User Manual

Autonomous Navigation Solutions

Inertial Sense, Inc.

Table of Contents

Software Release 1.8.6 - 16 March 2022

INERTIAL SENSE
autonomous navigation solutions

1. Overview

The µINS GPS aided Inertial Navigation System, µAHRS Attitude Heading Reference System, and the µIMU Inertial Measurement Unit monitor many different types of measurements including rotation, acceleration, GPS position, magnetic flux density, pressure and velocity. The Inertial Sense SDK provides a software interface to allow communication with the device including setting configuration options, retrieving specific data, and listening for data broadcasts.

1.1 µIMU, µAHRS, and µINS

The **µIMU™** is a miniature calibrated sensor module consisting of an Inertial Measurement Unit (IMU), magnetometer, barometer, and L1 GPS (GNSS) receiver. Data out includes angular rate, linear acceleration, magnetic field, barometric altitude, and GPS WGS84 geo-position. All systems include a comprehensive sensor calibration for bias, scale-factor, and cross-axis alignment, minimizing manufacturing variation and maximizing system performance.

The **µAHRS™** is an Attitude Heading Reference System (AHRS) that includes all functionality of the µIMU™ and fuses IMU and magnetometer data to estimate roll, pitch, and heading.

The **µINS™** is a GPS (GNSS) aided inertial navigation system (GPS-INS) module that includes all functionality of the **µIMU™** and provides orientation, velocity, and position. Sensor data from MEMs gyros, accelerometers, magnetometers, barometric pressure, and GPS/GNSS is fused to provide optimal measurement estimation.

The patented package is smaller than 3 stacked dimes and fits into most industrial and commercial application designs.

1.2 Features

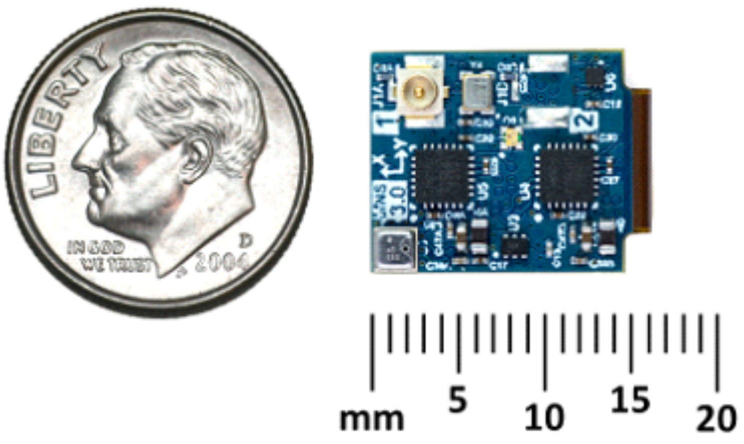
- Up to 1KHz IMU Update Rate
- Attitude (Roll, Pitch, Yaw, Quaternions, DCM), Velocity, and Position (LLA, ECEF, NED)UTC Time Synchronized
- Dual Redundant IMUs Calibrated for Bias, Scale Factor, and Cross-Axis Alignment
- Coning & Sculling Integrals (Δ theta, Δ velocity)
- Barometric Pressure and Humidity
- u-Blox L1 GPS (GNSS) Receiver
- -40°C to 85°C Temperature Compensation
- Configurable Binary and NMEA ASCII Protocol
- Strobe In/Out Data Sync (Camera Shutter Event)
- Fast Integration using SDK and Example Software
- Data Logging (SDK and Application Software)
- Miniature Surface Mount Package:
    - 16.5 x 12.6 x 4.6 mm, 1.3 grams

1.3 Interfaces

	Module	**EVB 1.x**	**EVB 2.x**	**Rugged 1.0**
USB	Yes	Yes	Yes	Yes
TTL/UART	Yes	Yes	Yes	Yes
RS232/RS422/RS485	No	Yes	Yes	Yes
CAN	Yes	Yes	Yes	Yes
SPI	Yes	Yes	Yes	No
Integrated XBee Radio (RTK)	No	No	Yes (Option)	No
WiFi/BTLE	No	No	Yes	No
GPS Antenna Ports (Dual=Compassing)	Dual (Option)	Single	Dual (Option)	Dual (Option)

1.4 Applications

- Drone Navigation
- Unmanned Vehicle Payloads
- Stabilized Platforms
- Antenna and Camera Pointing
- First Responder and Personnel Tracking
- Pedestrian and Auto Outdoor / Indoor Navigation
- Health, Fitness, and Sport Monitors
- Hand-held Devices
- Robotics and Ground Vehicles
- Maritime



Inertial Sense, Inc.
3000 S Sierra Vista Way Suite 2, Provo, UT 84606 USA
Phone 801-610-6771
Email support@inertialsense.com
Website: InertialSense.com

2. Data Sheet

Download PDF

3. Dimensions and Pinouts

3.1 Module (µINS, µAHRS, or µIMU)

Download PDF

3.2 Rugged-1 (µINS, µAHRS, or µIMU)

Download PDF

3.3 Rugged-2 (µINS, µAHRS, or µIMU)

Download PDF

3.4 Hardware Design Files

The Inertial Sense hardware design files are available on our IS-hdw repository to facilitate product hardware development and integration.

- **PCB Libraries** - Schematic and layout files for printed circuit board designs.
- **Products** - 3D models and resources for the uINS, Rugged, EVB, and products useful for CAD and circuit board designs.

4. Getting Started

The following video is a quick intro to the EvalTool for Windows. For other applications scroll below.

4.1 1. Installing Software

Inertial Sense software provides a way to view, manipulate, stream, and record the data generated by a µINS, µAHRS, or µIMU.

4.1.1 EvalTool (Windows Only)

The EvalTool is a graphical Windows-based desktop program that allows users to explore and test functionality of the Inertial Sense products in real-time.

Download the EvalTool installer from the Inertial Sense releases page. Run the .exe file and follow the instructions to complete the installation.

4.1.2 CLTool (Windows, Linux, and OS X)

The CLTool is a command line utility that can be used to read and display data, update firmware, and log data from Inertial Sense products.

CLTool must be compiled from our source code. Follow the instructions on the CLTool page.

4.1.3 SDK (Windows, Linux)

Software development kit to interface with Inertial Sense products.

Download the file named "Source code" from our releases page. The extracted folder contains code libraries as well as example projects.

4.2 2. Connecting Your Hardware

Select the evaluation product from the list below to view instructions on basic connection to a computer.

- EVB-1
- EVB-2
- Rugged Units
- PCB Module

4.3 3. Configuring Settings

The configuration settings found in DID_FLASH_CONFIG need to be set to appropriate values using either the EvalTool, the CLTool, or the SDK. A full explanation can be found on the System Configuration page. Most users initially configure the unit using the EvalTool. This is done by going to the Data Sets Tab and selecting DID_FLASH_CONFIG. The values of each field can then be edited. Pay special attention to the following fields:

1. gps1AntOffset[X-Z] - Position offset of sensor frame with respect to GPS1 antenna.
2. gps2AntOffset[X-Z] - Position offset of sensor frame with respect to GPS2 antenna.
3. insDynModel - Dynamic model that most closely represents sensor application.*

   *Note - This feature is experimental. For most applications setting 8 will yield the best results. The user is encouraged to try different settings.

4.4 4. Evaluation and Testing

Once a connection to the unit has been established, please follow one of the following guides to get started with the software tool of choice:

- EvalTool
- CLTool
- SDK Example Projects

4.5 Troubleshooting

If at any time issues are encountered, please check the troubleshooting sections of this manual.

5. IS Hardware

5.1 Hardware Integration: EVB-1

The Inertial Sense EVB1.x is a development board which contains the Inertial Sense µINS, µAHRS, or µIMU module. The EVB1.x functions as a breakout and communications board with the following features:

- Access to all communications pins on the module
- USB connection, either directly to the module or through an on-board FTDI chip
- RS232/RS422/RS485 transceiver

5.1.1 Connecting the board

For the purposes of basic evaluation, the easiest interface available on the EVB 1.x is the micro USB port. Connecting the micro USB port will provide power and communications with the installed module via the on-board FTDI chip. There are a variety of other interfaces available on the EVB 1.x.

If using GPS with the module, connect an antenna to the on-board SMA port. More information on compatible antennas is available on the GNSS Antennas page.

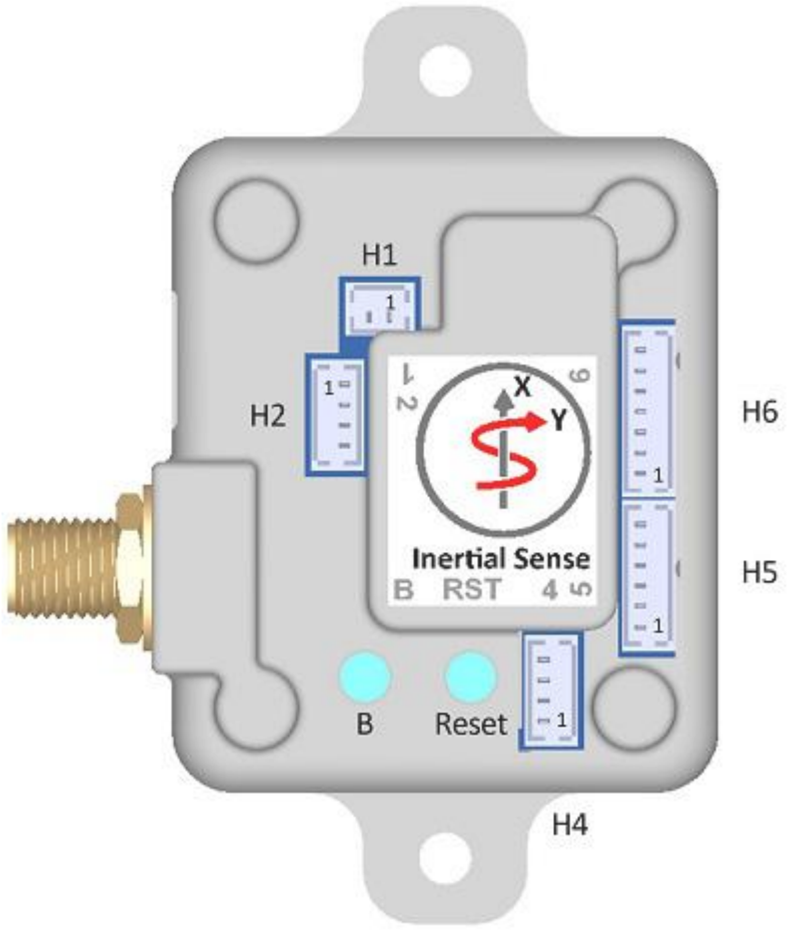
5.1.2 Power

The EVB-1 can be powered in using the following methods:

- 4-20V DC regulated supply via H1
- 3.3V DC regulated supply via any +3.3V header pin
- 5V USB connection

5.1.3 Pinout

Use Molex PicoBlade™ series connectors for the EVB-1 headers.



EVB Header and Pin 1 Locations

Warning

Please note that EVB-1 header pin 1 location and pin order is reversed from that designated by the header manufacturer, Molex.

H1 (Power)

Pin	Name	I/O	Description
⬛ 1	GND	-	-
🟥 2	VIN	-	4V – 20V supply voltage

H2 (I2C)

Pin	Name	I/O	Description
⬛ 1	GND	-	-
🟥 2	3.3V	-	3.3V supply. Output if H1 is supplied. Otherwise can be 3.3V input to supply uINS. Do NOT power VIN and 3.3V simultaneously!
🟧 3	G1/SDA/ I2C_EN	I/O	GPIO1/*I2C data (Hold HIGH during boot to enable *I2C)
🟨 4	G2/SCL/ STROBE	I/O	GPIO2/*I2C clock/Strobe time sync input.

H4 (Serial 0)

Pin	Name	I/O	Description
⬛ 1	GND	-	-
🟥 2	3.3V	-	3.3V supply. Output if H1 is supplied. Otherwise can be 3.3V input to supply uINS. Do NOT power VIN and 3.3V simultaneously!
🟧 3	G4/Rx0	I	‡GPIO4/Serial0 Input (TTL)
🟨 4	G3/Tx0	O	‡GPIO3/Serial0 Output (TTL)

H5 (RS232/RS485)

Pin	Name	I/O	Description
⬛ 1	GND	-	-
🟥 2	3.3V	-	3.3V supply. Output if H1 is supplied. Otherwise can be 3.3V input to supply uINS. Do NOT power VIN and 3.3V simultaneously!
🟧 3	232Rx1/485Rx1+	I	Serial 1 input (RS232)/Serial 1 input+ (RS485)
🟨 4	232Rx0/485Rx1-	I	Serial 0 input (RS232)/Serial 1 input- (RS485)
🟩 5	232Tx1/485Tx1+	O	Serial 1 output (RS232)/Serial 1 output+ (RS485)
🟦 6	232Tx0/485Tx1-	O	Serial 0 output (RS232)/Serial 1 output- (RS485)

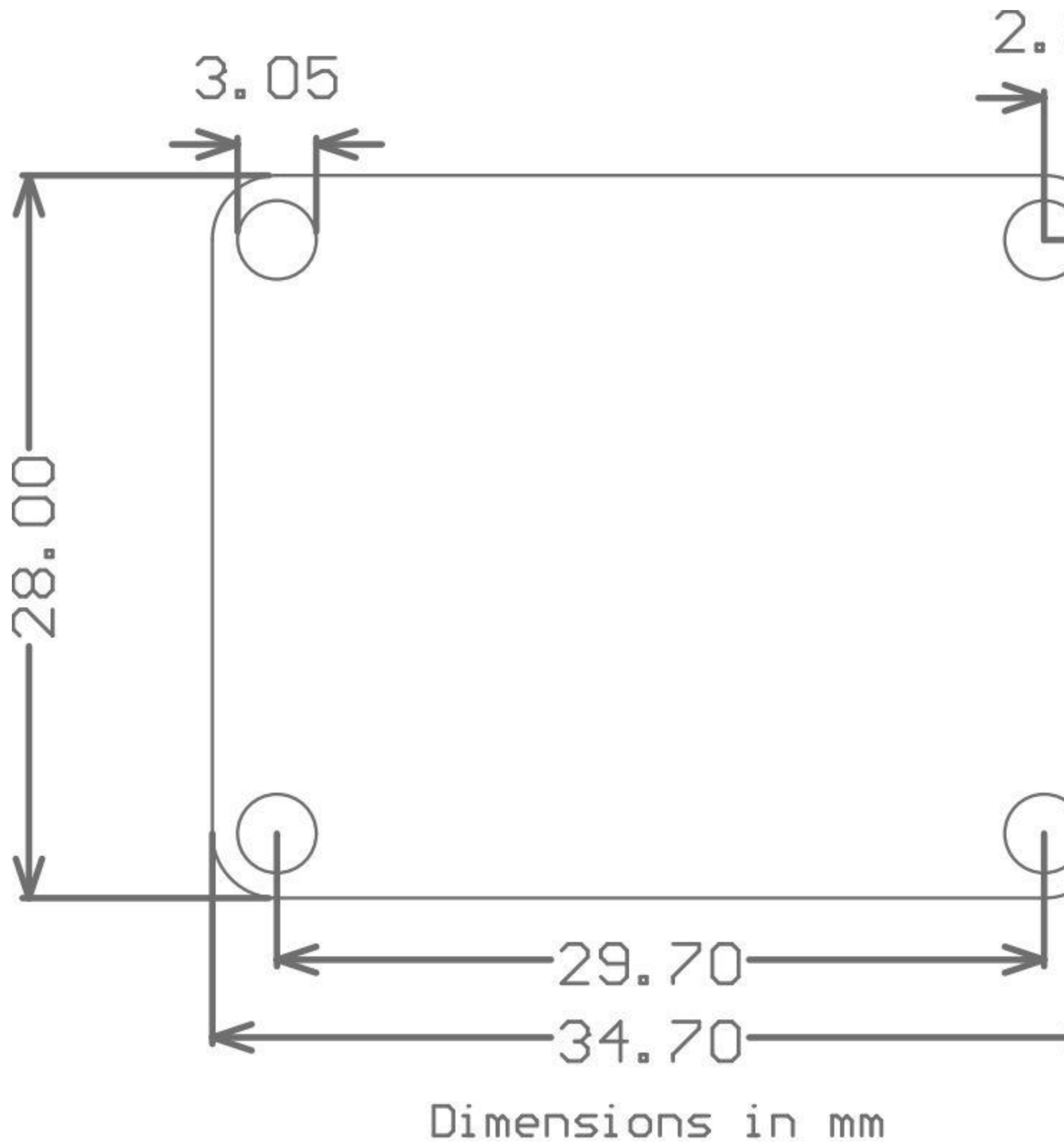
H6 (Serial 1/SPI)

Pin	Name	I/O	Description
⬛ 1	GND	-	-
🟥 2	3.3V	-	3.3V supply. Output if H1 is supplied. Otherwise can be 3.3V input to supply uINS. Do NOT power VIN and 3.3V simultaneously!
🟧 3	G5/SCLK/ STROBE	I/O	GPIO5/SPI SCLK/Strobe time sync input.
🟨 4	G6/Rx1/MOSI	I	GPIO6/Serial1 Input (TTL)/SPI MOSI
🟩 5	G7/Tx1/MISO	O	GPIO7/Serial1 Output (TTL)/SPI MISO
🟦 6	G8/CS/ STROBE	I/O	GPIO8/SPI CS/Strobe time sync input.
🟪 7	GPS_PPS	O	GPS PPS time synchronization output pulse (1Hz, 10% duty cycle)

* Feature will be available in future firmware update.

‡Tied to FTDI USB to serial converter ONLY when USB is connected.

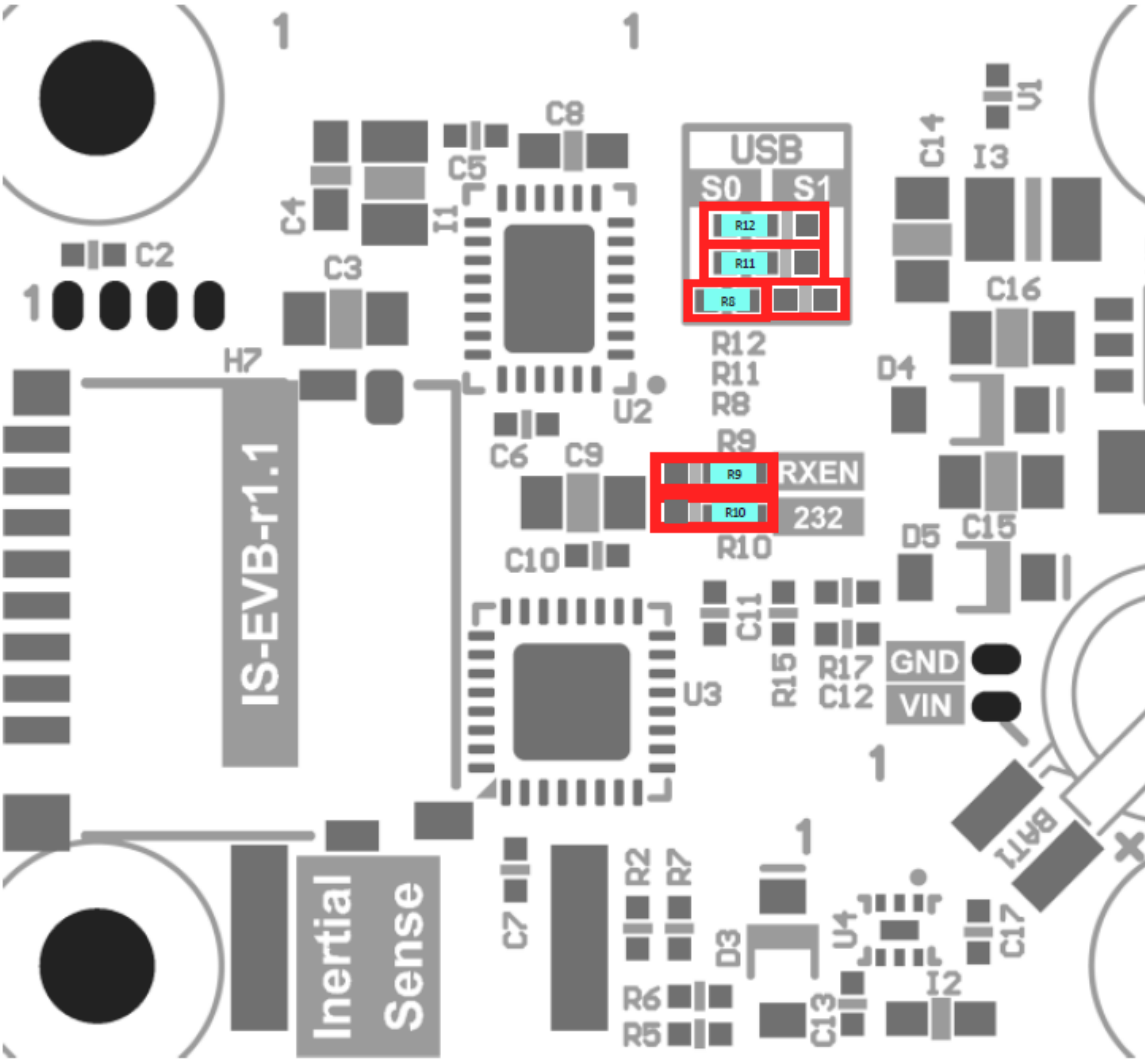
5.1.4 Mechanical



5.1.5 Jumpers

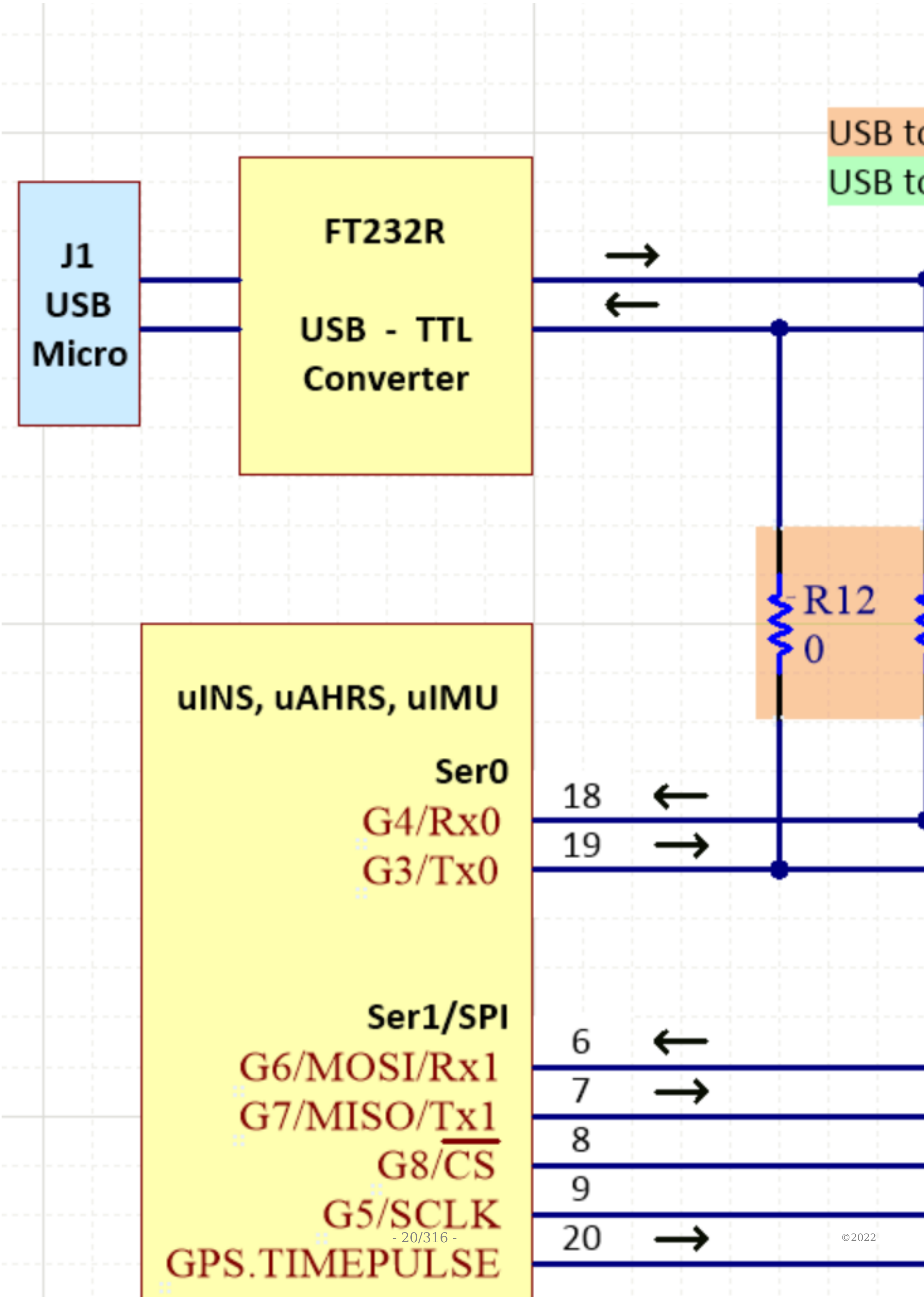
The jumpers identified in the following table are used to configure RS485/RS422 features and select which serial port the USB is connected to on the evaluation board.

Jumper	Label	Default*	Description
R9	RXEN	RXEN	Receiver enable for RS232 and RS485 function.
R10	232	232	Select RS232 or RS485/RS422 mode on H5. Setting jumper in the "232" position enables RS232 mode.
R8, R11, R12	S0 S1	S0	Connects USB to either Ser0 or Ser1. All three jumpers must move together into either the "S0" or "S1" positions.



EVB Bottom View

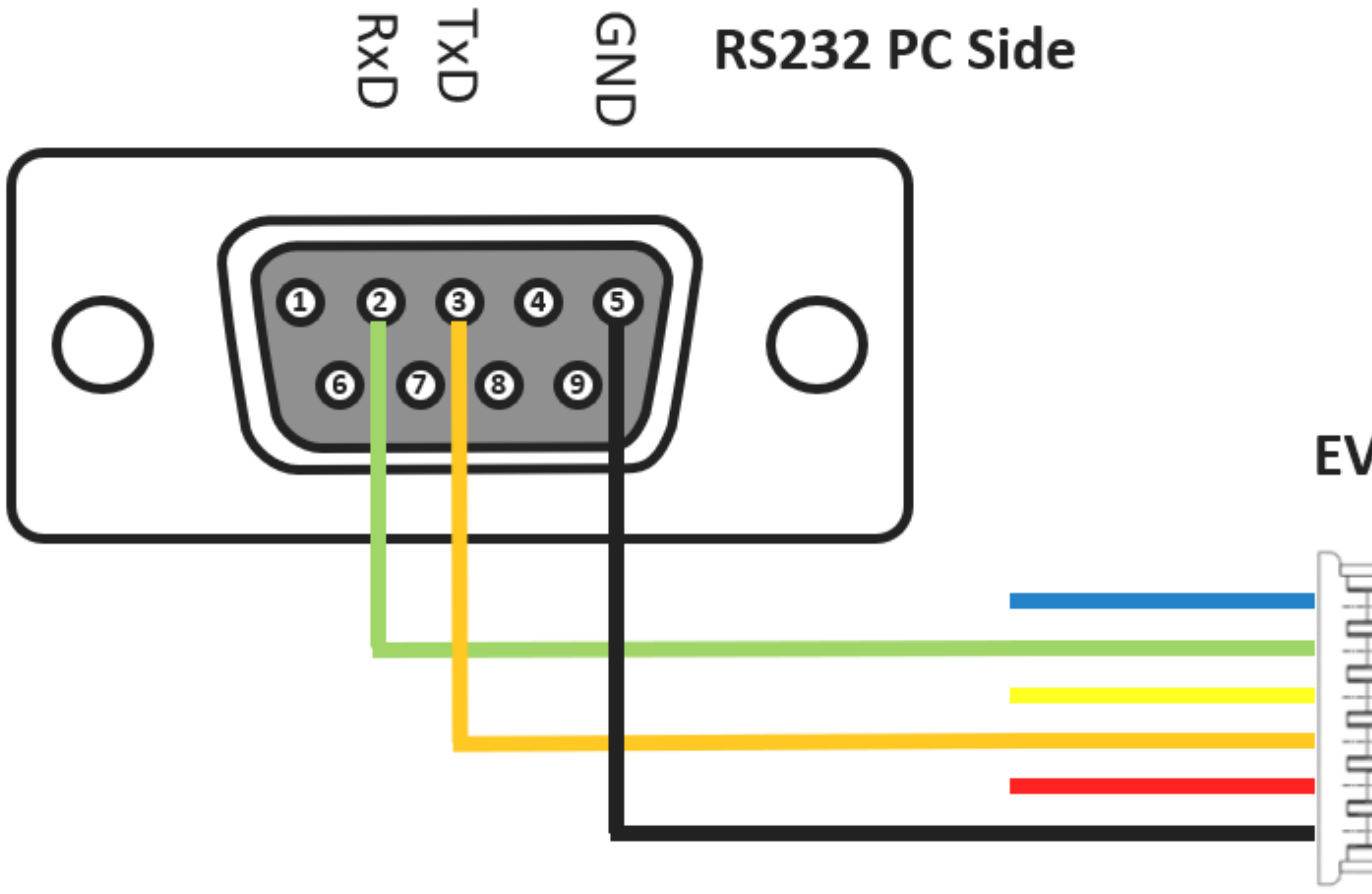
5.1.6 Schematic


USB to
USB to
J1
USB
Micro
FT232R
USB - TTL
Converter
R12
0
uINS, uAHRS, uIMU
Ser0
G4/Rx0
G3/Tx0
18
19
Ser1/SPI
G6/MOSI/Rx1
G7/MISO/Tx1
G8/CS
G5/SCLK
GPS.TIMEPULSE
6
7
8
9
20

EVB Simplified Schematic

5.1.7 RS232 DB9 Adapter

The EVB RS232 interface for the serial port 0 is enabled by default and available through header H5. The figure to the right illustrates how a standard DB9 connector is wired to this port.



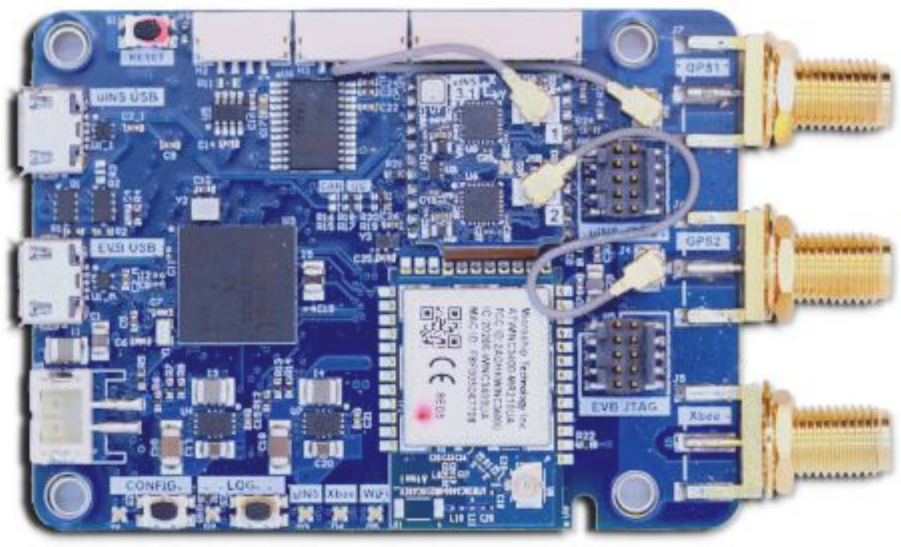
5.1.8 USB Driver

The EVB 1.x uses the FTDI FT232R USB to UART IC to provide a serial port over connection over USB. Depending on the operating system, it may be necessary to download and install the FTDI device driver for the FT232R to register properly as a serial port.

5.1.9 Using with Inertial Sense Software

Please return to the getting started page to get started programming, updating firmware, viewing data, and logging.

5.2 Hardware Integration: EVB-2



The Inertial Sense EVB-2 is a development board which contains the Inertial Sense µINS, µAHRS, or µIMU module. The EVB-2 builds on the foundation established by the EVB-1, but adds new features including:

- 915MHz XBee radio for RTK (real-time-kinematics)
- Wi-Fi and Bluetooth Low energy (BLE) for remote data viewing and logging operation
- Onboard logging to micro SD card.
- Dual antenna ports for GPS compassing
- Companion Microchip SAME70 processor that serves as a communication bridge between the µINS, µAHRS, or µIMU and all other interfaces.

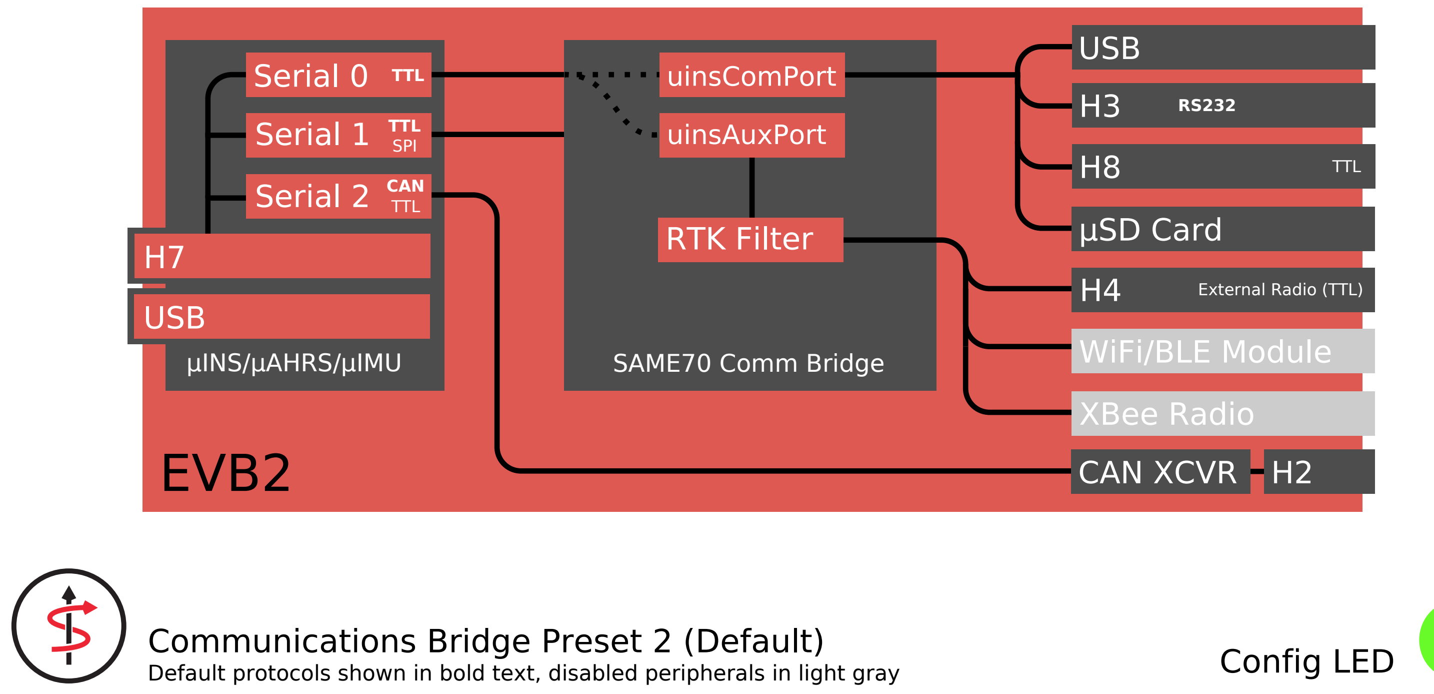
5.2.1 Configurations

The EVB-2 can be configured to preform a multitude of operations. Below are diagrams of connectivity and configuration options available. The configuration can be changed by pressing the tactile switch labeled "CONFIG" on the EVB-2 until the Config LED shows the desired mode.

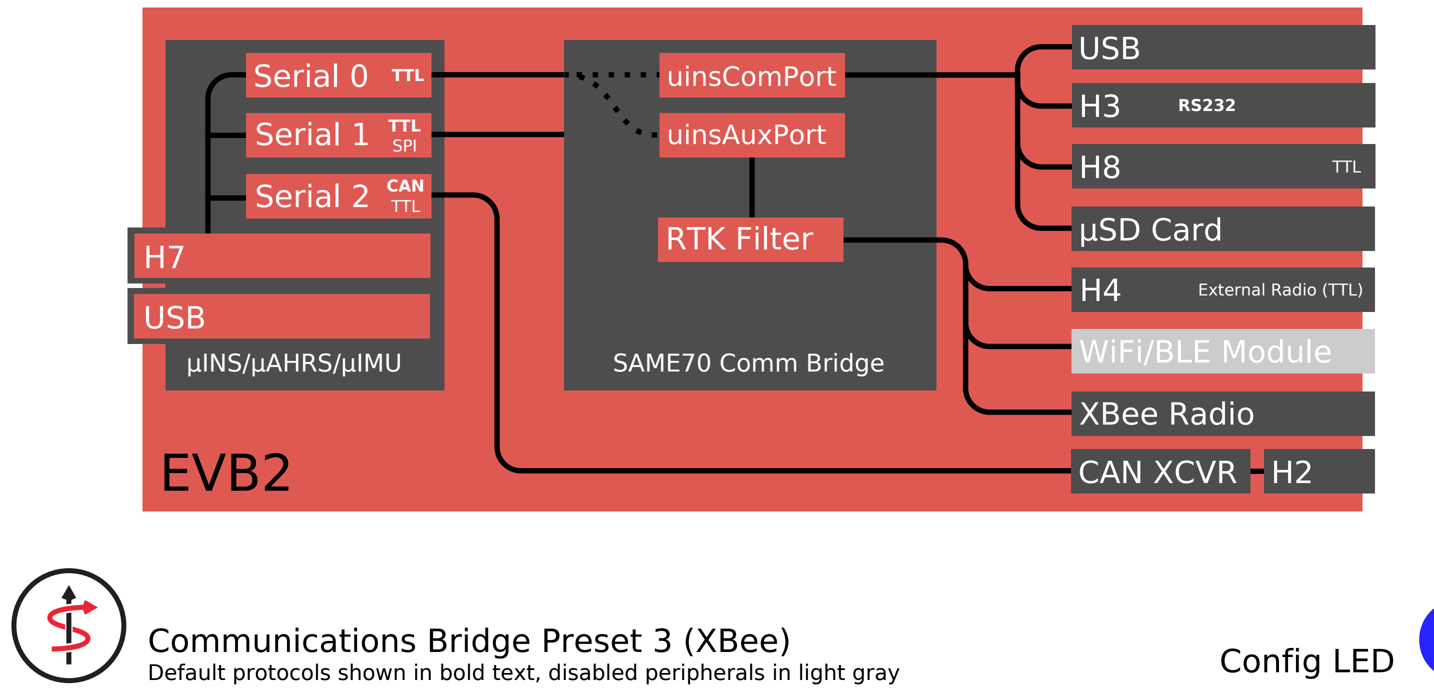
CBPreset	Config LED	Mode	ON	OFF
2	🟢	**Default**	RS232	WiFi/BLE Module, XBee
3	🔵	**XBee**	RS232, XBee	WiFi/BLE Module
4	🟣	**WiFi/BLE**	Wi-Fi/BLE Module, RS422/485	XBee
5	🔵	**SPI***	SPI*	Wi-Fi/BLE Module, XBee
6	🟡	**USB/232**	RS232, XBee	EVB2 to uINS Connection
7	⚪	**USB/422/485**	RS422/485	EVB2 to uINS Connection
1	⚫	**Off**		EVB2 to uINS Connection WiFi/BLE Module, XBee

*A reset is required following selection of this CBPreset to enable SPI on the uINS, in order to assert the uINS pin 10 (G9/nSPI_EN) during bootup.

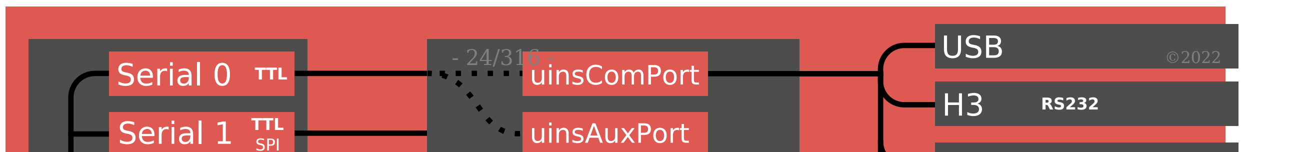
● Default



● XBee



● WiFi/BLE



In the USB hub modes (6-7), the following communications bridge/forwarding exists:

`DID_EVB_FLASH_CFG.uinsComPort` <-> All ports except XBee, WiFi, and XRadio `DID_EVB_FLASH_CFG.uinsAuxPort` <-> XBee, WiFi, and XRadio. XBee and WiFi only enabled when `DID_EVB_FLASH_CFG.cbPreset == EVB2_CB_PRESET_RS232_XBEE`

`EVB2_PORT_UINS0` is the default value for both of these. See the source code here.

Bit `EVB2_PORT_OPTIONS_RADIO_RTK_FILTER` of `DID_EVB_FLASH_CFG.portOptions` only allows RTK corrections to pass through the `uinsAuxPort`, reducing the wireless communications burden to only essential RTK base connections. This bit is enabled by default.

5.2.2 EVB-2 Connections

USB

The most commonly used user interface available on the EVB-2 is the `EVB USB` port. Connecting to the `EVB USB` port will provide power to the device as well communications with the onboard SAME70 processor. After connecting to a PC the EVB2 will appear as a virtual COM port and can be configured to communicate with every other communication bus on the board. This USB port should be used when updating the EVB-2 firmware or bootloader.

The EVB-2 also has a second USB port, `uINS USB`. This USB port also supplies power, but it connects directly to the µINS, µAHRS, or µIMU onboard the EVB-2. This USB port should be used when updating the µINS, µAHRS, or µIMU firmware or bootloader.

GPS Antenna(s)

If using GPS with the module, connect an appropriate antenna to `GPS1 (J7)`. If using the board for GPS compassing, connect a second antenna to `GPS2 (J5)`. More information on GPS antennas is available on the GNSS Antennas page.

XBee Antenna

If using the onboard XBee radio, ensure that the XBee radio U.FL connector (U10) is connected to the EVB-2 U.FL connector (J8). Connect an appropriate antenna to the EVB-2 RP-SMA connector (J5).

To communicate with another XBee radio the PID and NID need to match on both radios. The PID and NID can be set using `DID_EVB_FLASH_CFG.radioPID` and `DID_EVB_FLASH_CFG.radioNID`. After setting the PID and NID, the EVB-2 needs to be reset so the radio can be configured. The XBee LED will flash Yellow then Green if the configuration is successful. A red LED signifies a failed radio configuration.

Wi-Fi Antenna

The XBee radio can be used along with the Wi-Fi module. In this case, a Wi-Fi antenna must be provided and connected to the U.FL port on the Wi-Fi module (U8).

To use Wi-Fi alone, connect the U.FL connector on the Wi-Fi module (U8) to the EVB-2 U.FL connector (J8), and connect the Wi-Fi antenna to the XBee RP-SMA connector (J5).

Header Pinouts

Use JST-PH series connectors for EVB 2.x header H1. Maximum current is 2A per pin.
Use JST-GH series connectors for all other EVB 2.x headers. Maximum current is 1A per pin.

H1 (POWER)

Pin	Name	Type	Description
🟥 1	VIN	-	4.5–17V supply voltage
⬛ 2	GND	-	-

H2 (CAN)

To enable uINS CAN interface on H2, the U5 transceiver IC (TCAN334) must be loaded and the R27 jumper removed. To enable EVB CAN interface on H2, the U13 transceiver IC (TCAN334) must be loaded and the R51 jumper removed.

Pin	Name	Type	Description
⬛ 1	GND	-	
🟥 2	3.3V	-	3.3V supply
🟫 3	CANL	I/O	Low level CAN bus line.
🟧 4	CANH	I/O	High level CAN bus line.

H3 (RS-232/RS-485)

Pin	Name	Type	Description
⬛ 1	GND	-	-
🟫 2	232Tx-/485Tx-	O	Serial output (RS232 transmit-, RS485 transmit-)
🟧 3	485Tx+	O	Serial output (RS485 transmit+)
🟨 4	485Rx-	I	Serial input (RS485 receive-)
🟩 5	232Rx/485Rx+	I	Serial input (RS232 receive, RS485 receive+)

H4 (EXTERNAL RADIO)

Pin	Name	Type	Description
⬛ 1	GND	-	-
⬛ 2	GND	-	-
⬛ 3	GND	-	-
🟥 4	3.3V	-	Separate radio 3.3V supply. Regulator can supply up to 3.5A.
🟥 5	3.3V	-	" "
🟥 6	3.3V	-	" "
🟫 7	RxD	I	Serial input from radio transmit pin (TTL) PD25.
🟧 8	TxD	O	Serial output to radio receive pin (TTL) PD26.
🟨 9	$\overline{RST}$	O	Reset pin

H7 (MINS CONNECTIONS)

Pins on H7 (uINS) are shared with the EVB-2 processor. To prevent conflict when using H7, set EVB-2 CBPreset to 6 or 7 (USB hub mode) to prevent the EVB-2 processor from asserting any I/O on the H7.

Pin	Name	Type	Default Type	Description
⬛ 1	GND	-		-
⬛ 2	GND	-		-
🟥 3	3.3V	-		3.3V supply. Output if H1 is supplied. Otherwise can be 3.3V input.
🟥 4	DATA-RDY	I/O	I	Indicates data is read to be read from uINS.
🟫 5	G1/Rx2*/RxCAN*	I/O	I	GPIO1. Serial 2 input (TTL). Serial input pin from CAN transceiver.
🟨 6	G2/Tx2*/TxCAN*/ STROBE	I/O	O	GPIO2. Serial 2 output (TTL). Serial output pin to CAN transceiver. Strobe time sync input.
🟨 7	G3/Tx0	I/O	O	GPIO3. Serial 0 output (TTL) (tied to uINS-Ser0 Rx)
🟩 8	G4/Rx0	I/O	I	GPIO4. Serial 0 input (TTL) (tied to uINS-Ser0 Tx)
🟦 9	G5/SCLK/STROBE	I/O	I	GPIO5. SPI SCLK. Strobe time sync input.
🟪 10	G6/Rx1/MOSI	I/O	I	GPIO6. Serial 1 input (TTL). SPI MOSI (tied to uINS-Ser1 Tx)
⬜ 11	G7/Tx1/MISO	I/O	O	GPIO7. Serial 1 output (TTL). SPI MISO (tied to uINS-Ser1 Rx)
▨ 12	G8/CS/STROBE	I/O	O	GPIO8. SPI CS. Strobe time sync input.
🟫 13	G9/nSPI_EN/ STROBE /STROBE_OUT	I/O	I	GPIO9. Hold LOW during boot to enable SPI on G5-G8. Strobe time sync input or output.
🟨 14	GPS_PPS	O	O	GPS PPS time synchronization output pulse (1Hz, 10% duty cycle)

*Available on uINS-3.2 and later.

H8 (SAME70 CONNECTIONS)

Pin	Name	Type	Default Type	Description
⬛ 1	GND	-		-
⬛ 2	GND	-		-
🟥 3	3.3V	-		3.3V supply. Output if H1 is supplied. Otherwise can be 3.3V input.
🟥 4	3.3V	-		" "
🟫 5	M1/TXD1	O	O	GPIO1, USART 1 Output (TTL/SPI), Inverted Serial 1 Output (TTL)
🟧 6	M2/RXD1	I	I	GPIO2, USART 1 Input (TTL/SPI), Inverted Serial 1 Input (TTL)
🟨 7	M3/SCK1/QDA	I/O		GPIO3, USART 1 Clock (SPI), Quadrature Encoder Input A
🟩 8	M4/CS1/QDB	I/O		GPIO4, USART 1 Chip Select (SPI), Quadrature Encoder Input B
🟦 9	M5/DA0/AD4	I/O		GPIO5, DAC 0, ADC 4
🟪 10	M6/DA1/AD5	I/O		GPIO6, DAC 1, ADC 5
⬜ 11	M7/AD10	I/O		GPIO7, ADC 10
⬜ 12	M8/AD4	I/O		GPIO8, ADC 4
🟫 13	M9/AD1	I/O		GPIO9, ADC 1
🟧 14	M10/AD2	I/O		GPIO10, ADC 2

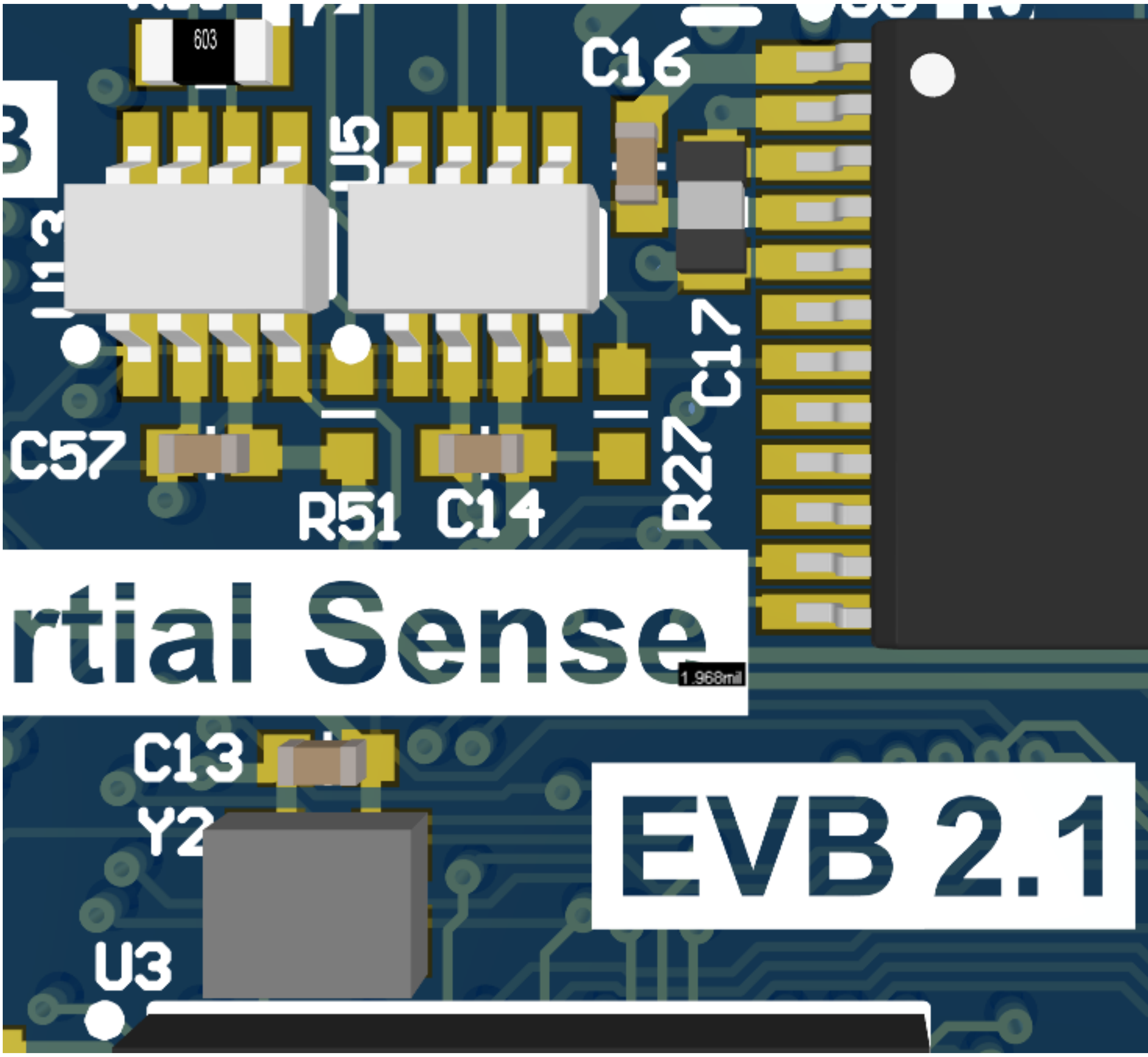
PWM and timer interrupt functions have been excluded from this table. Please see the MCU pin definition spreadsheet for more info.

5.2.3 uINS Connections

The EVB-2 ATSAME70 (E70) processor interfaces with the uINS over UART (serial 0 and 1) and SPI (serial 1).

CAN Bus

To use uINS CAN bus interface on the EVB-2, U5 (TCAN334 transceiver) should be loaded and R27 should not be loaded.



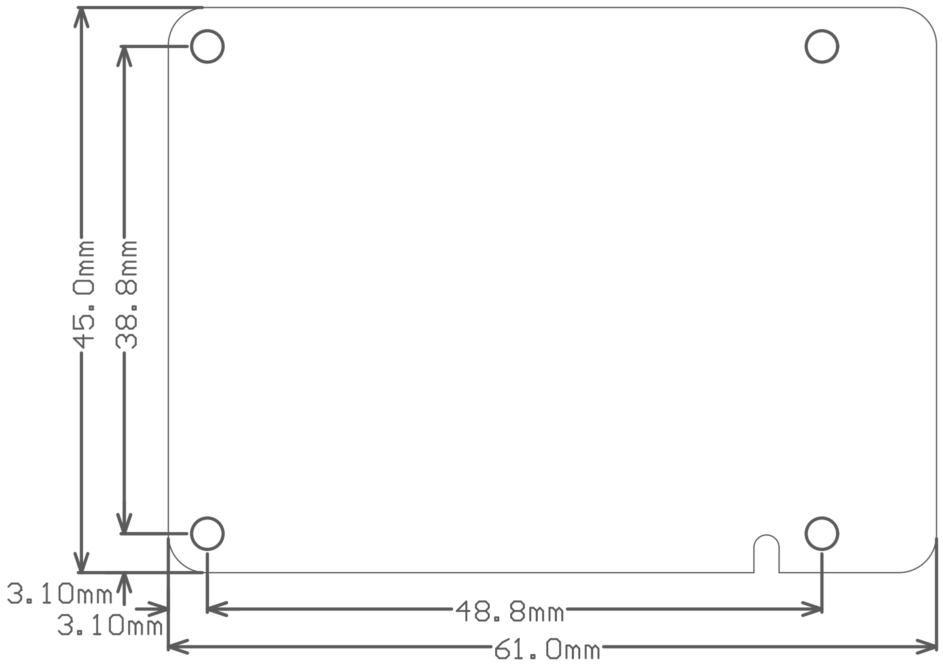
SPI

The EVB-2 must be put into CBPreset mode 6 (CONFIG led color cyan) followed by a system reset to enable SPI mode interface with the uINS. The EVB-2 (E70) project source code is available in the SDK for reference.

Serial 2

To use serial 2, the uINS CAN transciever (U5) and R27 must be depopulated to avoid contention on the data lines.

5.2.4 Mechanical Dimensions



PCB w/ SMA connectors: 70.0 x 45.0 x 12.0 mm

Enclosure: 78.0 x 48.0 x 18.0 mm

5.2.5 Using with Inertial Sense Software

Please return to the getting started page to get started programming, updating firmware, viewing data, and logging.

5.2.6 Updating Firmware

The EVB-2 and uINS firmware can be updated using the EVB-USB connector and only the uINS firmware when using the uINS-USB connector.

5.2.7 EVB-2 Design Files

The EVB-2 PCB assembly design files are available as open source hardware on GitHub.

In particular, full schematics for the board can be found here

5.2.8 XBee Radio Frequencies

The EVB-2 shipped standard with the XBee radio default frequency is 915MHz. An alternate frequency can be achieve by using the European version of the XBee Pro SX module.

XBee Pro SX Part#	Frequency	Frequency Band
XBP9X-DMUS-001 (default)	915 MHz	902MHz ~ 928MHz
XB8X-DMUS-001 (Europe)	868 MHz	863MHz ~ 870MHz

5.3 Hardware Integration: Rugged-1

The Inertial Sense Rugged-1 is a ruggedized carrier board and case for the Inertial Sense µINS, µAHRS, or µIMU module. The Rugged-1 has similar functions compared to the EVB-1, but in a more compact form factor with the following added features:

- Dual antenna ports for GPS compassing
- Integrated CAN transceiver

5.3.1 Connecting Your Unit

For the purposes of basic evaluation, the easiest interface available on the rugged is the included USB to Gecko connector cable, included in the evaluation kit. The cable provides power and communications with the installed module via the on-board FTDI chip.

GPS Antenna Ports

If using GPS with the module, connect an appropriate antenna to MMCX port ***1***. If the module is used for RTK compassing, connect a second antenna to MMCX port ***2***. MMCX port ***1*** is for ***GPS1*** and MMXC port ***2*** is ***GPS2***. These port were labeled ***A*** and ***B*** on older Rugged-1 units.

5.3.2 Pinout

Pin	Name	I/O	Description
1	GND	-	-
2	VIN	-	4V-20V supply voltage input[1]
3	USB.VCC	-	5V system supply input[1] from USB bus. Using this pin will enable the FTDI USB. Use the VIN pin instead to disable the FTDI USB.
4	USB.D+	I/O	USB Data Positive Line (Serial 0)
5	GPS_PPS	O	GPS PPS time synchronization output pulse (1Hz, 10% duty cycle)
6	USB.D-	I/O	USB Data Negative Line (Serial 0)
7	G3/Tx0/485Tx1-	I/O	Serial 0 output (TTL or RS232)[2] Serial 1 output- (RS485/RS422)[3]
8	G7/Tx1/485Tx1+	I/O	Serial 1 output (TTL or RS232)[2] Serial 1 output+ (RS485/RS422)[3]
9	G4/Rx0/485Rx1-	I/O	Serial 0 input (TTL or RS232)[2] Serial 1 input- (RS485/RS422)[3]
10	G6/Rx1/485Rx1+	I/O	Serial 1 input (TTL/RS232)[3] Serial 1 input+ (RS485 or RS422)[3]
11	G1/CANL[4]/Rx2[4]	I/O	High level (CAN bus)[4]. Serial 2 input (TTL)[4].
12	G2/CANH[4]/Tx2[4]/ STROBE	I/O	Low level (CAN bus)[4]. Serial 2 output (TTL)[4]. Strobe time sync input.

[1]The System can be powered either by VIN or USB.VCC.

[2]Serial 0 is configured with SMD jumpers for TTL, RS232, or FTDI USB (default, USB.D+ and USB.D-).

[3]Serial 0 is configured with SMT jumpers for TTL, RS232 (default), or RS485/RS422.

[4]Only available with uINS-3.2 and later.

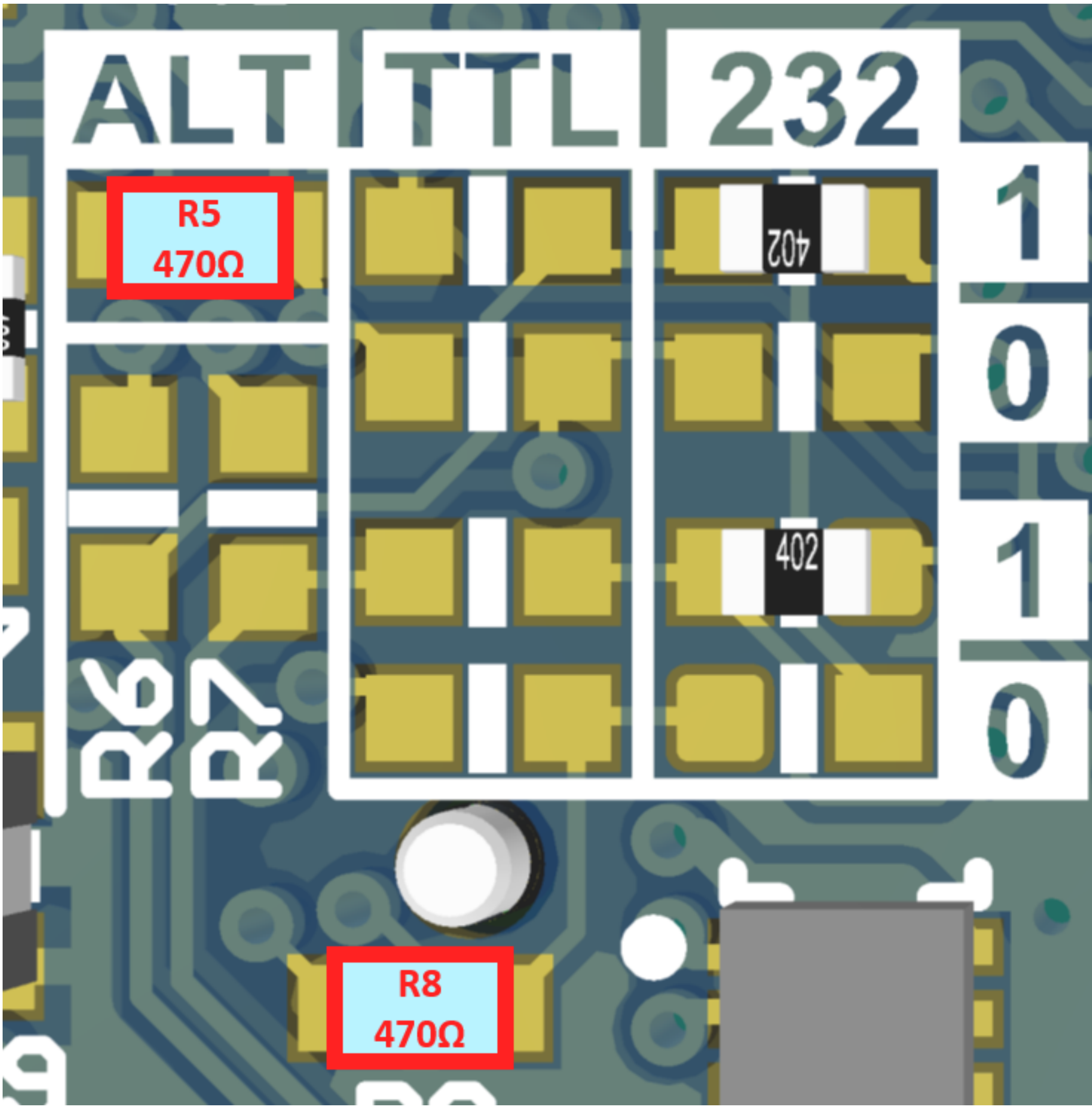
5.3.3 Jumpers

The "MAIN" connector pinout on the Rugged product line can be configured for USB, TTL, RS232, CAN, and RS485 by setting the dip switches for Rugged v1.1 and by setting the onboard PCB surface mount jumpers for Rugged-1.0. Jumper resistors are 470 Ω resistors. All jumpers are 0402 SMD 1/16W (5% or better tolerance) resistors.

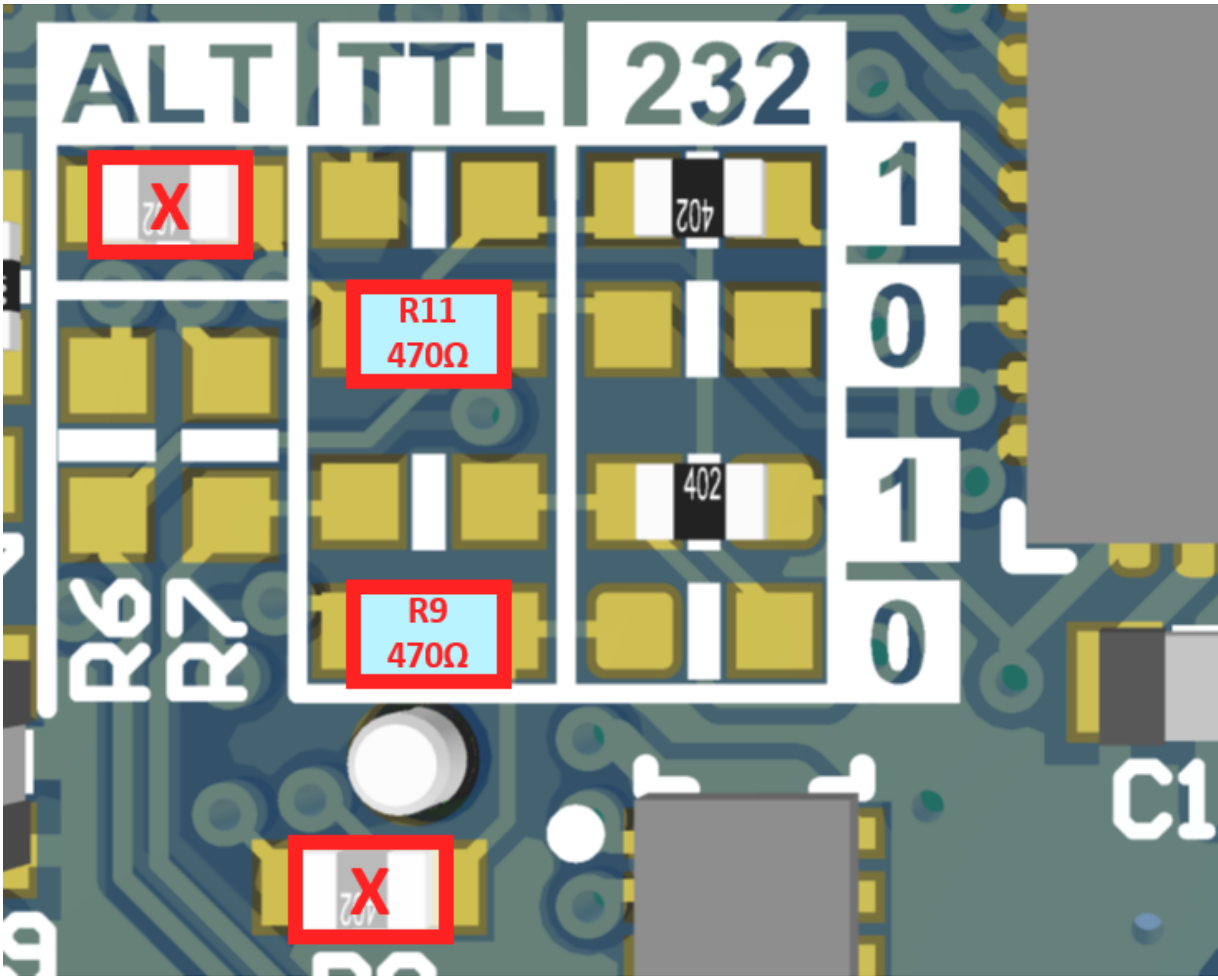
Serial Port 0

To enable FTDI USB on pins 4 and 6, provide system supply voltage input on USB.VCC (pin 3). To disable FTDI USB and use TTL or RS232 on pins 7 and 9, provide system supply voltage input on VIN (pin 2).

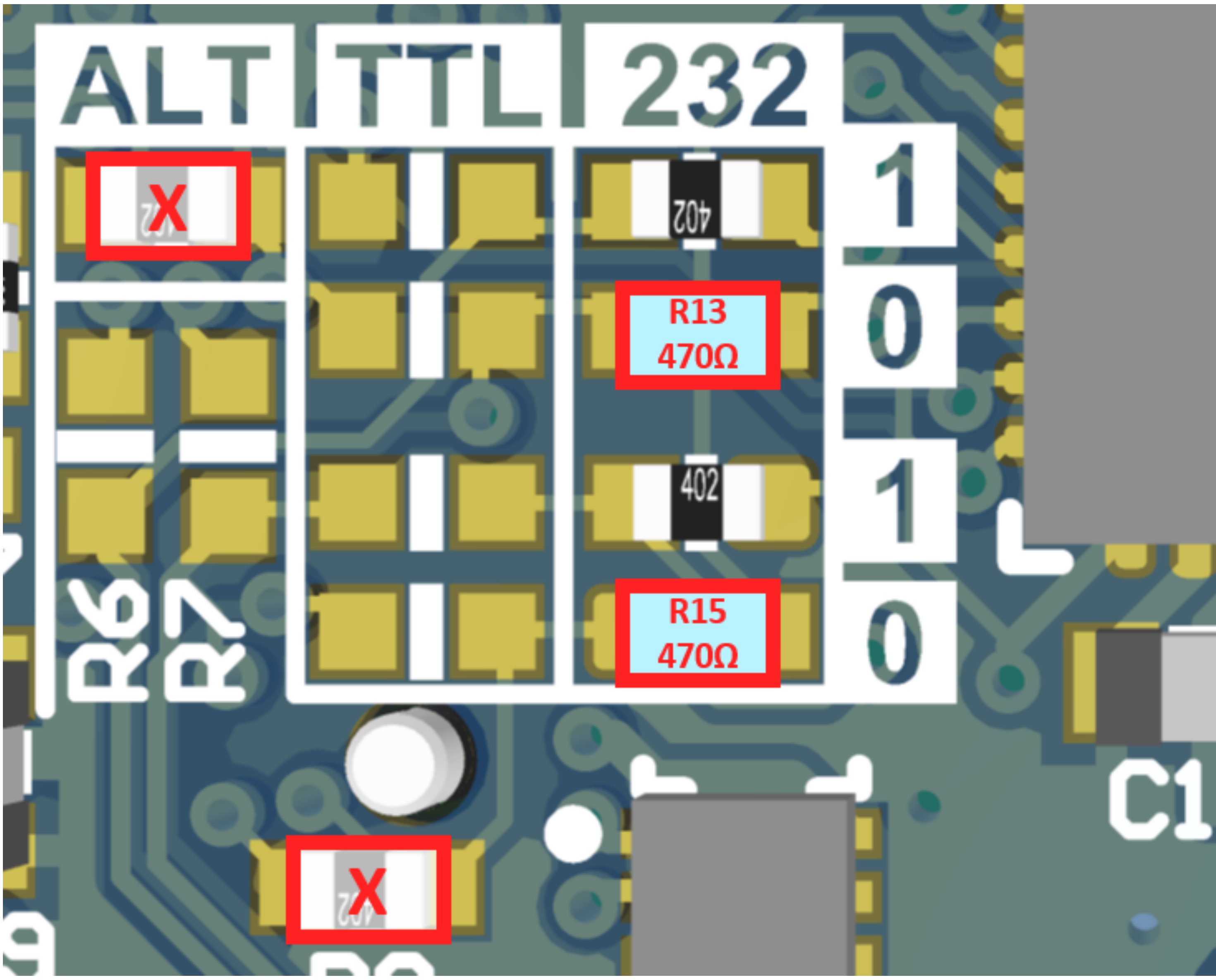
SER0: FTDI USB (DEFAULT)



SER0: TTL

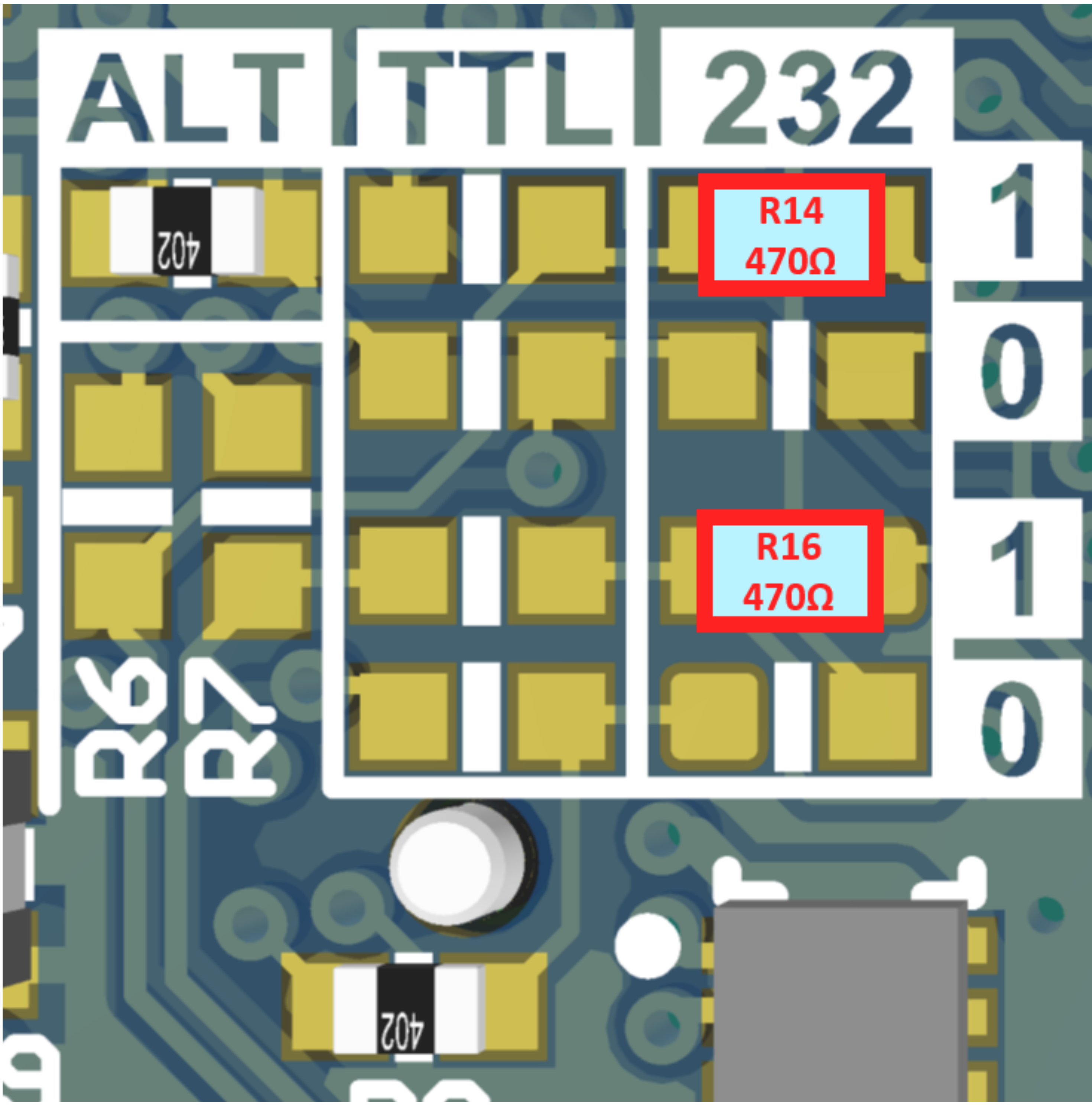


SER0: RS232

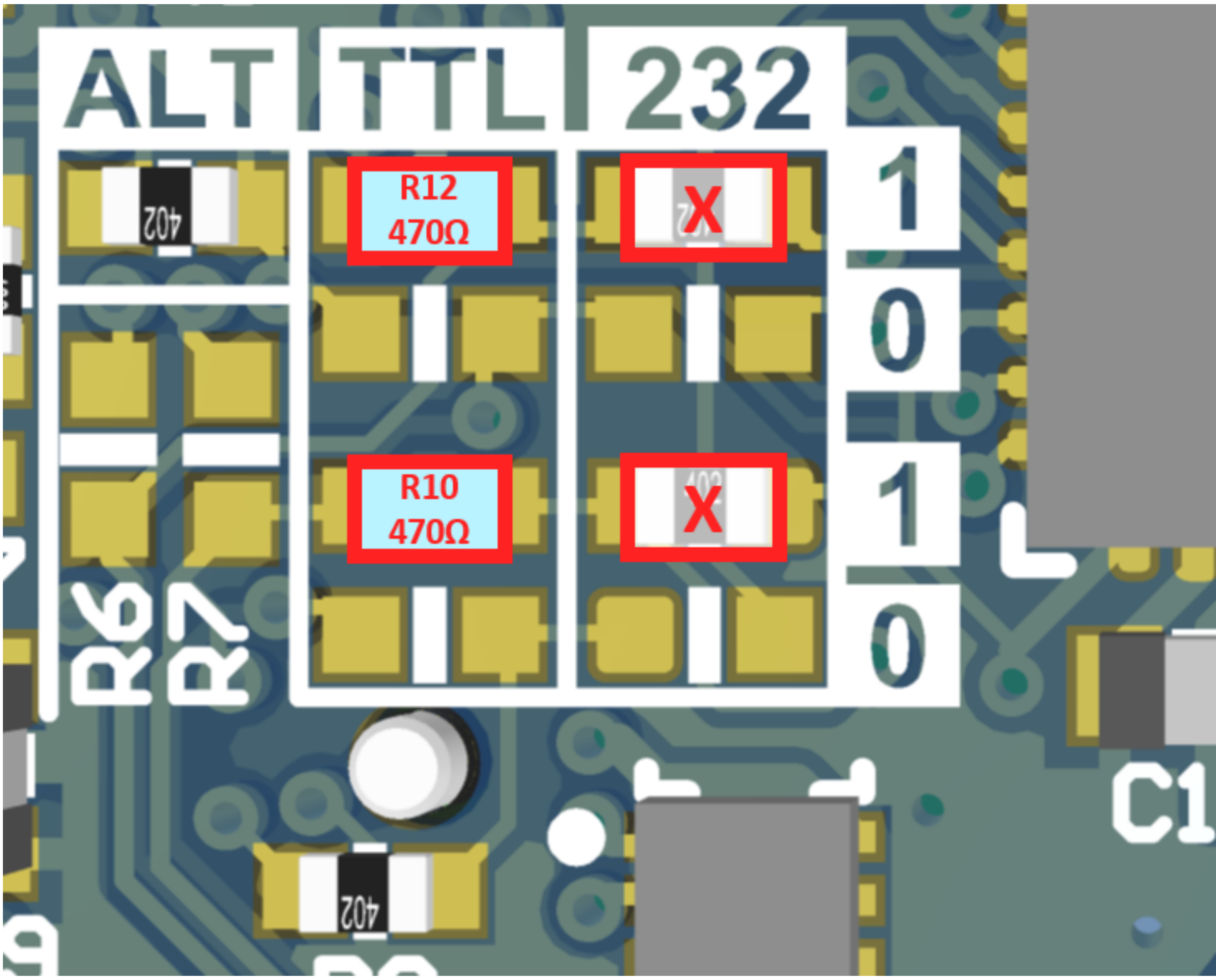


Serial Port 1

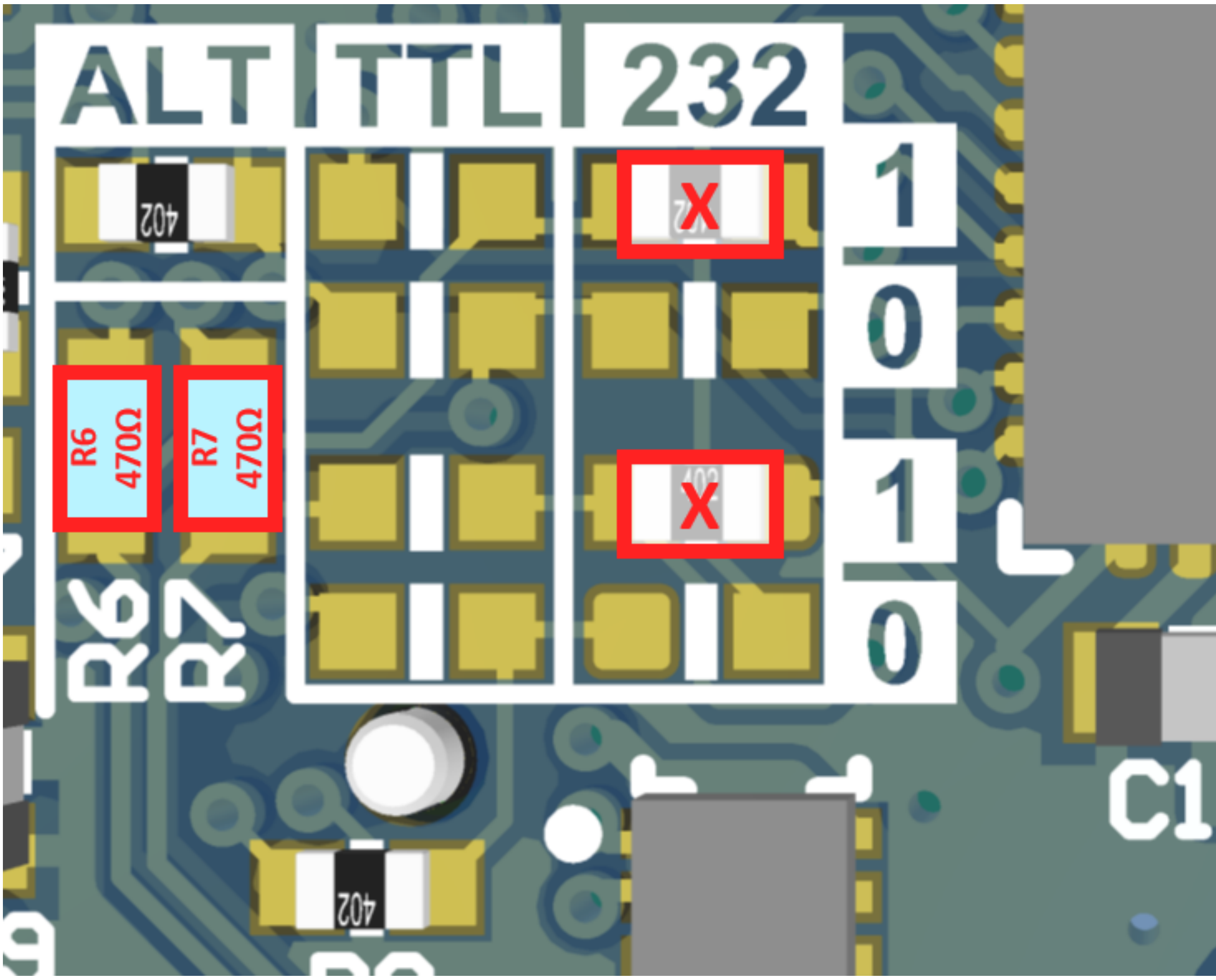
SER1: RS232 (DEFAULT)



SER1: TTL

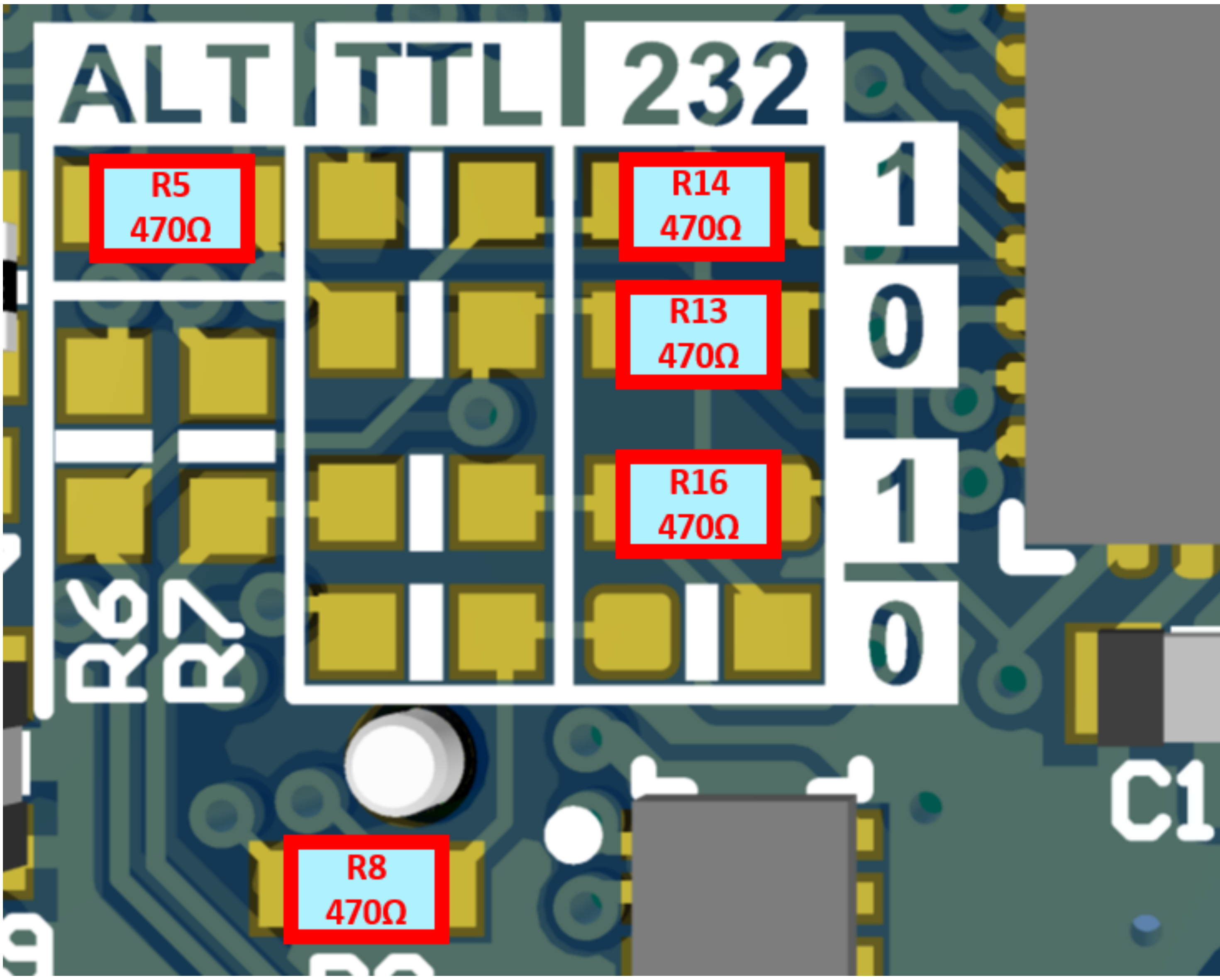


SER1: CAN



SER1: RS485/RS422 AND SER0: FTDI USB

To enable RS485/RS422, jumper R22 must be set in the direction of the "485" silkscreen label. RS485/RS422 signals Tx- and Tx+ are on pins 7 and 8 and Rx- and Rx+ are on pins 9 and 10. In this configuration, serial port 0 can only be accessed through the FTDI USB interface and cannot be accessed through pins 7 and 9.

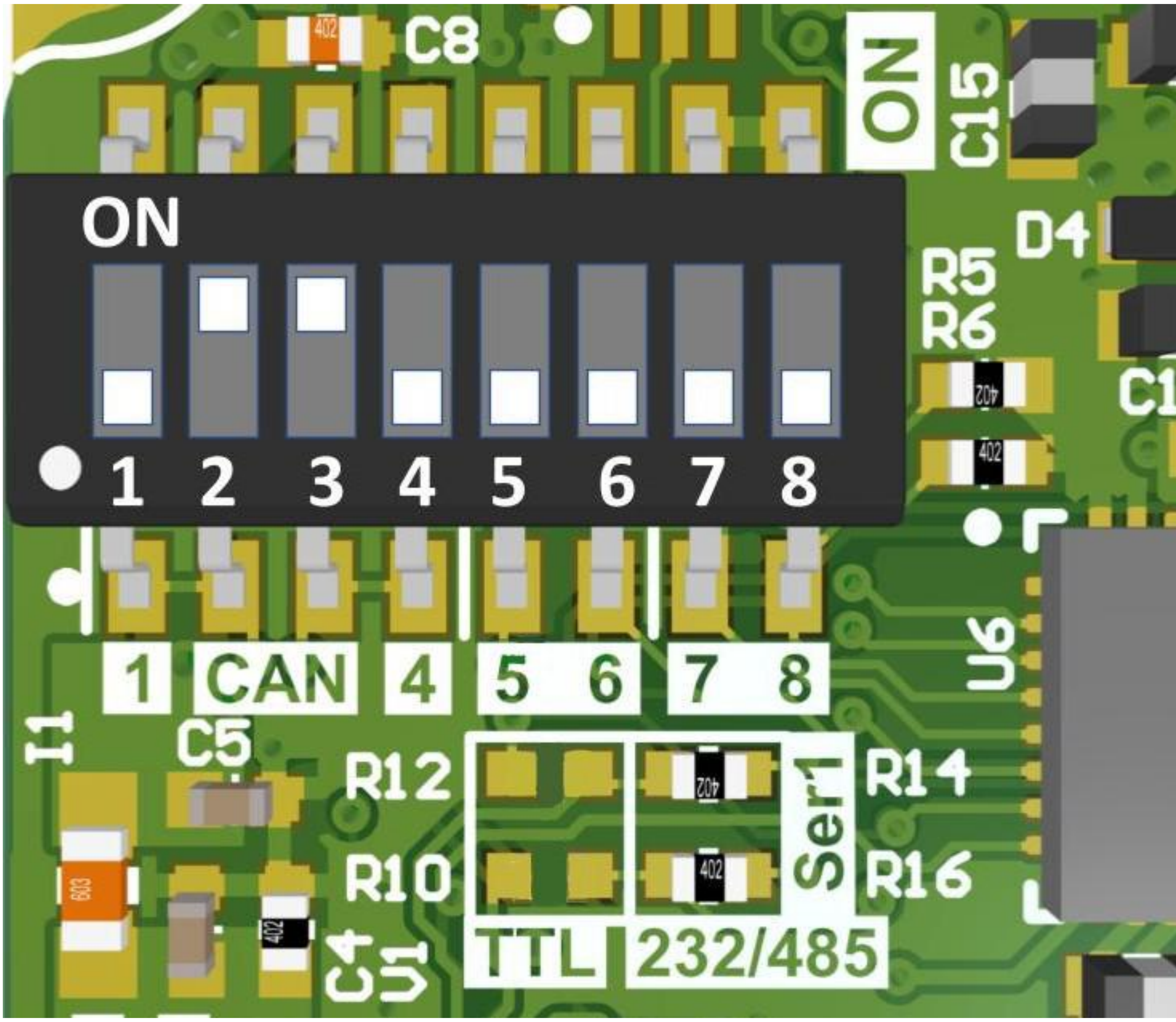


5.3.4 USB Driver

The rugged unit uses the FTDI FT232R USB to UART IC to provide a serial port from the USB connection. Depending on the operating system, it may be necessary to download and install the FTDI device driver for the FT232R to register properly as a serial port.

5.3.5 Rugged v1.1 Dipswitch Config

Version 1.1 has dip switches that replaced the jumpers of v1.0 for common configurations.



Mode	Switches	Jumpers
** CAN	* 2,3 - ON * 1,4 - OFF	n/a
CAN Disabled G2_Strobe on pin 12	1,4 - ON 2,3 - OFF	n/a
Ser0: RS232, Ser1: RS232	* 5,6,7,8 - OFF	* R10/R12 - No Load * R14/R16 - Load
Ser0: RS232, Ser1: TTL	* 5,6,7,8 - OFF	R10/R12 - Load R14/R16 - No Load
Ser0: TTL, Ser1: TTL	5,6,7 - ON 8 - OFF	R10/R12 - Load R14/R16 - No Load
Ser1: RS485, Ser0: Disabled	8 - ON 7 - OFF	* R10/R12 - No Load * R14/R16 - Load

* Factory default setting

**** CAN Bus Operation**

System input voltage monitor surface mount resistors and capacitor (R2, R3, and C1) must be removed for proper high speed CAN bus operation.

r1.1

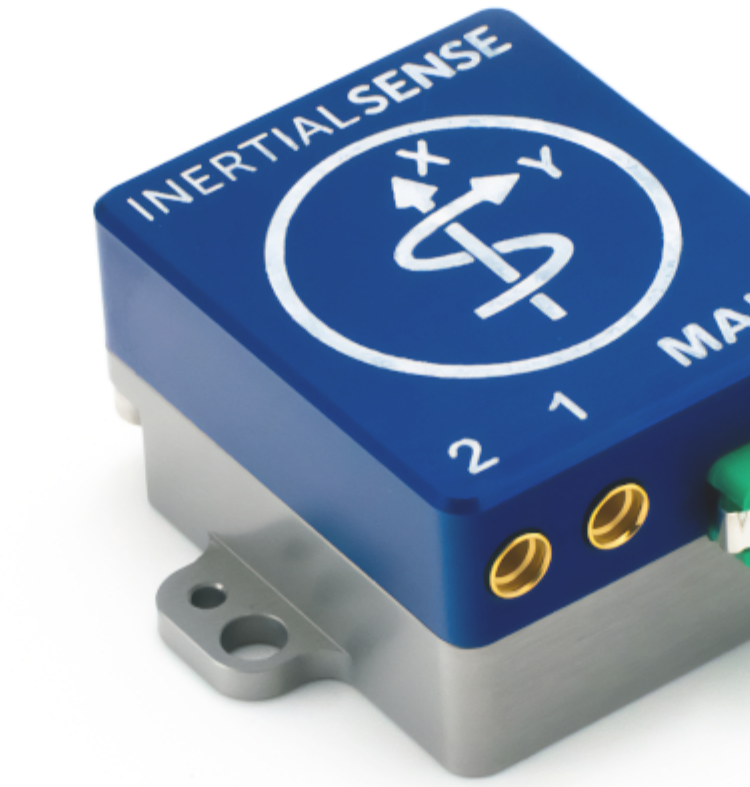
5.3.6 Related Parts

Part	Manufacturer	Manufacturer #	Description
Main Connector	Harwin	G125-FC11205L0-0150F	1.25MM F/F 12POS 26AWG 150MM
GPS antenna SMA adapter	Crystek Corporation	CCSMX-FBM-RG178-6	6" MMCX to SMA GPS antenna adaptor cable.
GPS antenna SMA adapter	Crystek Corporation	CCSMX1-FBM-RG178-6	6" R/A MMCX to SMA GPS antenna adaptor cable.

5.3.7 Using with Inertial Sense Software

Please return to the getting started page to get started programming, updating firmware, viewing data, and logging.

5.4 Hardware Integration: Rugged-2



The Inertial Sense Rugged-2 is a ruggedized carrier board and case for the Inertial Sense µINS, µAHRS, or µIMU module. The Rugged-2 has similar functions compared to the EVB-1, but in a more compact form factor with the following added features:

- Onboard multi-band GNSS receiver(s)
- Dual antenna ports for GPS compassing
- Integrated CAN transceiver

5.4.1 Connecting Your Unit

For the purposes of basic evaluation, the easiest interface available on the rugged is the included USB to Gecko connector cable, included in the evaluation kit. The cable provides power and communications with the installed module via the on-board FTDI chip.

GPS Antenna Ports

If using GPS with the module, connect an appropriate antenna to MMCX port ***1***. If the module is used for RTK compassing, connect a second antenna to MMCX port ***2***. MMCX port ***1*** is for ***GPS1*** and MMXC port ***2*** is ***GPS2***.

5.4.2 Pinout

Pin	Name	I/O	Description
1	GND	PWR	-
2	G5/STROBE	I/O	Strobe time sync input. (Includes 390 ohm series resistor)
3	VIN	PWR	4V-20V supply voltage input
4	USB.D+	I/O	USB Data Positive Line (Serial 0)
5	GPS_PPS	O	GPS PPS time synchronization output pulse (1Hz, 10% duty cycle)
6	USB.D-	I/O	USB Data Negative Line (Serial 0)
7	G3/Tx0/485Tx2-	I/O	Serial 0 output (TTL or RS232) Serial 2 output- (RS485/RS422)
8	G2/Tx2/485Tx2+	I/O	Serial 2 output (TTL or RS232) Serial 2 output+ (RS485/RS422)
9	G4/Rx0/485Rx2-	I/O	Serial 0 input (TTL or RS232) Serial 2 input- (RS485/RS422)
10	G1/Rx2/485Rx2+	I/O	Serial 2 input (TTL/RS232) Serial 2 input+ (RS485 or RS422)
11	G1/CANL[1]/Rx2[1]	I/O	High level (CAN bus). Serial 2 input (TTL).
12	G2/CANH[1]/Tx2[1]/ STROBE	I/O	Low level (CAN bus). Serial 2 output (TTL). Strobe time sync input.

[1]Only available with uINS-3.2 and later.

5.4.3 I/O Configuration

The "MAIN" connector pinout on the Rugged product line can be configured for USB, TTL, RS232, CAN, and RS485 by setting the dipswitches.

5.4.4 Dipswitch Config

The Rugged-2 dip switches are used for setting the following I/O configurations.

Pins	Mode	Switches
-	Tx0, Rx0: onboard GPS2	* 8 - ON
7 9	Tx0 (RS232) Rx0 (RS232)	5,6 - ON 8 - OFF
7 9	Tx0 (TTL) Rx0 (TTL)	6,8 - OFF
8 10	Tx2 (RS232) Rx2 (RS232)	5,6 - ON 7 - OFF
8 10	Tx2 (TTL) Rx2 (TTL)	6,7 - OFF
7 8 9 10	Tx2- (RS485) Tx2+ (RS485) Rx2- (RS485) Rx2+ (RS485)	6 - ON 5,7 - OFF
11 12	CAN-H CAN-L	* 1,2,3,7 - ON
11 12	G1-Rx2, CAN-Rx** G2-Tx2, CAN-Tx**, STROBE	7 - ON 1,2,3 - OFF

* Factory default settings:

- Dual GPS units: All dip switches ON by default (Serial 0 used for onboard GPS2, CAN on pins 11 and 12)
- Single GPS units: Dip switch 8 OFF by default (Serial 0 used for RS232 on pins 7 and 9, CAN on pins 11 and 12)

** CAN transceiver bypassed.

See the Multi-Band GNSS page for configuration information.

5.4.5 **To open the Rugged-2:**

Caution! Use of a grounding strap or other ESD protection device is advised.

1. Completely remove power from the unit.
2. Remove the 3 5/16" screws from the bottom of the Rugged-2.
3. Gently separate the top and bottom halves from each other pealing them apart opening from the side with the green connector.
    - The two halves maybe somewhat adhered due to the thermal pad used in the device. Consistent gentle pressure will separate them.
    - As the device starts to open, do not open the unit past 90 degrees. *If flexed to often and beyond 90 degrees the ribbon will break causing the unit to be damaged beyond repair.*
4. Remove Kapton tape from the DIP Switch. (Save the tape!)
5. Make your adjustment for the desired configuration. (see table above for switch configurations)
    - Use of a small screwdriver or tweezers work well.
    - It should not take much of force to move the desired switch.
6. Replace the Kapton tape.
7. Closed the two halves and install the 3 5/16" screws back in the bottom.

    Done!

5.4.6 Related Parts

Part	Manufacturer	Manufacturer #	Description
Main Connector	Harwin	G125-FC11205L0-0150F	1.25MM F/F 12POS 26AWG 150MM
GPS antenna SMA adapter	Crystek Corporation	CCSMX-FBM-RG178-6	6" MMCX to SMA GPS antenna adaptor cable.
GPS antenna SMA adapter	Crystek Corporation	CCSMX1-FBM-RG178-6	6" R/A MMCX to SMA GPS antenna adaptor cable.

See the Multi-Band GNSS page for GNSS antenna options.

5.4.7 Using with Inertial Sense Software

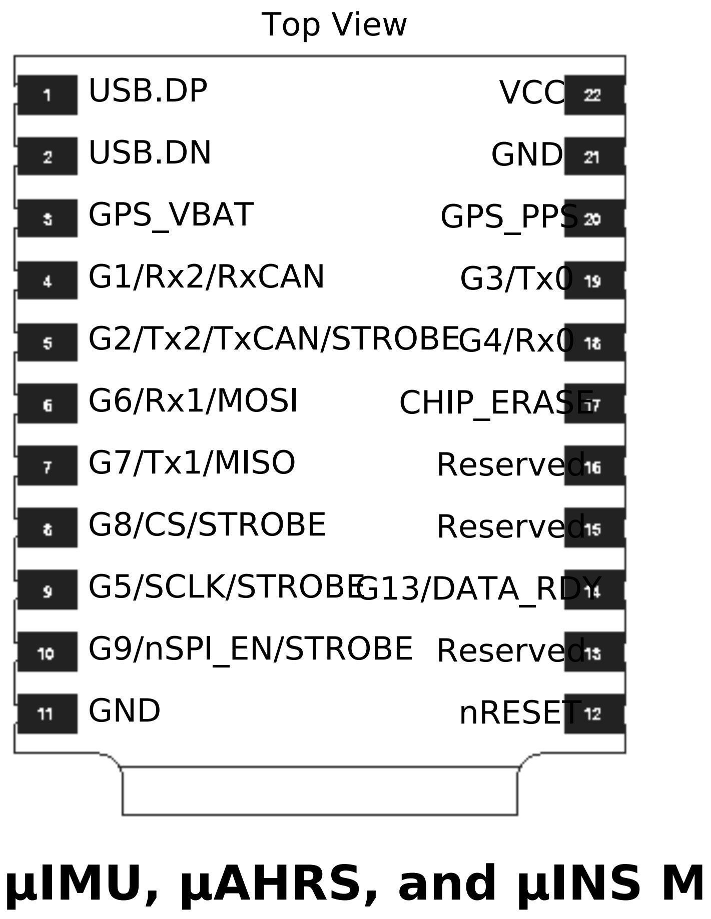
Please return to the getting started page to get started programming, updating firmware, viewing data, and logging.

5.5 Hardware Integration: Module

Warning

Our module must be hand soldered ONLY! Solder reflow may result in damage! See Soldering for details.

5.5.1 Pinout


Top View
1 USB.DP
2 USB.DN
3 GPS_VBAT
4 G1/Rx2/RxCAN
5 G2/Tx2/TxCAN/STROBE
6 G6/Rx1/MOSI
7 G7/Tx1/MISO
8 G8/CS/STROBE
9 G5/SCLK/STROBE
10 G9/nSPI_EN/STROBE
11 GND
12 nRESET
13 Reserved
14 G13/DATA_RDY
15 Reserved
16 Reserved
17 CHIP_ERASE
18 G4/Rx0
19 G3/Tx0
20 GPS_PPS
21 GND
22 VCC
µIMU, µAHRS, and µINS M

Pin	Name	I/O	Description
1	USB_P	I/O	USB Data Positive Line
2	USB_N	I/O	USB Data Negative Line
3	GPS_VBAT	-	GPS backup supply voltage. (1.4V to 3.6V) enables GPS hardware backup mode for hot or warm startup (faster GPS lock acquisition). MUST connect GPS_VBAT to VCC if no backup battery is used.
4	G1/Rx2[*]/RxCAN[*]	I/O	GPIO1. Serial 2 input (TTL). Serial input pin from CAN transceiver[**].
5	G2/Tx2[*]/TxCAN[*]/ STROBE	I/O	GPIO2. Serial 2 output (TTL). Serial output pin to CAN transceiver[**]. Strobe time sync input.
6	G6/Rx1/MOSI	I/O	GPIO6. Serial 1 input (TTL). SPI MOSI
7	G7/Tx1/MISO	I/O	GPIO7. Serial 1 output (TTL). SPI MISO
8	G8/CS/STROBE	I/O	GPIO8. SPI CS. Strobe time sync input.
9	G5/SCLK/ STROBE	I/O	GPIO5. SPI SCLK. Strobe time sync input.
10	G9/nSPI_EN/ STROBE /STROBE_OUT	I/O	GPIO9. Hold LOW during boot to enable SPI on G5-G8. Strobe time sync input or output.
11	GND	-	-
12	nRESET	I	System reset on logic low. May be left unconnected if not used.
13	Reserved	-	
14	Reserved	-	
15	Reserved	-	
16	Reserved	-	
17	Reserved (CE)	-	Leave unconnected. CHIP ERASE used in manufacturing. !!! WARNING !!! Asserting a logic high (+3.3V) will erase all uINS flash memory, including calibration data.
18	G4/Rx0	I/O	GPIO4. Serial 0 input (TTL)
19	G3/Tx0	I/O	GPIO3. Serial 0 output (TTL)
20	GPS_PPS	O	GPS PPS time synchronization output pulse (1Hz, 10% duty cycle)
21	GND	-	-
22	VCC	-	3.3V regulated supply

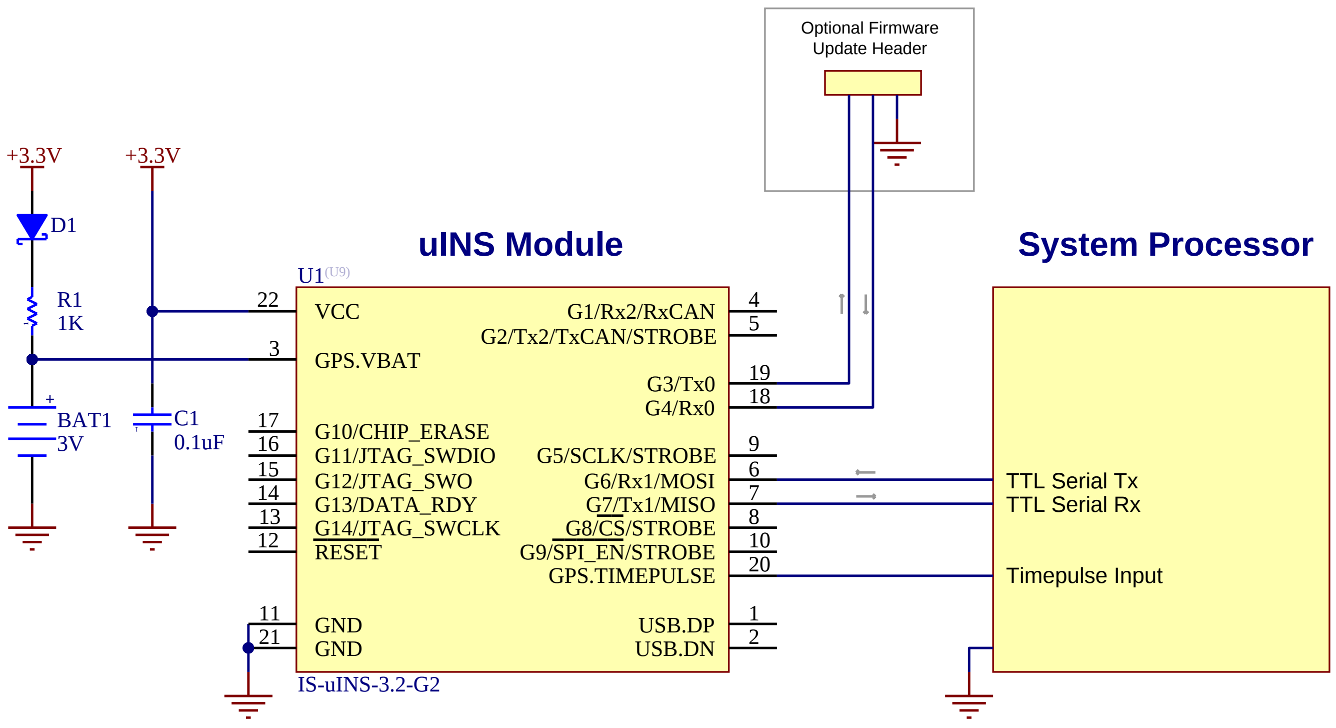
[*]Available on uINS-3.2 and later.

[**]External transceiver required for CAN interface.

5.5.2 Application

Serial Interface

The following schematic demonstrates a typical setup for the µINS module. A rechargeable lithium backup battery enables the GPS to perform a warm or hot start. If no backup battery is connected, GPS.VBAT should be connected to VCC and the module will perform a cold start on power up. If the system processor is not capable of updating the µINS firmware, it is recommended to add a header to an alternate µINS serial port for firmware updates via an external computer. The reset line is not necessary for typical use.

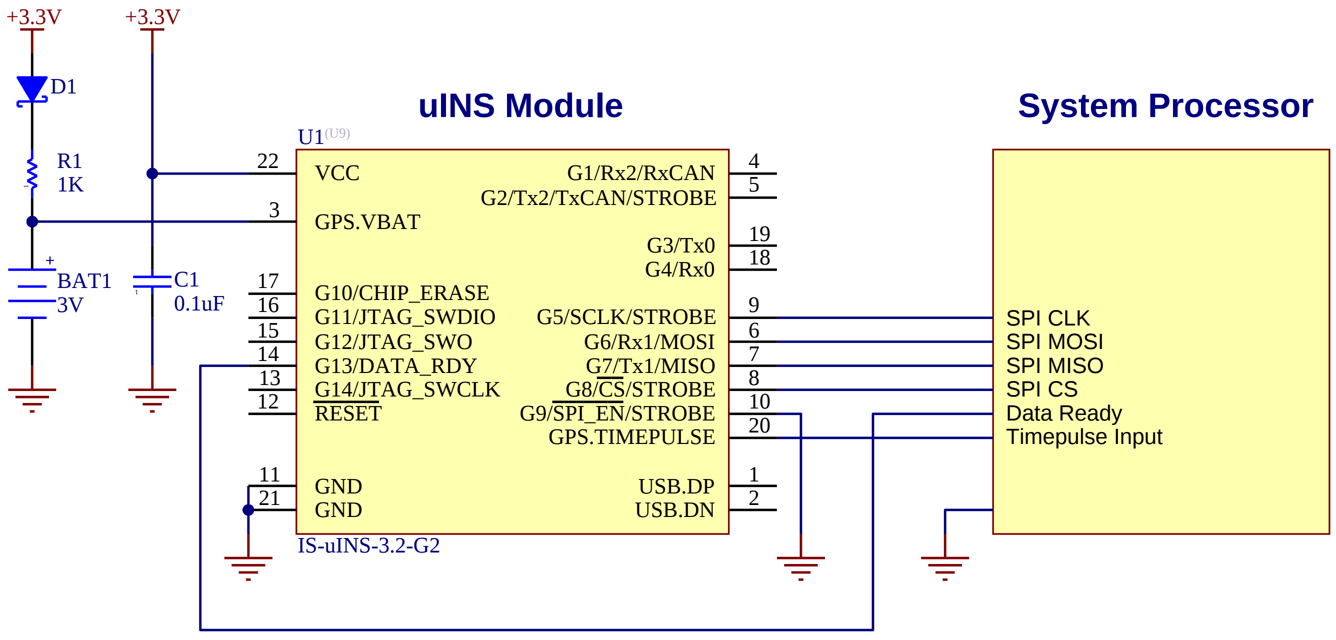


The following are recommended components for the typical application. Equivalent or better components may be used.

Designator	Manufacturer	Manufacturer #	Description
BAT1	Panasonic	ML-614S/FN	BATTERY LITHIMU 3V RECHARGABLE SMD
D1	Panasonic	DB2J31400L	DIODE SCHOTTKY 30V 0.03A SMINI2
R1			RES 1.00K OHM 1/16W 1%
C1			CAP CER .10UF 50V X7R 10%

SPI Interface

The SPI interface is enabled by holding the pin 10 low during boot up.



5.5.3 Soldering

Warning

These parts must be hand soldered ONLY! Solder reflow may result in damage!

The uINS, uAHRS, and uIMU are designed as surface mount components that can be hand soldered onto another circuit board. These parts are not designed to withstand the high temperatures associated with standard solder reflow processes. Solder assembly must be done using a soldering iron.

5.5.4 Hardware Design

Recommend PCB Footprint and Layout

A single ceramic 100nF decoupling capacitor should be placed in close proximity between the Vcc and GND pins. It is recommended that this capacitor be on the same side of the PCB as the µINS and that there not be any vias between the capacitor and the Vcc and GND pins. The default forward direction is indicated in the PCB footprint figure and on the µINS silkscreen as the X axis. The forward direction is reconfigurable in software as necessary.

Download PDF

Design Files

- uINS PCB Design Libraries - Schematic and layout files for printed circuit board designs.
- uINS, uAHRS, uIMU CAD Model - 3D step model of uINS and EVB used for CAD and circuit board designs.

EVB-2 Reference Design

A reference design for implementation of uINS module can be found in the EVB-2 PCB assembly design files available as open source hardware on GitHub.

5.6 Hardware Design Files

The Inertial Sense hardware design files are available on our IS-hdw repository to facilitate product hardware development and integration.

- **PCB Libraries** - Schematic and layout files for printed circuit board designs.
- **Products** - 3D models and resources for the uINS, Rugged, EVB, and products useful for CAD and circuit board designs.

6. IS Software

6.1 CLTool

6.1.1 Overview

The Inertial Sense CLTool is a command line utility that can be used to read and display data, update firmware, and log data from Inertial Sense products. Additionally, CLTool serves as example source code that demonstrates integration of the Inertial Sense SDK into your own source code. The CLTool can be compiled in Linux, Mac, Windows and embedded platforms.

6.1.2 Help Menu

```
Run "cltool -h" to display the help menu.
-----------------------------------------------------------------

DESCRIPTION
    Command line utility for communicating, logging, and updating firmware with Inertial Sense product line.

EXAMPLES
    cltool -c /dev/ttyS2 -did DID_INS_1 DID_GPS1_POS DID_PREINTEGRATED_IMU      # stream DID messages
    cltool -c /dev/ttyS2 -did 4 13 3                # stream same as line above
    cltool -c /dev/ttyS2 -did 3=5                   # stream DID_PREINTEGRATED_IMU at startupNavDtMs x 5
    cltool -c /dev/ttyS2 -presetPPD                 # stream post processing data (PPD) with INS2
    cltool -c /dev/ttyS2 -presetPPD -lon -lts=1     # stream PPD + INS2 data, logging, dir timestamp
    cltool -c /dev/ttyS2 -edit DID_FLASH_CFG        # edit DID_FLASH_CONFIG message
    cltool -c /dev/ttyS2 -baud=115200 -did 5 13=10  # stream at 115200 bps, GPS streamed at 10x startupGPSDtMs
    cltool -c /dev/ttyS2 -rover=RTCM3:192.168.1.100:7777:mount:user:password # Connect to RTK NTRIP base
    cltool -rp logs/20170117_222549                 # replay log files from a folder
    cltool -c /dev/ttyS2 -uf fw/IS_uINS-3.hex -ub fw/SAMx70-Bootloader.bin -uv
                                                    # update application firmware and bootloader
    cltool -c * -baud=921600                        # 921600 bps baudrate on all serial ports

OPTIONS (General)
    -h --help       Display this help menu
    -c COM_PORT     Select the serial port. Set COM_PORT to "*" for all ports and "*4" to use
                    only the first four ports.
    -baud=BAUDRATE  Set serial port baudrate.  Options: 115200, 230400, 460800, 921600 (default)
    -magRecal[n]    Recalibrate magnetometers: 0=multi-axis, 1=single-axis
    -q              Quiet mode, no display
    -reset          Issue software reset.  Use caution.
    -s              Scroll displayed messages to show history
    -stats          Display statistics of data received
    -survey=[s],[d] Survey-in and store base position to refLla: s=[2=3D, 3=float, 4=fix], d=durationSec
    -uf FILEPATH    Update application firmware using .hex file FILEPATH.  Add -baud=115200 for systems w/ baud rate limits.
    -ub FILEPATH    Update bootloader using .bin file FILEPATH if version is old.
    -uv             Run verification after application firmware update.

OPTIONS (Message Streaming)
    -did [DID#<=PERIODMULT> DID#<=PERIODMULT> ...]  Stream 1 or more datasets and display w/ compact view.
    -edit [DID#<=PERIODMULT>]                        Stream and edit 1 dataset.
          Each DID# can be the DID number or name and appended with <=PERIODMULT> to decrease message frequency.
          Message period = source period x PERIODMULT. PERIODMULT is 1 if not specified.
          Common DIDs: DID_INS_1, DID_INS_2, DID_INS_4, DID_PREINTEGRATED_IMU, DID_IMU, DID_GPS1_POS,
          DID_GPS2_RTK_CMP_REL, DID_BAROMETER, DID_MAGNETOMETER, DID_FLASH_CONFIG (see data_sets.h for complete list)
    -dids           Print list of all DID datasets
    -persistent     Save current streams as persistent messages enabled on startup
    -presetPPD      Stream preset post processing datasets (PPD)
    -presetINS2     Stream preset INS2 datasets

OPTIONS (Logging to file, disabled by default)
    -lon            Enable logging
    -lt=TYPE        Log type: dat (default), sdat, kml or csv
    -lp PATH        Log data to path (default: ./IS_logs)
    -lms=PERCENT    Log max space in percent of free space (default: 0.5)
    -lmf=BYTES      Log max file size in bytes (default: 5242880)
    -lts=0          Log sub folder, 0 or blank for none, 1 for timestamp, else use as is
    -r              Replay data log from default path
    -rp PATH        Replay data log from PATH
    -rs=SPEED       Replay data log at x SPEED. SPEED=0 runs as fast as possible.

OPTIONS (Read or write flash configuration from command line)
    -flashCfg       List all uINS "keys" and "values"
   "-flashCfg=[key]=[value]|[key]=[value]"
    -evbFlashCfg    List all EVB "keys" and "values"
   "-evbFlashCfg=[key]=[value]|[key]=[value]"
                    Set key / value pairs in flash config. Surround with "quotes" when using pipe operator.
EXAMPLES
    cltool -c /dev/ttyS2 -flashCfg  # Read from device and print all keys and values
    cltool -c /dev/ttyS2 -flashCfg=insRotation[0]=1.5708|insOffset[1]=1.2
                                  # Set multiple flashCfg values
OPTIONS (RTK Rover / Base)
    -rover=[type]:[IP or URL]:[port]:[mountpoint]:[username]:[password]
```

```
          As a rover (client), receive RTK corrections.  Examples:
              -rover=TCP:RTCM3:192.168.1.100:7777:mountpoint:username:password   (NTRIP)
              -rover=TCP:RTCM3:192.168.1.100:7777
              -rover=TCP:UBLOX:192.168.1.100:7777
              -rover=SERIAL:RTCM3:/dev/ttyS2:57600       (port, baud rate)
      -base=[IP]:[port]   As a Base (sever), send RTK corrections.  Examples:
              -base=TCP::7777                            (IP is optional)
              -base=TCP:192.168.1.43:7777
              -base=SERIAL:/dev/ttyS2:921600
```

6.1.3 Compile & Run (Linux/Mac)

1. You must have cmake installed on your machine. To do this, download the cmake application at https://cmake.org/download/. Then, using the command line, you will need to install cmake with either of the following commands depending on your platform:

   ```
   Mac:
   sudo "/Applications/CMake.app/Contents.bin/cmake-gui" --install

   Linux:
   sudo apt-get install cmake
   ```

2. Create build directory

   ```
   $ cd cltool
   $ mkdir build
   ```

3. Run cmake from within build directory

   ```
   $ cd build
   $ cmake ..
   ```

4. Compile using make

   ```
   $ make
   ```

5. If necessary, add current user to the "dialout" group in order to read and write to the USB serial communication ports:

   ```
   $ sudo usermod -a -G dialout $USER
   $ sudo usermod -a -G plugdev $USER
   (reboot computer)
   ```

6. Run executable

   ```
   $ ./cltool
   ```

6.1.4 Compile & Run (Windows MS Visual Studio)

1. Open Visual Studio solution file (InertialSenseSDK/cltool/VS_project/cltool.sln)
2. Build (F7)
3. Run executable

   ```
   C:\InertialSenseSDK\cltool\VS_project\Release\cltool.exe
   ```

6.1.5 Update the Firmware

In order to update the firmware of your unit on the CLTool, follow these steps:

1. Navigate to the directory with the CLTool executable
2. Set the unit's COM port as an option, e.g. `-c COM15`
3. Specify the FILEPATH to the .hex file, e.g. `-uf foo/bar/IS_uINS-3.hex`
4. Optionally specify the bootloader BLFILEPATH to the .bin file, e.g. `-ub foo/bar/SAMx70-Bootloader.bin`
5. Run the executable

*Note: The firmware can only be updated at the following baud rates: 300000, 921600, 460800, 230400, 115200

6.1.6 Logging

1. Make sure you have followed the steps shown above to compile your CLTool.
2. Change your current directory to `/cpp/SDK/cltool/build`
3. Type `./cltool` into the command line while appending your desired options (All possible options can be accessed from the CLTool's help menu which is accessed by entering `./cltool -h` into the command line)
4. The options you will minimally need to log are these: * `-lt=#` (Defines the log type. Either e.g. -lt=dat or -lt=csv) * `-lp / directory1/directory2/directory3` (Specifies the path into which your files will be placed.) * If you don't include this option, your data will be saved to `/build/IS_logs` if that directory has already been created. * `-lon` (Must be placed after all other options specified)
5. This is an example of what you could use as your logging options:

```
cltool -lon -lts=1 -lp /media/usbdrive/data
```

6.1.7 Command Line Options

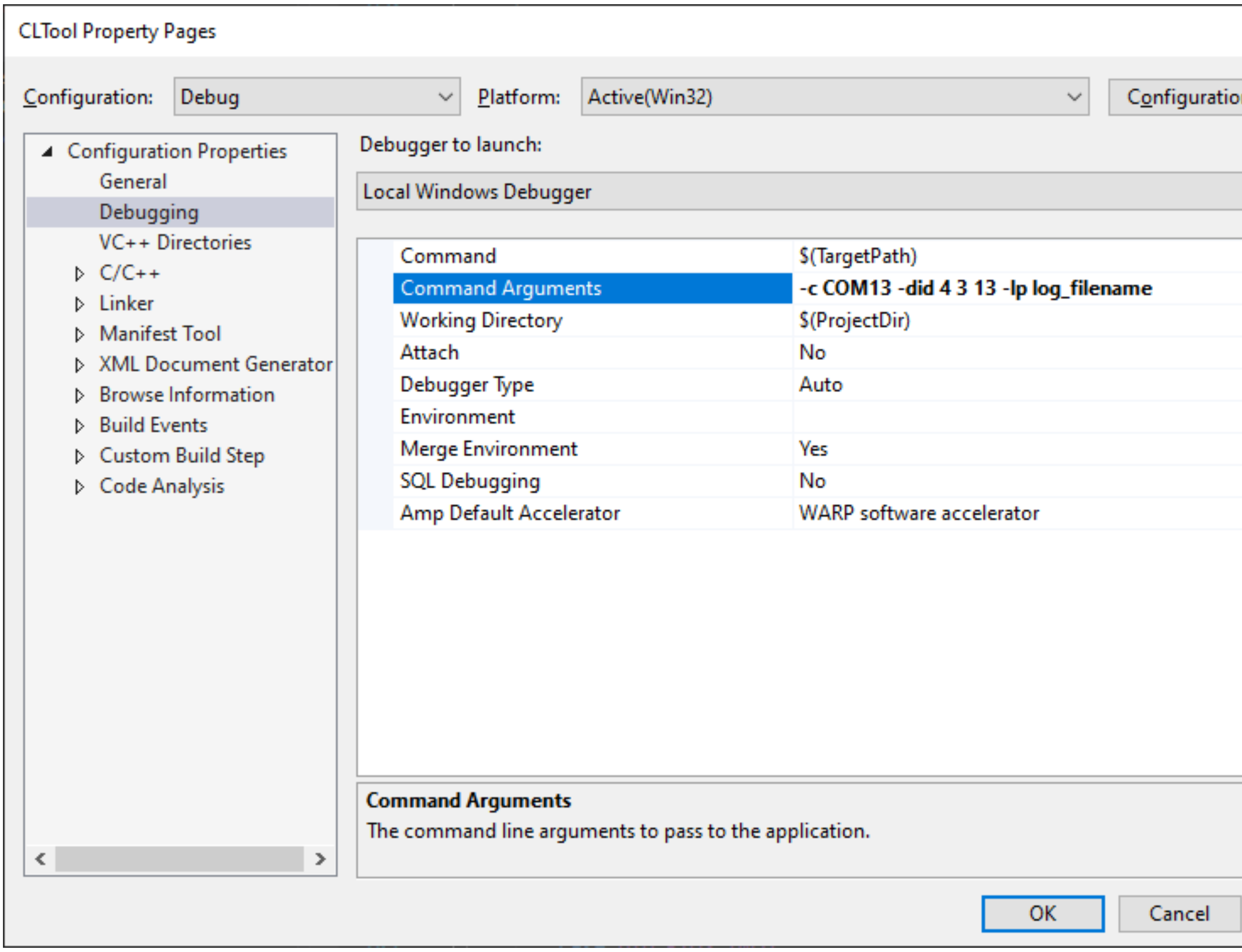
Navigate to the directory `/cpp/SDK/cltool/build` and run the CLTool with the help option, "`-h`"

```
$ ./cltool -h
```

to display the command line options

6.1.8 Command Line Options in MS Visual Studio

When using MS Visual Studio IDE, command line arguments can be supplied by right clicking the project in the solution explorer and then selecting **Configuration Properties -> Debugging -> Command Arguments** (see image below).


CLTool Property Pages

Configuration: Debug   Platform: Active(Win32)   Configuratio

Debugger to launch:

Local Windows Debugger

Command	$(TargetPath)
Command Arguments	**-c COM13 -did 4 3 13 -lp log_filename**
Working Directory	$(ProjectDir)
Attach	No
Debugger Type	Auto
Environment	
Merge Environment	Yes
SQL Debugging	No
Amp Default Accelerator	WARP software accelerator

Command Arguments
The command line arguments to pass to the application.

6.2 EvalTool

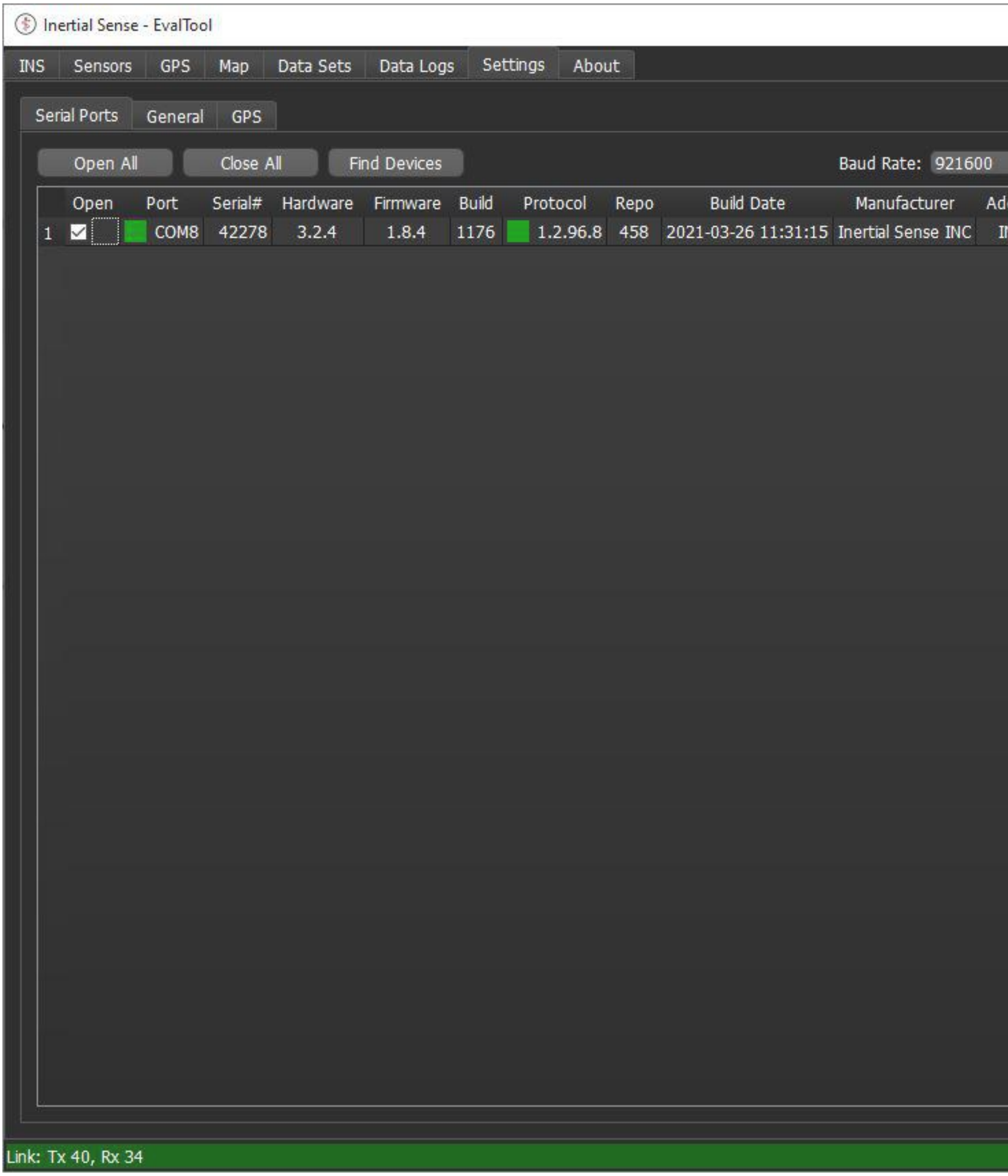
6.2.1 Overview

The EvalTool (Evaluation Tool) is a desktop GUI application that allows you to explore and test functionality of the Inertial Sense products in real-time. It has scrolling plots, 3D model representation, table views of all data, data logger, and firmware updating interface for the uINS, uAHRS, or uIMU. The EvalTool can simultaneously interface with multiple Inertial Sense devices.

6.2.2 Download and Install

The EvalTool Windows desktop app installer (.exe) can be downloaded from the Inertial Sense releases page.

6.2.3 Getting Started



Baud Rate: 921600

	Open	Port	Serial#	Hardware	Firmware	Build	Protocol	Repo	Build Date	Manufacturer	Ad
1	✓ 🟩	COM8	42278	3.2.4	1.8.4	1176	🟩 1.2.96.8	458	2021-03-26 11:31:15	Inertial Sense INC	In

Link: Tx 40, Rx 34

With a device connected to your computer:

1. Connect your INS to your computer using directions in the getting started section of this guide
2. Open the “Settings” tab -> "Serial Ports" tab
3. Click, “Find Devices”
4. Ensure that the checkbox under “Open” is selected for your unit
5. Wait until the box under “Port” turns green and shows a com-port number
6. You can select a specific baud-rate for data transmission on this same tab. This is done in the drop-down menu shown at the top-middle of the page.

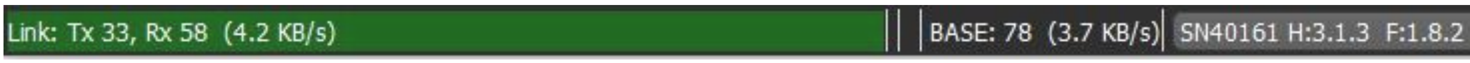
DATA LOGGING STEPS

In order to log data from your INS device, follow the steps listed below:

1. Make sure your unit is connected to the EvalTool and the port is open.
2. Open the “Data Logs” tab.
3. Select “Open Folder” to verify that the log will be saved in your desired location.
4. Change the selection in “Format:” to your desired file format (.dat .csv .sdat etc..)
5. Change the selection in “Data Streams” to the data that you would like collected.
6. Press “Enable” in "Data Log" when you are ready to record the data.
7. The data you are currently recording will be shown in the “Log Summary” sub-tab.
8. When you are finished recording data, press “Disable”. Your data will be saved in the location shown in “Open Folder”.

6.2.4 Info Bar

The Info Bar can be seen from any tab and shows basic connection information for the unit selected.


Link: Tx 33, Rx 58 (4.2 KB/s) BASE: 78 (3.7 KB/s) SN40161 H:3.1.3 F:1.8.2

1. Link Status - Shows Packets being Transmitted and Received. counts to 99 then resets to 0.
2. Error Message - Shows error messages for the selected unit. The kinds of messages vary from data packets lost to system had a reset.
3. RTK Base Messages - The number in this field will increment as your rover unit continues to receive RTK messages from your base station. Use this field as the main signifier that RTK messages are coming through.
4. Currently selected unit - The unit with the serial number shown here will have its live data shown on each tab in EvalTool.

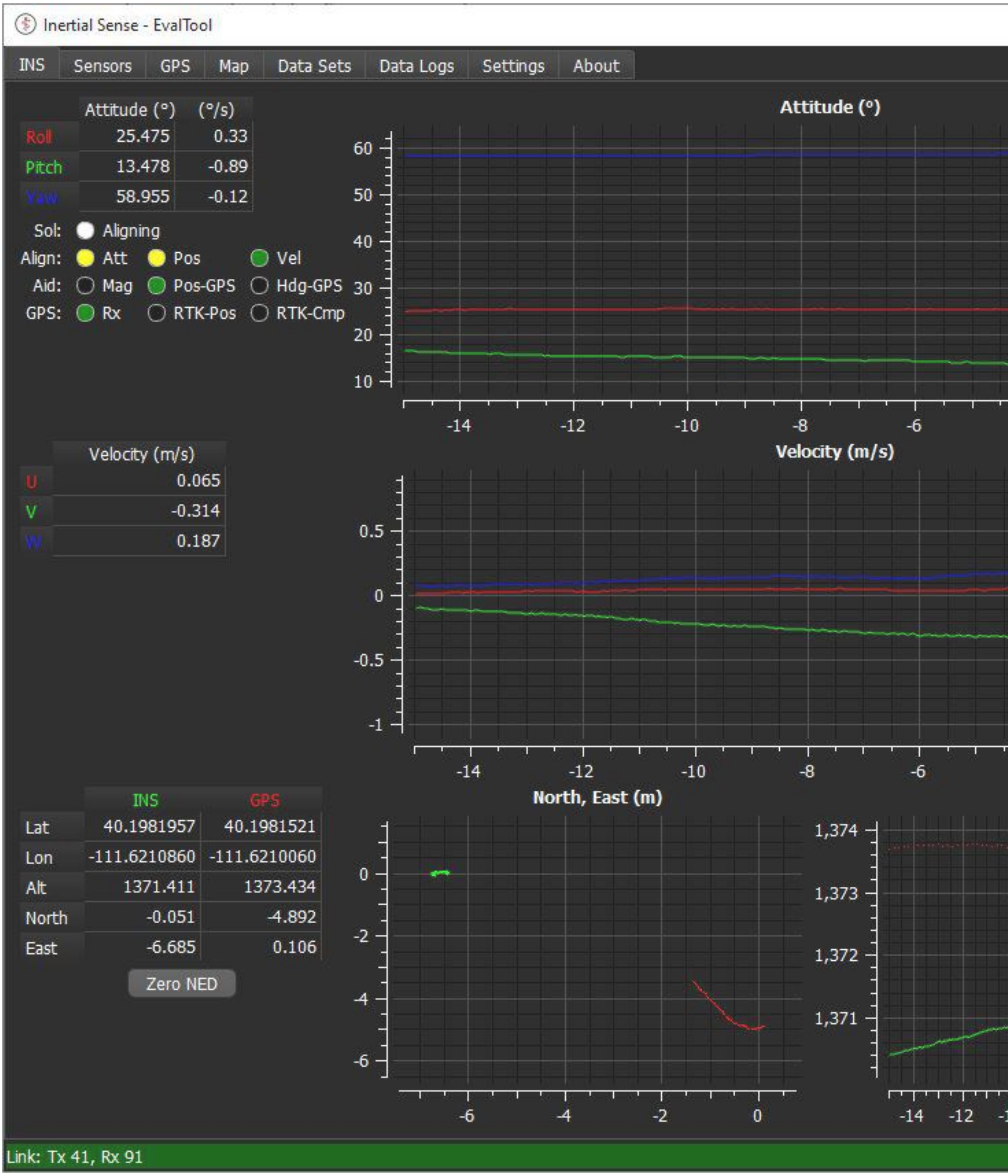
6.2.5 Update Firmware

1. Enter the Settings tab.
2. Open the COM ports of the units you would like to update. 1. If the units don't open up, you may have to change the baud rate.
3. Click "Update Firmware".
4. Choose the firmware file by clicking on the ellipsis (three dots) button next to the file name and navigating to the directory where your .hex file is located. Select it and press "Open".
5. Repeat for the bootloader .bin file or check the bootloader update "Skip" checkbox. The bootloader will only be updated if the unit has an older bootloader than the file provided.
6. Click "Start".
7. Wait for the firmware to fully load or your units will enter into "bootloader mode" and you will have to reload the firmware again.
8. After completion, click "Done".

*Note: The firmware can only be updated at the following baud rates: 300000, 921600, 460800, 230400, 115200.

6.2.6 Tab Descriptions

INS Tab



	Attitude (°)	(°/s)
Roll	25.475	0.33
Pitch	13.478	-0.89
Yaw	58.955	-0.12

Sol: ○ Aligning
Align: ● Att ● Pos ● Vel
Aid: ○ Mag ● Pos-GPS ○ Hdg-GPS
GPS: ● Rx ○ RTK-Pos ○ RTK-Cmp

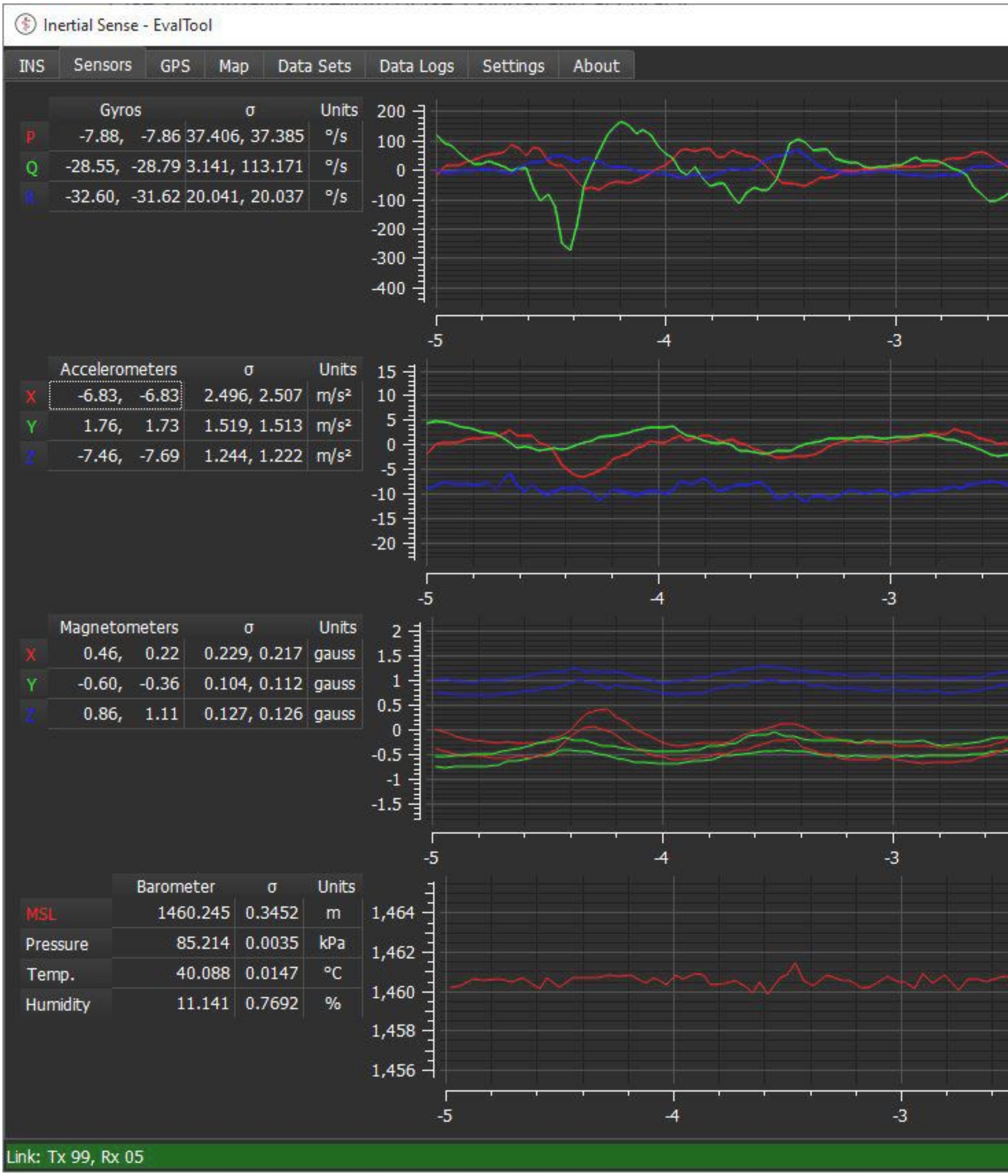
	Velocity (m/s)
U	0.065
V	-0.314
W	0.187

	INS	GPS
Lat	40.1981957	40.1981521
Lon	-111.6210860	-111.6210060
Alt	1371.411	1373.434
North	-0.051	-4.892
East	-6.685	0.106

Link: Tx 41, Rx 91

1. Attitude Plot and Table - shows the Roll, Pitch, and Yaw values of the selected unit. Hover the cursor of the radio buttons to see more descriptions.
2. Velocity Plot and Table - U,V,W velocities.
3. LLA Plot and Table - Tabular values and plot of Latitude, Longitude, and Altitude.
4. Simulation - Real-time, simulated image of the INS orientation.
5. GPS Summary - Strength of GPS signal and accuracy.
6. Mag Recal Button - Allows you to calibrate your units about either a single axis (for heavy, ground based vehicles) or multi-axes.
7. BIT (Built-In Test) Button - Runs a system of checks on your unit.
8. Link Messages - shows the performance information on connected units and displays error messages.

Sensors Tab



	Gyros	σ	Units
P	-7.88, -7.86	37.406, 37.385	°/s
Q	-28.55, -28.79	3.141, 113.171	°/s
R	-32.60, -31.62	20.041, 20.037	°/s

	Accelerometers	σ	Units
X	-6.83, -6.83	2.496, 2.507	m/s²
Y	1.76, 1.73	1.519, 1.513	m/s²
Z	-7.46, -7.69	1.244, 1.222	m/s²

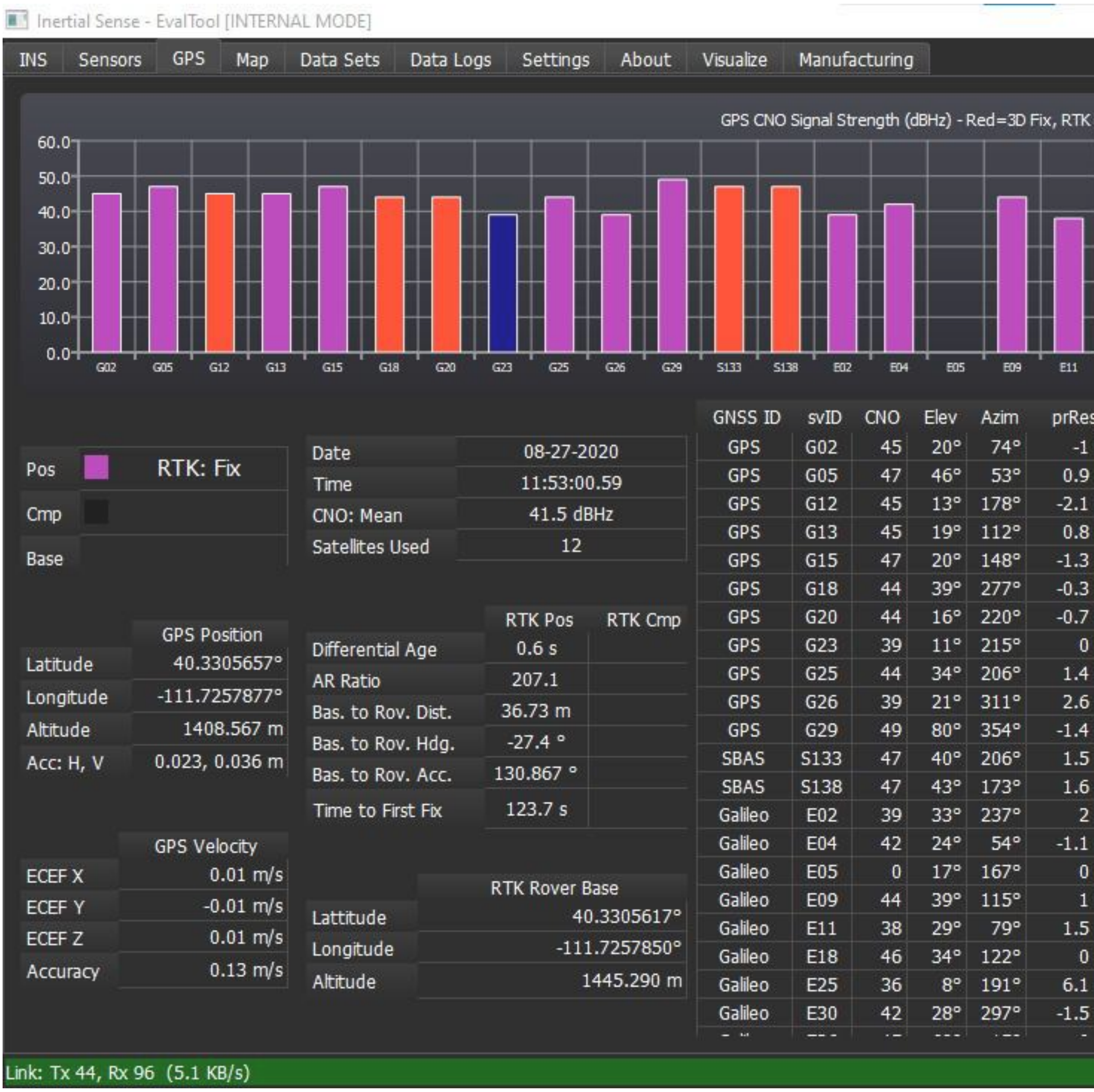
	Magnetometers	σ	Units
X	0.46, 0.22	0.229, 0.217	gauss
Y	-0.60, -0.36	0.104, 0.112	gauss
Z	0.86, 1.11	0.127, 0.126	gauss

	Barometer	σ	Units
MSL	1460.245	0.3452	m
Pressure	85.214	0.0035	kPa
Temp.	40.088	0.0147	°C
Humidity	11.141	0.7692	%

Link: Tx 99, Rx 05

1. Gyros Plot and Table - Gyroscopic data on the selected unit. Includes standard deviation.
2. Accelerometers Plot and Table - Accelerometer data on the selected unit. Includes standard deviation.
3. Magnetometers Plot and Table - Magnetometer data on the selected unit. Includes standard deviation.
4. Barometer Plot and Table - Barometric, temperature, and humidity data on the selected unit. Includes standard deviation.

GPS Tab



Pos	🟪	RTK: Fix
Cmp	⬛	
Base		

Date	08-27-2020
Time	11:53:00.59
CNO: Mean	41.5 dBHz
Satellites Used	12

	GPS Position
Latitude	40.3305657°
Longitude	-111.7257877°
Altitude	1408.567 m
Acc: H, V	0.023, 0.036 m

	RTK Pos	RTK Cmp
Differential Age	0.6 s	
AR Ratio	207.1	
Bas. to Rov. Dist.	36.73 m	
Bas. to Rov. Hdg.	-27.4 °	
Bas. to Rov. Acc.	130.867 °	
Time to First Fix	123.7 s	

	GPS Velocity
ECEF X	0.01 m/s
ECEF Y	-0.01 m/s
ECEF Z	0.01 m/s
Accuracy	0.13 m/s

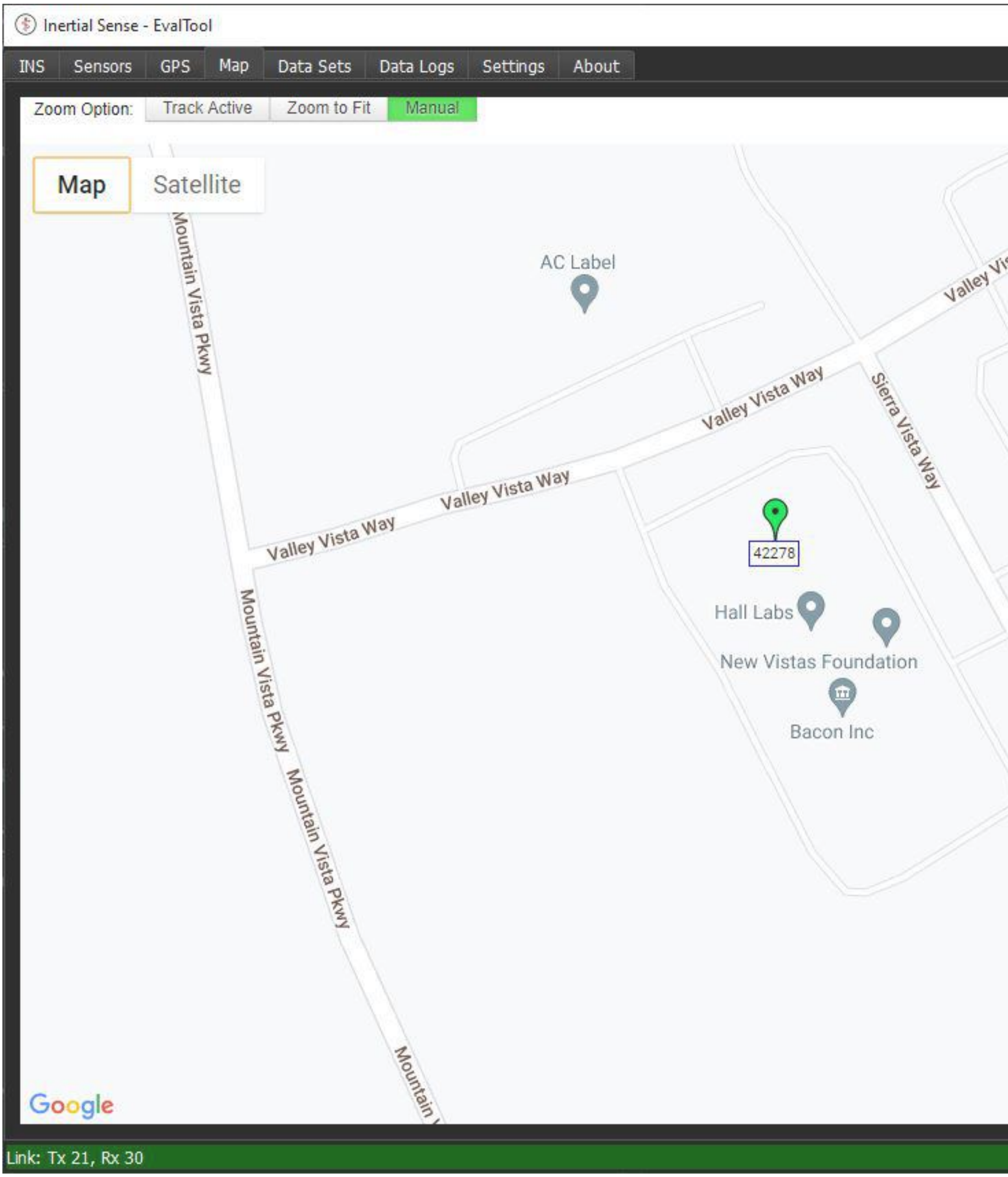
	RTK Rover Base
Lattitude	40.3305617°
Longitude	-111.7257850°
Altitude	1445.290 m

GNSS ID	svID	CNO	Elev	Azim	prRes
GPS	G02	45	20°	74°	-1
GPS	G05	47	46°	53°	0.9
GPS	G12	45	13°	178°	-2.1
GPS	G13	45	19°	112°	0.8
GPS	G15	47	20°	148°	-1.3
GPS	G18	44	39°	277°	-0.3
GPS	G20	44	16°	220°	-0.7
GPS	G23	39	11°	215°	0
GPS	G25	44	34°	206°	1.4
GPS	G26	39	21°	311°	2.6
GPS	G29	49	80°	354°	-1.4
SBAS	S133	47	40°	206°	1.5
SBAS	S138	47	43°	173°	1.6
Galileo	E02	39	33°	237°	2
Galileo	E04	42	24°	54°	-1.1
Galileo	E05	0	17°	167°	0
Galileo	E09	44	39°	115°	1
Galileo	E11	38	29°	79°	1.5
Galileo	E18	46	34°	122°	0
Galileo	E25	36	8°	191°	6.1
Galileo	E30	42	28°	297°	-1.5

Link: Tx 44, Rx 96 (5.1 KB/s)

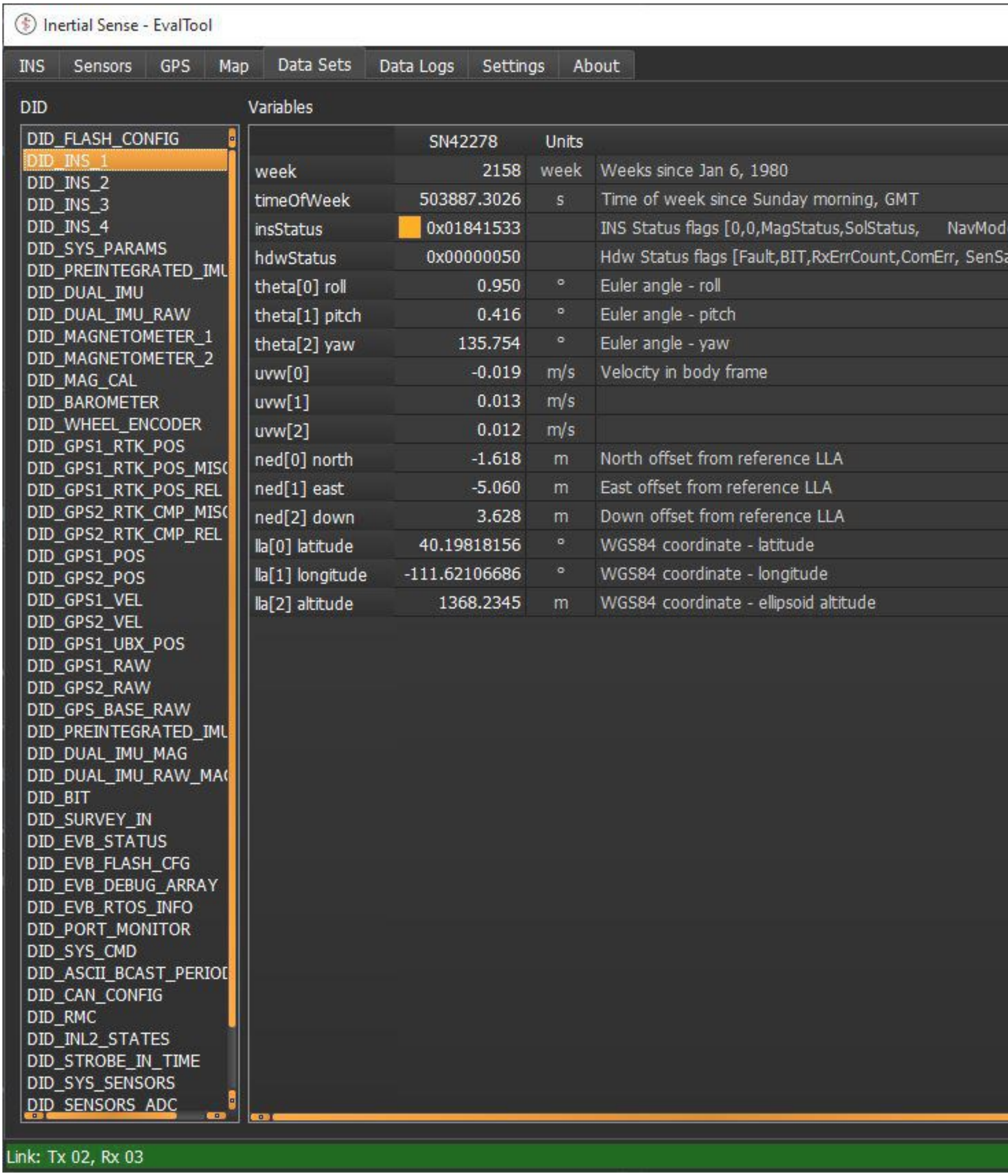
1. GPS CNO Signal Strength - Bar graphs of each satellite being used in your solution and its strength in dBHz(CNO).
2. Position Accuracy Plot and Table - RTK mode and status. Includes number of satellites used in the RTK solution (max and mean).
3. Satellites Used Table - The GNSS ID for each satellite seen by your unit and the subsequent connection details.

Map Tab


Inertial Sense - EvalTool
INS
Sensors
GPS
Map
Data Sets
Data Logs
Settings
About
Zoom Option:
Track Active
Zoom to Fit
Manual
Map
Satellite
AC Label
Mountain Vista Pkwy
Valley Vista Way
Sierra Vista Way
42278
Hall Labs
New Vistas Foundation
Bacon Inc
Google
Link: Tx 21, Rx 30

1. Track Active - Tracks all units on window view.
2. Zoom to Fit - Zooms your window view around each unit being used.
3. Manual - Requires manual movement of the window view.
4. Location of Units - GPS location of each of your units. Shows RTK, GPS ublox, and INS solution.

Data Sets Tab



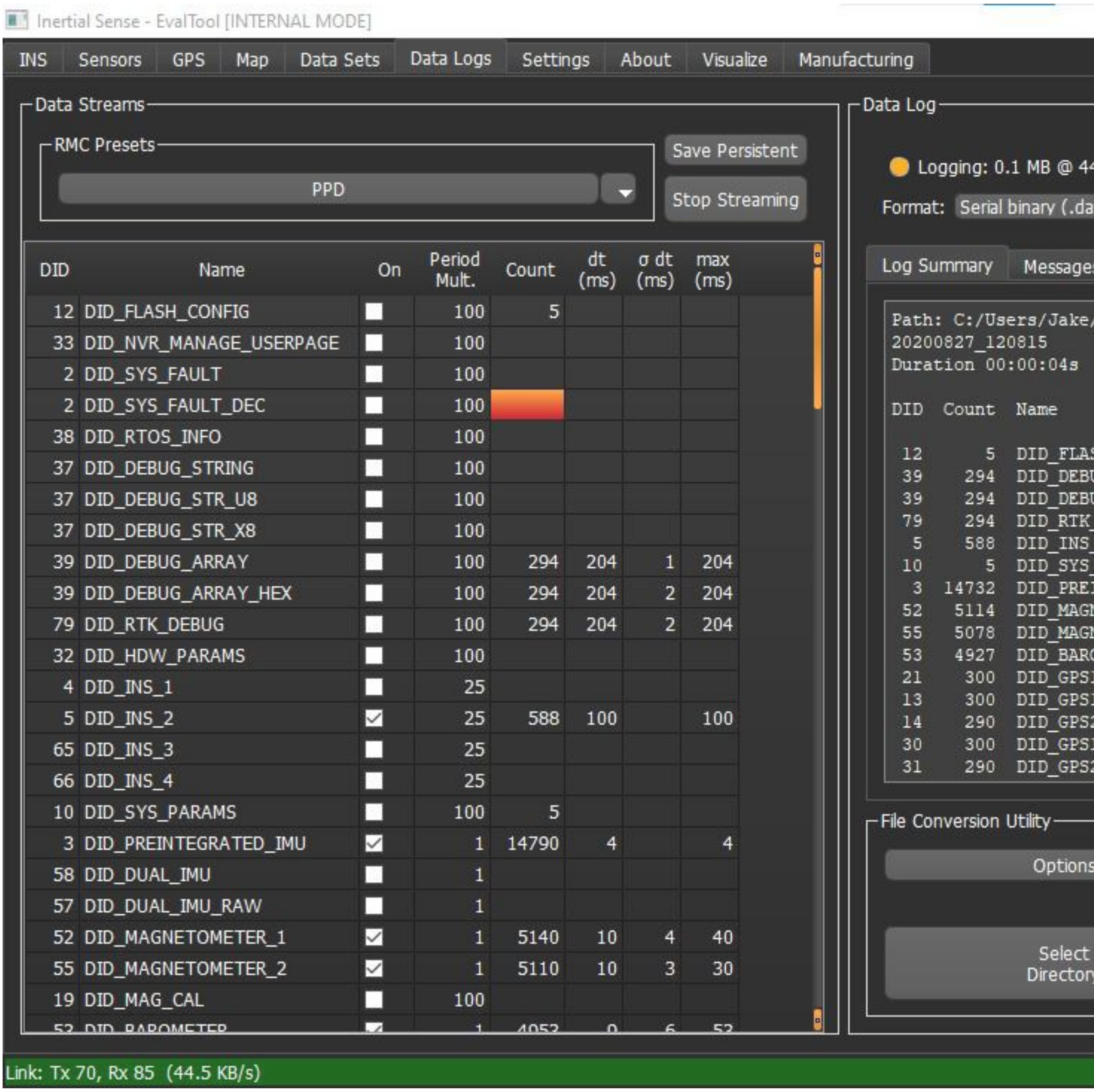
DID_FLASH_CONFIG
DID_INS_1
DID_INS_2
DID_INS_3
DID_INS_4
DID_SYS_PARAMS
DID_PREINTEGRATED_IMU
DID_DUAL_IMU
DID_DUAL_IMU_RAW
DID_MAGNETOMETER_1
DID_MAGNETOMETER_2
DID_MAG_CAL
DID_BAROMETER
DID_WHEEL_ENCODER
DID_GPS1_RTK_POS
DID_GPS1_RTK_POS_MISC
DID_GPS1_RTK_POS_REL
DID_GPS2_RTK_CMP_MISC
DID_GPS2_RTK_CMP_REL
DID_GPS1_POS
DID_GPS2_POS
DID_GPS1_VEL
DID_GPS2_VEL
DID_GPS1_UBX_POS
DID_GPS1_RAW
DID_GPS2_RAW
DID_GPS_BASE_RAW
DID_PREINTEGRATED_IMU
DID_DUAL_IMU_MAG
DID_DUAL_IMU_RAW_MAG
DID_BIT
DID_SURVEY_IN
DID_EVB_STATUS
DID_EVB_FLASH_CFG
DID_EVB_DEBUG_ARRAY
DID_EVB_RTOS_INFO
DID_PORT_MONITOR
DID_SYS_CMD
DID_ASCII_BCAST_PERIOD
DID_CAN_CONFIG
DID_RMC
DID_INL2_STATES
DID_STROBE_IN_TIME
DID_SYS_SENSORS
DID_SENSORS_ADC

	SN42278	Units	
week	2158	week	Weeks since Jan 6, 1980
timeOfWeek	503887.3026	s	Time of week since Sunday morning, GMT
insStatus	🟧 0x01841533		INS Status flags [0,0,MagStatus,SolStatus, NavMod
hdwStatus	0x00000050		Hdw Status flags [Fault,BIT,RxErrCount,ComErr, SenSa
theta[0] roll	0.950	°	Euler angle - roll
theta[1] pitch	0.416	°	Euler angle - pitch
theta[2] yaw	135.754	°	Euler angle - yaw
uvw[0]	-0.019	m/s	Velocity in body frame
uvw[1]	0.013	m/s	
uvw[2]	0.012	m/s	
ned[0] north	-1.618	m	North offset from reference LLA
ned[1] east	-5.060	m	East offset from reference LLA
ned[2] down	3.628	m	Down offset from reference LLA
lla[0] latitude	40.19818156	°	WGS84 coordinate - latitude
lla[1] longitude	-111.62106686	°	WGS84 coordinate - longitude
lla[2] altitude	1368.2345	m	WGS84 coordinate - ellipsoid altitude

Link: Tx 02, Rx 03

1. List of DIDs (Data IDentifiers) - The data identifiers that you might need to view for measurements. See the User Manual (Binary Protocol Data Sets) for a detailed description of frequently used DIDs.
2. List of Variables within DIDs - shows what is recorded in each DID in real-time.

Data Logs



DID	Name	On	Period Mult.	Count	dt (ms)	σ dt (ms)	max (ms)
12	DID_FLASH_CONFIG	☐	100	5			
33	DID_NVR_MANAGE_USERPAGE	☐	100				
2	DID_SYS_FAULT	☐	100				
2	DID_SYS_FAULT_DEC	☐	100	🟥			
38	DID_RTOS_INFO	☐	100				
37	DID_DEBUG_STRING	☐	100				
37	DID_DEBUG_STR_U8	☐	100				
37	DID_DEBUG_STR_X8	☐	100				
39	DID_DEBUG_ARRAY	☐	100	294	204	1	204
39	DID_DEBUG_ARRAY_HEX	☐	100	294	204	2	204
79	DID_RTK_DEBUG	☐	100	294	204	2	204
32	DID_HDW_PARAMS	☐	100				
4	DID_INS_1	☐	25				
5	DID_INS_2	✓	25	588	100		100
65	DID_INS_3	☐	25				
66	DID_INS_4	☐	25				
10	DID_SYS_PARAMS	☐	100	5			
3	DID_PREINTEGRATED_IMU	✓	1	14790	4		4
58	DID_DUAL_IMU	☐	1				
57	DID_DUAL_IMU_RAW	☐	1				
52	DID_MAGNETOMETER_1	✓	1	5140	10	4	40
55	DID_MAGNETOMETER_2	✓	1	5110	10	3	30
19	DID_MAG_CAL	☐	100				
53	DID_BAROMETER	✓	1	4953	9	6	53

```
Path: C:/Users/Jake/
20200827_120815
Duration 00:00:04s

DID  Count  Name

 12      5  DID_FLAS
 39    294  DID_DEBU
 39    294  DID_DEBU
 79    294  DID_RTK_
  5    588  DID_INS_
 10      5  DID_SYS_
  3  14732  DID_PREI
 52   5114  DID_MAGN
 55   5078  DID_MAGN
 53   4927  DID_BARO
 21    300  DID_GPS1
 13    300  DID_GPS1
 14    290  DID_GPS2
 30    300  DID_GPS1
 31    290  DID_GPS2
```

Link: Tx 70, Rx 85 (44.5 KB/s)

DATA STREAMS

This area allow users to enable streaming of various DIDs.

1. RMC Presets Button - Enable a group of data sets. PPD (post process data) is the preferred preset for post processing and debug analysis.
2. Save Persistent Button - Save currently enabled data streams to automatically begin streaming after system restart. To clear persistent streams, first stop streaming and then click Save Persistent.
3. Stop Streaming - Stops all data streams. Any streams previously saved as persistent will begin streaming at startup.

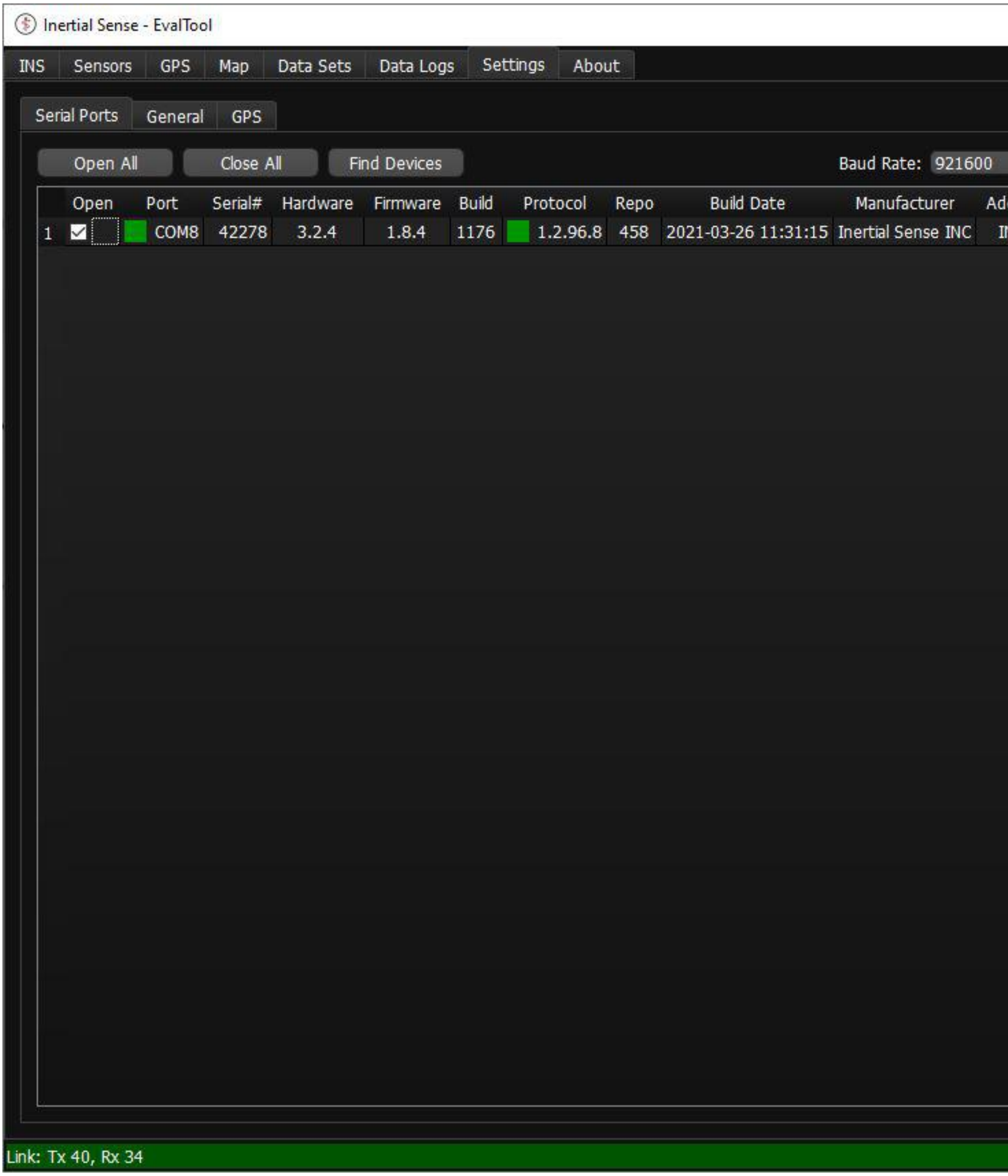
DATA LOG

1. Enable/Disable Button - Starts/stops a log of all currently streaming data and saves it to a sub-folder with the current time-stamp within your "Logs" folder.
2. Open Folder Button - Opens the "Logs" folder where your previous logs are saved.
3. Format Dropdown - Select the file output type of the data log , such as .dat, .sdat, .csv, or .kml.
4. Summary Window - Shows the log directly path, the elapsed time the data log has been running, the total size of the log file, and a list currently recording DIDs with corresponding dt (time between measurements).
5. File Conversion Utility - Enables you to convert the data log file type in a specified directory. (e.g. .dat to .csv)

Settings Tab

The Settings tab has 3 sub tabs and they are as follows:

Settings - Serial Ports Tab



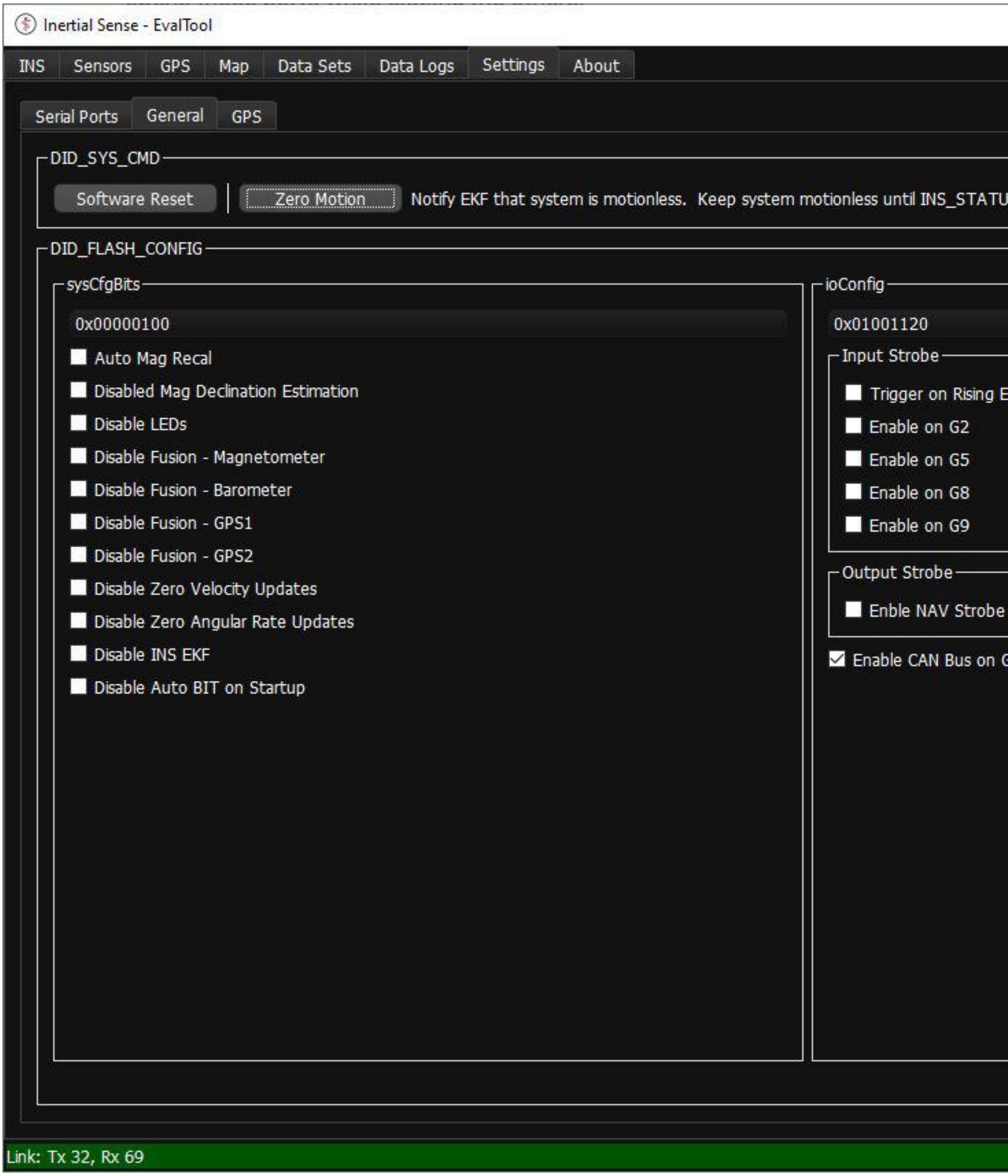
Baud Rate: 921600

	Open		Port	Serial#	Hardware	Firmware	Build	Protocol	Repo	Build Date	Manufacturer	Ad
1	✓	🟩	COM8	42278	3.2.4	1.8.4	1176	🟩 1.2.96.8	458	2021-03-26 11:31:15	Inertial Sense INC	Ir

Link: Tx 40, Rx 34

1. Open All - Opens all of the ports shown.
2. Close All - Closes all of the ports shown.
3. Find Devices - Determines which peripherals into your computer are Inertial Sense units, and opens those ports while closing the others.
4. Baud Rate - The rate at which data will be communicated over your data channel.
5. Update Firmware - Allows you to update your unit's firmware when an update is released from Inertial Sense.
6. Port Status - Shows a list of all connected comports and basic information for each of them. Clicking the check box opens the port.

Settings - General Tab



DID_SYS_CMD

Notify EKF that system is motionless. Keep system motionless until INS_STATU

DID_FLASH_CONFIG

sysCfgBits

0x00000100

- [ ] Auto Mag Recal
- [ ] Disabled Mag Declination Estimation
- [ ] Disable LEDs
- [ ] Disable Fusion - Magnetometer
- [ ] Disable Fusion - Barometer
- [ ] Disable Fusion - GPS1
- [ ] Disable Fusion - GPS2
- [ ] Disable Zero Velocity Updates
- [ ] Disable Zero Angular Rate Updates
- [ ] Disable INS EKF
- [ ] Disable Auto BIT on Startup

ioConfig

0x01001120

Input Strobe

- [ ] Trigger on Rising E
- [ ] Enable on G2
- [ ] Enable on G5
- [ ] Enable on G8
- [ ] Enable on G9

Output Strobe

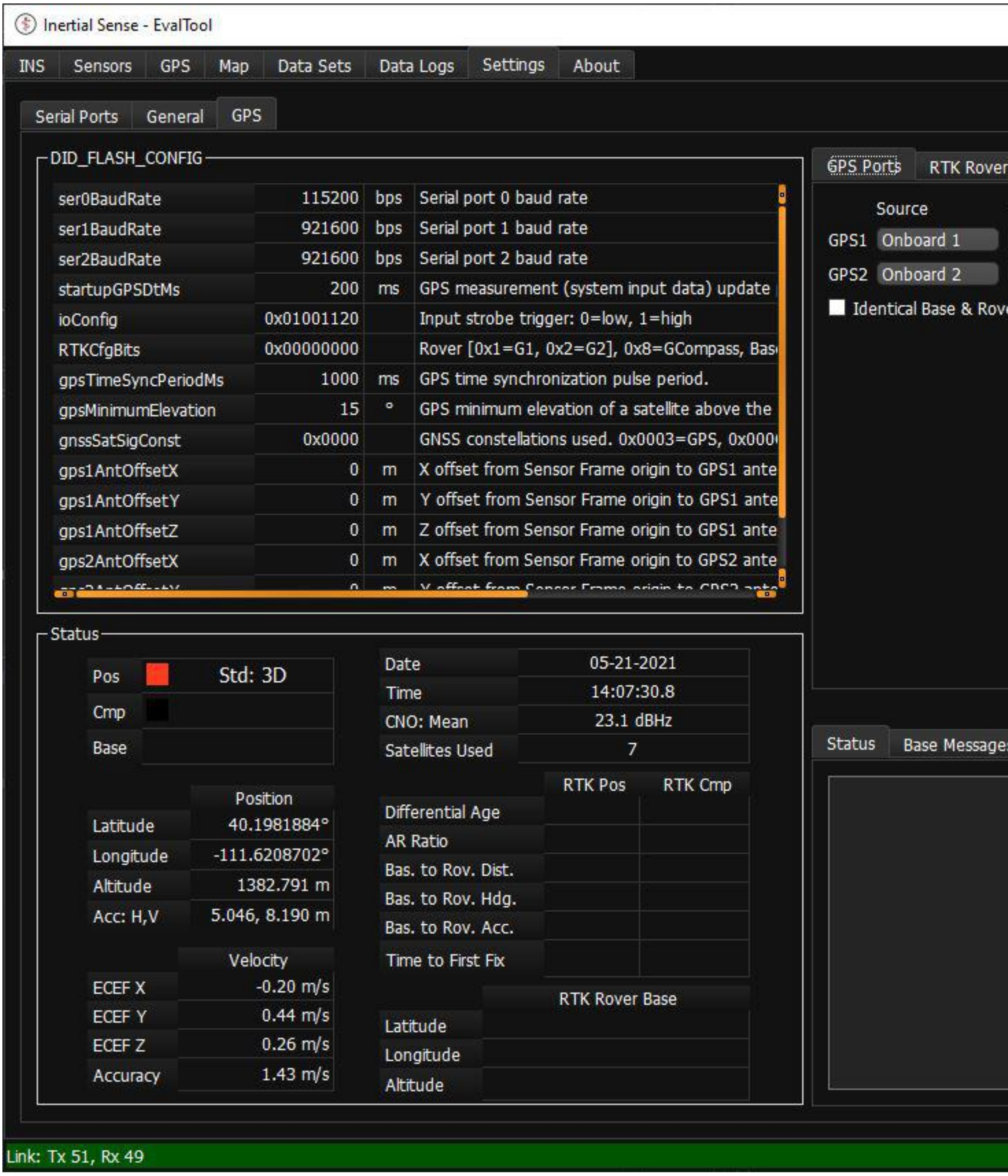
- [ ] Enble NAV Strobe

- [x] Enable CAN Bus on G

Link: Tx 32, Rx 69

1. Software Reset - Allows the user to issue a reset to the unit. has options for all open comports and only the currently connected unit.
2. Zero Motion - Allows the user to informs the EKF that the system is stationary on the ground and is used to aid in IMU bias estimation which can reduce drift in the INS attitude.
3. DID_Flash_Config - Gives the user option to disable or enable different features normally found in the "Data Sets" tab. For more information about the Flash Config see Data sets.

Settings - GPS Tab



DID_FLASH_CONFIG

ser0BaudRate	115200	bps	Serial port 0 baud rate
ser1BaudRate	921600	bps	Serial port 1 baud rate
ser2BaudRate	921600	bps	Serial port 2 baud rate
startupGPSDtMs	200	ms	GPS measurement (system input data) update
ioConfig	0x01001120		Input strobe trigger: 0=low, 1=high
RTKCfgBits	0x00000000		Rover [0x1=G1, 0x2=G2], 0x8=GCompass, Bas
gpsTimeSyncPeriodMs	1000	ms	GPS time synchronization pulse period.
gpsMinimumElevation	15	°	GPS minimum elevation of a satellite above the
gnssSatSigConst	0x0000		GNSS constellations used. 0x0003=GPS, 0x000
gps1AntOffsetX	0	m	X offset from Sensor Frame origin to GPS1 ante
gps1AntOffsetY	0	m	Y offset from Sensor Frame origin to GPS1 ante
gps1AntOffsetZ	0	m	Z offset from Sensor Frame origin to GPS1 ante
gps2AntOffsetX	0	m	X offset from Sensor Frame origin to GPS2 ante

GPS1 Onboard 1
GPS2 Onboard 2
☐ Identical Base & Rov

Status

Pos	🟥	Std: 3D
Cmp	■	
Base		

Date	05-21-2021
Time	14:07:30.8
CNO: Mean	23.1 dBHz
Satellites Used	7

	Position
Latitude	40.1981884°
Longitude	-111.6208702°
Altitude	1382.791 m
Acc: H,V	5.046, 8.190 m

	RTK Pos	RTK Cmp
Differential Age		
AR Ratio		
Bas. to Rov. Dist.		
Bas. to Rov. Hdg.		
Bas. to Rov. Acc.		
Time to First Fix		

	Velocity
ECEF X	-0.20 m/s
ECEF Y	0.44 m/s
ECEF Z	0.26 m/s
Accuracy	1.43 m/s

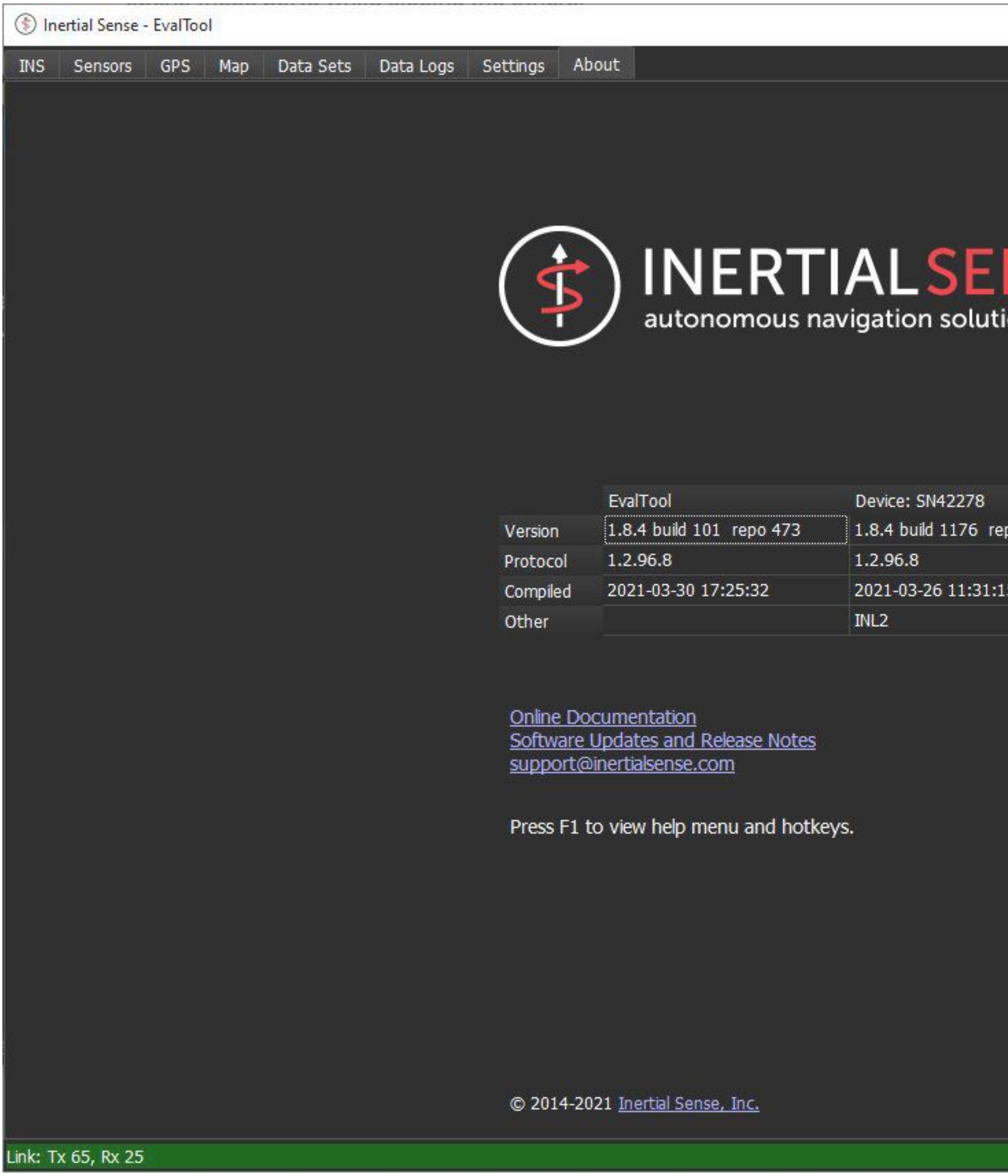
	RTK Rover Base
Latitude	
Longitude	
Altitude	

Link: Tx 51, Rx 49

1. uINS Parameters - Shows flash config settings commonly used when setting up RTK units
2. Status - Shows information important to using RTK.
3. Rover/Base Mode - Used in setup of RTK Rovers and RTK base Stations.
4. Message Window - Shows confirmation messages and Flash Config writes.

About Tab

Shows General information about the EvalTool and where more help information can be found.



	EvalTool	Device: SN42278
Version	1.8.4 build 101 repo 473	1.8.4 build 1176 rep
Protocol	1.2.96.8	1.2.96.8
Compiled	2021-03-30 17:25:32	2021-03-26 11:31:1
Other		INL2

Online Documentation
Software Updates and Release Notes
support@inertialsense.com

Press F1 to view help menu and hotkeys.

© 2014-2021 Inertial Sense, Inc.

Link: Tx 65, Rx 25

6.3 SDK

Overview

The Inertial Sense open source software development kit (SDK) provides quick integration for communication with the Inertial Sense product line, including the µIMU, µAHRS, and µINS. It includes data logger, math libraries, and serial port interface for Linux and Windows environments.

6.3.1 C vs. C++ Implementation

The Inertial Sense SDK provides both C and C++ programming language implementation. The following compares differences between these implementations.

C

- Easier implementation
- Light weight
- Smaller code size
- Minimal subset of SDK files
- Recommended for smaller projects that require lower memory usage.
- SDK files: ISComm.c

C++

- Object oriented device representation
- Fully integrated support for:
    - Commutations: single or multiple data type callback functions.
    - Serial port handling included
    - Datalogging
    - Firmware update (bootloader)
- Recommended for typical to advanced C++ applications and production level integration.
- SDK files: InertialSense.cpp

6.3.2 Installing and Configuring Visual Studio

The SDK example projects can be conveniently compiled using gcc with cmake or Visual Studio. The following sections outline how to setup Visual Studio for use with the SDK example projects.

Installing

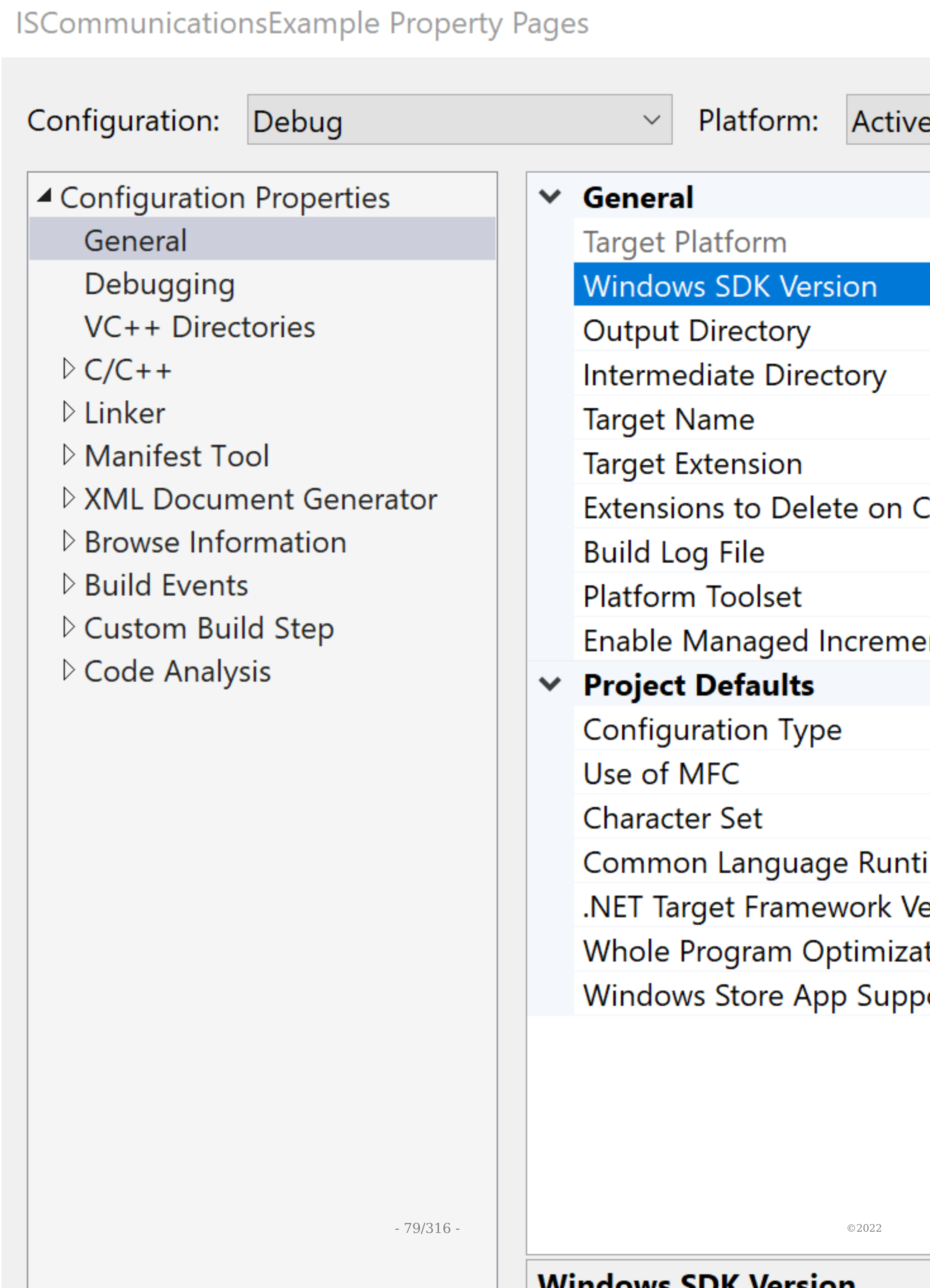
The SDK example projects can be compiled using the Community (free) version of Visual Studio. Windows SDK should be installed in addition to Visual Studio, as an added option in the Visual Studio installer or using the separate Windows SDK installer.

- Visual Studio
- Windows SDK - Can be installed using option in Visual Studio Installer or using separate Windows SDK installer.

Configuring

When compiling an Inertial Sense SDK example project in Visual Studio, the currently installed version of Windows SDK must be selected in the project properties, as illustrated below:

`Project properties > General > Windows SDK Version > [Currently Installed Version]`


ISCommunicationsExample Property Pages

Configuration: Debug Platform: Active

- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
  - Linker
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step
  - Code Analysis

General

- Target Platform
- Windows SDK Version
- Output Directory
- Intermediate Directory
- Target Name
- Target Extension
- Extensions to Delete on C
- Build Log File
- Platform Toolset
- Enable Managed Increme

Project Defaults

- Configuration Type
- Use of MFC
- Character Set
- Common Language Runti
- .NET Target Framework Ve
- Whole Program Optimizat
- Windows Store App Suppo

Windows SDK Version

6.3.3 SDK Example Projects Overview

Example Project	Language	Description
ASCII Communications	C	How to use SDK for ASCII NMEA communications.
Binary Communications	C	How to use SDK for binary communications.
Fimrware Update	C	How to use bootloader for embedded firmware update.
Data Logger	C++	How to use SDK data logging.
CLTool	C++	Open source project illustrating how to use the InertialSense C++ class. It combines all SDK capabilities including serial communications, data logging to file, and embedded firmware update.

6.3.4

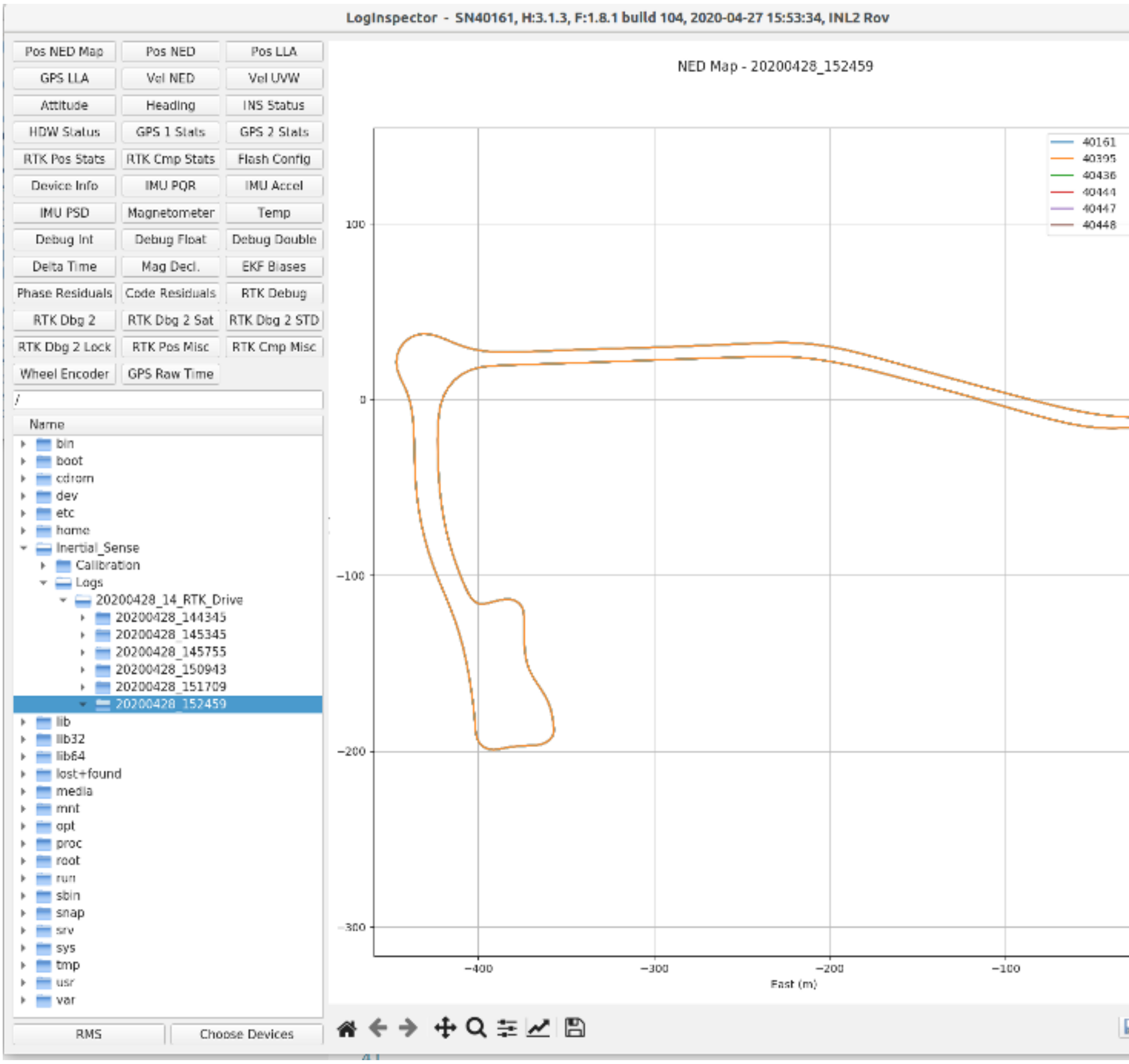
6.4 Log Inspector

6.4.1 Overview

Log Inspector is an open source python utility for viewing and scrubbing InertialSense data log (.dat) files.

6.4.2 Getting Started

Log Inspector can open and plot .dat PPD log files. The lower left hand corner file browser allows you to enter a "working directory" in the directory field. The whole directory containing the desired is selected from the directory tree. Once the log is opened, the buttons in the upper left hand corner are used to graph various data sets.



6.4.3 Standard data sets

- POS NED Map - Used to plot INS position data in NED frame.
- POS NED - INS position in NED frame.
- POS LLA - INS and GNSS position in LLA.
- GPS LLA - GNSS LLA position.
- Vel NED - Velocity in NED frame.
- Vel UVW - Velocity in body frame.
- Attitude - Euler angle attitude in degrees.
- Heading - Heading data from magnetometer, INS, and RTK.
- INS Status - Plots of status flags vs time.
- HDW Status - plots of hardware status flags vs time.

6.4.4 Building

Note - logInspector requires Python 3.

Navigate to the Inertial Sense SDK directory

```
pip3 install logInspector/ # (this will return an error message, but will install all the dependencies you need)
cd logInspector
python3 setup.py build_ext --inplace
```

Create a config file.

```
C:\Users\[USER]\Documents\Inertial_Sense\config.yaml
```

Add the following or similar contents to this file.

```
directory: C:\Users\<username>\Documents\Inertial_Sense\Logs\20181116_SKI\morning_run_1\back\20181116_175352
logs_directory: C:\Users\<username>\Documents\Inertial_Sense\Logs

serials: ["ALL"]
```

6.4.5 Running

To run logInspector open a shell and navigate to the logInspector directory and enter the following commands:

```
python3 logInspector.py
```

6.4.6 Other Directory Contents

The *logInspector* also contains some example implementations for dealing with log files directly in python.

logReader

This python module is responsible for loading the log file through a pybind11 interface. All the data in the log is eventually put in the `log.data` array.

logPlotter

This python module is responsible for creating plots. Adding new plots is easy, data is directly accessed using the member `logReader` object.

logInspector

A pyqt5 GUI which uses logPlotter to generate plots.

7. Communication Protocols

7.1 Protocol Overview

The Inertial Sense products support binary and ASCII protocol for communication.

7.1.1 Binary vs. ASCII

The following table compares the differences and advantages between the binary and ASCII protocols.

	ASCII (NMEA) Protocol	**Binary Protocol**
Data Efficient	No. Numbers must be converted to IEEE float and integers for application. Data occupies more memory.	Numbers are in floating point and integer binary format used in computers. Data occupies less memory.
Human Readable	Yes	No
Complexity	Packet are easier to parse.	Packet encoding, decoding, and parsing are MORE complicated. Using SDK is recommended.
SDK Support	Yes, less	Yes, more
Data Access	Limited to sensor and INS output.	Comprehensive access to all data and configuration settings.
Recommended Use	Rapid prototypes and simple projects. Devices supporting NMEA.	Moderate to advanced applications.
Apps and Examples	ASCII Communications Example	EvalTool, CLTool, Binary Communications Example, Fimrware Update Example, Data Logger Example

7.2 Data Sets (DIDs)

7.2.1 Data Sets (DIDs)

Data Sets in the form of C structures are available through binary protocol and provide access to system configuration and output data. The data sets are defined in SDK/src/data_sets.h of the InertialSense SDK.

INS / AHRS Output

DID_INS_1

INS output: euler rotation w/ respect to NED, NED position from reference LLA.

`ins_1_t`

Field	Type	Description
week	uint32_t	GPS number of weeks since January 6th, 1980
timeOfWeek	double	GPS time of week (since Sunday morning) in seconds
insStatus	uint32_t	INS status flags (eInsStatusFlags). Copy of DID_SYS_PARAMS.insStatus
hdwStatus	uint32_t	Hardware status flags (eHdwStatusFlags). Copy of DID_SYS_PARAMS.hdwStatus
theta	float[3]	Euler angles: roll, pitch, yaw in radians with respect to NED
uvw	float[3]	Velocity U, V, W in meters per second. Convert to NED velocity using "vectorBodyToReference(uvw, theta, vel_ned)".
lla	double[3]	WGS84 latitude, longitude, height above ellipsoid (degrees,degrees,meters)
ned	float[3]	North, east and down offset from reference latitude, longitude, and altitude to current latitude, longitude, and altitude

DID_INS_2

INS output: quaternion rotation w/ respect to NED, ellipsoid altitude

`ins_2_t`

Field	Type	Description
week	uint32_t	GPS number of weeks since January 6th, 1980
timeOfWeek	double	GPS time of week (since Sunday morning) in seconds
insStatus	uint32_t	INS status flags (eInsStatusFlags). Copy of DID_SYS_PARAMS.insStatus
hdwStatus	uint32_t	Hardware status flags (eHdwStatusFlags). Copy of DID_SYS_PARAMS.hdwStatus
qn2b	float[4]	Quaternion body rotation with respect to NED: W, X, Y, Z
uvw	float[3]	Velocity U, V, W in meters per second. Convert to NED velocity using "quatRot(vel_ned, qn2b, uvw)".
lla	double[3]	WGS84 latitude, longitude, height above ellipsoid in meters (not MSL)

DID_INS_3

Inertial navigation data with quaternion NED to body rotation and ECEF position.

`ins_3_t`

Field	Type	Description
week	uint32_t	GPS number of weeks since January 6th, 1980
timeOfWeek	double	GPS time of week (since Sunday morning) in seconds
insStatus	uint32_t	INS status flags (eInsStatusFlags). Copy of DID_SYS_PARAMS.insStatus
hdwStatus	uint32_t	Hardware status flags (eHdwStatusFlags). Copy of DID_SYS_PARAMS.hdwStatus
qn2b	float[4]	Quaternion body rotation with respect to NED: W, X, Y, Z
uvw	float[3]	Velocity U, V, W in meters per second. Convert to NED velocity using "quatRot(vel_ned, qn2b, uvw)".
lla	double[3]	WGS84 latitude, longitude, height above ellipsoid in meters (not MSL)
msl	float	height above mean sea level (MSL) in meters

DID_INS_4

INS output: quaternion rotation w/ respect to ECEF, ECEF position.

`ins_4_t`

Field	Type	Description
week	uint32_t	GPS number of weeks since January 6th, 1980
timeOfWeek	double	GPS time of week (since Sunday morning) in seconds
insStatus	uint32_t	INS status flags (eInsStatusFlags). Copy of DID_SYS_PARAMS.insStatus
hdwStatus	uint32_t	Hardware status flags (eHdwStatusFlags). Copy of DID_SYS_PARAMS.hdwStatus
qe2b	float[4]	Quaternion body rotation with respect to ECEF: W, X, Y, Z
ve	float[3]	Velocity in ECEF (earth-centered earth-fixed) frame in meters per second
ecef	double[3]	Position in ECEF (earth-centered earth-fixed) frame in meters

Inertial Measurement Unit (IMU)

DID_DUAL_IMU

Dual inertial measurement unit data down-sampled from 1KHz to navigation update rate (DID_FLASH_CONFIG.startupNavDtMs) as an anti-aliasing filter to reduce noise and preserve accuracy. Minimum data period is DID_FLASH_CONFIG.startupNavDtMs (1KHz max).

`dual_imu_t`

Field	Type	Description
time	double	Time since boot up in seconds. Convert to GPS time of week by adding gps.towOffset
I	imus_t[2]	Inertial Measurement Units (IMUs)
status	uint32_t	IMU Status (eImuStatus)

DID_DUAL_IMU_RAW

Dual inertial measurement unit data directly from IMU. We recommend use of DID_DUAL_IMU or DID_PREINTEGRATED_IMU. Minimum data period is DID_FLASH_CONFIG.startupImuDtMs or 4, whichever is larger (250Hz max).

`dual_imu_t`

Field	Type	Description
time	double	Time since boot up in seconds. Convert to GPS time of week by adding gps.towOffset
I	imus_t[2]	Inertial Measurement Units (IMUs)
status	uint32_t	IMU Status (eImuStatus)

DID_PREINTEGRATED_IMU

Coning and sculling integral in body/IMU frame. Updated at IMU rate. Also know as Delta Theta Delta Velocity, or Integrated IMU. For clarification, we use the name "Preintegrated IMU" through the User Manual. This data is integrated from the IMU data at the IMU update rate (startupImuDtMs, default 1ms). The integration period (dt) and output data rate are the same as the NAV rate (startupNavDtMs, default 4ms) and cannot be output at any other rate. If a different output data rate is desired, DID_DUAL_IMU which is derived from DID_PREINTEGRATED_IMU can be used instead. Preintegrated IMU data acts as a form of compression, adding the benefit of higher integration rates for slower output data rates, preserving the IMU data without adding filter delay and addresses antialiasing. It is most effective for systems that have higher dynamics and lower communications data rates. The minimum data period is DID_FLASH_CONFIG.startupImuDtMs or 4, whichever is larger (250Hz max).

`preintegrated_imu_t`

Field	Type	Description
time	double	Time since boot up in seconds. Convert to GPS time of week by adding gps.towOffset
theta1	float[3]	IMU 1 delta theta (gyroscope {p,q,r} integral) in radians in sensor frame
theta2	float[3]	IMU 2 delta theta (gyroscope {p,q,r} integral) in radians in sensor frame
vel1	float[3]	IMU 1 delta velocity (accelerometer {x,y,z} integral) in m/s in sensor frame
vel2	float[3]	IMU 2 delta velocity (accelerometer {x,y,z} integral) in m/s in sensor frame
dt	float	Integral period in seconds for delta theta and delta velocity. This is configured using DID_FLASH_CONFIG.startupNavDtMs.
status	uint32_t	IMU Status (eImuStatus)

Sensor Output

DID_BAROMETER

Barometric pressure sensor data

`barometer_t`

Field	Type	Description
time	double	Time since boot up in seconds. Convert to GPS time of week by adding gps.towOffset
bar	float	Barometric pressure in kilopascals
mslBar	float	MSL altitude from barometric pressure sensor in meters
barTemp	float	Temperature of barometric pressure sensor in Celsius
humidity	float	Relative humidity as a percent (%rH). Range is 0% - 100%

DID_MAGNETOMETER

Magnetometer sensor 1 output

`magnetometer_t`

Field	Type	Description
time	double	Time since boot up in seconds. Convert to GPS time of week by adding gps.towOffset
mag	float[3]	Magnetometers in Gauss

DID_MAG_CAL

Magnetometer calibration

`mag_cal_t`

Field	Type	Description
recalCmd	uint32_t	Set mode and start recalibration. 1 = multi-axis, 2 = single-axis, 101 = abort. (see eMagRecalMode)
progress	float	Mag recalibration progress indicator: 0-100 %
declination	float	Magnetic declination estimate

DID_SYS_SENSORS

System sensor information

`sys_sensors_t`

Field	Type	Description
time	double	Time since boot up in seconds. Convert to GPS time of week by adding gps.towOffset
temp	float	Temperature in Celsius
pqr	float[3]	Gyros in radians / second
acc	float[3]	Accelerometers in meters / second squared
mag	float[3]	Magnetometers in Gauss
bar	float	Barometric pressure in kilopascals
barTemp	float	Temperature of barometric pressure sensor in Celsius
mslBar	float	MSL altitude from barometric pressure sensor in meters
humidity	float	Relative humidity as a percent (%rH). Range is 0% - 100%
vin	float	EVB system input voltage in volts. uINS pin 5 (G2/AN2). Use 10K/1K resistor divider between Vin and GND.
ana1	float	ADC analog input in volts. uINS pin 4, (G1/AN1).
ana3	float	ADC analog input in volts. uINS pin 19 (G3/AN3).
ana4	float	ADC analog input in volts. uINS pin 20 (G4/AN4).

GPS / GNSS

DID_GPS1_POS

GPS 1 position data. This comes from DID_GPS1_UBX_POS or DID_GPS1_RTK_POS, depending on whichever is more accurate.

`gps_pos_t`

Field	Type	Description
week	uint32_t	GPS number of weeks since January 6th, 1980
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
status	uint32_t	(see eGpsStatus) GPS status: [0x000000xx] number of satellites used, [0x0000xx00] fix type, [0x00xx0000] status flags, NMEA input flag
ecef	double[3]	Position in ECEF {x,y,z} (m)
lla	double[3]	Position - WGS84 latitude, longitude, height above ellipsoid (not MSL) (degrees, m)
hMSL	float	Height above mean sea level (MSL) in meters
hAcc	float	Horizontal accuracy in meters
vAcc	float	Vertical accuracy in meters
pDop	float	Position dilution of precision (unitless)
cnoMean	float	Average of all non-zero satellite carrier to noise ratios (signal strengths) in dBHz
towOffset	double	Time sync offset between local time since boot up to GPS time of week in seconds. Add this to IMU and sensor time to get GPS time of week in seconds.
leapS	uint8_t	GPS leap second (GPS-UTC) offset. Receiver's best knowledge of the leap seconds offset from UTC to GPS time. Subtract from GPS time of week to get UTC time of week. (18 seconds as of December 31, 2016)
satsUsed	uint8_t	Number of satellites used
cnoMeanSigma	uint8_t	Standard deviation of cnoMean over past 5 seconds (dBHz x10)
reserved	uint8_t	Reserved for future use

DID_GPS1_RTK_POS

GPS RTK position data

`gps_pos_t`

Field	Type	Description
week	uint32_t	GPS number of weeks since January 6th, 1980
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
status	uint32_t	(see eGpsStatus) GPS status: [0x000000xx] number of satellites used, [0x0000xx00] fix type, [0x00xx0000] status flags, NMEA input flag
ecef	double[3]	Position in ECEF {x,y,z} (m)
lla	double[3]	Position - WGS84 latitude, longitude, height above ellipsoid (not MSL) (degrees, m)
hMSL	float	Height above mean sea level (MSL) in meters
hAcc	float	Horizontal accuracy in meters
vAcc	float	Vertical accuracy in meters
pDop	float	Position dilution of precision (unitless)
cnoMean	float	Average of all non-zero satellite carrier to noise ratios (signal strengths) in dBHz
towOffset	double	Time sync offset between local time since boot up to GPS time of week in seconds. Add this to IMU and sensor time to get GPS time of week in seconds.
leapS	uint8_t	GPS leap second (GPS-UTC) offset. Receiver's best knowledge of the leap seconds offset from UTC to GPS time. Subtract from GPS time of week to get UTC time of week. (18 seconds as of December 31, 2016)
satsUsed	uint8_t	Number of satellites used
cnoMeanSigma	uint8_t	Standard deviation of cnoMean over past 5 seconds (dBHz x10)
reserved	uint8_t	Reserved for future use

DID_GPS1_RTK_POS_MISC

RTK precision position related data.

`gps_rtk_misc_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
accuracyPos	float[3]	Accuracy - estimated standard deviations of the solution assuming a priori error model and error parameters by the positioning options. []: standard deviations {ECEF - x,y,z} or {north, east, down} (meters)
accuracyCov	float[3]	Accuracy - estimated standard deviations of the solution assuming a priori error model and error parameters by the positioning options. []: Absolute value of means square root of estimated covariance NE, EU, UN
arThreshold	float	Ambiguity resolution threshold for validation
gDop	float	Geometric dilution of precision (meters)
hDop	float	Horizontal dilution of precision (meters)
vDop	float	Vertical dilution of precision (meters)
baseLla	double[3]	Base Position - latitude, longitude, height (degrees, meters)
cycleSlipCount	uint32_t	Cycle slip counter
roverGpsObservationCount	uint32_t	Rover gps observation element counter
baseGpsObservationCount	uint32_t	Base station gps observation element counter
roverGlonassObservationCount	uint32_t	Rover glonass observation element counter
baseGlonassObservationCount	uint32_t	Base station glonass observation element counter
roverGalileoObservationCount	uint32_t	Rover galileo observation element counter
baseGalileoObservationCount	uint32_t	Base station galileo observation element counter
roverBeidouObservationCount	uint32_t	Rover beidou observation element counter
baseBeidouObservationCount	uint32_t	Base station beidou observation element counter
roverQzsObservationCount	uint32_t	Rover qzs observation element counter
baseQzsObservationCount	uint32_t	Base station qzs observation element counter
roverGpsEphemerisCount	uint32_t	Rover gps ephemeris element counter
baseGpsEphemerisCount	uint32_t	Base station gps ephemeris element counter
roverGlonassEphemerisCount	uint32_t	Rover glonass ephemeris element counter
baseGlonassEphemerisCount	uint32_t	Base station glonass ephemeris element counter
roverGalileoEphemerisCount	uint32_t	Rover galileo ephemeris element counter
baseGalileoEphemerisCount	uint32_t	Base station galileo ephemeris element counter
roverBeidouEphemerisCount	uint32_t	Rover beidou ephemeris element counter
baseBeidouEphemerisCount	uint32_t	Base station beidou ephemeris element counter
roverQzsEphemerisCount	uint32_t	Rover qzs ephemeris element counter
baseQzsEphemerisCount	uint32_t	Base station qzs ephemeris element counter
roverSbasCount	uint32_t	Rover sbas element counter
baseSbasCount	uint32_t	Base station sbas element counter
baseAntennaCount	uint32_t	Base station antenna position element counter

Field	Type	Description
ionUtcAlmCount	uint32_t	Ionosphere model, utc and almanac count
correctionChecksumFailures	uint32_t	Number of checksum failures from received corrections
timeToFirstFixMs	uint32_t	Time to first RTK fix.

DID_GPS1_RTK_POS_REL

RTK precision position base to rover relative info.

`gps_rtk_rel_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
differentialAge	float	Age of differential (seconds)
arRatio	float	Ambiguity resolution ratio factor for validation
baseToRoverVector	float[3]	Vector from base to rover (m) in ECEF - If Compassing enabled, this is the 3-vector from antenna 2 to antenna 1
baseToRoverDistance	float	Distance from base to rover (m)
baseToRoverHeading	float	Angle from north to baseToRoverVector in local tangent plane. (rad)
baseToRoverHeadingAcc	float	Accuracy of baseToRoverHeading. (rad)
status	uint32_t	(see eGpsStatus) GPS status: [0x000000xx] number of satellites used, [0x0000xx00] fix type, [0x00xx0000] status flags, NMEA input flag

DID_GPS1_SAT

GPS 1 GNSS and sat identifiers, carrier to noise ratio (signal strength), elevation and azimuth angles, pseudo range residual.

`gps_sat_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
numSats	uint32_t	Number of satellites in the sky
sat	gps_sat_sv_t[50]	Satellite information list

DID_GPS1_UBX_POS

GPS 1 position data from ublox receiver.

`gps_pos_t`

Field	Type	Description
week	uint32_t	GPS number of weeks since January 6th, 1980
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
status	uint32_t	(see eGpsStatus) GPS status: [0x000000xx] number of satellites used, [0x0000xx00] fix type, [0x00xx0000] status flags, NMEA input flag
ecef	double[3]	Position in ECEF {x,y,z} (m)
lla	double[3]	Position - WGS84 latitude, longitude, height above ellipsoid (not MSL) (degrees, m)
hMSL	float	Height above mean sea level (MSL) in meters
hAcc	float	Horizontal accuracy in meters
vAcc	float	Vertical accuracy in meters
pDop	float	Position dilution of precision (unitless)
cnoMean	float	Average of all non-zero satellite carrier to noise ratios (signal strengths) in dBHz
towOffset	double	Time sync offset between local time since boot up to GPS time of week in seconds. Add this to IMU and sensor time to get GPS time of week in seconds.
leapS	uint8_t	GPS leap second (GPS-UTC) offset. Receiver's best knowledge of the leap seconds offset from UTC to GPS time. Subtract from GPS time of week to get UTC time of week. (18 seconds as of December 31, 2016)
satsUsed	uint8_t	Number of satellites used
cnoMeanSigma	uint8_t	Standard deviation of cnoMean over past 5 seconds (dBHz x10)
reserved	uint8_t	Reserved for future use

DID_GPS1_VEL

GPS 1 velocity data

`gps_vel_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
vel	float[3]	GPS Velocity. Velocity is in ECEF {vx,vy,vz} (m/s) if status bit GPS_STATUS_FLAGS_GPS_NMEA_DATA (0x00008000) is NOT set. Velocity is in local tangent plane with no vertical velocity {vNorth, vEast, 0} (m/s) if status bit GPS_STATUS_FLAGS_GPS_NMEA_DATA (0x00008000) is set.
sAcc	float	Speed accuracy in meters / second
status	uint32_t	(see eGpsStatus) GPS status: [0x000000xx] number of satellites used, [0x0000xx00] fix type, [0x00xx0000] status flags, NMEA input flag

DID_GPS1_VERSION

GPS 1 version info

`gps_version_t`

Field	Type	Description
swVersion	uint8_t[30]	Software version
hwVersion	uint8_t[10]	Hardware version
extension	uint8_t[30]	Extension
reserved	uint8_t[2]	ensure 32 bit aligned in memory

DID_GPS2_POS

GPS 2 position data

`gps_pos_t`

Field	Type	Description
week	uint32_t	GPS number of weeks since January 6th, 1980
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
status	uint32_t	(see eGpsStatus) GPS status: [0x000000xx] number of satellites used, [0x0000xx00] fix type, [0x00xx0000] status flags, NMEA input flag
ecef	double[3]	Position in ECEF {x,y,z} (m)
lla	double[3]	Position - WGS84 latitude, longitude, height above ellipsoid (not MSL) (degrees, m)
hMSL	float	Height above mean sea level (MSL) in meters
hAcc	float	Horizontal accuracy in meters
vAcc	float	Vertical accuracy in meters
pDop	float	Position dilution of precision (unitless)
cnoMean	float	Average of all non-zero satellite carrier to noise ratios (signal strengths) in dBHz
towOffset	double	Time sync offset between local time since boot up to GPS time of week in seconds. Add this to IMU and sensor time to get GPS time of week in seconds.
leapS	uint8_t	GPS leap second (GPS-UTC) offset. Receiver's best knowledge of the leap seconds offset from UTC to GPS time. Subtract from GPS time of week to get UTC time of week. (18 seconds as of December 31, 2016)
satsUsed	uint8_t	Number of satellites used
cnoMeanSigma	uint8_t	Standard deviation of cnoMean over past 5 seconds (dBHz x10)
reserved	uint8_t	Reserved for future use

DID_GPS2_RTK_CMP_MISC

RTK Dual GNSS RTK compassing related data.

`gps_rtk_misc_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
accuracyPos	float[3]	Accuracy - estimated standard deviations of the solution assuming a priori error model and error parameters by the positioning options. []: standard deviations {ECEF - x,y,z} or {north, east, down} (meters)
accuracyCov	float[3]	Accuracy - estimated standard deviations of the solution assuming a priori error model and error parameters by the positioning options. []: Absolute value of means square root of estimated covariance NE, EU, UN
arThreshold	float	Ambiguity resolution threshold for validation
gDop	float	Geometric dilution of precision (meters)
hDop	float	Horizontal dilution of precision (meters)
vDop	float	Vertical dilution of precision (meters)
baseLla	double[3]	Base Position - latitude, longitude, height (degrees, meters)
cycleSlipCount	uint32_t	Cycle slip counter
roverGpsObservationCount	uint32_t	Rover gps observation element counter
baseGpsObservationCount	uint32_t	Base station gps observation element counter
roverGlonassObservationCount	uint32_t	Rover glonass observation element counter
baseGlonassObservationCount	uint32_t	Base station glonass observation element counter
roverGalileoObservationCount	uint32_t	Rover galileo observation element counter
baseGalileoObservationCount	uint32_t	Base station galileo observation element counter
roverBeidouObservationCount	uint32_t	Rover beidou observation element counter
baseBeidouObservationCount	uint32_t	Base station beidou observation element counter
roverQzsObservationCount	uint32_t	Rover qzs observation element counter
baseQzsObservationCount	uint32_t	Base station qzs observation element counter
roverGpsEphemerisCount	uint32_t	Rover gps ephemeris element counter
baseGpsEphemerisCount	uint32_t	Base station gps ephemeris element counter
roverGlonassEphemerisCount	uint32_t	Rover glonass ephemeris element counter
baseGlonassEphemerisCount	uint32_t	Base station glonass ephemeris element counter
roverGalileoEphemerisCount	uint32_t	Rover galileo ephemeris element counter
baseGalileoEphemerisCount	uint32_t	Base station galileo ephemeris element counter
roverBeidouEphemerisCount	uint32_t	Rover beidou ephemeris element counter
baseBeidouEphemerisCount	uint32_t	Base station beidou ephemeris element counter
roverQzsEphemerisCount	uint32_t	Rover qzs ephemeris element counter
baseQzsEphemerisCount	uint32_t	Base station qzs ephemeris element counter
roverSbasCount	uint32_t	Rover sbas element counter
baseSbasCount	uint32_t	Base station sbas element counter
baseAntennaCount	uint32_t	Base station antenna position element counter

Field	Type	Description
ionUtcAlmCount	uint32_t	Ionosphere model, utc and almanac count
correctionChecksumFailures	uint32_t	Number of checksum failures from received corrections
timeToFirstFixMs	uint32_t	Time to first RTK fix.

DID_GPS2_RTK_CMP_REL

Dual GNSS RTK compassing / moving base to rover (GPS 1 to GPS 2) relative info.

`gps_rtk_rel_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
differentialAge	float	Age of differential (seconds)
arRatio	float	Ambiguity resolution ratio factor for validation
baseToRoverVector	float[3]	Vector from base to rover (m) in ECEF - If Compassing enabled, this is the 3-vector from antenna 2 to antenna 1
baseToRoverDistance	float	Distance from base to rover (m)
baseToRoverHeading	float	Angle from north to baseToRoverVector in local tangent plane. (rad)
baseToRoverHeadingAcc	float	Accuracy of baseToRoverHeading. (rad)
status	uint32_t	(see eGpsStatus) GPS status: [0x000000xx] number of satellites used, [0x0000xx00] fix type, [0x00xx0000] status flags, NMEA input flag

DID_GPS2_SAT

GPS 2 GNSS and sat identifiers, carrier to noise ratio (signal strength), elevation and azimuth angles, pseudo range residual.

`gps_sat_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
numSats	uint32_t	Number of satellites in the sky
sat	gps_sat_sv_t[50]	Satellite information list

DID_GPS2_VEL

GPS 2 velocity data

`gps_vel_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
vel	float[3]	GPS Velocity. Velocity is in ECEF {vx,vy,vz} (m/s) if status bit GPS_STATUS_FLAGS_GPS_NMEA_DATA (0x00008000) is NOT set. Velocity is in local tangent plane with no vertical velocity {vNorth, vEast, 0} (m/s) if status bit GPS_STATUS_FLAGS_GPS_NMEA_DATA (0x00008000) is set.
sAcc	float	Speed accuracy in meters / second
status	uint32_t	(see eGpsStatus) GPS status: [0x000000xx] number of satellites used, [0x0000xx00] fix type, [0x00xx0000] status flags, NMEA input flag

DID_GPS2_VERSION

GPS 2 version info

`gps_version_t`

Field	Type	Description
swVersion	uint8_t[30]	Software version
hwVersion	uint8_t[10]	Hardware version
extension	uint8_t[30]	Extension
reserved	uint8_t[2]	ensure 32 bit aligned in memory

DID_GPS_RTK_OPT

RTK options - requires little endian CPU.

`gps_rtk_opt_t`

Field	Type	Description
mode	int32_t	positioning mode (PMODE_???)
soltype	int32_t	solution type (0:forward,1:backward,2:combined)
nf	int32_t	number of frequencies (1:L1,2:L1+L2,3:L1+L2+L5)
navsys	int32_t	navigation systems
elmin	double	elevation mask angle (rad)
snrmin	int32_t	Min snr to consider satellite for rtk
modear	int32_t	AR mode (0:off,1:continuous,2:instantaneous,3:fix and hold,4:ppp-ar)
glomodear	int32_t	GLONASS AR mode (0:off,1:on,2:auto cal,3:ext cal)
gpsmodear	int32_t	GPS AR mode (0:off,1:on)
sbsmodear	int32_t	SBAS AR mode (0:off,1:on)
bdsmodear	int32_t	BeiDou AR mode (0:off,1:on)
arfilter	int32_t	AR filtering to reject bad sats (0:off,1:on)
maxout	int32_t	obs outage count to reset bias
maxrej	int32_t	reject count to reset bias
minlock	int32_t	min lock count to fix ambiguity
minfixsats	int32_t	min sats to fix integer ambiguities
minholdsats	int32_t	min sats to hold integer ambiguities
mindropsats	int32_t	min sats to drop sats in AR
rcvstds	int32_t	use stdev estimates from receiver to adjust measurement variances
minfix	int32_t	min fix count to hold ambiguity
armaxiter	int32_t	max iteration to resolve ambiguity
dynamics	int32_t	dynamics model (0:none,1:velociy,2:accel)
niter	int32_t	number of filter iteration
intpref	int32_t	interpolate reference obs (for post mission)
rovpos	int32_t	rover position for fixed mode
refpos	int32_t	base position for relative mode
eratio	double[1]	code/phase error ratio
err	double[5]	measurement error factor
std	double[3]	initial-state std [0]bias,[1]iono [2]trop
prn	double[6]	process-noise std [0]bias,[1]iono [2]trop [3]acch [4]accv [5] pos
sclkstab	double	satellite clock stability (sec/sec)
thresar	double[8]	AR validation threshold
elmaskar	double	elevation mask of AR for rising satellite (rad)
elmaskhold	double	elevation mask to hold ambiguity (rad)
thresslip	double	slip threshold of geometry-free phase (m)

Field	Type	Description
varholdamb	double	variance for fix-and-hold pseudo measurements (cycle^2)
gainholdamb	double	gain used for GLO and SBAS sats to adjust ambiguity
maxtdiff	double	max difference of time (sec)
fix_reset_base_msgs	int	reset sat biases after this long trying to get fix if not acquired
maxinnocode	double	reject threshold of NIS
maxinnophase	double	reject threshold of gdop
maxnis	double	baseline length constraint {const,sigma before fix, sigma after fix} (m)
maxgdop	double	maximum error wrt ubx position (triggers reset if more than this far) (m)
baseline	double[3]	rover position for fixed mode {x,y,z} (ecef) (m)
max_baseline_error	double	base position for relative mode {x,y,z} (ecef) (m)
reset_baseline_error	double	max averaging epochs
max_ubx_error	float	output single by dgps/float/fix/ppp outage

Raw GPS Data

Raw GPS data is contained in the `DID_GPS1_RAW`, `DID_GPS2_RAW`, and `DID_GPS_BASE_RAW` messages of type `gps_raw_t`. The actual raw data is contained in the union member `gps_raw_t.data` and should be interpreted based on the value of `gps_raw_t.dataType` (i.e. as observation, ephemeris, SBAS, or base station position).

DID_GPS1_RAW

GPS raw data for rover (observation, ephemeris, etc.) - requires little endian CPU. The contents of data can vary for this message and are determined by dataType field. RTK positioning or RTK compassing must be enabled to stream this message.

`gps_raw_t`

Field	Type	Description
receiverIndex	uint8_t	Receiver index (1=RECEIVER_INDEX_GPS1, 2=RECEIVER_INDEX_EXTERNAL_BASE, or 3=RECEIVER_INDEX_GPS2)
dataType	uint8_t	Type of data (eRawDataType: 1=observations, 2=ephemeris, 3=glonassEphemeris, 4=SBAS, 5=baseAntenna, 6=IonosphereModel)
obsCount	uint8_t	Number of observations in data (obsd_t) when dataType==1 (raw_data_type_observation).
reserved	uint8_t	Reserved
data	uGpsRawData	Interpret based on dataType (see eRawDataType)

DID_GPS2_RAW

GPS raw data for rover (observation, ephemeris, etc.) - requires little endian CPU. The contents of data can vary for this message and are determined by dataType field. RTK positioning or RTK compassing must be enabled to stream this message.

`gps_raw_t`

Field	Type	Description
receiverIndex	uint8_t	Receiver index (1=RECEIVER_INDEX_GPS1, 2=RECEIVER_INDEX_EXTERNAL_BASE, or 3=RECEIVER_INDEX_GPS2)
dataType	uint8_t	Type of data (eRawDataType: 1=observations, 2=ephemeris, 3=glonassEphemeris, 4=SBAS, 5=baseAntenna, 6=IonosphereModel)
obsCount	uint8_t	Number of observations in data (obsd_t) when dataType==1 (raw_data_type_observation).
reserved	uint8_t	Reserved
data	uGpsRawData	Interpret based on dataType (see eRawDataType)

DID_GPS_BASE_RAW

GPS raw data for base station (observation, ephemeris, etc.) - requires little endian CPU. The contents of data can vary for this message and are determined by dataType field. RTK positioning or RTK compassing must be enabled to stream this message.

`gps_raw_t`

Field	Type	Description
receiverIndex	uint8_t	Receiver index (1=RECEIVER_INDEX_GPS1, 2=RECEIVER_INDEX_EXTERNAL_BASE, or 3=RECEIVER_INDEX_GPS2)
dataType	uint8_t	Type of data (eRawDataType: 1=observations, 2=ephemeris, 3=glonassEphemeris, 4=SBAS, 5=baseAntenna, 6=IonosphereModel)
obsCount	uint8_t	Number of observations in data (obsd_t) when dataType==1 (raw_data_type_observation).
reserved	uint8_t	Reserved
data	uGpsRawData	Interpret based on dataType (see eRawDataType)

RAW GPS DATA BUFFER UNION

`uGpsRawData`

Field	Type	Description
obs	obsd_t[]	Satellite observation data
eph	eph_t	Satellite non-GLONASS ephemeris data (GPS, Galileo, Beidou, QZSS)
gloEph	geph_t	Satellite GLONASS ephemeris data
sbas	sbsmsg_t	Satellite-Based Augmentation Systems (SBAS) data
sta	sta_t	Base station information (base position, antenna position, antenna height, etc.)
ion	ion_model_utc_alm_t	Ionosphere model and UTC parameters
buf	uint8_t[1000]	Byte buffer

GPS GALILEO QZSS EPHEMERIS

`eph_t`

Field	Type	Description
sat	int32_t	Satellite number in RTKlib notation. GPS: 1-32, GLONASS: 33-59, Galilleo: 60-89, SBAS: 90-95
iode	int32_t	IODE Issue of Data, Ephemeris (ephemeris version)
iodc	int32_t	IODC Issue of Data, Clock (clock version)
sva	int32_t	SV accuracy (URA index) IRN-IS-200H p.97
svh	int32_t	SV health GPS/QZS (0:ok)
week	int32_t	GPS/QZS: gps week, GAL: galileo week
code	int32_t	GPS/QZS: code on L2. (00 = Invalid, 01 = P Code ON, 11 = C/A code ON, 11 = Invalid). GAL/CMP: data sources
flag	int32_t	GPS/QZS: L2 P data flag (indicates that the NAV data stream was commanded OFF on the P-code of the in-phase component of the L2 channel). CMP: nav type
toe	gtime_t	Time Of Ephemeris, ephemeris reference epoch in seconds within the week (s)
toc	gtime_t	clock data reference time (s) (20.3.4.5)
ttr	gtime_t	T_trans (s)
A	double	Orbit semi-major axis (m)
e	double	Orbit eccentricity (non-dimensional)
i0	double	Orbit inclination angle at reference time (rad)
OMG0	double	Longitude of ascending node of orbit plane at weekly epoch (rad)
omg	double	Argument of perigee (rad)
M0	double	Mean anomaly at reference time (rad)
deln	double	Mean Motion Difference From Computed Value (rad)
OMGd	double	Rate of Right Ascension (rad/s)
idot	double	Rate of Inclination Angle (rad/s)
crc	double	Amplitude of the Cosine Harmonic Correction Term to the Orbit Radius (m)
crs	double	Amplitude of the Sine Harmonic Correction Term to the Orbit Radius (m)
cuc	double	Amplitude of the Cosine Harmonic Correction Term to the Argument of Latitude (rad)
cus	double	Amplitude of the Sine Harmonic Correction Term to the Argument of Latitude (rad)
cic	double	Amplitude of the Cosine Harmonic Correction Term to the Angle of Inclination (rad)
cis	double	Amplitude of the Sine Harmonic Correction Term to the Angle of Inclination (rad)
toes	double	Time Of Ephemeris, ephemeris reference epoch in seconds within the week (s), same as above but represented as double type. Note that toe is computed as eph->toe = gst2time(week, eph->toes)
fit	double	Fit interval (h) (0: 4 hours, 1: greater than 4 hours)
f0	double	SV clock offset, af0 (s)
f1	double	SV clock drift, af1 (s/s, non-dimensional)
f2	double	SV clock drift rate, af2 (1/s)
tgd	double[4]	Group delay parameters GPS/QZS: tgd[0] = TGD (IRN-IS-200H p.103). Galilleo: tgd[0] = BGD E5a/E1, tgd[1] = BGD E5b/E1. Beidou: tgd[0] = BGD1, tgd[1] = BGD2

Field	Type	Description
Adot	double	Adot for CNAV, not used
ndot	double	First derivative of mean motion n (second derivative of mean anomaly M), ndot for CNAV (rad/s/s). Not used.

GLONASS EPHEMERIS

`geph_t`

Field	Type	Description
sat	int32_t	Satellite number in RTKlib notation. GPS: 1-32, GLONASS: 33-59, Galilleo: 60-89, SBAS: 90-95
iode	int32_t	IODE (0-6 bit of tb field)
frq	int32_t	satellite frequency number
svh	int32_t	satellite health
sva	int32_t	satellite accuracy
age	int32_t	satellite age of operation
toe	gtime_t	Ephemeris reference epoch in seconds within the week in GPS time gpst (s)
tof	gtime_t	message frame time in gpst (s)
pos	double[3]	satellite position (ecef) (m)
vel	double[3]	satellite velocity (ecef) (m/s)
acc	double[3]	satellite acceleration (ecef) (m/s^2)
taun	double	SV clock bias (s)
gamn	double	relative frequency bias
dtaun	double	delay between L1 and L2 (s)

SBAS

`sbsmsg_t`

Field	Type	Description
week	int32_t	receiption time - week
tow	int32_t	reception time - tow
prn	int32_t	SBAS satellite PRN number
msg	uint8_t[29]	SBAS message (226bit) padded by 0
reserved	uint8_t[3]	reserved for alighment

STATION PARAMETERS

`sta_t`

Field	Type	Description
deltype	int32_t	antenna delta type (0:enu,1:xyz)
pos	double[3]	station position (ecef) (m)
del	double[3]	antenna position delta (e/n/u or x/y/z) (m)
hgt	double	antenna height (m)
stationId	int32_t	station id

OBSERVATION DATA

`obsd_t`

Field	Type	Description
time	gtime_t	Receiver local time approximately aligned to the GPS time system (GPST)
sat	uint8_t	Satellite number in RTKlib notation. GPS: 1-32, GLONASS: 33-59, Galilleo: 60-89, SBAS: 90-95
rcv	uint8_t	receiver number
SNR	uint8_t[1]	Cno, carrier-to-noise density ratio (signal strength) (0.25 dB-Hz)
LLI	uint8_t[1]	Loss of Lock Indicator. Set to non-zero values only when carrier-phase is valid (L > 0). bit1 = loss-of-lock, bit2 = half-cycle-invalid
code	uint8_t[1]	Code indicator: CODE_L1C (1) = L1C/A,G1C/A,E1C (GPS,GLO,GAL,QZS,SBS), CODE_L1X (12) = E1B+C,L1C(D+P) (GAL,QZS), CODE_L1I (47) = B1I (BeiDou)
qualL	uint8_t[1]	Estimated carrier phase measurement standard deviation (0.004 cycles), zero means invalid
qualP	uint8_t[1]	Estimated pseudorange measurement standard deviation (0.01 m), zero means invalid
reserved	uint8_t	reserved, for alignment
L	double[1]	Observation data carrier-phase (cycle). The carrier phase initial ambiguity is initialized using an approximate value to make the magnitude of the phase close to the pseudorange measurement. Clock resets are applied to both phase and code measurements in accordance with the RINEX specification.
P	double[1]	Observation data pseudorange (m). GLONASS inter frequency channel delays are compensated with an internal calibration table
D	float[1]	Observation data Doppler measurement (positive sign for approaching satellites) (Hz)

SATELLITE OBSERVATION

`obs_t`

Field	Type	Description
n	uint32_t	number of observation slots used
nmax	uint32_t	number of observation slots allocated
data	obsd_t	observation data buffer

SATELLITE INFORMATION

`gps_sat_sv_t`

Field	Type	Description
gnssId	uint8_t	GNSS identifier: 0 GPS, 1 SBAS, 2 Galileo, 3 BeiDou, 5 QZSS, 6 GLONASS
svId	uint8_t	Satellite identifier
cno	uint8_t	(dBHz) Carrier to noise ratio (signal strength)
elev	int8_t	(deg) Elevation (range: ±90)
azim	int16_t	(deg) Azimuth (range: ±180)
prRes	int16_t	(m) Pseudo range residual
flags	uint32_t	(see eSatSvFlags)

INERTIAL MEASUREMENT UNIT (IMU)

`imus_t`

Field	Type	Description
pqr	float[3]	Gyroscope P, Q, R in radians / second
acc	float[3]	Acceleration X, Y, Z in meters / second squared

Configuration

DID_ASCII_BCAST_PERIOD

Broadcast period for ASCII messages

`ascii_msgs_t`

Field	Type	Description
options	uint32_t	Options: Port selection[0x0=current, 0xFF=all, 0x1=ser0, 0x2=ser1, 0x4=ser2, 0x8=USB] (see RMC_OPTIONS_...)
pimu	uint16_t	Broadcast period (ms) - ASCII dual IMU data. 0 to disable.
ppimu	uint16_t	Broadcast period (ms) - ASCII preintegrated dual IMU: delta theta (rad) and delta velocity (m/s). 0 to disable.
pins1	uint16_t	Broadcast period (ms) - ASCII INS output: euler rotation w/ respect to NED, NED position from reference LLA. 0 to disable.
pins2	uint16_t	Broadcast period (ms) - ASCII INS output: quaternion rotation w/ respect to NED, ellipsoid altitude. 0 to disable.
pgpsp	uint16_t	Broadcast period (ms) - ASCII GPS position data. 0 to disable.
reserved	uint16_t	Broadcast period (ms) - Reserved. Leave zero.
gpgga	uint16_t	Broadcast period (ms) - ASCII NMEA GPGGA GPS 3D location, fix, and accuracy. 0 to disable.
gpgll	uint16_t	Broadcast period (ms) - ASCII NMEA GPGLL GPS 2D location and time. 0 to disable.
gpgsa	uint16_t	Broadcast period (ms) - ASCII NMEA GSA GPS DOP and active satellites. 0 to disable.
gprmc	uint16_t	Broadcast period (ms) - ASCII NMEA recommended minimum specific GPS/Transit data. 0 to disable.
gpzda	uint16_t	Broadcast period (ms) - ASCII NMEA Data and Time. 0 to disable.
pashr	uint16_t	Broadcast period (ms) - ASCII NMEA Inertial Attitude Data. 0 to disable.

DID_FLASH_CONFIG

Flash memory configuration

`nvm_flash_cfg_t`

Field	Type	Description
size	uint32_t	Size of group or union, which is nvm_group_x_t + padding
checksum	uint32_t	Checksum, excluding size and checksum
key	uint32_t	Manufacturer method for restoring flash defaults
startupImuDtMs	uint32_t	IMU sample (system input data) period in milliseconds set on startup. Cannot be larger than startupNavDtMs. Zero disables sensor/IMU sampling.
startupNavDtMs	uint32_t	Nav filter (system output data) update period in milliseconds set on startup. 1ms minimum (1KHz max). Zero disables nav filter updates.
ser0BaudRate	uint32_t	Serial port 0 baud rate in bits per second
ser1BaudRate	uint32_t	Serial port 1 baud rate in bits per second
insRotation	float[3]	Rotation in radians about the X, Y, Z axes from INS Sensor Frame to Intermediate Output Frame. Order applied: Z, Y, X.
insOffset	float[3]	X,Y,Z offset in meters from Intermediate Output Frame to INS Output Frame.
gps1AntOffset	float[3]	X,Y,Z offset in meters from Sensor Frame origin to GPS 1 antenna.
insDynModel	uint8_t	INS dynamic platform model (see eInsDynModel). Options are: 0=PORTABLE, 2=STATIONARY, 3=PEDESTRIAN, 4=GROUND VEHICLE, 5=SEA, 6=AIRBORNE_1G, 7=AIRBORNE_2G, 8=AIRBORNE_4G, 9=WRIST. Used to balance noise and performance characteristics of the system. The dynamics selected here must be at least as fast as your system or you experience accuracy error. This is tied to the GPS position estimation model and intend in the future to be incorporated into the INS position model.
reserved	uint8_t	Reserved
gnssSatSigConst	uint16_t	Satellite system constellation used in GNSS solution. (see eGnssSatSigConst) 0x0003=GPS, 0x000C=QZSS, 0x0030=Galileo, 0x00C0=Beidou, 0x0300=GLONASS, 0x1000=SBAS
sysCfgBits	uint32_t	System configuration bits (see eSysConfigBits).
refLla	double[3]	Reference latitude, longitude and height above ellipsoid for north east down (NED) calculations (deg, deg, m)
lastLla	double[3]	Last latitude, longitude, HAE (height above ellipsoid) used to aid GPS startup (deg, deg, m). Updated when the distance between current LLA and lastLla exceeds lastLlaUpdateDistance.
lastLlaTimeOfWeekMs	uint32_t	Last LLA GPS time since week start (Sunday morning) in milliseconds
lastLlaWeek	uint32_t	Last LLA GPS number of weeks since January 6th, 1980
lastLlaUpdateDistance	float	Distance between current and last LLA that triggers an update of lastLla
ioConfig	uint32_t	Hardware interface configuration bits (see eIoConfig).
cBrdConfig	uint32_t	Carrier board (i.e. eval board) configuration bits
gps2AntOffset	float[3]	X,Y,Z offset in meters from DOD_ Frame origin to GPS 2 antenna.
zeroVelRotation	float[3]	Euler (roll, pitch, yaw) rotation in radians from INS Sensor Frame to Intermediate ZeroVelocity Frame. Order applied: heading, pitch, roll.
zeroVelOffset	float[3]	X,Y,Z offset in meters from Intermediate ZeroVelocity Frame to Zero Velocity Frame.
magInclination	float	Earth magnetic field (magnetic north) inclination (negative pitch offset) in radians

Field	Type	Description
magDeclination	float	Earth magnetic field (magnetic north) declination (heading offset from true north) in radians
gpsTimeSyncPeriodMs	uint32_t	Time between GPS time synchronization pulses in milliseconds. Requires reboot to take effect.
startupGPSDtMs	uint32_t	GPS measurement (system input data) update period in milliseconds set on startup. 200ms minimum (5Hz max).
RTKCfgBits	uint32_t	RTK configuration bits (see eRTKConfigBits).
sensorConfig	uint32_t	Sensor config to specify the full-scale sensing ranges and output rotation for the IMU and magnetometer (see eSensorConfig in data_sets.h)
gpsMinimumElevation	float	Minimum elevation of a satellite above the horizon to be used in the solution (radians). Low elevation satellites may provide degraded accuracy, due to the long signal path through the atmosphere.
ser2BaudRate	uint32_t	Serial port 2 baud rate in bits per second
wheelConfig	wheel_config_t	Wheel encoder: euler angles describing the rotation from imu to left wheel

DID_RMC

Realtime Message Controller (RMC). The data sets available through RMC are driven by the availability of the data. The RMC provides updates from various data sources (i.e. sensors) as soon as possible with minimal latency. Several of the data sources (sensors) output data at different data rates that do not all correspond. The RMC is provided so that broadcast of sensor data is done as soon as it becomes available. All RMC messages can be enabled using the standard Get Data packet format.

`rmc_t`

Field	Type	Description
bits	uint64_t	Data stream enable bits for the specified ports. (see RMC_BITS_...)
options	uint32_t	Options to select alternate ports to output data, etc. (see RMC_OPTIONS_...)

Command

DID_SYS_CMD

System commands. Both the command and invCommand fields must be set at the same time for a command to take effect.

`system_command_t`

Field	Type	Description
command	uint32_t	System commands (see eSystemCommand) 1=save current persistent messages, 5=zero motion, 97=save flash, 99=software reset. "invCommand" (following variable) must be set to bitwise inverse of this value for this command to be processed.
invCommand	uint32_t	Error checking field that must be set to bitwise inverse of command field for the command to take effect.

EVB-2

DID_EVB_FLASH_CFG

EVB configuration.

`evb_flash_cfg_t`

Field	Type	Description
size	uint32_t	Size of this struct
checksum	uint32_t	Checksum, excluding size and checksum
key	uint32_t	Manufacturer method for restoring flash defaults
cbPreset	uint8_t	Communications bridge preset. (see eEvb2ComBridgePreset)
reserved1	uint8_t[3]	Communications bridge forwarding
cbf	uint32_t[EVB2_PORT_COUNT]	Communications bridge options (see eEvb2ComBridgeOptions)
cbOptions	uint32_t	Config bits (see eEvbFlashCfgBits)
bits	uint32_t	Radio preamble ID (PID) - 0x0 to 0x9. Only radios with matching PIDs can communicate together. Different PIDs minimize interference between multiple sets of networks. Checked before the network ID.
radioPID	uint32_t	Radio network ID (NID) - 0x0 to 0x7FFF. Only radios with matching NID can communicate together. Checked after the preamble ID.
radioNID	uint32_t	Radio power level - Transmitter output power level. (XBee PRO SX 0=20dBm, 1=27dBm, 2=30dBm)
radioPowerLevel	uint32_t	WiFi SSID and PSK
wifi	evb_wifi_t[3]	Server IP and port
server	evb_server_t[3]	Encoder tick to wheel rotation conversion factor (in radians). Encoder tick count per revolution on 1 channel x gear ratio x 2pi.
encoderTickToWheelRad	float	CAN baudrate
CANbaud_kbps	uint32_t	CAN receive address
can_receive_address	uint32_t	EVB port for uINS communications and SD card logging. 0=uINS-Ser0 (default), 1=uINS-Ser1, SP330=5, 6=GPIO_H8 (use eEvb2CommPorts)
uinsComPort	uint8_t	EVB port for uINS aux com and RTK corrections. 0=uINS-Ser0, 1=uINS-Ser1 (default), 5=SP330, 6=GPIO_H8 (use eEvb2CommPorts)
uinsAuxPort	uint8_t	Enable radio RTK filtering, etc. (see eEvb2PortOptions)
reserved2	uint8_t[2]	Baud rate for EVB serial port H3 (SP330 RS233 and RS485/422).
portOptions	uint32_t	Baud rate for EVB serial port H4 (TLL to external radio).
h3sp330BaudRate	uint32_t	Baud rate for EVB serial port H8 (TLL).
h4xRadioBaudRate	uint32_t	Wheel encoder configuration (see eWheelCfgBits)
h8gpioBaudRate	uint32_t	Wheel update period. Sets the wheel encoder and control update period. (ms)

DID_EVB_STATUS

EVB monitor and log control interface.

`evb_status_t`

Field	Type	Description
week	uint32_t	GPS number of weeks since January 6th, 1980
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
firmwareVer	uint8_t[4]	Firmware (software) version
evbStatus	uint32_t	Status (eEvbStatus)
loggerMode	uint32_t	Data logger control state. (see eEvb2LoggerMode)
loggerElapsedTimeMs	uint32_t	logger
wifiIpAddr	uint32_t	WiFi IP address
sysCommand	uint32_t	System command (see eSystemCommand). 99 = software reset

General

DID_BIT

System built-in self-test

`bit_t`

Field	Type	Description
state	uint32_t	Built-in self-test state (see eBitState)
hdwBitStatus	uint32_t	Hardware BIT status (see eHdwBitStatusFlags)
calBitStatus	uint32_t	Calibration BIT status (see eCalBitStatusFlags)
tcPqrBias	float	Temperature calibration bias
tcAccBias	float	Temperature calibration slope
tcPqrSlope	float	Temperature calibration linearity
tcAccSlope	float	Gyro error (rad/s)
tcPqrLinearity	float	Accelerometer error (m/s^2)
tcAccLinearity	float	Angular rate standard deviation
pqr	float	Acceleration standard deviation
acc	float	Self-test mode (see eBitTestMode)

DID_CAN_CONFIG

Addresses for CAN messages

`can_config_t`

Field	Type	Description
can_period_mult	uint32_t[]	Broadcast period (ms) - CAN time message. 0 to disable.
can_transmit_address	uint32_t[]	Transmit address.
can_baudrate_kbps	uint32_t	Baud rate (kbps) (See can_baudrate_t for valid baud rates)
can_receive_address	uint32_t	Receive address.

DID_DEV_INFO

Device information

`dev_info_t`

Field	Type	Description
reserved	uint32_t	Reserved bits
serialNumber	uint32_t	Serial number
hardwareVer	uint8_t[4]	Hardware version
firmwareVer	uint8_t[4]	Firmware (software) version
buildNumber	uint32_t	Build number
protocolVer	uint8_t[4]	Communications protocol version
repoRevision	uint32_t	Repository revision number
manufacturer	char[24]	Manufacturer name
buildDate	uint8_t[4]	Build date, little endian order: [0] = status ('r'=release, 'd'=debug), [1] = year-2000, [2] = month, [3] = day. Reversed byte order for big endian systems
buildTime	uint8_t[4]	Build date, little endian order: [0] = hour, [1] = minute, [2] = second, [3] = millisecond. Reversed byte order for big endian systems
addInfo	char[24]	Additional info

DID_DIAGNOSTIC_MESSAGE

Diagnostic message

`diag_msg_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
messageLength	uint32_t	Message length, including null terminator
message	char[256]	Message data, max size of message is 256

DID_DUAL_IMU_MAG

DID_DUAL_IMU + DID_MAGNETOMETER + MAGNETOMETER_2 Only one of DID_DUAL_IMU_RAW_MAG, DID_DUAL_IMU_MAG, or DID_PREINTEGRATED_IMU_MAG should be streamed simultaneously.

`imu_mag_t`

Field	Type	Description
imu	dual_imu_t	dual imu - raw or pre-integrated depending on data id
mag1	magnetometer_t	mag 1

DID_DUAL_IMU_RAW_MAG

DID_DUAL_IMU_RAW + DID_MAGNETOMETER + MAGNETOMETER_2 Only one of DID_DUAL_IMU_RAW_MAG, DID_DUAL_IMU_MAG, or DID_PREINTEGRATED_IMU_MAG should be streamed simultaneously.

`imu_mag_t`

Field	Type	Description
imu	dual_imu_t	dual imu - raw or pre-integrated depending on data id
mag1	magnetometer_t	mag 1

DID_EVB_DEBUG_ARRAY

`debug_array_t`

Field	Type	Description

DID_EVB_DEV_INFO

EVB device information

`dev_info_t`

Field	Type	Description
reserved	uint32_t	Reserved bits
serialNumber	uint32_t	Serial number
hardwareVer	uint8_t[4]	Hardware version
firmwareVer	uint8_t[4]	Firmware (software) version
buildNumber	uint32_t	Build number
protocolVer	uint8_t[4]	Communications protocol version
repoRevision	uint32_t	Repository revision number
manufacturer	char[24]	Manufacturer name
buildDate	uint8_t[4]	Build date, little endian order: [0] = status ('r'=release, 'd'=debug), [1] = year-2000, [2] = month, [3] = day. Reversed byte order for big endian systems
buildTime	uint8_t[4]	Build date, little endian order: [0] = hour, [1] = minute, [2] = second, [3] = millisecond. Reversed byte order for big endian systems
addInfo	char[24]	Additional info

DID_EVB_RTOS_INFO

EVB-2 RTOS information.

`evb_rtos_info_t`

Field	Type	Description
freeHeapSize	uint32_t	Heap high water mark bytes
mallocSize	uint32_t	Total memory allocated using RTOS pvPortMalloc()
freeSize	uint32_t	Total memory freed using RTOS vPortFree()
task	rtos_task_t[]	Tasks

DID_GROUND_VEHICLE

Static configuration for wheel transform measurements.

`ground_vehicle_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
status	uint32_t	Ground vehicle status flags (eGroundVehicleStatus)
mode	uint32_t	Current mode of the ground vehicle. Use this field to apply commands. (see eGroundVehicleMode)
wheelConfig	wheel_config_t	Wheel transform, track width, and wheel radius.

DID_INFIELD_CAL

Measure and correct IMU calibration error. Estimate INS rotation to align INS with vehicle.

`infield_cal_t`

Field	Type	Description
state	uint32_t	Used to set and monitor the state of the infield calibration system. (see eInfieldCalState)
status	uint32_t	Infield calibration status. (see eInfieldCalStatus)
sampleTimeMs	uint32_t	Number of samples used in IMU average. sampleTimeMs = 0 means "imu" member contains the IMU bias from flash.
imu	imus_t[2]	Dual purpose variable. 1.) This is the averaged IMU sample when sampleTimeMs != 0. 2.) This is a mirror of the motion calibration IMU bias from flash when sampleTimeMs = 0.
calData	infield_cal_vaxis_t[3]	Collected data used to solve for the bias error and INS rotation. Vertical axis: 0 = X, 1 = Y, 2 = Z

DID_INL2_MAG_OBS_INFO

INL2 magnetometer calibration information.

`inl2_mag_obs_info_t`

Field	Type	Description
timeOfWeekMs	uint32_t	Timestamp in milliseconds
Ncal_samples	uint32_t	Number of calibration samples
ready	uint32_t	Data ready to be processed
calibrated	uint32_t	Calibration data present. Set to -1 to force mag recalibration.
auto_recal	uint32_t	Allow mag to auto-recalibrate
outlier	uint32_t	Bad sample data
magHdg	float	Heading from magnetometer
insHdg	float	Heading from INS
magInsHdgDelta	float	Difference between mag heading and (INS heading plus mag declination)
nis	float	Normalized innovation squared (likelihood metric)
nis_threshold	float	Threshold for maximum NIS
Wcal	float[9]	Magnetometer calibration matrix. Must be initialized with a unit matrix, not zeros!
activeCalSet	uint32_t	Active calibration set (0 or 1)
magHdgOffset	float	Offset between magnetometer heading and estimate heading
Tcal	float	Scaled computed variance between calibrated magnetometer samples.
bias_cal	float[3]	Calibrated magnetometer output can be produced using: Bcal = Wcal * (Braw - bias_cal)

DID_INL2_NED_SIGMA

INL2 standard deviation in the NED frame

`inl2_ned_sigma_t`

Field	Type	Description
timeOfWeekMs	unsigned	Timestamp in milliseconds
PxyzNED	float[3]	NED position error sigma
PvelNED	float[3]	NED velocity error sigma
PattNED	float[3]	NED attitude error sigma
PABias	float[3]	Acceleration bias error sigma
PWBias	float[3]	Angular rate bias error sigma
PBaroBias	float	Barometric altitude bias error sigma
PDeclination	float	Mag declination error sigma

DID_INL2_STATES

`inl2_states_t`

Field	Type	Description
timeOfWeek	double	GPS time of week (since Sunday morning) in seconds
qe2b	float[4]	Quaternion body rotation with respect to ECEF
ve	float[3]	(m/s) Velocity in ECEF frame
ecef	double[3]	(m) Position in ECEF frame
biasPqr	float[3]	(rad/s) Gyro bias
biasAcc	float[3]	(m/s^2) Accelerometer bias
biasBaro	float	(m) Barometer bias
magDec	float	(rad) Magnetic declination
magInc	float	(rad) Magnetic inclination

DID_INL2_STATUS

`inl2_status_t`

Field	Type	Description

DID_INTERNAL_DIAGNOSTIC

Internal diagnostic info

`internal_diagnostic_t`

Field	Type	Description
gapCountSerialDriver	uint32_t[6]	Count of gap of more than 0.5 seconds receiving serial data, driver level, one entry for each com port
gapCountSerialParser	uint32_t[6]	Count of gap of more than 0.5 seconds receiving serial data, class / parser level, one entry for each com port
rxOverflowCount	uint32_t[6]	Count of rx overflow, one entry for each com port
txOverflowCount	uint32_t[6]	Count of tx overflow, one entry for each com port
checksumFailCount	uint32_t[6]	Count of checksum failures, one entry for each com port

DID_IO

I/O

`io_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
gpioStatus	uint32_t	General purpose I/O status

DID_MANUFACTURING_INFO

Manufacturing info

`manufacturing_info_t`

Field	Type	Description
serialNumber	uint32_t	Serial number
lotNumber	uint32_t	Lot number
date	char[16]	Manufacturing date (YYYYMMDDHHMMSS)
key	uint32_t	Key

DID_PORT_MONITOR

Data rate and status monitoring for each communications port.

`port_monitor_t`

Field	Type	Description
port	port_monitor_set_t[6]	Port monitor set

DID_POSITION_MEASUREMENT

External position estimate

`pos_measurement_t`

Field	Type	Description
timeOfWeek	double	GPS time of week (since Sunday morning) in seconds
ecef	double[3]	Position in ECEF (earth-centered earth-fixed) frame in meters
psi	float	Heading with respect to NED frame (rad

DID_PREINTEGRATED_IMU_MAG

DID_PREINTEGRATED_IMU + DID_MAGNETOMETER + MAGNETOMETER_2 Only one of DID_DUAL_IMU_RAW_MAG, DID_DUAL_IMU_MAG, or DID_PREINTEGRATED_IMU_MAG should be streamed simultaneously.

`pimu_mag_t`

Field	Type	Description
pimu	preintegrated_imu_t	dual preintegrated imu
mag1	magnetometer_t	mag 1

DID_REFERENCE_IMU

Reference or truth IMU used for manufacturing calibration and testing

`imu_t`

Field	Type	Description
time	double	Time since boot up in seconds. Convert to GPS time of week by adding gps.towOffset
I	imus_t	Inertial Measurement Unit (IMU)

DID_REFERENCE_MAGNETOMETER

Reference or truth magnetometer used for manufacturing calibration and testing `magnetometer_t`

Field	Type	Description
time	double	Time since boot up in seconds. Convert to GPS time of week by adding gps.towOffset
mag	float[3]	Magnetometers in Gauss

DID_ROS_COVARIANCE_POSE_TWIST

INL2 EKF covariances matrix lower diagonals

`ros_covariance_pose_twist_t`

Field	Type	Description
timeOfWeek	double	GPS time of week (since Sunday morning) in seconds
covPoseLD	float[21]	(rad^2, m^2) EKF attitude and position error covariance matrix lower diagonal in body (attitude) and ECEF (position) frames
covTwistLD	float[21]	((m/s)^2, (rad/s)^2) EKF velocity and angular rate error covariance matrix lower diagonal in ECEF (velocity) and body (attitude) frames

DID_RTOS_INFO

RTOS information.

`rtos_info_t`

Field	Type	Description
freeHeapSize	uint32_t	Heap high water mark bytes
mallocSize	uint32_t	Total memory allocated using RTOS pvPortMalloc()
freeSize	uint32_t	Total memory freed using RTOS vPortFree()
task	rtos_task_t[]	Tasks

DID_SCOMP

`sensor_compensation_t`

Field	Type	Description

DID_SENSORS_ADC

`sys_sensors_adc_t`

Field	Type	Description

DID_SENSORS_ADC_SIGMA

`sys_sensors_adc_t`

Field	Type	Description

DID_SENSORS_CAL1

Calibrated IMU1 output. Temperature compensated and motion calibrated.

`sensors_mpu_w_temp_t`

Field	Type	Description
pqr	f_t[3]	(rad/s) Angular rate. Units only apply for calibrated data.
acc	f_t[3]	(m/s^2) Linear acceleration. Units only apply for calibrated data.
mag	f_t[3]	(uT) Magnetometers. Units only apply for calibrated data.
temp	f_t	(°C) Temperature of MPU. Units only apply for calibrated data.

DID_SENSORS_CAL2

Calibrated IMU2 output. Temperature compensated and motion calibrated.

`sensors_mpu_w_temp_t`

Field	Type	Description
pqr	f_t[3]	(rad/s) Angular rate. Units only apply for calibrated data.
acc	f_t[3]	(m/s^2) Linear acceleration. Units only apply for calibrated data.
mag	f_t[3]	(uT) Magnetometers. Units only apply for calibrated data.
temp	f_t	(°C) Temperature of MPU. Units only apply for calibrated data.

DID_SENSORS_IS1

Uncalibrated IMU output. Common scale factor applied to ADC output.

`sensors_w_temp_t`

Field	Type	Description
mpu	sensors_mpu_w_temp_t[2]	Units only apply for calibrated data

DID_SENSORS_IS2

Temperature compensated IMU output.

`sensors_w_temp_t`

Field	Type	Description
mpu	sensors_mpu_w_temp_t[2]	Units only apply for calibrated data

DID_SENSORS_TC_BIAS

`sensors_t`

Field	Type	Description
time	double	Time since boot up in seconds. Convert to GPS time of week by adding gps.towOffset
temp	float	Temperature in Celsius
pqr	float[3]	Gyros in radians / second
acc	float[3]	Accelerometers in meters / second squared
mag	float[3]	Magnetometers in Gauss
bar	float	Barometric pressure in kilopascals
barTemp	float	Temperature of barometric pressure sensor in Celsius
mslBar	float	MSL altitude from barometric pressure sensor in meters
humidity	float	Relative humidity as a percent (%rH). Range is 0% - 100%
vin	float	EVB system input voltage in volts. uINS pin 5 (G2/AN2). Use 10K/1K resistor divider between Vin and GND.
ana1	float	ADC analog input in volts. uINS pin 4, (G1/AN1).
ana3	float	ADC analog input in volts. uINS pin 19 (G3/AN3).
ana4	float	ADC analog input in volts. uINS pin 20 (G4/AN4).

DID_STROBE_IN_TIME

Timestamp for input strobe.

`strobe_in_time_t`

Field	Type	Description
week	uint32_t	GPS number of weeks since January 6th, 1980
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
pin	uint32_t	Strobe input pin
count	uint32_t	Strobe serial index number

DID_SURVEY_IN

Survey in, used to determine position for RTK base station. Base correction output cannot run during a survey and will be automatically disabled if a survey is started.

`survey_in_t`

Field	Type	Description
state	uint32_t	State of current survey, eSurveyInStatus
maxDurationSec	uint32_t	Maximum time (milliseconds) survey will run if minAccuracy is not first achieved. (ignored if 0).
minAccuracy	float	Required horizontal accuracy (m) for survey to complete before maxDuration. (ignored if 0)
elapsedTimeSec	uint32_t	Elapsed time (seconds) of the survey.
hAccuracy	float	Approximate horizontal accuracy of the survey (m).
lla	double[3]	The current surveyed latitude, longitude, altitude (deg, deg, m)

DID_SYS_FAULT

System fault information

`system_fault_t`

Field	Type	Description
status	uint32_t	System fault status
g1Task	uint32_t	Fault Type at HardFault
g2FileNum	uint32_t	Multipurpose register - Line number of fault
g3LineNum	uint32_t	Multipurpose register - File number at fault
g4	uint32_t	Multipurpose register - at time of fault.
g5Lr	uint32_t	Multipurpose register - link register value at time of fault.
pc	uint32_t	Program Counter value at time of fault
psr	uint32_t	Program Status Register value at time of fault

DID_SYS_PARAMS

System parameters / info

`sys_params_t`

Field	Type	Description
timeOfWeekMs	uint32_t	GPS time of week (since Sunday morning) in milliseconds
insStatus	uint32_t	System status 1 flags (eInsStatusFlags)
hdwStatus	uint32_t	System status 2 flags (eHdwStatusFlags)
imuTemp	float	IMU temperature
baroTemp	float	Baro temperature
mcuTemp	float	MCU temperature (not available yet)
reserved1	float	Reserved
imuPeriodMs	uint32_t	IMU sample period in milliseconds. Zero disables sampling.
navPeriodMs	uint32_t	Nav filter update period in milliseconds. Zero disables nav filter.
sensorTruePeriod	double	Actual sample period relative to GPS PPS
reserved2	float[2]	Reserved
genFaultCode	uint32_t	General fault code descriptor (eGenFaultCodes). Set to zero to reset fault code.

DID_SYS_SENSORS_SIGMA

`sys_sensors_t`

Field	Type	Description
time	double	Time since boot up in seconds. Convert to GPS time of week by adding gps.towOffset
temp	float	Temperature in Celsius
pqr	float[3]	Gyros in radians / second
acc	float[3]	Accelerometers in meters / second squared
mag	float[3]	Magnetometers in Gauss
bar	float	Barometric pressure in kilopascals
barTemp	float	Temperature of barometric pressure sensor in Celsius
mslBar	float	MSL altitude from barometric pressure sensor in meters
humidity	float	Relative humidity as a percent (%rH). Range is 0% - 100%
vin	float	EVB system input voltage in volts. uINS pin 5 (G2/AN2). Use 10K/1K resistor divider between Vin and GND.
ana1	float	ADC analog input in volts. uINS pin 4, (G1/AN1).
ana3	float	ADC analog input in volts. uINS pin 19 (G3/AN3).
ana4	float	ADC analog input in volts. uINS pin 20 (G4/AN4).

DID_WHEEL_ENCODER

Wheel encoder data to be fused with GPS-INS measurements, set DID_GROUND_VEHICLE for configuration before sending this message `wheel_encoder_t`

Field	Type	Description
timeOfWeek	double	Time of measurement wrt current week
status	uint32_t	Status Word
theta_l	float	Left wheel angle (rad)
theta_r	float	Right wheel angle (rad)
omega_l	float	Left wheel angular rate (rad/s)
omega_r	float	Right wheel angular rate (rad/s)
wrap_count_l	uint32_t	Left wheel revolution count
wrap_count_r	uint32_t	Right wheel revolution count

7.2.2 Enumerations and Defines

System status and configuration is made available through various enumeration and #defines.

General

DID_EVB_FLASH_CFG.CBPRESET

(eEvb2ComBridgePreset)

Field	Value
EVB2_CB_PRESET_NA	0
EVB2_CB_PRESET_ALL_OFF	1
EVB2_CB_PRESET_RS232	2
EVB2_CB_PRESET_RS232_XBEE	3
EVB2_CB_PRESET_RS422_WIFI	4
EVB2_CB_PRESET_SPI_RS232	5
EVB2_CB_PRESET_USB_HUB_RS232	6
EVB2_CB_PRESET_USB_HUB_RS422	7
EVB2_CB_PRESET_COUNT	8

DID_EVB_FLASH_CFG.PORTOPTIONS

(eEvb2PortOptions)

Field	Value
EVB2_PORT_OPTIONS_RADIO_RTK_FILTER	0x00000001
EVB2_PORT_OPTIONS_DEFAULT	EVB2_PORT_OPTIONS_RADIO_RTK_FILTER

DID_EVB_STATUS.LOGGERMODE

(eEvb2LoggerMode)

Field	Value
EVB2_LOG_NA	0
EVB2_LOG_CMD_START	2
EVB2_LOG_CMD_STOP	4
EVB2_LOG_CMD_PURGE	1002

DID_FLASH_CONFIG.GNSSSATSIGCONST

(eGnssSatSigConst)

Field	Value
GNSS_SAT_SIG_CONST_GPS	0x0003
GNSS_SAT_SIG_CONST_QZSS	0x000C
GNSS_SAT_SIG_CONST_GAL	0x0030
GNSS_SAT_SIG_CONST_BDS	0x00C0
GNSS_SAT_SIG_CONST_GLO	0x0300
GNSS_SAT_SIG_CONST_SBAS	0x1000

DID_FLASH_CONFIG.SENSORCONFIG

(eSensorConfig)

Field	**Value**
SENSOR_CFG_GYR_FS_250	0x00000000
SENSOR_CFG_GYR_FS_500	0x00000001
SENSOR_CFG_GYR_FS_1000	0x00000002
SENSOR_CFG_GYR_FS_2000	0x00000003
SENSOR_CFG_GYR_FS_MASK	0x00000003
SENSOR_CFG_GYR_FS_OFFSET	(int)0
SENSOR_CFG_ACC_FS_2G	0x00000000
SENSOR_CFG_ACC_FS_4G	0x00000001
SENSOR_CFG_ACC_FS_8G	0x00000002
SENSOR_CFG_ACC_FS_16G	0x00000003
SENSOR_CFG_ACC_FS_MASK	0x00000003
SENSOR_CFG_ACC_FS_OFFSET	(int)2
SENSOR_CFG_GYR_DLPF_250HZ	0x00000000
SENSOR_CFG_GYR_DLPF_184HZ	0x00000001
SENSOR_CFG_GYR_DLPF_92HZ	0x00000002
SENSOR_CFG_GYR_DLPF_41HZ	0x00000003
SENSOR_CFG_GYR_DLPF_20HZ	0x00000004
SENSOR_CFG_GYR_DLPF_10HZ	0x00000005
SENSOR_CFG_GYR_DLPF_5HZ	0x00000006
SENSOR_CFG_GYR_DLPF_MASK	0x0000000F
SENSOR_CFG_GYR_DLPF_OFFSET	(int)8
SENSOR_CFG_ACC_DLPF_218HZ	0x00000000
SENSOR_CFG_ACC_DLPF_218HZb	0x00000001
SENSOR_CFG_ACC_DLPF_99HZ	0x00000002
SENSOR_CFG_ACC_DLPF_45HZ	0x00000003
SENSOR_CFG_ACC_DLPF_21HZ	0x00000004
SENSOR_CFG_ACC_DLPF_10HZ	0x00000005
SENSOR_CFG_ACC_DLPF_5HZ	0x00000006
SENSOR_CFG_ACC_DLPF_MASK	0x0000000F
SENSOR_CFG_ACC_DLPF_OFFSET	(int)12
SENSOR_CFG_SENSOR_ROTATION_MASK	0x000000FF
SENSOR_CFG_SENSOR_ROTATION_OFFSET	(int)16
SENSOR_CFG_SENSOR_ROTATION_0_0_0	(int)0
SENSOR_CFG_SENSOR_ROTATION_0_0_90	(int)1
SENSOR_CFG_SENSOR_ROTATION_0_0_180	(int)2

Field	Value
SENSOR_CFG_SENSOR_ROTATION_0_0_N90	(int)3
SENSOR_CFG_SENSOR_ROTATION_90_0_0	(int)4
SENSOR_CFG_SENSOR_ROTATION_90_0_90	(int)5
SENSOR_CFG_SENSOR_ROTATION_90_0_180	(int)6
SENSOR_CFG_SENSOR_ROTATION_90_0_N90	(int)7
SENSOR_CFG_SENSOR_ROTATION_180_0_0	(int)8
SENSOR_CFG_SENSOR_ROTATION_180_0_90	(int)9
SENSOR_CFG_SENSOR_ROTATION_180_0_180	(int)10
SENSOR_CFG_SENSOR_ROTATION_180_0_N90	(int)11
SENSOR_CFG_SENSOR_ROTATION_N90_0_0	(int)12
SENSOR_CFG_SENSOR_ROTATION_N90_0_90	(int)13
SENSOR_CFG_SENSOR_ROTATION_N90_0_180	(int)14
SENSOR_CFG_SENSOR_ROTATION_N90_0_N90	(int)15
SENSOR_CFG_SENSOR_ROTATION_0_90_0	(int)16
SENSOR_CFG_SENSOR_ROTATION_0_90_90	(int)17
SENSOR_CFG_SENSOR_ROTATION_0_90_180	(int)18
SENSOR_CFG_SENSOR_ROTATION_0_90_N90	(int)19
SENSOR_CFG_SENSOR_ROTATION_0_N90_0	(int)20
SENSOR_CFG_SENSOR_ROTATION_0_N90_90	(int)21
SENSOR_CFG_SENSOR_ROTATION_0_N90_180	(int)22
SENSOR_CFG_SENSOR_ROTATION_0_N90_N90	(int)23

DID_SYS_CMD.COMMAND

(eSystemCommand)

Field	Value
SYS_CMD_SAVE_PERSISTENT_MESSAGES	1
SYS_CMD_ENABLE_BOOTLOADER_AND_RESET	2
SYS_CMD_ENABLE_SENSOR_STATS	3
SYS_CMD_ENABLE_RTOS_STATS	4
SYS_CMD_ZERO_MOTION	5
SYS_CMD_ENABLE_GPS_LOW_LEVEL_CONFIG	10
SYS_CMD_SAVE_FLASH	97
SYS_CMD_SAVE_GPS_ASSIST_TO_FLASH_RESET	98
SYS_CMD_SOFTWARE_RESET	99
SYS_CMD_MANF_UNLOCK	1122334455
SYS_CMD_MANF_FACTORY_RESET	1357924680
SYS_CMD_MANF_CHIP_ERASE	1357924681

DID_SYS_PARAMS.GENFAULTCODE

(eGenFaultCodes)

Field	Value
GFC_INS_STATE_ORUN_UVW	0x00000001
GFC_INS_STATE_ORUN_LAT	0x00000002
GFC_INS_STATE_ORUN_ALT	0x00000004
GFC_UNHANDLED_INTERRUPT	0x00000010
GFC_INIT_SENSORS	0x00000100
GFC_INIT_SPI	0x00000200
GFC_CONFIG_SPI	0x00000400
GFC_INIT_GPS1	0x00000800
GFC_INIT_GPS2	0x00001000
GFC_FLASH_INVALID_VALUES	0x00002000
GFC_FLASH_CHECKSUM_FAILURE	0x00004000
GFC_FLASH_WRITE_FAILURE	0x00008000
GFC_SYS_FAULT_GENERAL	0x00010000
GFC_SYS_FAULT_CRITICAL	0x00020000
GFC_SENSOR_SATURATION	0x00040000

GNSS SATELLITE FLAGS

(eSatSvFlags)

Field	Value
SAT_SV_FLAGS_QUALITYIND_MASK	0x00000007
SAT_SV_FLAGS_SV_USED	0x00000008
SAT_SV_FLAGS_HEALTH_MASK	0x00000030
NAV_SAT_FLAGS_HEALTH_OFFSET	4
SAT_SV_FLAGS_DIFFCORR	0x00000040
SAT_SV_FLAGS_SMOOTHED	0x00000080
SAT_SV_FLAGS_ORBITSOURCE_MASK	0x00000700
SAT_SV_FLAGS_ORBITSOURCE_OFFSET	8
SAT_SV_FLAGS_EPHAVAIL	0x00000800
SAT_SV_FLAGS_ALMAVAIL	0x00001000
SAT_SV_FLAGS_ANOAVAIL	0x00002000
SAT_SV_FLAGS_AOPAVAIL	0x00004000
SAT_SV_FLAGS_RTK_SOL_FIX_STATUS_MASK	0x03000000
SAT_SV_FLAGS_RTK_SOL_FIX_STATUS_OFFSET	24
SAT_SV_FLAGS_RTK_SOL_FIX_STATUS_FLOAT	1
SAT_SV_FLAGS_RTK_SOL_FIX_STATUS_FIX	2
SAT_SV_FLAGS_RTK_SOL_FIX_STATUS_HOLD	3

GPS NAVIGATION FIX TYPE

(eGpsNavFixStatus)

Field	Value
GPS_NAV_FIX_NONE	0x00000000
GPS_NAV_FIX_POSITIONING_3D	0x00000001
GPS_NAV_FIX_POSITIONING_RTK_FLOAT	0x00000002
GPS_NAV_FIX_POSITIONING_RTK_FIX	0x00000003

GPS STATUS

(eGpsStatus)

Field	Value
GPS_STATUS_NUM_SATS_USED_MASK	0x000000FF
GPS_STATUS_FIX_NONE	0x00000000
GPS_STATUS_FIX_DEAD_RECKONING_ONLY	0x00000100
GPS_STATUS_FIX_2D	0x00000200
GPS_STATUS_FIX_3D	0x00000300
GPS_STATUS_FIX_GPS_PLUS_DEAD_RECK	0x00000400
GPS_STATUS_FIX_TIME_ONLY	0x00000500
GPS_STATUS_FIX_UNUSED1	0x00000600
GPS_STATUS_FIX_UNUSED2	0x00000700
GPS_STATUS_FIX_DGPS	0x00000800
GPS_STATUS_FIX_SBAS	0x00000900
GPS_STATUS_FIX_RTK_SINGLE	0x00000A00
GPS_STATUS_FIX_RTK_FLOAT	0x00000B00
GPS_STATUS_FIX_RTK_FIX	0x00000C00
GPS_STATUS_FIX_MASK	0x00001F00
GPS_STATUS_FIX_BIT_OFFSET	(int)8
GPS_STATUS_FLAGS_FIX_OK	0x00010000
GPS_STATUS_FLAGS_DGPS_USED	0x00020000
GPS_STATUS_FLAGS_RTK_FIX_AND_HOLD	0x00040000
GPS_STATUS_FLAGS_WEEK_VALID	0x00040000
GPS_STATUS_FLAGS_TOW_VALID	0x00080000
GPS_STATUS_FLAGS_RTK_POSITION_ENABLED	0x00100000
GPS_STATUS_FLAGS_STATIC_MODE	0x00200000
GPS_STATUS_FLAGS_RTK_COMPASSING_ENABLED	0x00400000
GPS_STATUS_FLAGS_RTK_RAW_GPS_DATA_ERROR	0x00800000
GPS_STATUS_FLAGS_RTK_BASE_DATA_MISSING	0x01000000
GPS_STATUS_FLAGS_RTK_BASE_POSITION_MOVING	0x02000000
GPS_STATUS_FLAGS_RTK_BASE_POSITION_INVALID	0x03000000
GPS_STATUS_FLAGS_RTK_BASE_POSITION_MASK	0x03000000
GPS_STATUS_FLAGS_RTK_POSITION_VALID	0x04000000
GPS_STATUS_FLAGS_RTK_COMPASSING_VALID	0x08000000
GPS_STATUS_FLAGS_RTK_COMPASSING_BASELINE_BAD	0x00002000
GPS_STATUS_FLAGS_RTK_COMPASSING_BASELINE_UNSET	0x00004000
GPS_STATUS_FLAGS_GPS_NMEA_DATA	0x00008000
GPS_STATUS_FLAGS_MASK	0x0FFFE000

Field	Value
GPS_STATUS_FLAGS_BIT_OFFSET	(int)16

HARDWARE STATUS FLAGS

(eHdwStatusFlags)

Field	Value
HDW_STATUS_MOTION_GYR_SIG	0x00000001
HDW_STATUS_MOTION_ACC_SIG	0x00000002
HDW_STATUS_MOTION_SIG_MASK	0x00000003
HDW_STATUS_MOTION_GYR_DEV	0x00000004
HDW_STATUS_MOTION_ACC_DEV	0x00000008
HDW_STATUS_MOTION_MASK	0x0000000F
HDW_STATUS_GPS_SATELLITE_RX	0x00000010
HDW_STATUS_STROBE_IN_EVENT	0x00000020
HDW_STATUS_GPS_TIME_OF_WEEK_VALID	0x00000040
HDW_STATUS_UNUSED_1	0x00000080
HDW_STATUS_SATURATION_GYR	0x00000100
HDW_STATUS_SATURATION_ACC	0x00000200
HDW_STATUS_SATURATION_MAG	0x00000400
HDW_STATUS_SATURATION_BARO	0x00000800
HDW_STATUS_SATURATION_MASK	0x00000F00
HDW_STATUS_SATURATION_OFFSET	8
HDW_STATUS_SYSTEM_RESET_REQUIRED	0x00001000
HDW_STATUS_UNUSED_3	0x00002000
HDW_STATUS_UNUSED_4	0x00004000
HDW_STATUS_UNUSED_5	0x00008000
HDW_STATUS_ERR_COM_TX_LIMITED	0x00010000
HDW_STATUS_ERR_COM_RX_OVERRUN	0x00020000
HDW_STATUS_COM_PARSE_ERR_COUNT_MASK	0x00F00000
HDW_STATUS_COM_PARSE_ERR_COUNT_OFFSET	20
HDW_STATUS_BIT_RUNNING	0x01000000
HDW_STATUS_BIT_PASSED	0x02000000
HDW_STATUS_BIT_FAULT	0x03000000
HDW_STATUS_BIT_MASK	0x03000000
HDW_STATUS_ERR_TEMPERATURE	0x04000000
HDW_STATUS_UNSUED_6	0x08000000
HDW_STATUS_FAULT_RESET_MASK	0x70000000
HDW_STATUS_FAULT_RESET_BACKUP_MODE	0x10000000
HDW_STATUS_FAULT_RESET_WATCHDOG	0x20000000
HDW_STATUS_FAULT_RESET_SOFT	0x30000000
HDW_STATUS_FAULT_RESET_HDW	0x40000000

Field	Value
HDW_STATUS_FAULT_SYS_CRITICAL	0x80000000

IMU STATUS

(eImuStatus)

Field	Value
IMU_STATUS_SATURATION_IMU1_GYR	0x00000001
IMU_STATUS_SATURATION_IMU2_GYR	0x00000002
IMU_STATUS_SATURATION_IMU1_ACC	0x00000004
IMU_STATUS_SATURATION_IMU2_ACC	0x00000008
IMU_STATUS_RESERVED1	0x00000020
IMU_STATUS_RESERVED2	0x00000040
IMU_STATUS_SATURATION_HISTORY	0x00000100
IMU_STATUS_SAMPLE_RATE_FAULT_HISTORY	0x00000200

INS STATUS FLAGS

(eInsStatusFlags)

Field	Value
INS_STATUS_ATT_ALIGN_COARSE	0x00000001
INS_STATUS_VEL_ALIGN_COARSE	0x00000002
INS_STATUS_POS_ALIGN_COARSE	0x00000004
INS_STATUS_ALIGN_COARSE_MASK	0x00000007
INS_STATUS_WHEEL_AIDING_VEL	0x00000008
INS_STATUS_ATT_ALIGN_FINE	0x00000010
INS_STATUS_VEL_ALIGN_FINE	0x00000020
INS_STATUS_POS_ALIGN_FINE	0x00000040
INS_STATUS_ALIGN_FINE_MASK	0x00000070
INS_STATUS_GPS_AIDING_HEADING	0x00000080
INS_STATUS_GPS_AIDING_POS	0x00000100
INS_STATUS_GPS_UPDATE_IN_SOLUTION	0x00000200
INS_STATUS_RESERVED_1	0x00000400
INS_STATUS_MAG_AIDING_HEADING	0x00000800
INS_STATUS_NAV_MODE	0x00001000
INS_STATUS_DO_NOT_MOVE	0x00002000
INS_STATUS_GPS_AIDING_VEL	0x00004000
INS_STATUS_KINEMATIC_CAL_GOOD	0x00008000
INS_STATUS_SOLUTION_MASK	0x000F0000
INS_STATUS_SOLUTION_OFFSET	16
INS_STATUS_SOLUTION_OFF	0
INS_STATUS_SOLUTION_ALIGNING	1
INS_STATUS_SOLUTION_ALIGNMENT_COMPLETE	2
INS_STATUS_SOLUTION_NAV	3
INS_STATUS_SOLUTION_NAV_HIGH_VARIANCE	4
INS_STATUS_SOLUTION_AHRS	5
INS_STATUS_SOLUTION_AHRS_HIGH_VARIANCE	6
INS_STATUS_RTK_COMPASSING_BASELINE_UNSET	0x00100000
INS_STATUS_RTK_COMPASSING_BASELINE_BAD	0x00200000
INS_STATUS_RTK_COMPASSING_MASK	(INS_STATUS_RTK_COMPASSING_BASELINE_UNSET| INS_STATUS_RTK_COMPASSING_BASELINE_BAD)
INS_STATUS_MAG_RECALIBRATING	0x00400000
INS_STATUS_MAG_INTERFERENCE_OR_BAD_CAL	0x00800000
INS_STATUS_GPS_NAV_FIX_MASK	0x03000000
INS_STATUS_GPS_NAV_FIX_OFFSET	24
INS_STATUS_RTK_COMPASSING_VALID	0x04000000

Field	Value
INS_STATUS_RTK_RAW_GPS_DATA_ERROR	0x08000000
INS_STATUS_RTK_ERR_BASE_DATA_MISSING	0x10000000
INS_STATUS_RTK_ERR_BASE_POSITION_MOVING	0x20000000
INS_STATUS_RTK_ERR_BASE_POSITION_INVALID	0x30000000
INS_STATUS_RTK_ERR_BASE_MASK	0x30000000
INS_STATUS_RTK_ERROR_MASK	(INS_STATUS_RTK_RAW_GPS_DATA_ERROR| INS_STATUS_RTK_ERR_BASE_MASK)
INS_STATUS_RTOS_TASK_PERIOD_OVERRUN	0x40000000
INS_STATUS_GENERAL_FAULT	0x80000000

MAGNETOMETER RECALIBRATION MODE

(eMagRecalMode)

Field	Value
MAG_RECAL_CMD_DO_NOTHING	(int)0
MAG_RECAL_CMD_MULTI_AXIS	(int)1
MAG_RECAL_CMD_SINGLE_AXIS	(int)2
MAG_RECAL_CMD_ABORT	(int)101

RTK CONFIGURATION

(eRTKConfigBits)

Field	Value
RTK_CFG_BITS_ROVER_MODE_RTK_POSITIONING	0x00000001
RTK_CFG_BITS_ROVER_MODE_RTK_POSITIONING_EXTERNAL	0x00000002
RTK_CFG_BITS_ROVER_MODE_RTK_COMPASSING_F9P	0x00000004
RTK_CFG_BITS_ROVER_MODE_RTK_COMPASSING	0x00000008
RTK_CFG_BITS_ROVER_MODE_RTK_POSITIONING_MASK	(RTK_CFG_BITS_ROVER_MODE_RTK_POSITIONING| RTK_CFG_BITS_ROVER_MODE_RTK_POSITIONING_EXTERNAL)
RTK_CFG_BITS_ROVER_MODE_RTK_COMPASSING_MASK	(RTK_CFG_BITS_ROVER_MODE_RTK_COMPASSING| RTK_CFG_BITS_ROVER_MODE_RTK_COMPASSING_F9P)
RTK_CFG_BITS_ROVER_MODE_MASK	0x0000000F
RTK_CFG_BITS_BASE_OUTPUT_GPS1_UBLOX_SER0	0x00000010
RTK_CFG_BITS_BASE_OUTPUT_GPS1_UBLOX_SER1	0x00000020
RTK_CFG_BITS_BASE_OUTPUT_GPS1_UBLOX_SER2	0x00000040
RTK_CFG_BITS_BASE_OUTPUT_GPS1_UBLOX_USB	0x00000080
RTK_CFG_BITS_BASE_OUTPUT_GPS1_RTCM3_SER0	0x00000100
RTK_CFG_BITS_BASE_OUTPUT_GPS1_RTCM3_SER1	0x00000200
RTK_CFG_BITS_BASE_OUTPUT_GPS1_RTCM3_SER2	0x00000400
RTK_CFG_BITS_BASE_OUTPUT_GPS1_RTCM3_USB	0x00000800
RTK_CFG_BITS_BASE_OUTPUT_GPS2_UBLOX_SER0	0x00001000
RTK_CFG_BITS_BASE_OUTPUT_GPS2_UBLOX_SER1	0x00002000
RTK_CFG_BITS_BASE_OUTPUT_GPS2_UBLOX_SER2	0x00004000
RTK_CFG_BITS_BASE_OUTPUT_GPS2_UBLOX_USB	0x00008000
RTK_CFG_BITS_BASE_OUTPUT_GPS2_RTCM3_SER0	0x00010000
RTK_CFG_BITS_BASE_OUTPUT_GPS2_RTCM3_SER1	0x00020000
RTK_CFG_BITS_BASE_OUTPUT_GPS2_RTCM3_SER2	0x00040000
RTK_CFG_BITS_BASE_OUTPUT_GPS2_RTCM3_USB	0x00080000
RTK_CFG_BITS_BASE_POS_MOVING	0x00100000
RTK_CFG_BITS_RESERVED1	0x00200000
RTK_CFG_BITS_RTK_BASE_IS_IDENTICAL_TO_ROVER	0x00400000
RTK_CFG_BITS_GPS_PORT_PASS_THROUGH	0x00800000
RTK_CFG_BITS_ROVER_MODE_ONBOARD_MASK	(RTK_CFG_BITS_ROVER_MODE_RTK_POSITIONING| RTK_CFG_BITS_ROVER_MODE_RTK_COMPASSING)
RTK_CFG_BITS_ALL_MODES_MASK	(RTK_CFG_BITS_ROVER_MODE_MASK| RTK_CFG_BITS_BASE_MODE)

SYSTEM CONFIGURATION

(eSysConfigBits)

Field	Value
UNUSED1	0x00000001
UNUSED2	0x00000002
SYS_CFG_BITS_AUTO_MAG_RECAL	0x00000004
SYS_CFG_BITS_DISABLE_MAG_DECL_ESTIMATION	0x00000008
SYS_CFG_BITS_DISABLE_LEDS	0x00000010
Magnetometer	multi-axis
SYS_CFG_BITS_MAG_RECAL_MODE_MASK	0x00000700
SYS_CFG_BITS_MAG_RECAL_MODE_OFFSET	8
SYS_CFG_BITS_DISABLE_MAGNETOMETER_FUSION	0x00001000
SYS_CFG_BITS_DISABLE_BAROMETER_FUSION	0x00002000
SYS_CFG_BITS_DISABLE_GPS1_FUSION	0x00004000
SYS_CFG_BITS_DISABLE_GPS2_FUSION	0x00008000
SYS_CFG_BITS_DISABLE_AUTO_ZERO_VELOCITY_UPDATES	0x00010000
SYS_CFG_BITS_DISABLE_AUTO_ZERO_ANGULAR_RATE_UPDATES	0x00020000
SYS_CFG_BITS_DISABLE_INS_EKF	0x00040000
SYS_CFG_BITS_DISABLE_AUTO_BIT_ON_STARTUP	0x00080000
SYS_CFG_BITS_DISABLE_WHEEL_ENCODER_FUSION	0x00100000
SYS_CFG_BITS_DISABLE_PACKET_ENCODING	0x00400000

7.3 ASCII (NMEA) Protocol

For simple use, the Inertial Sense device supports ASCII based communications protocol, including several common GPS NMEA messages. The ASCII protocol is human readable from in a command line terminal but is less optimal than the binary protocol.

7.3.1 Communications Examples

The ASCII Communications Example Project demonstrates how to implement the ASCII (NMEA) protocol.

7.3.2 Packet Structure

The Inertial Sense ASCII protocol follows the standard NMEA 0183 message structure:

- 1 byte – Start packet, always the $ byte (`0x24`)
- n bytes (usually 4 or 5) – packet identifier
- 1 byte – a comma (`0x2C`)
- n bytes – comma separated list of data, i.e. 1,2,3,4,5,6
- 1 byte – checksum marker, always the * byte (`0x2A`)
- 2 bytes – checksum in hex format (i.e. `f5` or `0a`), 0 padded if necessary and lowercase
- 2 bytes – End packet, always carriage return and line feed (`\r\n` or `0x0D` , `0x0A`)

The packet checksum is an 8 bit integer and is calculated by calculating the exclusive OR of all bytes in between and not including the $ and * bytes. The packet checksum byte is converted to a 2 byte ASCII hex code, and left padded with 0 if necessary to ensure that it is always 2 bytes. The checksum is always lowercase hexadecimal characters. See NMEA 0183 message structure for more details. The NMEA string checksum is automatically computed and appended to string when using the InertialSense SDK serialPortWriteAscii function or can be generated using an online checksum calculator. For example: MTK NMEA checksum calculator

7.3.3 Persistent Messages

The *persistent messages* option saves the current data stream configuration to flash memory for use following reboot, eliminating the need to re-enable messages following a reset or power cycle.

- **To save current ASCII persistent messages** - send the save persistent messages command.
- **To disable persistent messages** - send the stop all broadcasts followed by save persistent messages.

Binary persistent messages are also available.

Enabling Persistent Messages - EvalTool

To enable persistent ASCII messages using the EvalTool:

- Enable the desired ASCII messages in the EvalTool "Data Sets" tab. Select DID_ASCII_BCAST_PERIOD in the DID menu and set the desired ASCII messages period to a non-zero value.
- Press the "Save Persistent" button in the EvalTool "Data Logs" tab to store the current message configuration to flash memory.
- Reset the uINS and verify the messages are automatically streaming. You can use a generic serial port program like putty or the EvalTool->Data Logs->Data Log->Messages dialog to view the ASCII messages.

To disable all persistent messages using the EvalTool, click the "Stop Streaming" button and then "Save Persistent" button.

7.3.4 ASCII Input Messages

The following ASCII messages can be received by the uINS.

Message	Description
`$ASCB*13\r\n`	Query the broadcast rate of ASCII output messages.
ASCB	Set the broadcast rate of ASCII output messages.
`$INFO*0E\r\n`	Query device information.
`$PERS*13\r\n`	Save persistent message to flash.
`$STPB*15\r\n`	Stop broadcast of all messages (ASCII and binary) on all ports.
`$STPC*14\r\n`	Stop broadcast of all messages (ASCII and binary) on current port.

ASCB

Enable ASCII message and set broadcast periods. The period is in milliseconds with no thousands separator character. “xx” is the two-character checksum. Each field can be left blank in which case the existing broadcast period for that field is not modified, or 0 to disable.

```
$ASCB,d,d,d,d,d,d,d,d,d,d,d,d,d*xx\r\n
      1 2 3 4 5 6 7 8 9 0 1 2 3
```

Index	Field	Units	Description
1	options		Port selection. Combine by adding options together: 0=current, 1=ser0, 2=ser1, 4=ser2, 8=USB, 512=persistent (remember after reset)
2	PIMU	ms	Broadcast period for PIMU dual IMU message.
3	PPIMU	ms	Broadcast period for PPIMU preintegrated dual IMU message.
4	PINS1	ms	Broadcast period for PINS1 INS output (euler, NED) message.
5	PINS2	ms	Broadcast period for PINS2 INS outpout (quaterion, LLA) message.
6	PGPSP	ms	Broadcast period for PGPSP GPS position message.
7	-	ms	Reserved. Leave zero.
8	GPGGA	ms	Broadcast period for NMEA GPGGA (fix, 3D location, and accuracy) message.
9	GPGLL	ms	Broadcast period for NMEA GPGLL (2D location and time) message.
10	GPGSA	ms	Broadcast period for NMEA GPGSA (DOP and active satellites) message.
11	GPRMC	ms	Broadcast period for NMEA GPRMC (minimum specific GPS/ Transit) message.
12	GPZDA	ms	Broadcast period for NMEA GPZDA (UTC Time/Date) message.
13	PASHR	ms	Broadcast period for NMEA PASHR (euler) message.

PERS

Send this command to save current *persistent messages* to flash memory for use following reboot. This eliminates the need to re-enable messages following a reset or power cycle. In order to disable persistent messages, all messages must be disabled and then the 'save persistent messages' command should be sent.

```
$PERS*14\r\n
```

STPB

Stop all broadcasts (both binary and ASCII) on all ports by sending the following packet:

```
$STPB*15\r\n
```

The hexadecimal equivalent is:

```
24 53 54 50 42 2A 31 35 0D 0A
```

STPC

Stop all broadcasts (both binary and ASCII) on the current port by sending the following packet:

```
$STPC*14\r\n
```

The hexadecimal equivalent is:

```
24 53 54 50 43 2A 31 34 0D 0A
```

7.3.5 ASCII Output Messages

The following ASCII messages can be sent by the uINS.

Message	Description
ASCB	Broadcast rate of ASCII output messages.
PIMU	Dual IMU data (two sets of 3-axis gyros and accelerometers) in the body frame.
PPIMU	Preintegrated dual IMU: delta theta (rad) and delta velocity (m/s).
PINS1	INS output: euler rotation w/ respect to NED, NED position from reference LLA.
PINS2	INS output: quaternion rotation w/ respect to NED, ellipsoid altitude.
PGPSP	GPS position data.
GPGGA	NMEA GPGGA GPS 3D location, fix, and accuracy.
GPGLL	NMEA GPGLL GPS 2D location and time.
GPGSA	NMEA GSA GPS DOP and active satellites.
GPRMC	NMEA Recommended minimum specific GPS/Transit data.
GPZDA	NMEA UTC Time/Date message.
PASHR	NMEA PASHR (euler) message.
PSTRB	Strobe event input time.
INFO	Device information.

The field codes used in the message descriptions are: lf = double, f = float, d = int.

PIMU

Dual IMU sensor data (two sets of 3-axis gyros and accelerometers) in the body frame.

```
$PIMU,lf,f,f,f,f,f,f,f,f,f,f,f,f*xx\r\n
       1 2 3 4 5 6 7 8 9 0 1 2 3
```

Index	Field	Units	Description
1	time	sec	Time since system power up
2	IMU1 pqr[0]	rad/sec	IMU1 angular rate gyro – pitch
3	IMU1 pqr[1]	rad/sec	IMU1 angular rate gyro – roll
4	IMU1 pqr[2]	rad/sec	IMU1 angular rate gyro – yaw
5	IMU1 acc[0]	m/s2	IMU1 linear acceleration – X
6	IMU1 acc[1]	m/s2	IMU1 linear acceleration – Y
7	IMU1 acc[2]	m/s2	IMU1 linear acceleration – Z
8	IMU2 pqr[0]	rad/sec	IMU2 angular rate gyro – pitch
9	IMU2 pqr[1]	rad/sec	IMU2 angular rate gyro – roll
10	IMU2 pqr[2]	rad/sec	IMU2 angular rate gyro – yaw
11	IMU2 acc[0]	m/s2	IMU2 linear acceleration – X
12	IMU2 acc[1]	m/s2	IMU2 linear acceleration – Y
13	IMU2 acc[2]	m/s2	IMU2 linear acceleration – Z

PPIMU

Preintegrated dual inertial measurement unit (IMU) sensor data, delta theta in radians and delta velocity in m/s in the body frame. Also known as conning and sculling integrals.

```
$PPIMU,lf,f,f,f,f,f,f,f,f,f,f,f,f,f*xx\r\n
        1 2 3 4 5 6 7 8 9 0 1 2 3 4
```

Index	Field	Units	Description
1	time	sec	Time since system power up
2	theta1[0]	rad	IMU 1 delta theta integral – roll
3	theta1[1]	rad	IMU 1 delta theta integral – pitch
4	theta1[2]	rad	IMU 1 delta theta integral – yaw
5	theta2[0]	rad	IMU 2 delta theta integral – roll
6	theta2[1]	rad	IMU 2 delta theta integral – pitch
7	theta2[2]	rad	IMU 2 delta theta integral – yaw
8	vel1[0]	m/s	IMU 1 delta velocity integral – X
9	vel1[1]	m/s	IMU 1 delta velocity integral – Y
10	vel1[2]	m/s	IMU 1 delta velocity integral – Z
11	vel2[0]	m/s	IMU 2 delta velocity integral – X
12	vel2[1]	m/s	IMU 2 delta velocity integral – Y
13	vel2[2]	m/s	IMU 2 delta velocity integral – Z
14	dt	s	Integral period for delta theta vel

PINS1

INS output with Euler angles and NED offset from the reference LLA.

```
$PINS1,lf,d,d,d,f,f,f,f,f,f,lf,lf,lf,f,f,f*xx\r\n
        1 2 3 4 5 6 7 8 9 0  1  2  3 4 5 6
```

Index	Field	Units	Description
1	timeOfWeek	sec	Seconds since Sunday morning in GMT
2	GPS week	weeks	Number of weeks since January 1st of 1980 in GMT
3	insStatus		INS Status Flags
4	hdwStatus		Hardware Status Flags
5	theta[0]	rad	Euler angle – roll
6	theta[1]	rad	Euler angle – pitch
7	theta[2]	rad	Euler angle – yaw
8	UVW[0]	m/s	Velocity in body frame – X
9	UVW[1]	m/s	Velocity in body frame – Y
10	UVW[2]	m/s	Velocity in body frame – Z
11	Latitude	deg	WGS84 Latitude
12	Longitude	deg	WGS84 Longitude
13	HAE Altitude	m	Height above ellipsoid (vertical elevation)
14	NED[0]	m	Offset from reference LLA – North
15	NED[1]	m	Offset from reference LLA – East
16	NED[2]	m	Offset from reference LLA – Down

PINS2

INS output with quaternion attitude.

```
$PINS2,lf,d,d,d,f,f,f,f,f,f,f,lf,lf,lf*xx\r\n
       1 2 3 4 5 6 7 8 9 0 1  2  3  4
```

Index	Field	Units	Description
1	timeOfWeek	sec	Seconds since Sunday morning in GMT
2	GPS week	weeks	Number of weeks since January 1st of 1980 in GMT
3	insStatus		INS Status Flags
4	hdwStatus		Hardware Status Flags
5	qn2b[0]		Quaternion rotation (NED to body) – W
6	qn2b[1]		Quaternion rotation (NED to body) – X
7	qn2b[2]		Quaternion rotation (NED to body) – Y
8	qn2b[3]		Quaternion rotation (NED to body) – Z
9	UVW[0]	m/s	Velocity in body frame – X
10	UVW[1]	m/s	Velocity in body frame – Y
11	UVW[2]	m/s	Velocity in body frame – Z
12	Latitude	deg	WGS84 Latitude
13	Longitude	deg	WGS84 Longitude
14	HAE altitude	m	Height above ellipsoid (vertical elevation)

PGPSP

GPS navigation data.

```
$PGPSP,d,d,d,lf,lf,lf,f,f,f,f,f,f,f,f,f,f*xx\r\n
      1 2 3  4  5  6 7 8 9 0 1 2 3 4 5 6
```

```
$PGPSP,337272200,2031,1075643160,40.33057800,-111.72581630,1406.39,1425.18,0.95,0.37,0.55,-0.02,0.02,-0.03,0.17,39.5,337182.4521*4d
```

Index	Field	Units	Description
1	timeMsOfWeek	ms	Milliseconds since Sunday morning in GMT
2	GPS week	weeks	Number of weeks since January 1st of 1980 in GMT
3	status		(see eGpsStatus) GPS status: [0x000000xx] number of satellites used, [0x0000xx00] fix type, [0x00xx0000] status flags
4	Latitude	deg	WGS84 Latitude
5	Longitude	deg	WGS84 Longitude
6	HAE altitude	m	Height above WGS84 ellipsoid
7	MSL altitude	m	Elevation above mean sea level
8	pDOP	m	Position dilution of precision
9	hAcc	m	Horizontal accuracy
10	vAcc	m	Vertical accuracy
11	Velocity X	m/s	ECEF X velocity
12	Velocity Y	m/s	ECEF Y velocity
13	Velocity Z	m/s	ECEF Z velocity
14	sAcc	m/s	Speed accuracy
15	cnoMean	dBHz	Average of all satellite carrier to noise ratios (signal strengths) that non-zero
16	towOffset	s	Time sync offset between local time since boot up to GPS time of week in seconds. Add this to IMU and sensor time to get GPS time of week in seconds.
17	leapS	s	GPS leap second (GPS-UTC) offset. Receiver's best knowledge of the leap seconds offset from UTC to GPS time. Subtract from GPS time of week to get UTC time of week.

GPGGA

NMEA GPS fix, 3D location and accuracy data (see NMEA GPGGA specification).

```
$GPGGA,204153,4003.3433,N,11139.5187,W,1,25,0.93,1433.997,M,18.82,M,,*6d\r\n
            1         2 3          4 5 6  7    8     9    0     1 2 3 4
```

Index	Field	Units	Description	Example
1	HHMMSS		UTC time (fix taken at 12:39:19 UTC)	123519
2,3	Latitude	deg,min	WGS84 latitude (DDmm.mmmm,N)	4003.3433,N
4,5	Longitude	deg,min	WGS84 longitude (DDDmm.mmmm,E)	11139.5187,W
6	Fix quality		0 = invalid, 1 = GPS fix (SPS), 2 = DGPS fix, 3 = PPS fix, 4 = RTK Fix, 5 = RTK Float, 6 = estimated (dead reckoning), 7 = Manual input mode, 8 = Simulation mode	1
7	# Satellites		Number of satellites in use	15
8	hDop	m	Horizontal dilution of precision	0.9
9,10	MSL_altitude	m	Elevation above mean sea level (MSL)	545.4,M
11,12	Undulation	m	Undulation of geoid. Height of the geoid above the WGS84 ellipsoid.	46.9,M
13	empty	s	Time since last DGPS update	
14	empty		DGPS station ID number	

GPGLL

NMEA geographic position, latitude / longitude and time (see NMEA GPGLL specification).

```
$GPGLL,4916.4512,N,12311.1232,W,225444,A*33\r\n
               1 2          3 4      5 6
```

Index	Field	Units	Description	Example
1,2	Latitude	deg,min	WGS84 latitude (DDmm.mmmm,N)	4916.4512,N
3,4	Longitude	deg,min	WGS84 longitude (DDDmm.mmmm,E)	12311.1232,W
5	HHMMSS		UTC time (fix taken at 22:54:44 UTC)	225444
6	Valid		Data valid (A=active, V=void)	A

GPGSA

NMEA GPS DOP and active satellites (see NMEA GPGSA specification).

```
$GPGSA,A,3,04,05,,09,12,,,24,,,,,2.5,1.3,2.1*39\r\n
       1 2  3  4 ...              15  16  17
```

Index	Field	Units	Description	Example
1			Auto selection of 2D or 3D fix (M = manual)	A
2	Fix quality		Fix quality (1 = none, 2 = 2D, 3 = 3D)	3
3-14	Sat ID		Satellite ID (PRNs)	04,05,,09,12,,,24,,,,,
15	pDop	m	Dilution of precision	2.5
16	hDop	m	Horizontal dilution of precision	1.3
17	vDop	m	Vertical dilution of precision	2.1

GPRMC

NMEA GPS recommended minimum specific GPS/Transit data (see NMEA GPRMC specification).

```
eg1. $GPRMC,081836,A,3751.65,S,14507.36,E,000.0,360.0,130998,011.3,E*62
eg2. $GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*68

           225446       Time of fix 22:54:46 UTC
           A            Navigation receiver warning A = OK, V = warning
           4916.45,N    Latitude 49 deg. 16.45 min North
           12311.12,W   Longitude 123 deg. 11.12 min West
           000.5        Speed over ground, Knots
           054.7        Course Made Good, True
           191194       Date of fix  19 November 1994
           020.3,E      Magnetic variation 20.3 deg East
           *68          mandatory checksum

eg3. $GPRMC,220516,A,5133.82,N,00042.24,W,173.8,231.8,130694,004.2,W*70
              1    2    3    4    5     6    7    8      9     10  11 12

      1   220516     Time Stamp
      2   A          validity - A-ok, V-invalid
      3   5133.82    current Latitude
      4   N          North/South
      5   00042.24   current Longitude
      6   W          East/West
      7   173.8      Speed in knots
      8   231.8      True course
      9   130694     Date Stamp
      10  004.2      Variation
      11  W          East/West
      12  *70        checksum

eg4. $GPRMC,hhmmss.ss,A,llll.ll,a,yyyyy.yy,a,x.x,x.x,ddmmyy,x.x,a*hh
1    = UTC of position fix
2    = Data status (V=navigation receiver warning)
3    = Latitude of fix
4    = N or S
5    = Longitude of fix
6    = E or W
7    = Speed over ground in knots
8    = Track made good in degrees True
9    = UT date
10   = Magnetic variation degrees (Easterly var. subtracts from true course)
11   = E or W
12   = Checksum
```

GPZDA

NMEA GPS UTC Time and Date (see NMEA GPZDA specification).

```
$GPZDA,001924,06,01,1980,00,00*41\r\n
          1    2  3   4   5  6
```

Index	Field	Units	Description	Example
1	HrMinSec		UTC Time	001924
2	Day		Day	06
3	Month		Month	01
4	Year		Year	2020
16	localHrs		Local time zone hours	00
17	localMin		Local time zone minutes	00

PASHR

NMEA GPS DOP and active satellites (see NMEA GPGSA specification).

```
$PASHR,001924.600,95.81,T,+0.60,+1.05,+0.00,0.038,0.035,0.526,0,0*08\r\n
         1        2  3   4     5    6   7     8     9 10 11
```

Index	Field	Units	Description	Example
1			UTC Time	001924.600
2	Fix quality		Heading value in decimal degrees	95.81
3	Sat ID		T displayed if heading is relative to true north.	T
4	pDop	m	Roll in decimal degrees.	+0.60
5	hDop	m	Pitch in decimal degrees.	+1.05
6	vDop	m	Instantaneous heave in meters.	+0.00
7			Roll standard deviation in decimal degrees.	+0.038
8			Pitch standard deviation in decimal degrees.	0.035
9			Heading standard deviation in decimal degrees.	0.526
10			GPS Status	0
11			INS Status	0

PSTRB

Strobe input time. This message is sent when an assert event occurs on a strobe input pin.

```
$PSTRB,d,d,d,d*xx\r\n
       1 2 3 4
```

Index	Field	Units	Description
1	GPS week	weeks	Number of weeks since January 1st of 1980 in GMT
2	timeMsOfWeek	ms	Milliseconds since Sunday morning in GMT
3	pin		Strobe event input pin number
4	count		Strobe event serial index number

INFO

Device version information. Query this message by sending `$INFO*0E\r\n`.

```
$INFO,d,d.d.d.d,d.d.d.d,d,d.d.d.d,d,s,YYYY-MM-DD,hh:mm:ss.ms,s*xx\r\n
      1 2       3       4 5       6 7 8          9           10
```

Index	Field	Units	Description
1	Serial number		Manufacturer serial number
2	Hardware version		Hardware version
3	Firmware version		Firmware version
4	Build number		Firmware build number
5	Protocol version		Communications protocol version
6	Repo revision		Repository revision number
7	Manufacturer		Manufacturer name
8	Build date		Build date: [1] = year-2000, [2] = month, [3] = day
9	Build time		Build date: [0] = hour, [1] = minute, [2] = second, [3] = millisecond
10	Add Info		Additional information

7.3.6 ASCII Examples

This section illustrates common ASCII message strings for configuration.

Note

If the command strings below are altered, their checksum must be recalculated.

Note

All ASCII command strings must be followed with a carriage return and new line character (`\r\n` or `0x0D`, `0x0A`).

The NMEA string checksum is automatically computed and appended to string when using the InertialSense SDK serialPortWriteAscii function or can be generated using an online NMEA checksum calculator. For example: MTK NMEA checksum calculator

Stop streams on CURRENT port

```
$STPB*15
```

Stop all streams on ALL ports

```
$ASCB,255,0,0,0,0,0,0,0,0,0,0,0,0*0D
```

Query device version information

```
$INFO*0E
```

Response:

```
$INFO,30612,3.1.2.0,1.7.0.0,3522,1.2.74.7,6275,Inertial Sense INC,0018-10-16,23:20:38.41,INL2*58
```

Stream INS1 @25Hz on port 0

```
$ASCB,1,,,40,,,,,,,,,*0A
```

Stream INS1 @10Hz on current port

```
$ASCB,0,,,100,,,,,,,,,*3E
```

Response:

```
$PINS1,244272.398,2021,427888998,805306448,0.0468,-0.3830,-0.0909,0.232,-0.083,-0.089,40.05574940,-111.65861580,1438.451,-1.678,-5.086,-9.697*11
$PINS1,244272.498,2021,427888998,805306448,0.0469,-0.3830,-0.0902,0.232,-0.081,-0.089,40.05575000,-111.65861550,1438.451,-1.611,-5.060,-9.697*18
$PINS1,244272.598,2021,427888998,805306448,0.0469,-0.3830,-0.0902,0.232,-0.081,-0.089,40.05575022,-111.65861562,1438.449,-1.587,-5.070,-9.695*1e
```

Stream INS1 @50Hz on serial port 1

```
$ASCB,2,,,20,,,,,,,,,*0F
```

Response:

```
$PINS1,256270.627,2021,427888998,1073741912,0.1153,-0.1473,-0.1628,0.001,0.001,0.003,40.05569486,-111.65864500,1416.218,-7.738,-7.570,12.536*3d
$PINS1,256270.647,2021,427888998,1073741912,0.1153,-0.1473,-0.1632,0.001,0.001,0.003,40.05569486,-111.65864500,1416.219,-7.738,-7.570,12.535*32
$PINS1,256270.667,2021,427888998,1073741912,0.1153,-0.1473,-0.1631,0.001,0.001,0.003,40.05569486,-111.65864500,1416.220,-7.738,-7.570,12.534*38
```

Stream PIMU @50Hz and GPGGA @5Hz on current port

```
$ASCB,0,20,0,0,0,0,0,200,0,0,0,0,0*3F
```

Response:

```
$PIMU,3218.543,0.0017,-0.0059,-0.0077,-1.417,-1.106,-9.524,0.0047,0.0031,-0.0069,-1.433,-1.072,-9.585*1f
$GPGGA,231841,4003.3425,N,11139.5188,W,1,29,0.89,1434.16,M,18.82,M,,*59
$PIMU,3218.563,0.0022,-0.0057,-0.0091,-1.416,-1.123,-9.512,0.0061,0.0035,-0.0068,-1.430,-1.091,-9.563*19
$PIMU,3218.583,0.0007,-0.0059,-0.0081,-1.420,-1.125,-9.508,0.0056,0.0047,-0.0086,-1.432,-1.078,-9.591*1e
$PIMU,3218.603,0.0015,-0.0062,-0.0077,-1.419,-1.130,-9.499,0.0069,0.0044,-0.0066,-1.434,-1.079,-9.579*10
$PIMU,3218.623,-0.0001,-0.0052,-0.0097,-1.413,-1.123,-9.529,0.0066,0.0047,-0.0072,-1.427,-1.085,-9.593*39
$PIMU,3218.643,0.0012,-0.0060,-0.0080,-1.423,-1.122,-9.508,0.0071,0.0047,-0.0070,-1.425,-1.083,-9.563*19
$PIMU,3218.663,-0.0004,-0.0065,-0.0088,-1.413,-1.118,-9.540,0.0057,0.0029,-0.0059,-1.430,-1.082,-9.604*3a
$PIMU,3218.683,-0.0001,-0.0064,-0.0096,-1.418,-1.121,-9.511,0.0059,0.0033,-0.0070,-1.431,-1.084,-9.585*39
$PIMU,3218.703,0.0007,-0.0055,-0.0079,-1.417,-1.128,-9.497,0.0046,0.0043,-0.0077,-1.428,-1.085,-9.565*18
$PIMU,3218.723,0.0024,-0.0054,-0.0085,-1.416,-1.106,-9.510,0.0051,0.0033,-0.0079,-1.429,-1.089,-9.588*1b
$PIMU,3218.743,0.0019,-0.0058,-0.0081,-1.430,-1.126,-9.533,0.0063,0.0032,-0.0073,-1.435,-1.093,-9.585*1d
$GPGGA,231841,4003.3425,N,11139.5188,W,1,29,0.89,1434.19,M,18.82,M,,*56
$PIMU,3218.763,0.0019,-0.0062,-0.0086,-1.426,-1.114,-9.509,0.0054,0.0029,-0.0070,-1.431,-1.085,-9.579*13
```

Using Tera Term App

The example ASCII command strings can be sent using standard serial port terminal that supports sending the line ending characters (carriage return and new line). The Windows desktop app Tera Term can be used for this.

Setup:

1. Start Tera Term > New connection. Select the correct serial port.
2. Setup > Terminal... - Receive: CR - Transmit: CR+LF **(IMPORTANT - Ensures each ASCII string ends with `\r\n` or `0x0D, 0x0A`)** - Local echo: Yes This shows what was sent.
3. Setup > Serial port... - Speed: 921600 This is the default setting and must match the uINS serial port baudrate. - Data: 8 bit - Parity: none - Stop bits: 1 bit - Flow control: none
4. menu bar > setup > save setup - This saves time the next time Tera Term is used.

Saving a log:

1. File > log - Be sure to save as a .txt so it can be viewed in notepad. Controls for the log are in the Logger popup window after starting a log.

Sending an ASCII command:

1. Copy one of the command messages above to the clipboard.
2. Use the Tera Term > Edit > `Paste<CR>` option to send the copied command. This method in conjunction with the above "CR+LF" terminal transmit setting ensures that a carriage return and line feed character terminate the sent string. Only one ASCII string command can be sent at a time.

7.4 Inertial Sense Binary Protocol

The Inertial Sense binary protocol provides the most efficient way to communicate with the µINS, µAHRS, and µIMU because it preserved the native floating point and integer binary format used in computers. Binary protocol is not human readable like ASCII Protocol. Binary protocol uses Data Set (DID) C structures defined in SDK/src/data_sets.h of the InertialSense SDK.

7.4.1 Communication

Writing to and reading from InertialSense products is done using "Set" and "Get" commands.

Setting Data

The `is_comm_set_data()` function will encode a message used to set data or configurations.

```
// Set INS output Euler rotation in radians to 90 degrees roll for mounting
float rotation[3] = { 90.0f*C_DEG2RAD_F, 0.0f, 0.0f };
int messageSize = is_comm_set_data(comm, _DID_FLASH_CONFIG, offsetof(nvm_flash_cfg_t, insRotation), sizeof(float) * 3, rotation);
if (messageSize != serialPortWrite(serialPort, comm->buffer, messageSize))
{
    printf("Failed to encode and write set INS rotation\r\n");
    return -3;
}
```

Getting Data

Data broadcasting or streaming is enabled by using the Realtime Message Controller (RMC) or the get data command.

GET DATA COMMAND

The `is_comm_get_data()` function will encode a PID_GET_DATA message that enables broadcast of a given message at a multiple of the *Data Source Update Rates*. Set the data rate (period multiple) to zero disable message broadcast and pull a single packet of data. Set the data size and offset to zero to request the entire data set.

```
// Ask for INS message w/ update 40ms period (4ms source period x 10).  Set data rate to zero to disable broadcast and pull a single packet.
int messageSize = is_comm_get_data(comm, _DID_INS_LLA_EULER_NED, 0, 0, 10);
if (messageSize != serialPortWrite(serialPort, comm->buffer, messageSize))
{
    printf("Failed to encode and write get INS message\r\n");
}
```

DATA SOURCE UPDATE RATES

DID	Update Rate (Period)
DID_INS_[1-4]	(4ms default) Configured with DID_FLASH_CONFIG.startupNavDtMs
DID_DUAL_IMU, DID_PREINTEGRATED_IMU*	(4ms default) Configured with DID_FLASH_CONFIG.startupImuDtMs
DID_BAROMETER	~8ms
DID_MAGNETOMETER_[1-2]	~10ms
DID_GPS[1-2]_[X] (Any DID beginning with DID_GPS)	(200ms default) Configured with DID_FLASH_CONFIG. startupGPSDtMs
All other DIDs	1ms

*DID_PREINTEGRATED_IMU integration period (dt) and output data rate are the same as DID_FLASH_CONFIG.startupNavDtMs and cannot be output at any other rate. If a different output data rate is desired, DID_DUAL_IMU which is derived from DID_PREINTEGRATED_IMU can be used instead.

REALTIME MESSAGE CONTROLLER (RMC)

The RMC is used to enable message broadcasting and provides updates from onboard data as soon as it becomes available with minimal latency. All RMC messages can be enabled using the *Get Data Command*, which is the preferred method, or by directly setting the RMC bits. The RMC bits are listed below. Message data rates are listed in the *Data Source Update Rates* table.

RMC Message
RMC_BITS_INS[1-4]
RMC_BITS_DUAL_IMU, RMC_BITS_PREINTEGRATED_IMU
RMC_BITS_BAROMETER
RMC_BITS_MAGNETOMETER[1-2]
RMC_BITS_GPS[1-2]_NAV
RMC_BITS_GPS_RTK_NAV, RMC_BITS_GPS_RTK_MISC
RMC_BITS_STROBE_IN_TIME

The following is an example of how to use the RMC. The `rmc.options` field controls whether RMC commands are applied to other serial ports. `rmc.options = 0` will apply the command to the current serial port.

```
rmc_t rmc;

// Enable broadcasts of DID_INS_1 and DID_GPS_NAV
rmc.bits = RMC_BITS_INS1 | RMC_BITS_GPS1_POS;
// Remember configuration following reboot for automatic data streaming.
rmc.options = RMC_OPTIONS_PERSISTENT;
// INS output data rate at 20Hz
rmc.insPeriodMs = 50;

int messageSize = is_comm_set_data(comm, _DID_RMC, 0, sizeof(rmc_t), &rmc);
if (messageSize != serialPortWrite(serialPort, comm->buffer, messageSize))
{
    printf("Failed to encode and write RMC message\r\n");
}
```

PERSISTENT MESSAGES

The *persistent messages* option saves the current data stream configuration to flash memory for use following reboot, eliminating the need to re-enable messages following a reset or power cycle.

- **To save persistent messages** - (to flash memory), bitwise OR `RMC_OPTIONS_PERSISTENT (0x200)` with the RMC option field or set DID_CONFIG.system = 0x00000001 and DID_CONFIG.system = 0xFFFFFFFE. See the save persistent messages example in the Binary Communications example project.
- **To disable persistent messages** - a stop all broadcasts packet followed by a *save persistent messages* command.

ASCII persistent messages are also available.

Enabling Persistent Messages - EvalTool

- Enable the desired messages in the EvalTool "Data Sets" tab.
- Press the "Save Persistent" button in the EvalTool "Data Logs" tab to store the current message configuration to flash memory for use following reboot.
- Reset the system and verify the messages are automatically streaming. You can use the EvalTool->Data Logs dialog to view the streaming messages.

To disable all persistent messages using the EvalTool, click the "Stop Streaming" button and then "Save Persistent" button.

Enabling Persistent Messages - CLTool

Persistent messages are enabled using the CLTool by including the `-persistent` option along with the options for the desired messages in the command line.

```
cltool -c /dev/ttyS3 -persistent -msgINS2 -msgGPS
```

EXAMPLE PROJECTS

Examples on how to use the Inertial Sense SDK for binary communications are found in the Binary Communications Example Project and cltool project. ASCII communications examples are found in the ASCII Example Project.

Parsing Data

The ISComm library in the InertialSenseSDK provides a communications parser that can parse InertialSense binary protocol as well as other protocols.

ONE BYTE (SIMPLE METHOD)

The following parser code is simpler to implement. This method uses the `is_comm_parse_byte()` function to parse one byte at a time of a data stream. The return value is a non-zero protocol_type_t when valid data is found.

```
    uint8_t c;
    protocol_type_t ptype;
    // Read from serial buffer until empty
    while (mySerialPortRead(&c, 1))
    {
        if ((ptype = is_comm_parse_byte(comm, c)) != _PTYPE_NONE)
        {
            switch (ptype)
            {
            case _PTYPE_INERTIAL_SENSE_DATA:
            case _PTYPE_INERTIAL_SENSE_CMD:
                break;
            case _PTYPE_UBLOX:
                break;
            case _PTYPE_RTCM3:
                break;
            case _PTYPE_ASCII_NMEA:
                break;
            }
        }
    }
```

SET OF BYTES (FAST METHOD)

The following parser code uses less processor time to parse data by copying multiple bytes at a time. This method uses `is_comm_free()` and `is_comm_parse()` along with a serial port read or buffer copy. The return value is a non-zero protocol_type_t when valid data is found.

```
    // Read a set of bytes (fast method)
    protocol_type_t ptype;

    // Get available size of comm buffer
    int n = is_comm_free(comm);

    // Read data directly into comm buffer
    if ((n = mySerialPortRead(comm->buf.tail, n)))
    {
        // Update comm buffer tail pointer
        comm->buf.tail += n;

        // Search comm buffer for valid packets
        while ((ptype = is_comm_parse(comm)) != _PTYPE_NONE)
        {
            switch (ptype)
            {
            case _PTYPE_INERTIAL_SENSE_DATA:
            case _PTYPE_INERTIAL_SENSE_CMD:
                break;
            case _PTYPE_UBLOX:
                break;
            case _PTYPE_RTCM3:
                break;
            case _PTYPE_ASCII_NMEA:
                break;
            }
        }
    }
```

7.4.2 Packet Structure

The SDK is provided to encode and decode binary packets. This section can be disregareded if the SDK is used. This section is provided to detail the packet and data structures format used in the binary protocol. The µINS, µAHRS, and µIMU communicate using the following binary packet formatting:

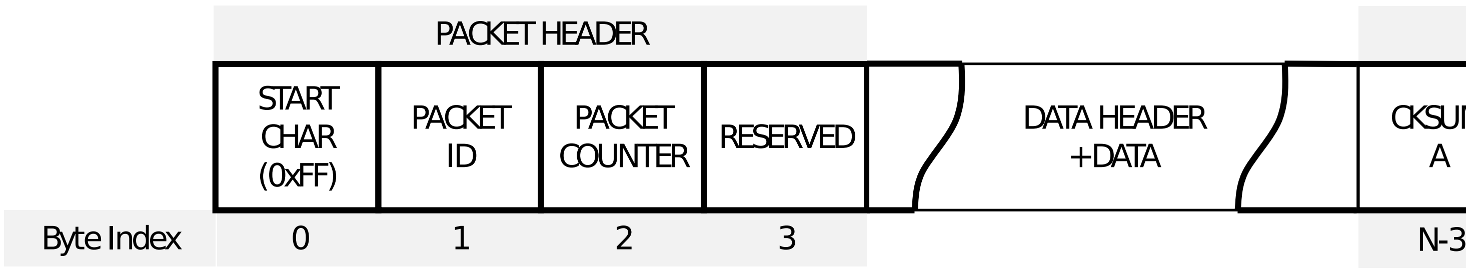


Figure 1 – Packet Structure

Packet Header (4 bytes)
1 byte – packet start byte (`0xFF`)
1 byte – packet identifier, determines the format of packet data
1 byte – packet counter (for retries)
1 byte – flags, ePktHdrFlags enum

```
#include "SDK/src/com_manager.h"
enum ePktHdrFlags
{
// bit set for little endian, bit cleared for big endian
CM_PKT_FLAGS_LITTLE_ENDIAN = 0x01,
// has any valid packet been received
CM_PKT_FLAGS_RX_VALID_DATA = 0x02,
// multi-packet data set
CM_PKT_FLAGS_MORE_DATA_AVAILABLE = 0x04,
// Allow for arbitrary length in bytes of data, not necessarily multiple of 4. Don't
auto-swap bytes for endianness
CM_PKT_FLAGS_RAW_DATA_NO_SWAP = 0x08,
// Checksum is the new 24 bit algorithm instead of the old 16 bit algorithm
CM_PKT_FLAGS_CHECKSUM_24_BIT = 0x10
};
```

Packet data may be 0 bytes, depending on packet identifier (PID)

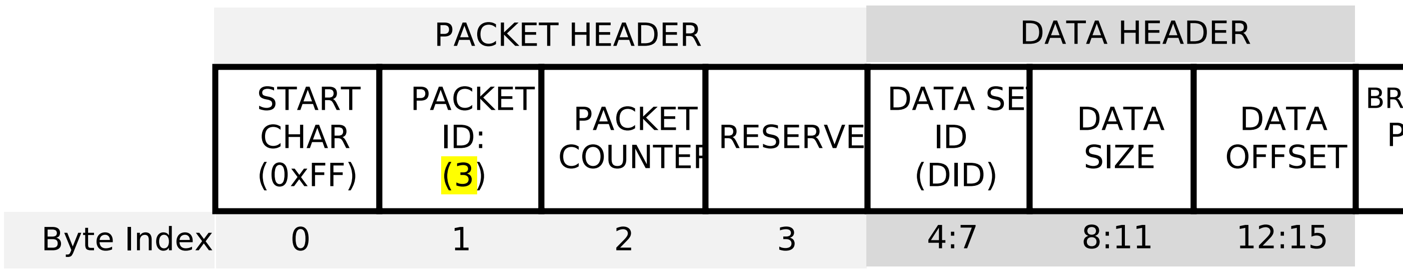


Figure 2 – PID_GET_DATA Packet

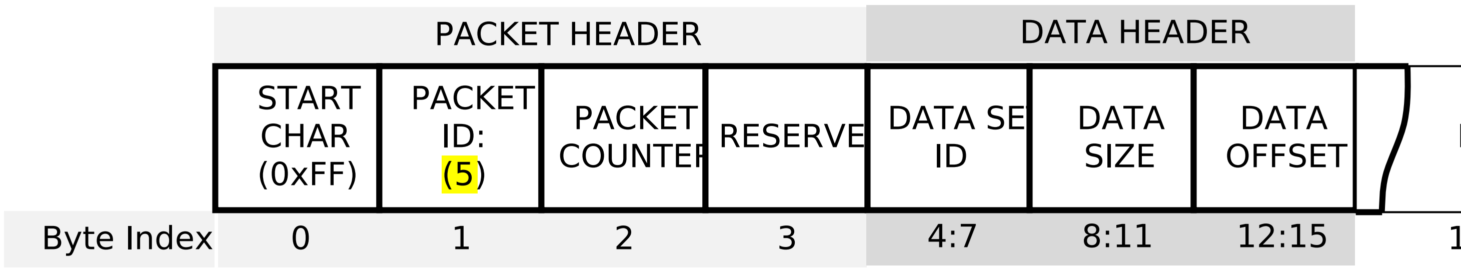


Figure 3 – PID_SET_DATA and PID_DATA Packet

The format of the packet data is determined by the packet identifier. For a data packet (PID_DATA (4) or PID_SET_DATA (5)) the first 12 bytes are always the data identifier (4 byte int), the offset into the data (4 byte int), and the length of data (4 byte int, not including the 12 bytes or packet header / footer).

Packet footer (4 bytes)
1 byte – checksum byte A (most significant byte)
1 byte – checksum byte B (middle byte)
1 byte – checksum byte C (least significant byte)
1 byte – packet stop byte (`0xFE`)

The packet checksum is a 24-bit integer that can be created as follows:

- Start the checksum value at `0x00AAAAAA` and skip the packet start byte
- Exclusive OR the checksum with the packet header (identifier, counter, and packet flags) * checksum ^= packetId * checksum ^= (counter << 8) * checksum ^= (packetFlags << 16)
- Exclusive OR the checksum with each data byte in packet, repeating the following three steps, until all data is included (i == dataLength). * checksum ^= (dataByte[i++]) * checksum ^= (dataByte[i++] << 8) * checksum ^= (dataByte[i++] << 16)
- Decode encoded bytes (bytes prefixed by 0xFD) before calculating checksum for that byte.
- Perform any endianness byte swapping AFTER checksum is fully calculated.

If the CPU architecture does not match the packet flags, the bytes need to be swapped appropriately. The SDK does this automatically.

Packet Encoding/Decoding - Special Bytes

Bytes `0x0A` , `0x24` , `0xB5` , `0xD3` , `0xFD` , `0xFE` and `0xFF` are reserved bytes, with `0xFD` being a reserved byte prefix. When those bytes are written to a packet, they are prefixed with `0xFD` and then written with all the bits inverted. The packet start and end byte are never encoded. When calculating a checksum, decode the byte if necessary first and then add the byte to the checksum calculation.

A raw packet can never be more than 2048 bytes. A decoded packet will never be more than 1024 bytes.

For a full example of encoding and decoding binary packets, please reference the following files in the SDK source: `com_manager.c` , `com_manager.h` , `data_structures.c` and `data_structures.h` , and in particular, the functions `encodeBinaryPacket()` and `decodeBinaryPacket()` .

7.4.3 Stop Broadcasts Packets

Two *stop all broadcasts* packets are special packet types that will disable all binary and ASCII data streams. The following functions calls are provided in the SDK to generate and send the stop all broadcasts packet.

All Ports

```
int messageSize = is_comm_stop_broadcasts_all_ports(comm);
serialPortWrite(serialPort, comm->buffer, messageSize);
```

Hexadecimal value of the stop all broadcasts packet.

```
0xff 0x06 0x00 0x11 0xbb 0xaa 0xac 0xfe
```

Current Port Only

```
int messageSize = is_comm_stop_broadcasts_current_port(comm);
serialPortWrite(serialPort, comm->buffer, messageSize);
```

Hexadecimal value of the stop all broadcasts packet.

```
0xff 0x08 0x00 0x11 0xbb 0xaa 0xa2 0xfe
```

7.5 SPI Protocol

The SPI interface provides an alternative method of communications with the µINS, µAHRS, and µIMU. The SPI protocol uses much of the same structure and format as the serial communication binary protocol which is outlined in the Binary Protocol section of the users manual.

7.5.1 Enable SPI

To Enable SPI, hold pin G9 (nSPI_EN) low at startup.

Note: When external GPS PPS timepulse signal is enabled on G9, the module will ignore the nSPI_EN signal and SPI mode will be disabled regardless of G9 pin state.

7.5.2 Hardware

Inertial Sense SPI interface uses 5 lines to interface with other devices.

Line	**Function**
CS	SPI Chip Select
SCLK	SPI Clock Synchronization Pin
MISO	SPI Master In Slave Out
MOSI	SPI Master Out Slave In
DR	SPI Data Ready Pin (optional)

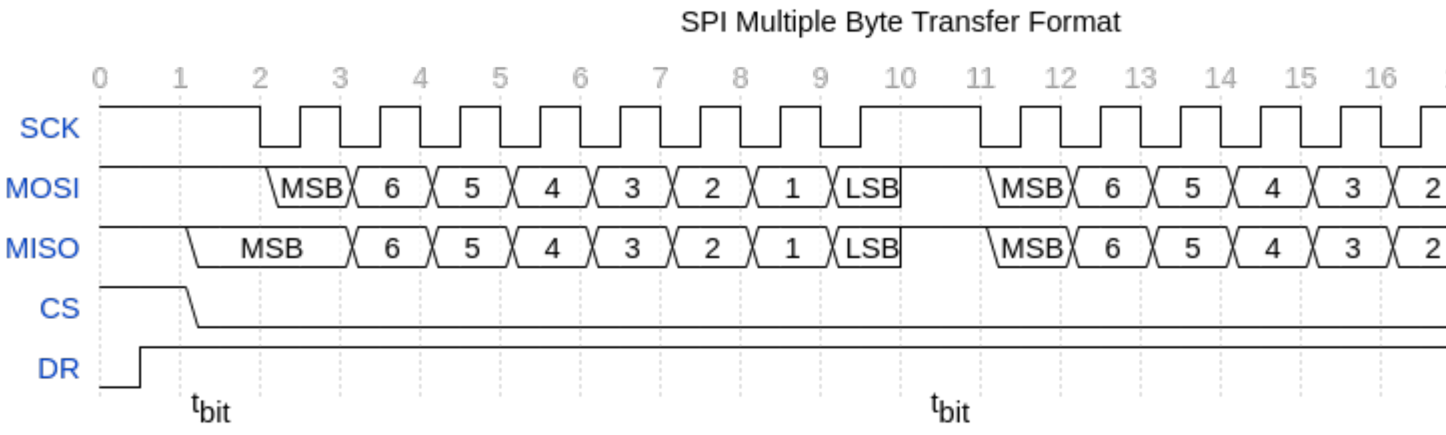
Hardware configuration

The Inertial Sense units operate as slave devices using SPI Mode 3:

SPI Master Settings	
Clock Polarity	Idle High (CPOL = 1)
Clock Phase	Falling Edge (CPHA = 1)
Chip Select	Active Low
Data Ready	Active High

Data Transfer

To ensure correct behavior of the receiver in SPI Slave mode, the master device sending the frame must ensure a minimum delay of one t_{bit} (t_{bit} being the nominal time required to transmit a bit) between each character transmission. Inertial Sense devices do not require a falling edge of the [Chip Select (CS)] to initiate a character reception but only a low level. However, this low level must be present on the [Chip Select (CS)] at least one t_{bit} before the first serial clock cycle corresponding to the MSB bit. (1)



When reading the uINS and there is no data ready it will send zeros for the data.

Keeping CS low should not cause any issues. However, if the clocking between the master and slave processors gets out of sync there is nothing to get them back into sync. Ground bounce or noise during a transition could cause the uINS sees two clock edges when there should have only been one (due to an ESD or a fast transient event). Raising and lowering the CS line resets the shift register will resynchronize the clocks.

Data Ready Pin Option

There is a data ready pin option. This signal will be raised when data becomes ready. Depending on when this happens there can be 1-4 bytes of zeros that will come out before the packet starts. Also this line will go inactive a byte or two before the end of the packet gets sent. There is not a "not in a data packet" character to send. It is strictly done by data ready pin and parsing.

If the chip select line is lowered during a data packet, the byte being transmitted (or that would be transmitted) can be lost. It is recommended to only lower the chip select when outside of a data packet and the data ready pin is inactive.

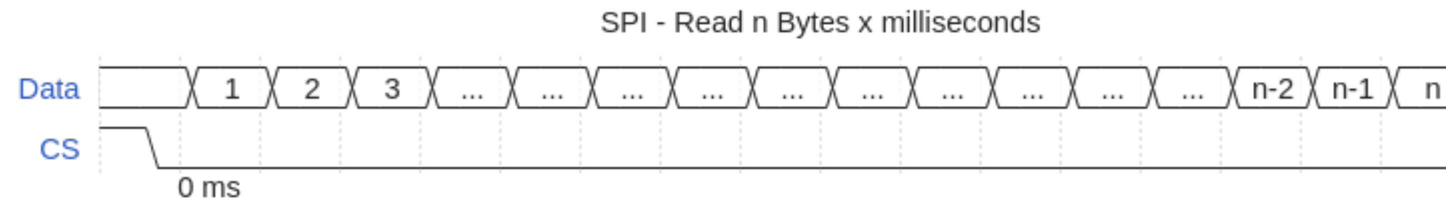
The internal SPI buffer is 4096 bytes. If there is a buffer overflow, the buffer gets dropped. This is indicated by a data ready pin that is high without data being there. When an overflow happens, it clears the buffer, so the system could be in the middle of a packet and the uINS would just send zeros. If a request is sent to the uINS or the uINS sends a packet periodically it will resolve the situation.

The SPI interface supports up to 3 Mbs data rate. (5 Mbs works if the data ready pin is used to receive the data - see B below.)

Reading Data

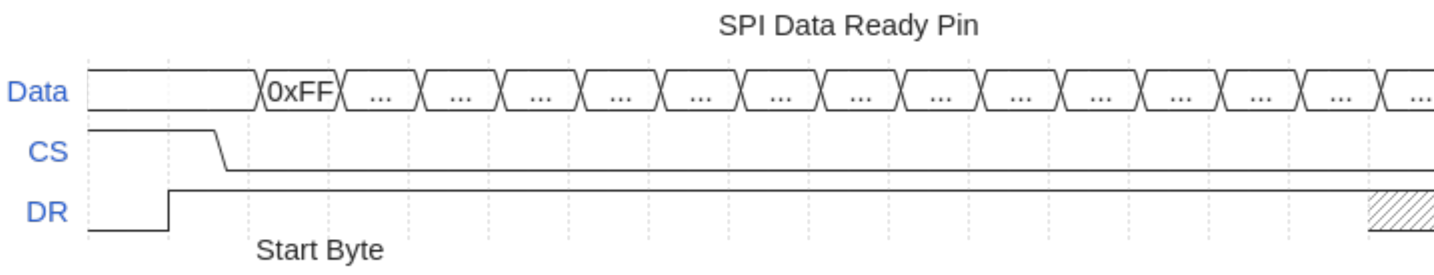
There are two strategies that can be used to read the data:

A. Read a fixed data size out every set time interval. More data will be read than the uINx will produce on a regular interval, for instance, reading 512 bytes every 4 ms.



Packet will be 0x00 padded if bytes read exceeds packet size.

B. Read while the data ready pin is active **or** we are inside a data packet. One anomaly is the data ready pin will drop a byte or two before the end gets clocked out, hence needing to watch for the end of the packet.



Pseudo Code for reading data:

1. Check data ready pin. If pin is low, delay and check pin again.
2. Lower CS line and read a block of data. Read sizes are arbitrary, but it tends to work better if the read count is long enough to contain most data packets.
3. After read completes, check data ready pin. If it is high, read more data. DO NOT raise the CS line while the data ready pin is high, it will cause data loss. If data ready is low, raise CS line. On a busy system (and depending on baud rate) this would need to happen along with the data read as the data ready pin might not go low in between packets.
4. Parse data looking for start of packet (0xFF) discarding data until found. Once found start saving the data.
5. Save and parse data looking for end of packet (0xFE). Once found send packet off for use. If a start of packet character is seen while looking for the end, discard previous data and start the packet saving over.

7.5.3 EVB-2 SPI Dev Kit

The EVB-2 demonstrates SPI interface with the uINS. The EVB-2 ATSAM-E70 (E70) processor provides the example SPI interface with the uINS. The EVB-2 must be put into CBPreset mode 6 (CONFIG led color cyan) followed by a system reset to enable SPI mode. The EVB-2 (E70) project source code is available in the SDK for reference.

7.5.4 Troubleshooting

If every other character from a packet is lost it might be that the CS line is being toggled after every byte.

The uINS 3.1 uses a USART SPI peripherial which requires a minimum delay of one t_{bit} (t_{bit} being the nominal time required to transmit a bit) spacing between characters sent. Reading bytes one by one may cause signifacnt time delays when streaming data. Depending on the ammount of data streaming, the uINS mable to keep up and the buffer could be overflow. Single message requests should work properly, but streaming probably will not work well. If the master hardware can't handle the delay, the uINS 3.2 hardware should be used.

7.5.5 Resources

(1) *SAM E70/S70/V70/V71 Family*. Microchip Technology Inc., https://ww1.microchip.com/downloads/en/DeviceDoc/SAM-E70-S70-V70-V71-Family-Data-Sheet-DS60001527E.pdf

7.6 CAN Protocol

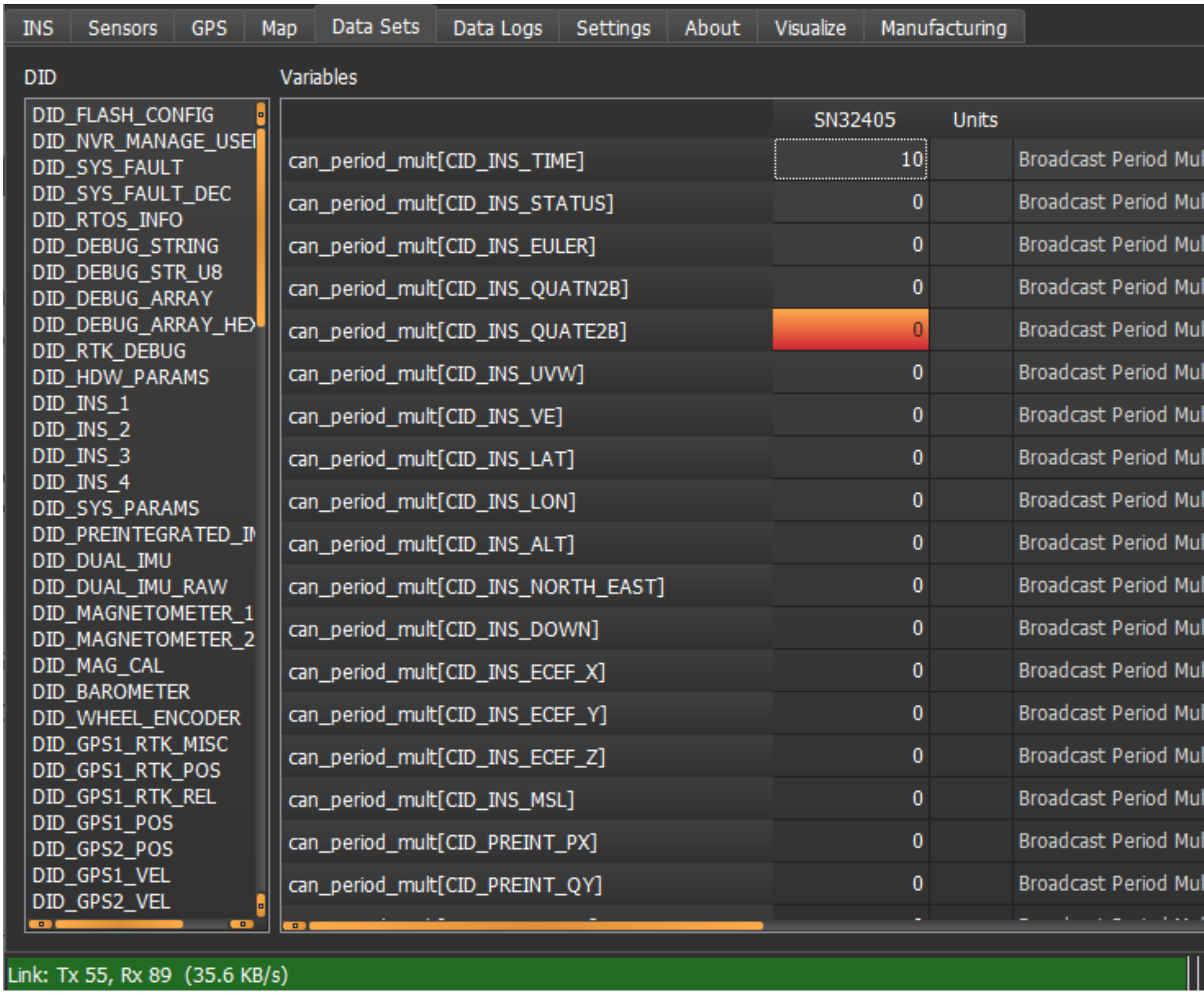
The CAN interface allows the output of the the µINS, µAHRS, and µIMU to be published on a CAN bus. The Inertial Sense CAN implementation is based on CAN2.0b specification and follows a specific structure and format which is outlined below. All of the CAN configuration is done using the data set DID_CAN_CONFIG.

7.6.1 Enable CAN

To enable the CAN bus interface on the uINS, set bit `IO_CONFIG_G1G2_CAN_BUS` in `DID_FLASH_CONFIG.ioConfig`. This bit can be set using the EvalTool >> Settings >> General >> DID_FLASH_CONFIG >> ioConfig >> `Enable CAN Bus on G1,G2` option.

A CAN message is enabled by entering a non-zero value in the DID_CAN_CONFIG.*can_period_mult* field of the desired message. The *can_period_mult* field is an integer which is multiplied by the *Data Source Update Rate* to determine the message broadcast period. Set *can_period_mult* to zero disable message broadcasting.

In the image below the CID_INS_TIME message is set to broadcast data at 10 times the data source rate.



DID_FLASH_CONFIG
DID_NVR_MANAGE_USEI
DID_SYS_FAULT
DID_SYS_FAULT_DEC
DID_RTOS_INFO
DID_DEBUG_STRING
DID_DEBUG_STR_U8
DID_DEBUG_ARRAY
DID_DEBUG_ARRAY_HE
DID_RTK_DEBUG
DID_HDW_PARAMS
DID_INS_1
DID_INS_2
DID_INS_3
DID_INS_4
DID_SYS_PARAMS
DID_PREINTEGRATED_I
DID_DUAL_IMU
DID_DUAL_IMU_RAW
DID_MAGNETOMETER_1
DID_MAGNETOMETER_2
DID_MAG_CAL
DID_BAROMETER
DID_WHEEL_ENCODER
DID_GPS1_RTK_MISC
DID_GPS1_RTK_POS
DID_GPS1_RTK_REL
DID_GPS1_POS
DID_GPS2_POS
DID_GPS1_VEL
DID_GPS2_VEL

	SN32405	Units	
can_period_mult[CID_INS_TIME]	10		Broadcast Period Mul
can_period_mult[CID_INS_STATUS]	0		Broadcast Period Mul
can_period_mult[CID_INS_EULER]	0		Broadcast Period Mul
can_period_mult[CID_INS_QUATN2B]	0		Broadcast Period Mul
can_period_mult[CID_INS_QUATE2B]	0		Broadcast Period Mul
can_period_mult[CID_INS_UVW]	0		Broadcast Period Mul
can_period_mult[CID_INS_VE]	0		Broadcast Period Mul
can_period_mult[CID_INS_LAT]	0		Broadcast Period Mul
can_period_mult[CID_INS_LON]	0		Broadcast Period Mul
can_period_mult[CID_INS_ALT]	0		Broadcast Period Mul
can_period_mult[CID_INS_NORTH_EAST]	0		Broadcast Period Mul
can_period_mult[CID_INS_DOWN]	0		Broadcast Period Mul
can_period_mult[CID_INS_ECEF_X]	0		Broadcast Period Mul
can_period_mult[CID_INS_ECEF_Y]	0		Broadcast Period Mul
can_period_mult[CID_INS_ECEF_Z]	0		Broadcast Period Mul
can_period_mult[CID_INS_MSL]	0		Broadcast Period Mul
can_period_mult[CID_PREINT_PX]	0		Broadcast Period Mul
can_period_mult[CID_PREINT_QY]	0		Broadcast Period Mul

Link: Tx 55, Rx 89 (35.6 KB/s)

The baud rate is configurable by setting the field DID_CAN_CONFIG.can_baudrate_kbps. The following standard baud rates are supported:

- 20 kbps
- 33 kbps
- 50 kbps
- 83 kbps
- 100 kbps
- 125 kbps
- 200 kbps
- 250 kbps
- 500 kbps
- 1000 kbps

The message ID for each message can be entered into the can_transmit_address field corresponding to the desired message.

*Note: Any message ID greater than 0x7FF will be transmitted in the extended ID format.

The values set in any field of DID_CAN_CONFIG are saved to flash when a 'Save Persistent' command is received by the module. For example, this can be done in the EvalTool by clicking the Save Persistent button in the Data Logs tab. When the module is turned on, all the fields will be repopulated with the saved values.

All messages are disabled when a Stop Streaming message is received by the module. However, the values in each field will be repopulated to the values present when a 'Save Persistent' command was last received.

7.6.2 Hardware

Inertial Sense module exposes the RxCAN and TxCAN pins. The selection and implementation of a CAN transceiver is left to the user.

Line	Function
G1	RxCAN
G2	TxCAN

The Inertial Sense evaluation boards and Rugged unit have a built in transceiver.

7.6.3 CAN Data Sets (CIDs)

The CAN Data Sets, in the form of C structures, define the format of the output data. The data sets are defined in SDK\hw-libs\communications\CAN_comm.h of the InertialSense SDK. The CID data is selected data from the standard Inertial Sense DIDs. The data types generally have been changed and scaled to fit the CAN2.0 8 byte payload restrictions.

CID_INS_TIME

INS time output

`is_can_time`

GMT information

Field	Type	Description
week	uint32_t	GPS number of weeks since January 6th, 1980
timeOfWeek	float	GPS time of week (since Sunday morning) in seconds

CID_INS_STATUS

`is_can_ins_status`

INS status flags

Field	Type	Description
insStatus	uint32_t	INS status flags (see eInsStatusFlags)
hdwStatus	uint32_t	Hardware status flags (see eHdwStatusFlags)

CID_INS_EULER

`is_can_ins_euler`

Euler angles: roll, pitch, yaw in radians with respect to NED (scaled by 10000)

Field	Type	Description
theta1	int16_t	Roll (4 decimal places precision)
theta2	int16_t	Pitch (4 decimal places precision)
theta3	int16_t	Yaw (4 decimal places precision)

CID_INS_QUATN2B

`is_can_ins_quatn2b`

Quaternion body rotation with respect to NED: W, X, Y, Z (scaled by 10000)

Field	Type	Description
qn2b1	int16_t	W (4 decimal places precision)
qn2b2	int16_t	X (4 decimal places precision)
qn2b3	int16_t	Y (4 decimal places precision)
qn2b4	int16_t	Z (4 decimal places precision)

CID_INS_QUATE2B

`is_can_ins_quate2b`

Quaternion body rotation with respect to ECEF: W, X, Y, Z (scaled by 10000)

Field	Type	Description
qe2b1	int16_t	W (4 decimal places precision)
qe2b2	int16_t	X (4 decimal places precision)
qe2b3	int16_t	Y (4 decimal places precision)
qe2b4	int16_t	Z (4 decimal places precision)

CID_INS_UVW

`is_can_uvw`

Velocity U, V, W in body frame in meters per second (scaled by 100).

Field	Type	Description
uvw1	int16_t	U (2 decimal places precision)
uvw2	int16_t	V (2 decimal places precision)
uvw3	int16_t	W (2 decimal places precision)

CID_INS_VE

`is_can_ve`

Velocity in ECEF (earth-centered earth-fixed) frame in meters per second (scaled by 100).

Field	Type	Description
ve1	int16_t	ve1 (2 decimal places precision)
ve2	int16_t	ve2 (2 decimal places precision)
ve3	int16_t	ve3 (2 decimal places precision)

CID_INS_LAT

`is_can_ins_lat`

WGS84 latitude.

Field	Type	Description
lat	double	Latitude (degrees) (more than 8 decimal places precision)

CID_INS_LON

`is_can_ins_lon`

WGS84 longitude.

Field	Type	Description
lon	double	Longitude (degrees) (more than 8 decimal places precision)

CID_INS_ALT

`is_can_ins_alt`

WGS84 height above ellipsoid and GPS status flags

Field	Type	Description
alt	float	Altitude (meters) (more than 8 decimal places precision)
status	uint32_t	(see eGpsStatus) GPS status: [0x000000xx] number of satellites used, [0x0000xx00] fix type, [0x00xx0000] status flags

CID_INS_NORTH_EAST

`is_can_north_east`

Offset from reference latitude, longitude, and altitude to current latitude, longitude, and altitude.

Type	Field	Description
float	ned1	North (meters)
float	ned2	East (meters)

CID_INS_DOWN

`is_can_down`

Down offset from reference LLA and INS status flags

Type	Field	Description
float	ned3	Down (meters)
float	insStatus	INS status flags

CID_INS_ECEF_X

`is_can_ecef_x`

X Position in ECEF (earth-centered earth-fixed) frame.

Type	Field	Description
ecef1	double	X (meters)

CID_INS_ECEF_Y

`is_can_ecef_y`

Y Position in ECEF (earth-centered earth-fixed) frame.

Type	Field	Description
ecef2	double	Y (meters)

CID_INS_ECEF_Z

`is_can_ecef_z`

Z Position in ECEF (earth-centered earth-fixed) frame.

Type	Field	Description
ecef2	double	Z (meters)

CID_INS_MSL

`ins_can_msl`

Height above Mean Sea Level

Type	Field	Description
msl	float	MSL (meters)

CID_PREINT_PX

`is_can_preint_imu_px`

Preintegrated IMU values delta theta and delta velocity (X axis), and Integral period in body/IMU frame of accelerometer 0.

Type	Field	Description
theta0	int16_t	Delta theta (rad, scaled by 1000, 3 decimal places precision)
vel0	int16_t	Delta velocity (m/s, scaled by 100, 2 decimal places precision)
dt	uints16_t	Integral Period (meters, scaled by 1000)

CID_PREINT_QY

`is_can_preint_imu_qy`

Preintegrated IMU values delta theta and delta velocity (Y axis), and Integral period in body/IMU frame of accelerometer 0.

Type	Field	Description
theta1	int16_t	Delta theta (rad, scaled by 1000, 3 decimal places precision)
vel1	int16_t	Delta velocity (m/s, scaled by 100, 2 decimal places precision)
dt	uints16_t	Integral Period (meters, scaled by 1000)

CID_PREINT_RZ

`is_can_preint_imu_rz`

Preintegrated IMU values delta theta and delta velocity (Z axis), and Integral period in body/IMU frame of accelerometer 0.

Type	Field	Description
theta2	int16_t	Delta theta (rad, scaled by 1000, 3 decimal places precision)
vel2	int16_t	Delta velocity (m/s, scaled by 100, 2 decimal places precision)
dt	uints16_t	Integral Period (meters, scaled by 1000)

CID_DUAL_PX

`is_can_dual_imu_px`

Dual IMU gyro and accelerometer values from accelerometer 0

Type	Field	Description
theta0	int16_t	Theta (rad/s, scaled by 1000, 3 decimal places precision)
vel0	int16_t	Acceleration (m/s^2, scaled by 100, 2 decimal places precision)
status	uints32_t	IMU status (see eImuStatus)

CID_DUAL_QY

`is_can_dual_imu_qy`

Dual IMU gyro and accelerometer values from accelerometer 0

Type	Field	Description
theta1	int16_t	Theta (rad/s, scaled by 1000, 3 decimal places precision)
vel1	int16_t	Acceleration (m/s^2, scaled by 100, 2 decimal places precision)
status	uints32_t	IMU status (see eImuStatus)

CID_DUAL_RZ

`is_can_dual_imu_rz`

Dual IMU gyro and accelerometer values from accelerometer 0

Type	Field	Description
theta2	int16_t	Theta (rad/s, scaled by 1000, 3 decimal places precision)
vel2	int16_t	Acceleration (m/s^2, scaled by 100, 2 decimal places precision)
status	uints32_t	IMU status (see eImuStatus)

CID_GPS1_POS

`is_can_gps1_pos_status`

GPS CNO Mean and GPS status flags

Type	Field	Description
status	uint32_t	(see eGpsStatus) GPS status: [0x000000xx] number of satellites used, [0x0000xx00] fix type, [0x00xx0000] status flags
cnoMean	uint32_t	(dBHz) Average of all satellite carrier to noise ratios (signal strengths) that are non-zero

CID_GPS1_RTK_REL

`is_can_gps1_rtk_rel`

RTK-GPS performance metrics

Type	Field	Description
arRatio	uint8_t	Ambiguity resolution ratio factor for validation
differentialAge	uint8_t	Age of differential (seconds)
distanceToBase	float	Distance to Base (m)
headingToBase	int16_t	Angle from north to vectorToBase in local tangent plane. (rad, scaled by 1000)

CID_ROLL_ROLLRATE

`is_can_roll_rollRate`

Combination or INS roll estimation and preintegrated IMU of both accelerometers

Type	Field	Description
insRoll	int16_t	INS Roll (scaled by 10000, 4 decimal places precision)
pImu1	int16_t	Delta theta (rad, scaled by 1000, 3 decimal places precision)
pImu2	int16_t	Delta theta (rad, scaled by 1000, 3 decimal places precision)

8. GNSS - RTK

8.1 Multi-band GNSS

8.1.1 Advantages

The advent of multi-band GNSS (multiple frequency global navigation satellite systems) improves accuracy by reducing the impact of errors caused by multi-path and atmospheric distortion. When compared to traditional single-band GNSS, dual-band technology provides about a 2x reduction in average position error (circular error probable - CEP). Benefits of multi-band GNSS systems like the uBlox ZED-F9P receiver include:

- Concurrent reception of GPS, GLONASS, Galileo and BeiDou for better coverage.
- Faster convergence time (GPS time to fix).
- More reliable / robust performance.
- ~2x reduction in average position error (CEP).
- Centimeter-level RTK position accuracy.
- Small and energy efficient module.
- Easy integration of RTK for fast time-to-market.

8.1.2 Overview

The uINS (GPS-INS) can be interfaced with external multi-band (multi-frequency) GNSS receiver(s) connected via serial port(s) to improve precision the EKF solution. The supported message protocols are uBlox binary and NMEA ASCII. The following are the GPS settings (accessible in the EvalTool GPS Settings tab and uINS `DID_FLASH_CONFIG.ioConfig` and `DID_FLASH_CONFIG.RTKCfgBits`):

Setting	**Value**
GPS Source	Serial port of the GNSS (serial 0 or 1)
GPS Type	GNSS model or protocol (ublox M8, ublox F9, or NMEA)
GPS RTK	*Position* for L1 RTK precision positioning *Compass* for L1 RTK Dual GNSS heading *F9 Position* for ZED-F9P mult-frequency RTK precision positioning *F9 Compass* for ZED-F9P multi-frequency Dual GNSS heading
GPS1 Timepulse	Source of the GNSS PPS time synchronization, uBlox GPS type only.

Refer to the Hardware section of this manual for serial port pinout information.

8.1.3 uBlox ZED-F9P GNSS

The uINS (GPS-INS) can be configured for use with uBlox ZED-F9P multi-band GNSS receivers. This can be done using either the EvalTool GPS Setting tab or the uINS `DID_FLASH_CONFIG.ioConfig` and `DID_FLASH_CONFIG.RTKCfgBits` fields.

GPS Ports	Value
GPS Source	serial 0, serial 1, or serial 2
GPS Type	ublox F9P
GPS1 Timepulse	*Disable* or uINS pin connected to ZED-F9P PPS

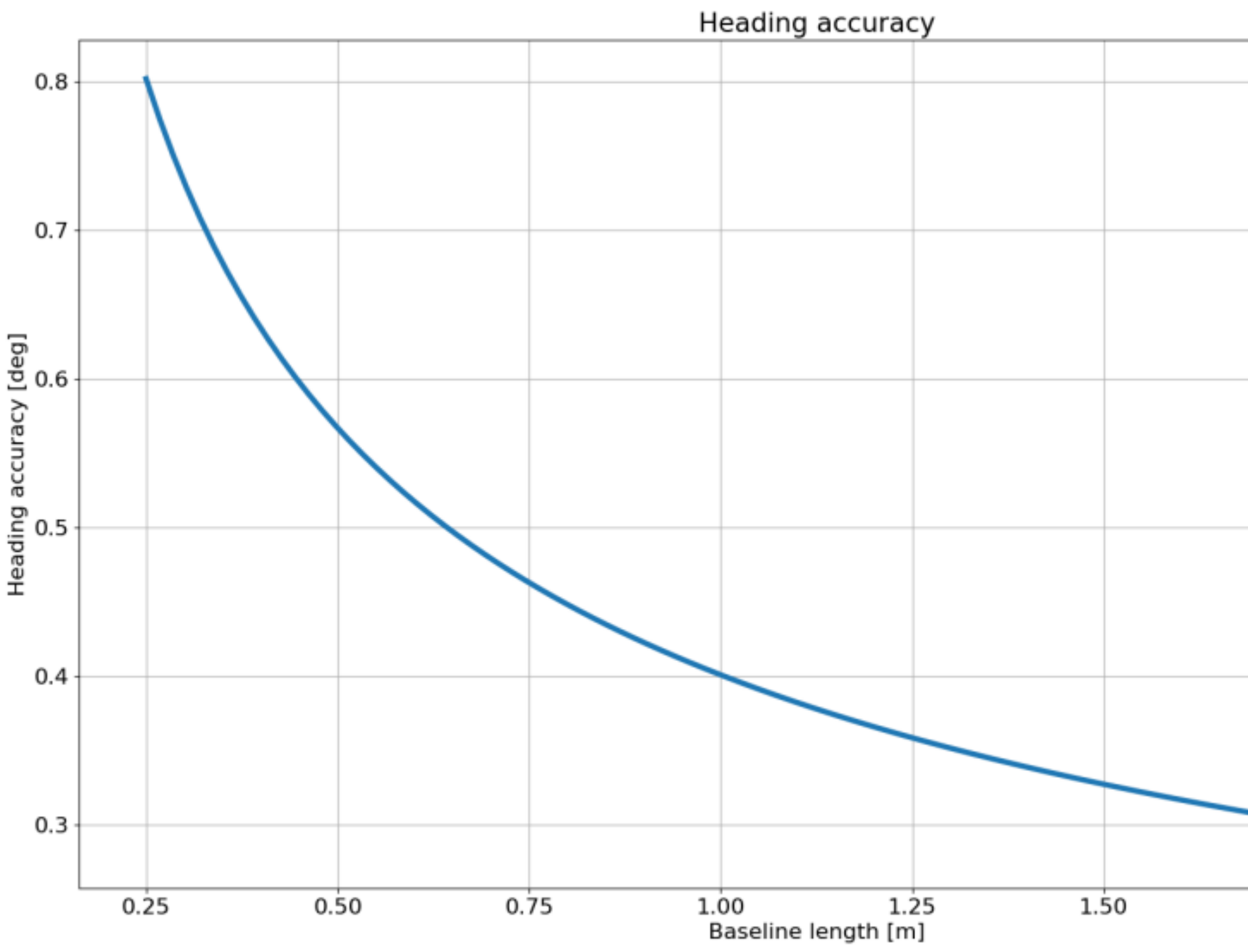
RTK Rover	Value
GPS RTK Mode	F9P Position or F9P Compass

RTK Base	Value
Serial Port 0 (Single GNSS only)	GPS1 - RTCM3
USB Port	GPS1 - RTCM3

The following sections detail how to interface and configure the uINS for operation using the ZED-F9P. See RTK precision positioning and RTK compassing for RTK operation principles.

DUAL ZED-F9 HEADING ACCURACY

When using two multi-band ZED-F9 GNSS receivers in moving baseline mode (RTK compassing) such as the EVB-2 Dual ZED-F9, the baseline error is composed of the measurement error plus the RTK solution error. The heading accuracy with ideal conditions is shown in the following plot.

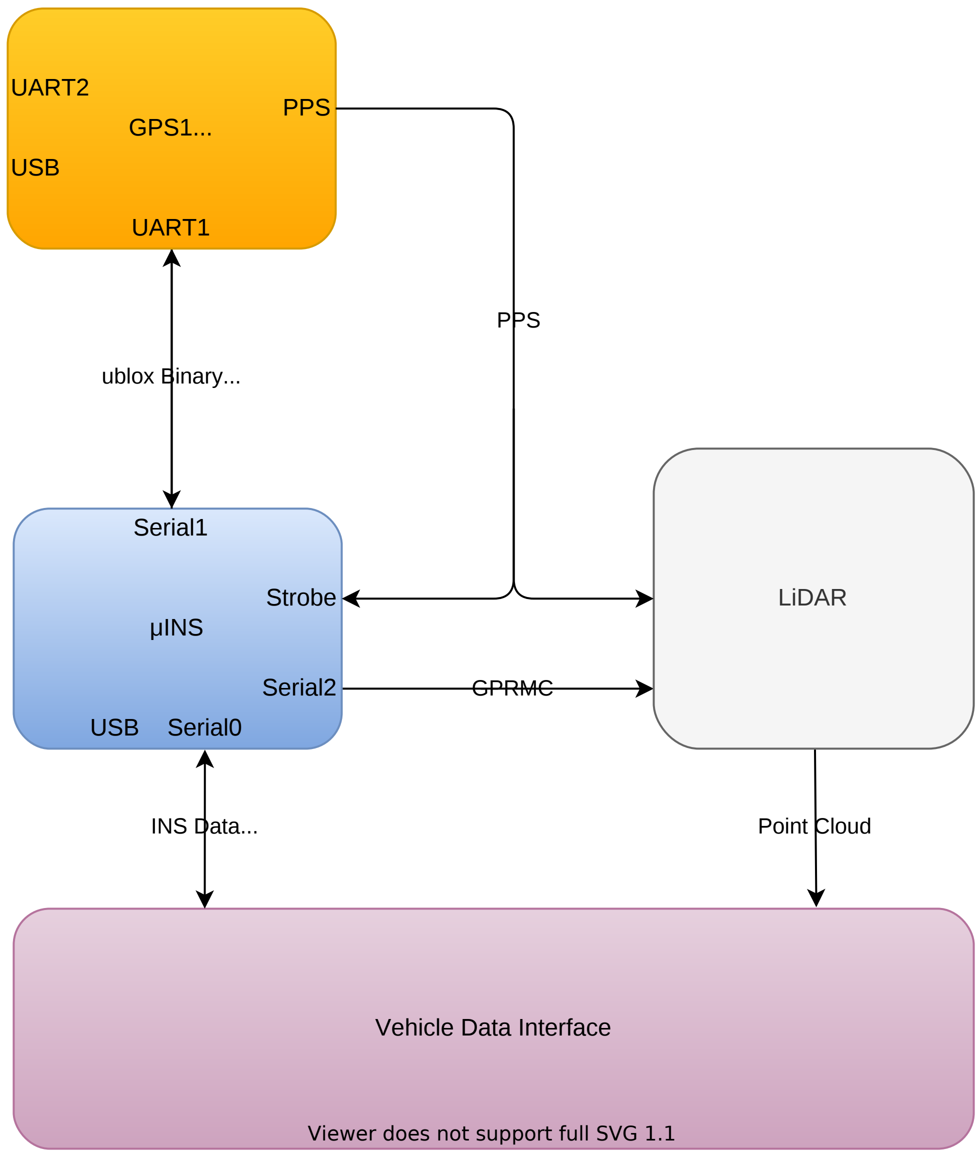


Typical Interface

The uINS will automatically configure the ZED-F9P for communications.

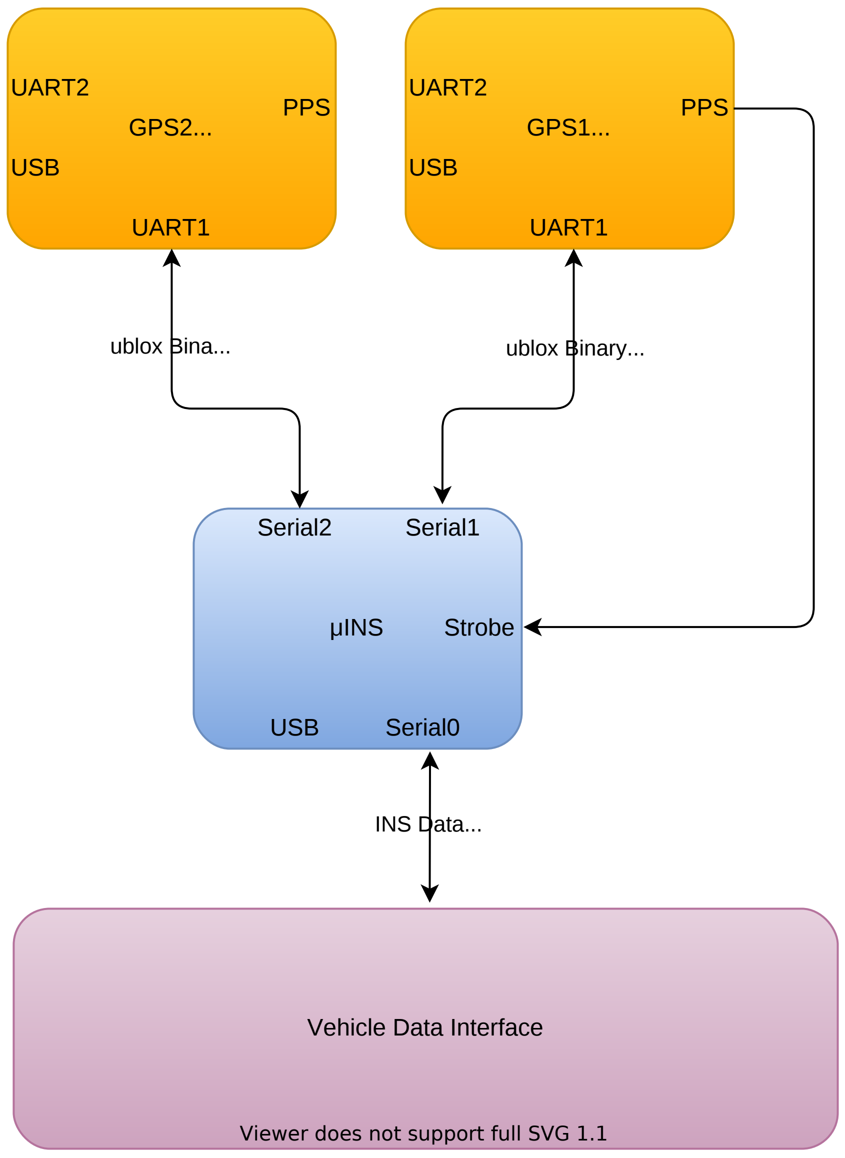
SINGLE GNSS RTK POSITIONING W/ LIDAR

RTK base messages (RTMC3) supplied to any of the uINS serial ports are forwarded to GPS1 for RTK positioning. The RTK precision position from GPS 1 is used in the uINS EKF solution. The uINS can be configured to output NMEA messages such as GPGGA or GPRMC on any serial port.



DUAL GNSS RTK POSITIONING AND RTK COMPASSING

RTK base messages (RTMC3) supplied to any of the uINS serial ports are forwarded to GPS1 for RTK positioning. RTK moving base messages from GPS1 are forwarded to GPS2 for RTK compassing. The RTK precision position from GPS 1 and the RTK compassing heading from GPS2 are used in the uINS EKF solution. Note that typically the Rugged-2 uses Serial 0 and the EVB-2 uses Serial 2 to communicate with the GPS2 F9P receiver.


UART2
GPS2...
PPS
USB
UART1
UART2
GPS1...
PPS
USB
UART1
ublox Bina...
ublox Binary...
Serial2
Serial1
µINS
Strobe
USB
Serial0
INS Data...
Vehicle Data Interface
Viewer does not support full SVG 1.1

Rugged-2

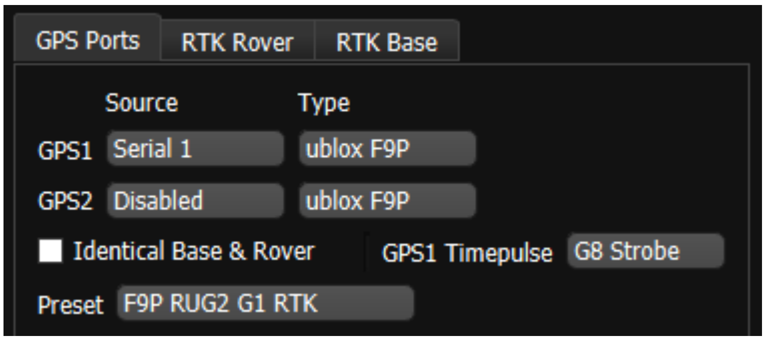
The Rugged-2 INS contains the either single or dual ZED-F9P onboard supporting RTK positioning and compassing. GPS 1 and GPS 2 are connected to serial ports 1 and 0 respectively on the uINS.

SINGLE GNSS SETTINGS

Use the following uINS settings with the Rugged-2-G1 (single GNSS receiver). These settings can be applied either using the EvalTool GPS Settings tab or the uINS `DID_FLASH_CONFIG.ioConfig` and `DID_FLASH_CONFIG.RTKCfgBits` fields.

GPS Ports

Set the GPS1 source to **Serial 1** and type to **ublox F9P**.



GPS Ports | RTK Rover | RTK Base

	Source	Type
GPS1	Serial 1	ublox F9P
GPS2	Disabled	ublox F9P

☐ Identical Base & Rover

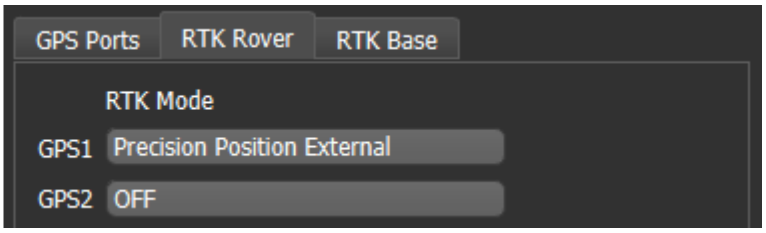
GPS1 Timepulse: G8 Strobe

Preset: F9P RUG2 G1 RTK

DID_FLASH_CONFIG	Value
ioConfig (firmware >=1.8.5)	0x0244a040

RTK Rover

Enable RTK rover mode by selecting **F9P Precision Position**.



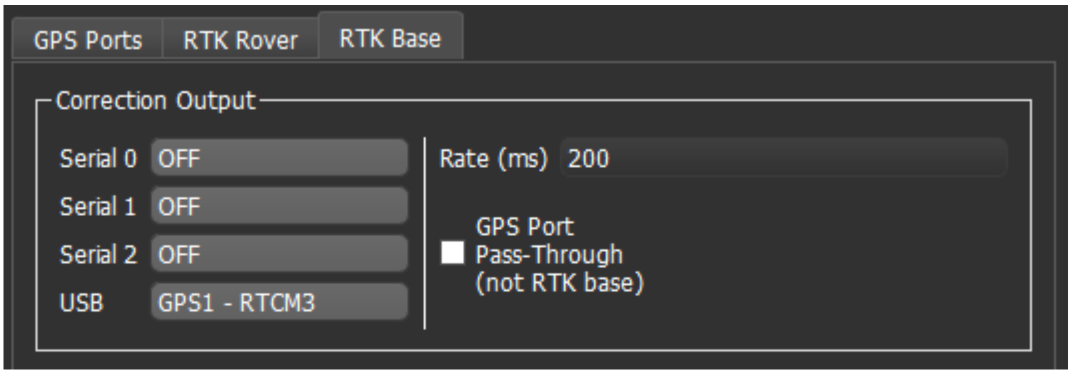
GPS Ports | RTK Rover | RTK Base

	RTK Mode
GPS1	Precision Position External
GPS2	OFF

DID_FLASH_CONFIG	Value
RTKCfgBits	0x00000002

RTK Base

To configuring a system as an RTK base, disable the RTK Rover by setting the GPS1 and GPS2 RTK Mode to **OFF**, and select the appropriate correction output port on the uINS.



GPS Ports | RTK Rover | RTK Base

Correction Output

Serial 0	OFF
Serial 1	OFF
Serial 2	OFF
USB	GPS1 - RTCM3

Rate (ms): 200

☐ GPS Port Pass-Through (not RTK base)

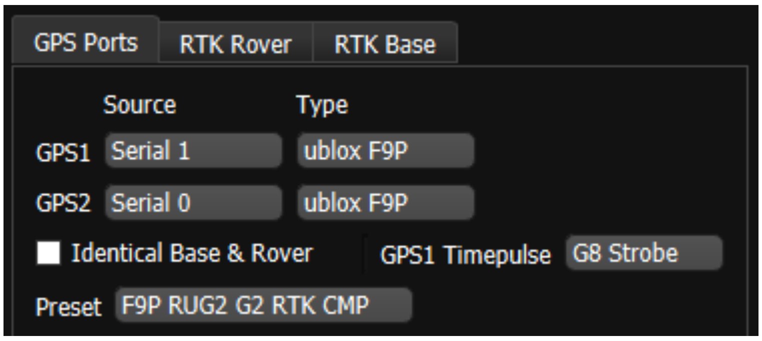
DID_FLASH_CONFIG	Value
RTKCfgBits	0x00000900

DUAL GNSS SETTINGS

Use the following uINS settings with the Rugged-2-G2 (dual GNSS receivers). These settings can be applied either using the EvalTool GPS Settings tab or the uINS `DID_FLASH_CONFIG.ioConfig` and `DID_FLASH_CONFIG.RTKCfgBits` fields.

GPS Ports

Set GPS 1 and 2 to source **Serial 1** and **Serial 0**. the serial port that the ZED-F9P is connected to and type to **ublox F9P**.



	Source	Type
GPS1	Serial 1	ublox F9P
GPS2	Serial 0	ublox F9P

☐ Identical Base & Rover

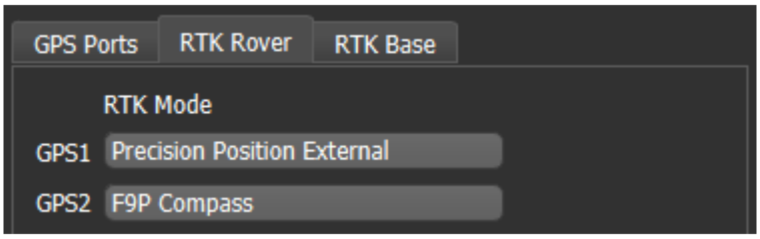
GPS1 Timepulse: G8 Strobe

Preset: F9P RUG2 G2 RTK CMP

DID_FLASH_CONFIG	Value
ioConfig (firmware >=1.8.5)	0x025ca040

RTK Rover

Enable RTK rover mode by selecting **Precision Position External**. **GPS1** is designated for **Precision Position External** and **GPS2** for **F9P Compass settings**. Either or both can be enabled at the same time.

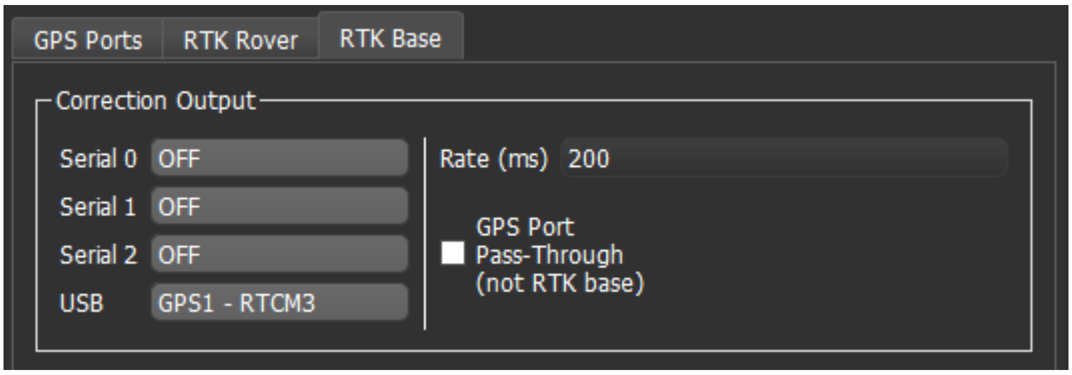


	RTK Mode
GPS1	Precision Position External
GPS2	F9P Compass

DID_FLASH_CONFIG	Value
RTKCfgBits	0x00000006

RTK Base

To configuring a system as an RTK base, skip the RTK rover settings, and select the appropriate correction output port on the uINS. Notice that uINS serial port 0 and 1 may be unavailable and occupied by the dual ZED-F9P receivers.



Correction Output

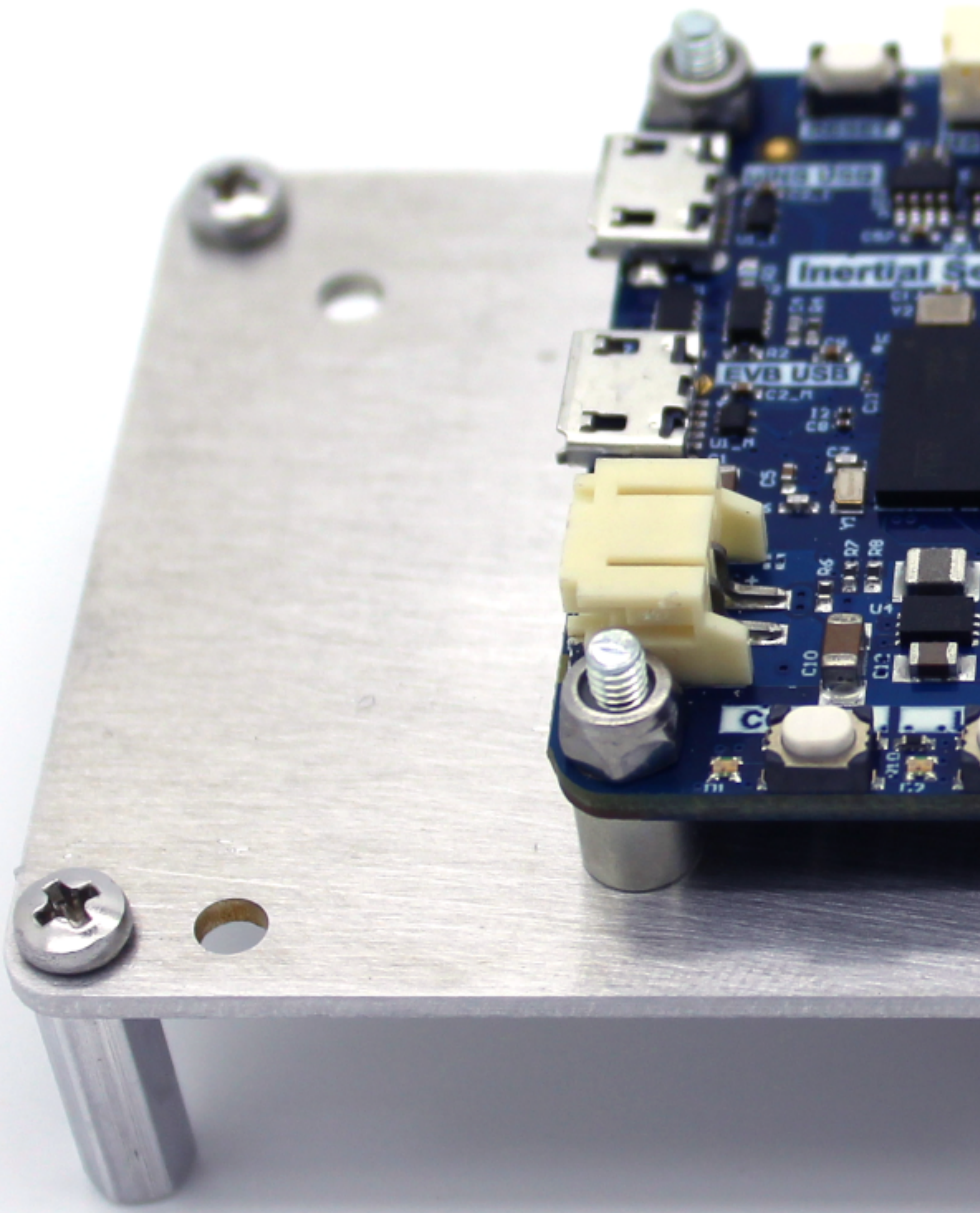
Serial 0	OFF
Serial 1	OFF
Serial 2	OFF
USB	GPS1 - RTCM3

Rate (ms): 200

☐ GPS Port Pass-Through (not RTK base)

DID_FLASH_CONFIG	Value
RTKCfgBits	0x00000900

EVB-2 to ZED-F9P


GNSS USB
EVB USB
Inertial Se

The ZED-F9P can be powered using the EVB-2 +3.3V output. Either serial 0 or serial 1 can be used to communicate with the ZED-F9P. See the EVB-2 H7 pinout for details.

SINGLE GNSS PINOUT

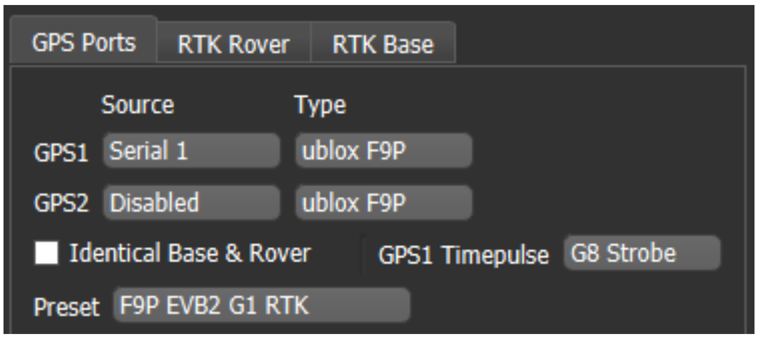
EVB-2	uINS	ZED-F9P
⬛ H7-1	GND	GND
🟥 H7-3	+3.3V	3V3
⬜ H7-11	Ser1 Tx	GPS RxD
🟪 H7-10	Ser1 Rx	GPS TxD
⬜ H7-12	G8 TIMEPULSE	PPS

SINGLE GNSS SETTINGS

Use the following settings when only one GPS receiver is connected to the uINS. These settings can be applied either using the EvalTool GPS Settings tab or the uINS `DID_FLASH_CONFIG.ioConfig` and `DID_FLASH_CONFIG.RTKCfgBits` fields.

GPS Ports

Set the serial port that the ZED-F9P is connected to and type to **ublox F9P**.

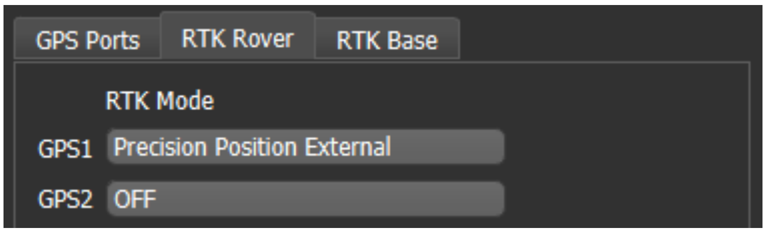


Source / Type
GPS1: Serial 1, ublox F9P
GPS2: Disabled, ublox F9P
Identical Base & Rover
GPS1 Timepulse: G8 Strobe
Preset: F9P EVB2 G1 RTK

DID_FLASH_CONFIG	Value
ioConfig (firmware >=1.8.5)	0x0244a040

RTK Rover

Enable RTK rover mode by selecting **Precision Position External**.

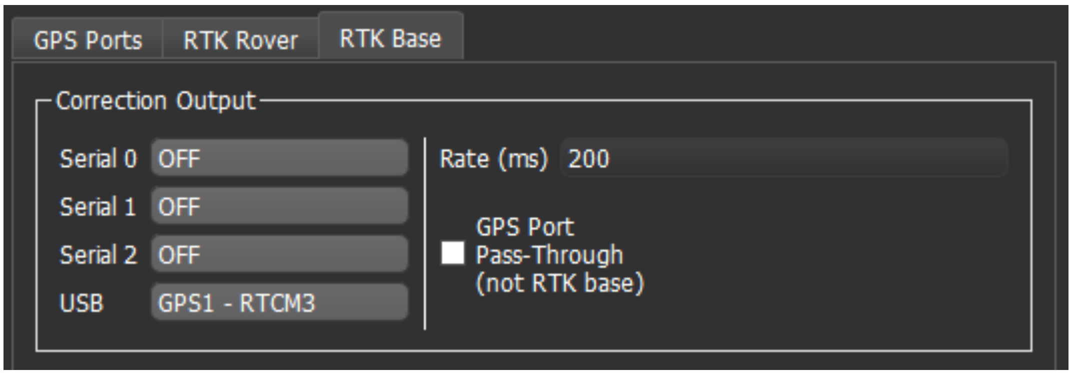


RTK Mode
GPS1: Precision Position External
GPS2: OFF

DID_FLASH_CONFIG	Value
RTKCfgBits	0x00000002

RTK Base

To configuring a system as an RTK base, disable the RTK Rover by setting the RTK Mode for GPS1 and GPS2 to off, and select the appropriate correction output port on the uINS.



Serial 0 OFF
Serial 1 OFF
Serial 2 OFF
USB GPS1 - RTCM3
Rate (ms) 200
☐ GPS Port Pass-Through (not RTK base)

DID_FLASH_CONFIG	Value
RTKCfgBits	0x00000900

Rugged-1 to ZED-F9P

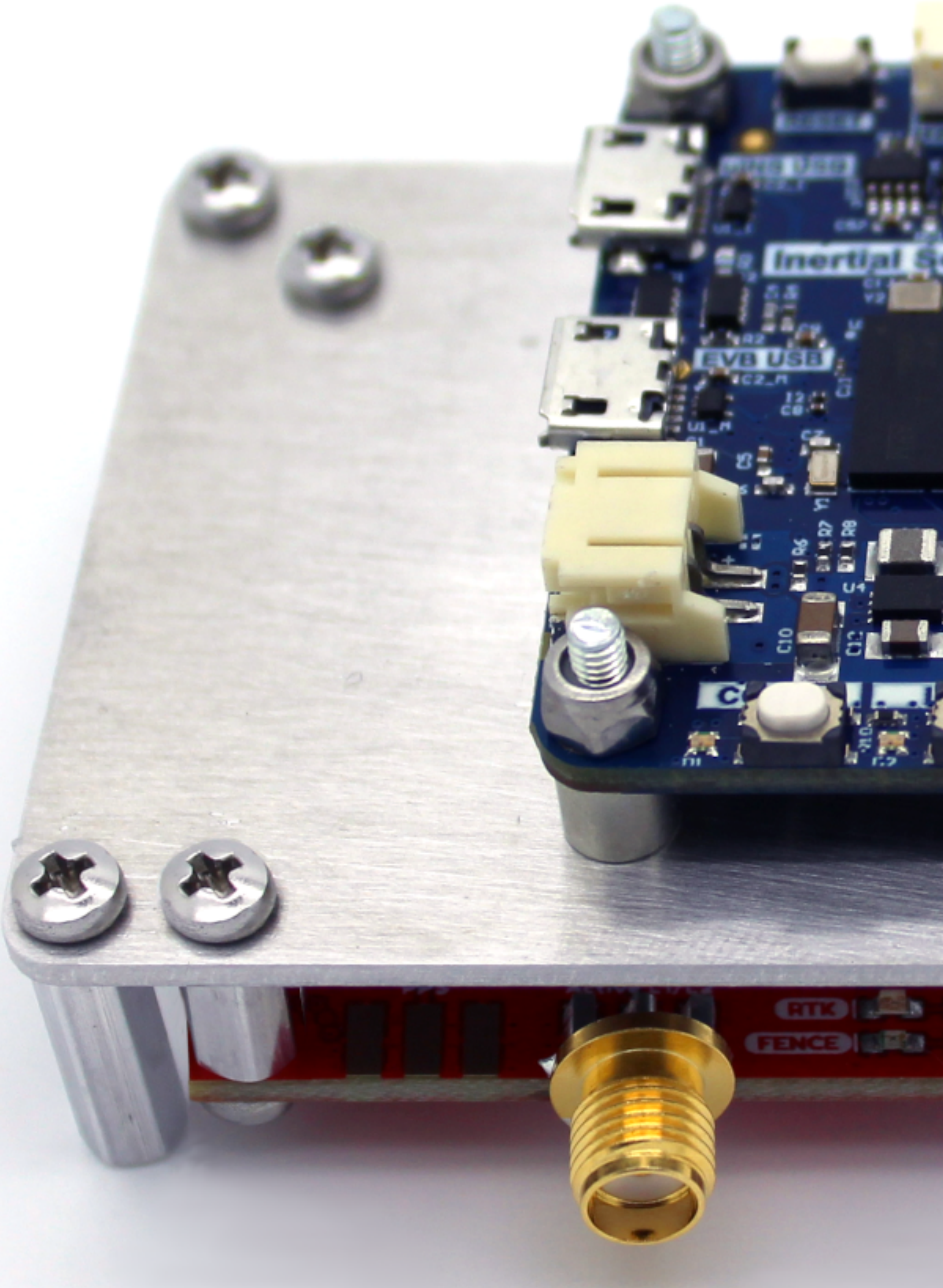
ZED-F9P
GND
3V3
RX2
TX2
RTCM Correction
GPS RTK-SMA
QWIIC
I2C
GND
3V3
SDA
SCL
FENCE
RTK
PPS
RST
SAFE
INT
PPS
PPS
Antenna Active L1/L2
PWR
PPS
RTK
FENCE
GND
5V
3V3
TX2
RX2
CS
RX/MOSI
TX/MISO
SCK
u blox
ZED-F9P-00B-02
F1100049102
109102000.1002
22
19/30
u-blox AG
All Rights Reserved

A +3.3V or +5V supply is needed to power the ZED-F9P when using the Rugged-1 uINS. A USB +5V supply can be used if available. The Rugged-1 must be configured for Serial Port 1 TTL voltage. See hardware configuration for Rugged v1.0 or Rugged v1.1 for details.

SETTINGS

See the single GNSS settings.

EVB-2 to Dual ZED-F9


RESET
Inertial S
EVB USB
RTK
FENCE

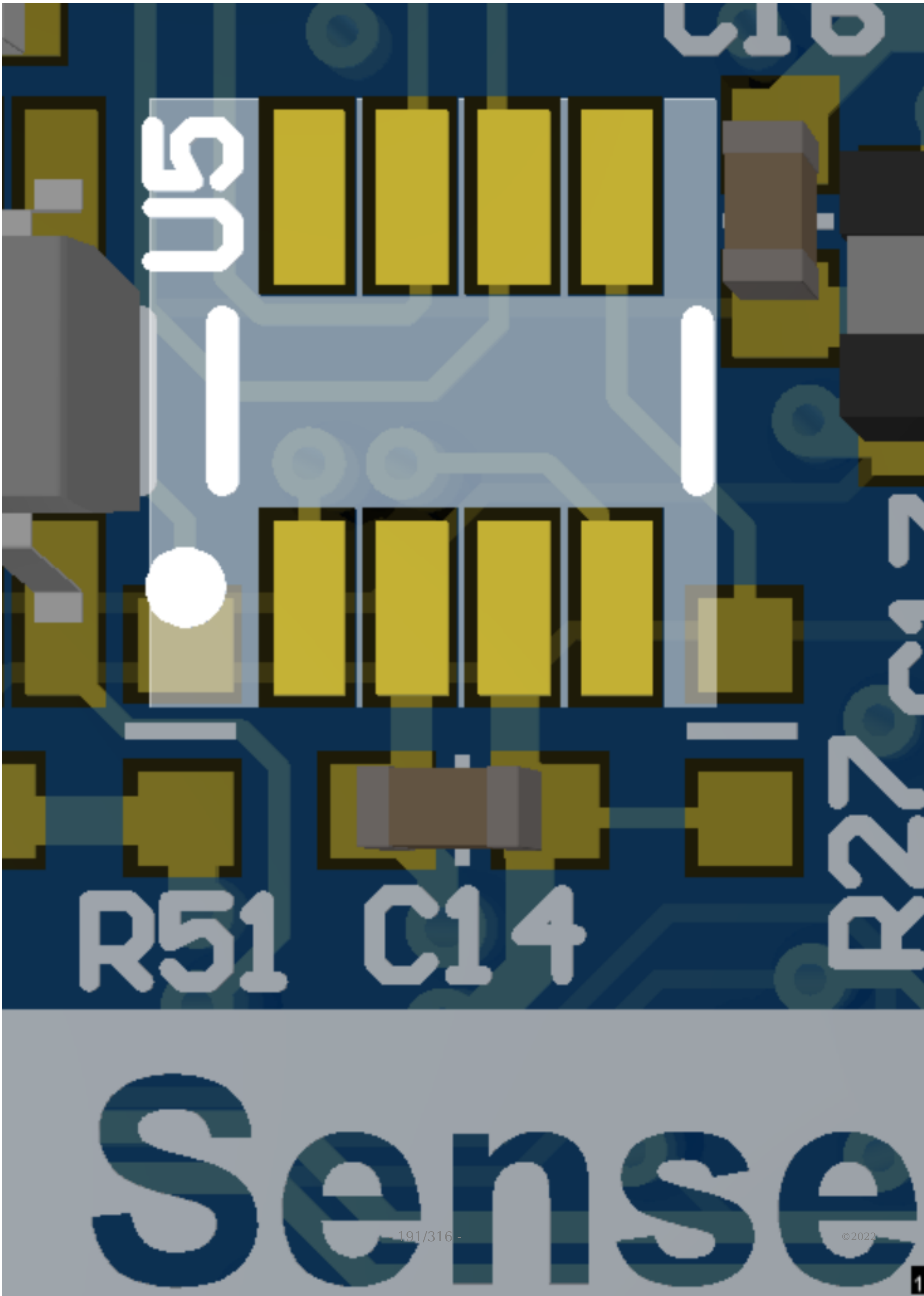
Two ZED-F9 units can be used to provide either or both multi-band RTK compassing and RTK positioning for the INS solution. The ZED-F9Ps can be powered using the EVB-2 +3.3V output. Serial port 0 and 1 must both be used to communicate with the ZED-F9P.

DUAL GNSS PINOUT

EVB-2	uINS	GPS1 ZED-F9P
⬛ H7-1	GND	GND
🟥 H7-3	+3.3V	3V3
⬜ H7-11	Ser1 Tx	RxD
🟪 H7-10	Ser1 Rx	TxD
◻ H7-12	G8 TIMEPULSE	PPS

EVB-2	uINS	GPS2 ZED-F9P
⬛ H7-2	GND	GND
🟥 H7-3	+3.3V	3V3
🟨 H7-6	Ser2 Tx*	RxD
🟫 H7-5	Ser2 Rx*	TxD

*R17 and R18 on the EVB-2 must be loaded with a zero ohm jumpers to connect Ser2 to H7 and U5 must be removed from the EVB-2.

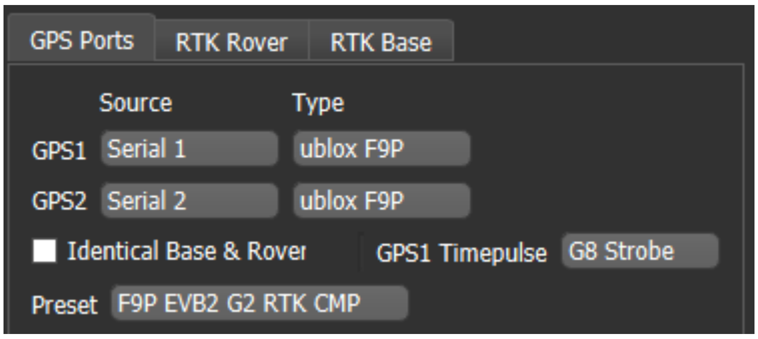

C16
U5
R51
C14
R27
C1
Sense

DUAL GNSS SETTINGS

The following settings are used when two GPS receivers are connected to the uINS. These settings can be applied either using the EvalTool GPS Settings tab or the uINS `DID_FLASH_CONFIG.ioConfig` and `DID_FLASH_CONFIG.RTKCfgBits` fields.

GPS Ports

Set the serial port that the ZED-F9P is connected to and type to **ublox F9P**.



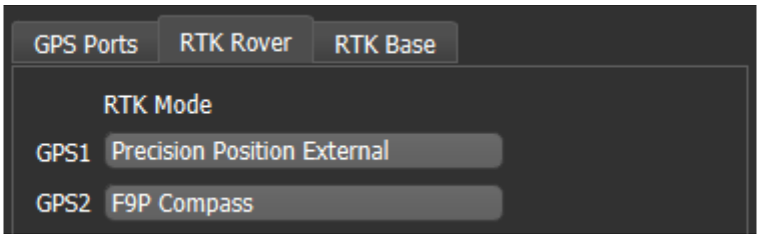
	Source	Type
GPS1	Serial 1	ublox F9P
GPS2	Serial 2	ublox F9P

☐ Identical Base & Rover
GPS1 Timepulse: G8 Strobe
Preset: F9P EVB2 G2 RTK CMP

DID_FLASH_CONFIG	Value
ioConfig (firmware >=1.8.5)	0x026ca040

RTK Rover

Enable RTK rover mode by selecting **Precision Position External**. **GPS1** is designated for **Precision Position External** and **GPS2** for **F9P Compass settings**. Either or both can be enabled at the same time.

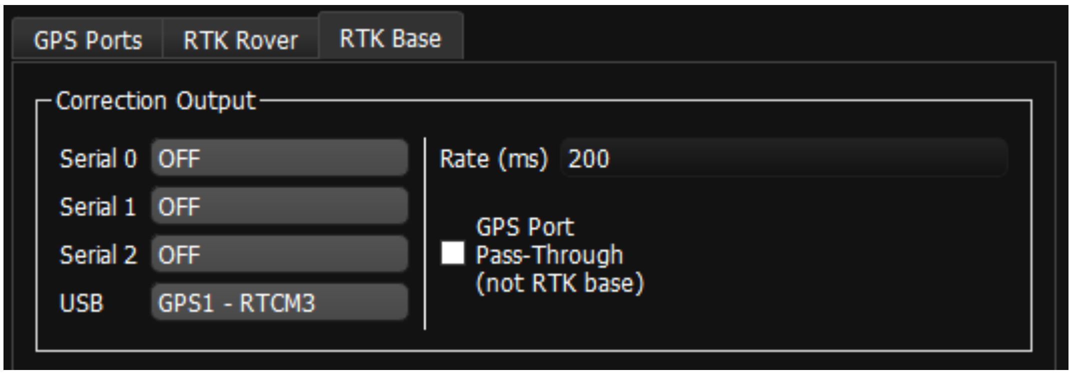


	RTK Mode
GPS1	Precision Position External
GPS2	F9P Compass

DID_FLASH_CONFIG	Value
RTKCfgBits	0x00000006

RTK Base

To configuring a system as an RTK base, skip the RTK rover settings, and select the appropriate correction output port on the uINS. Notice that uINS serial port 0 and 1 may be unavailable and occupied by the dual ZED-F9P receivers.



GPS Ports | RTK Rover | RTK Base

Correction Output

Serial 0	OFF
Serial 1	OFF
Serial 2	OFF
USB	GPS1 - RTCM3

Rate (ms) 200

☐ GPS Port Pass-Through (not RTK base)

DID_FLASH_CONFIG	Value
RTKCfgBits	0x00000900

8.1.4 RTK Base Messages

In RTK mode, the ZED-F9P requires RTCM version 3 messages supporting DGNSS according to RTCM 10403.3.

ZED-F9 Rover Messages

The ZED-F9P operating in RTK rover mode can decode the following RTCM 3.3 messages.

Message type	Description
RTCM 1001	L1-only GPS RTK observables
RTCM 1002	Extended L1-only GPS RTK observables
RTCM 1003	L1/L2 GPS RTK observables
RTCM 1004	Extended L1/L2 GPS RTK observables
RTCM 1005	Stationary RTK reference station ARP
RTCM 1006	Stationary RTK reference station ARP with antenna height
RTCM 1007	Antenna descriptor
RTCM 1009	L1-only GLONASS RTK observables
RTCM 1010	Extended L1-only GLONASS RTK observables
RTCM 1011	L1/L2 GLONASS RTK observables
RTCM 1012	Extended L1/L2 GLONASS RTK observables
RTCM 1033	Receiver and antenna description
RTCM 1074	GPS MSM4
RTCM 1075	GPS MSM5
RTCM 1077	GPS MSM7
RTCM 1084	GLONASS MSM4
RTCM 1085	GLONASS MSM5
RTCM 1087	GLONASS MSM7
RTCM 1094	Galileo MSM4
RTCM 1095	Galileo MSM5
RTCM 1097	Galileo MSM7
RTCM 1124	BeiDou MSM4
RTCM 1125	BeiDou MSM5
RTCM 1127	BeiDou MSM7
RTCM 1230	GLONASS code-phase biases
RTCM 4072.0	Reference station PVT (u-blox proprietary RTCM Message)

ZED-F9 Base Output Messages

The ZED-F9P operating in RTK base mode will generate the following RTCM 3.3 output messages depending on whether the satellite constellation have been enabled. See the Constellation Selection for information on enabling and disabling satellite constellations.

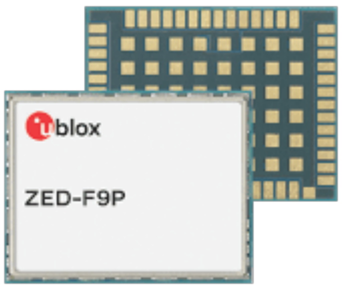
Message Type	Period (sec)	Description
RTCM 1005	2	Stationary RTK reference station ARP
RTCM 1074	0.4	GPS MSM4
RTCM 1077	0.4	GPS MSM7
RTCM 1084	0.4	GLONASS MSM4
RTCM 1087	0.4	GLONASS MSM7
RTCM 1094	0.4	Galileo MSM4
RTCM 1097	0.4	Galileo MSM7
RTCM 1124	0.4	BeiDou MSM4
RTCM 1127	0.4	BeiDou MSM7
RTCM 1230	2	GLONASS code-phase biases

NTRIP Messages

The NTRIP server must provide the necessary subset of RTCM3 messages supported by the uINS-RTK. See the NTRIP page for an overview of NTRIP.

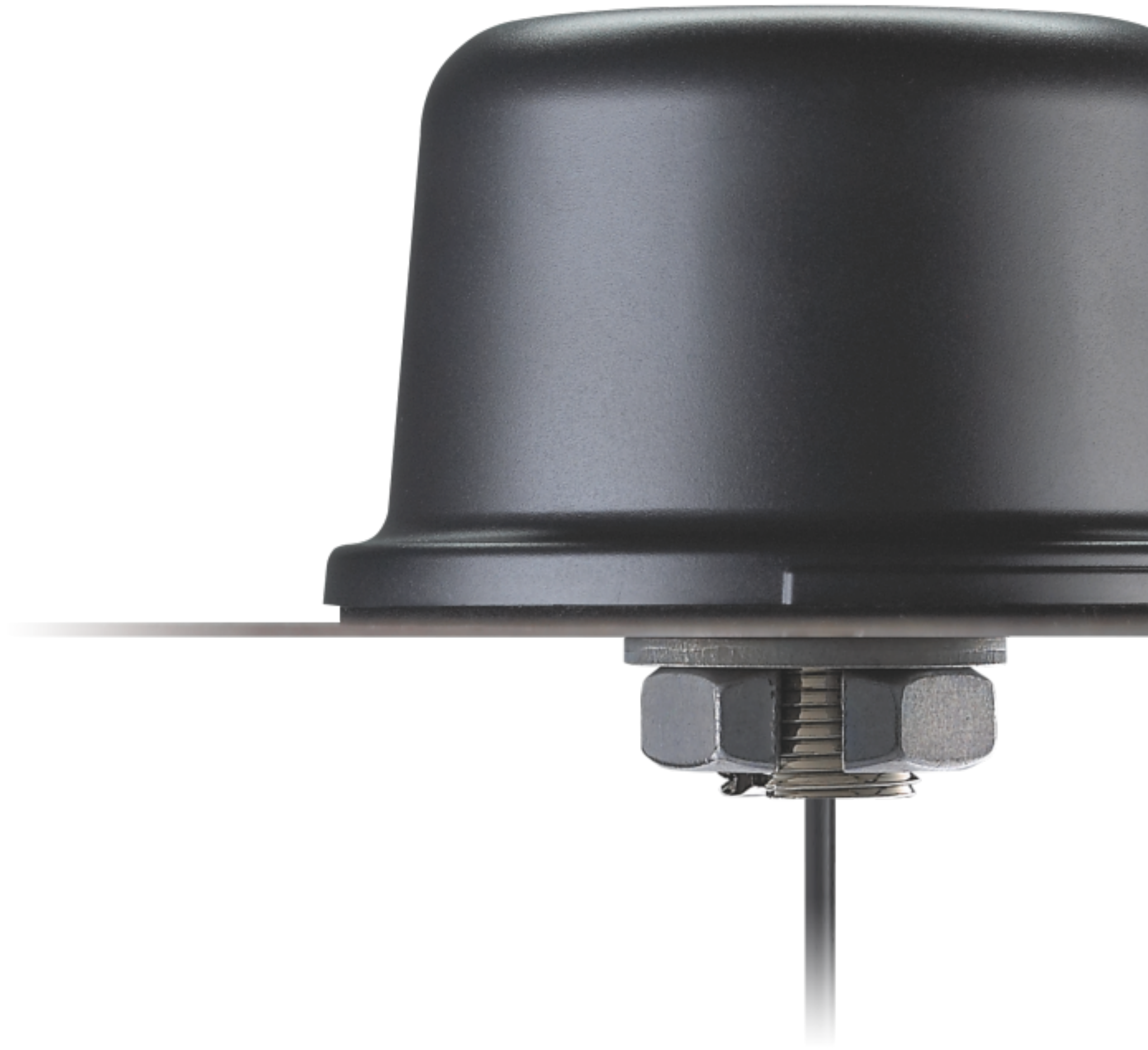
8.1.5 Multi-Band GNSS Components

The following is a list of the ZED-F9P GNSS receivers and compatible antenna(s).

Item


Item

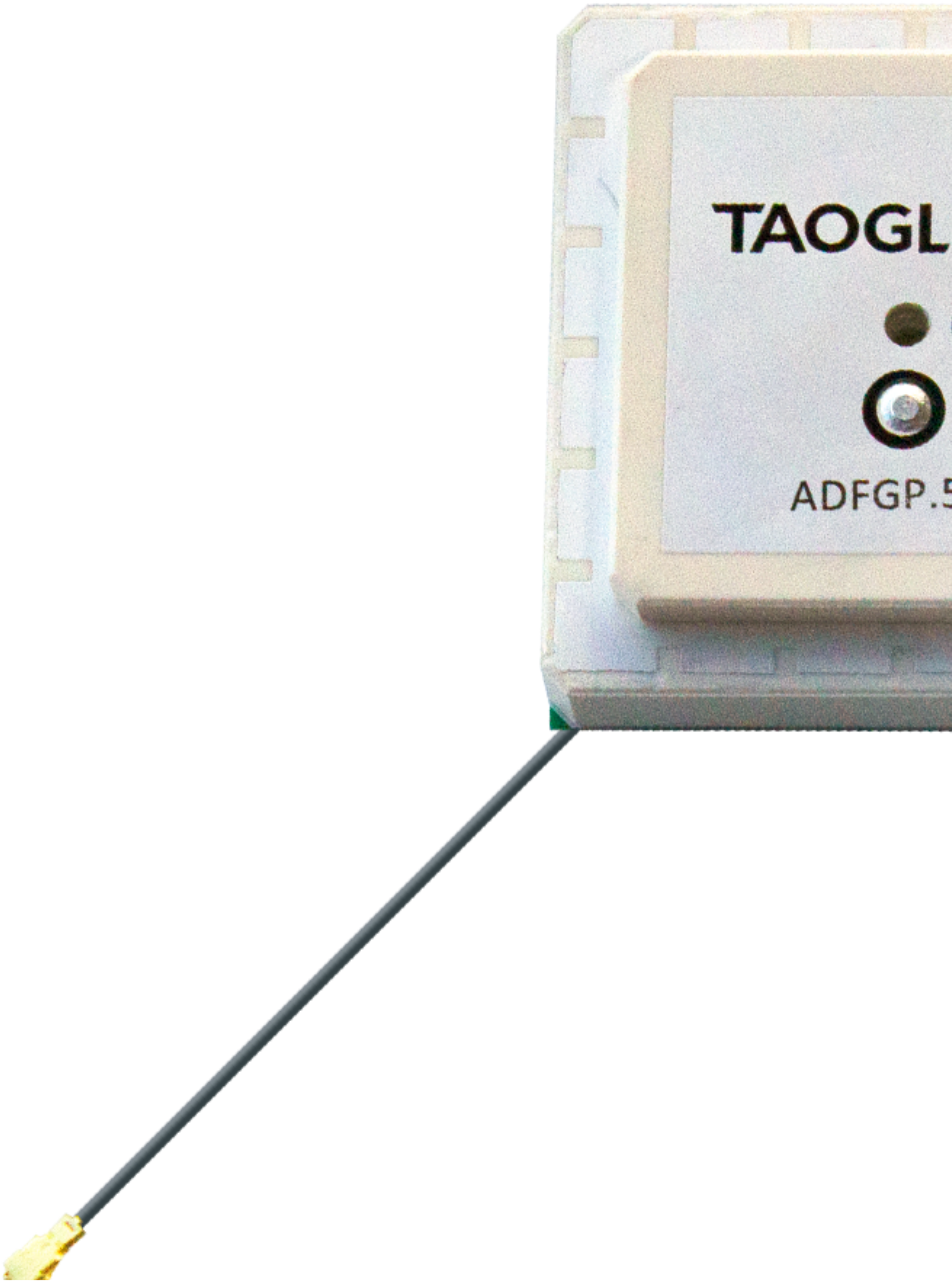
Item



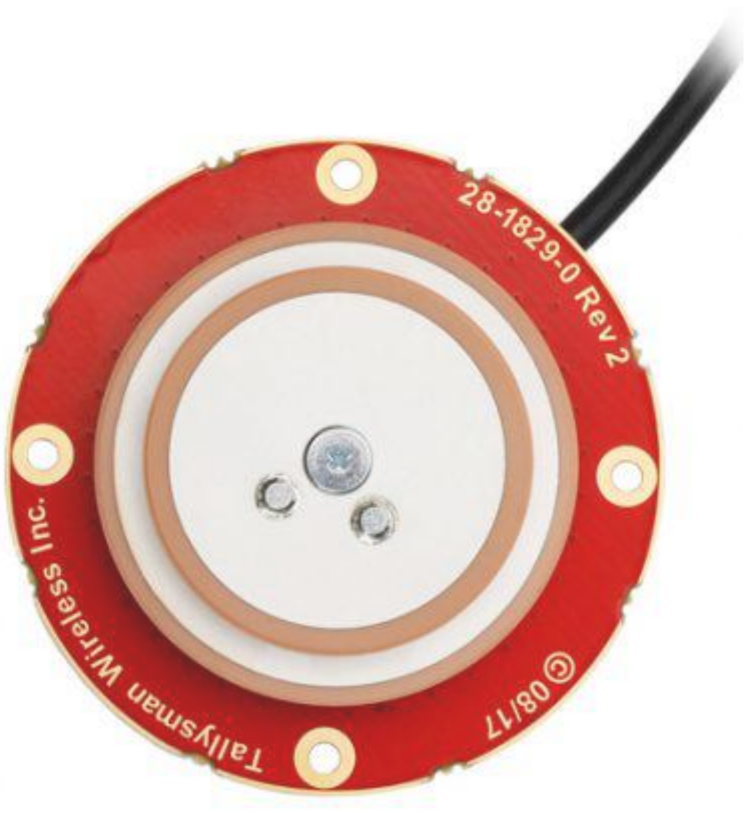
Item

Item

Item


TAOGL
ADFGP.5

Item


28-1829-0 Rev 2
©08/17
Tallysman Wireless Inc.

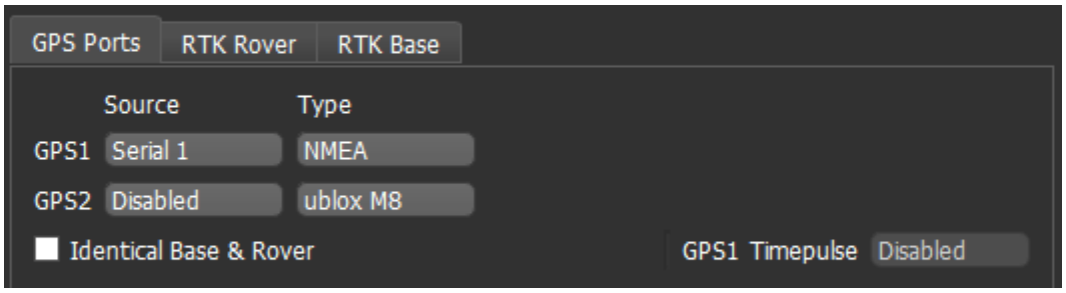
©10/16
Tallysman Wireless Inc.

8.2 External NMEA GNSS

GNSS receivers that output NMEA ascii protocol can be used to aid the uINS EKF.

8.2.1 Configure uINS for NMEA GNSS Input

1. Set serial port baudrate, matching DID_FLASH_CONFIG.serXBaudRate.
2. Configure GPS1 using EvalTool GPS Setting tab or the DID_FLASH_CONFIG.ioConfig.



	Source	Type
GPS1	Serial 1	NMEA
GPS2	Disabled	ublox M8

☐ Identical Base & Rover

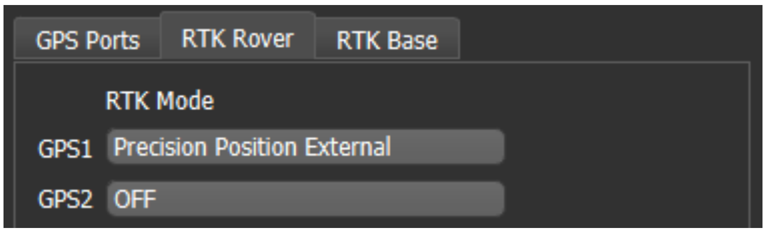
GPS1 Timepulse: Disabled

DID_FLASH_CONFIG	Value
ioConfig (firmware >=1.8.5)	0x00840040

3. Enabled the NMEA messages on the external GNSS:

Message	Description
GNS	GNS Fix data (preferred) or **GGA** - Time, position, and fix related data.
ZDA	UTC time and date.
RMC	Position, velocity, and time (optional / recommended).
GSV	Satellite signal strength (optional / recommended).

4. If RTK positioning is supported by the NMEA receiver, Enable RTK rover mode by selecting **Precision Position External**. This will run the INS kalman filter in high accuracy mode and forward any RTK base station corrections to the external GNSS receiver.



	RTK Mode
GPS1	Precision Position External
GPS2	OFF

DID_FLASH_CONFIG	Value
RTKCfgBits	0x00000002

8.2.2 Electrical Interface

The external NMEA GNSS receiver can be connected to Serial 0, Serial 1, and Serial 2 ports (3.3V TTL UART) on the uINS. See the PCB Module hardware page for a description of the uINS pinout. Serial 0 and 2 can be accessed on the main connector of Rugged-1 and Rugged-2 and all serial ports can be accessed on header H7 of the EVB-2.

8.2.3 Enabling NMEA on ZED-F9P

The recommended procotol with the uINS and ZED-F9P receiver is the uBlox binary protocol. However, the ZED-F9 can operate using NMEA protocol if necessary. The following steps can be used to enable NMEA protocol output on the ublox ZED-F9P receiver.

1. Enable NMEA output using the u-blox u-center application.
    - **Set the configuration**: (ublox u-center menu -> View -> Configuration View) change the following. You must press the "Send" button to apply each change.
        - **PRT (Ports)** - Set Baudrate to match the GPS port baudrate (i.e. ser1BaudRate 921600)
        - **PRT (Ports)** - Enable NMEA on the connected port/UART
        - **PRT (Ports)** - Enable RTCM3 on the connected port/UART is using RTK
        - **RATE (Rates)** - Measurement Period: 200ms
        - **RATE (Rates)** - Navigation Rate: 1cyc
        - **MSG (Messages)** - Enable NMEA messages listed above for the connected port/UART (i.e. UART1 On)
            - F0-0D NMEA GxGNS
            - F0-08 NMEA GxZDA
            - F0-04 NMEA GxRMC
            - F0-03 NMEA GxGSV
    - **Save the configuration**: Send the CFG (Configuration) to 1 - FLASH or press the "Save Config" button with the small gear save icon (or menu Receiver -> Action -> Save Config).

8.3 GNSS Antennas

8.3.1 Selecting a GNSS Antenna

- Using a passive GNSS antenna is possible but not recommended. This requires a high RHCP antenna gain, good view of the sky, and a short matched/tuned 50 Ω input impedance line. This option may be appropriate to minimize BOM costs.
- Best performance is achieved by using an active antenna with integrated LNA. The LNA gain must be >17dB for standard GPS-INS use.
- For RTK and dual antenna (GPS compassing) use, the following characteristics are recommended: gain >26dB, multipath signal rejection, better signal to noise ratio, and improved carrier phase linearity.
- Antennas with integrated SAW filter may be necessary to reject interference from near frequencies or harmonic signals, such as wireless and LTE.
- For RTK and dual antenna applications we recommend dual feed (dual element) GNSS antennas

8.3.2 GNSS Antenna Integration Considerations

GNSS Antenna Ground Plane

A GNSS antenna ground plane blocks multipath signals, creating a shadow area for the antenna to hide in. The ground plane is acting as an RF blocking device. It is made of any material that attenuates (or totally blocks or reflects) RF signals. It creates a shadow area for the antenna to hide in. That shadow is a cone above the ground plane. Any signals that come down from the satellites and are bouncing back upward from the earth can't get to the antenna. Only signals coming directly from above can get to the antenna. The distance of the physical antenna above the ground plane changes the shape of the RF blocked shadow area.

The signal gain on some antennas can be improved by increasing the ground plane size up to a given size. Beyond that given size the antenna gain is not affected much.

A ground plane width of 8 to 12 cm is typically large enough for most applications.

HELPFUL LINKS:

u-Blox: RF design considerations for GNSS receivers Application Note

Taoglas: GPS Patch Integration Application Note

electronics.stackexchange.com: How big a ground plane does a GPS antenna need?

8.3.3 Recommended GNSS Components

The following components are optional components that may be used with the µINS, µAHRS, and µIMU.

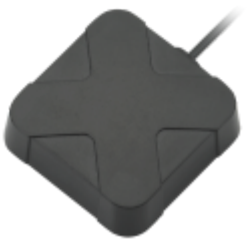
Frequencies: GPS (L1), GLONASS (G1), Beidou (B1), and Galileo (E1).

Recommended for RTK indicates the GNSS antenna will have better performance for applications using RTK and dual GNSS antenna (GNSS compassing).

For multi-frequency GNSS antennas, see Purchasing the ZED-F9.

Enclosed GNSS Antennas

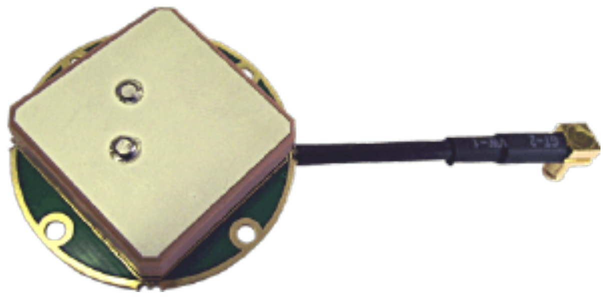
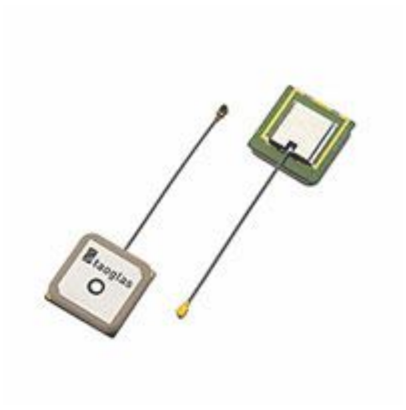
The following GNSS antennas have an environmental case rated at IP67 or better.

	Manufacturer Part Number	**Description**
	Tallysman TW4722	Magnet Mount, L1/G1/B1/E1 freq., Dual-feed, 26dB LNA, SAW filter, SMA 3m cable, **Recommended for RTK**
	Tallysman TW2712	Through-Hole Mount, L1/G1/B1/E1 freq., Dual-feed, 26dB LNA, SAW filter, SMA 3m cable, **Recommended for RTK**
	Tallysman TW3712	Through-Hole Mount, L1/G1/E1/B1 freq., Dual-feed, 26dB LNA, SAW filter, SMA 3m cable, **Recommended for RTK**
	Taoglas Limited AA.162.301111	Magnet Mount, L1/G1 freq., 29dB LNA, SAW Filter, SMA 3m cable.
	Taoglas Limited AA.171.301111	Magnet Mount, L1/G1/E1/B1 freq., 29dB LNA, SAW Filter, SMA 3m cable.

	Manufacturer Part Number	Description
	Abracon LLC APAMPG-130	Magnet Mount, L1/G1 freq., 30dB LNA, SAW Filter, SMA 3m cable.

OEM GNSS Antennas

These non-enclosed embedded antennas have exposed PCA with no environmental protection. OEM antennas are easily detuned by the local environment (caused by mounting inside enclosures). We recommend contacting the manufacturer for custom tuning services for optimized integration into OEM end-user modules.

	Manufacturer Part Number	Description
	Tallysman TW2708	Dual-feed, L1/G1/B1/E1 freq., 1-3 dB axial ratio, 28dB LNA and SAW Filter, 56mm dia. x 7.6mm, RG174 cable, **Recommended for RTK**
	Tallysman TW1722	Dual-feed, L1/G1/B1/E1 freq., 28dB LNA and SAW Filter, 35mm dia. x 6mm, RG174 cable, **Recommended for RTK**
	Taoglas Limited AGGBP.25A. 07.0060A	L1/G1/B1/E1 freq.. 28dB LNA and SAW Filter, 25x25mm, U.FL 6cm cable

Related GNSS Parts

	Manufacturer Part Number	**Description**
GNSS Backup Battery	Seiko Instruments MS621T-FL11E	Coin, 6.8mm 3V Lithium Battery Rechargeable (Secondary) 3mAh
GNSS Backup Battery	Panasonic ML-614S/FN	Coin, 6.8mm 3V Lithium Battery Rechargeable (Secondary) 3.4mAh

8.4 GNSS Satellite Constellations

The uINS supports onboard M8 and external (off-board) uBlox GNSS receivers. These receivers use multiple GNSS constellations in the global positioning solution.

The M8 receiver supports use of 3 concurrent constellations and the ZED-F9 receivers support 4 concurrent constellations (i.e. GPS, GLONASS, Galileo, and BeiDou).

8.4.1 Constellation Selection

The satellite constellations can be enabled or disabled by setting the corresponding enable bits in `DID_FLASH_CONFIG.gnssSatSigConst` as defined by eGnssSatSigConst in data_sets.h. The following are commonly used and recommended configuration groups.

```
// 3 constellations is supported by uINS onboard M8 reciever.
// (SBAS is not considered a constellation)
DID_FLASH_CONFIG.gnssSatSigConst = 0x133F   // GPS/QZSS, Galileo, GLONASS, SBAS
DID_FLASH_CONFIG.gnssSatSigConst = 0x10FF   // GPS/QZSS, Galileo, BeiDou, SBAS
DID_FLASH_CONFIG.gnssSatSigConst = 0x130F   // GPS/QZSS, GLONASS, SBAS

// 4 constellations is supported by ZED-F9 receiver (not uINS onboard M8 receiver).
DID_FLASH_CONFIG.gnssSatSigConst = 0x13FF   // GPS/QZSS, Galileo, GLONASS, BeiDou, SBAS
```

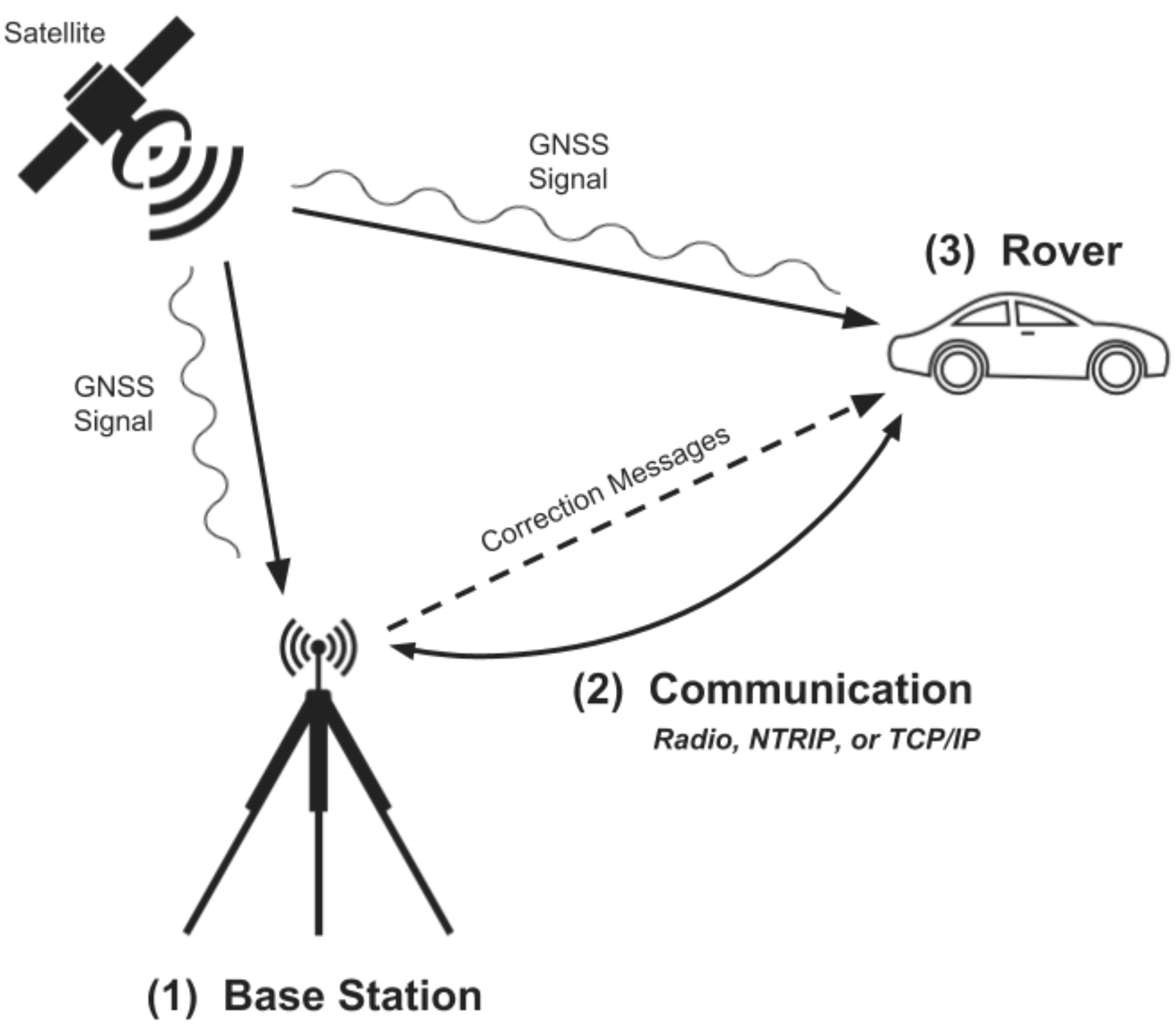
8.5 RTK Positioning

8.5.1 RTK Precision Positioning

Overview

Real Time Kinematic (RTK) is a precision satellite positioning technique which utilizes a base station to transmit position corrections to a receiver. The Inertial Sense RTK solution provides centimeter level position accuracy.

To use RTK, a base station, arover (receiver), and a method to send corrections from the base to the rover are required.



See the multi-band GNSS section for details on using our multi-frequency ZED-F9 GNSS system.

RTK Hardware Setup

BASE STATION OPTIONS

Any of the following devices can be used as a RTK base station. All Inertial Sense base station options require a GPS antenna.

- **Inertial Sense EVB 2** - Sends corrections using the onboard 915 MHz radio, the onboard WiFi module, either serial port, or USB.
- **Inertial Sense µINS module, EVB 1 or Rugged** - Sends corrections on either serial port or USB that can then be forwarded to a rover using a communication method of choice.
- **3rd Party Base Station** - e.g. Emlid Reach Receiver.
- **Public NTRIP Caster** - e.g. CORS Network.

ROVER OPTIONS

The following configurations can be used for the RTK rover:

- **Inertial Sense EVB 2** - Can receive corrections via the onboard 915 MHz radio, onboard WiFi module, serial ports, or USB.
- **Inertial Sense µINS module, EVB 1 or Rugged** - Can receive corrections via either serial port or USB.

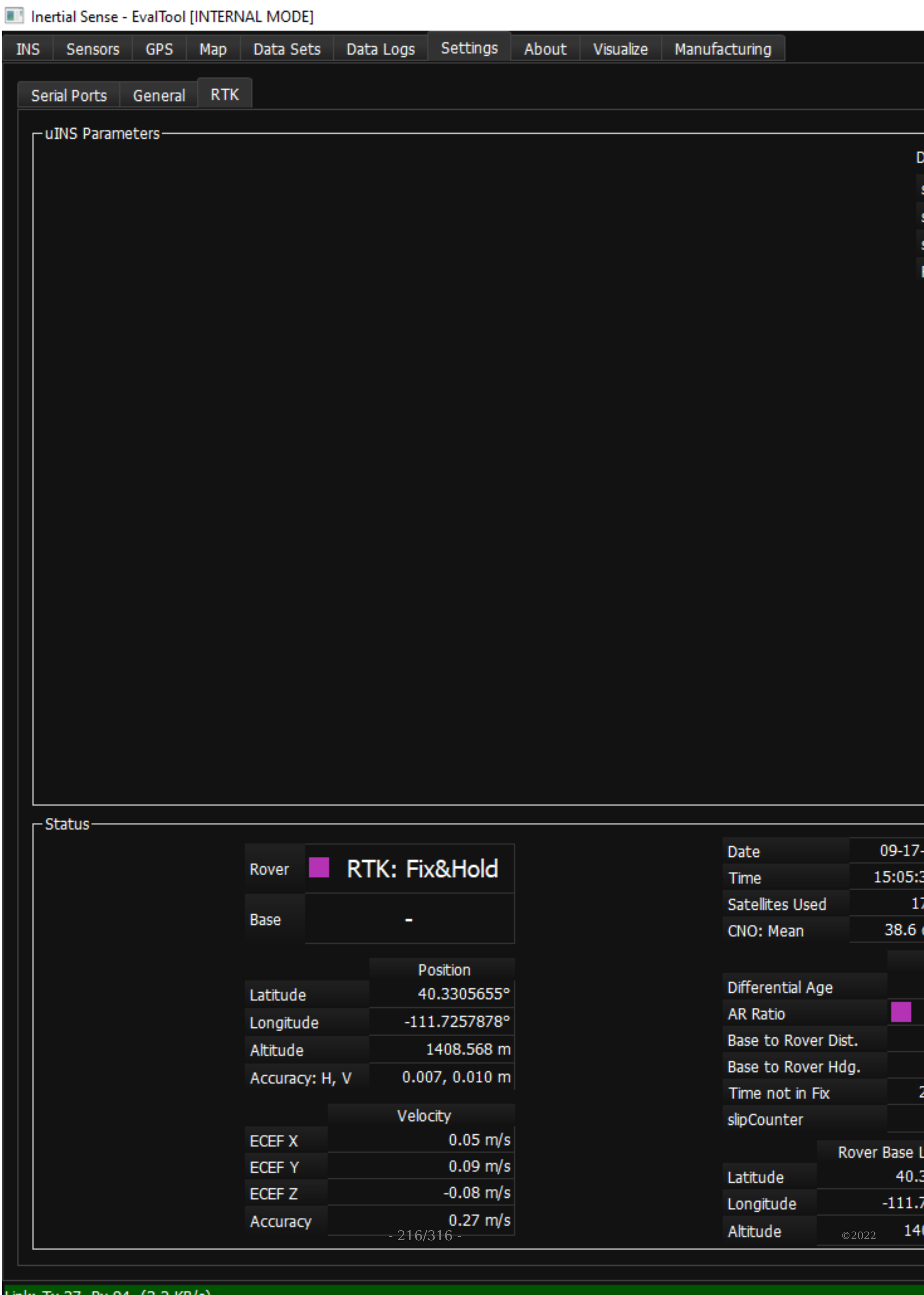
BASE TO ROVER COMMUNICATION

1. **Direct Serial** - Using USB, RS232, RS422/485, or TTL to pass corrections from Base to Rover.
2. **Radio Link** - Inertial Sense EVB 2 uses the Digi Xbee Pro SX module to send RTK corrections. Other communication methods such as Bluetooth may also work for the chosen application.
3. **NTRIP** - Transmits RTK correction data over the Internet. To receive messages with NTRIP, the user must supply a URL, port number, and mount point . Often a username and password are also required.
4. **TCP/IP** - A protocol for communicating directly between computers. In order to receive messages using TCP/IP, an address (IP Address or DNS) must be suppled to the Base where the corrections will be transmitted.

How to Know RTK is Working

USING THE EVALTOOL

1. Connect the µINS Rover to a computer with the EvalTool running. Open the comport for the unit in the Settings > Serial Ports.
2. Navigate to Settings > RTK.
3. Under the Status section, RTK functionality can be verified in 3 ways:
    - Status field will show Single. Over the course of several minutes this status will change to Float then Fix.
    - The Differential Age will show a timestamp that increments and resets back to zero about every second. This shows that the Rover is receiving Base messages.
    - The Accuracy: H, V will show a large number at first. This number will decrease over time as the system acquires RTK Fix. Once in Fix, this number will average at +- 0.08, 0.14 m.



Rover	🟪	RTK: Fix&Hold
Base		-

	Position
Latitude	40.3305655°
Longitude	-111.7257878°
Altitude	1408.568 m
Accuracy: H, V	0.007, 0.010 m

	Velocity
ECEF X	0.05 m/s
ECEF Y	0.09 m/s
ECEF Z	-0.08 m/s
Accuracy	0.27 m/s

Date	09-17-
Time	15:05:3
Satellites Used	17
CNO: Mean	38.6 (

Differential Age	
AR Ratio	🟪
Base to Rover Dist.	
Base to Rover Hdg.	
Time not in Fix	2
slipCounter	

	Rover Base L
Latitude	40.3
Longitude	-111.7
Altitude	14(

USING THE CLTOOL

1. Connect the µINS Rover to a computer with the CLTool running.
2. Include the argument -msgPresetPPD in the CLTool command.
3. Observe the DID_GPS_RTK_NAV message, Status: 0x******** (Single) over the course of several minutes this will change to (Float) then (Fix).

RTK Fix Status

LED INDICATORS

The LEDs on the uINS will indicate RTK fix status.

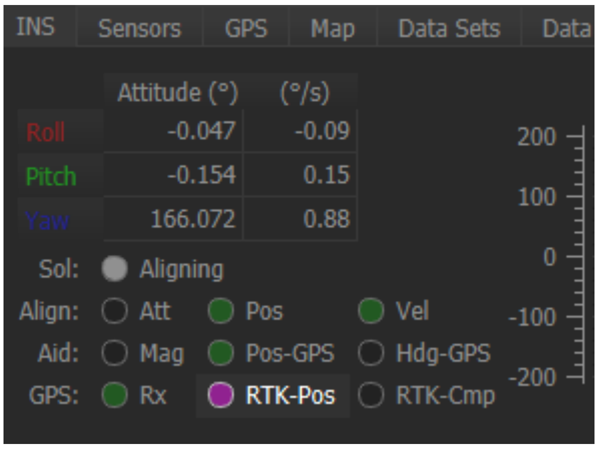
LED Behavior	Status	Description
🟢🔴 🟢🔴	3D Fix, RTK Float	Allows improved accuracy up to ~1m
🟢🟣 🟢🟣	RTK Fix	Allows increased accuracy up to ~3cm

RTK POSITIONING VALID FLAGS

The RTK precision positioning fix status can be identified using the valid bit in the INS and GPS status flags.

```
// INS status
INS_STATUS_NAV_FIX_STATUS(DID_INS_1.insStatus) == GPS_NAV_FIX_POSITIONING_RTK_FIX

// GPS status
DID_GPS1_POS.status & GPS_STATUS_FLAGS_RTK_POSITION_VALID
```

RTK precision positioning fix is indicated is indicated when the RTK-Pos radio button turns purple in the EvalTool INS tab.



	Attitude (°)	(°/s)
Roll	-0.047	-0.09
Pitch	-0.154	0.15
Yaw	166.072	0.88

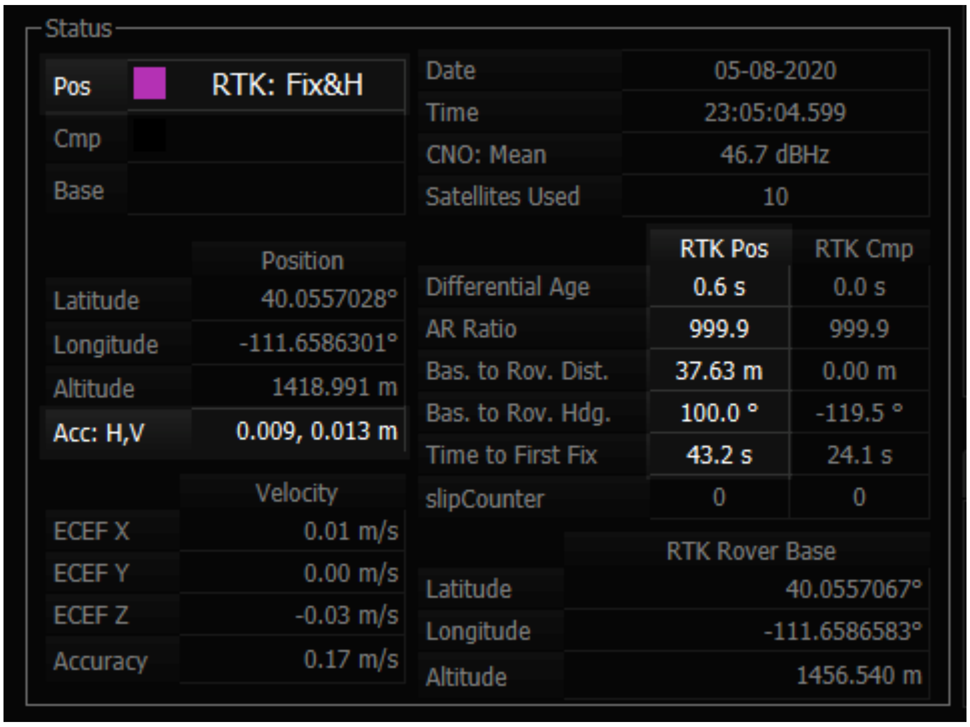
Sol: Aligning
Align: Att Pos Vel
Aid: Mag Pos-GPS Hdg-GPS
GPS: Rx RTK-Pos RTK-Cmp

PROGRESS AND ACCURACY

The ambiguity resolution ratio, `arRatio`, is a metric that indicates progress of the solution that ranges from 0 to 999. Typically values above 3 indicate RTK fix progress.

```
DID_GPS1_RTK_POS_REL.arRatio                        // Ambiguity resolution ratio
```

The DID_GPS1_RTK_POS_REL status can be monitored in the EvalTool GPS tab.



Status

Pos	🟪	RTK: Fix&H
Cmp	■	
Base		

	Position
Latitude	40.0557028°
Longitude	-111.6586301°
Altitude	1418.991 m
Acc: H,V	0.009, 0.013 m

	Velocity
ECEF X	0.01 m/s
ECEF Y	0.00 m/s
ECEF Z	-0.03 m/s
Accuracy	0.17 m/s

Date	05-08-2020
Time	23:05:04.599
CNO: Mean	46.7 dBHz
Satellites Used	10

	RTK Pos	RTK Cmp
Differential Age	0.6 s	0.0 s
AR Ratio	999.9	999.9
Bas. to Rov. Dist.	37.63 m	0.00 m
Bas. to Rov. Hdg.	100.0 °	-119.5 °
Time to First Fix	43.2 s	24.1 s
slipCounter	0	0

	RTK Rover Base
Latitude	40.0557067°
Longitude	-111.6586583°
Altitude	1456.540 m

RTK Base Messages

The uINS RTK solution accepts both RTCM3 and uBlox raw GNSS base correction messages. See the RTK Base or NTRIP pages for details on using base stations.

8.5.2 Rover Setup

System Configuration

A µINS must be configured as a Rover to receive RTK Base messages. This can be done through the EvalTool or the CLTool by enabling "Rover Mode".

EvalTool

1. Navigate to Settings > GPS > Rover > RTK.
2. Change the first drop-down menu to "Positioning (GPS1)", or one of the F9P options depending on the hardware setup.
3. Press Accept.
4. Verify the `RTKCfgBits` was automatically set correctly to any one of the rover modes listed in our binary communications protocol page.

CLTool

Use the `-flashConfig=rtkCfgBits=0x01` argument to configure the unit as rover where 0x01 can be any one of the rover modes listed in our binary communications protocol page.

Communications Setup

The uINS automatically parses data that arrives at any of the ports and recognizes base corrections data. Any communications method that sends the base corrections to one of the ports is suitable. Several common methods are described below.

EVB2 RADIO

EvalTool

The EVB-2 radio can be configured by pressing the "CONFIG" tactile switch until the light next to it is blue. This enables the radio and configures the radio settings. See the Configurations and EVB-2 Connections sections of the EVB-2 documentation.

1. Under "uINS Parameters" section verify the following:
    - Check the Baud Rate for the serial port of the radio (`ser0BaudRate` or `ser1BaudRate`). This should match the Baud Rate of the radio. The Digi Xbee Pro SX module on the EVB2 runs at **115200** baud.
2. Navigate to Data Sets > `DID_EVB_FLASH_CFG`
    - Change `cbPreset` - This should be set to `0x3` to enable the Digi Xbee Pro SX module.
    - Change `radioPID` - Radio Preamble ID. Should be the same number used as the Base radio. (`0x0 to 0x9`)
    - Change `radioNID` - Radio Network ID. Should be the same number used as the Base radio. (`0x0 to 0x7FFF`)
    - Change `radioPowerLevel` - Used to adjust the radio output power level. (0=20dbm, 1=27dbm, and 2=30dbm)
3. Reset the EVB2 and Rover radio setup is complete.

For more information on `DID_EVB_FLASH_CFG` see DID-descriptions.

NTRIP CLIENT

For the Rover to receive messages from an NTRIP Caster, it must be connected to an interface with internet access (e.g. computer).

EvalTool

Follow the proceeding steps in order to set up the Rover to receive messages through NTRIP:

1. Navigate to Settings > RTK > Rover Mode.
2. Change the first drop-down menu to "RTK - GPS1"

3. Under Correction Input:

    - Type = `NTRIP`
    - Address:Port = : Ex: `rtgpsout.unavco.org:2101`
    - Username/Password = Enter the Username and Password to the account used as the NTRIP Caster. Some Casters do not require this field.
    - Format = RTCM3 or UBLOX
    - Mount Point = Specify the mount point of the caster. Ex: `P016_RTCM3` 4. Press Apply.

CLTool

With the Rover µINS connected to the computer, use the -rover argument when running the CLTool executable:

- `-rover=TCP:` Set the type to "TCP".
- `PROTOCOL:` Set the protocol to "RTCM3" or "UBLOX". UBLOX requires more bandwidth and is not available from NTRIP casters.
- `URL:` The URL for the NTRIP Caster.
- `Port:` The port number will be provided by the NTRIP Caster.
- `MountPoint:` The mount point specifies which base station the corrections come from. This number will be provided by the NTRIP Caster.
- `Username:Password` The username and password for the account at the given URL (Not required by some public NTRIP casters).

Example:

```
cltool.exe -c COM10 -flashConfig=rtkCfgBits=0x01 -baud=57600 -rover=TCP:RTCM3:rtgpsout.unavco.org:2101:P016_RTCM3:username:password
```

TCP/IP

For the Rover to receive messages from a Base Station on a local network, it must be connected to an interface with network access (e.g. computer).

EvalTool

Follow these steps:

1. Navigate to Settings > GPS1
2. Under Correction Input:
    - Type = `TCP`
    - Address:Port = : e.g. `192.168.1.145:2001`
    - Change Format to "ublox" or "RTCM3". Ublox requires more bandwidth but will result in better performance.
3. Press Accept.

Hint

For **serial ports**, view available comport numbers in the Settings tab of the EvalTool.

CLTool

With the µINS Rover connected to the computer, enter the -rover argument when running the CLTool executable:

- `-rover=TCP:` Set the type to "TCP".
- `RTCM3` Set the message type to "RTCM3" or "UBLOX". UBLOX requires more bandwidth and may be unavailable from some NTRIP Casters.
- `IP_Address` The IP Address of the Base Station to receive messages from.
- `Port` You may choose any number here. This should match the port number used for the Base Station.

Example: `cltool.exe -c COM10 -flashConfig=rtkCfgBits=0x01 -baud=57600 -rover=RTCM3:100.100.1.100:7777`

EVB2 WIFI

Using the EVB2 WiFi module to connect to the TCP/IP Base. EVB2 can save up to 3 Networks information. (Wifi[0], Wifi[1], Wifi[2]) Follow these steps using the EvalTool:

1. Under "uINS Parameters" section verify the following:
    - Verify the `RTKCfgBits` was automatically set to 0x00000001
2. Navigate to Data Sets > `DID_EVB_FLASH_CFG`
    - Change `cbPreset` - This should be set to `0x4` to enable the WiFi module.
    - Change `wifi[0].ssid` - WiFi [0] Service Set Identifier or network name.
    - Change `wifi[0].psk` - WiFi [0] Pre-Shared Key authentication or network password.
    - Change `server[0].ipAddr` - server [0] IP address.
    - Change `server[0].port` - server [0] port.
3. Reset the EVB2 and Rover WiFi setup is complete.

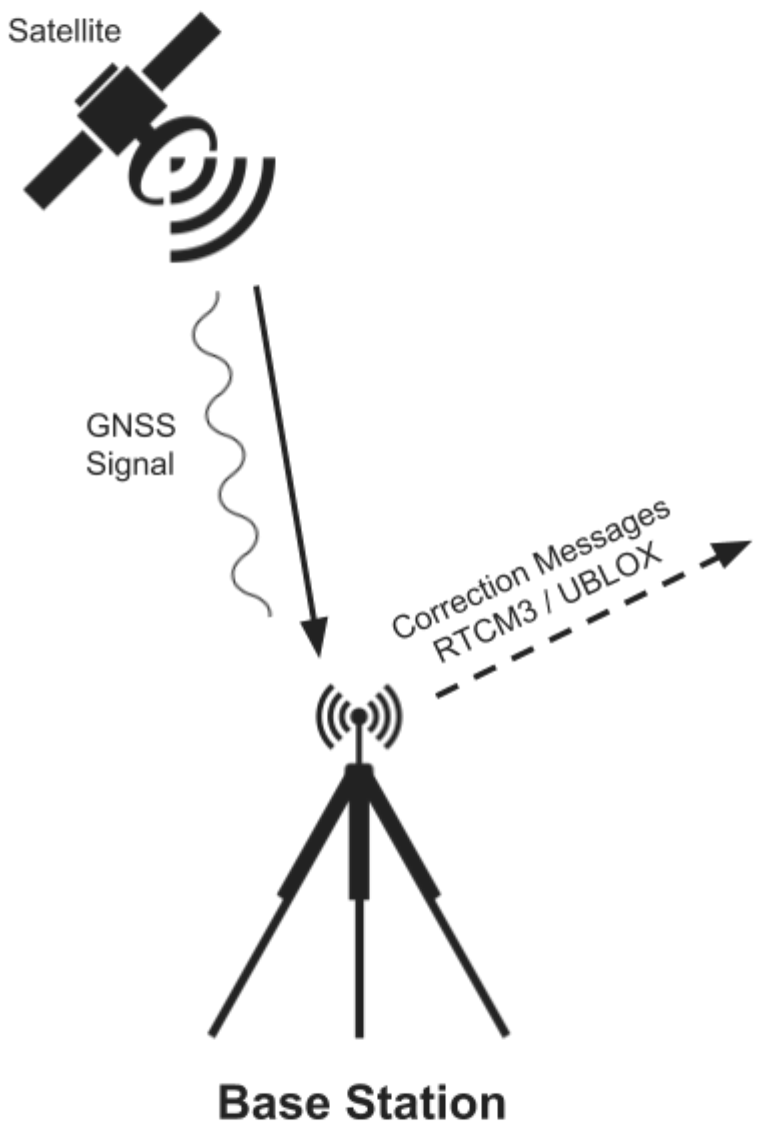
8.5.3 Base Setup

RTK Base Configuration

Note

If using an **NTRIP** service or **3rd Party Base Station** instead of your own base station, please skip this page and see the **NTRIP** page or reference the setup instructions for the 3rd Party Base Station. **NTRIP services** do not require additional setup.

An essential part of an RTK system is the Base Station which supplies correction messages from a known, surveyed location to the RTK Rover. The µINS Rover supports receiving RTCM3 and UBLOX correction messages.



Surveying In Base Position

Important

M M M M M M M M M M
in for rover absolute position accuracy.

The base survey cannot happen at the same time as base correction output messages are enabled. If a survey is started the base correction output will automatically be disabled.

The base position is stored in `DID_FLASH_CONFIG.reflla` and transmitted to the rover during RTK operation. The following steps outline how to survey in the base position.

1. **Mount base station in fixed location** - The location should not change during or following a survey.

2. **Set survey-in parameters** - This step can either be done using the EvalTool or programmatically using the data set (`DID_SURVEY_IN`).

EvalTool

1. Navigate to Settings > RTK > Base Mode.

2. In the "Survey In" section select one of the States:

    - **Manual**: Direct entry of base position.
    - **Average GPS 3D** - Requires standard GPS 3D lock (non-RTK mode) for survey.*
    - **Average RTK Float** - Requires RTK float state for survey.*
    - **Average RTK Fix** - Requires RTK fix for survey.

    *The average methods will not run if the minimum requirements are not met. The system will wait until the requirements are met and then begin the survey.

3. Use the slider to select the Survey In runtime. Generally the longer the survey runs the more accurate the results will be.

4. Press the Start button.

Note

The current estimate of the survey is listed in the Position area above the Survey In section. If the survey completes successfully the results stored in flash memory (`DID_FLASH_CONFIG.reflla`) which will only change if the survey is re-run.

Using DID_SURVEY_IN

1. The location of the base can be manually entered using (`DID_FLASH_CONFIG.RefLLA`) if location is known.

2. Set `DID_SURVEY_IN.maxDurationSec` - Maximum time in milliseconds the survey will run. This is ignored if it is set to 0.

3. Set `DID_SURVEY_IN.minAccuracy` - Minimum horizontal accuracy in meters for survey to complete before maxDuration. This is ignored if it is set to 0.

4. Set (`DID_SURVEY_IN.state`) to begin the survey according to the desired survey State:

    - **2 = Average GPS 3D** - Requires standard GPS 3D lock (non-RTK mode) or better for survey.*
    - **3 = Average RTK Float** - Requires RTK float fix or better for survey.*
    - **4 = Average RTK Fix** - Requires RTK fix for survey.

    *The average methods will not run if the minimum requirements are not met. The system will wait until the requirements are met and then begin the survey.

Communications Setup

RADIO

The Base uINS must be configured to stream base corrections to the radio so it can be broadcast to the rover.

EvalTool

1. Open the COM port for the µINS under Settings > Serial Ports.
2. Navigate to Settings > RTK > Base Mode.
3. Under "Correction Output", find the fields for serial ports 0, 1, or USB. Select the serial port from which the corrections will be transmitted. This port must also be connected to the radio. Choose one of the options listed below. Leave the unused serial port off.
    - "GPS1 - RTCM3": *Output standard RTCM3 messages.*
    - "GPS1 - uBlox": *Output uBlox messages. This will provide more accuracy but requires significantly more bandwidth.*
4. Change the "Data Rate(ms)" field. This determines how many milliseconds pass between message outputs (e.g. Data Rate(ms) = 1,000 means one message/second). It is usually best to match the startupGPSDtMs value found in DID_FLASH_CONFIG.
5. In the "Position" section, a the Base Station position is required so that it can transmit accurate corrections. Please refer to Surveying In Base Position if the base station location is unknown.
6. Click Apply, and reset the µINS. The unit will now start up in Base Station mode. Verify the base station is working by looking in the section labeled "Status". It will display the serial port of the radio and the message type. e.g. "SER1:UBX"
7. Navigate to Data Sets > `DID_EVB_FLASH_CFG`
    - Change `cbPreset` - This should be set to `0x3` to enable the Digi Xbee Pro SX module.
    - Change `radioPID` - Radio Preamble ID. Should be the same number used as the Rover radio. (`0x0 to 0x9`)
    - Change `radioNID` - Radio Network ID. Should be the same number used as the Rover radio. (`0x0 to 0x7FFF`)
    - Change `radioPowerLevel` - Used to adjust the radio output power level. (0 = 20dbm, 1 = 27dbm, and 2 = 30dbm)
8. Reset the EVB2 and Base radio setup is complete.

For more information on `DID_EVB_FLASH_CFG` see DID-descriptions.

CLTool

The RTK config bit must be set manually when using the CLTool. Use the following command line arguments when executing the CLTool from a prompt/terminal.

- `-c #` Open the COM port of the µINS. Windows users will use the name of the COM port, e.g. COM7. Linux users must enter the path to the correct COM port, e.g. /dev/ttyUSB0.
- `-baud=#` Set the baud rate for communications output (Replace # with baud rate number). This number will vary depending on setup. For lower quality radios it maybe necessary to use a lower baud rate (ex: 57600).
- `-flashConfig=rtkCfgBits=0x00` Configure the unit to cast Base corrections. For more configuration options see eRTKConfigBits

Example:

```
cltool.exe -c COM29 -baud=57600 -flashConfig=rtkCfgBits=0x80
```

Warning

If the Base Station is not communicating properly, it maybe necessary to verify that the baud rate is set to match that of the radios used. This rate varies depending on radio type.

TCP/IP SETUP

CLTool

It is required to manually set the RTK config bits in the CLTool. Passed these to the CLTool when run from the command prompt/terminal.

- `-c #` Open the COM port of the µINS. Windows users will use the name of the COM port, e.g. COM7. Linux users must enter the path to the correct COM port, e.g. /dev/ttyUSB0.
- `-baud=#` Set the baud rate for communications output (Replace # with baud rate number).
- `` `-flashConfig=rtkCfgBits=0x00 `` Configure the unit to cast Base corrections. For more configuration options see eRTKConfigBits
- `-base=:#` Create the port over which corrections will be transmitted. Choose any unused port number.

Example:

```
cltool.exe -c COM29 -baud=921600 -flashConfig=rtkCfgBits=0x10 -base=:7777
```

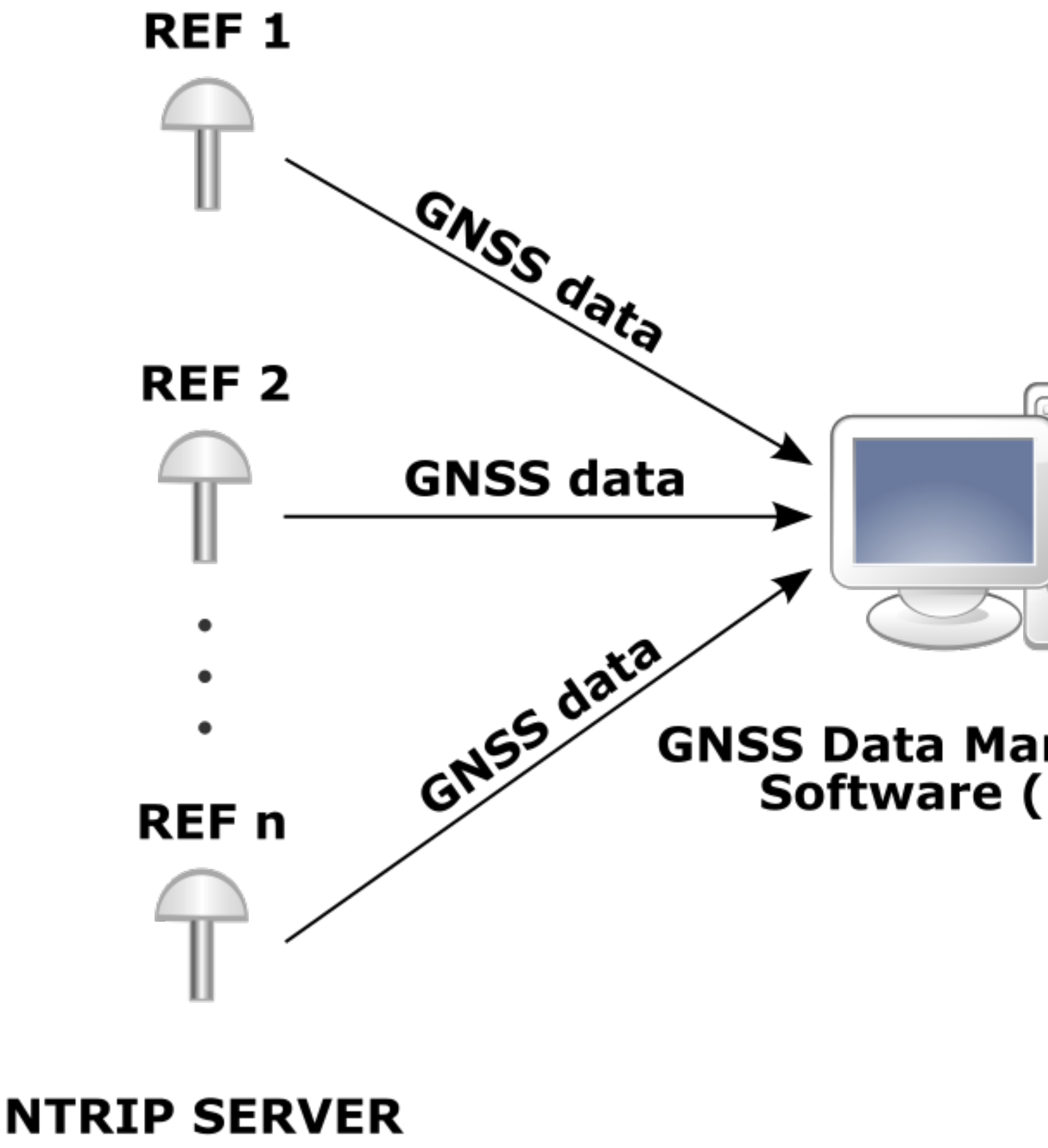
Important

If the console displays the error **"Failed to open port at COMx"**, reset the device immediately after attempting to change the baud rate in the CLTool.

8.5.4 NTRIP

Networked Transport of RTCM via internet protocol, or NTRIP, is an open standard protocol for streaming differential data over the internet in accordance with specifications published by RTCM. There are three major parts to the NTRIP system: The NTRIP client, the NTRIP server, and the NTRIP caster:

1. The NTRIP server is a PC or on-board computer running NTRIP server software communicating directly with a GNSS reference station.
2. The NTRIP caster is an HTTP server which receives streaming RTCM data from one or more NTRIP servers and in turn streams the RTCM data to one or more NTRIP clients via the internet.
3. The NTRIP client receives streaming RTCM data from the NTRIP caster to apply as real-time corrections to a GNSS receiver.



The EvalTool/CLTool software applications provide NTRIP client functionality to be used with the uINS RTK rover. Typically an EvalTool NTRIP client connects over the internet to an NTRIP service provider. The EvalTool/CLTool NTRIP client then provides the RTCM 3.3 corrections to the uINS and ZED-F9P rover connected over USB or serial. Virtual reference service (VRS) is also supported by the EvalTool/CLTool NTRIP client.

Important

If using a **virtual reference service** (**VRS**), the rover must output the **NMEA GGA** message to return to the NTRIP caster. Without this, the NTRIP caster will not provide correction information.

NTRIP RTCM3 Messages

The NTRIP server must provide the necessary subset of RTCM3 messages supported by the uINS-RTK. The following is an example of compatible RTCM3 base output messages provided from a Trimble NTRIP RTK base station.

Message Type	Period (sec)	Description
RTCM 1005	5	Stationary RTK reference station ARP
RTCM 1007	5	Antenna Descriptor
RTCM 1030	3	GPS Network RTK Residual
RTCM 1031	3	GLONASS Network RTK Residual
RTCM 1032	1	Physical Reference Station Position
RTCM 1033	5	Receiver and Antenna Descriptors
RTCM 1075	1	GPS MSM5
RTCM 1085	1	GLONASS MSM5
RTCM 1095	1	Galileo MSM5
RTCM 1230	5	GLONASS code-phase biases
RTCM 4094	5	Assigned to : Trimble Navigation Ltd.

8.6 Dual GNSS RTK Compassing

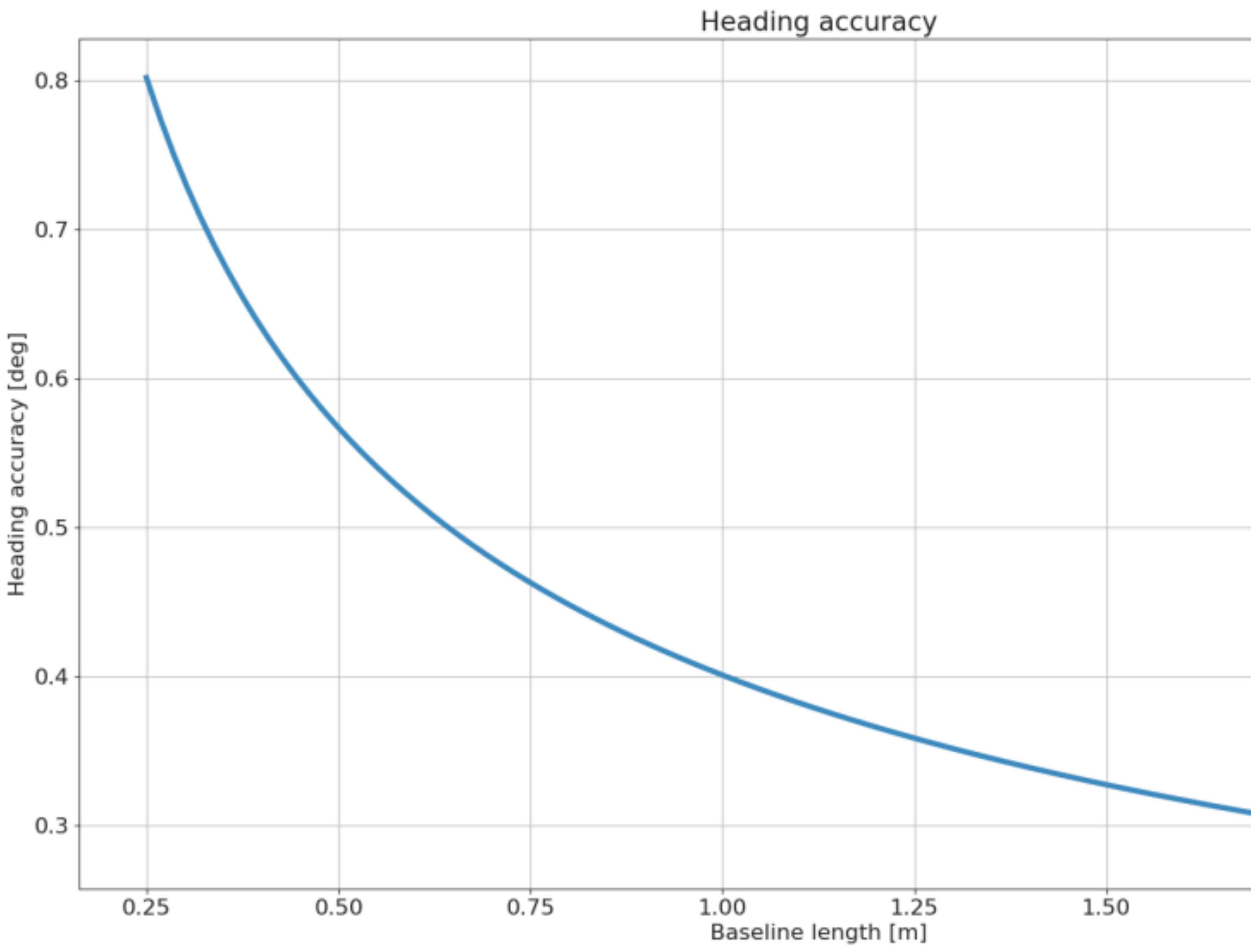
8.6.1 Overview

RTK Compassing (Dual GNSS) is a system that determines heading by use of two GNSS receivers and antennas. It replaces the need for magnetometers which can be problematic in the presence of ferrous materials (e.g. steel) and EMI generating circuits (e.g. electric motors and drivers).

See the multi-band dual GNSS section for details on using our multi-frequency dual ZED-F9 GNSS system.

8.6.2 Heading Accuracy

The generalized heading accuracy for both the single-band (L1) and the dual GNSS multi-band systems under ideal conditions is shown in the following plot.



Recommenced Minimum Baseline

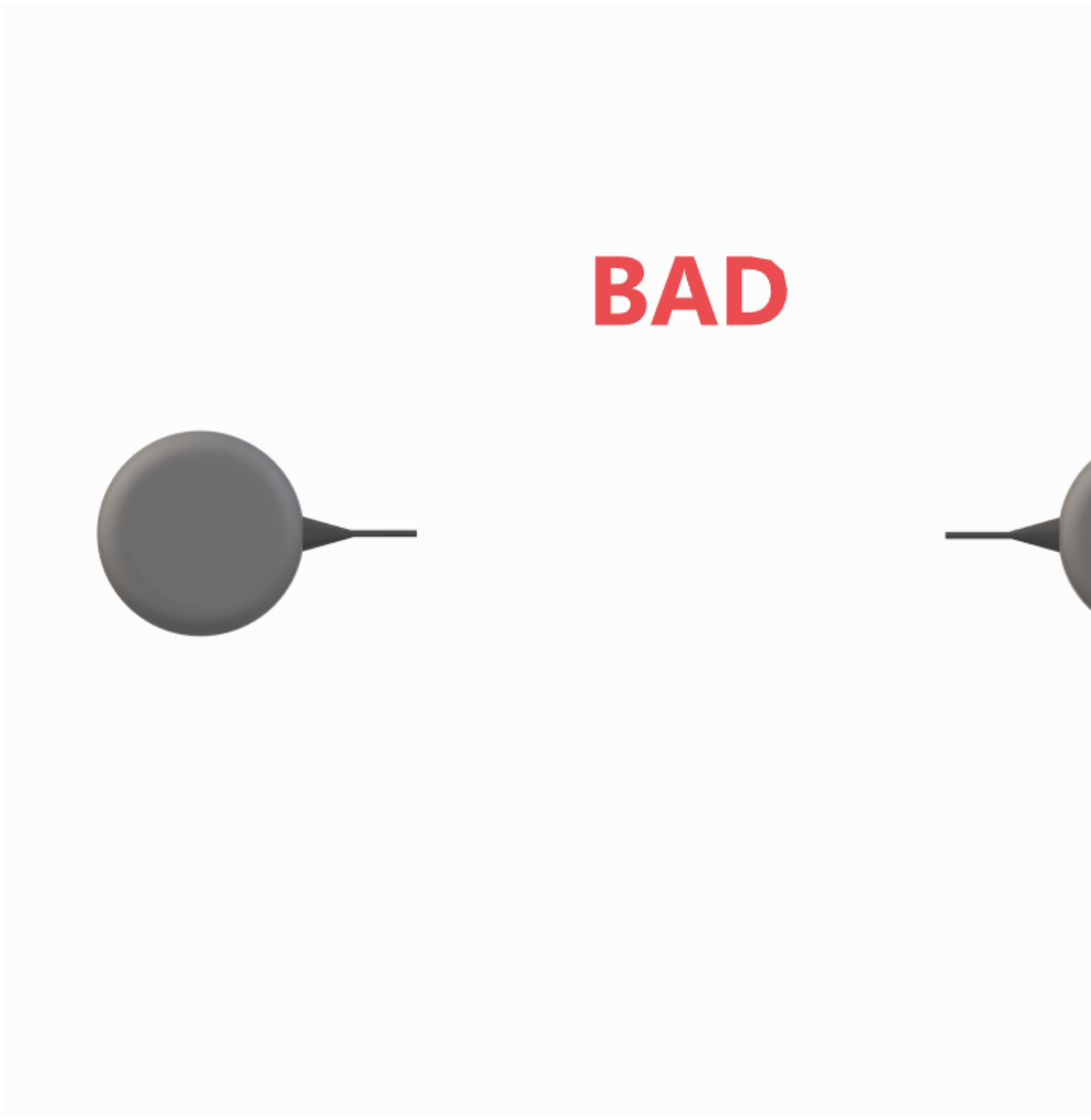
The recommended minimum baseline (distance between dual GNSS antennas) is ***0.3 meters*** for single-band (L1) GNSS compassing and ***0.25 meters*** for multi-band ZED-F9 GNSS compassing. The solution can operate at shorter baseline distances but is less robust and more susceptible to getting caught in a local minimum which may not converge to the correct heading.

8.6.3 Antenna Orientation

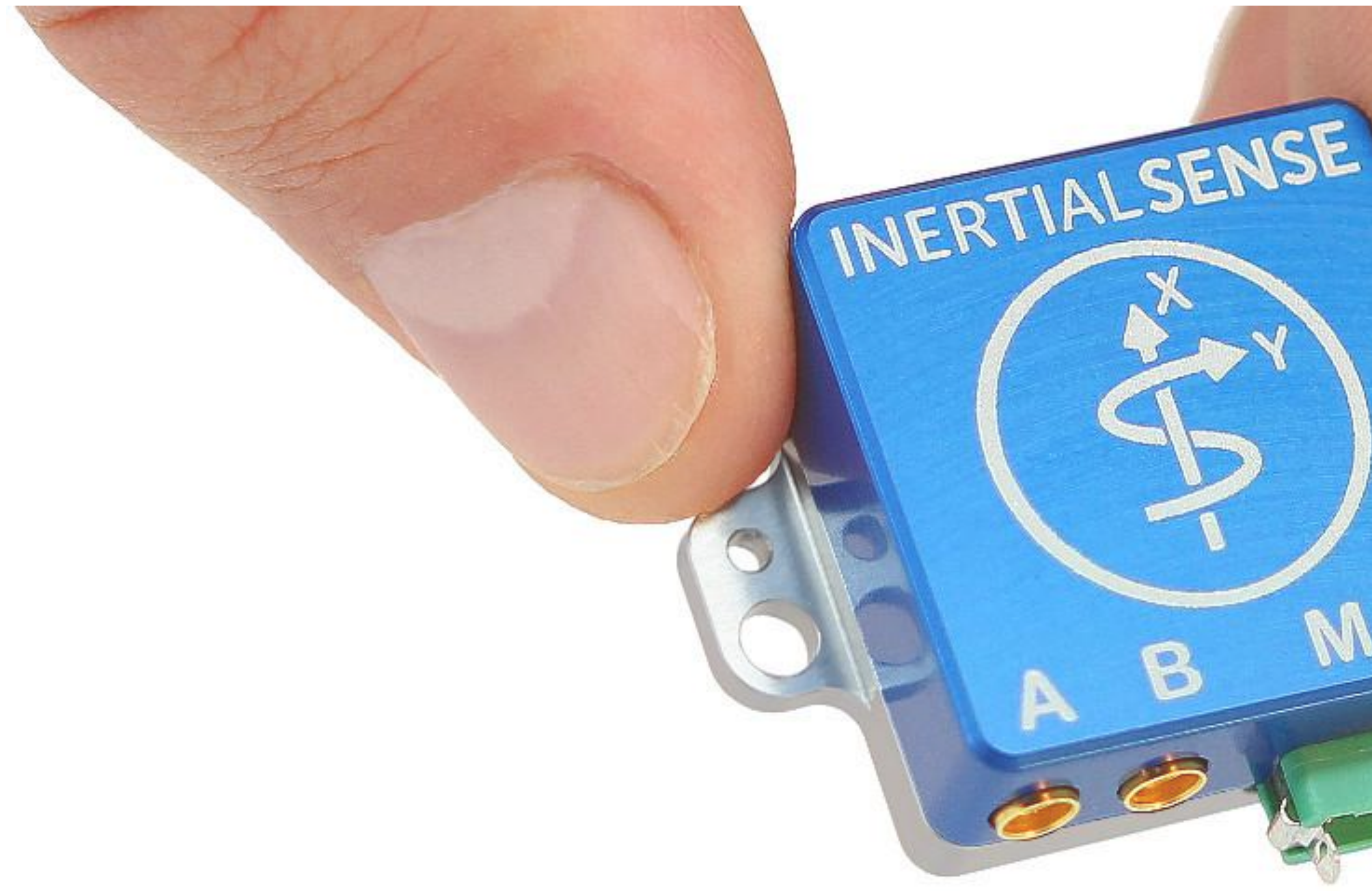
Important

It is recommended that both GNSS antennas be identical and have the same physical orientation relative to each other (i.e. the antenna cable should exit in the same direction on both antennas). This will ensure best RF phase center alignment and heading accuracy. The actual RF phase center is often offset from the physical center of the antenna case.

Mismatch



8.6.4 Rugged GNSS Antenna Ports



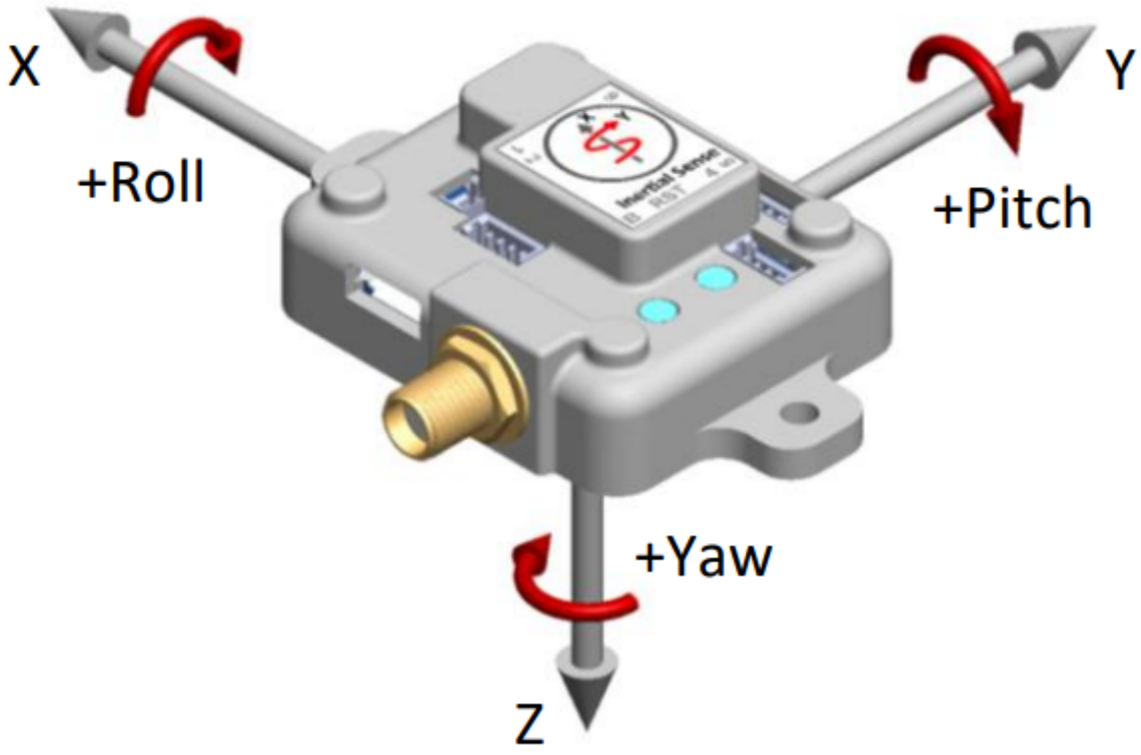
On the Rugged uINS, the MMCX port ***A*** is for ***GPS1*** and MMXC port ***B*** is ***GPS2***. These port labels are changed to ***1*** and ***2*** on newer Rugged units.

8.6.5 Dual Antenna Locations

The location for both GPS antennae must be correctly specified by the user in the DID_FLASH_CONFIG variables within 1 cm accuracy:

```
DID_FLASH_CONFIG.gps1AntOffset[X,Y,Z]
DID_FLASH_CONFIG.gps2AntOffset[X,Y,Z]
```

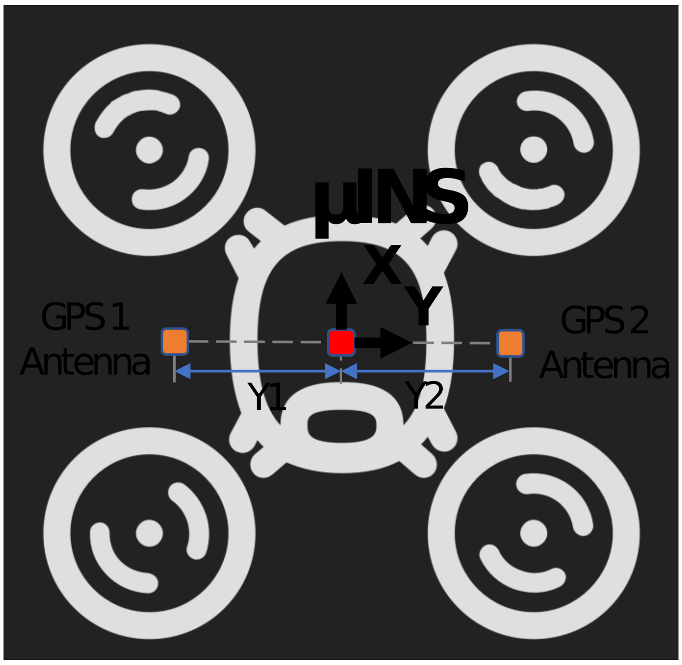
These values describe the distance of each GPS antenna from the uINS Sensor Frame origin in the direction of the Sensor Frame axes. The Sensor Frame is identified by the X, Y, Z axes labeled on the hardware. The Z-axis is positive in the downward direction.



Example Antennae Configurations

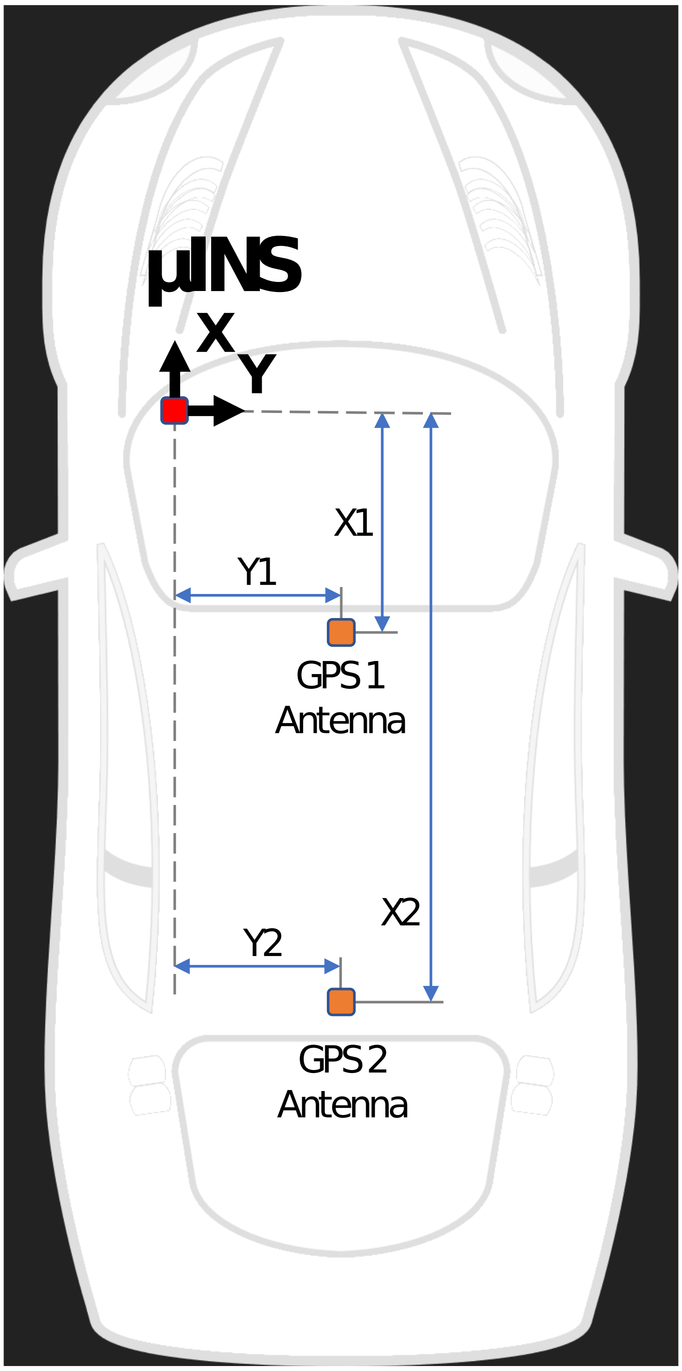
The following are examples that illustrate what the GPS antenna offsets should be for two different antenna configurations.

DRONE



```
DID_FLASH_CONFIG.gps1AntOffsetX =  0.0
DID_FLASH_CONFIG.gps1AntOffsetY = -0.3  (negative direction of Y axis)
DID_FLASH_CONFIG.gps1AntOffsetZ =  0.0

DID_FLASH_CONFIG.gps2AntOffsetX =  0.0
DID_FLASH_CONFIG.gps2AntOffsetY =  0.3
DID_FLASH_CONFIG.gps2AntOffsetZ =  0.0
```

AUTOMOBILE



```
DID_FLASH_CONFIG.gps1AntOffsetX = -0.5  (negative direction of X axis)
DID_FLASH_CONFIG.gps1AntOffsetY =  0.5
DID_FLASH_CONFIG.gps1AntOffsetZ = -0.5  (negative direction of Z axis, above uINS)

DID_FLASH_CONFIG.gps2AntOffsetX = -1.5  (negative direction of X axis)
DID_FLASH_CONFIG.gps2AntOffsetY =  0.5
DID_FLASH_CONFIG.gps2AntOffsetZ = -0.5  (negative direction of Z axis, above uINS)
```

GPS Antenna Ports

The following table explains how ports A and B on the Rugged uINS map to GPS antennas 1 and 2.

Ports	Rugged uINS	uINS Module and EVB-2
GPS 1 antenna port	A	1
GPS 2 antenna port	B	2

8.6.6 **Setup**

Step 1 - Specify Offsets for Both Antennae

Refer to the Dual Antenna Locations section for a description of the GPS antenna offset.

```
DID_FLASH_CONFIG.gps1AntOffsetX = ?
DID_FLASH_CONFIG.gps1AntOffsetY = ?
DID_FLASH_CONFIG.gps1AntOffsetZ = ?

DID_FLASH_CONFIG.gps2AntOffsetX = ?
DID_FLASH_CONFIG.gps2AntOffsetY = ?
DID_FLASH_CONFIG.gps2AntOffsetZ = ?
```

Using EvalTool - select `Data Sets -> DID_FLASH_CONFIG` and set `gps1AntOffset[X,Y,Z]` and `gps2AntOffset[X,Y,Z]` with the GPS antenna offsets.

Using CLTool - run the CLTool using the following options replacing the `[OFFSET]` with the GPS antenna offsets.

```
-flashconfig=gps1AntOffsetX=[OFFSET]
-flashconfig=gps1AntOffsetY=[OFFSET]
-flashconfig=gps1AntOffsetZ=[OFFSET]
-flashconfig=gps2AntOffsetX=[OFFSET]
-flashconfig=gps2AntOffsetY=[OFFSET]
-flashconfig=gps2AntOffsetZ=[OFFSET]
```

Step 2 - Enable GPS Dual Antenna

Set the `RTK_CFG_BITS_COMPASSING (0x00000008)` bit of RTKCfgBits.

```
DID_FLASH_CONFIG.RTKCfgBits |= RTK_CFG_BITS_COMPASSING      // |= 0x00000008
```

Using EvalTool - go to `Settings -> RTK -> Rover Mode`, set the dropdown menu to `GPS Compassing`, and press the `Apply` button.

Using CLTool - run the CLTool using the `-flashconfig=RTKCfgBits=0x8` option to enable GPS Dual Antenna.

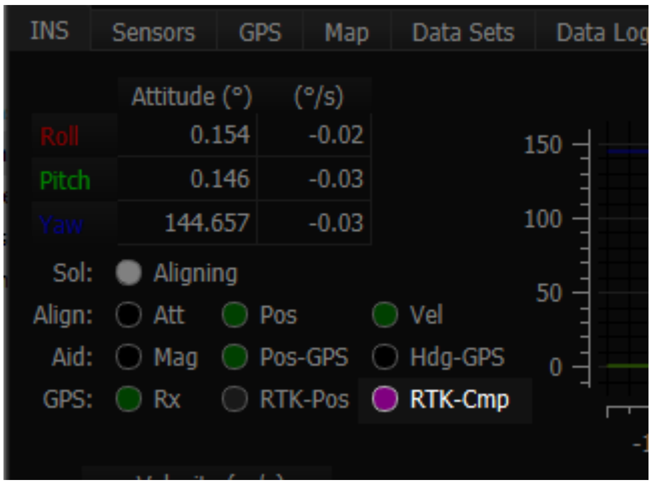
8.6.7 RTK Compassing Fix Status

INS and GPS Status Flags

The RTK compassing fix status can be identified using the valid bit in the INS and GPS status flags.

```
DID_INS_1.insStatus & INS_STATUS_RTK_COMPASSING_VALID          // INS status
DID_GPS1_POS.status & GPS_STATUS_FLAGS_RTK_COMPASSING_VALID    // GPS status
```

RTK compassing fix is indicated when the RTK-Cmp radio button turns purple in the EvalTool INS tab.



	Attitude (°)	(°/s)
Roll	0.154	-0.02
Pitch	0.146	-0.03
Yaw	144.657	-0.03

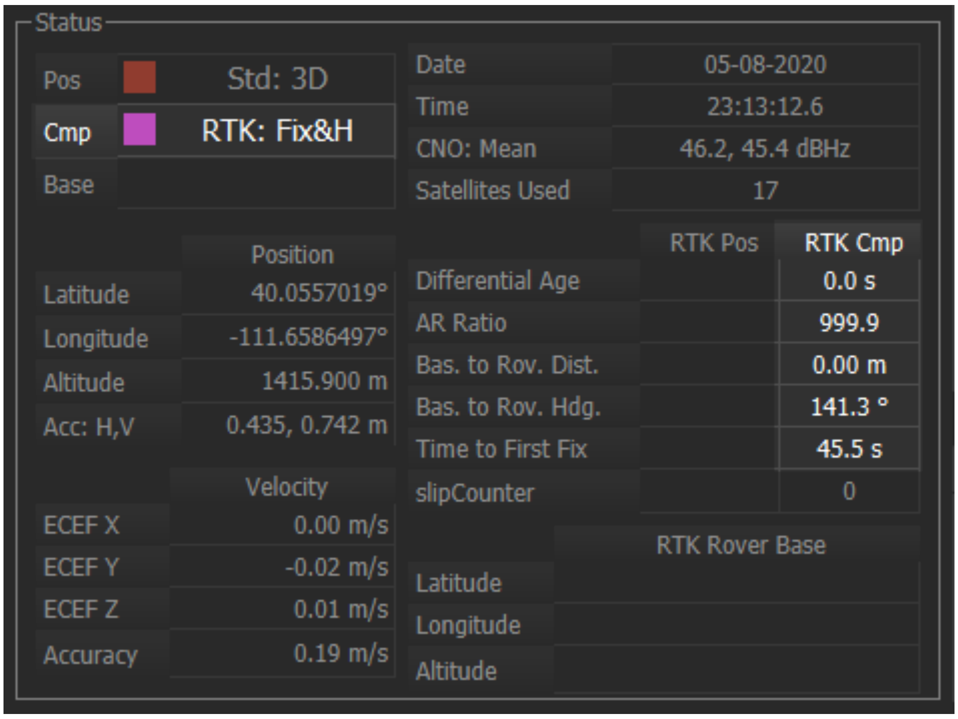
Sol: ● Aligning
Align: ○ Att ● Pos ● Vel
Aid: ○ Mag ● Pos-GPS ○ Hdg-GPS
GPS: ● Rx ○ RTK-Pos ● RTK-Cmp

Progress and Accuracy

The ambiguity resolution ratio, `arRatio`, is a metric that indicates progress of the solution that ranges from 0 to 999. Typically values above 3 indicate RTK fix progress. The base to rover heading accuracy indicates how much error is in the base to rover heading (RTK compassing heading).

```
DID_GPS1_RTK_CMP_REL.arRatio                    // Ambiguity resolution ratio
DID_GPS1_RTK_CMP_REL.baseToRoverHeadingAcc      // (rad) RTK compassing accuracy
```

The DID_GPS1_RTK_CMP_REL status can be monitored in the EvalTool GPS tab.



Status

Pos	🟥	Std: 3D
Cmp	🟪	RTK: Fix&H
Base		

Date	05-08-2020
Time	23:13:12.6
CNO: Mean	46.2, 45.4 dBHz
Satellites Used	17

	Position
Latitude	40.0557019°
Longitude	-111.6586497°
Altitude	1415.900 m
Acc: H,V	0.435, 0.742 m

	RTK Pos	RTK Cmp
Differential Age		0.0 s
AR Ratio		999.9
Bas. to Rov. Dist.		0.00 m
Bas. to Rov. Hdg.		141.3 °
Time to First Fix		45.5 s
slipCounter		0

	Velocity
ECEF X	0.00 m/s
ECEF Y	-0.02 m/s
ECEF Z	0.01 m/s
Accuracy	0.19 m/s

	RTK Rover Base
Latitude	
Longitude	
Altitude	

8.6.8 Stationary Application

For RTK compassing stationary application, enabling the STATIONARY INS dynamic model (DID_FLASH_CONFIG.insDynModel = 2) is recommended to reduce heading noise and drift. This will reduce heading error during RTK compassing fix or loss of fix. See INS-GNSS Dynamic Model and Zero Motion Command for details.

9. Dead Reckoning

9.1 Ground Vehicle Dead Reckoning

9.1.1 Overview

The uINS inertial navigation integrates IMU data to dead reckon (estimate position and velocity) when GPS position fix is not available. The amount of position error during dead reckoning can vary based on several factors including system runtime, motion experienced, and sensor bias stability.

Knowledge about the vehicle's kinematic constraints is applied to reduce drift and improve position estimation.

9.1.2 Installation

Important

It is critical to ensure the uINS remains fixed relative to the vehicle. Any shift or change in the uINS location relative to the vehicle will result in degraded or inaccurate dead reckoning solution.

Important

Heavy vibrations can degrade the uINS measurements and dead reckoning solution.

1. Mount the uINS and GNSS antenna at fixed locations on the vehicle.
2. Set the GPS antenna offsets relative to the uINS origin in meters. EvalTool > Data Sets > DID_FLASH_CONFIG > gps1AntOffsetX/Y/Z.

Enabling

Dead reckoning is enabled by setting the `DID_FLASH_CONFIG.insDynModel` to 4 for ground vehicles. This is done automatically during Learning Mode and stored to flash memory.

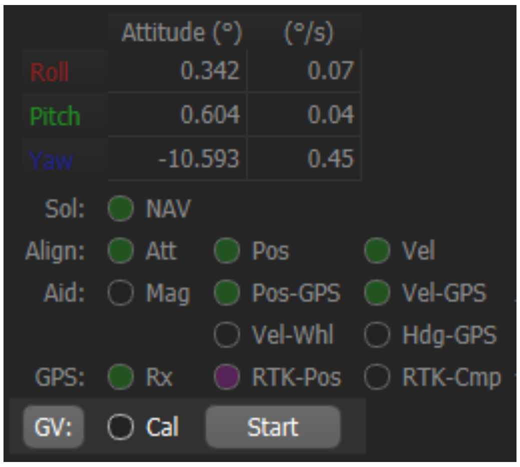
Learning Mode

Learning mode is be used following installation or any change in the uINS position relative to the vehicle. Learning is used to estimate the vehicle kinematic calibration which is used during normal operation.

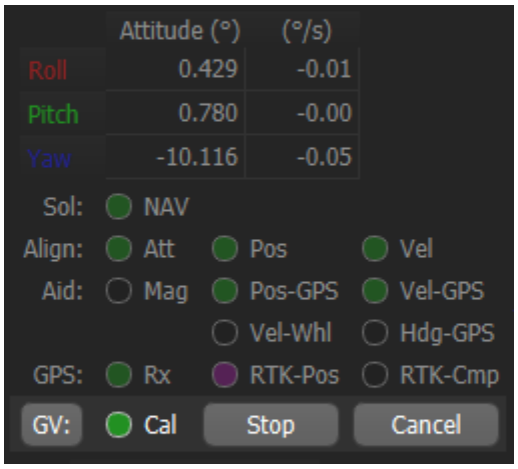
LEARNING MODE INSTRUCTIONS

1. `Start` learning mode.
2. Drive with sufficient motion for learning. This is identified with the EvalTool `GV: Cal` indicator in the INS tab turns GREEN (`DID_GROUND_VEHICLE.status & GV_STATUS_LEARNING_CONVERGED` is not zero). Either of the following patterns is typically adequate.
    - At least 200 meters straight, 5 left turns (+90 degrees) and 5 right turns
    - Three figure eight patterns.
3. `Stop` learning and save kinematic calibration to flash memory.

Using the EvalTool



	Attitude (°)	(°/s)
Roll	0.342	0.07
Pitch	0.604	0.04
Yaw	-10.593	0.45



	Attitude (°)	(°/s)
Roll	0.429	-0.01
Pitch	0.780	-0.00
Yaw	-10.116	-0.05

From the EvalTool INS tab:

1. Press the `GV:` button to reveal the ground vehicle options.
2. Press the `Start` button to clear and start learning.
3. Press the `Stop` button to stop learning and save kinematic calibration to flash memory.

Using the DID_GROUND_VEHICLE Message

1. Enable learning mode by setting the `DID_GROUND_VEHICLE.mode` to any of the following commands. The `DID_GROUND_VEHICLE.mode` value will toggle to 1 indicating the system is in learning mode and 0 to indicate learning mode is off.

   2 "Start" - Start with user supplied values in the `DID_GROUND_VEHICLE.transform` and enable learning mode.

   3 "Resume" - Start with the existing calibration and enable learning mode.

   4 "Clear & Start" - Set transform to zero and start with aggressive learning mode. This is the same as the "Start" button in the EvalTool INS tab.

   5 "Stop & Save" - End learning mode and save kinematic calibration to flash memory.

2. Disable learning and save kinematic calibration to flash memory by setting `DID_GROUND_VEHICLE.mode` to 5.

9.1.3 Examples

Dead reckoning examples can be found here.

9.2 uINS Dead Reckoning Examples

Dead Reckoning is the process of calculating the current position of a moving object by using a previously determined position, or fix, and then incorporating estimations of speed, heading direction, and course over elapsed time. Knowledge about the vehicle's kinematic constraints (i.e. wheels on the ground) is applied to reduce drift and improve position estimation.

Inertial Sense has added dead reckoning capability to uINS to estimate position for extended periods of time during GNSS outages. In this report RTK-GNSS is used.

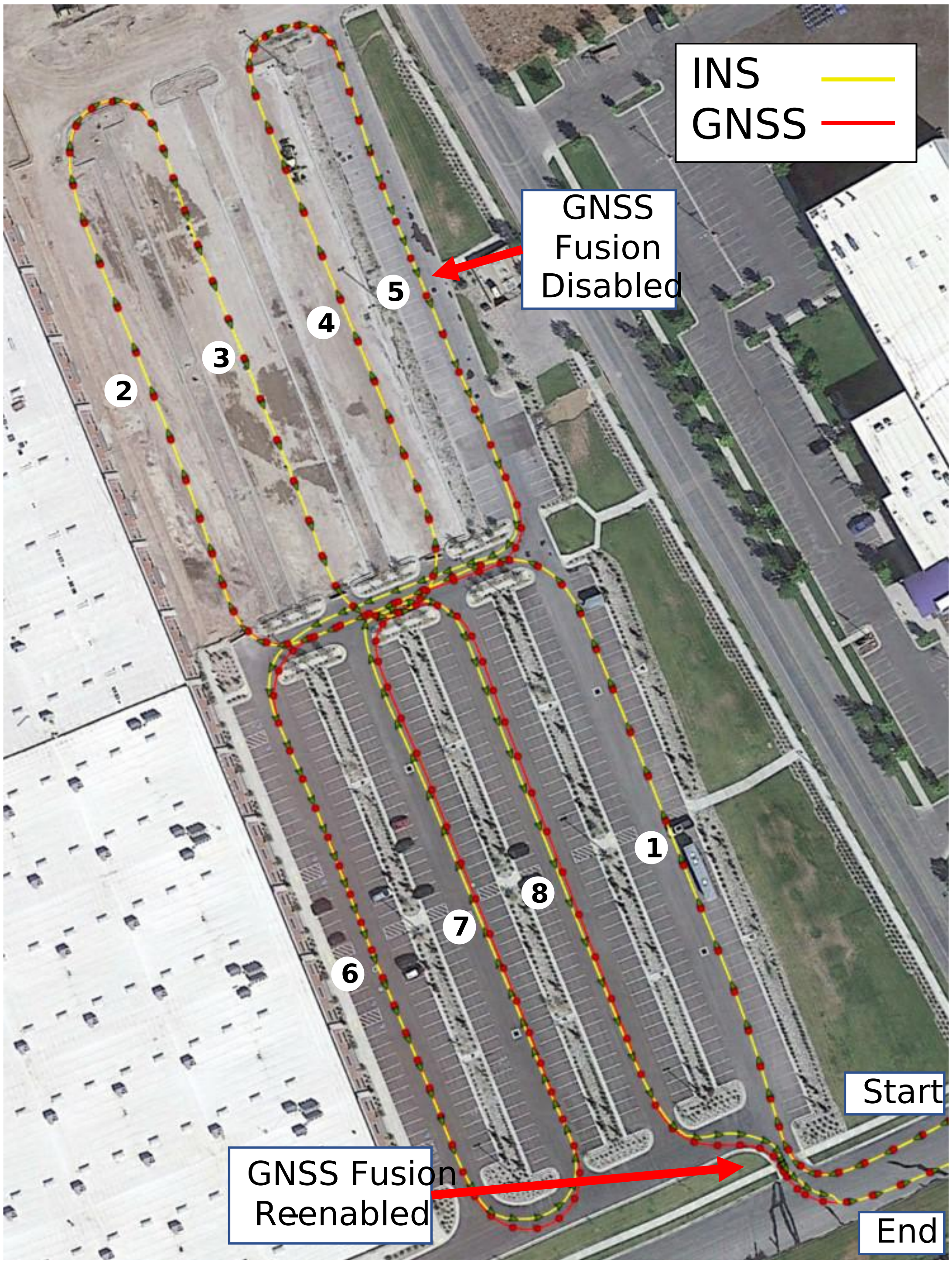
The following are examples dead reckoning of a car test vehicle. No wheel sensors were used in these examples. The dead reckoning position is shown in the yellow "INS" line and GNSS position in the red "GNSS" line.

9.2.1 Parking Lot Simulated GNSS Outage

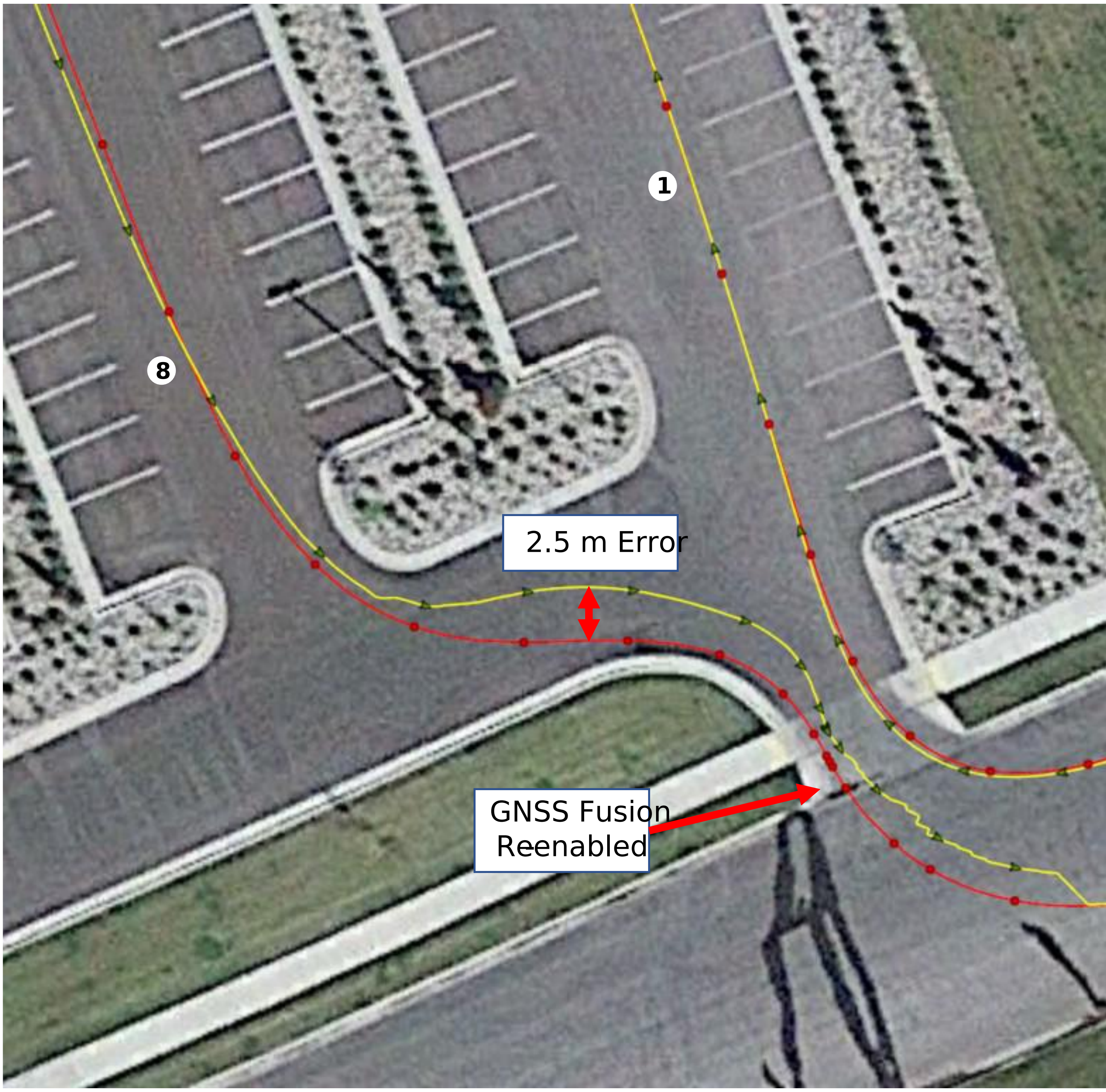
In this example GNSS outage was simulated by disabling GNSS fusion into the INS Kalman filter (EKF). This was done by setting the `Disable Fusion - GPS1` option found in the General settings of the EvalTool app. By disabling GPS fusion and keeping fix, we can use the GNSS position as truth and compare it to the dead reckoning solution.

Dead reckoning duration: **30 seconds**, **605 meters**

Max position error: **2.5 meters**, **0.4% drift**



In the drive the car starts and ends the drive at the bottom right corner of the image. The numbered path segments show the order of travel. GPS fusion was disabled in the middle of path segment 5.



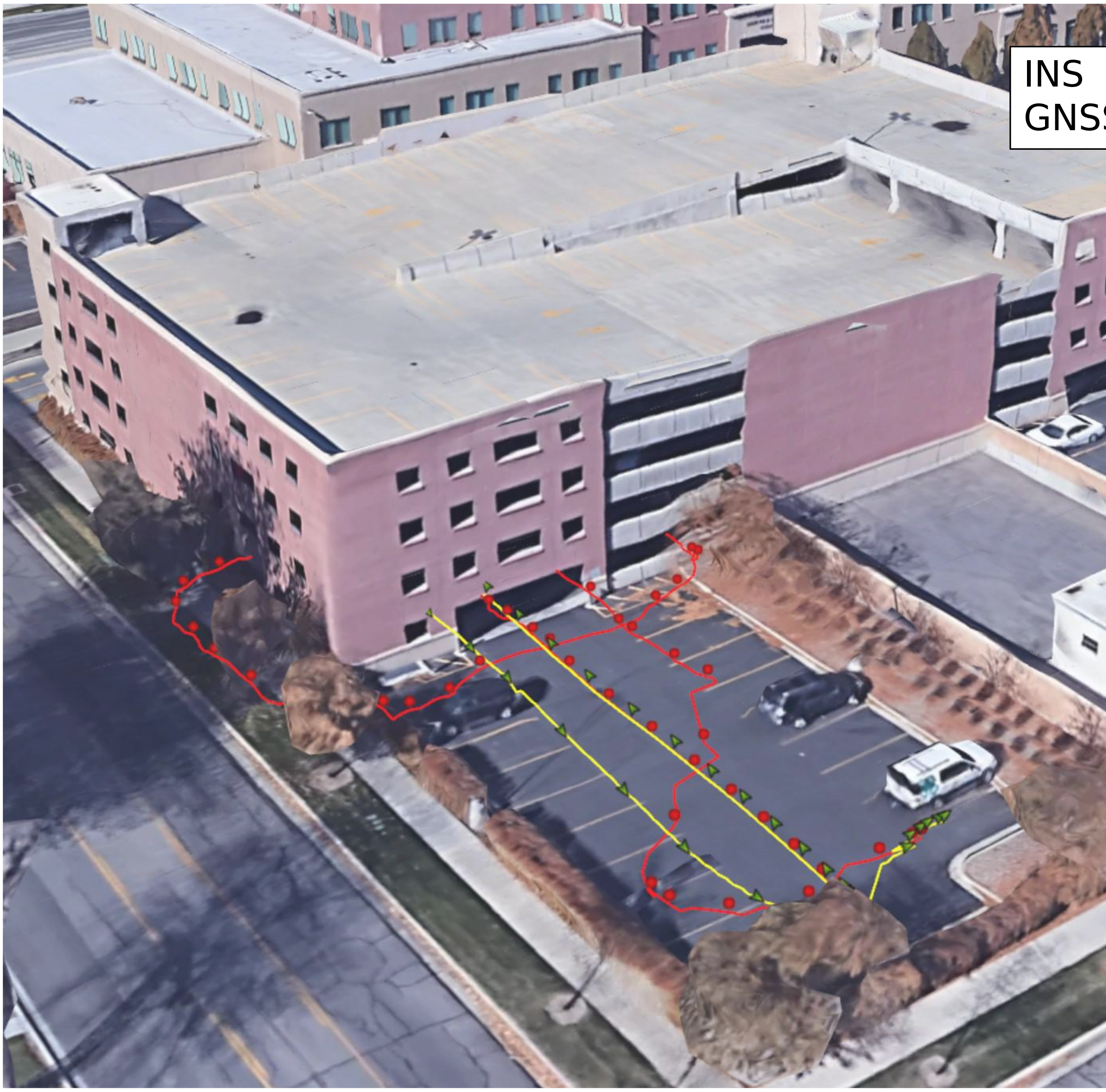
When GNSS fusion is re-enabled, error in the INS solution is removed and the INS position estimate jumps back onto the GNSS position. There is 2.5m of error between the dead reckoning position and the GNSS position.

9.2.2 Multi-Level Parking Garage

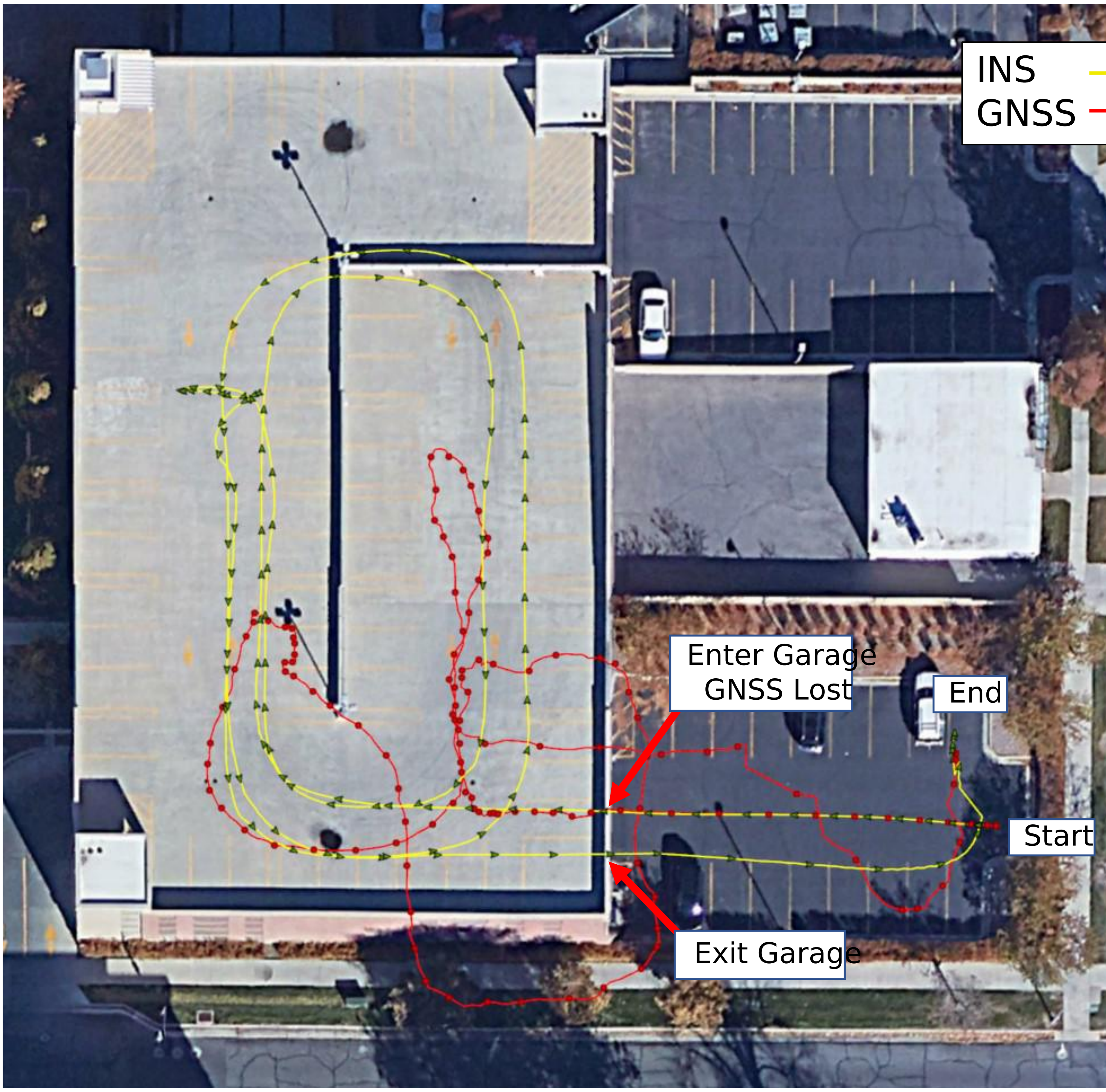
In this example our test vehicle drove in and out of a parking garage. The drive consisted of starting outside with GNSS fix, entering the garage (losing GNSS fix), driving up one level, parking, and then following the path back down and out of the garage where GNSS fix was regained.

Dead reckoning duration: **105 seconds**, **349 meters**

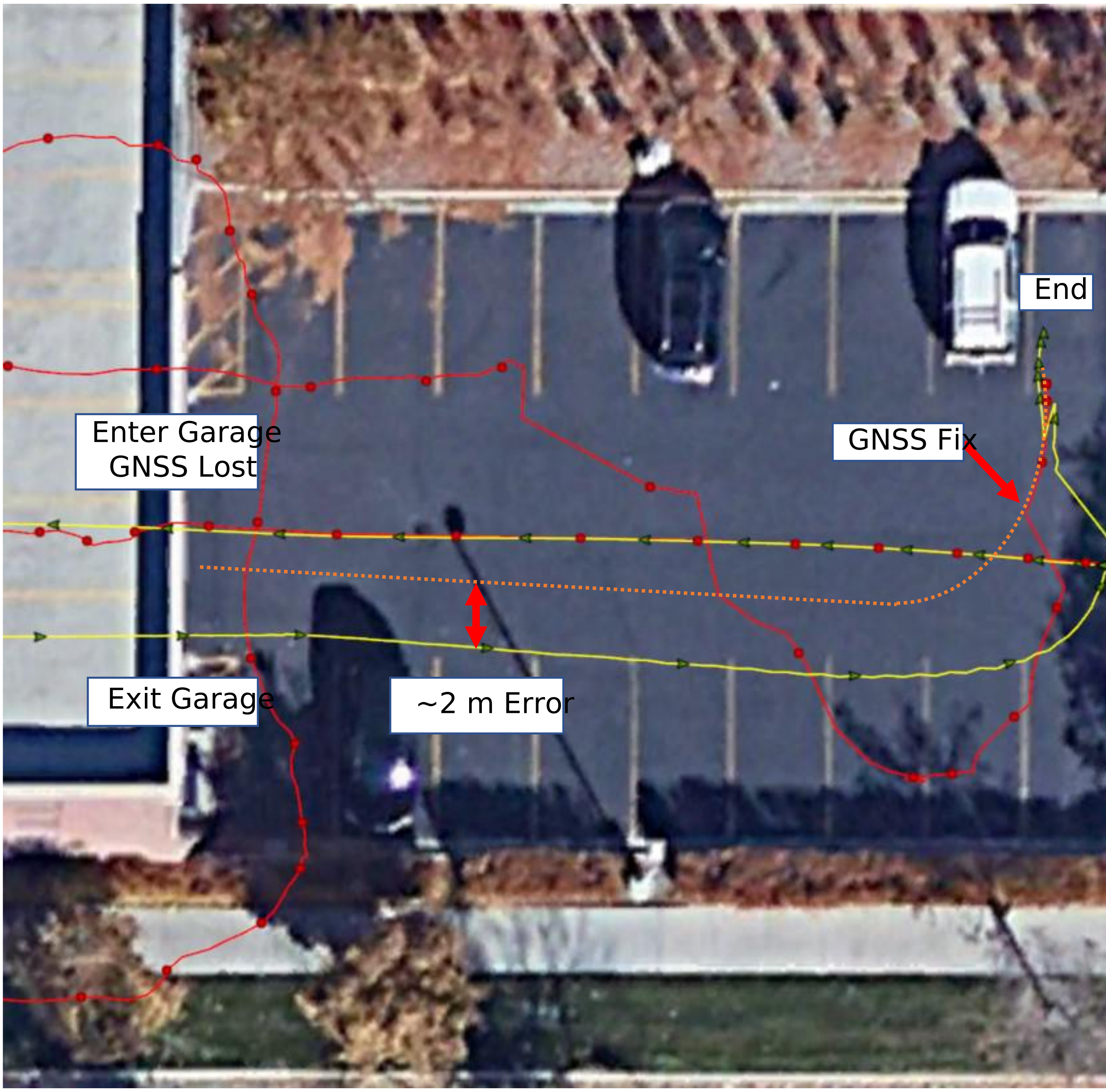
Exit position error: **~2 meters**, **0.6% drift**



Here we see outside parking lot where the test vehicle started and ended. GNSS fix was lost upon entry of the garage and regained several seconds after exiting the garage.



Above is the top view of the parking garage. When inside the garage, the GNSS fix is lost shown by the red line erratic deviation. The dead reckoning (INS) position shown by the yellow line matches the actual driven path.



GNSS fix was not regained until about 20 meters after exiting the garage, just prior to parking at the top right corner of the outside parking lot. The actual position is shown by the orange truth dotted line. GNSS position is shown by the red line and dead reckoning by the yellow INS line. GNSS fix occurs when the red GNSS line jumps and joins the orange truth dotted line. When exiting the garage, the position error was approximately 2 meters following 105 seconds of dead reckoning from GNSS outage.

9.2.3 Conclusion

The uINS with dead reckoning and without wheel sensor can estimate position to within ~3m over 100 seconds of typical automotive parking lot driving.

10. General Configuration

10.1 Infield Calibration

The *Infield Calibration* provides a method to 1.) zero IMU biases and 2.) zero INS attitude to align the INS output frame with the vehicle frame. These steps can be run together or independently.

10.1.1 Zeroing IMU Bias

Zeroing IMU bias is a way to remove permanent offsets in the sensor output that may have occurred as a result of manufacturing or high shock. The system must be completely stationary for accurate bias measurement. The current value for IMU biases stored in flash memory is viewable in `DID_INFIELD_CAL.imu` when infield calibration is inactive and `DID_INFIELD_CAL.sampleCount` is zero.

Accelerometer Bias

In order to correct accelerometer bias on a given axis, that axis must be sampled while measuring full gravity. Thus, only the accelerometer axes that are sampled while in the vertical direction can be corrected. In order to correct all accelerometer axes, all three axes must be sampled while oriented vertically. The sample can be done while the axis is pointed up, down, or both up and down for averaging.

Gyro Bias

All three axes of the gyros are sampled simultaneously, and the bias is stored in flash memory. The system must be completely stationary for accurate bias measurement. The system does not need to be level to zero the gyro biases.

10.1.2 Zeroing INS Attitude

The Infield Calibration process can be used to align or level the INS output frame with the vehicle frame. This is done by observing the X,Y,Z axes rotations necessary to level the orientation(s) sampled. Zeroing the INS attitude as part of the Infield Calibration routine provides a optimal and highly accurate method for measuring the attitude while stationary by averaging raw bias corrected accelerations.

Rotations cannot be computed for axes that are pointed vertically. For example, a single orientation sample with X and Y in the horizontal plane and Z pointed down will only be able to produce an X,Y rotation, and the Z rotation will remain zero. To compute all three rotations for the X,Y,Z axes, the system must be sampled at least twice, once while level and once while on its side.

The infield calibration process is generally useful for only small angle INS rotations and is not intended for those larger than 15° per axis. The user must set the INS rotation manually for larger rotations. The INS rotation is stored and accessed in `DID_FLASH_CONFIG.insRotation` in flash memory.

Because the sampled orientations are averaged together, it is recommended to only sample orientations that are at true 90° multiples of the vehicle frame.

The zero INS attitude feature assumes there are flat rigid surface(s) attached to the uINS about which the system can be leveled. If the working surface is not level or additional precision is desired, each orientation sampled can have an additional sample taken with ~180° yaw offset to cancel out tilt of the working surface.

If Infield Calibration is not adequate, the INS may be leveled or aligned manually.

10.1.3 Infield Calibration Process

The following process can be used to used to improve the IMU calibration accuracy and also align or level the INS to the vehicle frame.

1. **Prepare Leveling Surface** - Ensure the system is stable and stationary on a near-level surface with one of three axes in the vertical direction.

2. **Initialize the Mode** - Clear any prior samples and set the calibration mode by setting `DID_INFIELD_CAL.state` to one of the following:

```
INFIELD_CAL_STATE_CMD_INIT_ZERO_IMU            = 1, // Zero accel and gyro biases.
INFIELD_CAL_STATE_CMD_INIT_ZERO_GYRO           = 2, // Zero only gyro  biases.
INFIELD_CAL_STATE_CMD_INIT_ZERO_ACCEL          = 3, // Zero only accel biases.
INFIELD_CAL_STATE_CMD_INIT_ZERO_ATTITUDE       = 4, // Zero (level) INS attitude by adjusting INS rotation.
INFIELD_CAL_STATE_CMD_INIT_ZERO_ATTITUDE_IMU   = 5, // Zero gyro and accel biases.  Zero (level) INS attitude by adjusting INS rotation.
INFIELD_CAL_STATE_CMD_INIT_ZERO_ATTITUDE_GYRO  = 6, // Zero only gyro  biases.  Zero (level) INS attitude by adjusting INS rotation.
INFIELD_CAL_STATE_CMD_INIT_ZERO_ATTITUDE_ACCEL = 7, // Zero only accel biases.  Zero (level) INS attitude by adjusting INS rotation.

INFIELD_CAL_STATE_CMD_INIT_OPTION_DISABLE_MOTION_DETECT = 0x00010000,    // Bitwise AND this with the above init commands to disable motion detection dur
```

   Zeroing accelerometer biases requires that any of the X,Y,Z axes be vertically aligned with gravity during sampling. This is indicated by bit `INFIELD_CAL_STATUS_AXIS_NOT_VERTICAL = 0x01000000` in `DID_INFIELD_CAL.status`.

   By default, the system must also be stationary without any movement during sampling. This is indicated by bit `INFIELD_CAL_STATUS_MOTION_DETECTED = 0x02000000` is set in `DID_INFIELD_CAL.status`. Motion detection can be disabled to make the system more tolerant during sampling. To do this, bitwise and `INFIELD_CAL_STATE_CMD_INIT_OPTION_DISABLE_MOTION_DETECT = 0x00010000` with the initialization command . As an example, the command to initialize *INS alignment with zero IMU bias* with motion detection disabled is as follows:

```
(INFIELD_CAL_STATE_CMD_INIT_ZERO_ATTITUDE_IMU | INFIELD_CAL_STATE_CMD_INIT_OPTION_DISABLE_MOTION_DETECT);

0x00010101 = (0x00000101 | 0x00010000);
```

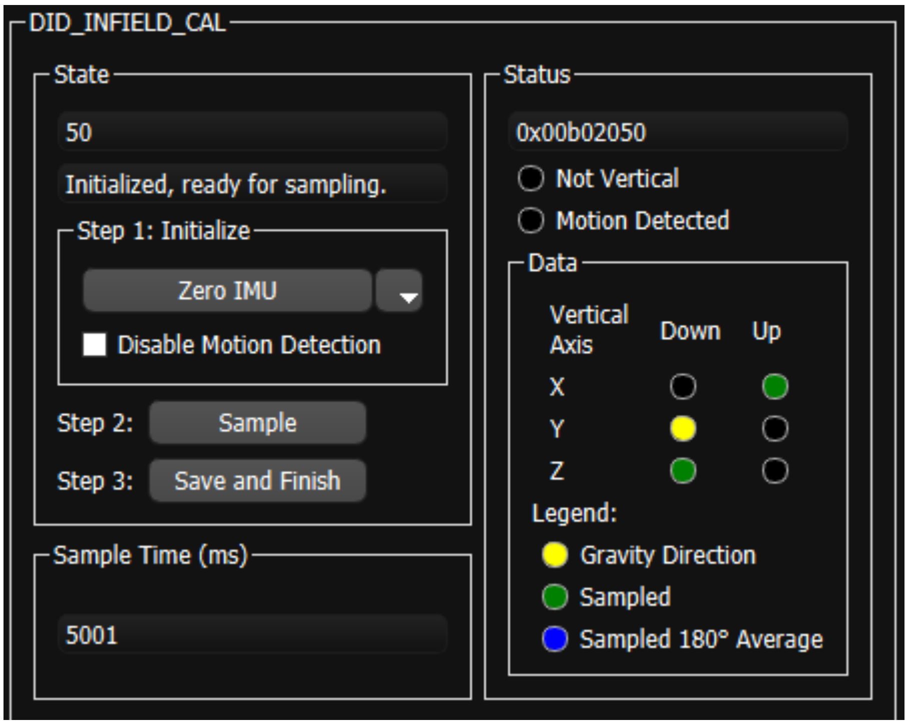
3. **Sample Orientation(s)** - Initiate sampling of one or more orientations by setting `DID_INFIELD_CAL.state` to `INFIELD_CAL_STATE_CMD_START_SAMPLE = 8`. Sampling per orientation will take 5 seconds and completion is indicated when `DID_INFIELD_CAL.state` switches to `INFIELD_CAL_STATE_SAMPLING_WAITING_FOR_USER_INPUT = 50`.

   - **Sample Same Orientation w/ +180° Yaw** - If the working surface is not level, two samples per orientation can be taken to cancel out the tilt of the working surface. Rotate the system approximately 180° in yaw (heading) and initiate the sampling a second time for a given orientation.
   - **Sample Up to Six Orientations** - The sampling process can be done for up to six orientations (X,Y,Z pointed up and down). Each sample will be automatically associated with the corresponding vertical axis and direction. All orientations will be averaged together for both the zero IMU bias and zero INS attitude.

4. **Store IMU Bias and/or Align INS** - Following sampling of the orientations, set `DID_INFIELD_CAL.state` to `INFIELD_CAL_STATE_CMD_SAVE_AND_FINISH = 9` to process and save the infield calibration to flash memory. The built-in test (BIT) will run once following this to verify the newly adjusted calibration and `DID_INFIELD_CAL.state` will be set to `INFIELD_CAL_STATE_FINISHED`.

EvalTool or CLTool for Infield Cal

The EvalTool IMU Settings tab provides a user interface to read and write the DID_INFIELD_CAL message.



DID_INFIELD_CAL

State

50

Initialized, ready for sampling.

Step 1: Initialize

Zero IMU

☐ Disable Motion Detection

Step 2: Sample

Step 3: Save and Finish

Sample Time (ms)

5001

Status

0x00b02050

○ Not Vertical

○ Motion Detected

Data

Vertical Axis	Down	Up
X	○	🟢
Y	🟡	○
Z	🟢	○

Legend:

🟡 Gravity Direction

🟢 Sampled

🔵 Sampled 180° Average

CLTOOL INFIELD CAL

The following options can be used with the CLTool to edit the infield calibration (DID_INFIELD_CAL).

```
cltool -c /dev/ttyS2 -edit DID_INFIELD_CAL
```

Below is an example of the CLTool edit view of the DID_INFIELD_CAL message.

```
$ Inertial Sense.  Connected.  Press CTRL-C to terminate.  Rx 13657

(94) DID_INFIELD_CAL:      W up, S down
                     0   calData[2].down.dev[1].acc[1]
                     0   calData[2].down.dev[1].acc[2]
                     0   calData[2].down.yaw
                     0   calData[2].up.dev[0].acc[0]
                     0   calData[2].up.dev[0].acc[1]
                     0   calData[2].up.dev[0].acc[2]
                     0   calData[2].up.dev[1].acc[0]
                     0   calData[2].up.dev[1].acc[1]
                     0   calData[2].up.dev[1].acc[2]
                     0   calData[2].up.yaw
          0.0398919582   imu[0].acc[0]
        0.000717461109   imu[0].acc[1]
            9.67872334   imu[0].acc[2]
         0.00583727891   imu[0].pqr[0]
          0.0135380113   imu[0].pqr[1]
        -0.00342554389   imu[0].pqr[2]
          0.0874974579   imu[1].acc[0]
          -0.167159081   imu[1].acc[1]
            9.67817783   imu[1].acc[2]
        -0.00111889921   imu[1].pqr[0]
        -0.00523020467   imu[1].pqr[1]
         0.00455262465   imu[1].pqr[2]
                     0   sampleTimeMs
                    50   state
            0x00B01000 * status
```

10.2 INS & GNSS Configuration

10.2.1 Translation

The uINS can be mounted and operated in any arbitrary orientation. It is often desirable and conventional to translate the uINS output so that it is rotated into the vehicle frame and located at certain location for control and navigation of the vehicle. This is done using the ***IMU Rotation***, ***INS Rotation***, and ***INS Offset*** parameters.

In most common applications, output is translated to the vehicle frame (X to the front, Y to the right, and Z down):

- *IMU Rotation* provides gross rotation of the IMU output in multiples of 90°.
- *INS Rotation* provides small angle alignment of the INS output.
- *INS Offset* shifts the location from the INS output.

IMU Rotation (Hardware Native Frame to Sensor Frame)

The *IMU rotation* is used to rotate the IMU and magnetometer output from the hardware native frame to the sensor frame by **multiples of 90°**. This is done using the `SENSOR_CFG_SENSOR_ROTATION_MASK` bits of the `DID_FLASH_CONFIG.sensorConfig` as defined in `enum eSensorConfig`. The IMU rotation is defined in X,Y,Z rotations about the corresponding axes and applied in the order of Z,Y,X. This rotation is recommended for gross rotations.

INS Rotation

The *INS rotation* is used to convert the INS output from the sensor frame to the vehicle frame. This is useful if the sensor frame and vehicle frame are not aligned. The actual INS rotation parameters are `DID_FLASH_CONFIG.insRotation[3]` (X, Y, Z) in radians. The *INS rotation* values describes the rotation from the INS sensor frame to the intermediate frame in order of Z, Y, X.

INS Offset

The *INS offset* is used to shift the location of the INS output and is applied following the INS Rotation. This offset can be used to move the uINS location from the origin of the sensor frame to any arbitrary location, often a control navigation point on the vehicle.

Manually Aligning the INS After Mounting

NOTE for use:

- The Infield Calibration process can be used instead of this process to automatically measure and align the INS with the vehicle frame for INS rotations less than 15°.
- If using software release 1.8.4 or newer, we recommend using the `DID_FLASH_CONFIG.sensorConfig` to rotate the sensor frame by 90° to near level before following the steps below.

The following process uses the uINS to measure and correct for the uINS mounting angle.

1. Set `DID_FLASH_CONFIG.insRotation` to zero.

2. Set the sensor on the ground at various known orientations and record the INS quaternion output (DID_INS_2). Using the Euler output (DID_INS_1) can be used if the pitch is less than 15°. It is recommended to use the EKF Zero Motion Command to ensure the EKF bias estimation and attitude have stabilized quickly before measuring the INS attitude.

3. Find the difference between the known orientations and the measured INS orientations and average these differences together.

4. Negate this average difference and enter that into the `DID_FLASH_CONFIG.insRotation`. This value is in Euler, however it is OK for this step as this rotation should have just been converted from quaternion to Euler and will be converted back to quaternion on-board for the runtime rotation.

10.2.2 Infield Calibration

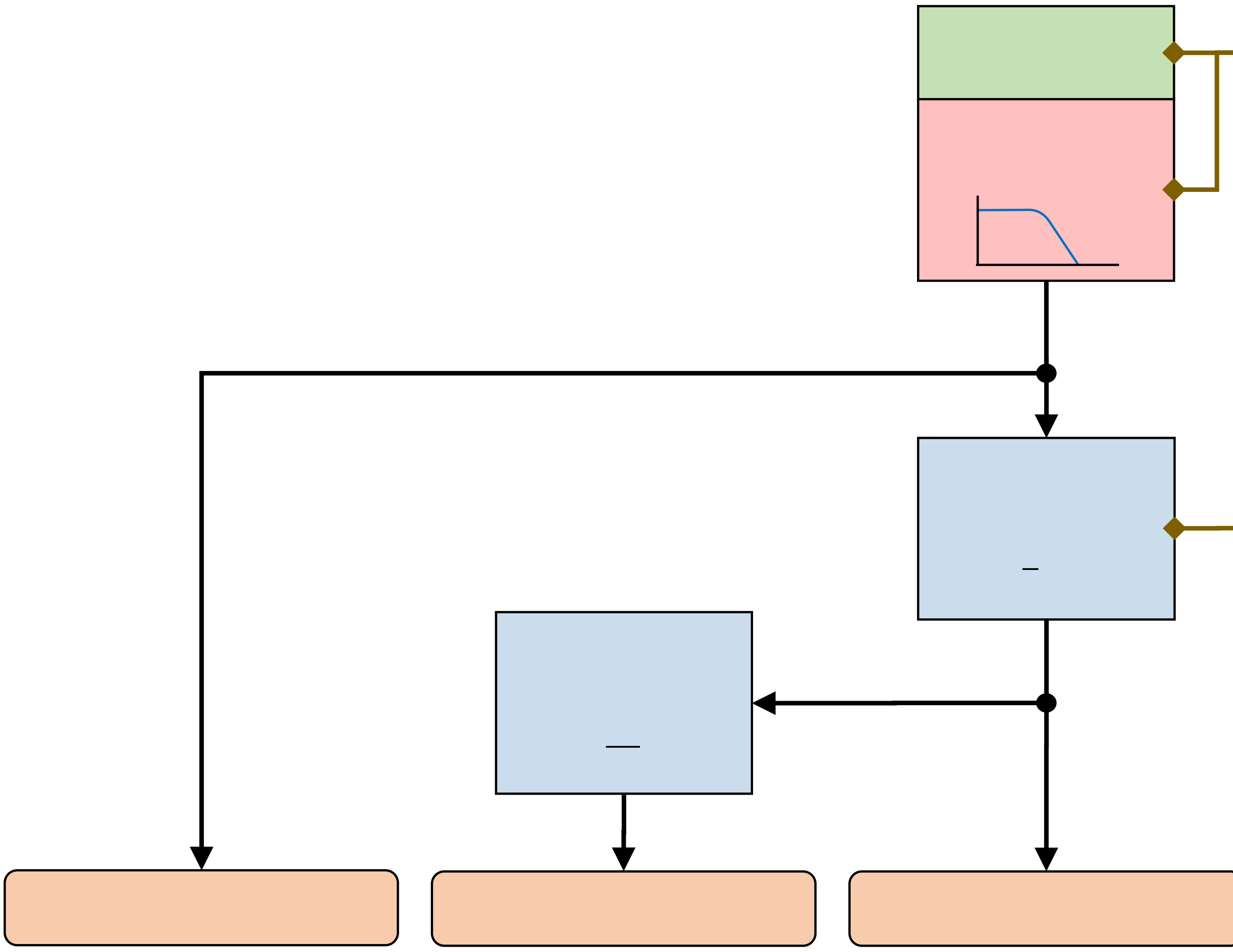
The *Infield Calibration* provides a method to 1.) zero IMU biases and 2.) zero INS attitude to align the INS output frame with the vehicle frame. These steps can be run together or independently.

10.2.3 GNSS Antenna Offset

If the setup includes a significant distance (40cm or more) between the GPS antenna and the uINS central unit, enter a non-zero value for the GPS lever arm, `DID_FLASH_CONFIG.gps1AntOffset` (or `DID_FLASH_CONFIG.gpsAnt2Offset`) X,Y,Z offset in meters from Sensor Frame origin to GPS antenna. The sensor frame is labeled on the uINS EVB case.

10.2.4 IMU Sample and Navigation Periods

The IMU sample period is configured by setting `DID_FLASH_CONFIG.startupImuDtMs` in milliseconds. This parameter determines how frequently the IMU is measured and data integrated into the `DID_PREINTEGRATED_IMU` data. `DID_FLASH_CONFIG.startupImuDtMs` also automatically sets the bandwidth of the IMU anti-aliasing filter to less than one half the Nyquist frequency (i.e. < 250 / startupImuDtMs). The IMU anti-aliasing filter bandwidth can also be overridden to another frequency by setting bits `SENSOR_CFG_GYR_DLPF` and `SENSOR_CFG_ACC_DLPF` in `DID_FLASH_CONFIG.sensorConfig`.



The INS and AHRS kalman filter update period is configured using `DID_FLASH_CONFIG.startupNavDtMs`. This parameter also sets the integration period for the Preintegrated IMU or conning and sculling (delta theta, delta velocity) integrals. The `DID_DUAL_IMU` is the derivative of the `DID_PREINTEGRATED_IMU` value over a single integration period and serves as an anti-aliased moving average of the IMU value.

10.2.5 INS-GNSS Dynamic Model

The `DID_FLASH_CONFIG.insDynModel` setting allows the user to adjust how the EKF behaves in different dynamic environments. All values except for 2 (STATIONARY) and 8 (AIR <4g) are experimental. The user is encouraged to attempt to use different settings to improve performance, however in most applications the default setting, 8: airborne <4g, will yield best performance.

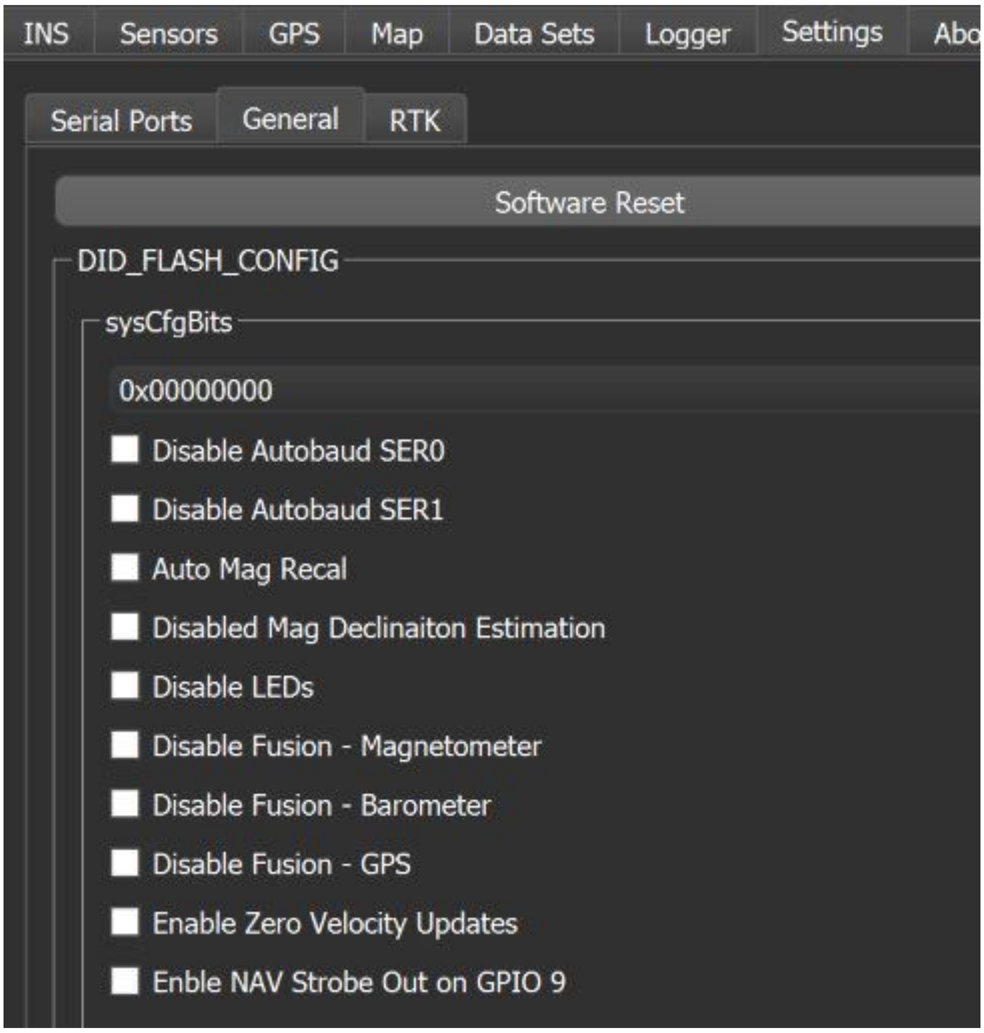
The STATIONARY configuration (insDynModel = 2) can be used to configure the EKF for static applications. It is a permanent implementation of the Zero Motion Command which will reduce EKF drift under stationary conditions.

10.2.6 Disable Magnetometer and Barometer Updates

Magnetometer and barometer updates (fusion) into the INS and AHRS filter (Kalman filter) can be disabled by setting the following bits in `DID_FLASH_CONFIG.sysCfgBits`.

Bit Name	Bit Value	Description
SYS_CFG_BITS_DISABLE_MAGNETOMETER_FUSION	0x00001000	Disable magnetometer fusion into EKF
SYS_CFG_BITS_DISABLE_BAROMETER_FUSION	0x00002000	Disable barometer fusion into EKF

These settings can be disabled using the General Settings tab of the EvalTool.



0x00000000

- [ ] Disable Autobaud SER0
- [ ] Disable Autobaud SER1
- [ ] Auto Mag Recal
- [ ] Disabled Mag Declinaiton Estimation
- [ ] Disable LEDs
- [ ] Disable Fusion - Magnetometer
- [ ] Disable Fusion - Barometer
- [ ] Disable Fusion - GPS
- [ ] Enable Zero Velocity Updates
- [ ] Enble NAV Strobe Out on GPIO 9

10.2.7 Disable Zero Velocity Updates

Zero velocity updates (ZUPT) rely on GPS and/or wheel encoder data. In some cases there can be a slight lag/deviation when starting motion while simultaneously rotating. This is because GPS data is updated at 5 Hz and it takes a few samples to detect motion after a period of no motion. When ZUPT is enabled, it acts as a virtual velocity sensor telling the system that its velocity is zero. It may conflict briefly with GPS velocity observation when starting motion. If a slight lag at the beginning of motion is an issue, ZUPT may be disabled. Generally it should be enabled (Default). It can be disabled using `DID_FLASH_CONFIG.sysCfgBits` or using the General Settings tab of the EvalTool.

10.2.8 Disable Zero Angular Rate Updates

Zero angular rate updates (ZARU) rely on analysis of either IMU (gyro) data or wheel encoders when available. When angular motion is very slow and no wheel encoders are available a zero angular rate may be mistakenly detected, which will lead to gyro bias estimation errors. In these cases it can be beneficial to disable ZARU if the applications has slow rotation rates (approximately below 3 deg/s). It is not encouraged to disable ZARU if there is no rotation or faster rotation. It can be disabled using `DID_FLASH_CONFIG.sysCfgBits` or using the General Settings tab of the EvalTool.

10.3 System Configuration

See the Binary Protocol page for descriptions of each flash configuration value and enumeration bit values.

10.3.1 Serial Port Baud Rate

Choose the baud rate for serial communications through either available port. (3000000, 921600, 460800, 230400, 115200, 57600, 38400, 19200).

Manual Baud Configuration

The uINS baud rate can be manually set by changing the following flash configuration parameters:

Configuration	Description
`DID_FLASH_CONFIG.ser0BaudRate`	baud rate for uINS serial port 0
`DID_FLASH_CONFIG.ser1BaudRate`	baud rate for uINS serial port 1
`DID_FLASH_CONFIG.ser2BaudRate`	baud rate for uINS serial port 2

These parameters can be changed using the EvalTool or the CLTool. The following examples show how the EvalTool and CLtool can be used to set the uINS serial port 1 baud rate to 460,800 bps.

`EvalTool -> Data Sets -> DATA_FLASH_CONFIG.ser1BaudRate = 460800`

```
cltool -c COM# -flashConfig=ser0BaudRate=460800
```

10.4 Time Synchronization

10.4.1 INS & GPS Timestamps

The uINS output messages are timestamped using GPS time-base because this time is known immediately following GPS signal reception. Conversion from GPS time to UTC time requires knowledge of the number of leap seconds (GPS-UTC) offset. This value is received periodically (every 12.5 minutes) and is available in the `DID_GPS1_POS` and `DID_GPS1_RTK_POS` (gps_pos_t) messages. GPS leap seconds is 18 seconds as of December 31, 2016 and will change in the future.

The original designers of GPS chose to express time and date as an integer week number (starting with the first full week in January 1980) and a time of week (often abbreviated to TOW) expressed in seconds. Working with time/date in this form is easier for digital systems than the more "conventional" year/month/day, hour/minute/second representation. Most GNSS receivers use this representation internally and converting to a more "conventional form" externally.

GPS to UTC Time Conversion

UTC time is found by subtracting GPS leap seconds from the GPS time.

10.4.2 GPS Time Synchronization

Systems connected to the uINS can be time synchronized using the GPS PPS timepulse signal and any message containing GPS time. The actual time of the GPS PPS timepulse signal is the same as any message with GPS time rounded down to the second. The following pseudo code illustrates how this is done.

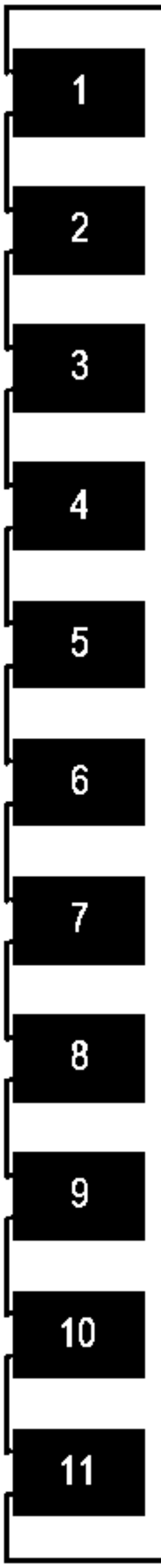
```
// GPS Time Synchronization - Find the difference between local time and GPS time:

// 1. Sample your local time on the rising edge of the GPS PPS timepulse signal.
double ppsLocalTime = localTime();

// 2. Read the GPS time from any message WITHIN ONE SECOND FOLLOWING the GPS PPS timepulse signal.
double gpsTime = readGpsMessageTime();             // within one second after GPS PPS

// 3. Find the difference between the GPS PPS local time and the GPS time rounded down to the
//    nearest second (443178.800 s down to 443178 s, or 443178800 ms down to 443178 s).
double localToGpsTimeTemp = ppsLocalTime - floor(gpsTime);

// 4. Error check to ensure you have a consistent solution
static double localToGpsTimeLast;
double localToGpsTime;

if (fabs(localToGpsTimeLast - localToGpsTimeTemp) < 0.002)        // within 2ms
{
        localToGpsTime = localToGpsTimeTemp;
}
localToGpsTimeLast = localToGpsTimeTemp; // Update history

// Local time can now be converted at anytime to GPS time using 'localToGpsTime' difference.
double currentGpsTime = localTime() + localToGpsTime;
```

10.4.3 Using the Strobe Input Pins

The uINS has several strobe input pins which can be configured to cause the uINS to report both its internal time and full navigation solution at the moment when triggered.

Strobe I/O Events

Strobe input and output (I/O) events are used for time and data synchronization.


1
2
3
4
5
6
7
8
9
10
11

STROBE pins on the µIMU, µAHRS, and µINS Module - Top View

Strobe Input (Time Sync Input)

Strobe inputs are used to timestamp digital events observed on any of the pins labeled STROBE, e.g. camera shutter signals. A STROBE input event occurs when the logic level of any STROBE pin is toggled. The transition direction can be set so that the STROBE event triggers on a rising edge, or a falling edge. An internal 100K pull-up or pull-down resistor is enabled, depending on the assertion direction. External pull-up or pull-down resistors are not necessary.

The STROBE input will trigger on the edge type specified. However, the minimum period between STROBE input pulses is 1 ms. The measurement and timestamp resolution are both 1 ms.

The following pins can be used for STROBE input.

Signal	Module Pin	EVB-1 Pin	Rugged Pin	EVB-2
G2	5	H2-4	12	H7-6
G5	9	H6-3		H7-9
G8	8	H6-6		H7-12
G9	10	Button "B"		H7-13

To use a pin as a Strobe Input pin, the I/O must be configured as a strobe input. Additionally, the triggering edge must be set using the following bits in `DID_FLASH_CONFIG.ioConfig`.

Bit Name	Bit Value	Description
IO_CONFIG_STROBE_TRIGGER_LOW	0x00000000	Trigger strobe on falling edge
IO_CONFIG_STROBE_TRIGGER_HIGH	0x00000001	Trigger strobe on rising edge

Pushbutton “B” on the EVB asserts a logic low to G9 (pin 10) of the uINS and can be used to test the STROBE input functionality.

**Note: Holding pin 9 low at startup enables SPI which uses pins 5 and 8 making them unavailable to be used as Strobe Inputs. If pin 9 is not held low, the internal pullup resistor holds it high at startup. This sets pins 5 and 8 as inputs which can be used as Strobe Inputs.

A STROBE input event causes a timestamp message and INS2 message to be transmitted. The ASCII messages `$PSTRB` and `$PINS2` messages are sent by default but can be disabled and replaced by the binary messages `DID_STROBE_IN_TIME` and `DID_INS_2` if the RMC bit `RMC_BITS_STROBE_IN_TIME` is set for the given serial port.

Example:

```
rmc_t rmc;
rmc.bits = RMC_BITS_STROBE_IN_TIME;

int messageSize = is_comm_set_data(comm, DID_RMC, offsetof(rmc_t, bits), sizeof(uint64_t), &rmc);
if (messageSize != serialPortWrite(serialPort, comm->buffer, messageSize))
{
    printf("Failed to write save persistent message\r\n");
}
```

Table 2 - DID_STROBE_IN_TIME message transmitted following a SYNC input event.

Field	Type	Description
week	uint32_t	Weeks since January 6th, 1980
timeOfWeekMs	uint32_t	Time of week (since Sunday morning) in milliseconds, GMT.
pin	uint32_t	STROBE input pin
count	uint32_t	STROBE serial index number

The STROBE input event also causes the HDW_STATUS_STROBE_IN_EVENT (0x00000020) bit of the hdwStatus field in INS output (DID_INS_1, DID_INS_2, DID_INS_3, and DID_INS_4) to be set, allowing users to identify strobe input events using the INS output.

TROUBLESHOOTING INPUT STROBE

If the STOBE input does not appear to be functioning properly, an oscilloscope or fast multi-meter can be used to probe the actual STROBE line to ensure the proper 0V to 3.3V voltage swing is present. The following two tests can be used to evaluate the proper function of the strobe source and the uINS strobe input.

TEST 1: Identify if the uINS strobe input is configured properly and has a low input impedance:

1. Disconnect your strobe source from the uINS. Use a 1K ohm resistor as a pull-up resistor between 3.3V and the uINS strobe input and measure the strong input line.
2. Repeat using the resistor as a pull-down resistor between ground and the strobe input and measure the strobe input line.

This will tell if the uINS strobe input is somehow being driven internally or not configured correctly. If it is functioning correctly, the line will toggle from 0V to +3.3V following the resistor pull-down and pull-up.

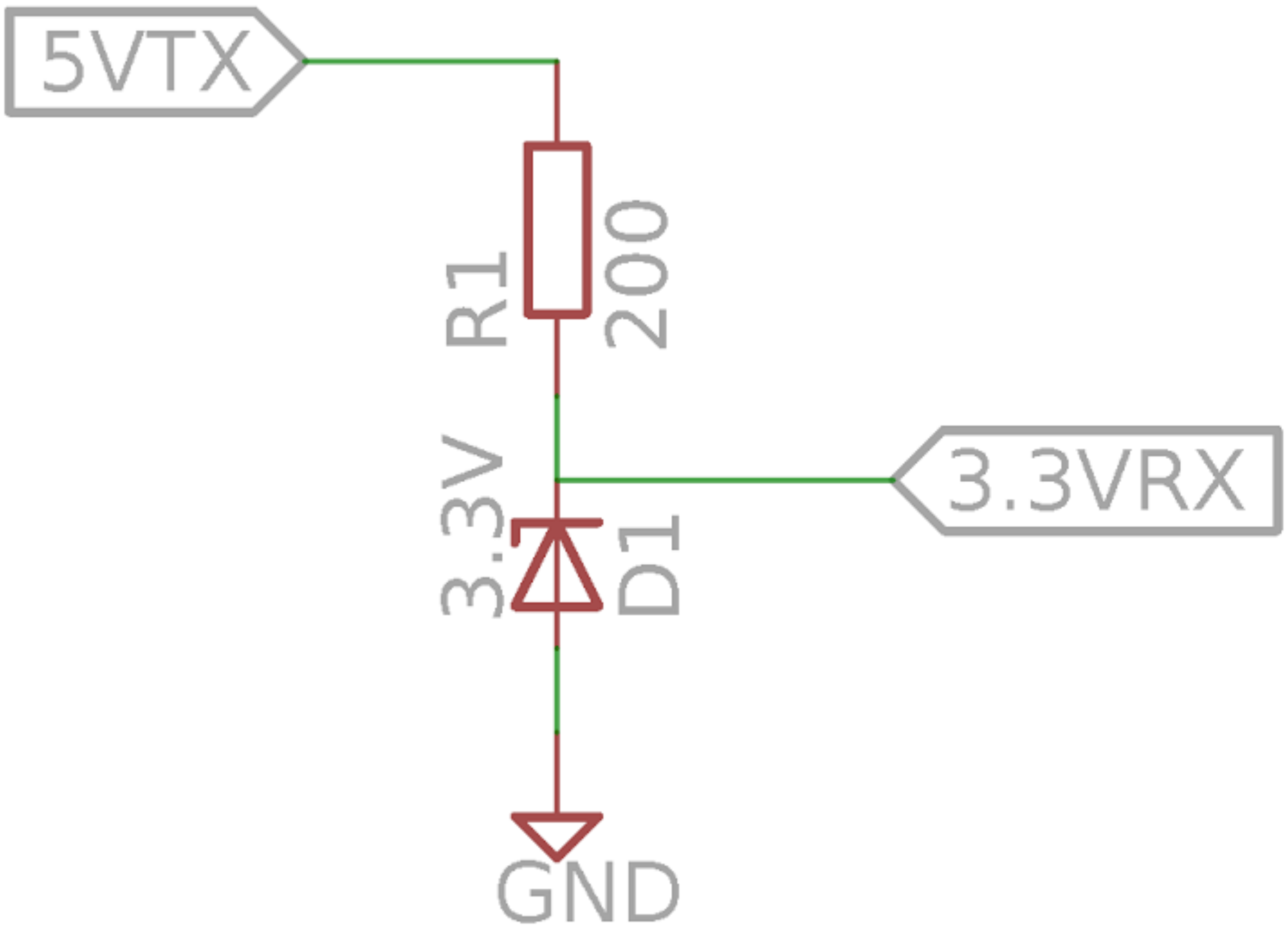
TEST 2: Identify if your strobe source is driving correctly:

1. With the uINS strobe input disconnected from your strobe driving circuit, probe the output of the strobe driving circuit and observe what levels it toggles between.
2. Attach a 1M ohm pull-down resistor from ground to the strobe output and observe the strobe voltage swing.

If the circuit is working correctly, it should drive the strobe output from 0V to +3.3V despite the 1M ohm pull-down resistor.

INPUT VOLTAGE LEVEL SHIFTER

The maximum input voltage for strobe lines (any pin on the uINS) is 3.6V. A level shifter may be used to convert any strobe signal that is larger than 3.3V. The following figure shows two passive level shifter circuits, a zener diode voltage clamp and a resistor voltage divider.



With a Zener diode

These circuits are beneficial because of their simplicity. An active, powered level shifter may also be used and necessary.

Strobe Output (Preintegrated IMU Period)

The STROBE output feature is used to indicate start and end of the preintegrated IMU period (and navigation filter IMU updates) by toggling . STROBE output is enabled on pin 9 by setting bit SYS_CFG_BITS_ENABLE_NAV_STROBE_OUT_GPIO_9 (0x00100000) of DID_FLASH_CONFIG.sysCfgBits.

Configuring Message Output

By default, triggering a strobe input event will cause the uINS to produce an ASCII PINS2 message as well as a PSTRB message which contains the time stamp of the strobe event.

To instead send a binary DID_INS_2 and DID_STROBE_IN_TIME message, set the `RMC_BITS_STROBE_IN_TIME` flag of `DID_RMC/bits` field.

10.5 Zero Motion Command

The *Zero Motion Command* is user initiated and informs the EKF that the system is stationary on the ground. It is used to aid in IMU bias estimation which can reduce drift in the INS attitude. It works as a virtual velocity and angular rate sensor to provide velocity and angular rate observations when the INS is stationary (zero velocity and zero angular rate). This is done for a period of two seconds after the *Zero Motion Command* is received. The Zero Motion Command is beneficial for the following reasons:

- Overriding incorrect GPS motion caused by weak GPS signal.
- Speeding up gyro biases convergence time when there is no GPS signal.

In normal AHRS mode (stationary with or without GPS), only the IMU gyro biases are estimated by the EKF. Setting `DID_FLASH_CONFIG.insDynModel = DYN_STATIONARY (2)` is equivalent to continually issuing the zero motion command.

To use the *Zero Motion Command*:

1. Ensure the system is stationary on the ground.
2. Send the *Zero Motion Command* either once or continuously while the system is stationary. This can be done either by using the *Zero Motion* button in the EvalTool General Settings tab or by sending the DID_SYS_CMD binary message.
3. After sending the *Zero Motion Command*, wait for the INS_STATUS_DO_NOT_MOVE status bit to clear in DID_INS_x.insStatus before moving the system. This flag takes about 2 seconds to clear following the last *Zero Motion Command*.

Applying this command more than one time can further improve the IMU bias estimation.

Warning

M *Zero MMtion ComMmand* while the system is moving can cause incorrect IMU bias estimates and lead to poor INS performance. It is important to make sure that the system is stationary when using the *Zero Motion Command*.

10.6 UART Interface

The uINS has different UART TTL serial ports. These serial ports can be converted from TTL to RS232 or RS422 using a level converter, such as found on the Rugged, EVB-1, and EVB-2 carrier boards.

10.6.1 Actual Baud Rates - UART (Serial 0 and Serial 2)

The serial ports use different peripherals so the actual baud rates of the ports differ. Serial ports 0 and 2 are UART and Serial 1 is USART.

Due to UART hardware integer rounding on the uINS serial ports 0 and 2, the following table outlines the difference between desired and actual UART baud rate settings. Note that the difference is more significant at higher baud rates.

Desired Baud Rate (bps)	Actual Baud Rate (bps)
19200	19211
38400	38422
57600	57870
115200	115740
230400	234375
460800	468750
921600	937500
3000000	3125000

11. SDK

INERTIALSENSE
autonomous navigation solutions

11.1 Inertial Sense SDK

The Inertial Sense software development kit (SDK) is hosted on GitHub.

SDK - The Inertial Sense open source software development kit provides quick integration for communication with the Inertial Sense product line, including the uINS, uAHRS, and uINS. It includes data logger, math libraries, and serial port interface for Linux and Windows environments.

EvalTool executable - Graphical Windows-based desktop program that allows you to explore and test functionality of the Inertial Sense products in real-time. It has scrolling plots, 3D model representation, table views of all data, data logger, and firmware updating interface for the uINS, uAHRS, or uIMU. The EvalTool can simultaneously interface with multiple Inertial Sense devices.

CLTool - Command line utility that can be used to communicate, log data, and update firmware for Inertial Sense products. Additionally, InertialSenseCLTool serves as example source code to demonstrate how to integrate the Inertial Sense SDK into your own source code. The InertialSenseCLTool can be compiled in Linux and Windows.

EVB-2 - Multi-purpose hardware evaluation and development kit for the uINS. The EVB-2 includes the uINS-G2 with Dual GNSS, RTK heading / positioning, onboard logging to micro SD card, 915MHz XBee radio for RTK base corrections, WiFi and BLE interface, serial and SPI communications to uINS interface, and Microchip SAME70 processor as communications bridge and user project development environment.

Documents

- User Manual, Datasheet, and Dimensions

Downloads

- SDK Example Projects - Compliable source code projects that demonstrations of how to use the SDK.
- Software Releases - uINS, uAHRS, uIMU, and EVB-2 firmware and application installers.
- SDK & CLTool Source Code - open source SDK repository with command line tool and example C/C++ source code.

Hardware Design Files

- uINS PCB Design Libraries - Schematic and layout files for printed circuit board designs.
- uINS, uAHRS, uIMU CAD Model - 3D step model of uINS and EVB used for CAD and circuit board designs.

Support

- Email - support@inertialsense.com

Open Source License

11.2 Example Projects

11.2.1 Binary Communications Example Project

This IS Communications Example project demonstrates binary communications with the Inertial Sense Products (uINS, uAHRS, and uIMU) using the Inertial Sense SDK.

Files

Project Files

- ISCommunicationsExample.c

SDK Files

- data_sets.c
- data_sets.h
- ISComm.c
- ISComm.h
- serialPort.c
- serialPort.h
- serialPortPlatform.c
- serialPortPlatform.h

Implementation

STEP 1: ADD INCLUDES

```
// Change these include paths to the correct paths for your project
#include "../../src/ISComm.h"
#include "../../src/serialPortPlatform.h"
#include "../../src/ISPose.h"
```

STEP 2: INIT COMM INSTANCE

```
    is_comm_instance_t comm;
    uint8_t buffer[2048];

    // Initialize the comm instance, sets up state tracking, packet parsing, etc.
    is_comm_init(&comm, buffer, sizeof(buffer));
```

STEP 3: INITIALIZE AND OPEN SERIAL PORT

```
    serial_port_t serialPort;

    // Initialize the serial port (Windows, MAC or Linux) - if using an embedded system like Arduino,
    //  you will need to handle the serial port creation, open and reads yourself. In this
    //  case, you do not need to include serialPort.h/.c and serialPortPlatform.h/.c in your project.
    serialPortPlatformInit(&serialPort);

    // Open serial, last parameter is a 1 which means a blocking read, you can set as 0 for non-blocking
    // you can change the baudrate to a supported baud rate (IS_BAUDRATE_*), make sure to reboot the uINS
    //  if you are changing baud rates, you only need to do this when you are changing baud rates.
    if (!serialPortOpen(&serialPort, argv[1], IS_BAUDRATE_921600, 1))
    {
        printf("Failed to open serial port on com port %s\r\n", argv[1]);
        return -2;
    }
```

STEP 4: STOP ANY MESSAGE BROADCASTING

```
    int messageSize = is_comm_stop_broadcasts_all_ports(comm);
    if (messageSize != serialPortWrite(serialPort, comm->buf.start, messageSize))
    {
        printf("Failed to encode and write stop broadcasts message\r\n");
    }
```

STEP 5: SET CONFIGURATION (OPTIONAL)

```
    // Set INS output Euler rotation in radians to 90 degrees roll for mounting
    float rotation[3] = { 90.0f*C_DEG2RAD_F, 0.0f, 0.0f };
    int messageSize = is_comm_set_data(comm, _DID_FLASH_CONFIG, offsetof(nvm_flash_cfg_t, insRotation), sizeof(float) * 3, rotation);
    if (messageSize != serialPortWrite(serialPort, comm->buf.start, messageSize))
    {
        printf("Failed to encode and write set INS rotation\r\n");
    }
```

STEP 6: ENABLE MESSAGE BROADCASTING

```
    // Ask for INS message w/ update 40ms period (4ms source period x 10).  Set data rate to zero to disable broadcast and pull a single packet.
    int messageSize = is_comm_get_data(comm, _DID_INS_LLA_EULER_NED, 0, 0, 10);
    if (messageSize != serialPortWrite(serialPort, comm->buf.start, messageSize))
    {
        printf("Failed to encode and write get INS message\r\n");
    }

    // Ask for GPS message at period of 200ms (200ms source period x 1).  Offset and size can be left at 0 unless you want to just pull a specific field from a
    messageSize = is_comm_get_data(comm, _DID_GPS1_POS, 0, 0, 1);
    if (messageSize != serialPortWrite(serialPort, comm->buf.start, messageSize))
    {
        printf("Failed to encode and write get GPS message\r\n");
    }

    // Ask for IMU message at period of 100ms (1ms source period x 100).  This could be as high as 1000 times a second (period multiple of 1)
    messageSize = is_comm_get_data(comm, _DID_IMU_DUAL, 0, 0, 100);
    if (messageSize != serialPortWrite(serialPort, comm->buf.start, messageSize))
    {
        printf("Failed to encode and write get IMU message\r\n");
    }
```

STEP 7: SAVE PERSISTENT MESSAGES

(OPTIONAL) Save currently enabled streams as persistent messages enabled after reboot.

```
    system_command_t cfg;
    cfg.command = SYS_CMD_SAVE_PERSISTENT_MESSAGES;
    cfg.invCommand = ~cfg.command;

    int messageSize = is_comm_set_data(comm, DID_SYS_CMD, 0, sizeof(system_command_t), &cfg);
    if (messageSize != serialPortWrite(serialPort, comm->buf.start, messageSize))
    {
        printf("Failed to write save persistent message\r\n");
    }
```

STEP 8: HANDLE RECEIVED DATA

```
    int count;
    uint8_t inByte;

    // You can set running to false with some other piece of code to break out of the loop and end the program
    while (running)
    {
        // Read one byte with a 20 millisecond timeout
        while ((count = serialPortReadCharTimeout(&serialPort, &inByte, 20)) > 0)
        {
            switch (is_comm_parse_byte(&comm, inByte))
            {
            case _PTYPE_INERTIAL_SENSE_DATA:
                switch (comm.dataHdr.id)
                {
                case _DID_INS_LLA_EULER_NED:
                    handleIns1Message((ins_1_t*)comm.dataPtr);
                    break;

                case DID_INS_2:
                    handleIns2Message((ins_2_t*)comm.dataPtr);
                    break;

                case _DID_GPS1_POS:
                    handleGpsMessage((gps_pos_t*)comm.dataPtr);
                    break;

                case _DID_IMU_DUAL:
                    handleImuMessage((dual_imu_t*)comm.dataPtr);
                    break;

                // TODO: add other cases for other data ids that you care about
                }
                break;

            default:
                break;
            }
        }
    }
```

Compile & Run (Linux/Mac)

1. Create build directory

```
$ cd InertialSenseSDK/ExampleProjects/Communications
$ mkdir build
```

2. Run cmake from within build directory

```
$ cd build
$ cmake ..
```

3. Compile using make

```
$ make
```

4. If necessary, add current user to the "dialout" group in order to read and write to the USB serial communication ports:

```
$ sudo usermod -a -G dialout $USER
$ sudo usermod -a -G plugdev $USER
(reboot computer)
```

5. Run executable

```
$ ./ISCommunicationsExample /dev/ttyUSB0
```

Compile & Run (Windows MS Visual Studio)

1. Install and Configure Visual Studio
2. Open Visual Studio solution file (InertialSenseSDK\ExampleProjects\Communications\VS_project\ISCommunicationsExample.sln)
3. Build (F7)
4. Run executable

```
C:\InertialSenseSDK\ExampleProjects\Communications\VS_project\Release\ISCommunicationsExample.exe COM3
```

Summary

This section has covered the basic functionality you need to set up and communicate with Inertial Sense products. If this doesn't cover everything you need, feel free to reach out to us on the Inertial Sense SDK GitHub repository, and we will be happy to help.

11.2.2 ASCII Communications Example Project

This IS Communications Example project demonstrates binary communications with the Inertial Sense Products (uINS, uAHRS, and uIMU) using the Inertial Sense SDK. See the ASCII protocol section for details on the ASCII packet structures.

Files

Project Files

- ISAsciiExample.c

SDK Files

- data_sets.c
- data_sets.h
- ISComm.c
- ISComm.h
- ISConstants.h
- serialPort.c
- serialPort.h
- serialPortPlatform.c
- serialPortPlatform.h

Implementation

STEP 1: ADD INCLUDES

```
// Change these include paths to the correct paths for the project
#include "../../src/ISComm.h"
#include "../../src/serialPortPlatform.h"
```

STEP 2: INITIALIZE AND OPEN SERIAL PORT

```
    serial_port_t serialPort;

    // Initialize the serial port (Windows, MAC or Linux) - if using an embedded system like Arduino,
    //  you will need to handle the serial port creation, open and reads yourself. In this
    //  case, you do not need to include serialPort.h/.c and serialPortPlatform.h/.c in your project.
    serialPortPlatformInit(&serialPort);

    // Open serial, last parameter is a 1 which means a blocking read, you can set as 0 for non-blocking
    // you can change the baudrate to a supported baud rate (IS_BAUDRATE_*), make sure to reboot the uINS
    //  if you are changing baud rates, you only need to do this when you are changing baud rates.
    if (!serialPortOpen(&serialPort, argv[1], IS_BAUDRATE_921600, 1))
    {
        printf("Failed to open serial port on com port %s\r\n", argv[1]);
    }
```

STEP 3: DISABLE PRIOR MESSAGE BROADCASTING

```
// Stop all broadcasts on the device on all ports.  We don't want binary message coming through while we are doing ASCII
if (!serialPortWriteAscii(&serialPort, "STPB", 4))
{
    printf("Failed to encode stop broadcasts message\r\n");
}
```

STEP 4: ENABLE MESSAGE BROADCASTING

```
    // ASCII protocol is based on NMEA protocol https://en.wikipedia.org/wiki/NMEA_0183
    // turn on the INS message at a period of 100 milliseconds (10 hz)
    // serialPortWriteAscii takes care of the leading $ character, checksum and ending \r\n newline
    // ASCB message enables ASCII broadcasts
    // ASCB fields: 1:options, 2:PIMU, 3:PPIMU, 4:PINS1, 5:PINS2, 6:PGPSP, 7:reserved, 8:GPGGA, 9:GPGLL, 10:GPGSA, 11:GPRMC
    // options can be 0 for current serial port, 1 for serial 0, 2 for serial 1 or 3 for both serial ports
    // Instead of a 0 for a message, it can be left blank (,,) to not modify the period for that message
    // please see the user manual for additional updates and notes

    // Get PINS1 @ 10Hz on the connected serial port, leave all other broadcasts the same, and save persistent messages.
    const char* asciiMessage = "ASCB,512,,,1000,,,,,,,,,";

    // Get PINS1 @ 50Hz and PGPSP @ 5Hz on the connected serial port, leave all other broadcasts the same
    // const char* asciiMessage = "ASCB,,,,20,,200,,,,,,,";

    // Get PIMU @ 50Hz, GPGGA @ 5Hz, both serial ports, set all other periods to 0
    //  const char* asciiMessage = "ASCB,3,20,0,0,0,0,0,100,0,0,0,0,0";

    if (!serialPortWriteAscii(&serialPort, asciiMessage, (int)strnlen(asciiMessage, 128)))
    {
        printf("Failed to encode ASCII get INS message\r\n");
    }
```

STEP 5: SAVE PERSISTENT MESSAGES

(OPTIONAL) This remembers the current communications and automatically streams data following reboot.

```
if (!serialPortWriteAscii(&serialPort, "PERS", 4))
{
    printf("Failed to encode ASCII save persistent message\r\n");
}
```

STEP 6: HANDLE RECEIVED DATA

```
    // STEP 4: Handle received data
    unsigned char* asciiData;
    unsigned char asciiLine[512];

    // you can set running to false with some other piece of code to break out of the loop and end the program
    while (running)
    {
        if (serialPortReadAscii(&serialPort, asciiLine, sizeof(asciiLine), &asciiData) > 0)
        {
            printf("%s\n", asciiData);
        }
    }
```

Compile & Run (Linux/Mac)

1. Create build directory

```
$ cd InertialSenseSDK/ExampleProjects/Ascii
$ mkdir build
```

2. Run cmake from within build directory

```
$ cd build
$ cmake ..
```

3. Compile using make

```
$ make
```

4. If necessary, add current user to the "dialout" group in order to read and write to the USB serial communication ports:

```
$ sudo usermod -a -G dialout $USER
$ sudo usermod -a -G plugdev $USER
(reboot computer)
```

5. Run executable

```
$ ./ISAsciiExample /dev/ttyUSB0
```

Compile & Run (Windows MS Visual Studio)

1. Install and Configure Visual Studio
2. Open Visual Studio solution file (InertialSenseSDK\ExampleProjects\Ascii\VS_project\ISAsciiExample.sln)
3. Build (F7)
4. Run executable

```
C:\InertialSenseSDK\ExampleProjects\Ascii\VS_project\Release\ISAsciiExample.exe COM3
```

Summary

This section has covered the basic functionality you need to set up and communicate with Inertial Sense products. If this doesn't cover everything you need, feel free to reach out to us on the Inertial Sense SDK GitHub repository, and we will be happy to help.

11.2.3 Basic Arduino Communications Example Project

Interfacing with the uINS over serial

This example shows how to communicate with the uINS using the Inertial Sense Binary Communications Protocol. The example code can be found in the Inertial Sense SDK/ExampleProjects/Arduino.

Important

Update the uINS to the latest firmware

This example demonstrates how to use the Inertial Sense EVB with an Arduino Due. The Due was selected because it has two serial ports. This way the Arduino can communicate with the uINS using one of the ports, and write the output over the Serial Monitor to the computer using the other.

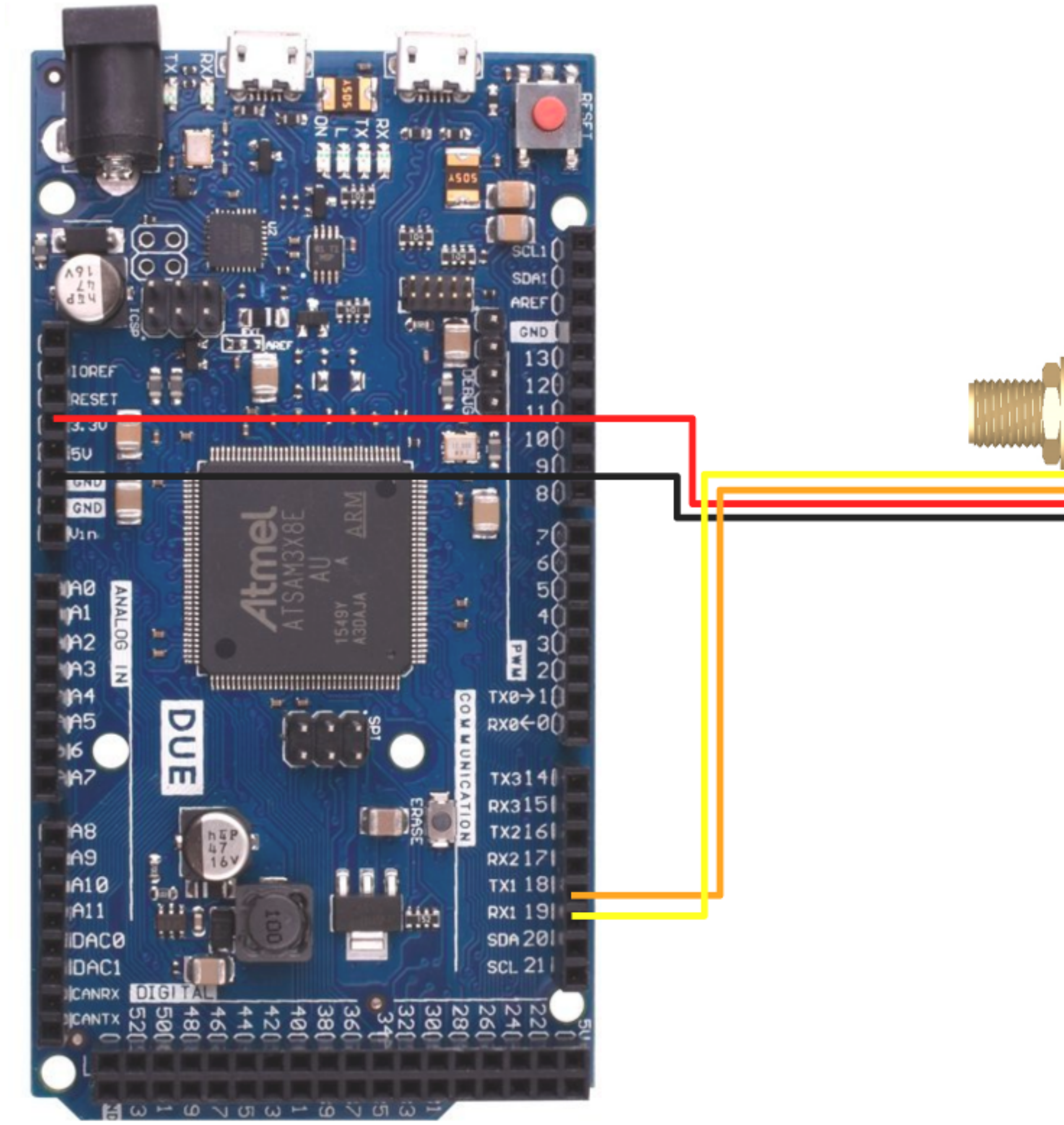
Warning

The InertialSense SDK requires 64-bit double support. 32-bit processors (Arduino Due, Zero, and M0) are supported. 8-bit processors (i.e. Arduino Mega and Uno) are NOT supported. The ASCII protocol (not covered in this example) may be used on an 8-bit Arduino.

Note

A Raspberry PI (similar in price to the Arduino) is a good alternative to the Arduino. Either the Binary Communications and ASCII Communications example projects can be run on a Raspberry PI.

Wiring Guide



After downloading the Inertial Sense SDK, Navigate to ExampleProjects/Arduino/ReadIS. Use the ImportSdkFiles.bat (Windows) or ImportSdkFiles.sh (Linux) to copy the required files from the SDK into src/ISsdk directory. The resulting file structure for the ReadIS Arduino sketch should look like the following:

```
|-ReadIS
  | - ImportSdkFiles.bat
  | - ImportSdkFiles.sh
  | - ReadIS.ino
  | - src
    | - ISsdk
      | - data_sets.c
      | - data_sets.h
      | - ISComm.c
      | - ISComm.h
      | - ISConstants.h
```

What is an ino file?

An `.ino` file is the arduino extension for a sketch. It is actually C++ code.

Note

Note that there are two `.c` files in the tree. You'll need to make sure that these files are compiled by the toolchain, otherwise `xxxx is not defined` errors can occur.

SDK Implementation

`ReadIS.ino` file explained:

STEP 1: ADD INCLUDES

The `"ISComm.h"` header file includes all the other required code. `stddef.h` file from the standard library is required for the `offsetof` function.

```
#include "src/ISsdk/ISComm.h"
#include <stddef.h>
```

STEP 2: CREATE BUFFERS

Next, define a buffer to hold data. As the uINS sends data, this buffer is used to hold the data until a full message arrives. This buffer only needs to be as big as the largest message expected, multiplied by two + 32 (worst case scenario if there is a bad transmission). For this example a 1KB buffer is used.

```
// This buffer is going to be used to hold messages as they come in.
// You can make this 512 size if memory is tight.
static uint8_t s_buffer[1024];
// create an instance to hold communications state
static is_comm_instance_t comm;
```

STEP 3: SERIAL PORT INITIALIZATION

```
void setup()
{
    // Initialize both serial ports:
    Serial.begin(115200);
    Serial1.begin(115200);

    if (sizeof(double) != 8)
    {
        Serial.println("Inertial Sense SDK requires 64 bit double support");
        while (true)
        {
        };
    }

    Serial.println("initializing");

    // Initialize comm interface - call this before doing any comm functions
    is_comm_init(&comm, s_buffer, sizeof(s_buffer));

    // Stop all the broadcasts on the device
    int messageSize = is_comm_stop_broadcasts_all_ports(&comm);
    Serial1.write(comm.buf.start, messageSize); // Transmit the message to the inertialsense device

    // Ask for ins_1 message 20 times per second.  Ask for the whole thing, so
    // set 0's for the offset and size
    messageSize = is_comm_get_data(&comm, DID_INS_1, 0, sizeof(ins_1_t), 1000);
    Serial1.write(comm.buf.start, messageSize); // Transmit the message to the inertialsense device
}
```

Initialize the communication using the following steps as shown above:

1. Initialize the serial ports
2. Tell the communication interface where to find the buffer to use to hold messages, and how big that buffer is.
3. Reset communications on the device
4. Perform configuration of the uINS
5. Tell the uINS what data to stream, and how often

Whenever sending a command to the uINS, the command is put into the buffer, and the length of the message is returned by one of the configuration functions. That buffer needs to be written out to the uINS for the command to be received.

Tip

M M M M M `data_sets.h` such as `SYS_CFG_BITS_RTK_ROVER` to configure the device. This aids code readability and reduces the chance for errors.

In this example, the `DID_INS_1` message is streamed. All available messages can be found in the `data_sets.h` file, defined as C-style structs.

STEP 4: HANDLE RECEIVED DATA

```
void loop()
{
    // Read from port 1, and see if we have a complete inertialsense packet
    if (Serial1.available())
    {
        uint8_t inByte = Serial1.read();
        // This function returns the DID of the message that was just parsed, we can then point the buffer to
        // the right function to handle the message.  We can use a cast to interpret the s_buffer as the
        // kind of message that we received.
        uint32_t message_type = is_comm_parse_byte(&comm, inByte);
        switch (message_type)
        {
        case _PTYPE_INERTIAL_SENSE_DATA:
            switch (comm.dataHdr.id)
            {
            case DID_NULL:
                break;
            case DID_INS_1:
                handleINSMessage((ins_1_t *)(comm.dataPtr));
                break;
            default:
                Serial.print("Got an unexpected message DID: ");
                Serial.println(message_type, DEC);
            }
        }
    }
}
```

In this code, every byte that we receive from the uINS is passed to the `is_comm_parse` function. For each byte received, this function waits for a complete message in the buffer and decodes it. Once a full message is received, it identifies what kind of message is in the buffer so it can be handled correctly. The easiest way to deal with this is to us a case structure as shown above, with separate "callback" functions for each message type.

The INS message handler is just printing the position in lla, velocity and euler angle attitude to the screen. Other parameterizations of position and attitude are available in other `DID_INS_x` messages.

```
static void handleINSMessage(ins_1_t *ins)
{
    Serial.print("Lat: ");
    Serial.print((float)ins->lla[0], 6);
    Serial.print("\t");
    Serial.print(", Lon: ");
    Serial.print((float)ins->lla[1], 6);
    Serial.print("\t");
    Serial.print(", Alt: ");
    Serial.print((float)ins->lla[2], 2);
    Serial.print("\t");
    Serial.print(", roll: ");
    Serial.print(ins->theta[0] * C_RAD2DEG_F);
    Serial.print("\t");
    Serial.print(", pitch: ");
    Serial.print(ins->theta[1] * C_RAD2DEG_F);
    Serial.print("\t");
    Serial.print(", yaw: ");
    Serial.print("\t");
    Serial.println(ins->theta[2] * C_RAD2DEG_F);
}
```

11.2.4 Firmware Update (Bootloader) Example Project

This ISBootloaderExample project demonstrates firmware update with the InertialSense products (uINS, uAHRS, and uIMU) using the Inertial Sense SDK.

Files

Project Files

- ISBootloaderExample.c

SDK Files

- data_sets.c
- data_sets.h
- inertialSenseBootLoader.c
- inertialSenseBootLoader.h
- ISComm.c
- ISComm.h
- serialPort.c
- serialPort.h
- serialPortPlatform.c
- serialPortPlatform.h

Implementation

STEP 1: ADD INCLUDES

```
// Change these include paths to the correct paths for your project
#include "../../src/ISComm.h"
#include "../../src/serialPortPlatform.h"
#include "../../src/inertialSenseBootLoader.h"
```

STEP 2: INITIALIZE AND OPEN SERIAL PORT

```
    serial_port_t serialPort;

    // initialize the serial port (Windows, MAC or Linux) - if using an embedded system like Arduino,
    //  you will need to either bootload from Windows, MAC or Linux, or implement your own code that
    //  implements all the function pointers on the serial_port_t struct.
    serialPortPlatformInit(&serialPort);

    // set the port - the bootloader uses this to open the port and enable bootload mode, etc.
    serialPortSetPort(&serialPort, argv[1]);
```

STEP 3: SET BOOTLOADER PARAMETERS

```
    // bootloader parameters
    bootload_params_t param;

    // very important - initialize the bootloader params to zeros
    memset(&param, 0, sizeof(param));

    // the serial port
    param.port = &serialPort;
    param.baudRate = atoi(argv[2]);

    // the file to bootload, *.hex
    param.fileName = argv[3];

    // optional - bootloader file, *.bin
    param.forceBootloaderUpdate = 0;    //do not force update of bootloader
    if (argc == 5)
        param.bootName = argv[4];
    else
        param.bootName = 0;
```

STEP 4: RUN BOOTLOADER

```
    if (bootloadFileEx(&param)==0)
    {
        printf("Bootloader success on port %s with file %s\n", serialPort.port, param.fileName);
        return 0;
    }
    else
    {
        printf("Bootloader failed! Error: %s\n", errorBuffer);
        return -1;
    }
```

Compile & Run (Linux/Mac)

1. Create build directory

```
$ cd InertialSenseSDK/ExampleProjects/Bootloader
$ mkdir build
```

2. Run cmake from within build directory

```
$ cd build
$ cmake ..
```

3. Compile using make

```
$ make
```

4. If necessary, add current user to the "dialout" group in order to read and write to the USB serial communication ports:

```
$ sudo usermod -a -G dialout $USER
$ sudo usermod -a -G plugdev $USER
(reboot computer)
```

5. Run executable

```
$ ./ISBootloaderExample /dev/ttyUSB0 IS_uINS-3.hex SAMx70-Bootloader.bin
```

Compile & Run (Windows MS Visual Studio)

1. Install and Configure Visual Studio
2. Open Visual Studio solution file (InertialSenseSDK\ExampleProjects\Bootloader\VS_project\ISBootloaderExample.sln)
3. Build (F7)
4. Run executable

```
C:\InertialSenseSDK\ExampleProjects\Bootloader\VS_project\Release\ISBootloaderExample.exe COM3 IS_uINS-3.hex SAMx70-Bootloader.bin
```

Summary

This section has covered the basic functionality you need to set up and communicate with Inertial Sense products. If this doesn't cover everything you need, feel free to reach out to us on the Inertial Sense SDK GitHub repository, and we will be happy to help.

11.2.5 C++ API - Inertial Sense Class and CLTool Example Project

The InertialSense C++ class, defined in InertialSense.h/.cpp, provides all SDK capabilities including serial communications, data logging to file, and embedded firmware update for InertialSense products.

CLTool Example

The Command Line Tool (CLTool) is an open source project designed to illustrate InertialSense C++ class implementation. The CLTool project can be compiled on most operating systems using cmake and gcc and can be used to communicate, log data, and update firmware for Inertial Sense products. A Visual Studio project for Windows is also included. See Using CLTool for details on compiling and running the CLTool.

IMPLEMENTATION KEYWORDS

The following keywords are found in the CLTool soure code identify the steps for InertialSense class implementation.

```
/* SDK Implementation Keywords:
 * [C++ COMM INSTRUCTION] - C++ binding API, InertialSense class with binary
 * communication protocol and serial port support for Linux and Windows.
 * [LOGGER INSTRUCTION] - Data logger.
 * [BOOTLOADER INSTRUCTION] - Firmware update feature.
 */
```

Serial Communications

STEP 1: INSTANTIATE INERTIALSENSE CLASS

Include the InertialSense header file. Create InertialSense object.

```
#include "InertialSense.h"

// [C++ COMM INSTRUCTION] 1.) Create InertialSense object, passing in data callback function pointer.
InertialSense inertialSenseInterface(cltool_dataCallback);
```

STEP 2: OPEN SERIAL PORT

Open the serial by specifying the com port number, buadrate, and and The serial port used for communications

```
if (!inertialSenseInterface.Open(g_commandLineOptions.comPort.c_str(),
   g_commandLineOptions.baudRate,
   g_commandLineOptions.disableBroadcastsOnClose))
{
    cout << "Failed to open serial port at " << g_commandLineOptions.comPort.c_str() << endl;
    return -1; // Failed to open serial port
}
```

STEP 3: ENABLE DATA BROADCASTING

The following enables data broadcasting from the uINS at a specified data rate or period in milliseconds.

```
cltool_setupCommunications(inertialSenseInterface)
```

STEP 4: READ DATA

Call the Update() method at regular intervals to send and receive data.

```
// Main loop. Could be in separate thread if desired.
while (!g_inertialSenseDisplay.ControlCWasPressed())
{
    if (!inertialSenseInterface.Update())
    {
        // device disconnected, exit
        break;
    }
}
```

STEP 5: HANDLE RECEIVED DATA

New data is available in the data callback function.

```
static void cltool_dataCallback(InertialSense* i, p_data_t* data, int pHandle)
{
    // Print data to terminal
    g_inertialSenseDisplay.ProcessData(data);

    // uDatasets is a union of all datasets that we can receive. See data_sets.h for a full list of all available datasets.
    uDatasets d = {};
    copyDataPToStructP(&d, data, sizeof(uDatasets));

    // Example of how to access dataset fields.
    switch (data->hdr.id)
    {
    case DID_INS_2:
        d.ins2.qn2b; // quaternion attitude
        d.ins2.uvw; // body velocities
        d.ins2.lla; // latitude, longitude, altitude
        break;
    case DID_INS_1:
        d.ins1.theta; // euler attitude
        d.ins1.lla; // latitude, longitude, altitude
    break;
    case DID_DUAL_IMU: d.dualImu; break;
    case DID_DELTA_THETA_VEL: d.dThetaVel; break;
    case DID_IMU_1: d.imu; break;
    case DID_IMU_2: d.imu; break;
    case DID_GPS: d.gps; break;
    case DID_MAGNETOMETER: d.mag; break;
    case DID_BAROMETER: d.baro; break;
    case DID_SYS_SENSORS: d.sysSensors; break;
    }
}
```

STEP 6: CLOSE INTERFACE

Close the interface when your application finishes.

```
// Close cleanly to ensure serial port and logging are shutdown properly. (optional)
inertialSenseInterface.Close();
```

Data Logging

STEP 1: CONFIGURE AND START LOGGING

```
// [LOGGER INSTRUCTION] Setup and start data logger
if (!cltool_setupLogger(inertialSenseInterface))
{
    cout << "Failed to setup logger!" << endl;
    return -1;
}
```

Compile & Run (Linux/Mac)

1. Create build directory

```
$ cd cltool
$ mkdir build
```

2. Run cmake from within build directory

```
$ cd build
$ cmake ..
```

3. Compile using make

```
$ make
```

4. If necessary, add current user to the "dialout" group in order to read and write to the USB serial communication ports:

```
$ sudo usermod -a -G dialout $USER
$ sudo usermod -a -G plugdev $USER
(reboot computer)
```

5. Run executable

```
$ ./cltool
```

Compile & Run (Windows MS Visual Studio)

1. Install and Configure Visual Studio
2. Open Visual Studio solution file (InertialSenseSDK/cltool/VS_project/cltool.sln)
3. Build (F7)
4. Run executable

```
C:\InertialSenseSDK\cltool\VS_project\Release\cltool.exe
```

Summary

This section has covered the basic functionality you need to set up and communicate with Inertial Sense products. If this doesn't cover everything you need, feel free to reach out to us on the Inertial Sense SDK GitHub repository, and we will be happy to help.

11.2.6 Data Logging Example Project

This ISLoggerExample project demonstrates data logging with the InertialSense products (uINS, uAHRS, and uIMU) using the Inertial Sense SDK.

Files

Project Files

- ISLoggerExample.cpp

SDK Files

- SDK

Implementation

STEP 1: ADD INCLUDES

```
// Change these include paths to the correct paths for your project
#include "../../src/InertialSense.h"
```

STEP 2: INSTANTIATE INERTIALSENSE CLASS

```
    // InertialSense class wraps communications and logging in a convenient, easy to use class
    InertialSense inertialSense(dataCallback);
    if (!inertialSense.Open(argv[1]))
    {
        std::cout << "Failed to open com port at " << argv[1] << std::endl;
    }
```

STEP 3: ENABLE DATA LOGGER

```
    // get log type from command line
    cISLogger::eLogType logType = (argc < 3 ? cISLogger::eLogType::LOGTYPE_DAT : cISLogger::ParseLogType(argv[2]));
    inertialSense.SetLoggerEnabled(true, "", logType);
```

STEP 4: ENABLE DATA BROADCASTING

```
    // broadcast the standard set of post processing messages (ins, imu, etc.)
    inertialSense.BroadcastBinaryDataRmcPreset();

    // instead of the rmc preset (real-time message controller) you can request individual messages...
    // inertialSense.BroadcastBinaryData(DID_DUAL_IMU, 10); // imu every 10 milliseconds (100 hz)
```

By default, data logs will be stored in the "IS_logs" directory in the current directory.

```
build/IS_logs/LOG_SN30664_20180323_112822_0001.dat
```

Compile & Run (Linux/Mac)

1. Create build directory

```
$ cd InertialSenseSDK/ExampleProjects/Logger
$ mkdir build
```

2. Run cmake from within build directory

```
$ cd build
$ cmake ..
```

3. Compile using make

```
$ make
```

4. If necessary, add current user to the "dialout" group in order to read and write to the USB serial communication ports:

```
$ sudo usermod -a -G dialout $USER
$ sudo usermod -a -G plugdev $USER
(reboot computer)
```

5. Run executable

```
$ ./ISLoggerExample /dev/ttyUSB0
```

Compile & Run (Windows MS Visual Studio)

1. Install and Configure Visual Studio
2. Open Visual Studio solution file (InertialSenseSDK\ExampleProjects\Logger\VS_project\InertialSenseCLTool.sln)
3. Build (F7)
4. Run executable

```
C:\InertialSenseSDK\ExampleProjects\Logger\VS_project\Release\ISLoggerExample.exe COM3
```

Summary

This section has covered the basic functionality you need to set up and communicate with Inertial Sense products. If this doesn't cover everything you need, feel free to reach out to us on the Inertial Sense SDK GitHub repository, and we will be happy to help.

12. Data Logging/Plotting

12.1 Data Logging/Plotting

Inertial Sense provides data a logging capability in the EvalTool, CLTool, and SDK (C++) that can record data in binary, comma separated (.CSV), and KML file formats. This logging capability is useful for storing, replaying, and analyzing data.

12.1.1 Data Log Types

Comma Seperated Values (`*.csv`)

The comma separated value (.csv) file format can be imported into many software packages, including Excel, Matlab, and Python.

KML (`*.kml`)

KML is a file format used to display geographic data in an Earth browser such as Google Earth.

Binary Data Log Types (`*.dat` and `*.sdat`)

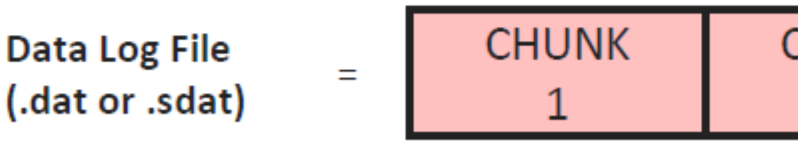
	Serial Logger (*.dat)	Sorted Logger (*.sdat)
Description	Stores data to file in the same serial order it was passed into the logger. This is the default logger used in the CLTool and EvalTool.	Sorts data of similar types into separate chunks, allowing for faster load times into analysis tools.
Advantages	Optimized for real-time data logging.	Optimized for loading data into analysis tools (i.e. Matlab, Python).
Source File	DeviceLogSerial.h / .cpp	DeviceLogSorted.h / .cpp
File extension	.dat	.sdat

12.1.2 Binary Data Log Format

This section outlines the Inertial Sense binary data log types known as serial data and sorted data (`.dat` and `.sdat` file extensions). Both data log file types are composed of several data containers know as chunks. Each chunk contains a header, sub-header, and data.

File

The data log file name has the format *LOG_SNXXXXX_YYYYMMDD_HHMMSS_CNT.dat* which contains the device serial number, date, time, and log file count. The two primary log file formats are `.dat` and `.sdat`. These log files consist of a series of data Chunks.

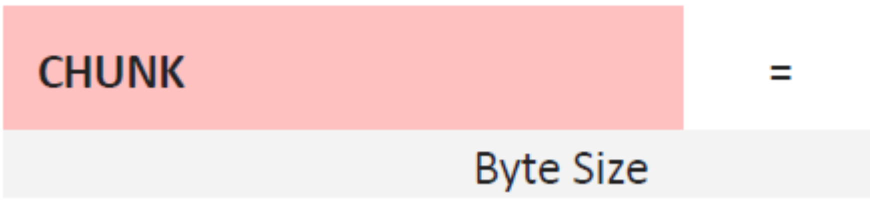


Standard data types are stored in the log files and are defined as:

U32	unsigned int
U16	unsigned short
S8	char
U8	unsigned char

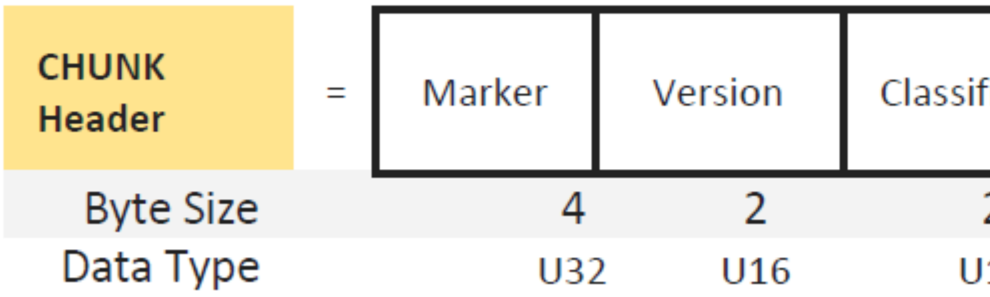
Chunk

The data log file is composed of Chunks. A Chunk is a data container that provides an efficient method for organizing, handling, and parsing data in a file. A Chunk starts with a header which has a unique identifiable marker and ends with the data to be stored.



Chunk Header

The header, found at the start of each Chunk, is as follows:

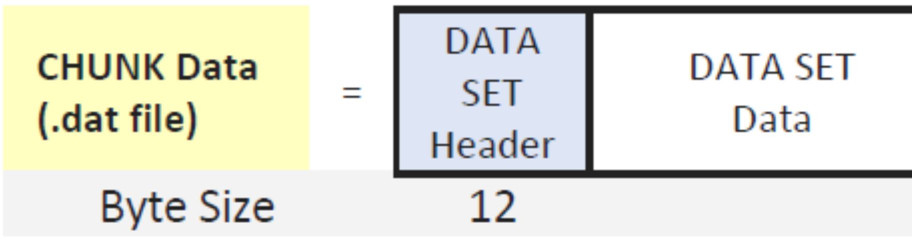


The C structure implementation of the Chunk header is:

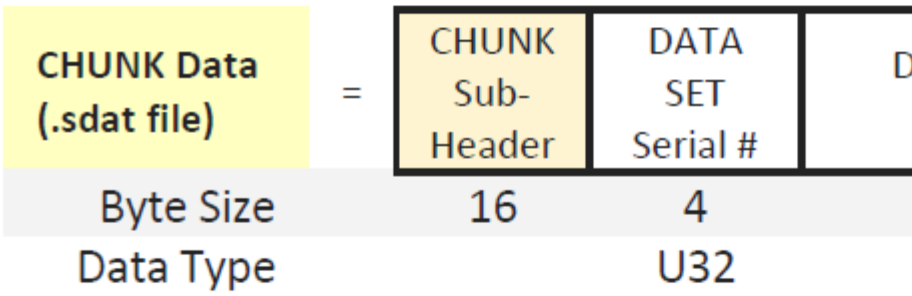
```
//!< Chunk Header
#pragma pack(push,1)
struct sChunkHeader
{
uint32_t marker; //!< Chunk marker (0xFC05EA32)
uint16_t version; //!< Chunk Version
uint16_t classification; //!< Chunk classification
char name[4]; //!< Chunk name
char invName[4]; //!< Bitwise inverse of chunk name
uint32_t dataSize; //!< Chunk data length in bytes
uint32_t invDataSize; //!< Bitwise inverse of chunk data length
uint32_t grpNum; //!< Chunk Group Number: 0 = serial data, 1 = sorted data...
uint32_t devSerialNum; //!< Device serial number
uint32_t pHandle; //!< Device port handle
uint32_t reserved; //!< Unused
};
#pragma pack(pop)
```

Chunk Data

The Chunk data is defined for both the `.dat` and `.sdat` file types.



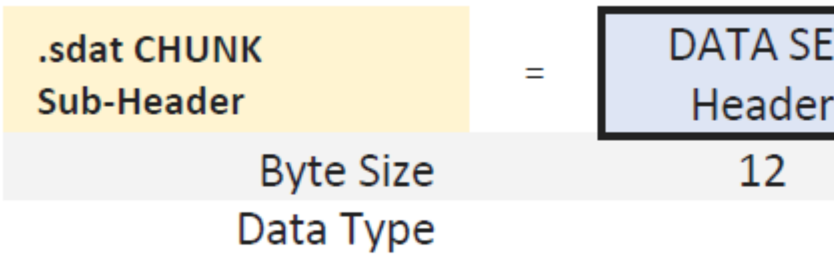
CHUNK Data (.dat file)	=	DATA SET Header	DATA SET Data
Byte Size		12	



CHUNK Data (.sdat file)	=	CHUNK Sub-Header	DATA SET Serial #	D
Byte Size		16	4	
Data Type			U32	

Chunk Sub-Header

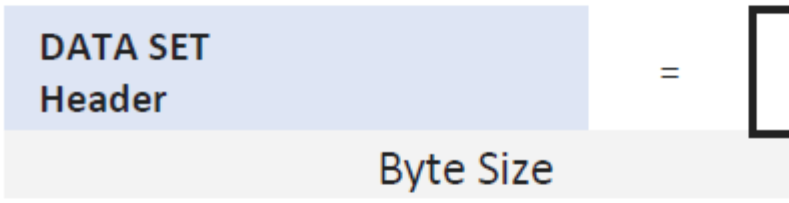
The Chunk sub-header is used for `.sdat` file types.



.sdat CHUNK Sub-Header	=	DATA SE Header
Byte Size		12
Data Type		

Data Set Header

The Data set header is used for both the `.dat` and `.sdat` file types.



DATA SET Header	=	
Byte Size		

12.2 Logging

The SDK logging interface is defined in SDK/src/ISLogger.h. Data logs can be converted between file formats using the Inertial Sense data logger. The logging interface is used in the Inertial Sense software described below.

12.2.1 Logging using Inertial Sense software

EvalTool

1. Go to **“Data Logs”** tab in EvalTool.
2. Select the **“Format”** file type from the drop-down menu.
3. Select the data to record within the Data Streams section of the **"Data Logs"** tab:
    1. **Manual Selection** – Allows the user to select the specific datasets to stream and their update rates by setting the checkbox and period multiple in Manual Selection table.
    2. **INS** – Log INS output (attitude, velocity, position) at 100 Hz by selecting "**INS**" from the RMC Presets dropdown.
    3. **Post Process** – Used for beta testing and internal testing. Includes IMU, GPS, INS and other messages. Log by selecting "**PPD**" from the RMC Presets dropdown.
4. Press **“Enable”** to begin logging data.
5. Press **“Disable”** to stop logging data.
6. The **“Open Folder”** button opens the File Explorer location to the data logs, i.e. `C:\Users\[username]\Documents\Inertial_Sense\logs`.
7. To change the root log folder in the Eval Tool, edit `Documents/Inertial Sense/settings.json`, and add or change this key: "LOG_FOLDER": "FOLDER_FOR_LOGS".

CLTool

The CLTool, provided in the SDK, is a command line application that can record post process data. The CLTool help menu is displayed using the option `-h`. See the CLTool section for more information on using the CLTool.

12.2.2 Post Process Data (PPD) Logging Instructions

Post process data (PPD) logs include both the input to and output from the navigation filter. The data is used for analyzing, troubleshooting, and improving system performance. PPD logs can be recorded using the EvalTool, CLTool, or SDK.

PPD RMC bits Preset

PPD logs are created by enabling PPD data streaming by setting the RMC bits to `RMC_PRESET_PPD_BITS` and logging this stream to a .dat binary file. `RMC_PRESET_PPD_BITS` is defined in data_sets.h.

Logging PPD in EvalTool

The following steps outline how to record post process data in the EvalTool

1. Go to the "Data Logs" tab in the EvalTool.
2. Press "**Data Streams: RMC Preset: PPD**" button to start PPD data streaming.
3. Toggle the "**Data Log: Enable**" button to start logging.
4. Toggle the "**Data Log: Disable**" button to stop logging.
5. The "**Open Folder**" button will open the directory where the data logs are stored.

Logging PPD in CLTool

Streaming and logging a PPD log using the CLTool is done using the `-presetPPD -lon` options:

```
cltool -c /dev/ttyS2 -presetPPD -lon
```

See the CLTool section for more information on using the CLTool.

12.3 Plotting

12.3.1 Log Inspector

Log Inspector is a convenient way to quickly plot Inertial Sense PPD logs that of of the .dat format. The source code is in the SDK and can be modified and expanded.

12.3.2 CLTool

The CLTool can be used to load and replay `.dat` log files. The source code for the CLTool is located in the SDK and can be expanded by a user to analyze log data.

- `-rp PATH` replay data log from specified path
- `-rs=SPEED` replay data log at x SPEED

The following example replays data at 1x speed from the specified directory.

```
cltool -rp IS_logs/20180801_222310
```

The following example will replay data as fast a possible in quiet mode (without printing to the screen). This is useful to quickly reprocess the data.

```
cltool -rp IS_logs/20180801_222310 -rs=0 -q
```

12.3.3 3rd Party Software

The various file types described in the overview section can be analyzed using various software packages. Matlab, Python, and Excel are popular choices and are well suited for Inertial Sense data logs.

13. Reference

13.1 Bootloader

The bootloader is embedded firmware stored on the uINS and is used to update the uINS application firmware. Updating the bootloader firmware is required occasionally when new functionality is required.

13.1.1 Bootloader Update

The bootloader is now checked and updated at the same time as loading new firmware. The following steps outline how to update the uINS bootloader and firmware.

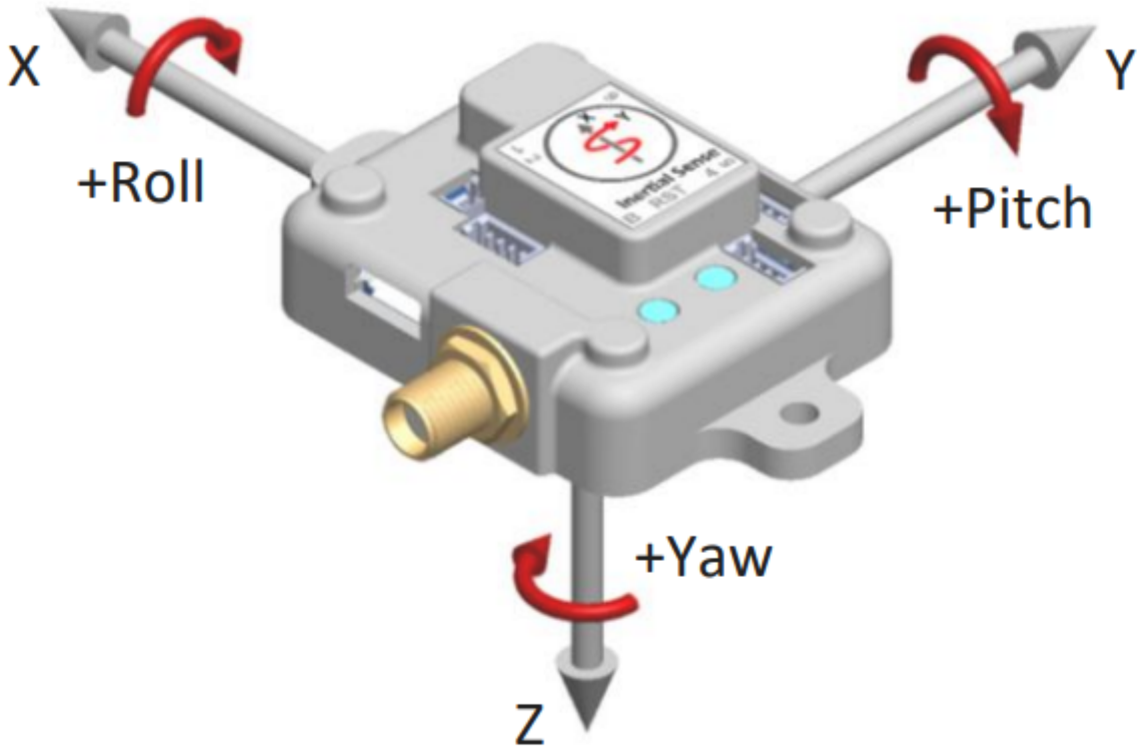
1. **Ensure uINS Firmware is Running** - *(This step is not necessary if the uINS firmware is running and the EvalTool is communicating with the uINS)*. If the bootloader is running but the firmware is not, version information will not appear in the EvalTool. The LED will also be a fading cyan.
2. **Select Baud Rate** - Select a slower baud rate (i.e. 115,200 or 230,400) for systems with known baud rate limits.
3. **Update the Bootloader and Firmware** - Use the EvalTool "Update Firmware" button in the Settings tab to upload the latest bootloader and the latest firmware. **The bootloader can only be updated using serial0 or the native USB ports.**

13.2 Coordinate Frames

In this manual, coordinate frame systems are simply referred to as frames. This page is to assist the developer in choosing and implementing the appropriate coordinate frames for their respective application. It should be noted that the following frames are in relation to the uINS itself.

13.2.1 Sensor Frame

IMU, magnetometer, and INS velocity data are in the Sensor Coordinate Frame, or Sensor Frame, and are identified by the X, Y, and Z axes labeled on the hardware. The z-axis is positive down into the image.



13.2.2 INS Output Frame

The INS output data (DID_INS_1, DID_INS_2, DID_INS_3) is in the INS Output Frame which is the same as the Sensor Frame after default rotations and offsets are entered to the Flash Configuration. Translation from Sensor Frame to INS Output Frame is defined as:

1. Sensor Frame → Intermediate Output Frame by rotation of DID_FLASH_CONFIG.insRotation euler angles (in order of heading, pitch, roll angle) In radians.
2. Intermediate Output Frame → INS Output Frame: Offset by DID_FLASH_CONFIG.insOffset in meters.

If DID_FLASH_CONFIG.insRotation and DID_FLASH_CONFIG.insOffset are zero, the Sensor Frame and the INS Output Frame are the same.

13.2.3 North-East-Down (NED) Frame

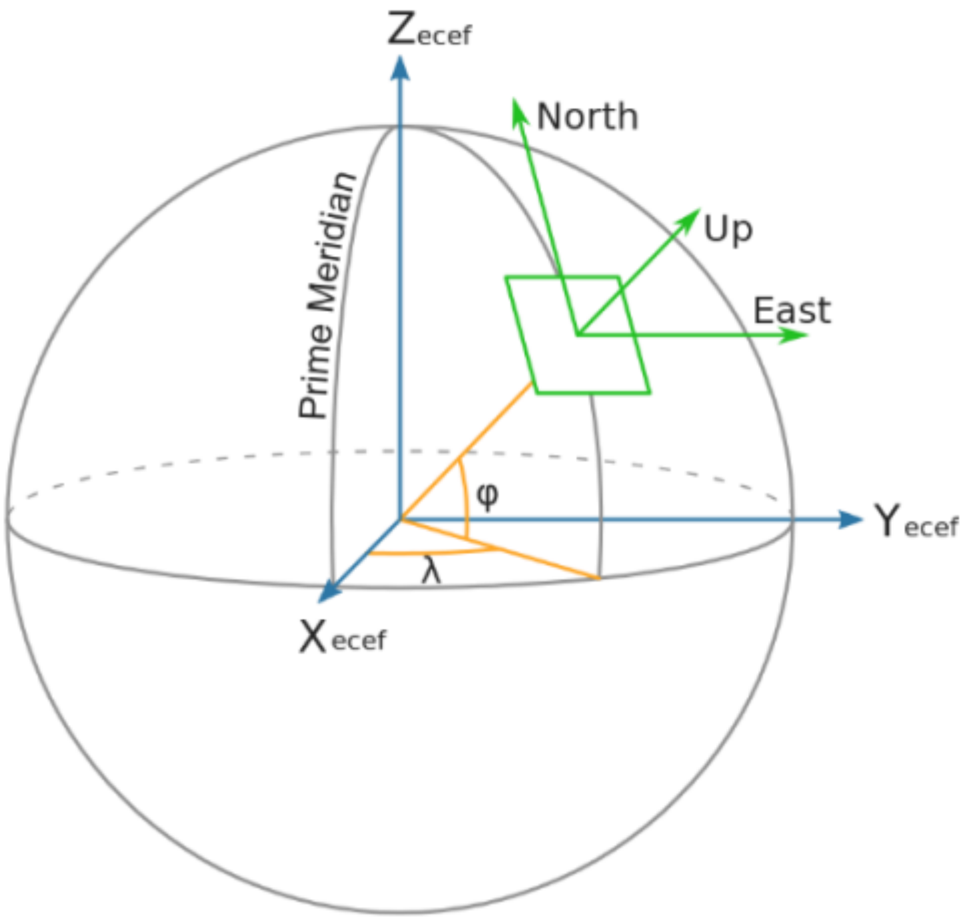
Position estimates can be output in the North-East-Down (NED) coordinate frame defined as follows:

* Right-handed, Cartesian, non-inertial, geodetic frame with origin located at the surface of Earth (WGS84 ellipsoid). * Positive X-axis points towards North, tangent to WGS84 ellipsoid. * Positive Y-axis points towards East, tangent to WGS84 ellipsoid. * Positive Z-axis points down into the ground completing the right-handed system.

13.2.4 Earth-Centered Earth-Fixed (ECEF) Frame

The Earth-Centered Earth-Fixed (ECEF) frame is defined as follows:

* Right-handed, Cartesian, non-inertial frame with origin located at the center of Earth. * Fixed to and rotates with Earth. * Positive X-axis aligns with the WGS84 X-axis, which aligns with the International Earth Rotation and Reference Systems Service (IERS) Prime Meridian. * Positive Z-axis aligns with the WGS84 Z-axis, which aligns with the IERS Reference Pole (IRP) that points towards the North Pole. * Positive Y-axis aligns with the WGS84 Y-axis, completing the right-handed system.



13.2.5 Coordinate Frames Transformation Functions

This section is intended to be an example of how to rotate between frames using utility functions defined in the InertialSenseSDK.

Body frame to NED frame

The following example converts body velocity `DID_INS_2.uvw` to NED velocity `vel_ned`.

```
#include "SDK/src/ISPose.h"
quatRot( vel_ned, DID_INS_2.qn2b, DID_INS_2.uvw );
```

This following example removes gravity from the IMU measured acceleration.

```
#include "SDK/src/ISPose.h"
Vector gravityNED = { 0, 0, -9.80665 }; // m/s^2
Vector gravityBody;
Vector accMinusGravity;
// Rotate gravity into body frame
quatConjRot( gravityBody, DID_INS_2.qn2b, gravityNED );
// Subtract gravity from IMU acceleration output
sub_Vec3_Vec3( accMinusGravity, DID_DUAL_IMU.I[0].acc, gravityBody );
```

ECEF frame to NED frame

This example converts ECEF velocity `vel_ecef` to NED velocity `vel_ned`.

```
#include "SDC/src/ISPose.h"
quat_ecef2ned( lla[0], lla[1], qe2n );
quatConj( qn2e, qe2n );
quatRot( vel_ned, qn2e, vel_ecef );
```

13.3 Definitions

13.3.1 GPS Time To Fix

The time it takes for the GPS receiver to get “fix” or produce a navigation solution from the visible satellites is affected by the following GPS startup conditions:

- Cold start - In cold start mode, the receiver has no information from the last position (e.g. time, velocity, frequency etc.) at startup. Therefore, the receiver must search the full time and frequency space, and all possible satellite numbers. If a satellite signal is found, it is tracked to decode the ephemeris (18-36 seconds under strong signal conditions), whereas the other channels continue to search satellites. Once there are enough satellites with valid ephemeris, the receiver can calculate position and velocity data. Other GNSS receiver manufacturers call this startup mode Factory Startup.
- Warm start - In warm start mode, the receiver has approximate information for time, position, and coarse satellite position data (Almanac). In this mode, after power-up, the receiver normally needs to download ephemeris before it can calculate position and velocity data. As the ephemeris data usually is outdated after 4 hours, the receiver will typically start with a Warm start if it has been powered down for more than 4 hours.
- Hot start - In hot start mode, the receiver was powered down only for a short time (4 hours or less), so that its ephemeris is still valid. Since the receiver doesn't need to download ephemeris again, this is the fastest startup method.

Battery backed-up power supplied to the uINS preserves the GPS time, position, and coarse satellite position (almanac) while off. GPS almanac data is typically valid for several weeks while the GPS is off.

13.3.2 Preintegrated IMU

Also known as Coning and Sculling Integrals, Δ Theta Δ Velocity, or Integrated IMU. For clarification, we will use the name "Preintegrated IMU" through the User Manual. They are integrated by the IMU at IMU update rates (1KHz). These integrals are reset each time they are output. Preintegrated IMU data acts as a form of compression, adding the benefit of higher integration rates for slower output data rates, preserving the IMU data without adding filter delay. It is most effective for systems that have higher dynamics and lower communications data rates.

13.3.3 IMU Bias Repeatability (Turn-on to Turn-on Bias)

The initial bias will be different for each power up of the IMU due to signal processing initial conditions and physical properties. A more repeatable bias allows for better tuning of INS parameters and faster estimate of the bias, whereas a more variable initial turn-on bias causes more difficult and longer INS convergence startup time.

13.3.4 IMU Bias Stability (In-Run Bias)

Describes the amount of bias change during any one run-time following poweron. This change is caused by temperature, time, and mechanical stress. The INS navigation filter estimates the IMU biases in order to improve the state estimate. The IMU bias stability directly impacts the accuracy of the INS output.

13.3.5 Random Walk

The IMU sensors measure a signal as well as noise or error, described as a stochastic process. During IMU integration in the INS, sensor noise is accumulated and produces a random walk or drift on the final solution. Random walk has a direct effect on the accuracy of the INS output.

13.3.6 Sensor Orthogonality (Cross-Axis Alignment Error)

The three axis gyro and accelerometer sensors found in an IMU have measurement axes at 90 degrees from each other, maximizing the observability of the system. In practice, these sensing axes in a three axis sensor are not perfectly at 90 degrees of each other, or misaligned slightly due to manufacturing imperfection. This misalignment results in integration error in the INS and impacts accuracy. To correct for cross-axis alignment error, the IMU is calibrated during manufacturing in a controlled motion environment.

13.4 Interference Considerations

Electrical interference or noise can be coupled into the Inertial Sense module in the form of electromagnetic interference (EMI) through the air or electrically conducted through wiring. Sources for interference include:

- EMI at the GPS antenna.
- EMI at the uINS module.
- EMI conducted through the power supply or I/O lines.

Common sources for noise and interference are digital lines, USB 3.x, noisy power supplies, etc.

13.4.1 Detecting Interference

To detect if interference is being coupled into the Inertial Sense sensor module, it can be compared with a stock EVB demo unit to compare noise figures. This is done by using the following steps. If both steps pass, there is no noise being coupled into the module. Optionally connect multiple sensor modules can be connected to the EvalTool in parallel to compare noise.

1. **Evaluate the IMU sensor** - Make sure the unit is stationary (on a table or non-moving surface) and not seeing any vibrations. Watch the standard deviation columns labeled "σ" in the Sensors tab of the EvalTool. This shows the noise level over the past 5 seconds, which means the device needs to be completely stable for 5 seconds to be accurate. Compare this figure between the integrated sensor module and EVB demo unit.
2. **Evaluate GPS sensitivity** – In clear view of the sky, monitor the satellite signal strength through the `DID_GPS_NAV.cnoMax` and `DID_GPS_NAV.cnoMean` fields in the EvalTool "Data Sets" tab or in the EvalTool "GPS" tab. See that the strongest (largest) CNO values are roughly the same between the integrated sensor module and the EVB demo unit.

13.4.2 Interference Mitigation

The best solution is to stop the EMI (emitted) or conducted noise at its source. If it is not possible to completely eliminate the source, the following methods should be considered depending on the cause of interference:

- **GPS antenna location** - Position the GPS antenna at the top of the vehicle, clear of obstructions and away from noise sources such as motors, processors, USB cables, digital devices, etc. USB 3.x typically generates quite a bit of EMI and interferes with the GPS. Added shielding and/or signal inline USB filters can be added USB 3.x to reduce EMI and mitigate GPS interference. Symptoms of GPS interference are poor GPS CNO signal strength, long times to lock, slip out of lock, etc.
- **GPS antenna ground plane** - Adding a 2"-3" diameter metallic ground plane below the GPS antenna will improve CNO noise ratio and improve sensitivity. This can help in noisy environments. You can use any scrap metal or PCB to test the concept by simply placing it below the GPS antenna (no electrical grounding required). The ground plane can have holes to reduce the weight and aerodynamic effects.

- **Shielding around the Inertial Sense module** - This can further prevent EMI from being absorbed by potentially GPS sensitive circuitry on the module.
- **Digital signals** - Making sure best practices for electrical current return paths for both common mode and differential mode signals can be key for this. Customers have seen GPS interference caused by USB 3.x and have have resolved the issues using shielding and inline USB filters.
- **Power supply filtering** - This may be necessary on systems with significant digital noise. LC filters or similar filters can be added inline between the power supply and the uINS supply input (Vcc). Common switching mode/buck voltage regulators should be fine for use with the uINS module and not require additional filtering.

Please contact us at support@inertialsense.com if further support is needed.

13.4.3 Magnetic Interference

Magnetic interference may impact uINS magnetometer performance if surrounded by steel or ferrous material or near motors, motor drivers, or other electronics that cause EMI. This interference can be observed in the magnetometer output, magnetometer status, and INS heading. Make sure all components are fixed in location during this test. While powering and actuating the various interference sources, observe the following:

- **Magnetometer Output** should remain constant and not deviate. (EvalTool Sensors tab)
- **Magnetometer Status** should remain good and not indicate interference. (EvalTool INS tab, "Mag used" green = good, yellow = interference)
- **INS Heading** (yaw) estimate should not drift or change direction. (EvalTool INS tab, "Yaw")

If any of these items are affected during the test, the system may result in incorrect magnetometer and heading values.

13.4.4 Mechanical Vibration

The system accuracy may degrade in the presence of mechanical vibrations that exceed 3 g of acceleration. Empirical data shows degradation at approximately 100 - 150 Hz. Adding **vibration isolation** to the mount may be necessary to reduce the vibrations seen by the product and to improve accuracy.

13.4.5 Temperature Sensitivity

The system is designed to compensate for the effects of temperature drift, which can be found in typical operation. However, rapid hardware temperature changes can result in degraded accuracy of the IMU calibration, GPS position, and INS estimate. Rapid temperature change can be caused by direct exposure to wind, sun, and other elements.

13.5 Magnetometer

The magnetometer is used to estimate heading when the system is in any of the following conditions:

1. is in AHRS mode
2. has no GPS fix
3. has GPS fix and constant velocity (non-accelerating motion)

To have accurate heading under these conditions, the magnetometer must be calibrated. The system allows for two types of modes for recalibration, external initiated and automatically initiated re-calibration. Regardless of the recalibration mode, a slower online background calibration runs that continuously improves the magnetometer calibration to handle local magnetic environment changes. All magnetometer calibration is stored in flash memory and available on bootup.

13.5.1 Disable Magnetometer Updates

Magnetometer fusion into the INS and AHRS filter can be disabled by setting bit `SYS_CFG_BITS_DISABLE_MAGNETOMETER_FUSION` (0x00001000) of `DID_FLASH_CONFIG.sysCfgBits`.

13.5.2 Magnetometer Recalibration

Occasionally the magnetometer will require a complete recalibration, replacing the old calibration with an entirely new calibration. This is accomplished either through external or automatic initiated recalibration. Use of the different modes is generally governed by the particular use case for the end customer and is intended to allow for the most flexibility in an integrated product design.

External Recalibration

External magnetometer recalibration allows the most flexibility in determining when an end user will need to recalibrate the system. This control over the timing of the recalibration is critical for many use cases and allows product designers to implement their desired workflows for customers. Further there are use cases where automatic recalibration is not possible because the quality of the magnetometer calibration is not observable. Such use cases would include AHRS operation, extended periods without motion or no GNSS fix. External magnetometer recalibration, as the name suggests is triggered by an external command from the application managing the uINS hardware. The uINS provides a set of status messages indicating the quality of the magnetometer calibration and leaves the timing and implementation of a recalibration up to the product designer. Specifically, `INS_STATUS_MAG_INTERFERENCE_OR_BAD_CAL` is an indication of the quality of the magnetometer calibration (see system status flags for details).

During the calibration process, the system should be clear of steel, iron, magnets, or other ferrous materials (i.e. steel desks, tables, building structures). The uINS should be attached to the system in which it is operating and rotated together during the calibration process. The following is the

Force magnetometer recalibration procedure:

1. Set `DID_MAG_CAL.recalCmd` to either: * `MAG_RECAL_CMD_MULTI_AXIS` (0) for Multi-Axis which is more accurate and requires 360º rotation about two different axes. * `MAG_RECAL_CMD_SINGLE_AXIS` (1) for Single-Axis which is less accurate and requires 360º rotation about one axis.
2. Rotate the system accordingly.

Recalibration completion is indicated by either:

1. `INS_STATUS_MAG_RECALIBRATING` bit of the insStatus word set to zero. The insStatus word is found in standard INS output messages (`DID_INS_1`, `DID_INS_2`, `DID_INS_3`, and `DID_INS_4`).
2. `DID_MAG_CAL.progress` is 100.

Recalibration progress is indicated as a percentage (0-100%) is indicated can be observed from variable `DID_MAG_CAL.progress`. The recalibration process can be canceled and the prior calibration restored anytime by setting `DID_MAG_CAL.enMagRecal` = `MAG_RECAL_MODE_ABORT` (101).

The “Mag used” indicator in the EvalTool INS tab will be green when magnetometer data is being fused into the solution, black when not being fused into the solution, and red during recalibrating.

Example code:

```
#include "com_manager.h"
// Set DID_MAG_CAL.enMagRecal = 0 for multi-axis recalibration
int32_t value = MAG_RECAL_MODE_MULTI_AXIS;
sendDataComManager(0, DID_MAG_CAL, &value, 4, offsetof(mag_cal_t, enMagRecal));
// Enable broadcast of DID_MAG_CAL.progress every 100ms to observe the percent complete
sendDataComManager(0, DID_MAG_CAL, 0, sizeof(mag_cal_t), 100);
```

Automatic Recalibration

Automatic magnetometer recalibration is useful for systems where user intervention to start external calibration is not convenient or practical. In this mode, the solution will determine that the system needs to be recalibrated and will attempt to do so while in normal operation. In the period while the system is recalibrating, the uncalibrated magnetometer data is used to prevent the INS heading from drifting but it does not provide heading measurements to the state estimator. This feature is enabled by setting bit `SYS_CFG_BITS_AUTO_MAG_RECAL` (0x00000004) of `DID_FLASH_CONFIG.sysCfgBits` non-volatile word.

```
// Enable automatic magnetometer calibration.
DID_FLASH_CONFIG.sysCfgBits |= SYS_CFG_BITS_AUTO_MAG_RECAL;
// Disable automatic magnetometer calibration.
DID_FLASH_CONFIG.sysCfgBits &= ~SYS_CFG_BITS_AUTO_MAG_RECAL;
```

13.5.3 Magnetometer Continuous Calibration

To mitigate the need for recalibration (completely replace calibration data), continuous calibration improves the magnetometer calibration slowly over time. Continuous calibration always runs in the background.

13.5.4 Magnetometer Calibration Settings

The magnetometer calibration algorithm can produce higher quality calibrations when data more data is collected across multiple axes of rotation. However, there are use cases where data collection beyond a single axis is impractical if not impossible. To address this issue there is a setting in the flash to configure the data requirement threshold for magnetometer calibration. The available settings include:

- Single Axis Calibration – This setting requires a full rotation in the yaw axis (relative to earth) to determine the calibration. Additional data that is collected via motion on other axes is used but not required. Once a full rotation is completed the calibration is calculated and the online continuous calibration is started.
- Multi Axis Calibration – This setting requires data to be collected across at least two axes, where one is the yaw axis. This calibration mode does not have any specific angular rotational requirements in any given axes, but it does require that data has been collected across a sufficient angular span. There is an indicator (**mag_cal_threshold_complete**) in the **`DID_SYS_PARAMS`** message that relates the total percent of the required threshold that has been collected. As more data is collected this value will increment to 100% at which point the calibration will be calculated and the online continuous calibration will continue to run

```
/*! Magnetometer recalibration. 0 = multi-axis, 1 = single-axis */
SYS_CFG_BITS_MAG_RECAL_MODE_MASK = (int)0x00000700,
SYS_CFG_BITS_MAG_RECAL_MODE_OFFSET = 8,
```

13.6 System Status

13.6.1 Solution Status

Solution Alignment

Solution alignment occurs when aiding sensor and state estimation are in agreement and indicates that solution output can be trusted.

HEADING ALIGNMENT

Heading alignment varies based on available sensors and conditions of motion.

Stationary INS and AHRS (no GPS) use the magnetometer for heading alignment. The INS solution will start and remain in INS_ALIGNING status (1) until the magnetometer calibration is validated. The magnetometer requires +- 45 degrees of rotation to validate calibration and ensure accurate heading.

Moving INS (under accelerated conditions) or Dual GNSS (two GPS antennas using RTK compassing) can align the heading without use of the magnetometer. The INS solution will start in INS_ALIGNING status (1) and immediately proceed to INS_NAV_IS_GOOD status (3) with GPS lock for Dual GNSS and accelerated horizontal movement for single GNSS INS use.

LED Status

System status including GPS lock, INS alignment, and time synchronization are reported via the top tri-color LED. This LED can be disabled (turned off) by setting bit 0x4 of DID_FLASH_CONFIG.sysCfgBits.

LED Behavior	Solution Status #	Status	Description
White ⚪	1	Solution Aligning	The solution is aligning on startup
Cyan 🔵	2	Solution Alignment Complete	The solution has aligned but insufficient dynamics have been completed for the variance to reach nominal conditions.
Green 🟢	3	Solution Good – NAV	The solution is in Navigation mode and state estimate is good.
Blue 🔵	5	Solution Good – AHRS	The solution is in AHRS mode and state estimate is good. There is no valid position or velocity data from GPS or other aiding sensor. Only the attitude states are estimated
Orange 🟠	4,6	Solution High Variance	The solution is in Navigation or AHRS mode but variance (uncertainty) is high. This may be caused by excessive sensor noise such as vibration, magnetic interference, or poor GPS visibility or multipath errors. See DID_INL2_VARIANCE.
Purple 🟣		Magnetometer Recalibration	The system is collecting new magnetometer calibration data and requires rotation.
Cyan Fading 🔵🔵🔵🔵		Bootloader Waiting	The bootloader has started and is waiting for communications.
Purple Fast Blink 🟣🟣🟣🟣🟣🟣		Firmware Upload	The bootloader is uploading the embedded firmware.
Orange Fast Blink 🟠🟠🟠🟠🟠🟠		Firmware Verification	The bootloader is verifying the embedded firmware.
Red 🔴		Bootloader Failure	The bootloader has experienced a failure on startup.
Can combine with behaviors above			
Red pulse every 1s 🔴 🔴	GPS PPS Sync		The system has received and synchronized local time to UTC time using the GPS PPS signal.
Red/purple pulse every 1s 🔴🟣 🔴🟣	RTK Base Data Received		The system is receiving RTK base station data.
Purple pulse every 1s 🟣 🟣	RTK Fix Status		The GPS has valid RTK fix and high precision positioning

Status flags described in this section that can be observed by bitwise ANDing any of the status flag bitmasks with the corresponding status flags variable.

Status Flags

This section lists the commonly used status flags. A complete listing of status flags is available in data_sets.h.

INSSTATUS – INS STATUS FLAGS

The INS status flags, **insStatus**, are found in the DID_INS1, DID_INS2, DID_INS3, and DID_SYS_PARAMS messages. Bitmasks for the **insStatus** flags are defined in *eInsStatusFlags* in data_sets.h.

Field	**Description**
INS_STATUS_ALIGN_COARSE_MASK	Position, Velocity & Attitude are usable. Variance of the state estimate are outside spec.
INS_STATUS_ALIGN_GOOD_MASK	Position, Velocity & Attitude are good. Variance of state estimate are within spec.
INS_STATUS_GPS_AIDING_HEADING	INS heading are being corrected by GPS.
INS_STATUS_GPS_AIDING_POS	INS position and velocity are being corrected by GPS.
INS_STATUS_GPS_UPDATE_IN_SOLUTION	GPS update event occurred in INS, potentially causing discontinuity in position path.
INS_STATUS_MAG_AIDING_HEADING	INS heading are being corrected by magnetometer.
INS_STATUS_NAV_MODE	AHRS = 0 (no position or velocity), NAV = 1
INS_STATUS_MAG_RECALIBRATING	Magnetometer is recalibrating.
INS_STATUS_MAG_NOT_GOOD	Magnetometer is experiencing interference.
INS_STATUS_SOLUTION_MASK	0=INS_INACTIVE – The INS is not runnning **1=INS_STATUS_SOLUTION_ALIGNING** – The INS is aligning on startup **2=INS_STATUS_SOLUTION_ALIGNMENT_COMPLETE** – The INS has aligned but insufficient dynamics have been completed for the variance to reach nominal conditions. **3=INS_STATUS_SOLUTION_NAV** – The INS is in NAV mode and the state estimate is good. **4=INS_STATUS_SOLUTION_NAV_HIGH_VARIANCE** – The INS is in NAV mode and the state estimate is experiencing high variance. This may be caused by excessive noise on one or more sensors, such as vibration, magnetic interference, poor GPS sky visibility and/or GPS multipath errors. See DID_INL2_VARIANCE. **5=INS_STATUS_SOLUTION_AHRS** – INS is in AHRS mode and the solution is good. There is no valid position correction data from GPS or other aiding sensor. Only the attitude states are estimated. **6=INS_STATUS_SOLUTION_AHRS_HIGH_VARIANCE** – INS is in AHRS mode and the state estimate has high variance. See DID_INL2_VARIANCE.

HDWSTATUS – HARDWARE STATUS FLAGS

The hardware status flags, **hdwStatus**, are found in the `DID_INS1`, `DID_INS2`, `DID_INS3`, and `DID_SYS_PARAMS` messages. Bitmasks for the **hdwStatus** flags are defined in `eHdwStatusFlags` in data_sets.h.

Field	Description
HDW_STATUS_MOTION_MASK	Accelerometers and Gyros are operational
HDW_STATUS_GPS_SATELLITE_RX	Antenna is connected to the GPS receiver and signal is received
HDW_STATUS_STROBE_IN_EVENT	Event occurred on strobe input pin
HDW_STATUS_SATURATION_MASK	Acc., Gyro, Mag or Baro is saturated
HDW_STATUS_SELF_TEST_FAULT	BIT has failed
HDW_STATUS_ERR_TEMPERATURE	Outside of operational range
HDW_STATUS_FAULT_BOD_RESET	Low Power Reset
HDW_STATUS_FAULT_POR_RESET	Software or Triggered Reset

13.6.2 Built-in Test (BIT)

Built-in test (BIT) is enabled by setting `DID_BIT.state` to any of the following values.

```
BIT_STATE_CMD_FULL_STATIONARY               = (int)2, // (FULL) Comprehensive test.  Requires system be completely stationary without vibrations.
BIT_STATE_CMD_BASIC_MOVING                  = (int)3, // (BASIC) Ignores sensor output.  Can be run while moving.  This mode is automatically run after bootup.
BIT_STATE_CMD_FULL_STATIONARY_HIGH_ACCURACY = (int)4, // Same as BIT_STATE_CMD_FULL_STATIONARY but with higher requirements for accuracy.
```

BIT takes about 5 seconds to run, and is completed when `DID_BIT.state == BIT_STATE_DONE (1)`. All BIT tests except those related to GPS require the system to be stationary to be accurate.

hdwBitStatus – Hardware BIT Flags

Hardware BIT flags are contained in **hdwBitStatus**, found in the `DID_BIT` message. Bitmasks for the **hdwBitStatus** flags are defined in `eHdwBitStatusFlags` in data_sets.h.

Field	Description
HDW_BIT_PASSED_ALL	All HBIT are passed
HDW_BIT_PASSED_AHRS	All Self Tests passed without GPS signal
HDW_BIT_FAILED_MASK	One of the built-in tests failed
HDW_BIT_FAULT_GPS_NO_COM	No GPS Signal
HDW_BIT_FAULT_GPS_POOR_CNO	Poor GPS signal. Check Antenna
HDW_BIT_FAULT_GPS_ACCURACY	Poor GPS Accuracy or Low number of satellites

calBitStatus – Calibration BIT Flags

Calibration BIT flags are contained in **calBitStatus**, found in the `DID_BIT` message. Bitmasks for the **calBitStatus** flags are defined in `eCalBitStatusFlags` in data_sets.h.

Field	Description
CAL_BIT_PASSED_ALL	Passed all calibration checks
CAL_BIT_FAILED_MASK	One of the calibration checks failed

13.6.3 Health Monitoring

This section illustrates tests used for system health monitoring in common application.

1. **Communication Check**

   To check that cabling between the unit and the application is working after initialization, expect that the initial expected packets are received within 3 seconds.

2. **Sensor Test (Must be Stationary)**

   These tests are ideal for manufacturing and periodic in-field testing. Initiate by setting `DID_BIT.state = 2`.

Test	Description
hdwBitStatus & HDW_BIT_PASSED_ALL	Hardware is good
calBitStatus & CAL_BIT_PASSED_ALL	Sensor calibration is good

1. **GPS Hardware Test**

   Initiate by setting `DID_BIT.state = 2`.

Test	Description
hdwBitStatus & HDW_BIT_FAULT_GPS_NO_COM	No GPS serial communications.
hdwBitStatus & HDW_BIT_FAULT_GPS_POOR_CNO	Poor GPS signal strength. Check antenna.

1. **GPS Lock Test**

Test	Description
hdwStatus & INS_STATUS_USING_GPS_IN_SOLUTION	GPS is being fused into INS solution

1. **INS Output Valid**

Test	Description
insStatus & INS_STATUS_ATT_ALIGN_GOOD	Attitude estimates are valid
insStatus & INS_STATUS_VEL_ALIGN_GOOD	Velocity estimates are valid
insStatus & INS_STATUS_POS_ALIGN_GOOD	Position estimates are valid

1. **System Temperature**

   System temperature is available at DID_SYS_SENSORS.temp.

2. **Communications Errors**

   HDW_STATUS_COM_PARSE_ERROR_COUNT(DID_SYS_SENSORS.hStatus) is the number of parsed packet errors encountered.

14. User Manual PDF

This document is provided to allow users to capture a snapshot of our current documentation. **The PDF is an automatically generated printout of the current online documents so there are known link and image sizing issues.** We encourage use the online documentation which is maintained for customer accessibility.

Download PDF

15. Frequently Asked Questions

15.1 What is Inertial Navigation?

Inertial navigation is a technique of estimating position, velocity, and orientation (roll, pitch, heading) by integrating IMU inertial motion data from gyros and accelerometers to continuously calculate the dead reckoning position. The inertial sensors are supplemented with other sensors such as GPS, altimeter, and magnetometer. Inertial navigation is commonly used on moving vehicles such as mobile robots, ships, aircraft, submarines, guided missiles, and spacecraft.

15.2 What does an Inertial Navigation System (INS) offer over GPS alone?

Dead Reckoning - An inertial navigation system (INS) integrates the IMU data to dead reckon (estimate position and velocity) between GPS updates and during GPS outage.

Higher Data Rates - Typical GPS receivers data rates vary from 1Hz to 20Hz whereas INS systems like the uINS have data rates up to 1KHz.

Signal Conditioning - An INS filters out noise in the GPS data and provides a smoother, more continuous data stream.

Orientation Data - An INS is capable of observing the orientation (roll, pitch, and heading) of the system regardless of the motion or direction of travel. This is because of how an INS fuses inertial data with GPS data. A GPS with one antenna can measure direction of travel (ground track heading) but cannot estimate vehicle roll, pitch, or heading.

15.3 How long can the uINS dead reckoning estimate position without GPS?

The uINS inertial navigation integrates the IMU data to dead reckoning position and velocity estimation between GPS updates and for a short period of time during GPS outages. This dead reckoning is designed to filter out GPS noise and provide cleaner faster updates than are available via GPS alone. The uINS dead-reckons, or estimates position and velocity, between GPS updates and through brief GPS outages. However, it is not designed for extended position navigation without GPS aiding. Dead reckoning is disabled after 5 seconds of GPS outage in order to constrain position and velocity drift. The amount of position drift during dead reckoning can vary based on several factors, including system runtime, motion experienced, and bias stability.

15.4 Can the uINS estimate position without GPS?

No. GPS is required to provide initial position estimation and to aid in IMU bias estimation. The uINS can dead reckon (estimate position without GPS) for brief periods of time. However, the quality of dead reckoning is a function of IMU bias estimation, which improves while the GPS is aiding the INS.

15.5 How does vibration affect navigation accuracy?

The uINS accuracy may degrade in the presence of mechanical vibrations that exceed 3g of acceleration. Empirical data shows degradation at approximately 100 - 150 Hz. Adding vibration isolation to the mount may be necessary to reduce the vibrations seen by the product and to improve accuracy.

15.6 Can the uINS operate underwater?

The uINS can only dead reckon for short periods of time and in general requires GPS to provide position and velocity data. The GPS antenna must be above the water surface in order for the GPS to function properly. It is ideal that the GPS antenna be fixed relative to the uINS (IMU) module in order to maintain precision when moving faster than 2 m/s or 0.8 m/s^2. However, the GPS antenna may be tethered above the uINS, where the GPS antenna is floating on the water surface and the uINS is below the water surface. System position will reflect the GPS antenna position and attitude (roll, pitch, heading) will reflect the uINS module orientation.

15.7 Can the uINS operate at >4g acceleration?

Typical L1 GPS receivers lose fix above 4g acceleration because the doppler variation starts to get too large and the receiver may become unstable or not be able to get/keep a fix. Additionally, the acceleration begins to affect the stability of the GPS XTAL oscillator.

On the uINS, the GPS will regain fix within seconds after acceleration drops below 4g. The uINS will track the velocity and position using inertial navigation for up to 5 seconds of GPS outage. As long as GPS outage is below 5 seconds, the uINS should be able to track position through a launch.

15.8 Customer Support

Have other questions or needs? Please email us at support@inertialsense.com.

16. Troubleshooting

16.1 Firmware Troubleshooting

Please email support@inertialsense.com for assistance or to provide feedback on this user guide.

16.1.1 Data doesn't look right

If the EvalTool or SDK are from a different release from the firmware on the unit, there may be communication protocol related issues. It's best to keep both the software and firmware in sync with each other. The EvalTool should flag a protocol mismatch in the settings tab.

16.1.2 Bootloader Not Responding

Check the following:

- The input supply is at 3.3V and clean without noise.
- The serial connection is grounded (no floating grounds).
- The serial wires between the uINS module and the next active device (buffer, converter, or processor) are not longer than 1 meter when bootloading firmware.

16.1.3 Bootloader Update fails first time

If updating the bootloader firmware and using the USB direct connection on the uINS module (pins 1 and 2) or the EVB-2 (EVB USB connector), the serial port number will change when the device switches from application mode to Bootloader Update mode. This is expected and requires reselecting the new serial port and running the Bootloader Update process a second time.

16.1.4 System in AHRS mode despite GPS messages being received

If attempting to enter NAV mode but the system reports AHRS despite GPS data beig received, then assure your units are not set to Rover RTK mode. This will override your ability to lock in GPS Nav mode.

16.1.5 "Bricked" System Recovery

There are different reasons a system may appear unresponsive and not communicate. The following sections describe how to recover a system from these states.

Attention

M M Y indicator that the bootloader is running is the fading cyan module LED. NO communications will appear in the EvalTool or CLTool. **Attempt to update the firmware before performing a chip erase.**

Attention

M M M M M -3_v1.2.1.0_b287_2017-09-17_103826.hex and older will not com following these instructions to be recovered. Do NOT use the chip erase procedure for this scenario.

Stuck in Bootloader Mode

In some cases, the bootloader may fail to completely update firmware. This is indicated by the fading cyan status LED on the uINS module. This can happen if older bootloader firmware is on the uINS and firmware version 1.7.1 is uploaded. If this happens, the system will appear to be unresponsive in the EvalTool. The following process can be used to recover the system to a working state:

If the **bootloader is running**, identified with the fading cyan color LED on the uINS module, following these steps:

1. **Ensure uINS Firmware is Running** - *(This step is not necessary if the uINS firmware is running and the EvalTool is communicating with the uINS)*. Select the device serial port and update the firmware using 1.6.4 or earlier. If the bootloader is running, version information will not appear in the EvalTool. The following bootloader update step will not work unless the EvalTool is communicating with the uINS firmware.
2. **Update the Bootloader** - Use the EvalTool "Update Bootloader" button in the Settings tab to upload the latest bootloader firmware. If it has a fading cyan color on the uINS module, the bootloader is running and ready for new firmware to be loaded. **The bootloader can only be updated using serial0 or the native USB connection.**
3. **Update the Firmware** - Use the EvalTool "Update Firmware" button to upload the latest uINS firmware.

If neither the bootloader or the uINS firmware are running, identified with the solid or no LED status on the uINS module, please contacts us.

Recovery for Firmware v1.2.1.0

Hardware v3.1.3 and newer and firmware IS_uINS-3_v1.2.1.0_b287_2017-09-17_103826.hex and older result in a system that runs but will not communicate properly. This older firmware was not designed for the newer hardware and consequently runs the processor at a slower speed, which alters all of the predefined baud rates to non-standard irregular baud rates. A symptom of this problem is the LED flashing to indicate the processor activity and the module never communicates properly.

The following steps are provided to recover communications with the system.

1. Install and run the hotfix release 1.1.3 EvalTool.
2. Select the special baud rate **560,000** in the EvalTool and open the serial port.
3. Update the firmware using any version **newer than** IS_uINS-3_v1.2.1.0_b287_2017-09-17_103826.hex.

The latest EvalTool, CLTool, SDK, and firmware can be used once the firmware has been updated on the module.

Steps for Chip-Erase Recovery

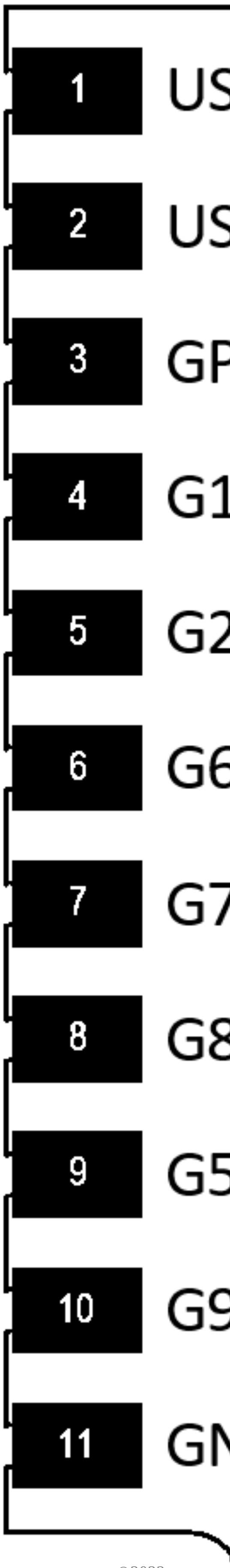
Warning

The CHIP ERASE (Reserved (CE) pin 17) erases all flash memory and should only be used as a last resort. This step should ONLY be used if the steps for Stuck in Bootloader Mode fail and there is NO other method to recover communications.

Important

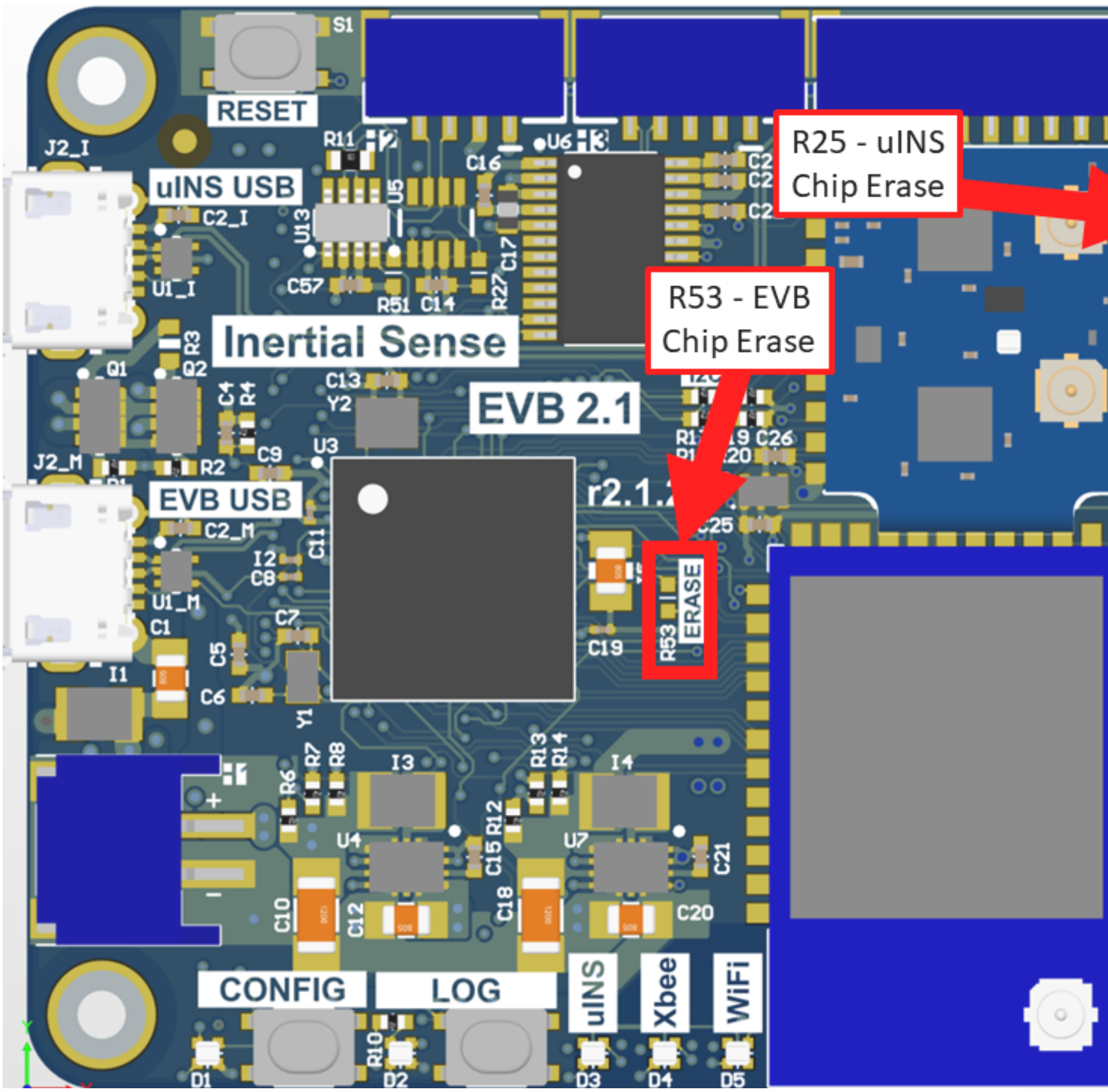
Please notify support@inertialsense.com if this step is necessary so that we can keep track of cause of failures and provide you any necessary support.

UINS CHIP ERASE PAD


1
US
2
US
3
GP
4
G1
5
G2
6
G6
7
G7
8
G8
9
G5
10
G9
11
GN

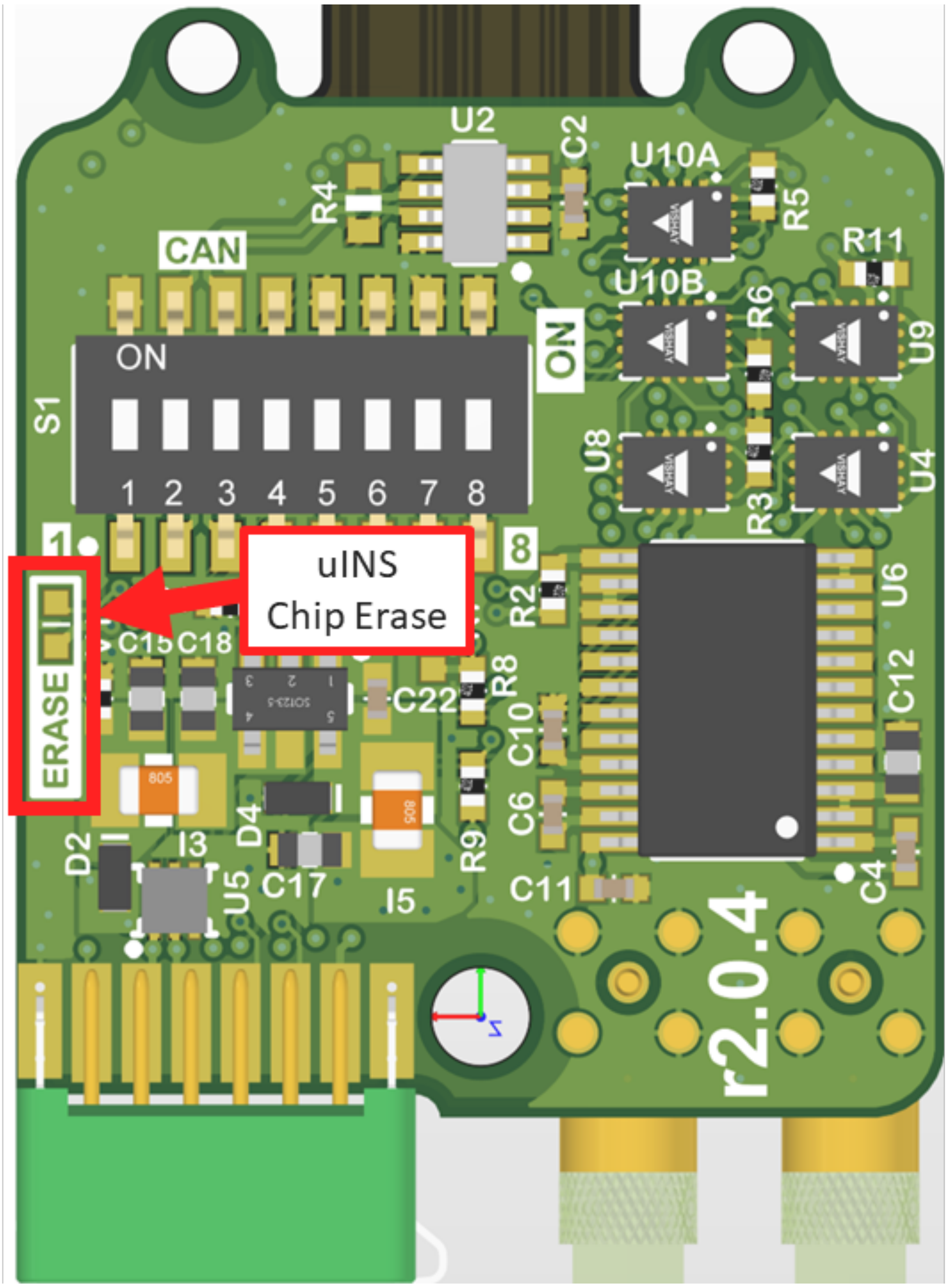
Short pin 17 to pin 22 to chip erase uINS.

EVB-2 CHIP ERASE PADS



Short R25 pads to chip erase uINS.
Short R53 pads to chip erase EVB-2.

RUGGED-2 CHIP ERASE PADS



Short pads to chip erase uINS.

RESTORE FIRMWARE

1. **Power on system**
2. **Record your uINS Serial Number** - If you can read the serial number, record it for reference. On older firmware versions the serial number will be erased. New firmware versions store the serial number in a location that chip erase won't touch.
3. **Chip Erase uINS** - Momentarily assert Chip Erase (Reserved (CE) pin 17) on the uINS module longer than 100ms by connecting to +3.3V. +3.3V is available on pin 2 of all EVB headers. **Warning!!!** - CHIP ERASE erases all flash memory and should only be used as a last resort. This step should ONLY be used if there is NO other method to recover communications.

4. **Reset the system**
5. **Enable EvalTool Internal Mode** - This exposes the "SAM-BA" button and "Manufacturing" tab.
6. **Restore the bootloader firmware** - Open the device serial port and use the "SAM-BA" button in the EvalTool Settings tab to load the bootloader firmware.
7. **Restore the uINS firmware** - Open the device serial port and use the "Update Firmware" button in the EvalTool Settings tab to load the uINS firmware.

ENABLE EVALTOOL INTERNAL MODE

EvalTool internal mode is used to access the EvalTool Manufacturing tab, used to restore serial numbers and calibration data.

1. Close the EvalTool so it isn't running.
2. Using a text editor, change the value of "DBGINT" to 99 (i.e. `"DBGINT": 99,`) in settings file: C:\Users\[USERNAME]\Documents\Inertial Sense\settings.json.
3. Restart EvalTool and verify "[INTERNAL MODE]" is in the title bar.

RESTORE SENSOR CALIBRATION

Contact InertialSense and provide your uINS serial number to request the sensor calibration that corresponds with your uINS. Use the EvalTool to upload the senor calibration onto your uINS.

1. Ensure the EvalTool is in Internal Mode which provides access the Manufacturing tab.
2. Ensure uINS is communicating with EvalTool.
3. Upload calibration data: EvalTool -> Manufacturing Tab -> "Load" button next to "System Test" button.
4. Verify "TC Pts" which is the number of calibration points located just below the "Load" button changes from "0,0" to two numbers larger than 12 (i.e. "18,18").
5. Reset the uINS.
6. **Run "Built-In Test"** - Verify the built-in test passes by pressing the "Built-In Test" button in the EvalTool INS tab.
7. **Verify IMU output** - Place the uINS on a flat level surface. Using the EvalTool Sensor tab, verify that the gyro rates are near zero, the accelerometer X and Y axes are near zero, and accelerometer Z axis is near -9.8m/s^2 for gravity.

16.1.6 Troubleshooting with EvalTool

Units Not Connecting

In the case that your units do not connect properly to the EvalTool, verify:

1. The baud rate is the same that you previously had when the Com Ports last opened correctly.
2. The LED on the unit is not showing solid white, flashing white, or solid red. These mean a failure occured in loading the bootloader (see User Guide for full LED descriptions).
3. If you are using a USB3.0 connection, the com port might take longer to show up than with USB2.0
4. Check your computer's Device Manager to see if your unit shows up there. If it doesn't show up, you may have an FTDI driver issue.
    1. If you suspect you don't have the FTDI driver installed on your Windows computer, use the following links to download the driver: - Executable for the FTDI USB driver: - http://www.ftdichip.com/Drivers/CDM/CDM21228_Setup.zip - Drives without executable. - http://www.ftdichip.com/Drivers/D2XX.htm

16.1.7 1.7.6 Bug RTK Base GPS Raw work around

If you are having base raw errors on your Rover, in the bottom right of the Evaltool, or a climbing Diffrential Age, in Data Sets DID_GPS1_RTK_REL, you maybe having this bug. Try this workaround.

1. Go-to settings tab, open the Base serial COM port.
2. Go-to Data Logs tab, under RCM Presets dropdown select PPD.
3. **NOTE:** You must leave the comport open on the Base.
4. Check your Rover to see if its still getting raw errors messages.

16.2 uINS Firmware Troubleshooting

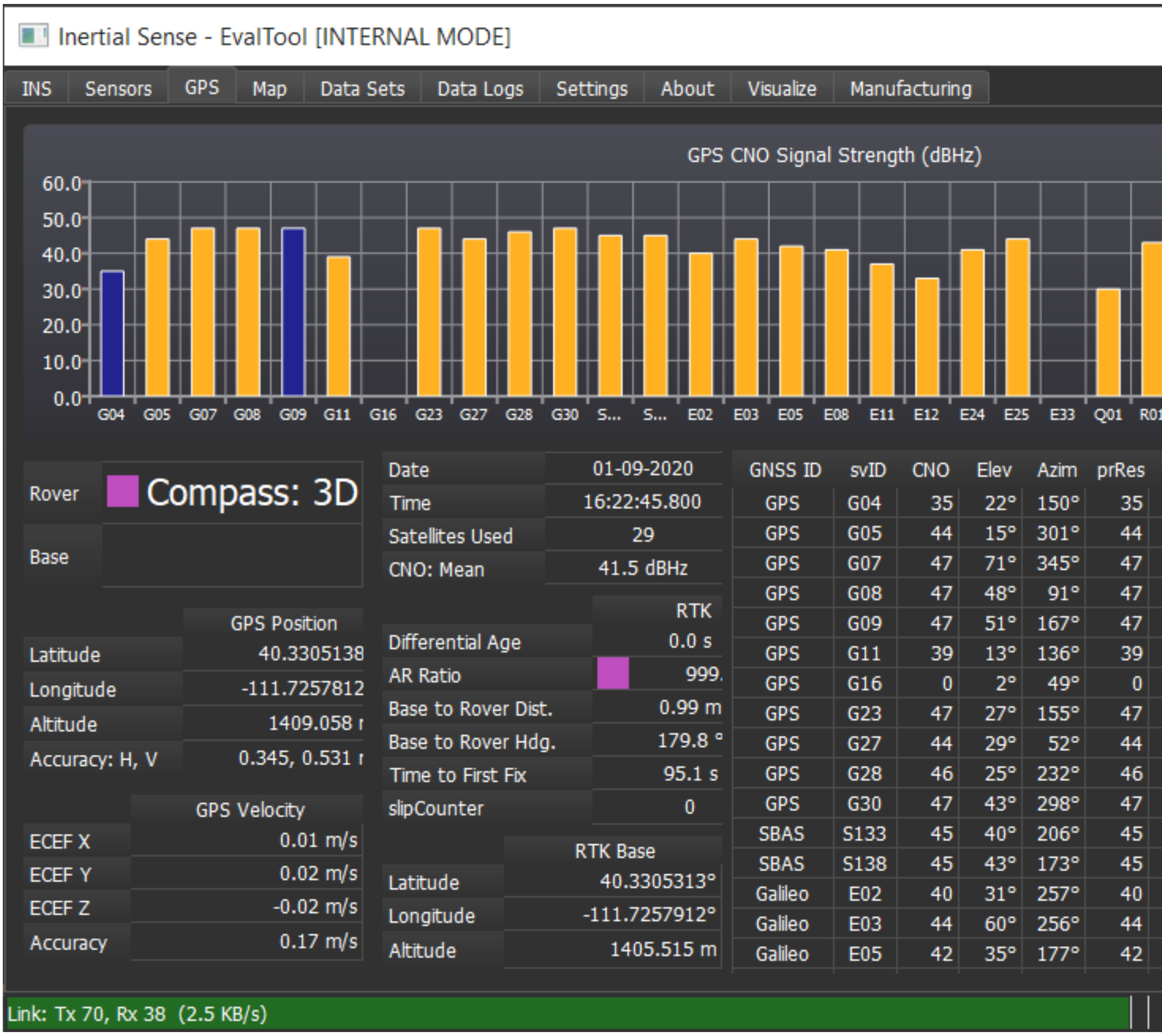
16.2.1 Antenna Baseline

Separation between GNSS antennas (or baseline distance) impacts the accuracy and fix time of the solution. Typical Dual GNSS heading fix time is 60-90 seconds using a 1 meter baseline. Baseline distances shorter than 1 meter will impact both heading accuracy and time to fix. However, having a short baseline of 0.35m should not cause an extremely long fix time.

ITEM TO TEST: Try increasing the antenna baseline to 0.5m or greater during initial testing.

16.2.2 Satellite CNO Strength, RFI and EMI

What is the satellite CNO (signal strength) level? The mean CNO would ideally be above 38-40. Anything lower could indicate the presence of RF interference (RFI) or electromagnetic interference (EMI). You can see this in the EvalTool GPS tab or the DID_GPS1_POS message.



Rover	🟪 Compass: 3D
Base	

	GPS Position
Latitude	40.3305138
Longitude	-111.7257812
Altitude	1409.058 r
Accuracy: H, V	0.345, 0.531 r

	GPS Velocity
ECEF X	0.01 m/s
ECEF Y	0.02 m/s
ECEF Z	-0.02 m/s
Accuracy	0.17 m/s

Date	01-09-2020
Time	16:22:45.800
Satellites Used	29
CNO: Mean	41.5 dBHz

	RTK
Differential Age	0.0 s
AR Ratio	🟪 999.
Base to Rover Dist.	0.99 m
Base to Rover Hdg.	179.8 °
Time to First Fix	95.1 s
slipCounter	0

	RTK Base
Latitude	40.3305313°
Longitude	-111.7257912°
Altitude	1405.515 m

GNSS ID	svID	CNO	Elev	Azim	prRes
GPS	G04	35	22°	150°	35
GPS	G05	44	15°	301°	44
GPS	G07	47	71°	345°	47
GPS	G08	47	48°	91°	47
GPS	G09	47	51°	167°	47
GPS	G11	39	13°	136°	39
GPS	G16	0	2°	49°	0
GPS	G23	47	27°	155°	47
GPS	G27	44	29°	52°	44
GPS	G28	46	25°	232°	46
GPS	G30	47	43°	298°	47
SBAS	S133	45	40°	206°	45
SBAS	S138	45	43°	173°	45
Galileo	E02	40	31°	257°	40
Galileo	E03	44	60°	256°	44
Galileo	E05	42	35°	177°	42

Link: Tx 70, Rx 38 (2.5 KB/s)

ITEM TO TEST: Try powering off portions of the system while running the uINS. You may even try running the uINS independently to a separate computer to monitor the system so you can completely power off your system with the uINS still running. Pay close attention to the GPS CNO during each change.

16.2.3 USB Interface

USB-3 has been known to interfere with wireless and GNSS systems. It is the most common source of interference that has been experienced. Properly shielded cables or signal filters can address this.

ITEM TO TEST: Disable USB-3, digital busses, or various switching supplies and observe the satellite CNO level. CNO mean should be near or above 40 dB/Hz.

16.2.4 Antenna Ground Planes

The ground planes should be adequately sized. Larger ground planes help but are generally not the root cause of poor performance. Separate and common ground planes are both acceptable.

ITEM TO TEST: Try doubling the ground plane below the antenna. A simple sheet of metal placed below the antenna is fine. This is also not likely the root cause but worth testing.

16.2.5 Antenna Cable Ground Loops

In come cases the GNSS antenna cable can form an electrical loop and cause interfere.

ITEM TO TEST: Try to ensure cables never loop back or are bundled. If possible shorten cables to smallest required length. Monitor GPS CNO before and after.

16.2.6 Local Interference

In some cases we have seen object in close proximity to the GNSS antennas act as a multi-path surface, reflect GNSS signals onto the GNSS antennas. Ensure no objects are near the antennas above the plane of the antennas.

16.2.7 Antenna Orientation

A more accurate heading can be achieved if the GNSS antennas have the same antenna orientation (point the same way). You should consider rotating one or both GNSS antennas so the coax cable exit in the same direction on each antennas.

Mismatch