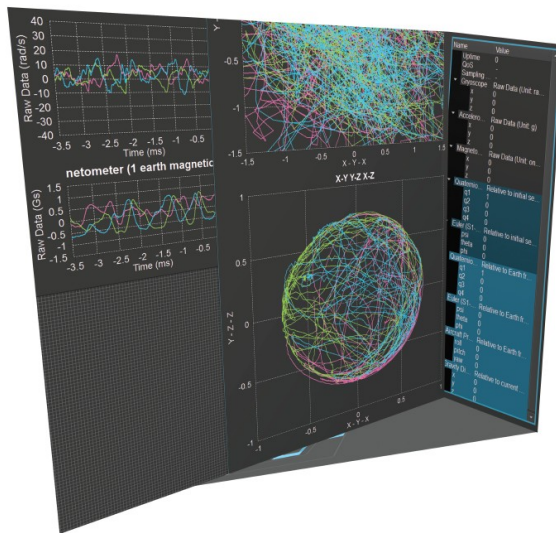


TransducerM User Guide

For product series: TM300

This document also applies to TM500 series basic operation

TransducerM is a 9 Degree-of-Freedom attitude and heading reference system (AHRS)



Version	Date	Revision Info
V1.1.1 (D)	Nov 09, 2017	Creation. Derived from QuickStartGuide_V2-2-3.
V1.2.4 (R)	Jun 27, 2018	Release version. Detailed description on the communication protocol.
V1.2.5 (R)	Jul 24, 2018	Release version. Compatible with ImuAssistant V3.7.9 and later.
V1.2.6 (R)	Jul 30, 2018	Release version. Enhanced Forced Calibration description.
V1.2.7 (R)	Dec 10, 2018	Release version. Fix spelling mistakes. Use CAN 2.0 B instead of A.
V1.2.8 (R)	Aug 22, 2019	Release version. Add Setting Object, Acknowledge Object description.
V1.3.1 (R)	Mar 18, 2020	Release version. ‘Connecting the Hardware’ section updated.

** This document is non-public and is only for intended recipients.*

* Actual product might be different from the photo illustrated.

* Specifications are subject to change without notice.

Table of Contents

Introduction.....	3
What is TransducerM.....	3
Software Versions.....	3
Quick Start.....	4
Prepare.....	4
Install GUI Configuration Software.....	4
Connecting the Hardware.....	5
Install TransducerM.....	8
Power On.....	8
Finding Your Device.....	9
Opening the Device.....	10
GUI Explanation.....	10
Visualizing Data.....	11
Save Setting.....	12
Standard Setup and Test Procedure.....	13
Step 1 – Restore Default Setting.....	13
Step 2 – Heat Up.....	14
Step 3 – Record data.....	15
Step 4 – Run the Test.....	15
Step 5 – Save the Recorded Data and Finish.....	15
In-depth Description.....	16
TransducerM – Node ID, Firmware Version, UUID.....	16
ImuAssistant – Version Number.....	16
Enable and Disable Sensors.....	17
Boot Mode.....	17
Calibration Panel.....	18
Sensor Fusion.....	19
Output Data Types.....	19
Communication Protocol.....	20
Change UART Baudrate.....	20
Increase Output Rate.....	21
Roll Pitch Yaw Display.....	21
Quaternion Display.....	21
Raw Data Display.....	21
X-Y Display.....	22
Save the Setting.....	22
Export Communication Library.....	22
Data Recorder.....	23
Use SYD Dynamics Communication Library.....	25
C++ Library (Recommended, Full API).....	25
C Library (Basic API).....	26
Avoid Buffer Overflow.....	27
Write Your Own Communication Library.....	28
Protocol Overview.....	28
EasyProtocol.....	29
EasyPipeline.....	43
Technical Support.....	46

Introduction

What is TransducerM

SYD Dynamics TransducerM is a complete solution for motion sensing applications, capable of providing computed data for determining orientation of an object in 3D space.

Out-of-box, it provides orientation data in terms of Euler angles, Quaternion, and, most commonly used Roll/Pitch/Yaw all of which can be computed with the reference to world frame (based on Earth's magnetic field and gravity direction). It can also output calibrated raw sensor data, including angular rate, acceleration and magnetometer measurement.

Magnetometer is equipped with 'Active Magnetic Field Compensator' to detect and remove any disturbances and ensure stable magnetometer data.

Software Versions

This user manual is intended to be used with the following software versions.

For instructions on how to check the version number, please refer to section 'TransducerM – Node ID, Firmware Version, UUID' on page 16 and section 'ImuAssistant – Version Number' on page 16.

<i>Firmware Version</i>	
Version Number	Comments
V 3.1.6 (3)	More features are supported as the version number increases. The version number is fixed and determined by a particular TransducerM model.
V 3.1.8 (3)	
V 4.7.5 (3)	
V 4.7.x (*) (x > 5)	For an application using TransducerM with earlier firmware version: it can usually be directly replaced with a later firmware version TransducerM without any modification in the host application software unless new features are to be used.
V 4.9.x (*) (x > 3)	
V 5.1.x (*) (x > 5)	
V 5.x.y (*) (x > 1, y > 1)	

<i>ImuAssistant Version</i>	
Version Number	Comments
V 3.7.9	ImuAssistant V3.7.9 and V3.7.x (x>9) are designed to be used with all the above mentioned firmware versions.
V 3.7.x (x > 9)	Please restart ImuAssistant when connecting to TransducerM with different firmware versions, as the current ImuAssistant does not support connecting to multiple TransducerM's with different firmware versions at the same time.
V 3.8.x (x > 1)	

Note: for other software versions, please contact the support team.
Contact information can be found in section 'Technical Support' on page 46.

Quick Start

This guide aims to explain the use of TransducerM, a 9 Degree-of-Freedom attitude and heading reference system (AHRS), its development kit and accompanying GUI software, ImuAssistant. It will guide you through installing the software, connecting hardware, navigating GUI and understanding basic parts of it.

Prepare

Completing this Quick Start guide will require the following items:

- PC running Windows 7/8/8.1/10.
- ImuAssistant software (ImuAssistant_Setup_Win32_Vx-x-x.zip)
- USB-to-Serial TTL converter (or other means of interfacing serial device using TTL logic from PC)
- TransducerM PCBA/Module

Install GUI Configuration Software

TransducerM comes with a graphical user interface software – ImuAssistant – for configuration and data visualization purpose.

Use the latest version of the software installer that comes with this document.

Run the file “ImuAssistant_Setup_Win32_Vx-x-x.exe” and follow the instructions from the installer.

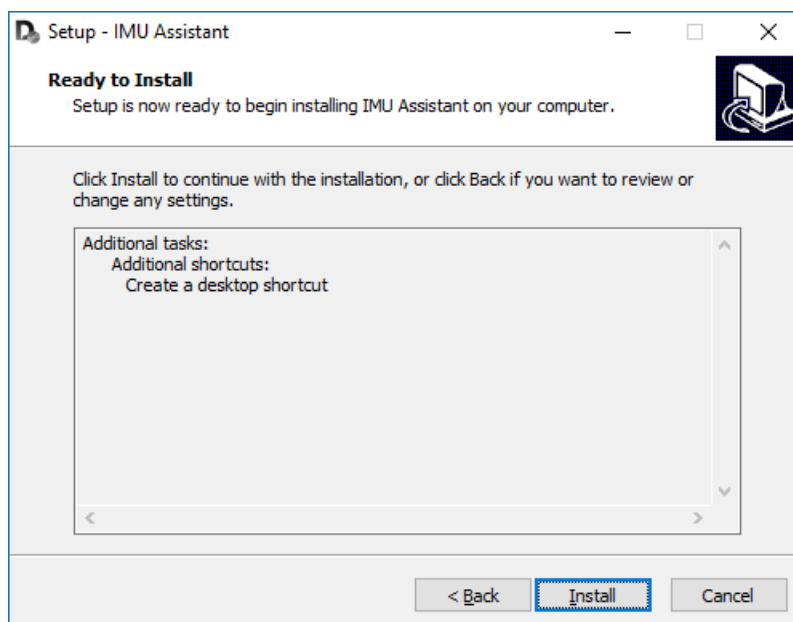


Figure 1: Software installer



Figure 2: Installer icon

After the installation has completed, run GUI software. You should see the screen similar to the one below if installation was successful.

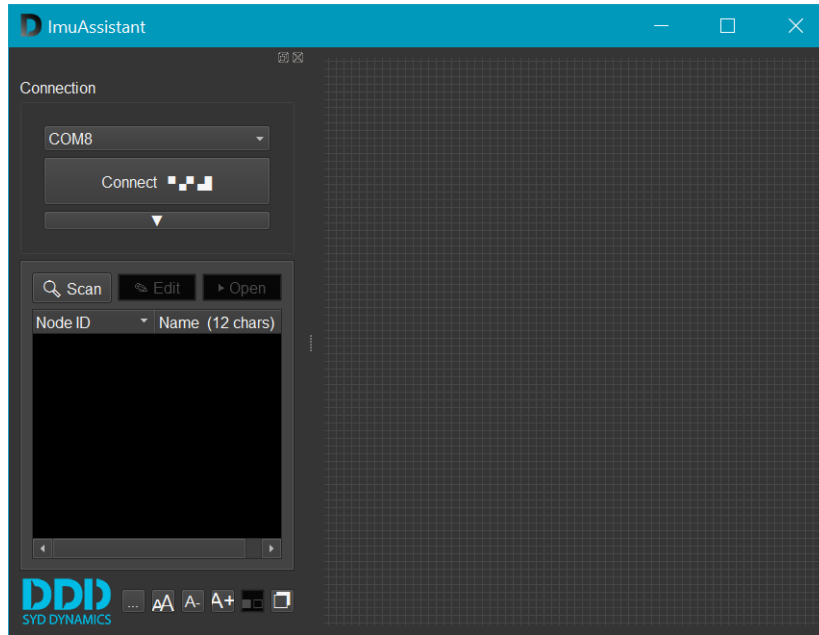


Figure 3: GUI window

You can leave the GUI running while going through the next step.

Connecting the Hardware

Using the UART Interface

The UART interface allows to either connect TransducerM to a micro-controller directly, or to a USB port of a PC through a converter.

For TransducerM with rugged design (Part Number TM352, TM353), if a USB-to-Serial adapter is shipped together with your TransducerM, simply connect in a similar way as shown in Figure 4 and Figure 6. Please check pin-to-pin definition according to product datasheet with file name 'TransducerM_TM35x_Datasheet_EN_Vx-x-x.pdf' which comes along with this document.

You can also choose to use your own USB-to-Serial adapter.



Figure 5: TransducerM with Rugged Design (TM352)

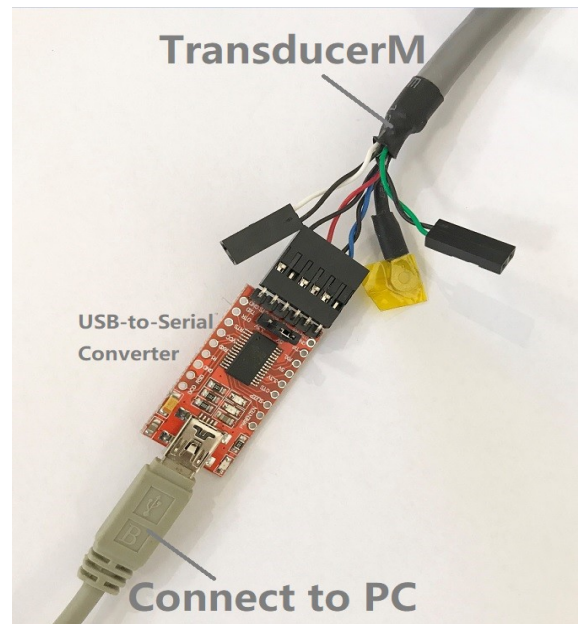


Figure 4: Connect TransducerM with Rugged Design to a USB-to-Serial Adapter

1. Connect adapter cable's pin1(red wire, **VCC**) to USB-UART adapter **5V**;
2. Connect adapter cable's pin7(blue wire, **GND**) to USB-UART adapter **GND**;

3. Connect adapter cable's pin2(white wire, **TXD**) to USB-UART adapte **RXD**;
4. Connect adapter cable's pin6(white wire, **RXD**) to USB-UART adapte **TXD**.

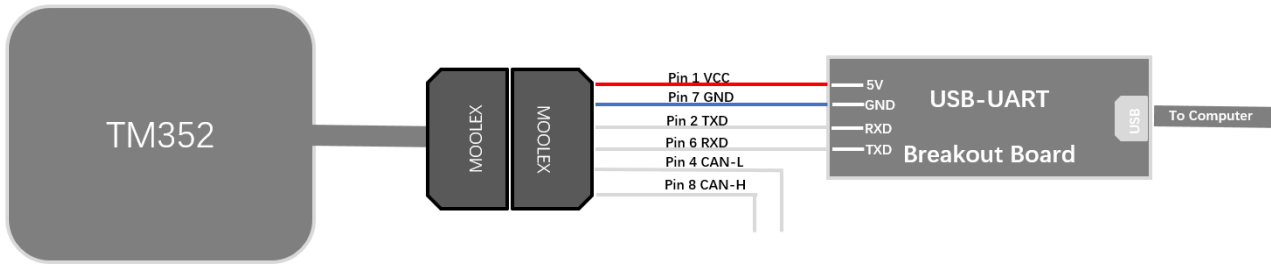


Figure 6: Connect to USB-UART adapter



Please carefully connect and double check the power supply (5V and GND). Reversing the polarity will damage the module.

Using the CAN Bus Interface

For TransducerM with CAN (Controller Area Network) interface (Part Number TM352, TM353), you may interface multiple TransducerM's with your host computer in a network topology as shown in Figure 7.

The interface is CAN 2.0B with 11-bit identifier, compatible with ISO 11898-2 Standard.

The bit rate can be adjusted through GUI, maximum 1Mbps for high speed data output rate (typically up to 200 Hz), and the recommended maximum length of the twisted-pair cable is 40 meters (131 feet).

There is no termination resistor installed inside the TransducerM, therefore the twisted-pair should be terminated with 120Ω at both ends. The length of the cable-stubs, or the 'branch' as referred to in Figure 7, should not exceed 20 centimeters (7.87 inches) and the wiring should be as close as possible to a single-line topology, as a mean to reduce reflections of the signal.

The CAN interface of TransducerM is driven by a CAN Bus Transceiver with single 3.3V supply; the dominant differential voltage is minimum 2.45V, and the recessive differential voltage is 0V (nominal). The impedance of the transceiver allows for up to 120 nodes within a bus. The differential voltage between CAN-H and CAN-L should not exceed ± 20 V (assume the voltage is applied through 100Ω impedance), and the voltage at either the CAN-H or CAN-L should always be within -4V and 16V range.

For pin-to-pin definition of wires, please refer to TransducerM Datasheet.

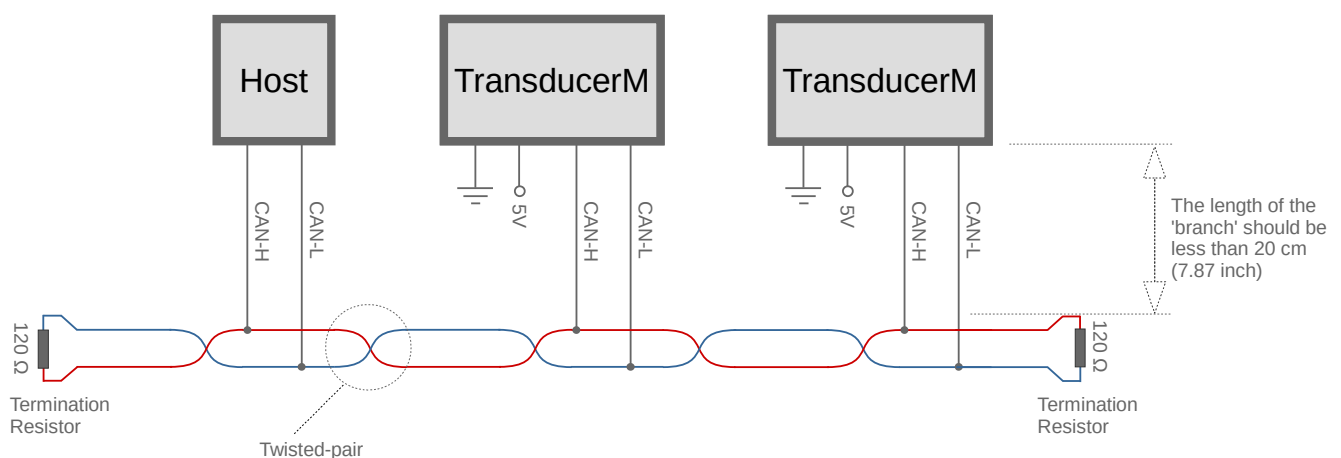



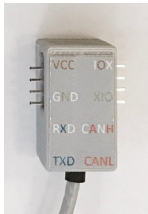

Figure 7: CAN Bus Network Topology

The TransducerM data communication protocol running above the CAN bus data link layer is according to the Communication Abstraction Layer with Pipeline Technology (CALP) defined by SYD Dynamics. While the protocol looks similar to that such as the ISO-TP (ISO15765-2) at first glance, it is a new implementation with significantly improved performance for transferring blocks of data exceeding the size of 8-bytes limitation. The CALP library together with examples should be provided along with this document for customers who require the CAN Bus functionality.

Please also refer to section 'EasyPipeline' on page 43 if you would like to have your own software implementation of the CAN Bus protocol. Note that this implementation does not include advanced features of CALP.

TransducerM PCBA Modules or Customized Versions

For TransducerM in other packages, please refer to this section for hardware setup.

Type of Hardware Interface	RJ45	Customized
Picture of the Interface	Pin:1-8 	Pin:1-8 Shape varies, below is an example 
Pin Definition	1. Orange-White: CAN-H 2. Orange: CAN-L 3. Green-White: GND 4. Blue: TXD 5. Blue-White: RXD 6. Green: X 7. Brown-White: X 8. Brown: 5V	Pin definition is marked on the connector or on the material comes with the module.
Connect to USB-to-TTL Converter	TransducerM – USB-to-TTL Serial Converter 5V – 5V TXD – RXD RXD – TXD GND – GND <div style="display: flex; align-items: center; margin-top: 10px;">  <p>Please carefully connect the power (5V and GND). Reversing the polarity will damage the module.</p> </div> <p>Note: TXD and RXD are running at TTL 3.3V</p>	

Install TransducerM

On TransducerM, you should see axis definition similar to the one in Figure 8, Point the X-Axis to the forward direction of the vehicle being measured. The Z-Axis of TransducerM should point to the sky when the vehicle is sitting on the ground. The 'vehicle' being measured can be ground vehicles, flying vehicles, underwater drones or other types of movable structure.

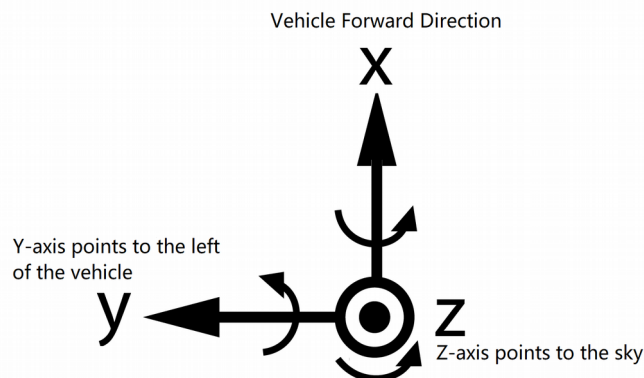


Figure 8: TransducerM Axis Definition

In general, installing TransducerM in a place keeping away as much as possible the magnetic interference and strong vibration allows it to provide better performance. However, this is not a strict requirement if it is not possible to do so, for example, if the TransducerM cannot avoid installing close to a motor (such as in a robot arm, or quad-copter). Select the most relevant software configuration profile as illustrated in section 'Step 1 – Restore Default Setting' on page 13 (Only for TransducerM with firmware version V4.7.5 (3) or higher) to allow the TransducerM compensate for the disturbance.



Figure 9: Example Install of a TransducerM on a Model Car (Not drawn to scale). The X-Axis points to the forward direction.

Power On

Once 5V and GND are correctly connected to a power supply, the module is powered on.



If 'Boot mode' is set to 'Static Mode', please keep the module stationary for at least 13.5 seconds during and after power on. This is essential for the module to initialize and find the earth frame.

This also applies when the 'Boot mode' is set to 'Auto mode' and the module detects a nearly static environment shortly after power on. For more information, please refer to section "Boot Mode".

For users of TransducerM PCBA Module:

When 'Boot mode' is set to 'Static Mode', shortly after power on, you should see an orange LED flashing. The Orange LED signals that the device is acquiring initial state data (Initial sensor frame) and it should be left stationary. After the orange LED has turned off and the green LED starts flashing, the module is ready to provide reliable data to either the GUI on your computer, or another device requesting data.

Finding Your Device

If you've closed the GUI, reopen it, then connect TransducerM to your PC. The GUI is designed to automatically detect any TransducerM attached to system's serial ports. The GUI should now look as seen in figure below:

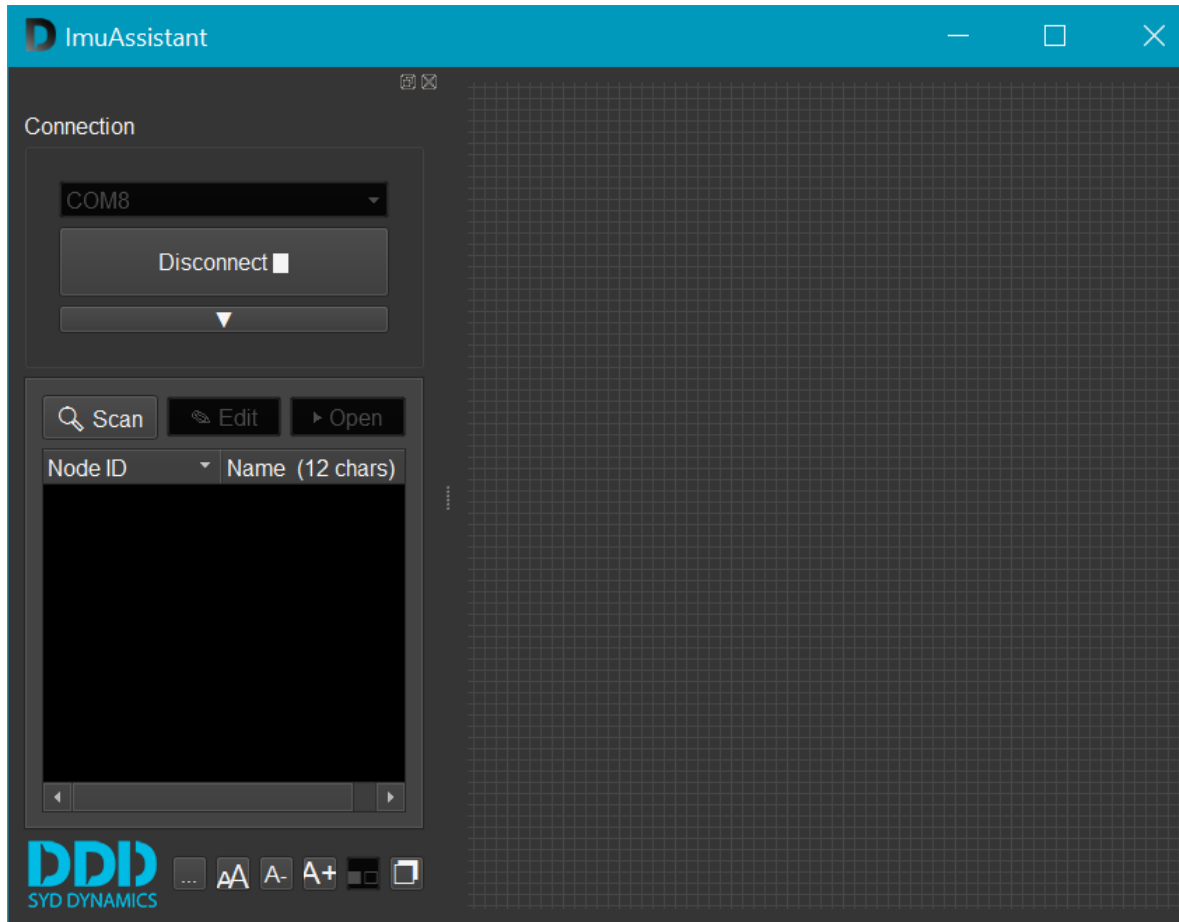


Figure 10: GUI after detecting TransducerM. Note that in this example, COM8 as auto-detected serial port

If software does not open the port on its own, simply reconnect the TransducerM; or by choosing the correct COM port, and then click 'Connect' button found underneath the COM port name.

Next, you need to scan the opened port for any devices attached to it. In this example, we're looking for a single device called 'Evo board'. Scanning the bus is performed by clicking **Scan** button placed below the 'Connect/Disconnect' button.

Note that a single port can have multiple devices attached to it when using CAN bus converter provided by SYD Dynamics, otherwise only one device can be attached.

Once the scan is started you should see the 'Scan' button changing to 'Stop', and a blue loading bar right under it.

Shortly after, your device should be visible in the list.

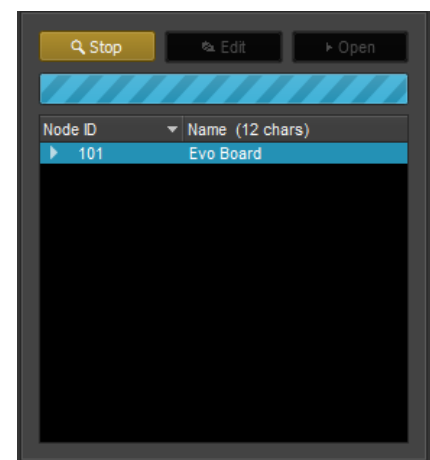


Figure 11: GUI appearance during port scan

Opening the Device

To run the actual sensor and get data out, select the sensor you wish to open from the list and either double-click it with your left mouse button or press 'Open' button above the list (Figure 12, red square). This action loads data-manipulation portion of the user interface in empty space on the right side of the window.



If 'Boot mode' is set to 'Static Mode' or 'Auto Mode', it is recommended to open the device after its initialization, i.e. at least 13.5 seconds after its power on, as the module may ignore the command from GUI during its initialization process.

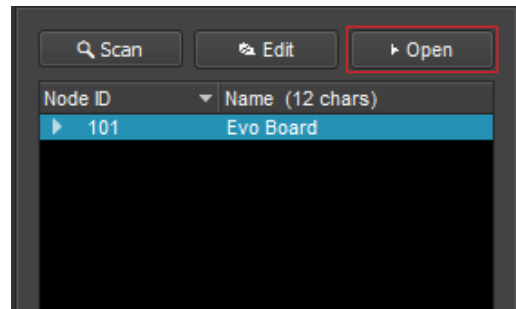


Figure 12: Opening a device

GUI Explanation

Once the device is opened, the data-manipulation portion of the user interface is visible. Shown in Figure 13 (section 1), which is a part of GUI that allows you to fully customize the data path inside your TransducerM.

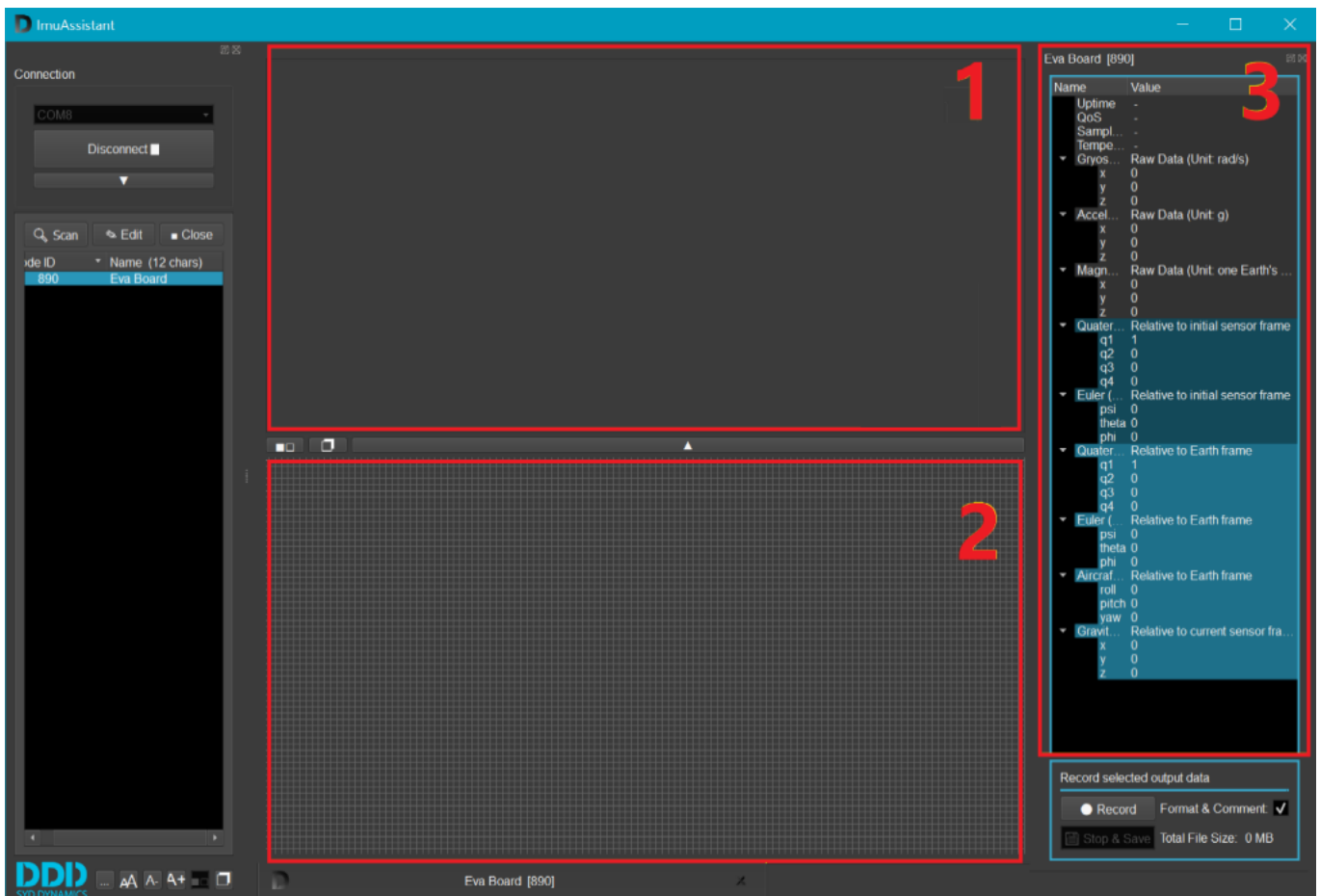


Figure 13: Fully expanded GUI
(Depending on the TransducerM model number and firmware version, you will see the panel configuration slightly differently.)

Visualizing Data

Certain output data can be shown in visual form as well, inside the widgets. All widgets for visualizing data are created in the section number 2 of GUI shown in Figure 13, while corresponding numerical data is still shown under corresponding title in section number 3 in Figure 13.

Raw Data

Selecting 'Raw data' in 'Output data' section will create a widget with 3 graphs to constantly visualize calibrated sensory output on all 3 axes (Figure 14).

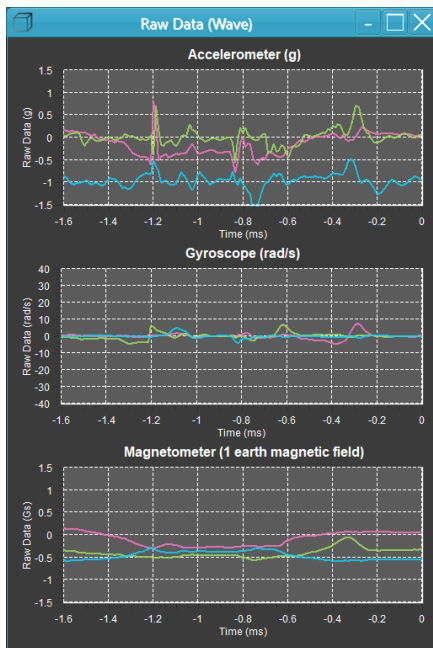


Figure 14: Raw data visualization

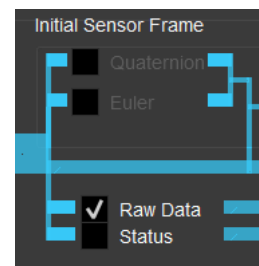


Figure 15: Output data selection

Quaternion

Selecting the 'Quaternion' output under 'World frame' opens a widget with a 3D cube whose orientation in 3D space is aligned relative to the Earth's frame based on the module output (Figure 17). Rotating the module around any of its principal axes also changes the orientation of the cube in its 3D space (Figure 16).

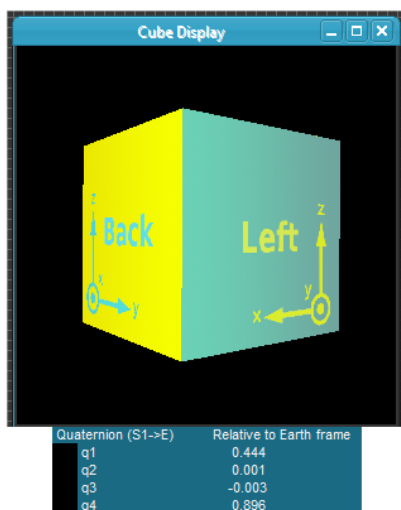


Figure 17: Initial acquired state visualization (relative to Earth's frame)

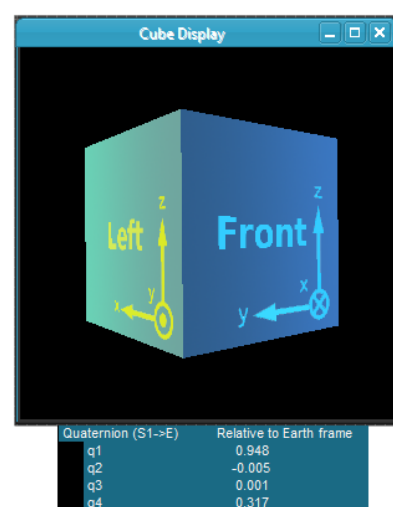


Figure 16: State visualization after rotating 90° CW (looking from top) around Z-axis

Roll Pitch Yaw

Another output with graphical representation is Roll/Pitch/Yaw, or 'RPY' under 'World frame'. It shows the Roll, Pitch and Yaw angles around the aircraft's principal axes (with respect to Earth's frame) and visualizes them in two widgets seen in Figure 18. Widget on the left (Figure 18) combines Roll & Pitch data into a single instrument, while compass on the right shows Yaw angle, or Heading of an object. Same as before, numerical data is still visible in the right part (Figure 23, section 3) of GUI, under 'Aircraft Principal Axes'.

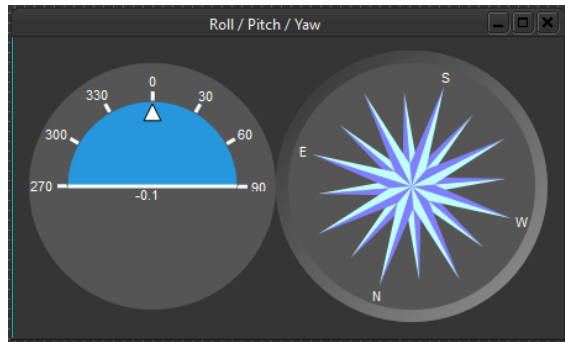


Figure 18: Roll, Pitch, Yaw visualization

Aircraft Principal Axes	Relative to Earth frame
roll	359.8
pitch	-0.1
yaw	107.7

Figure 19: Roll, Pitch, Yaw numerical data

For TransducerM with firmware version V 5.1.x and earlier, it calculates the Yaw (heading) angle based on the magnetometer reading during its initialization process shortly after its power on. That is, the local magnetic field during the TransducerM start up is recognized as the reference magnetic field, and if the X-Axis of TransducerM (the sensor frame) is pointing to the north direction of this local magnetic field, the Yaw (heading) reading is zero degree.

For TransducerM with firmware version V 5.2.x and later, it only calculates the Yaw (heading) when magnetometer is enabled in the setting panel as shown in Figure 13 section 1. Regardless of whether the magnetometer is enabled or not, the Yaw (heading) correction can be performed using magnetometer by sending a command to TransducerM.

Save Setting

It is, however, worth mentioning that all the settings, except calibration button, selected in Figure 13 section 1, will be effective only for current session and once the module is started up again they will be lost. To make settings permanent you can click 'Save Settings' button to burn them into the flash memory of TransducerM from which they can be recalled next time it starts up.

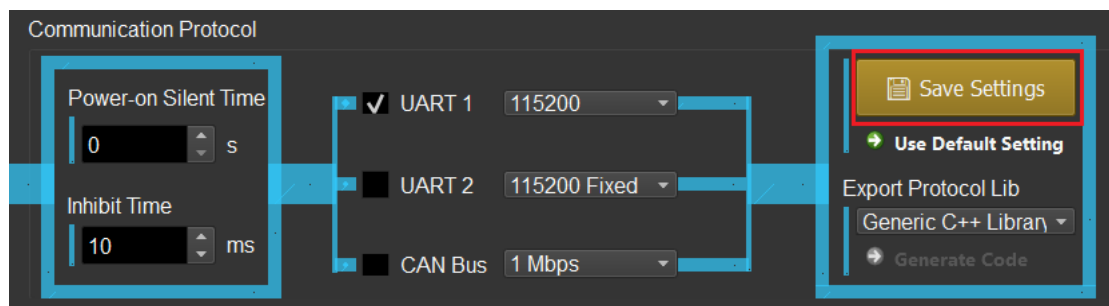


Figure 20: Configuring communication protocol

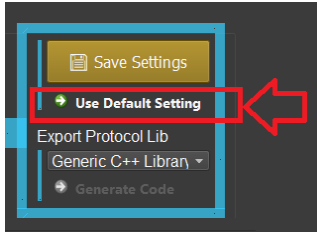
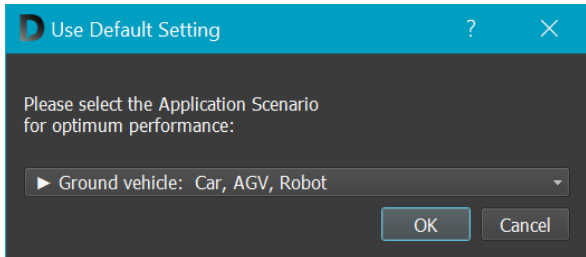
You can use the **Use Default Setting** button (as shown in Figure 20) to reset the setting to its default, and then click 'Save Settings' to save the settings into the TransducerM flash memory.

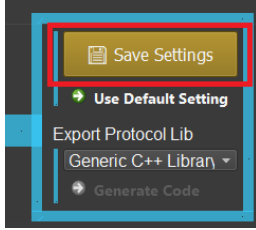
Standard Setup and Test Procedure

In this section, a standard procedure for testing the performance of TransducerM is described, during which you will also become familiar with the ImuAssistant build-in data recorder function.

Step 1 – Restore Default Setting

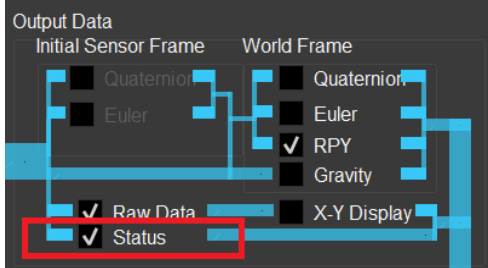
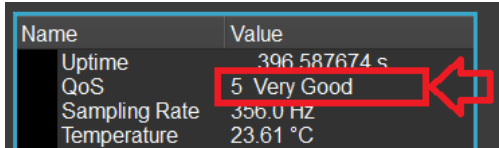

Before the test, a few settings must be done to make sure that TransducerM is properly configured.

Procedure	Instruction	Comment														
1	TransducerM power on															
2	Wait 15 seconds	To ensure the initialization procedure finishes.														
3	Open ImuAssistant, scan and open TransducerM															
4	In ImuAssistant, Click 'Use Default Setting'	<p>This button can be found in section number 3 of Figure 13</p> 														
5	Select the default settings for a particular user scenario. (Only for TransducerM with firmware version V4.7.5 (3) or higher)	<p>A pop-up window will show up, asking which scenario best describes the intended user case.</p>  <table><tr><th>Options of Default Settings</th><th>Comment</th></tr><tr><td>► Ground vehicle: Car, AGV, Robot</td><td>Recommended option for testing.</td></tr><tr><td>► Flying vehicle: Drone, UAV</td><td>Recommended option for testing.</td></tr><tr><td>► Marine application: Boat, Ferry, UUV</td><td>Recommended option for testing.</td></tr><tr><td>► Other applications: with strong and continuous vibration</td><td>Not recommended for testing. Only when you fully understand how 'Auto Boot' works. Refer to 'Boot Mode'.</td></tr><tr><td>► Other applications: without strong and continuous vibration</td><td>Not recommend for testing. Only when you fully understand how 'Auto Boot' works. Refer to 'Boot Mode'.</td></tr><tr><td>► Compatible Mode: Use legacy Firmware V3.x.x default configuration (Lower Performance)</td><td>Lower performance. Not recommended for high demanding usage.</td></tr></table>	Options of Default Settings	Comment	► Ground vehicle: Car, AGV, Robot	Recommended option for testing.	► Flying vehicle: Drone, UAV	Recommended option for testing.	► Marine application: Boat, Ferry, UUV	Recommended option for testing.	► Other applications: with strong and continuous vibration	Not recommended for testing. Only when you fully understand how 'Auto Boot' works. Refer to 'Boot Mode'.	► Other applications: without strong and continuous vibration	Not recommend for testing. Only when you fully understand how 'Auto Boot' works. Refer to 'Boot Mode'.	► Compatible Mode: Use legacy Firmware V3.x.x default configuration (Lower Performance)	Lower performance. Not recommended for high demanding usage.
Options of Default Settings	Comment															
► Ground vehicle: Car, AGV, Robot	Recommended option for testing.															
► Flying vehicle: Drone, UAV	Recommended option for testing.															
► Marine application: Boat, Ferry, UUV	Recommended option for testing.															
► Other applications: with strong and continuous vibration	Not recommended for testing. Only when you fully understand how 'Auto Boot' works. Refer to 'Boot Mode'.															
► Other applications: without strong and continuous vibration	Not recommend for testing. Only when you fully understand how 'Auto Boot' works. Refer to 'Boot Mode'.															
► Compatible Mode: Use legacy Firmware V3.x.x default configuration (Lower Performance)	Lower performance. Not recommended for high demanding usage.															

6	Save Settings and wait for 3 seconds.	<p>Save the settings by clicking the big 'Save Settings' button.</p> 
7	Power off TransducerM	

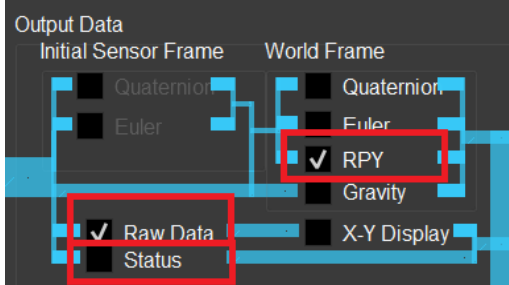
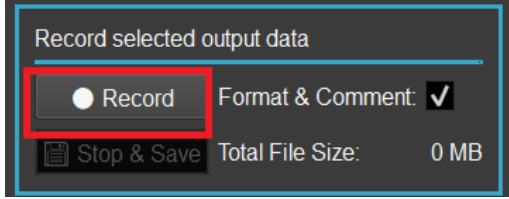
Step 2 – Heat Up

When the 'Step 1 – Restore Default Setting' is completed, continue with the following procedures, which allows and makes sure that TransducerM is ready to deliver its optimal performance.

Procedure	Instruction	Comment
1	TransducerM power on	If it is already powered on, restart it by power cycling it.
2	Wait 15 seconds	To ensure the initialization procedure finishes.
3	Open ImuAssistant, scan and open TransducerM	You will see the output data being displayed on ImuAssistant after opening.
4	Enable the 'Status' output in the ImuAssistant	<p>To enable the 'Status' output, check it as shown below:</p> 
5	Put TransducerM on static ground. Keep it completely stationary and do not move it.	
6	Wait for approximately 5 minutes, until ImuAssistant shows QoS (Quality-of-Service) as ' 5 Very Good '. Heat up done.	<p>QoS information can be found in the top-right corner of ImuAssistant:</p> 
7	Keep TransducerM and ImuAssistant running, and perform the next step.	 <p>If TransducerM is powered off accidentally, repeat all the procedures in section 'Step 2 – Heat Up'. It is extremely important for QoS to reach the fifth level.</p>

Step 3 – Record data

If you would like to record TransducerM testing output data into log files saved on your PC, perform this step.

Procedure	Instruction	Comment
1	Disable the 'Status' output in the ImuAssistant	We don't want to record 'Status' data, instead, what we are interested in are the 'Raw Data' and 'RPY'. Disabling unwanted output data speeds up the refreshing rate of useful data.
2	Make sure 'Raw Data' and 'RPY' is selected	The configuration should now look like this: 
3	Click the 'Record' button on the bottom-right of the ImuAssistant	
4	Select the file saving directory	It is recommended to select file locations such as 'My Document', instead of the application installation path, to avoid possible 'writing permission forbidden' issue.

Step 4 – Run the Test

If TransducerM is attached to a vehicle, align the X-Axes of TransducerM with the front direction of the vehicle.

You can now move and rotate the TransducerM or drive the vehicle with TransducerM attached. All selected data outputs are recorded, which can be used for later analysis. The Total File Size should accumulate over time.

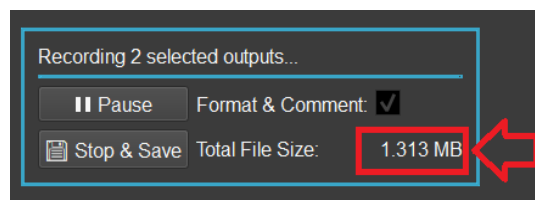


Figure 21: Data recorder panel during the test

Step 5 – Save the Recorded Data and Finish

Click the 'Stop & Save' button at the end of the test.

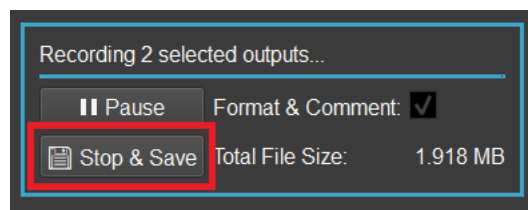


Figure 22: Data recorder saves and closes files

For more details regarding the data recorder, refer also to section 'Data Recorder' on page 23.


In-depth Description

Before going through this section, please make sure that you have also read the previous sections since some basic usage instructions are only available there.

TransducerM – Node ID, Firmware Version, UUID

When your device is discovered and made visible in the list, you can click the arrow (Figure 23, red square) left from the 'Node ID' in order to get more information about this particular device.

Here you can firstly see 'Node ID', which is a shortened unique ID of the device on current port (unlike 'UUID'). To the right of it, there's a device name, which allows for giving a device a more user-friendly label (up to 12 characters) that can be used when working with bigger networks of TransducerM.

Both properties ('Node ID' and 'Name') are customizable and can be changed by clicking the property and then  button placed above the list or slow-double-click the property itself.

In the next two lines, you can see the 'UUID' of the node, a specific set of characters unique to every device; and 'Firmware' version currently running inside the TransducerM.

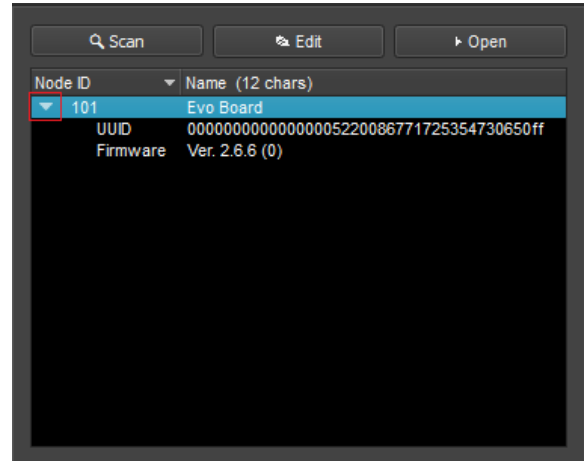


Figure 23: Device information

ImuAssistant – Version Number

To check the version number of ImuAssistant GUI software, click the '...' button located on the bottom-left corner of ImuAssistant, as shown in Figure 24.



Figure 24: ImuAssistant about button

A pop-up window will show up. Figure 25 is an example, saying ImuAssistant version number is V3.6.8.

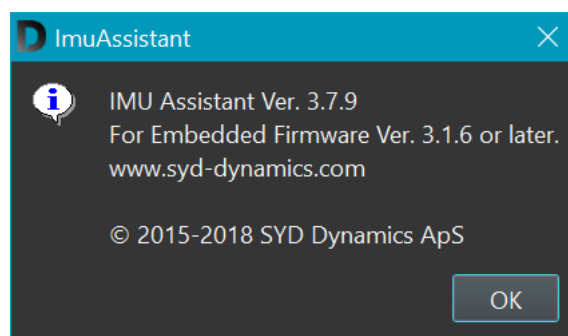


Figure 25: ImuAssistant version number

Enable and Disable Sensors

In section number 1 of the data-manipulation portion of the user interface, shown in Figure 13, it starts by selecting which of the sensors are going to be included in sensor fusion.

'Enable Gyro' refers to 'Enable Gyroscope Sensor'.

'Enable Accel' refers to 'Enable Accelerometer'.

'Enable Mag' refers to 'Enable Magnetometer'.

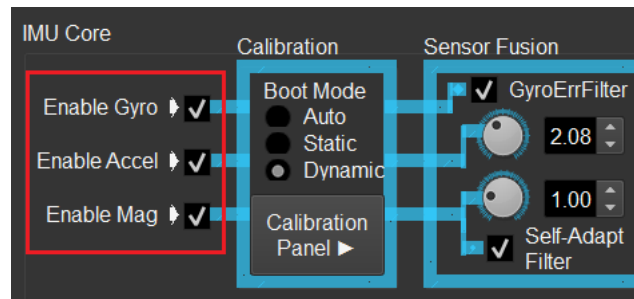


Figure 26: Data path (partial, enable/disable sensors)

Normally, gyroscope and accelerometer should always be enabled to provide necessary performance.

However, magnetometer should be turned off if the user scenario involves extremely complex and strong magnetic interference. Please note that, after disabling the 'Enable Mag' option, magnetometer will not be directly used for sensor fusion after boot is completed. However, this option does not affect Raw Data output nor the sensor fusion algorithm during boot time.

Boot Mode

Boot Mode defines how the TransducerM behaviors when it is powered on, before entering its normal operation conditions. You can change the 'Boot mode' option, as shown in Figure 27, within the Calibration box under tab named 'B' (B means 'boot').

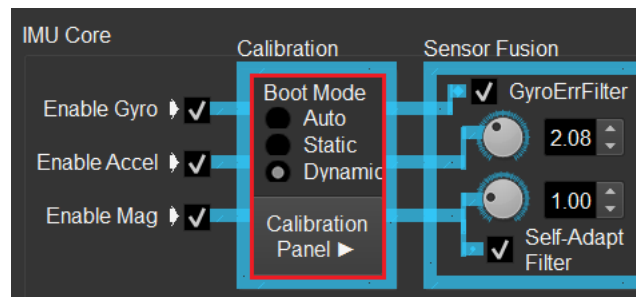


Figure 27: Data path (partial, boot mode and calibration panel)

If '**Static Mode**' is chosen, you have to keep the module stationary for at least 13.5 seconds during and after power on. Static mode will provide relatively good performance possible by TransducerM immediately after. Note that the module should be sat completely static during boot; any vibration during the 13.5 seconds will significantly reduce the performance.

If '**Dynamic Mode**' is chosen, you can power on the TransducerM in a non-static environment, such as on a moving vehicle or boat. The module will load the calibration data from the latest reliable calibration. Please note that, if the environment changes significantly, for example temperature changes from 10°C outdoor to 28°C indoor, the calibration record may not be reliable enough. When this happens, either of the following two methods can be used to improve the performance:

1. Keep the module stationary as required in the 'Static Mode' in the new environment for a while and until the yaw heading drift be suppressed to a reasonable level;
2. Keep the module stationary as required in the 'Static Mode' in the new environment and then click 'Calibration Panel' button to open the panel. Inside the panel, click the 'CalibB' button. Wait for 20 seconds to obtain the newest calibration data and burn it into Flash Memory of TransducerM.



Please make sure power supply of TransducerM is stable while re-calibrating the module by clicking the 'CalibB' button. Unstable power supply, and particularly a power-off event during the 20-seconds calibration period, may result in unrecoverable damage to the module.

If 'Auto Mode' is chosen, the module will monitor the power-on environment. When the platform is stationary, the module will switch to 'Static Boot', otherwise 'Dynamic Boot' will be used. Note that under rare conditions, the module makes bad decisions. 'Auto Mode' may also take longer time to boot. As such, it is recommended to explicitly tell the module which mode to boot if the user scenario is known.

Calibration Panel

The calibration panel can be opened through the button as shown in Figure 28. It will then pop-up in section 3 shown in Figure 13.

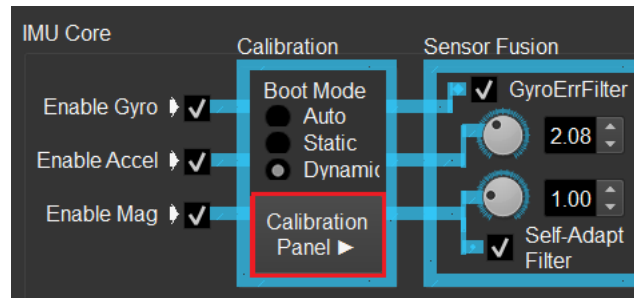


Figure 28: Button to open Calibration Panel

Depending on the TransducerM model and firmware version, you will see different configurations in the pop-up panel. For example, a minimum setup includes a static calibration 'CalibB' button. Shown in Figure 29. The use case of the 'CalibB' has been described in section 'Boot Mode' on page 17. For other options in the panel, they are usually special controls for a particular customer type and there should be descriptions in the user interface. Please also contact us for Technical Support (page 46) in case of any doubt.

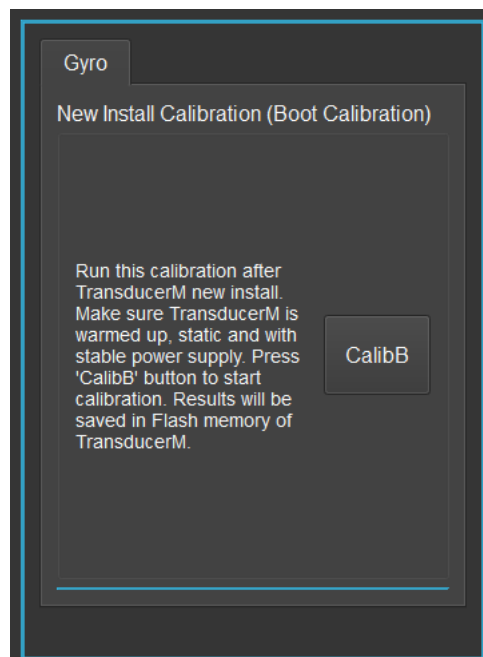


Figure 29: Calibration Panel minimum setup

Sensor Fusion

Sensor output then continues to 'Sensor fusion', an internal algorithm used to combine sensor data to provide stable output.

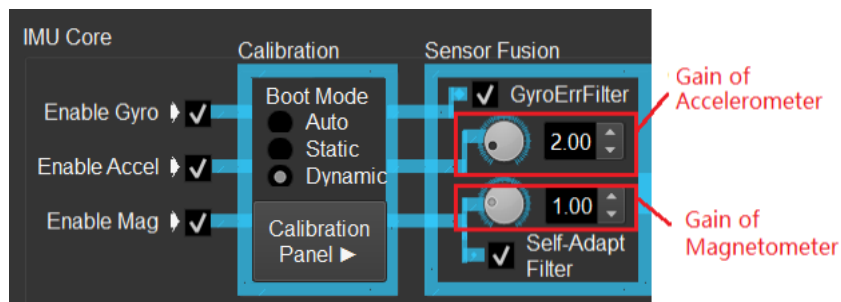


Figure 30: Data path (Partial, Sensor Fusion)

Inside the 'Sensor fusion' block, there are:

- '**GyroErrFilter**': a functionality only available with Firmware Version V4.x.x (3) and later which compensates for gyroscope error during run time.
- Two parameters (**knobs**) which allow you to increase the contribution of Accelerometer and Magnetometer to the output of the algorithm in terms of allowing you to customize the behavior based on your application.

We can reset both parameters to their respective optimal values for a particular user scenario by clicking the 'Use Default Setting' button (please also see section 'Step 1 – Restore Default Setting' on page 13).

Generally, larger values indicate that the sensor fusion algorithm should trust more on either accelerometer or magnetometer, or both. When '**GyroErrFilter**' is turned on, it is strongly recommended to have the gain of accelerometer no less than 2.0. For applications with strong and continuous vibration, the gain of accelerometer should be at least 2.5.

- Also, you get to control '**Self-Adapt filter**', a functionality which compensates for disturbances in magnetic field to provide stable magnetic heading information.

Output Data Types

After the data has been processed, you're free to select your preferred output format. The next portion of data path allows you to output (and visualize where possible) orientation data with reference to 'World frame' (with respect to the Earth's magnetic field and Gravity direction). The output from Initial Sensor Frame is not available.

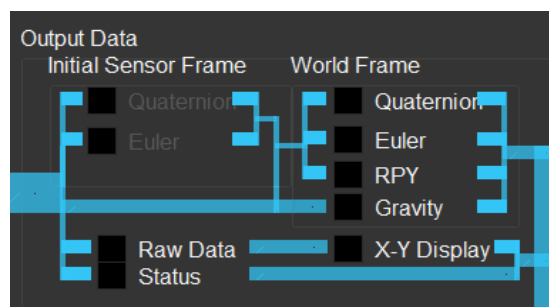


Figure 31: Configuring output of the module

Communication Protocol

The last part of the data-path refers to ‘Communication protocol’ between TransducerM and PC (or any other device acquiring data from the TransducerM).

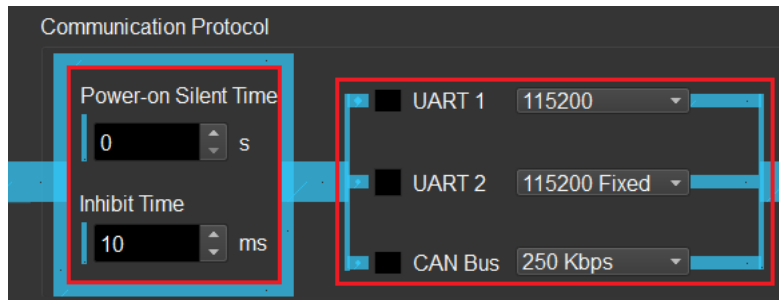


Figure 32: Configuring Communication Protocol

Here, you're able to set the following:

- **‘Power-on Silent Time’** (only available with Firmware Version V4.x.x (3) and later). This setting defines the time since power-on during which the TransducerM will not spontaneously send out any data package. This setting is useful, for example, when TransducerM is permanently connected to a PC via USB interface, where the TransducerM powers on together with the PC and where the PC is not supposed to receive any data during its booting period. However, TransducerM still responds to requests during the ‘Silent Time’. The minimum value of the Silent Time is zero, which is the default value.
- **‘Inhibit time’**: the minimum time interval between two data packages. It is useful to avoid possible overwhelming of the host device who is receiving TransducerM data. This setting only applies to data packages sent out spontaneously from the TransducerM (i.e. do not apply to data packages responding to requests). A smaller inhibit time value (can be as small as zero) results in higher output data package rate. On Window PC and while using ImuAssistant to display data, it is recommended to have an inhibit time larger than 8 ms so that the PC has enough time to respond to each data package received.
- **Port to be used** (UART/USB/CAN Bus) and their respective baud rate (data transfer rate). Note that the UART1 is the main UART interface, while UART 2 is only available on customized version TransducerM. For details regarding changing of UART baudrate, please refer to section ‘Change UART Baudrate’ on page 20.

Change UART Baudrate

The data rate of UART1 can be changed by selecting the options as shown in Figure 33.

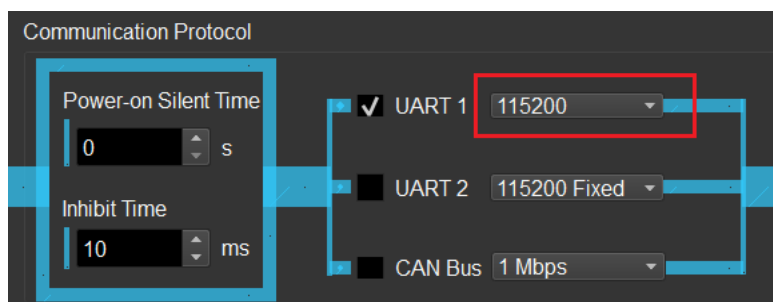



Figure 33: Configuring UART 1 baudrate

If you choose a data rate other than 115200 bps, since the ImuAssistant works with 115200 bps by default, you will experience difficulties connecting to the module the next time. The solution is to simply execute the  **Scan** as described in 'Finding Your Device' on page 9 within 6 seconds after powering on the TransducerM, under which circumstance the module will lock its UART1 data rate to 115200 bps for the current session, and the communication between the module and ImuAssistant will go normally.

Increase Output Rate

To increase the output data rate (i.e. the number of data packages per second received by the host reading TransducerM), simply do the following:

- Use a **higher UART baudrate** (such as 921600 bps).
- **Reduce 'Inhibit Time'** (can be as small as zero). The 'Inhibit Time' specifies the minimum time interval between two data packages. Please refer to the section 'Communication Protocol' on page 20 for more details.
- Save the setting and **restart TransducerM**. If you need to reconnect TransducerM to the ImuAssistant after changing the baudrate to a value higher than 115200 bps, please refer to the section 'Change UART Baudrate' on page 20.

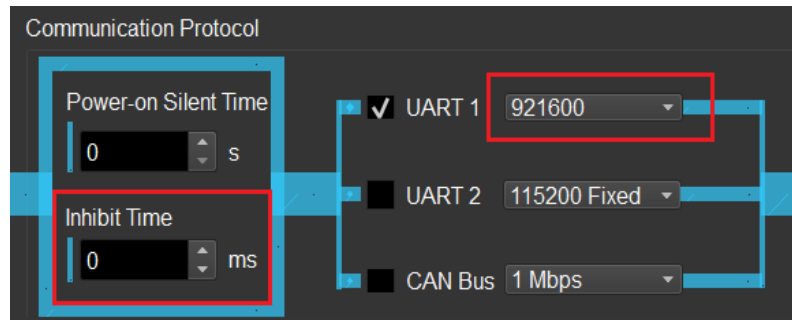


Figure 34 Example Setting to Increase Output Rate

Figure 34 shows a typical setting where around 200~300 Hz total output rate can be achieved. If not, a buffer overflow maybe occurred in your host system, please slightly adjust your code according to the section 'Avoid Buffer Overflow' on page 27.

You can also try to increase the baudrate further and get even faster output rate. However, a higher baudrate, when using long connection wires, makes the system more susceptible to external electronic magnetic interference. Normally, around 1Mbps should be enough.

The output bandwidth is shared among different data types selected. For example, if you have 300Hz total output rate, with only 'Roll/Pitch/Yaw' data type selected for output, you get full 300Hz; if you both 'Roll/Pitch/Yaw' and 'Raw Data' are selected, you get 150 Hz for each data type, and so on.



Please note that, when using a USB-to-Serial converter connecting TransducerM to a Windows PC, you only get around 88Hz maximum data rate (Inhibit Time: 10 ms, Baudrate: 115200 bps); output rate higher than that will usually cause package loss due to the limitation of the Windows serial driver. Linux or any other embedded system such as micro-controller typically will not have such a problem.

Roll Pitch Yaw Display

Please refer to section 'Roll Pitch Yaw' on page 12.

Quaternion Display

Please refer to section 'Quaternion' on page 11.

Raw Data Display

Please refer to section 'Raw Data' on page 11.

Note that the raw data is calibrated sensor data without applying any data processing filter.

X-Y Display

X-Y Display plots calibrated RAW sensor data from multiple axes of Accelerometer and Gyroscope into a single 2D plot (one per instrument). For it to work, it is necessary to have the 'Raw data' output enabled, as shown in Figure 36.

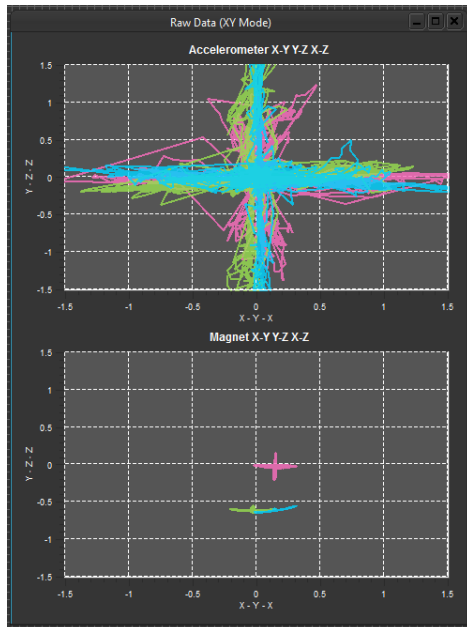


Figure 35: RAW data 'X-Y Display' visualization

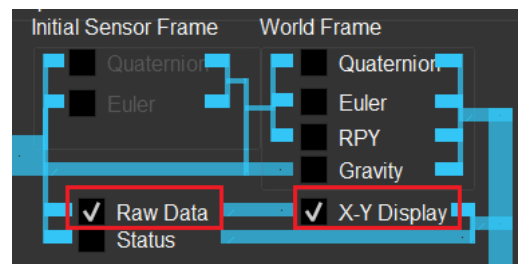


Figure 36: X-Y display output configuration

Save the Setting

Once you are satisfied with the configurations, click the 'Save Settings' button as shown in Figure 13 section number 1. This will make the settings permanent by saving the configurations into the flash inside the TransducerM.

Please also refer to the 'Save Setting' section on page 12.

Export Communication Library

To embed the TransducerM into your own system, you need a communication library installed that talks to the module. The library is provided by SYD Dynamics in the form of C/C++ in source code which only uses general features of the programming language, meaning it can be ported and deployed into your target system easily.

To acquire the communication library, simply extract the provided zip file comes together with this document.

Please refer to the section 'Use SYD Dynamics Communication Library' on page 25 for instructions on how to use the communication library.



The feature of exporting code library (shown in Figure 37) from GUI is only available on ImuAssistant version 3.8.1 and later. When having earlier versions, please use the code library zip files come along with this document or contact SYD Dynamics for a more recent version ImuAssistant.

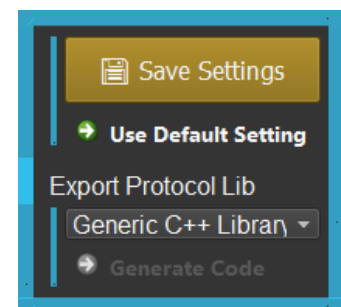


Figure 37: Save and Export

Data Recorder

The ImuAssistant has build-in data recorder function. TransducerM output displayed in ImuAssistant can be logged into files.

Select Data Types to be Recorded

To use the data recorder, firstly we need to select the data types to be logged. This is done by enabling one or more output types in the data-manipulation portion of the user interface, as shown in Figure 38 (the data types within the confine of the red polygon).

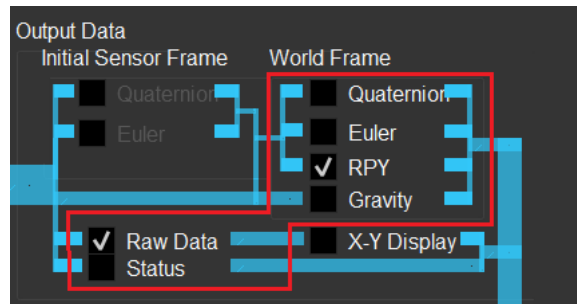


Figure 38: Data recorder step 1 - select the data types to be logged

You should then see the selected data being displayed and updated in real time in the data table on the right side of ImuAssistant (the section number 3 of GUI as shown in Figure 13). If not, please check the procedures described in section 'Finding Your Device' on page 9, and 'Communication Protocol' on page 20.

Configure Settings and Start to Record

The data recorder panel is located on the bottom-right corner of the ImuAssistant. Before recording, you can choose to enable 'Format & Comment' feature in the log files, as shown in Figure 39, which makes it easier for us to understand the meaning of each column of the log files to be generated.

Then we press the 'Record' button, as shown in Figure 39. A pop-up window appears asking the location for saving the log files. Choose a location and press the 'Save' button, the recording starts from now on.

It is recommended to select file locations such as 'My Document', instead of the application installation path, to avoid the possible 'writing permission forbidden' issue.

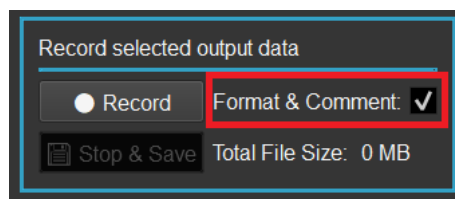


Figure 39: Data recorder step 2 - settings

Stop and Save Log Files

During the recording process, you should see the 'Total File Size' accumulates over time.

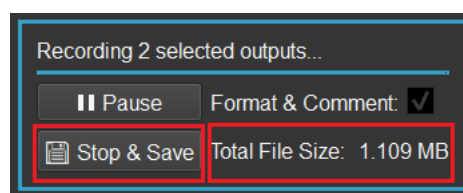


Figure 40: Data recorder step 3 - Stop & Save

Click the 'Stop & Save' button when the data recording is finished. This will make ImuAssistant flush unsaved data in the volatile memory and write into files located in the hard-drive of your computer and then close the files. The data recorder panel shows 'Files saved successfully' shortly after, as demonstrated in Figure 41, which indicates you are safe to close the ImuAssistant without losing any data.

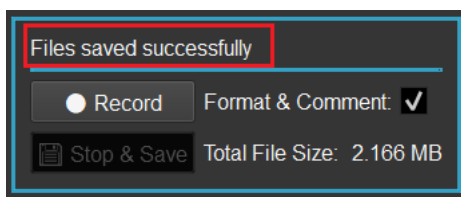


Figure 41: Data recorder step 3 - files saved successfully

The log data is saved in the file path as specified in section 'Configure Settings and Start to Record' on page 23. Figure 42 shows example data log files when RawData and Roll/Pitch/Yaw data types are selected for recording.



 DataLogin_NodeID-890_11-11-2017_18-54-46_RawData.txt
 DataLogin_NodeID-890_11-11-2017_18-54-46_RPY.txt

Figure 42: Data recorder - example of the recorded data files

Use SYD Dynamics Communication Library

With TransducerM, SYD Dynamics provides dedicated communication library in source code to ease the development effort required while integrating the TransducerM into your target systems.

C++ Library (Recommended, Full API)

Using the SYD Dynamics C++ library for TransducerM is the most recommended way to interface with TransducerM in your target applications, as it is very reliable, well tested, time saving and can also be easily upgraded to future versions whenever new APIs are released.

Figure 43 shows a typical SYD Dynamics C++ communication library in source code, which can be used independently (no need for C++ STL support, no third-party library is required in order to make it work).









 BasicTypes.h	C++ Header file
 EasyObjectDictionary.cpp	C++ Source file
 EasyObjectDictionary.h	C++ Header file
 EasyProfile.cpp	C++ Source file
 EasyProfile.h	C++ Header file
 EasyProtocol.cpp	C++ Source file
 EasyProtocol.h	C++ Header file
 EasyQueue.h	C++ Header file

Figure 43: Communication library (C++ example)

Below is an example, which will give us a glimpse on how to use the C++ communication library.

Firstly, include necessary header files:

```
// To use the communication library, we need to include the following
// two header files:
#include "libraryFolder/EasyObjectDictionary.h"
#include "libraryFolder/EasyProfile.h"
```

Instantiate the communication library:

```
// TransducerM communication library instantiation:
EasyObjectDictionary eOD;
EasyProfile          eP(&eOD);
```

Then implement a function which is called every time new serial data from the TransducerM is available:

```
/**
 * Serial Data Receive - Example
 * @note This function is called when new serial data is available
 */
void HelloMotionModule::On_SerialRX(
    char* rxData,    ///< [INPUT] Pointer to the RX data array
    int  rxSize      ///< [INPUT] Size of the RX data array
){
    Ep_Header header;
    if(EP_SUCC_ == eP.On_RecvPkg(rxData, rxSize, &header)){ // Step 2: Tell the library that new data has arrived.
        // It does not matter if the new data only contains a fraction
        // of a complete data package, nor does it matter if the data stream is corrupted
        // during the transmission. On_RecvPkg() will only return EP_SUCC_
        // when a complete and correct package has arrived.

        // Example Reading of the Short ID of the device who sends the data:
        uint32 fromId = header.fromId; // Step 3.1: Now we are able to read the received payload data.
    }
}
```

```

// header.fromId tells us from which TransducerM the data comes.

switch (header.cmd) {
// Step 3.2: header.cmd tells what kind of data is inside the payload.
// We can use a switch() as demonstrated here to do different
// tasks for different types of data.
case EP_CMD_ACK_: {
    Ep_Ack ep_Ack;
    if(EP_SUCC_ == eOD.Read_Ep_Ack(&ep_Ack)){

    }
}break;
case EP_CMD_Q_S1_E_: {
    Ep_Q_s1_e ep_Q_s1_e;
    if(EP_SUCC_ == eOD.Read_Ep_Q_s1_e(&ep_Q_s1_e)){ // Step 3.3: If we decided that the received Quaternion should be used,
// Here is an example of how to access the Quaternion data.

        float q1 = ep_Q_s1_e.q[0];
        float q2 = ep_Q_s1_e.q[1];
        float q3 = ep_Q_s1_e.q[2];
        float q4 = ep_Q_s1_e.q[3];
        uint32 timeStamp = ep_Q_s1_e.timeStamp; // TimeStamp indicates the time point (since the TransducerM powers on),
// when this particular set of Quaternion was calculated. (Unit: uS)
// Note that overflow will occur when the uint32 type reaches its maximum value.
        uint32 deviceId = ep_Q_s1_e.header.fromId; // The ID indicates the device Short ID telling which TransducerM the data comes from.

        /// @todo Use data here:
        /// ...
    }
}break;
case EP_CMD_RPY_: {
    Ep_RPY ep_RPY;
    if(EP_SUCC_ == eOD.Read_Ep_RPY(&ep_RPY)){ // Another Example reading of the received Roll Pitch and Yaw
        float roll = ep_RPY.roll;
        float pitch = ep_RPY.pitch;
        float yaw = ep_RPY.yaw;

        /// @todo Use data here:
        /// ...
    }
}break;
}
}
}

```

It is worth to mention that the communication library automatically assembles the income data into complete data packages and verifies them; only valid reading from the TransducerM is exposed to the user application.

C Library (Basic API)

SYD Dynamics also provides simplified C language version communication library in source code, which should come along in zip format file together with the C++ library.

The C library is useful when the target system only supports C language compiler or is extremely sensitive to processing speed and memory resource. An example is, if your target system is an 8-bit C51 microprocessor with only hundreds of RAM bytes, the C library would likely to fit. For all the other mainstream microprocessors, embedded Linux systems and so on, the C++ library fits and performs efficiently.



IMPORTANT: If your system supports C++ compiler, we strongly recommend the C++ library mentioned in the previous section. The C library, while having all the basic functionalities communicating with TransducerM, it does not support Mutex protection and thus is overall less reliable. It also adds difficulties for communication library upgradation whenever new APIs are released since the code is not Object Oriented.

Avoid Buffer Overflow

In the section 'Increase Output Rate' on page 21 we mentioned how to increase the data output rate of TransducerM. To be able to fully make use of the output data stream and avoid buffer overflow in your host system, the following tips are suggested.

In your code handling low-level serial data, simply add a few more calls to 'On_SerialRX' as defined in the section 'C++ Library (Recommended, Full API)' on page 25 and read out the buffer in a faster way, as shown below:

```
/**
 * Low-level Serial Port Data Receive Function
 * In a micro-controller, this is usually an interrupt function.
 */
void HelloMotionModule::On_Low_Level_Serial_Hardware_Event(){
    char* rxData;
    int rxSize;

    // Real Serial Port: Set 'rxData' to point to the low-level serial buffer and get the buffer size:
    Low_Level_Serial_Hardware_Read(rxData, &rxSize);

    // Call the Data processing method as defined in section
    // 'C++ Library (Recommended, Full API)' on page 25:
    On_SerialRX(rxData, &rxSize);

    // < WHEN AND WHY IT OVERFLOW >
    // If you still experience low receiving data rate even if you have made the changes
    // according to the section 'Increase Output Rate' on page 21,
    // this means the low-level system serial buffer is larger than the entire length of
    // a data package from TransducerM. Due to the fact that 'On_SerialRx' will return as soon
    // as a valid package is found, this will result in too much residual data in the buffer which
    // accumulates over time causing buffer overflow.
    //
    // < SOLUTION >
    // The simplest way to overcome this is by calling the 'On_SerialRX' multiple times like below
    // to make sure the processing of data within the buffer is at least not slower
    // than its accumulation:
    On_SerialRX(0, 0);
    On_SerialRX(0, 0);
    On_SerialRX(0, 0);
    On_SerialRX(0, 0);
    On_SerialRX(0, 0);
}
```

This tip is useful especially in some host system which embeds its own very large internal serial port buffer (usually the case with Linux system).

Write Your Own Communication Library

You can write your own communication library, if you are working on a target system that neither supports the C++ nor the C compiler.



IMPORTANT: If your system does support C/C++ compiler, we would strongly recommend the communication library provided by SYD Dynamics for easy maintenance and technical support. It also saves plenty time implementing your own communication library.

Protocol Overview

TransducerM communication protocol is designed with different layers, as illustrated in Figure 44. It is recommended that the same layered architecture be implemented in Host computers as well.

A host computer refers to the computer that connects to and reads TransducerM data.

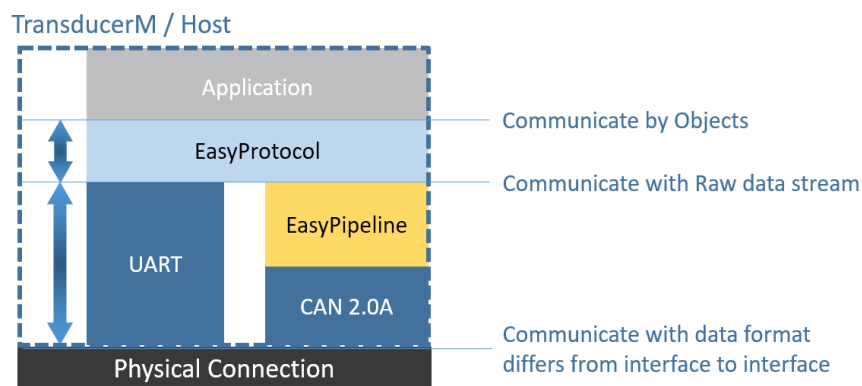


Figure 44: Communication Protocol Layers

Figure 44 is further explained as below.

Application ↔ EasyProtocol (Communicate by Objects)

On the top layer, the Application communicates with EasyProtocol layer using Objects. An Object is a compound memory item contains multiple values. For example, Roll, Pitch, Yaw and time stamp data put together in a data structure. In C language, an Object item may look like this:

```
typedef struct{
    uint32    timeStamp; // Time Stamp (uS)
    float32   roll;      // Roll (degree)
    float32   pitch;     // Pitch (degree)
    float32   yaw;       // Yaw heading (degree)
} Ep_RPY;
```

Easy Protocol ↔ UART or EasyPipeline (Communicate with Raw data stream)

The middle layer, which is called EasyProtocol, is a software implementation which turns the Object into a Data Package. A Data package is a stream of raw binary data with header, package length and checksum information. An example Raw data stream (i.e. a binary package) may look like this:

```
aa551423e04800ac9a1e83839a1c3f6b100341568d29c1170e
```

The Raw data stream can then be implemented by a layer below it.

If the layer below it is the Serial Port (i.e. UART interface or USB interface running virtual serial port profile), the Raw data stream can be transferred or acquired through it directly; however, if the layer below it is something like a CAN bus, which only supports maximum 8 bytes' payload, an additional layer called EasyPipeline is required.

The EasyPipeline layer

EasyPipeline is a software implementation of a communication protocol which turns CAN bus into a serial-bus-alike interface.

It can divide the Raw data stream into multiple segments if it exceeds 8 byte's limitation and can also assemble them upon receiving. With EasyPipeline layer, the CAN bus behaves like a serial bus, when looking from its upper layer. Therefore, different physical layers are unified.

The Physical Connection layer

This is where the actual data communication happens. Signal transmits using physical wire connections.

EasyProtocol

This section describes how EasyProtocol layer is implemented.

Overview of EasyProtocol

The EasyProtocol layer provides a service to convert between Objects and Raw data streams. As illustrated by Figure 45. A Data Package is a Raw data stream representation of an Object. Both are carrying the same main information.

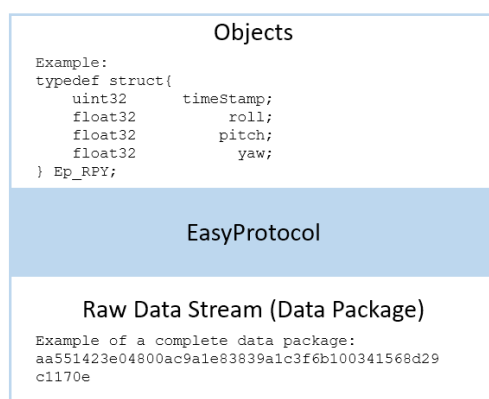


Figure 45: EasyProtocol Layer - Input Output Illustration

The Raw data stream of a complete Data Package consists of

Package Head (2 Bytes)		Package Length (1 Byte)	Payload		Checksum (2 Bytes)	
0xAA	0x55	0x04~0xff	Payload Information (4 Bytes)	Payload Content (x number of byte)	CRC first byte	CRC second byte

The **Package Head** consists of 2 bytes which have fixed values: 0xAA and 0x55 (hexadecimal).

The **Package Length** consists of 1 byte referring to the total number of bytes of the Payload fields (which includes Payload Information and Payload Content).

The **Payload Information** consists of 4 bytes, meaning 32 bits, which are divided into the following bit fields.

Payload Information Bit Fields	Bits (MSB to LSB)	Comment
Object Identifier (Data and Command Type, CMD)	7	Object type identifier. Each Object has a unique identifier.
Reserved	3	IMPORTANT: Should always be set to zero. Package with non-zero value in this field should be rejected as error might have occurred.
Source device ID of the package (FROM_ID)	11	If the package is sent from TransducerM, this field has a value equal to Node Id described in 'TransducerM – Node ID, Firmware Version, UUID' on page 16.
Destination device ID of the package (TO_ID)	11	The receiver can implement a filter and reject packages which are not addressed to itself.

For example, when using C/C++ language, the bit field can be defined as follows:

```
#define EP_CMD_BITS_          (7)
#define EP_RES_BITS_         (3)
#define EP_ID_BITS_          (11)

typedef struct{
    uint32_t      cmd   : EP_CMD_BITS_;
    uint32_t      res   : EP_RES_BITS_;
    uint32_t      fromId : EP_ID_BITS_;
    uint32_t      toId  : EP_ID_BITS_;
} Ep_Header;
```

The allocation of the 11-bit device ID is as follows:

Device ID	Description
0x0000	Broadcasting address.
0x0001	Means the address is undefined.
0x0002	Refers to the host address (usually, PC or other device reading TransducerM data can be regarded as the host).
0x0064 ~ 0x07FF	This is the normal TransducerM sensor node address range (which is the same as the Node ID specified in the ImuAssistant. Please refer to section 'TransducerM – Node ID, Firmware Version, UUID' on page 16).

The **Payload Content** is the actual content of an Object. The 'Object Identifier (CMD)' bit field of the Payload Information field specifies which Object is carried on the Payload Content. Below are two examples. For a full list of Object types, please refer to section 'Object Types' on page 31.

Object Type Name	Object Identifier (CMD) (Decimal number)
Quaternion data	32
Roll, Pitch, Yaw data	35

The **Checksum** consists of two bytes which are the last two bytes of a complete Data Package. The checksum is calculated according to the Modbus-Style 16-bit CRC checksum arithmetic.

The CRC checksum is generated from Package Length section and Payload section. (i.e. when calculating the CRC of a complete package, we should exclude the Package Head (two bytes) and the CRC (2 Bytes) fields).

The following is an example implementation of a CRC checksum generator using C/C++ language. The input is a data stream consisting of Package Length and Payload. The output is a two-byte integer (Little Endian).

```
uint16 Checksum_Generate(
    char* data,
    int  dataLength
){
    uint16 checkSum = 0;

    unsigned char*d = (unsigned char*)data;
    unsigned char c;
    checkSum = 0xffff;
    for(int i=0; i<dataLength; i++){
        checkSum ^= (unsigned int) (*(d++));
        for(int j=0; j<8; j++){
            c = checkSum & 0x0001;
            checkSum >>= 1;
            if(c) checkSum ^= 0xa001;
        }
    }
    return checkSum;
}
```

Object Types

Below lists the supported Object types of TransducerM.

About Endianness and Format

TransducerM follows Little-Endian format. The float number always consists of 4 bytes and is according to IEEE standard for floating-point arithmetic (IEEE 754). Please also refer to section 'Example of EasyProtocol' on page 41 for detailed numerical interpenetration examples.

Quaternion

Object Identifier: 32 (decimal)

Bytes	Message	Data Type	Unit	Description
0-3	Time Stamp	32-bit unsigned integer	Micro-seconds (uS)	Time stamp since TransducerM start. Caution about over-flow every 1.19 hours. When overflow occurs, the time stamp is reset to zero and then accumulates from there on.
4-7	Quaternion (q ₁)	32-bit float	N/A	(q ₁ , q ₂ , q ₃ , q ₄) forms a normalized quaternion representing the rotation from the current sensor frame (the TransducerM coordinate frame) to the earth frame.
8-11	Quaternion (q ₂)	32-bit float	N/A	
12-15	Quaternion (q ₃)	32-bit float	N/A	
16-19	Quaternion (q ₄)	32-bit float	N/A	

For C-Style programming, the Quaternion Object is defined as below

```
typedef struct{
    uint32    timeStamp; // Time Stamp (uS)
    float32    q[4]; // Quaternion data
} Ep_Q_sl_e;
```

Roll, Pitch, Yaw

Object Identifier: 35 (decimal)

Bytes	Message	Data Type	Unit	Description
0-3	Time Stamp	32-bit unsigned integer	Micro-seconds (uS)	Time stamp since TransducerM start. Caution about over-flow every 1.19 hours. When overflow occurs, the time stamp is reset to zero and then accumulates from there on.
4-7	Roll	32-bit float	Degree	-
8-11	Pitch	32-bit float	Degree	
12-15	Yaw	32-bit float	Degree	

For C-Style programming, the Roll, Pitch and Yaw Object is defined as below

```
typedef struct{
    uint32    timeStamp; // Time Stamp (uS)
    float32    roll; // Roll (degree)
    float32    pitch; // Pitch (degree)
    float32    yaw; // Yaw heading (degree)
} Ep_RPY;
```

Euler Angles

Object Identifier: 34 (decimal)

Bytes	Message	Data Type	Unit	Description
0-3	Time Stamp	32-bit unsigned integer	Micro-seconds (uS)	Time stamp since TransducerM start. Caution about over-flow every 1.19 hours. When overflow occurs, the time stamp is reset to zero and then accumulates from there on.
4-7	Psi	32-bit float	Degree	-
8-11	Theta	32-bit float	Degree	
12-15	Phi	32-bit float	Degree	

For C-Style programming, the Euler Angle Object is defined as below

```
typedef struct{
    uint32    timeStamp;
    float32    psi;
    float32    theta;
    float32    phi;
} Ep_Euler_sl_e;
```

Raw Sensor Data

Object Identifier: 41 (decimal)

Bytes	Message	Data Type	Unit	Description
0-3	Time Stamp	32-bit unsigned integer	Micro-seconds (uS)	Time stamp since TransducerM start. Caution about over-flow every 1.19 hours. When overflow occurs, the time stamp is reset to zero and then accumulates from there on.
4-7	Gyroscope X-Axes (g_1)	32-bit float	Rad/s	(g_1, g_2, g_3) represents the rotation rate of the TransducerM in its sensor frame (i.e. Coordinate system drawn on the TransducerM casing)
8-11	Gyroscope Y-Axes (g_2)	32-bit float	Rad/s	
12-15	Gyroscope Z-Axes (g_3)	32-bit float	Rad/s	
16-19	Accelerometer X-Axes (a_1)	32-bit float	g	(a ₁ , a ₂ , a ₃) represents the acceleration measured by TransducerM in its sensor frame. Gravity acceleration is part of the measurement. Conversion between g and m/s ² : 1g = 9.8158 m/s ²
20-23	Accelerometer Y-Axes (a_2)	32-bit float	g	
24-27	Accelerometer Z-Axes (a_3)	32-bit float	g	
28-31	Magnetometer X-Axes (m_1)	32-bit float	1 unit	(m ₁ , m ₂ , m ₃) represents the magnetic strength measured by TransducerM in its sensor frame. If norm(m ₁ , m ₂ , m ₃) equals to 1, this means the magnetic strength is equal to the magnetic strength measured during its factory calibration, which is the earth magnetic field itself in Denmark. As such, the absolute value of the magnetic strength is not accurate when TransducerM is moved to a different location, whereas the direction (m ₁ , m ₂ , m ₃) represents and the relative strengtheners counts for measuring the attitude or as a digital compass.
32-35	Magnetometer Y-Axes (m_2)	32-bit float	1 unit	
36-39	Magnetometer Z-Axes (m_3)	32-bit float	1 unit	

For C-Style programming, the Raw Sensor Data Object is defined as below

```
typedef struct{
    uint32    timeStamp; // Time Stamp (uS)
    float32    gyro[3]; // Rotation Rate (rad/s)
    float32    acc[3]; // Acceleration (g)
    float32    mag[3]; // Magnetometer Reading (Unit: one earth magnetic field)
} Ep_Raw_GyroAccMag;
```


Gravity

Object Identifier: 36 (decimal)

Bytes	Message	Data Type	Unit	Description
0-3	Time Stamp	32-bit unsigned integer	Micro-seconds (uS)	Time stamp since TransducerM start. Caution about over-flow every 1.19 hours. When overflow occurs, the time stamp is reset to zero and then accumulates from there on.
4-7	Gravity in X-Axes	32-bit float	g	Represents the vector of Earth Gravity relative to the TransducerM frame (sensor frame).
8-11	Gravity in Y-Axes	32-bit float	g	
12-15	Gravity in Z-Axes	32-bit float	g	

For C-Style programming, the Gravity Object is defined as below

```
typedef struct{
    uint32_t timeStamp; // Timestamp when the Gravity Vector is calculated (Unit: uS)
    float32_t g[3]; // (g[0],g[1],g[2])represents the vector of Earth Gravity in the sensor frame(Unit: g)
} Ep_Gravity;
```

Status

Object Identifier: 22 (decimal)

Bytes	Message	Data Type	Unit	Description																														
0-3	Time Stamp	32-bit unsigned integer	Micro-seconds (uS)	Time stamp since TransducerM start. Caution about over-flow every 1.19 hours. When overflow occurs, the time stamp is reset to zero and then accumulates from there on.																														
4-7	Temperature	32-bit float	Celsius	Temperature measured inside the main sensor chip. The temperature is influenced by environmental temperature, however, it does not represent the environmental temperature itself (usually higher than the actual environmental temperature due the heat generated during run-time.)																														
8-9	Internal Update Rate	16-bit unsigned integer	Hz	TransducerM internal update rate for sensor fusion.																														
10-11	System Status bit fields	16-bit unsigned integer	-	<div>Indicates more details of the internal running status of TransducerM. This message should be interpreted by bit fields. This feature is only for TransducerM with firmware version V4.7.5 (3) or higher; earlier versions will always return zero.</div> <table><tr><td>Byte 1</td><td>QoS[0]</td><td>QoS[1]</td><td>QoS[2]</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>Byte 2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> <div>bit endianness: Little-endian</div> <div>QoS[2..0] tells the Quality-of-Service of the system, which is an indication of how much performance the TransducerM delivers.</div> <div>Possible values of QoS[2..0] are</div> <table><tr><th>QoS[2..0]</th><th>Meaning</th></tr><tr><td>0</td><td>Service unavailable due to booting or restarting.</td></tr><tr><td>1</td><td>Service unavailable due to System Fault.</td></tr><tr><td>2</td><td>Limited Service – Some functions are not available. Performance limited. (e.g. Right after Dynamic Boot)</td></tr><tr><td>3</td><td>Basic Service – All functions available and provides basic performance. (e.g. Right after Static Boot)</td></tr><tr><td>4</td><td>Fine Service – All functions available and provide</td></tr></table>	Byte 1	QoS[0]	QoS[1]	QoS[2]	0	0	0	0	0	Byte 2	0	0	0	0	0	0	0	0	QoS[2..0]	Meaning	0	Service unavailable due to booting or restarting.	1	Service unavailable due to System Fault.	2	Limited Service – Some functions are not available. Performance limited. (e.g. Right after Dynamic Boot)	3	Basic Service – All functions available and provides basic performance. (e.g. Right after Static Boot)	4	Fine Service – All functions available and provide
Byte 1	QoS[0]	QoS[1]	QoS[2]	0	0	0	0	0																										
Byte 2	0	0	0	0	0	0	0	0																										
QoS[2..0]	Meaning																																	
0	Service unavailable due to booting or restarting.																																	
1	Service unavailable due to System Fault.																																	
2	Limited Service – Some functions are not available. Performance limited. (e.g. Right after Dynamic Boot)																																	
3	Basic Service – All functions available and provides basic performance. (e.g. Right after Static Boot)																																	
4	Fine Service – All functions available and provide																																	

					fine performance. The TransducerM technical specification applies when the system reaches at least at this service level.
				5	Very Good Service – All functions available and provide very good performance.
				<p>For example, for a complete Status Object data package received, such as 'aa551016ec41007e405b5c080b2642330305004062', the system status is represented by 0x05 and 0x00, where the first byte received is 0x05, which in binary format is 0000 0101 (big-endian), or 1010 0000 (little-endian), which means QoS[2..0] = 101 in binary format, which means QoS=5 (very good quality of service).</p>	

For C-Style programming, the Status Object is defined as below

```
typedef union{
    uint16 all_Bits;
    struct{
        uint16 qos    : 3;
        uint16        :13;           // Unused Bits
    }bits;
} Ep_Status_SysState;

typedef struct {
    uint32      timeStamp;           // Timestamp                (Unit: uS)
    float32     temperature;         // Sensor temperature       (Unit: Celsius)
    uint16      updateRate;          // Internal sampling rate   (Unit: Hz)
    Ep_Status_SysState sysState;
} Ep_Status;
```

Request

Object Identifier: 12 (decimal)

Bytes	Message	Data Type	Unit	Description
0	Object Identifier (CMD) of the Object requested	8-bit unsigned integer	N/A	<p>This Object is usually sent by the Host Computer to request an Object from TransducerM. The Raw data stream containing the Object responding to the request will always have the Destination device ID (TO_ID) set to the ID of the device who initiates the request.</p> <p>This request command is useful when an Object is not set to automatic output while the Host computer wishes to occasionally acquire data of the Object.</p> <p>For example, if the Host Computer needs to read the Status Object every 10 seconds. Then it can construct a request command and send to TransducerM at such frequency.</p> <p>An example request command in Raw data stream is 'aa55080c08000016000000e0ed', which means it is a broadcast command to request Status Object (Object Identifier CMD = 0x16, or 22 in decimal) from the TransducerM. After receiving the command, TransducerM will immediately send back the required Object.</p> <p>Typical broadcasting request examples: Request Status Object: aa55080c08000016000000e0ed Request Raw Data Object: aa55080c08000029000000ecf9 Request Quaternion Object: aa55080c08000020000000ef65 Request Roll-Pitch-Yaw: aa55080c08000023000000ef21 Request Gravity Object: aa55080c08000024000000ee55 Request Euler Angle: aa55080c08000022000000eedd</p>
1-3	Not used	N/A	N/A	Should be set to zero.

For C-Style programming, the Request Object is defined as below

```
typedef struct{
    uint8  cmdRequest;        // the command requested
    uint8  notUsed[3];        // the padding at the end of the structure
} Ep_Request;
```

Calibration

Object Identifier: 23 (decimal)

Bytes	Message	Data Type	Unit	Description
0	Calibration Type Identifier	8-bit unsigned integer	N/A	Specify which calibration should be performed.
1	Control Value	8-bit unsigned integer	N/A	Depending on the specified Calibration Type, the Control Value and Parameters have different meanings. Please refer to the table below.
2-3	Parameter 1	16-bit unsigned integer	Depending on the context	
4-7	Parameter 2	32-bit unsigned integer	Depending on the context	
8-11	Parameter 3	32-bit unsigned integer	Depending on the context	

For C-Style programming, the Calibration Object is defined as below

```
typedef struct {
    uint8      calibType;
    uint8      ctrlVal;
    uint16     param1;
    uint32     param2;
    uint32     param3;
} Ep_Calib;
```

When the Host Computer sends the Calibration Command Object to the TransducerM:

Calibration Name	Calibration Type Identifier (Hexadecimal)	Control Value (Hexadecimal)	Parameter 1 (Hexadecimal)	Parameter 2	Parameter 3	Meaning
New Install Calibration (Boot Calibration)	0x04	0x02	0xaa35	0	0	Start Static Calibration (the same as the CalibB button in the ImuAssistant GUI software). Refer to section 'Boot Mode' on page 17 for cautions during the calibration.
Run-time Static Calibration (Forced Calibration)	0x05	0x03	0xaa65	<p>Defines the accepted error limit. The unit is 0.01 degree/s.</p> <p>For example, by setting this parameter to 10 (decimal), this means maximum 0.1 degree/s error detected by the calibration will be compensated, otherwise the calibration result will not be used and the calibration failed.</p> <p>Possible values should be within the range of 5 ~ 500 (Decimal, both end inclusive)</p>	<p>Defines the confidence level on the vehicle static condition. The more vibration, the less confidence it should be.</p> <p>Set this value to 0 (Zero) to let TransducerM decide by itself.</p> <p>Any value between 1 and 10000 (decimal) reflects 0.01% - 100% confidence. The unit is 0.01%. When the confidence is set to 100%, it means the commander is completely sure the vehicle is very 'stationary' (i.e, stopped and exclude any vibration)</p>	<p>Start continuous forced calibration. This function is available with TransducerM with firmware version V 5.1.x and later.</p> <p>Run calibration when the vehicle is stopped for temporary.</p> <p>This feature is useful particularly for AGV applications for long term continuous operation and remove the TransducerM random drift over time.</p> <p>The calibration forces the TransducerM to recalibrate itself according to current motion, by assuming with certain level of confidence that the module is sitting "stationary".</p> <p>The module can run this calibration at the same time while it is outputting orientation data. i.e. this calibration does not affect the normal operation of TransducerM.</p>
	0x05	0x04	0xaa65	0	0	Finish continuous forced calibration.

						<p>This function is available with TransducerM with firmware version V 5.1.x and later.</p> <p>Send this command to TransducerM before the vehicle starts to move again to finish the calibration.</p> <p>If the calibration fails, the module will not adopt the result, and nothing will be changed.</p>
	0x05	0x02	0xaa65	<p>Defines the accepted error limit. The unit is 0.01 degree/s.</p> <p>For example, by setting this parameter to 10 (decimal), this means maximum 0.1 degree/s error detected by the calibration will be compensated, otherwise the calibration result will not be used and the calibration failed.</p> <p>Possible values should be within the range of 5 ~ 500 (Decimal, both end inclusive)</p>	0	<p>Start forced static calibration and finish automatically.</p> <p>This calibration forces the TransducerM to recalibrate itself according to current motion, by assuming with 100% confidence the module is sitting "stationary" and therefore correct Gyro bias error. This calibration will only return success if the rotation rate during the calibration is within a given limitation.</p> <p>The module can run this calibration at the same time while it is outputting orientation data. i.e. this calibration does not affect the normal operation of TransducerM.</p> <p>If the calibration fails, the module will not adopt the result, and nothing will be changed.</p>
	0x05	0x01	0xaa65	0	0	Revoke Forced Calibration results.

This Object will also be used by TransducerM to report its calibration status. i.e. TransducerM will use the same Calibration Object and send it back to the Host Computer (or any other device issuing the Calibration command) while it is running or has finished the calibration. The meaning of a status report is defined as follows:

Calibration Name	Calibration Type Identifier (Hexadecimal)	Control Value (Hexadecimal)	Parameter 1	Parameter 2	Parameter 3	Meaning
New Install Calibration (Boot Calibration)	0x14	0x01	Do not care	Do not care	Do not care	Static Calibration is successful and results are saved into Flash Memory of TransducerM. Power cycle TransducerM is required afterward.
	0x14	0xff	Do not care	Do not care	Do not care	Static Calibration failed. Power cycle TransducerM is required afterward.
Run-time Static Calibration (Forced Calibration)	0x15	0x00	Do not care	Do not care	Do not care	Continuous forced calibration has started and is processing and collecting data.
	0x15	0x01	Do not care	Do not care	Do not care	Calibration is successful. Calibration result is applied and saved into TransducerM RAM. The result will be lost after power cycling TransducerM. It is possible to receive this report even if ' Finish continuous forced calibration ' command has not been sent, under the condition that TransducerM determines the data collected is sufficient.
	0x15	0x05	Do not care	Do not care	Do not care	Calibration results are successfully revoked.
	0x15	0xff	Do not care	Do not care	Do not care	Calibration failed or revoke action failed.

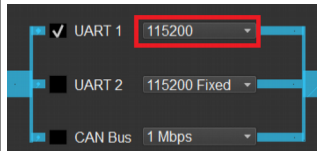
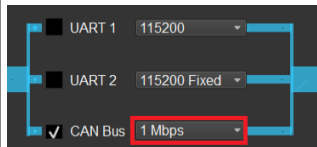
Setting

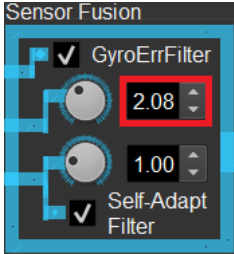
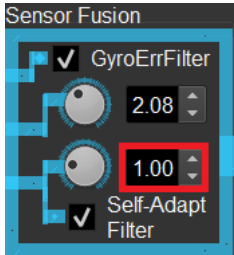
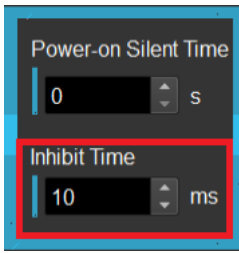
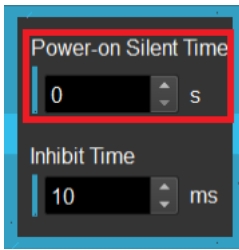
Object Identifier: 21 (decimal)

To reading the current settings from TransducerM, first send the Request Object with its 'Object Identifier (CMD) of the Object requested' field set to Setting Object which is 21. Then wait TransducerM to send back the requested Setting Object.

Send the Setting Object to TransducerM will trigger TransducerM to reconfigure itself according to the new settings.

To partially change TransducerM settings, it is a practice to firstly request Setting Object, make the modifications and then send back. This ensures only setting of interest gets changed.

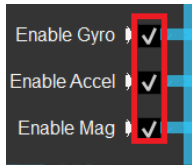
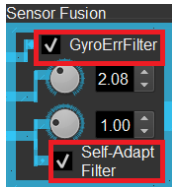
Bytes	Message	Data Type	Unit	Description	Corresponding ImuAssistant Setting GUI																						
0-3	Function enable/disable switches	32-bit unsigned integer	N/A	For configurations such as output data types, functions enable and disable. This 32-bit unsigned integer is defined by bit fields. Please refer to the table below.																							
4-5	Reserved	16-bit unsigned integer	N/A	Should always be set to value 1152. Do not change it.																							
6-7	UART 1 baudrate setting	16-bit unsigned integer	100 bps	<p>UART 1 baudrate setting. The UART 1 is the primary serial port which can be accessed from the TransducerM connector pins (RXD, TXD). Note that the unit is 100 bps. For example, when this field is set to value 1152, it means the target baudrate is 115200 bps; when set to 96, meaning 9600 bps, etc.</p> <p>Possible values:</p> <table><thead><tr><th>Value</th><th>Baudrate</th></tr></thead><tbody><tr><td>10000</td><td>1M bps</td></tr><tr><td>9216</td><td>921600 bps</td></tr><tr><td>4608</td><td>460800 bps</td></tr><tr><td>2304</td><td>230400 bps</td></tr><tr><td>1152</td><td>115200 bps</td></tr><tr><td>576</td><td>57600 bps</td></tr><tr><td>384</td><td>38400 bps</td></tr><tr><td>96</td><td>9600 bps</td></tr><tr><td>24</td><td>2400 bps</td></tr><tr><td>12</td><td>1200 bps</td></tr></tbody></table>	Value	Baudrate	10000	1M bps	9216	921600 bps	4608	460800 bps	2304	230400 bps	1152	115200 bps	576	57600 bps	384	38400 bps	96	9600 bps	24	2400 bps	12	1200 bps	
Value	Baudrate																										
10000	1M bps																										
9216	921600 bps																										
4608	460800 bps																										
2304	230400 bps																										
1152	115200 bps																										
576	57600 bps																										
384	38400 bps																										
96	9600 bps																										
24	2400 bps																										
12	1200 bps																										
8-9	CAN Bus baudrate setting	16-bit unsigned integer	100 bps	<p>CAN Bus baudrate setting. Note that the unit is 100 bps. For example, when this field is set to value 10000, this means the target baudrate is 1000000 bps (1M bps); when set to 5000, meaning 500 K bps.</p> <p>Possible values:</p> <table><thead><tr><th>Value</th><th>Baudrate</th></tr></thead><tbody><tr><td>10000</td><td>1M bps</td></tr><tr><td>5000</td><td>500 K bps</td></tr><tr><td>2500</td><td>250 K bps</td></tr><tr><td>1250</td><td>125 K bps</td></tr><tr><td>625</td><td>62.5 K bps</td></tr></tbody></table>	Value	Baudrate	10000	1M bps	5000	500 K bps	2500	250 K bps	1250	125 K bps	625	62.5 K bps											
Value	Baudrate																										
10000	1M bps																										
5000	500 K bps																										
2500	250 K bps																										
1250	125 K bps																										
625	62.5 K bps																										

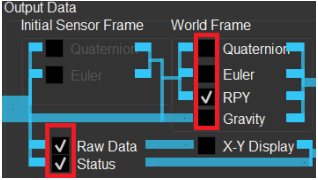
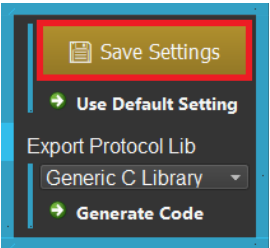
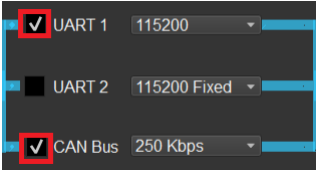
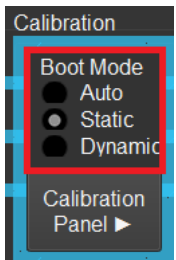
10-11	Gain for Accelerometer	16-bit unsigned integer	0.01	Gain for Accelerometer. For example, 210 (recommended value for ground vehicle) stands for 2.10. 321 stands for 3.21. Refer to ‘Sensor Fusion’ section on page 19 for more details.	
12-13	Gain for Magnetometer	16-bit unsigned integer	0.01	Gain for magnetometer. For example, 100 (default value) stands for 1.00, 123 stands for 1.23. Refer to ‘Sensor Fusion’ section on page 19 for more details.	
14-15	Inhibit Time setting	16-bit unsigned integer	milliseconds	Value range: 0 – 65530 (both ends inclusive) For example, 200 means 200 milliseconds.	
16-19	Silent Time setting	32-bit unsigned integer	seconds	Value range: 0 – 65 (both ends inclusive) For example, 8 means 8 seconds.	



Please DO NOT change settings values that are marked by comment “*should be set to..., do not change...*” or similar wordings, as changing of these reserved fields may result in unexpected behavior of the sensor module. In some cases, the default values (including allowed value range) are for forward compatibility purpose, so please do not set to other values.

The 32-bit ‘Function enable/disable switches’ is defined according to bit fields as below

Payload Information Bit Fields	Bits (MSB to LSB)	Comment	Corresponding ImuAssistant Setting GUI
Enable Gyroscope	1	Enable Gyroscope (1:Enable 0:Disable)	
Enable Accelerometer	1	Enable Accelerometer (1:Enable 0:Disable)	
Enable Magnetometer	1	Enable Magnetometer (1:Enable 0:Disable)	
Enable Gyro Error Filter	1	1:Enable 0:Disable	
Enable Magnetometer Self-Adapt Filter	1	1:Enable 0:Disable	

Output Status data continuously	1	Status output (1:Enable 0:Disable)	
Output raw sensor data continuously	1	Raw Data output (1:Enable 0:Disable)	
Reserved	1	Should always be 0 (zero). Do not change it.	
Output Quaternion continuously	1	Quaternion output (1:Enable 0:Disable)	
Reserved	1	Should always be 0 (zero). Do not change it.	
Output Euler angles continuously	1	Euler angle output (1:Enable 0:Disable)	
Output Roll-Pitch-Yaw continuously	1	Roll-Pitch-Yaw output (1:Enable 0:Disable)	
Output Gravity vector continuously	1	Gravity vector output (1:Enable 0:Disable)	
Reserved	1	Should always be 1. Do not change it.	
Reserved	3	Should always be 2 (i.e. in binary format 010). Do not change the value.	
Request acknowledge	1	If set to 1, the TransducerM will send an acknowledge message back to the host. Please refer to 'Acknowledge' section on page 40 for the definition of Acknowledge Object.	
Save setting to flash memory	1	<p>1: If it is required to save the new settings carried by this Setting Object data package into TransducerM Flash Memory in order to make the changes permanent, then set this bit to 1. Please make sure the power supply is stable in case of damaging (cannot be recovered) the TransducerM internal flash memory.</p> <p>0: Apply the new setting for the current session only (new setting will be saved into RAM and will be lost after power cycle. This is, however, safer than programming the flash memory).</p>	
Enable continuous data output through UART 1	1	1:Enable 0:Disable	
Reserved	1	Should always be 0 (zero). Do not change it.	
Enable continuous data output through CAN bus	1	1:Enable 0:Disable.	
Reserved	1	Should always be 0 (zero). Do not change it.	
Reserved	1	Should always be 0 (zero). Do not change it.	
Boot mode selection	2	<p>Boot Mode selection: Auto Boot Mode: 0 (in binary: 00) Static Boot Mode: 1 (in binary: 01) Dynamic Boot Mode: 2 (in binary: 10)</p> <p>To make boot mode setting effective, 'Save setting to flash memory' bit field (see above, within the same table) should be set to 1 as boot mode only takes effect after power cycle.</p>	
Unused bits	6	Unused bits, should always be set to 0 (zero).	



Please DO NOT change settings values that are marked by comment “*should be set to..., do not change...*” or similar wordings, as changing of these reserved fields may result in unexpected behavior of the sensor module. In some cases, the default values of the reserved bits are for forward compatibility purpose, so please do not set to other values.

For C-Style programming, the Setting Object is defined as below

```
typedef union{
    uint32 all_Bits;

    struct{
        // Sensor Selection:
        uint32 enable_Gyro           : 1;    // Enable Gyroscope      (1:Enable  0:Disable)
        uint32 enable_Acc            : 1;    // Enable Accelerometer  (1:Enable  0:Disable)
        uint32 enable_Mag            : 1;    // Enable Magnetometer   (1:Enable  0:Disable)

        // Sensor Fusion Setting
        uint32 enable_GyroErrFilter   : 1;    // GyroErrFilter         (1:Enable  0:Disable)
        uint32 enable_MagSelfAdaptFilter : 1;  // Self-Adapt Filter     (1:Enable  0:Disable)

        // Settings for continous output:
        uint32 output_Status          : 1;    // Status Output         (1:Enable  0:Disable)
        uint32 output_Raw_GyroAccMag  : 1;    // Raw Data Output       (1:Enable  0:Disable)
        uint32 reserved1              : 1;    // Should always be 0 (Zero). Do not change it.
        uint32 output_Q_s1_e          : 1;    // Quaternion Output     (1:Enable  0:Disable)
        uint32 reserved2              : 1;    // Should always be 0 (Zero). Do not change it.
        uint32 output_Euler_s1_e      : 1;    // Euler Angle Output    (1:Enable  0:Disable)
        uint32 output_RPY             : 1;    // Roll-Pitch-Yaw Output (1:Enable  0:Disable)
        uint32 output_Gravity         : 1;    // Gravity Vector output (1:Enable  0:Disable)

        // Communication Setting:
        uint32 reserved3              : 1;    // Should always be 1. Do not change it.
        uint32 reserved4              : 3;    // Should always be 2 (010 in binary).Do not change it.
        uint32 request_acknowledge    : 1;    // If set to 1, the module will acknowledge to the host
                                         // after receiving the setting.

        // Other Settings:
        uint32 save_Setting_Permanently : 1;  // If set to 1, new settings will be
                                         // saved into TransducerM's flash memory.
                                         // If set to 0, the setting is saved in RAM
                                         // and will be lost after power cycle.

        uint32 enable_UART1           : 1;    // Enable UART 1 Output (1:Enable  0:Disable)
        uint32 reserved5              : 1;    // Should always be 0 (zero).
        uint32 enable_CAN1            : 1;    // Enable CAN Bus Interface.
        uint32 reserved6              : 1;    // Should always be 0 (zero). Do not change it.
        uint32 reserved7              : 1;    // Should always be 0 (zero). Do not change it.

        uint32 bootMode               : 2;    // Boot Mode selection:
                                         //      AutoBoot:0  StaticBoot:1  DynamicBoot:2

        uint32 unused                 : 6;    // Unused Bits, Should always be 0 (zero).
    }bits;
} Ep_Switches;

typedef struct{
    Ep_Switches    switches;
    uint16         reserved;                // Should always be 1152. Do not change it.
    uint16         uart1_Baudrate;          // Unit: 100 bps
    uint16         can_Baudrate;            // Unit: 100 bps
    uint16         gain_Acc;                // Unit: 0.01, gain for Accelerometer.
    uint16         gain_Mag;                // Unit: 0.01, gain for Magnetometer.
    uint16         inhibitTime;             // Unit: ms.
    uint32         silentTime;              // Unit: seconds.
} Ep_Settings;
```

Acknowledge

Object Identifier: 13 (decimal)

Bytes	Message	Data Type	Unit	Description
0	Object Identifier (CMD) of the Object to be acknowledged	8-bit unsigned integer	N/A	Acknowledge the reception of an Object data package. For example, when the host computer sends a Setting Object (with its 'Request acknowledge' bit set to 1) to TransducerM, after successful reception of the Setting Object, the TransducerM sends back an acknowledge message with this CMD field set to the identifier of Setting Object (decimal value 21) .
1-3	Not used	N/A	N/A	Value undefined.

For C-Style programming, the Acknowledge Object is defined as below

```
typedef struct{
    uint8      cmdAck;           // The Object Identifier of the data package it acknowledges to
    uint8      notUsed[3];       // the padding at the end of the structure
} Ep_Ack;
```

Example of EasyProtocol

In-depth Case Study

TransducerM uses Little Endian conversion when mapping data to the memory.

For example, if the Host Computer receives a complete data package stream (hexadecimal format) from TransducerM:

aa551423ec4100a0f53813afb6043f6d5200bff380994192b9

where 0xaa is the first byte received, and 0xb9 is the last byte.

From section 'Overview of EasyProtocol' on page 29 we know that,

The **orange** part '14' represents the Package Length. In this case $0x14 = 20$ (decimal), which means the content followed by 0x14 consists of 20 bytes, i.e. '23ec4100a0f53813afb6043f6d5200bff3809941' which are in total 20 bytes.

The **purple** section '23ec4100' represents the Payload Information, which is interpreted by bit fields. By converting hexadecimal '23ec4100' into binary format, we get:

Hexadecimal	0x23	0xec	0x41	0x00
Binary Format Bit endianness: Big-endian This is what we are used to when hand writing, however, this is not how it is stored in the machines' memory.	00100011 MSB → LSB Bit offset 0 → 7	11101100	01000001	00000000
Binary Format Bit endianness: Little-endian. We are using this format.	11000100 LSB → MSB Bit offset 0 → 7	00110111	10000010	00000000

Recall section 'Overview of EasyProtocol', we can now divide the bits into four bit fields:

Payload Information bit fields	Bits	Binary (Little-endian)	Convert to Big-endian	Meaning
Object Identifier (Data and Command Type, CMD)	7	1100010	0100011 (Binary) = 23 (Hex) = 35 (Decimal)	The Object contained in the Payload Content is roll-pitch-yaw data. Refer to descriptions on 'Roll, Pitch, Yaw' on page 31.
Reserved	3	000	000	This field should always be zero, otherwise ignore the data stream.
Source device ID of the package (FROM_ID)	11	11011110000	00001111011 (Binary) = 123 (Decimal)	The roll-pitch-yaw data is sent from TransducerM with Node ID 123
Destination device ID of the package (TO_ID)	11	01000000000	00000000010 (Binary) = 2 (Decimal)	The roll-pitch-yaw data is sent to the Host computer. The Host computer always has a ID number of 2.

The **grey** part 'a0f53813afb6043f6d5200bff3809941' represents the Payload Content. Now we put these bytes into a table shown as below.

Memory offset per byte	+0	+1	+2	+3	+4	+5	+6	+7	...	+12	+13	+14	+15
Content	0xa0	0xf5	0x38	0x13	0xaf	0xb6	0x04	0x3f	...	0xf3	0x80	0x99	0x41

According to descriptions on 'Roll, Pitch, Yaw' Object on page 31, the first four bytes represent an unsigned 32-bit integer

0x1338f5a0 (which is 322500000 in decimal), which means the time stamp is 322.500000 seconds.

The fifth to the eighth bytes represents an IEEE-754 Floating Point number 0x3f04b6af (which is 0.51841253 in decimal), which means the Roll angle measured is 0.51841253 degree.

The last **red** part '92b9' represents the CRC checksum number 0xb992, which is the result when apply Modbus-Style 16-bit CRC checksum arithmetic to the data array '1423ec4100a0f53813afb6043f6d5200bff3809941'.

More Examples

Below lists a few more examples of complete data packages.

Object Type Name	Data Package (Raw data stream), Hexadecimal	Meaning
Raw Sensor Data	aa552c29ec41004029706b9d66383a508faab9da33c0b9a6e74d3c0267b9bb030480bf25bfac3d4fd40fbdcd4c4a3f5dffb	Gyro X: 0.000703433 rad/s Gyro Y: -0.000325317 rad/s Gyro Z: -0.000366597 rad/s Accel X: 0.0125674 g Accel Y: -0.00565803 g Accel Z: -1.00012 g Mag X: 0.084349 unit Mag Y: -0.0351146 unit Mag Z: 0.790234 unit TimeStamp: 1802512704 uS From device with Node ID: 123
Roll, Pitch, Yaw	aa551423e04800ac9a1e83839a1c3f6b100341568d29c1170e	Roll: 0.611733 degree Pitch: 8.19151 degree Yaw: -10.597 degree TimeStamp: 2199820972 uS From device with Node ID: 568
Quaternion	aa551820e048005fa679f405db7e3f697e353a960c97bd5eaa71bdce2c	Q1: 0.995529 Q2: 0.000692344 Q3: -0.0737545 Q4: -0.0590004 TimeStamp: 4101613151 uS From device with Node ID: 568

Note:

- Package Head** is marked by **Blue**,
- Package Length** is marked by **Orange**,
- Payload Information** is marked by **Purple**,
- Payload Content** is marked by **Grey**,
- CRC CheckSum** is marked by **Red**.

EasyPipeline

This section describes how EasyPipeline layer is implemented. This Layer is designed to work with CAN Bus 2.0. For a full picture of different layers, please refer to section 'Protocol Overview' on page 28. For the electric setup of CAN Bus interface, please refer to section 'Using the CAN Bus Interface' on page 6.

Overview of EasyPipeline

The EasyPipeline layer provides a service to convert between Raw data streams and Message Segments, as illustrated by Figure 46. A Raw data stream is a binary array, which is the accepted data format of EasyProtocol layer. Message Segments are small fragments of the Raw data stream.

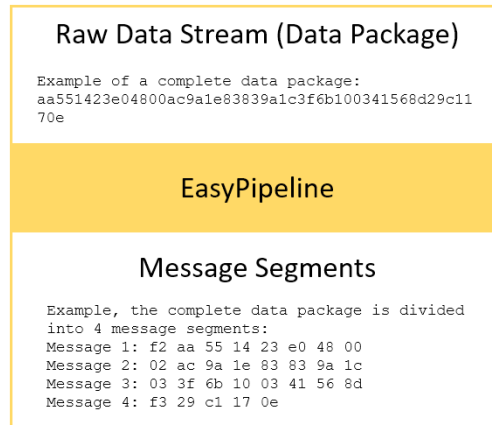


Figure 46: EasyPipeline Layer - Input Output Illustration

The **Message Segments** are constructed in a way that each segment includes a Segment Header followed by maximum 7 bytes payload. The Segment Header identifies the order number of a segment, and the payload of all the segments belonging to the same Raw data stream combined is the Raw data stream itself.

More specifically, recall that in section 'Example of EasyProtocol' on page 41, we have an example Raw data stream that contain the Roll, Pitch and Yaw Object data: **aa551423e04800ac9a1e83839a1c3f6b100341568d29c1170e**

This Raw data stream can be easily transmitted through a serial port, however, not with the case with a CAN 2.0 bus, since CAN 2.0 bus standard only supports maximum 8 bytes' payload in a message. EasyPipeline is designed to resolve this issue and make the CAN Bus a serial port alike interface for its upper layer.

The above data stream is then divided into 4 Message Segments, shown below.

	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Segment 1	f2	aa	55	14	23	e0	48	00
Segment 2	02	ac	9a	1e	83	83	9a	1c
Segment 3	03	3f	6b	10	03	41	56	8d
Segment 4	f3	29	c1	17	0e			

Byte 0 is the **Segment Header** (also called Protocol Control Information, or PCI). 0xF2 indicates this is the first segment of a multiple-segment message. 0xF3 indicates the last segment of a multiple-segment message. Any value between 0x02 and 0xEF indicates the segment order number between the first and the last segment.

Segment Header (PCI) is defined as follows:

Segment Header (PCI)	Description	Comments
0xF1	Single-segment message	Only if the Raw data stream is less than 8 bytes.
0xF2	First segment of multiple-segment message	
0xF3	Last segment of multiple-segment message	
0x02 ~ 0xEF	Number order of a segment of a multiple-segment message.	
0xF0, 0xF4~0xFF	Reserved.	

For a single-segment message, it only has one segment, and the maximum payload data size is 7 bytes. Shown as below.

	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Segment 1	0xF1	Payload[0]	Payload[1]	Payload[2]	Payload[3]	Payload[4]	Payload[5]	Payload[6]

For a multiple-segment message, it contains at least two segments, the minimum setup contains 8 bytes payload data.

	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Segment 1	0xF2	Payload[0]	Payload[1]	Payload[2]	Payload[3]	Payload[4]	Payload[5]	Payload[6]
Segment 2	0xF3	Payload[7]						

The maximum setup contains 240 segments and 1680 bytes of payload data.

	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Segment 1	0xF2	Payload[0]	Payload[1]	Payload[2]	Payload[3]	Payload[4]	Payload[5]	Payload[6]
Segment 2	0x02	Payload[7]	Payload[8]	Payload[9]	Payload[10]	Payload[11]	Payload[12]	Payload[13]
Segment 3	0x03	Payload[14]	Payload[15]	Payload[16]	Payload[17]	Payload[18]	Payload[19]	Payload[20]
...
Segment N	N	Payload [7*(N-1)]	Payload [7*(N-1)+1]	Payload [7*(N-1)+2]	Payload [7*(N-1)+3]	Payload [7*(N-1)+4]	Payload [7*(N-1)+5]	Payload [7*(N-1)+6]
...
Segment 239	0xEF	Payload [1666]	Payload [1667]	Payload [1668]	Payload [1669]	Payload [1670]	Payload [1671]	Payload [1672]
Segment 240	0xF3	Payload [1673]	Payload [1674]	Payload [1675]	Payload [1676]	Payload [1677]	Payload [1678]	Payload [1679]

Construct the CAN Bus Message

Each of the Message Segment can be transmitted by a CAN Bus message. The table below illustrates a CAN message and explains how it is related to a Message Segment of the Raw data stream of an Object.

Field Name	Identifier (COB-ID)	...	Data length code (DLC)	Data field	CRC	...
Bits	11 bits	...	4 bits	0-64 bits	15 bits	...
Value	Must be set to the same value as the Source device ID (FROM_ID) which is defined in the Payload Information section of the Raw data stream being transmitted. The definition of Payload Information can be found in 'Overview of EasyProtocol'		Specifies the number of bytes of the Data field. The value must be the actual number of meaningful bytes in the Data field.	Containing a Message Segment. A Message Segment is a fraction of a complete Raw data stream, constructed according to the rules described in 'Overview of EasyPipeline' on page 43.	Automatically implemented by CAN Bus hardware.	

	on page 29.				
	The Source device ID (FROM_ID), i.e. TransducerM Node ID, can be changed by ImuAssistant. Please refer to 'TransducerM – Node ID, Firmware Version, UUID' on page 16.				

The CAN messages should be sent by the same sequence as the segment order number, one after another. The segment which has a Segment Header of 0xF1 or 0xF2 is the first message to be transmitted.

The receiver, should decode the CAN message and assemble the segments into a complete Raw data stream. By further passing the Raw data stream up to the EasyProtocol layer, the Object data can be interpreted.

The transmitter, should create a set of CAN messages according to the rules mentioned above and then send them sequentially through the bus.

Example of EasyPipeline

Below illustrates how to interpret a Roll-Pitch-Yaw Object sent from TransducerM through CAN bus.

Assume the Host Computer receives the following CAN messages, where data fields notated by '...' are standard CAN bus formalities, which in our case are not used by EasyPipeline.

	Identifier (COB-ID)	...	Data length Code (DLC)	Data field (Hexadecimal)	CRC	...
CAN Message 1	123 (Decimal)	...	8 (Decimal)	f2aa551423ec4100
CAN Message 2	123 (Decimal)	...	8 (Decimal)	02a0f53813afb604
CAN Message 3	123 (Decimal)	...	8 (Decimal)	033f6d5200bff380
CAN Message 4	123 (Decimal)	...	5 (Decimal)	f3994192b9

We know from the COB-ID that these messages are from the same TransducerM with Node ID 123, thus they can be combined. According to the rules mentioned in 'Overview of EasyPipeline' on page 43, we get the Raw data stream:

aa551423ec4100a0f53813afb6043f6d5200bff380994192b9

The Raw data stream is constructed according to the rules mentioned in 'Overview of EasyProtocol' on page 29, which can be interpreted as below:

Package Head (2 Bytes)		Package Length (1 Byte)	Payload		Checksum (2 Bytes)	
Must be 0xAA	Must be 0x55	0x04~0xFF	Payload Information (4 Bytes)	Payload Content (x number of byte)	CRC first byte	CRC second byte
aa	55	14	23ec4100	a0f53813afb6043f6d5200bff3809941	92	B9

According to the CRC checksum rule described in 'Overview of EasyProtocol' on page 29, we confirm that the Raw data stream is complete and uncorrupted.

By using the method described in 'In-depth Case Study' on page 41, we get the following information.

Object ID: 35, which means it is Roll-Pitch-Yaw object.
FROM ID: The Object is sent from TransducerM with Node ID 123
TO ID: The Object is sent to Host Computer with fixed Node ID 2
Object time stamp: 322.500000 seconds since TransducerM start.
Roll: 0.51841253 degrees
Pitch: -0.5012577 degrees
Yaw: 19.187963 degrees

Interpretation complete.

Technical Support

For further technical support, please write emails to info@syd-dynamics.com

For faster processing, you can choose to use the following template:

Email Subject: Technical Support – (Brief description of your questions)

Email Content:

Dear support team,

I am writing in regards to the technical support for the TransducerM I have purchased:

TransducerM device information screenshot: _____

Note: Please refer to section 'TransducerM – Node ID, Firmware Version, UUID' on page 16.
A screen-shot similar to 'Figure 23: Device information' would be good enough.

Please remember to extend the window fully so that all the digits of the UUID are visible. Figure 47 demonstrates good and bad examples of the screenshots.

ImuAssistant version: _____

Note: Please refer to section 'ImuAssistant – Version Number' on page 16.

Problem description: _____

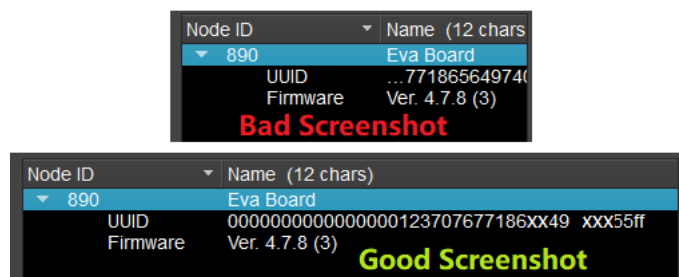


Figure 47: Screenshot of TransducerM device information