameters are unsigned 8-bit numbers. Bit 7 represents the sign of the value (0=positive, 1=negative)

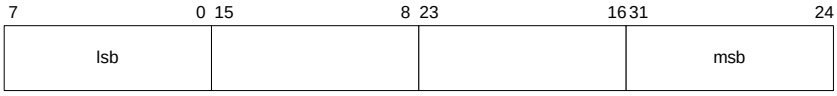


Unsigned 32-bit Integer (UInt32)

UInt32 based parameters are unsigned 32 bit numbers.



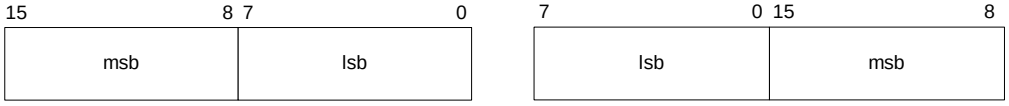
Big Endian



Little Endian

Unsigned 16-bit Integer (UInt16)

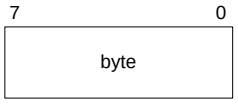
UInt16 based parameters are unsigned 16 bit numbers.



Big Endian Little Endian

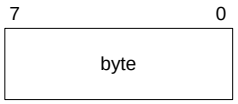
Unsigned 8-bit Integer (UInt8)

UInt8 based parameters are unsigned 8-bit numbers.



Boolean

Boolean is a 1-byte parameter that MUST have the value 0 (false) or 1 (true).



7.3 Commands & Communication Frames

The Prime's command set is given below and descriptions of each command follow.

Table 7-2: Command Set

Frame ID	Command	Description
1	kGetModInfo	Queries the modules type and firmware revision number.
2	kModInfoResp	Response to kGetModInfo
3	kSetDataComponents	Sets the data components to be output.
4	kGetData	Queries the module for data
5	kDataResp	Response to kGetData
6	kSetConfig	Sets internal configurations in the module
7	kGetConfig	Queries the module for the current internal configuration value
8	kConfigResp	Response to kGetConfig
9	kSave	Commands the module to save internal and field calibration
10	kStartCal	Commands the module to start field calibration
11	kStopCal	Commands the module to stop field calibration
12	kSetParam	Sets the FIR filter settings for the magnetic sensor & accelerometer sensors.
13	kGetParam	Queries for the FIR filter settings for the magnetic sensor & accelerometer sensors.
14	kParamResp	Contains the FIR filter settings for the magnetic sensor & accelerometer sensors.
15	kPowerDown	Used to completely power-down the module
16	kSaveDone	Response to kSave
17	kUserCalSampCount	Sent from the module after taking a calibration sample point
18	kUserCalScore	Contains the calibration score
19	kSetConfigDone	Response to kSetConfig
20	kSetParamDone	Response to kSetParam
21	kStartIntervalMode	Commands the module to output data at a fixed interval

22	kStopIntervalMode	Commands the module to stop data output at a fixed interval
23	kPowerUp	Sent after wake up from power down mode
24	kSetAcqParams	Sets the sensor acquisition parameters
25	kGetAcqParams	Queries for the sensor acquisition parameters
26	kAcqParamsDone	Response to kSetAcqParams
27	kAcqParamsResp	Response to kGetAcqParams
28	kPowerDownDone	Response to kPowerDown
29	kFactoryUserCal	Clears user magnetic sensor calibration coefficients
30	kFactorUserCalDone	Response to kFactoryUserCal
31	kTakeUserCalSample	Commands the unit to take a sample during field calibration
36	kFactoryInclCal	Clears user accelerometer calibration coefficients
37	kFactoryInclCalDone	Response to kFactoryInclCal

7.3.1 kGetModInfo (frame ID 1_d)

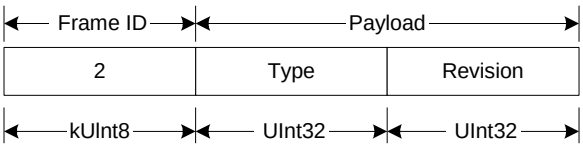
This frame queries the module's type and firmware revision number. The frame has no payload. The complete packet for the kGetModInfo command would be:

0005	01	EFD4

Where "0005" is the byte count, "01" is the kGetModInfo command, and "EFD4" is the CRC-16 checksum

7.3.2 kModInfoResp (frame ID 2_d)

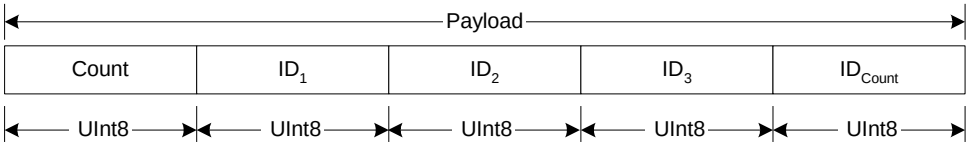
This frame is the response to kGetModInfo frame. The payload contains the module type identifier followed by the firmware revision number.



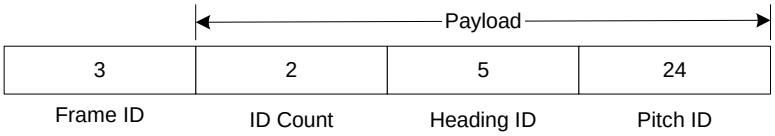
Note that the Type and Revision can be decoded from the binary format to character format using the ASCII standard. For example, the hex string "00 0D 02 54 43 4D 35 31 32 30 38 C7 87" can be decoded to read "TCM5 1208". Also, the TCM XB is referenced as Type "TCM6" since the number of Type characters is limited to 4.

7.3.3 kSetDataComponents (frame ID 3_d)

This frame sets the data components in the module's data output. This is not a query for the module's data (see kGetData). The first byte of the payload indicates the number of data components followed by the data component IDs.



Example: To query the heading and pitch, the payload should contain:



When querying for data (kGetData frame), the sequence of the data component output follows the sequence of the data component IDs as set in this frame.

Table 7-3: Component Identifiers

Component	Component ID_d	Format	Units	Range
kHeading	5	Float32	degrees	0.0˚ to 359.9˚
kDistortion	8	Boolean	True or False	False (Default) = no distortion
kCalStatus	9	Boolean	True or False	False (Default) = not calibrated
kPAligned	21	Float32	G	-1.0 to 1.0
kRAligned	22	Float32	G	-1.0 to 1.0
kIZAligned	23	Float32	G	-1.0 to 1.0
kPAngle	24	Float32	degrees	-90.0˚ to 90.0˚
kRAngle	25	Float32	degrees	-180.0˚ to 180.0˚
KXAligned	27	Float32	µT	
KYAligned	28	Float32	µT	
KZAligned	29	Float32	µT	

Component types for kSetDataComponents & kDataResp frames:

kHeading (Component ID 5_d)

Compass heading output, in degrees.

kDistortion (Component ID 8_d)

Read only flag that indicates that at least one magnetic sensor axis reading is beyond ±100 µT.

kCalStatus (Component ID 9_d)

Read only flag that indicates field calibration status. False (Default) = Not calibrated.

kPAligned, kRAligned & kIZAligned (Component IDs 21_d, 22_d, 23_d)

User calibrated Earth's gravity vector (G) component output.

kPAngle, kRAngle (Component IDs 24_d, 25_d)

The outputs provide pitch and roll angles. The pitch range is -90.0˚ to 90.0˚ and the roll range is -180.0˚ to +180.0˚. See Figure 4-1 for the pitch and roll conventions.

kXAligned, kYAligned, kZAligned (Component IDs 27_d, 28_d, 29_d)

Represent the field calibration Earth's magnetic field (M) vector components.

7.3.4 kGetData (frame ID 4_d)

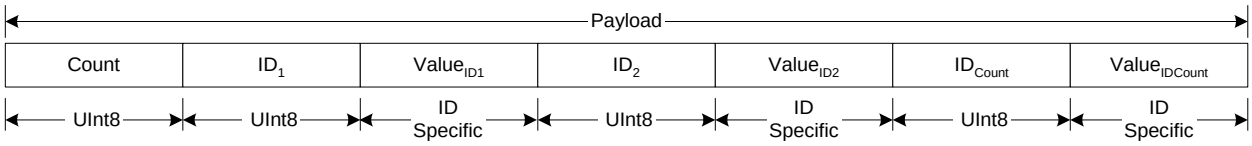
This frame queries the module for data as established in kSetDataComponents. The frame has no payload. The complete packet for the kGetModInfo command would be:

00 05	04	BF71

Where "00 05" is the byte count, "04" is the kGetData command, and "BF 71" is the CRC-16 checksum.

7.3.5 kDataResp (frame ID 5_d)

This frame is the response to kGetData frame. The first byte of the payload indicates the number of data components, followed by the component ID-value pairs. The sequence of the components IDs follows the sequence set in the kSetDataComponents frame.

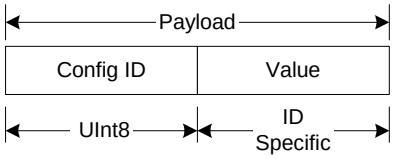


Example: If the response contains the heading and pitch, the payload would look like:

2	5	359.9	24	10.5
ID Count	Heading ID	Heading Output (Float32)	Pitch ID	Pitch Output (Float32)

7.3.6 kSetConfig (frame ID 6_d)

This frame configures the Prime. Configuration values must be set one at time.



Example: To configure the declination, the payload would look like:

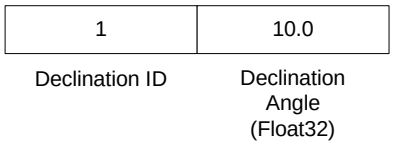


Table 7-4: Configuration Identifiers

Settings	Configuration ID	Format	Units/ Range	Default Values
kDeclination	1	Float32	-180˚ to 180˚	0˚
kTrueNorth	2	Boolean	True or False	False
kBigEndian	6	Boolean	True or False	True
kMountingRef	10	UInt8	1 = STD 0° 2 = X UP 0° 3 = Y UP 0° 4 = STD 90° 5 = STD 180° 6 = STD 270° 7 = Z DOWN 0° 8 = X UP 90° 9 = X UP 180° 10 = X UP 270° 11 = Y UP 90° 12 = Y UP 180° 13 = Y UP 270° 14 = Z DOWN 90° 15 = Z DOWN 180° 16 = Z DOWN 270° 17 = X DOWN 18 = X DOWN 90° 19 = X DOWN 180° 20 = X DOWN 270° 21 = Y DOWN 22 = Y DOWN 90° 23 = Y DOWN 180° 24 = Y DOWN 270°	1

kUserCalStableCheck	11	Boolean	True or False	True
kUserCalNumPoints	12	UInt32	12 – 32	12
kUserCalAutoSampling	13	Boolean	True or False	True
kBaudRate	14	UInt8	0 – 300 1 – 600 2 – 1200 3 – 1800 4 – 2400 5 – 3600 6 – 4800 7 – 7200 8 – 9600 9 – 14400 10 – 19200 11 – 28800 12 – 38400 13 – 57600 14 - 115200	12

Configuration parameters and settings for kSetConfig:

kDeclination (Config. ID 1_d)

This sets the declination angle to determine True North heading. Positive declination is easterly declination and negative is westerly declination. This is not applied until kTrueNorth is set to TRUE.

kTrueNorth (Config. ID 2_d)

Flag to set compass heading output to true north heading by adding the declination angle to the magnetic north heading.

kBigEndian (Config. ID 6_d)

Flag to set the Endianness of packets

kMountingRef (Config. ID 10_d)

This sets the reference orientation for the module. See Figure 7-2 for the comprehensive schematic of available orientations, with labels corresponding to those listed in Table 7-4.

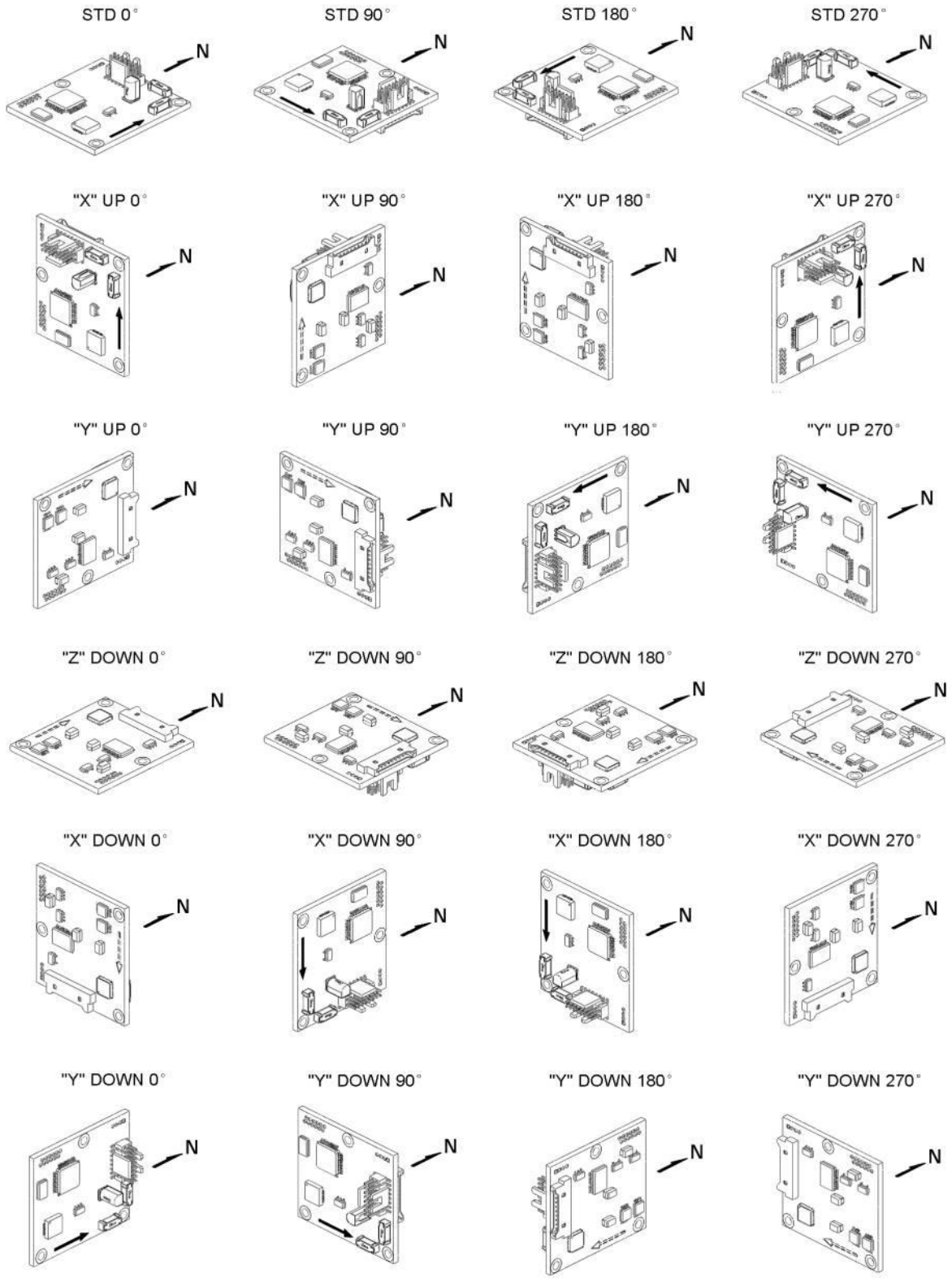


Figure 7-2: PNI Protocol Mounting Orientations

kUserCalStableCheck (Config. ID 11_d)

This flag is used during field calibration. If set to FALSE, a calibration point can be taken if the magnetic field is stable to <23 µT for each axis. If set to TRUE a calibration point can be taken if the magnetic field is stable to <5 µT for each axis.

kUserCalNumPoints (Config. ID 12_d)

The user must select the number of points to take during a field calibration. The number of points must be within "Allowable Range". Performance may suffer somewhat, but still be acceptable for the application, if the number of calibration points is less than the "Minimum Recommended" value.

Table 7-5: Number of Calibration Points

Calibration Mode	Number of Calibration Points	
	Allowable Range	Minimum Recommended
Magnetic Sensor Only	10 to 32	12
Accelerometer Only	12 to 32	18
Mag and Accel	12 to 32	18

kUserCalAutoSampling (Config. ID 13_d)

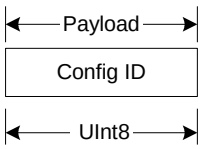
This flag is used during field calibration. If set to TRUE, the module automatically takes calibration sample points until the kUserCalNumPoints number of calibration points is taken. When auto-sampling, a new calibration point is taken when the kUserCalStableCheck condition is met and the change in the magnetic field from the prior calibration point exceeds 30 µT for at least one axis. If set to FALSE, the Prime waits for the kTakeUserCalSample frame to take a calibration point.

kBaudRate (Config. ID 14_d)

The baud rate index value sets the baud rate. A power-down power-up cycle is required when changing the baud rate.

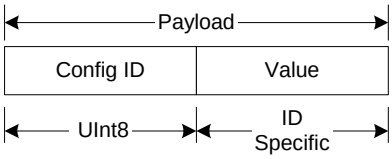
7.3.7 kGetConfig (frame ID 7_d)

This frame queries the module for the current internal configuration value. The payload contains the configuration ID requested.

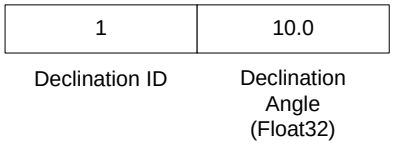


7.3.8 kConfigResp (frame ID 8_d)

This frame is the response to kGetConfig frame. The payload contains the configuration ID and value.



Example: If a request to get the set declination angle, the payload would look like:



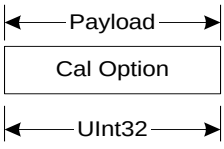
7.3.9 kSave (frame ID 9_d)

This frame commands the module to save internal configurations and field calibration to non-volatile memory. Internal configurations and field calibration is restored on power up. The frame has no payload. This is the ONLY command that causes the module to save information into non-volatile memory.

7.3.10 kStartCal (frame ID 10_d)

This frame commands the module to start field calibration with the current sensor acquisition parameters, internal configurations and FIR filter settings. See Section 6.2 for more information on the various calibration procedures.

Note: *The payload needs to be 32 bit (4 byte). If no payload is entered or if less than 4 bytes are entered, the unit will default to the previous calibration method.*



The CalOption values are given below for the 3 different calibration methods:

- Magnetic Sensor Only Calibration: 0_d or 0_h
- Accelerometer Only Calibration: 100_d or 64_h
- Mag and Accel Calibration: 110_d or $6E_h$

Below is a complete sample frame for an Accelerometer Only Calibration:

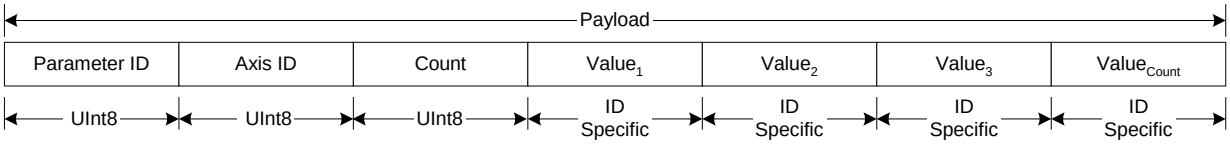
00 09 0A 00 00 00 64 5C F9

7.3.11 kStopCal (frame ID 11_d)

This command aborts the calibration process. The prior calibration results are retained.

7.3.12 kSetParam (frame ID 12_d)

The Prime incorporates a finite impulse response (FIR) filter to provide a more stable heading reading. The number of taps (or samples) represents the amount of filtering to be performed. Selecting a larger number of taps can significantly slow the time for the initial sample reading and, if "Flush Filters" is selected, the rate at which data is output.



Parameter ID should be set to 3 and the Axis ID should be set to 1. The third payload byte indicates the number of taps to use, which can be 0 (no filtering), 4, 8, 16, or 32. This is followed by the tap values (0 to 32 total Values can be in the payload), with each Value being a Float64, and suggested values given below in Table 7-6.

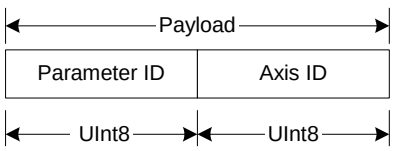
Note: *Selecting a large number of taps will significantly slow the time for the initial heading reading, since the filter must be fully populated before the Prime will output a heading reading. And, if "Flush Filters" is selected the rate at which the output is updated will be inversely proportional to the number of taps. For example, if 8 taps and flush filters are selected, the output will be updated at ~3.5 Hz, while it will be updated at ~1 Hz if 32 taps is selected.*

Table 7-6: Recommended FIR Filter Tap Values

Count	4 Tap Filter	8 Tap Filter	16 Tap Filter	32 Tap Filter
1	04.6708657655334e-2	01.9875512449729e-2	07.9724971069144e-3	01.4823725958818e-3
2	04.5329134234467e-1	06.4500864832660e-2	01.2710056429342e-2	02.0737124095482e-3
3	04.5329134234467e-1	01.6637325898141e-1	02.5971390034516e-2	03.2757326624196e-3
4	04.6708657655334e-2	02.4925036373620e-1	04.6451949792704e-2	05.3097803863757e-3
5		02.4925036373620e-1	07.1024151197772e-2	08.3414139286254e-3
6		01.6637325898141e-1	09.5354386848804e-2	01.2456836057785e-2
7		06.4500864832660e-2	01.1484431942626e-1	01.7646051430536e-2
8		01.9875512449729e-2	01.2567124916369e-1	02.3794805168613e-2
9			01.2567124916369e-1	03.0686505921968e-2
10			01.1484431942626e-1	03.8014333463472e-2
11			09.5354386848804e-2	04.5402682509802e-2
12			07.1024151197772e-2	05.2436112653103e-2
13			04.6451949792704e-2	05.8693165018301e-2
14			02.5971390034516e-2	06.3781858267530e-2
15			01.2710056429342e-2	06.7373451424187e-2
16			07.9724971069144e-3	06.9231186101853e-2
17				06.9231186101853e-2
18				06.7373451424187e-2
19				06.3781858267530e-2
20				05.8693165018301e-2
21				05.2436112653103e-2
22				04.5402682509802e-2
23				03.8014333463472e-2
24				03.0686505921968e-2
25				02.3794805168613e-2
26				01.7646051430536e-2
27				01.2456836057785e-2
28				08.3414139286254e-3
29				05.3097803863757e-3
30				03.2757326624196e-3
31				02.0737124095482e-3
32				01.4823725958818e-3

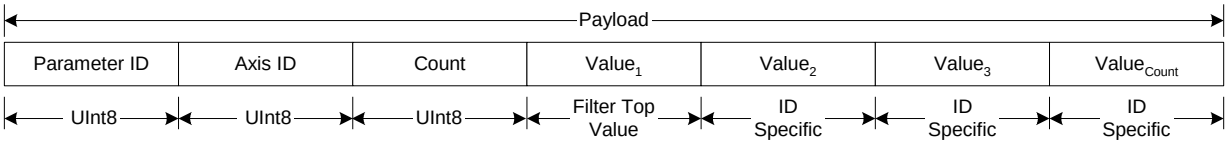
7.3.13 kGetParam (frame ID 13_d)

This frame queries the FIR filter settings for the sensors. Parameter ID should be set to 3 and the Axis ID should be set to 1.



7.3.14 kParamResp (frame ID 14_d)

This frame is the response to kGetParam and contains the current FIR filter settings. The format and values will be the same as defined by kSetParam.

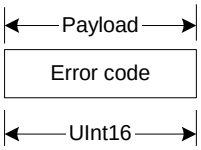


7.3.15 kPowerDown (frame ID 15_d)

This frame is used to completely power-down the module, which is referred to as putting the module in Sleep Mode. The frame has no payload. The module will power down all peripherals including the RS-232 driver but the driver chip has the feature to keep the Rx line enabled. Any character sent to the module causes it to exit Sleep Mode. It is recommended to send the byte FF_h.

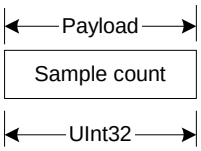
7.3.16 kSaveDone (frame ID 16_d)

This frame is the response to kSave frame. The payload contains a UInt16 error code, 0000_h indicates no error, 0001_h indicates an error when attempting to save data into non-volatile memory.



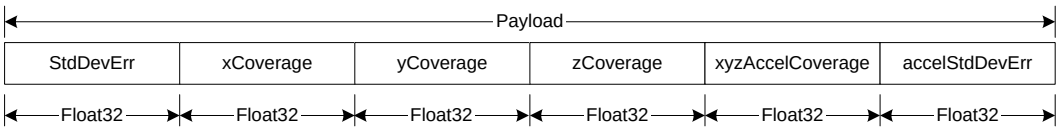
7.3.17 kUserCalSampCount (frame ID 17_d)

This frame is sent from the module after taking a calibration sample point. The payload contains the sample count with the range of 1 to 32



7.3.18 kUserCalScore (frame ID 18d)

This frame contains the calibration score, which is presented after completing or aborting a calibration. It is a series of Float32 values as shown in the payload diagram below.



Note: *If a kStopCal command is sent to abort a calibration, values that were to be updated will be set to "-1" except for xyzAccelCoverage, which will be "-101.01". Values for parameters that were not being updated will reflect the last calibration values. For example, if a Magnet Sensor Only Calibration is aborted, then StdDevErr, xCoverage, yCoverage, and zCoverage will all report -1, while the xyzAccelCoverage and accelStdDevErr will report values from the last successful accelerometer calibration.*

StdDevErr

Provides the magnetic field calibration's standard deviation error and represents the overall quality of the calibration. Smaller numbers are better, and a reasonable target value ~0.1. However, acceptable performance can be obtained with somewhat higher scores. The possible obtainable score will be a function of the host system, the specific Prime module, and the execution of the calibration procedure.

xCoverage, yCoverage, & zCoverage

Provides the percentage of the X, Y, or Z magnetic sensor axis covered by the sampling. Note that the only way to get a zCoverage value greater than 50% is to take some points with the module upside down.

xyzAccelCoverage

Provides the percentage of the accelerometer axes covered by the sampling, formatted as XXYY.ZZ. XX is the X axis coverage, YY is the Y axis coverage and ZZ is the Z axis coverage. For example xyzAccelCoverage = 8590.67 means accelerometer X coverage is 85%, Y coverage is 90% and Z coverage is 67%.

accelStdDevErr

Similar to StdDevErr, except this is for accelerometer calibration. A reasonable target accelStdDevErr value is ~2, but acceptable compass performance can be obtained with somewhat higher scores.

7.3.19 kSetConfigDone (frame ID 19_d)

This frame is the response to kSetConfig frame. The frame has no payload.

7.3.20 kSetParamDone (frame ID 20_d)

This frame is the response to kSetParam frame. The frame has no payload.

7.3.21 kStartIntervalMode (frame ID 21_d)

This frame commands the module to output data at a fixed time interval, otherwise known as Push Mode. See kSetAcqParams. The frame has no payload.

7.3.22 kStopIntervalMode (frame ID 22_d)

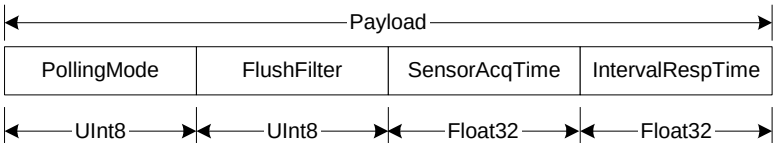
This frame commands the module to stop data output when in Push Mode. The frame has no payload.

7.3.23 kPowerUp (frame ID 23_d)

This frame is sent from the module after waking up from Sleep Mode. The frame has no payload. Since the module was previously powered down which drives the RS-232 driver TX line low (break signal), it is recommended to disregard the first byte.

7.3.24 kSetAcqParams (frame ID 24_d)

This frame sets the sensor acquisition parameters in the unit. The payload should contain the following:



PollingMode:

This flag sets whether output will be presented in Poll or Push Mode. Poll Mode is TRUE and is the default. Poll Mode should be selected when the host system will poll the TCM for data. Push Mode should be selected if the user will have the TCM output data at a relatively fixed rate to the host system. See kStartIntervalMode for starting a Push Mode command.

FlushFilter:

Setting this flag to TRUE results in the FIR filters being flushed (voided out) after each measurement reading. This can be useful if a single, accurate reading is desired and there is sufficient time to re-populate the filters. The default is FALSE.

Note: *If "Flush Filters" is selected the rate at which the output is updated will be inversely proportional to the number of taps. For example, if 8 taps and flush filters are selected, the output will be updated at ~3.5 Hz, while it will be updated at ~1 Hz if 32 taps is selected.*

SensorAcqTime:

The SensorAcqTime sets the time between samples taken by the module, in seconds. The default is 0.0 seconds, which means that the module will reacquire data immediately after the last acquisition. This is an internal setting that is NOT tied to the time with which the module transmits data to the host system. Generally speaking, the SensorAcqTime is either set to 0, in which case the TCM is constantly sampling, or set to equal the IntervalRespTime value. The advantage of running with a SensorAcqTime of 0 is that the module can run with a relatively high FIR filter tap value, thus providing stable and timely data. But using a greater SensorAcqTime can reduce power consumption.

IntervalRespTime:

The IntervalRespTime is relevant when Push Mode is selected, and is the time delay, in seconds, between completion of the Prime sending one set of sampled data and the start of sending the next sample set. The default is 0.0 seconds, which means the TCM will begin sending new data as soon as the previous data set has been sent. Note that the inverse of the IntervalRespTime is somewhat greater than the sample rate, since the IntervalRespTime does not include actual acquisition time. Also, if IntervalRespTime is less than SensorAcqTime, then repeated data will be sent which normally is undesirable.

7.3.25 kGetAcqParams (frame ID 25_d)

This frame queries the Prime for its acquisition parameters. The frame has no payload.

7.3.26 kAcqParamsDone (frame ID 26_d)

This frame is the response to kSetAcqParams frame. The frame has no payload.

7.3.27 kAcqParamsResp (frame ID 27_d)

This frame is the response to kGetAcqParams frame. The payload should be the same as for the kSetAcqParams frame.

7.3.28 kPowerDownDone (frame ID 28_d)

This frame is the response to kPowerDown frame. This indicates that the Prime successfully received the kPowerDone frame and is in the process of powering down. The frame has no payload.

7.3.29 kFactoryUserCal (frame ID 29_d)

This frame clears the magnetic sensor field calibration coefficients and returns to the factory-loaded coefficients. The frame has no payload. This frame must be followed by the kSave frame to change in non-volatile memory.

7.3.30 kFactoryUserCalDone (frame ID 30_d)

This frame is the response to kFactoryUserCal frame. The frame has no payload.

7.3.31 kTakeUserCalSample (frame ID 31_d)

This frame commands the unit to take a sample during field calibration, assuming kUserCalAutoSampling is set to FALSE. Note that the kUserCalStableCheck condition must be met for the sample to be taken. The frame has no payload.

7.3.32 kFactoryInclCal (frame ID 36_d)

This frame clears the user accelerometer calibration coefficients. The frame has no payload. This frame must be followed by the kSave frame to change in non-volatile memory.

7.3.33 kFactoryInclCalDone (frame ID 37_d)

This frame is the response to kFactoryInclCal frame. The frame has no payload.

8 Code Examples

The following example files (CommProtocol.h, CommProtocol.cp, Prime.h and Prime.cp) would be used together for proper communication with a Prime module.

NOTE: The user also needs to create the following: SystemSerPort.h; Processes.h, TickGenerator.h.

8.1 Header File & CRC-16 Function

```
// type declarations
typedef struct
{
   UInt8 pollingMode, flushFilter;
   Float32 sensorAcqTime, intervalRespTime;
} __attribute__ ((packed)) AcqParams;

typedef struct
{
   Float32 stdDevErr;
   Float32 xCoverage;
   Float32 yCoverage;
   Float32 zCoverage;
   Float32 xyzAccelCoverage;
   Float32 accelStdDevErr;
} __attribute__ ((packed)) CalScore;

enum
{
   // Frame IDs (Commands)
   kGetModInfo = 1,        // 1
   kModInfoResp,           // 2
   kSetDataComponents,     // 3
   kGetData,               // 4
   kDataResp,              // 5
   kSetConfig,             // 6
   kGetConfig,             // 7
   kConfigResp,            // 8
   kSave,                  // 9
   kStartCal,              // 10
   kStopCal,               // 11
   kSetParam,              // 12
   kGetParam,              // 13
   kParamResp,             // 14
   kPowerDown,             // 15
   kSaveDone,              // 16
   kUserCalSampCount,      // 17
   kUserCalScore,          // 18
   kSetConfigDone,         // 19
   kSetParamDone,          // 20
   kStartIntervalMode,     // 21
```

```
   kStopIntervalMode,     // 22
   kPowerUp,              // 23
   kSetAcqParams,         // 24
   kGetAcqParams,         // 25
   kAcqParamsDone,        // 26
   kAcqParamsResp,        // 27
   kPowerDoneDown,        // 28
   kFactoryUserCal,       // 29
   kFactoryUserCalDone,   // 30
   kTakeUserCalSample,    // 31
   kFactoryInclCal,       // 36
   kFactoryInclCalDone,   // 37

   // Param IDs
   kFIRConfig = 1,  // 3-AxisID(UInt8)+Count(UInt8)+Value(Float64)+...

   // Data Component IDs
   kHeading = 5,          // 5 - type Float32
   kDistortion = 8,       // 8 - type boolean
   kPAligned = 21,        // 21 - type Float32
   kRAligned,             // 22 - type Float32
   kIZAligned,            // 23 - type Float32
   kPAngle,               // 24 - type Float32
   kRAngle,               // 25 - type Float32
   kXAligned = 27,        // 27 - type Float32
   kYAligned,             // 28 - type Float32
   kZAligned,             // 29 - type Float32

   // Configuration Parameter IDs
   kDeclination = 1,      // 1 - type Float32
   kTrueNorth,            // 2 - type boolean
   kMountingRef = 10,     // 10 - type UInt8
   kUserCalStableCheck,   // 11 - type boolean
   kUserCalNumPoints,     // 12 - type UInt32
   kUserCalAutoSampling,  // 13 - type boolean
   kBaudRate,             // 14 - UInt8

   // Mounting Reference IDs
   kMountedStandard = 1,  // 1
   kMountedXUp,           // 2
   kMountedYUp,           // 3
   kMountedStdPlus90,     // 4
   kMountedStdPlus180,    // 5
   kMountedStdPlus270,    // 6
   // Result IDs
   kErrNone = 0,          // 0
   kErrSave,              // 1
};

// function to calculate CRC-16
UInt16 CRC(void * data, UInt32 len)
{
   UInt8 * dataPtr = (UInt8 *)data;
   UInt32 index = 0;
```

```
    // Update the CRC for transmitted and received data using
    // the CCITT 16bit algorithm (X^16 + X^12 + X^5 + 1).
    UInt16 crc = 0;
    while(len--)
    {
          crc = (unsigned char)(crc >> 8) | (crc << 8);
          crc ^= dataPtr[index++];
          crc ^= (unsigned char)(crc & 0xff) >> 4;
          crc ^= (crc << 8) << 4;
          crc ^= ((crc & 0xff) << 4) << 1;
    }
    return crc;
}
```

8.2 CommProtocol.h File

Note: This file contains objects used to handle the serial communication with the unit. Unfortunately, these files are not available as the program was written on a non-PC computer. The comments in the code should explain what is expected to be sent or received from these functions so that you can write this section for your specific platform. For example, with the TickGenerator.h, you would need to write a routing that generates 10msec ticks.

```
#pragma once

#include "SystemSerPort.h"
#include "Processes.h"


//
//CommHandler is a base class that provides a callback for
//incoming messages.
//
class CommHandler
{
   public:
         // Call back to be implemented in derived class.
         virtual void HandleComm(UInt8 frameType, void * dataPtr =
NULL, UInt16 dataLen = 0) {}
};

//
//CommProtocol handles actual serial communication with the unit.
//Process is a base class that provides CommProtocol with
//cooperative parallel processing.  The Control method will be
//called by a process manager on a continuous basis.
//
class CommProtocol : public Process
{
   public:
         enum
         {
               // Frame IDs (Commands)
               kGetModInfo = 1,       // 1
               kModInfoResp,          // 2
               kSetDataComponents,    // 3
               kGetData,              // 4
               kDataResp,             // 5

               // Data Component IDs

               kHeading = 5,          // 5 - type Float32
               kPAligned = 21,        // 21 - type Float32
               kRAligned,             // 22 - type Float32
               kIZAligned,            // 23 - type Float32
               kPAngle,               // 24 - type Float32
```

```
            kRAngle,                // 25 - type Float32
        };


        enum
        {
            //maximum size of our input buffer
kBufferSize = 512,

            //minimum size of a serial packet
            kPacketMinSize = 5
        };

        //SerPort is a serial communication object abstracting
//the hardware implementation
        CommProtocol(CommHandler * handler = NULL, SerPort * serPort =
NULL);

        void Init(UInt32 baud = 38400);

        void SendData(UInt8 frame, void * dataPtr = NULL, UInt32 len =
0);
        void SetBaud(UInt32 baud);

   protected:
        CommHandler * mHandler;
        SerPort * mSerialPort;

        UInt8 mOutData[kBufferSize], mInData[kBufferSize];
        UInt16 mExpectedLen;
        UInt32 mOutLen, mOldInLen, mTime, mStep;

        UInt16 CRC(void * data, UInt32 len);
        void Control();
};
```

8.3 CommProtocol.cp File

```
#include "CommProtocol.h"

// import an object that will provide a 10mSec tick count through
// a function called Ticks()
#include "TickGenerator.h"


//
//  SerPort is an object that controls the physical serial
// interface.  It handles sending out the characters, and buffers
// the characters read in until we are ready for them.
//
CommProtocol::CommProtocol(CommHandler * handler, SerPort * serPort)
 : Process("CommProtocol")
{
// store the object that will parse the data when it is fully
// received
   mHandler = handler;
   mSerialPort = serPort;
   Init();
}

//
// Initialize the serial port and variables that will control
// this process
//
void CommProtocol::Init(UInt32 baud)
{
   SetBaud(baud);
   mOldInLen = 0;         // no data previously received

   mStep = 1;             // goto the first step of our process
}

//
//  Put together the frame to send to the unit
//
void CommProtocol::SendData(UInt8 frameType, void * dataPtr, UInt32
len)
{
   UInt8 * data = (UInt8 *)dataPtr;  // the data to send
   UInt32 index = 0;      // our location in the frame we are
                          // putting together
   UInt16 crc;      // the CRC to add to the end of the packet
   UInt16 count;    // the total length the packet will be

   count = (UInt16)len + kPacketMinSize;

   // exit without sending if there is too much data to fit
   // inside our packet
```

```
   if(len > kBufferSize - kPacketMinSize) return;

   // Store the total len of the packet including the len
// bytes(2), the frame ID(1), the data (len), and the crc (2).
   // If no data is sent, the min len is 5
   mOutData[index++] = count >> 8;
   mOutData[index++] = count & 0xFF;

   // store the frame ID
   mOutData[index++] = frameType ;

   // copy the data to be sent
   while(len--) mOutData[index++] = *data++;

   // compute and add the crc
   crc = CRC(mOutData, index);
   mOutData[index++] = crc >> 8 ;
   mOutData[index++] = crc & 0xFF ;

   // Write block will copy and send the data out the serial port
   mSerialPort->WriteBlock(mOutData, index);
}



//
// Call the functions in serial port necessary to change the
// baud rate
//
void CommProtocol::SetBaud(UInt32 baud)
{
   mSerialPort->SetBaudRate(baud);

   // clear any data that was already waiting in the buffer
   mSerialPort->InClear();
}

//
// Update the CRC for transmitted and received data using the
// CCITT 16bit algorithm (X^16 + X^12 + X^5 + 1).
//
UInt16 CommProtocol::CRC(void * data, UInt32 len)
{
   UInt8 * dataPtr = (UInt8 *)data;
   UInt32 index = 0;

   UInt16 crc = 0;
   while(len--)
   {
         crc = (unsigned char)(crc >> 8) | (crc << 8);
         crc ^= dataPtr[index++];
         crc ^= (unsigned char)(crc & 0xff) >> 4;
         crc ^= (crc << 8) << 4;
         crc ^= ((crc & 0xff) << 4) << 1;
```

```
   }
   return crc;
}

//
//  This is called each time this process gets a turn to execute.
//
void CommProtocol::Control()
{
   // InLen returns the number of bytes in the input buffer of
   // the serial object that are available for us to read.
   UInt32 inLen = mSerialPort->InLen();

   switch(mStep)
   {
         case 1:
         {
               // wait for length bytes to be received by the
               // serial object
               if(inLen >= 2)
               {
                     // Read block will return the number of
                     // requested (or available) bytes that are in
                     // the serial objects input buffer.
                     // read the byte count
                     mSerialPort->ReadBlock(mInData, 2);

                     // byte count is ALWAYS transmitted in big
                     // Endian, copy byte count to mExpectedLen to
                     // native Endianess
                     mExpectedLen =
                           (mInData[0] << 8) | mInData[1];

                     // Ticks is a timer function.  1 tick =
                     // 10msec.
                     // wait up to 1/2s for the complete frame
                     // (mExpectedLen) to be received
                     mTime = Ticks() + 50 ;
                     mStep++ ;  //go to next step in the process
               }
               break ;
         }

         case 2:
         {
               // wait for msg complete or timeout
               if(inLen >= mExpectedLen - 2)
               {
                     // calculated and received crcs.
                     UInt16 crc, crcReceived;

                     // Read block will return the number of
                     // requested (or available) bytes that are in
                     // the serial objects input buffer.
```

```
                        mSerialPort->ReadBlock(&mInData[2],
                                    mExpectedLen - 2);
                        // in CRC verification, don't include the CRC
                        // in the recalculation (-2)
                        crc = CRC(mInData, mExpectedLen - 2);
                        // CRC is also ALWAYS transmitted in big
                        // Endian
                        crcReceived = (mInData[mExpectedLen - 2]
                              << 8) | mInData[mExpectedLen - 1] ;

                        if(crc == crcReceived)
                        {
                              // the crc is correct, so pass the frame
                              // up for processing.
                              if(mHandler)
                                    mHandler->HandleComm(mInData[2],
                                          &mInData[3], mExpectedLen -
                                                kPacketMinSize);
                        }
                        else
                        {
                              // crc's don't match so clear everything
                              // that is currently in the input buffer
                              // since the data is not reliable.
                              mSerialPort->InClear();
                        }

                        // go back to looking for the length bytes.
                        mStep = 1 ;
                  }
                  else
                  {
                        //Ticks is a timer function. 1 tick = 10msec.
                        if(Ticks() > mTime)
                        {
                              // Corrupted message.  We did not get
                              // the length we were expecting within
                              // 1/2sec of receiving the length bytes.
                              // Clear everything in the input buffer
                              // since the data is unreliable
                              mSerialPort->InClear();

                              // Look for the next length bytes
                              mStep = 1 ;
                        }
                  }
                  break ;
            }

            default:
                  break ;
      }
}
```

8.4 Prime.h File

```
#pragma once

#include "Processes.h"
#include "CommProtocol.h"


//
// This file contains the object providing communication to the
// Prime.  It will set up the unit and parse packets received
// Process is a base class that provides Prime with cooperative
// parallel processing.  The Control method will be
// called by a process manager on a continuous basis.
//
class Prime : public Process, public CommHandler
{
   public:
         Prime(SerPort * serPort);
         ~Prime();

   protected:
         CommProtocol * mComm;

         UInt32 mStep, mTime, mResponseTime;

         void HandleComm(UInt8 frameType, void * dataPtr = NULL, UInt16
dataLen = 0);
         void SendComm(UInt8 frameType, void * dataPtr = NULL, UInt16
dataLen = 0);

         void Control();
};
```

8.5 Prime.cp File

```
#include "Prime.h"
#include "TickGenerator.h"

const UInt8 kDataCount = 4;// We will be requesting 4 components
                           //(Heading, pitch, roll, temperature)
//
//  This object polls the Prime unit once a second for heading,
// pitch, roll and temperature.
//

Prime::Prime(SerPort * serPort)
 : Process("Prime")
{
   // Let the CommProtocol know this object will handle any
   // serial data returned by the unit
   mComm = new CommProtocol(this, serPort);

   mTime = 0;
   mStep = 1;
}

Prime::~Prime()
{
}

//
// Called by the CommProtocol object when a frame is completely
// received
//
void Prime::HandleComm(UInt8 frameType, void * dataPtr,
                                                  UInt16 dataLen)
{
   UInt8 * data = (UInt8 *)dataPtr;

   switch(frameType)
   {
         case CommProtocol::kDataResp:
         {
               // Parse the data response
               UInt8 count = data[0]; // The number of data
                                      // elements returned
               UInt32 pntr = 1;       // Used to retrieve the
                                      // returned elements

               // The data elements we requested
               Float32 heading, pitch, roll, temperature;

               if(count != kDataCount)
               {
```

```
        // Message is a function that displays a C
        // formatted string (similar to printf)
        Message("Received %u data elements instead of
              the %u requested\r\n", (UInt16)count,
         (UInt16)kDataCount);
        return;
}

// loop through and collect the elements
while(count)
{
        // The elements are received as {type (ie.
        // kHeading), data}
        switch(data[pntr++])   // read the type and
                               // go to the first
                               // byte of the data
        {
              // Only handling the 4 elements we are
              // looking for
              case CommProtocol::kHeading:
              {
                    // Move(source, destination, size
                    // (bytes)).  Move copies the
                    // specified number of bytes from
                    // the source pointer to the
                    // destination pointer.
                    // Store the heading.
                    Move(&(data[pntr]), &heading,
                                       sizeof(heading));

                    // increase the pointer to point
                    // to the next data element type
                    pntr += sizeof(heading);
                    break;
              }

              case CommProtocol::kPAngle:
              {
                    // Move(source, destination, size
                    // (bytes)).  Move copies the
                    // specified number of bytes from
                    // the source pointer to the
                    // destination pointer.
                    // Store the pitch.
                    Move(&(data[pntr]), &pitch,
                                       sizeof(pitch));

                    // increase the pointer to point
                    // to the next data element type
                    pntr += sizeof(pitch);
                    break;
              }

              case CommProtocol::kRAngle:
```

```
                {
                      // Move(source, destination, size
                      // (bytes)).  Move copies the
                      // specified number of bytes from
                      // the source pointer to the
                      // destination pointer.
                      // Store the roll.
                      Move(&(data[pntr]), &roll,
                                        sizeof(roll));

                      // increase the pointer to point
                      // to the next data element type
                      pntr += sizeof(roll);
                      break;
                }

                case CommProtocol::kTemperature:
                {
                      // Move(source, destination, size
                      // (bytes)).  Move copies the
                      // specified number of bytes from
                      // the source pointer to the
                      // destination pointer.
                      // Store the heading.
                      Move(&(data[pntr]), &temperature,
                                  sizeof(temperature));

                      // increase the pointer to point
                      // to the next data element type
                      pntr += sizeof(temperature);
                      break;
                }

                default:
                      // Message is a function that
                      // displays a formatted string
                      // (similar to printf)
                      Message("Unknown type: %02X\r\n",
                                        data[pntr - 1]);
                      // unknown data type, so size is
                      // unknown, so skip everything
                      return;
                      break;
          }

          count--;    // One less element to read in
    }

    // Message is a function that displays a formatted
    // string (similar to printf)
    Message("Heading: %f, Pitch: %f, Roll: %f,
          Temperature: %f\r\n", heading, pitch, roll,
                                        temperature);
    mStep--;          // send next data request
```

```
            break;
        }

        default:
        {
            // Message is a function that displays a formatted
            // string (similar to printf)
            Message("Unknown frame %02X received\r\n",
                                    (UInt16)frameType);
            break;
        }
    }
}

//
//  Have the CommProtocol build and send the frame to the unit.
//
void Prime::SendComm(UInt8 frameType, void * dataPtr, UInt16 dataLen)
{
    if(mComm) mComm->SendData(frameType, dataPtr, dataLen);
    // Ticks is a timer function.  1 tick = 10msec.
    mResponseTime = Ticks() + 300;     // Expect a response within
                                       // 3 seconds
}

//  This is called each time this process gets a turn to execute.
//
void Prime::Control()
{
    switch(mStep)
    {
        case 1:
        {
            // the compents we are requesting, preceded by the
            // number of...
            UInt8 pkt[kDataCount + 1];
            // ...components being requested

            pkt[0] = kDataCount;
            pkt[1] = CommProtocol::kHeading;
            pkt[2] = CommProtocol::kPAngle;
            pkt[3] = CommProtocol::kRAngle;

            SendComm(CommProtocol::kSetDataComponents,
                                    pkt, kDataCount + 1);

            // Ticks is a timer function.  1 tick = 10msec.
            mTime = Ticks() + 100; // Taking a sample in 1s.
            mStep++;          // go to next step of process
            break;
        }

        case 2:
        {
```

```
                // Ticks is a timer function.  1 tick = 10msec.
                if(Ticks() > mTime)
                {
                      // tell the unit to take a sample
                      SendComm(CommProtocol::kGetData);
                      mTime = Ticks() + 100; // take a sample every
                                             // second
                      mStep++;
                }
                break;
          }

          case 3:
          {
                // Ticks is a timer function.  1 tick = 10msec.
                if(Ticks() > mResponseTime)
                {
                      Message("No response from the unit.  Check
                                  connection and try again\r\n");
                      mStep = 0;
                }
                break;
          }

          default:
                break;
    }
}
```