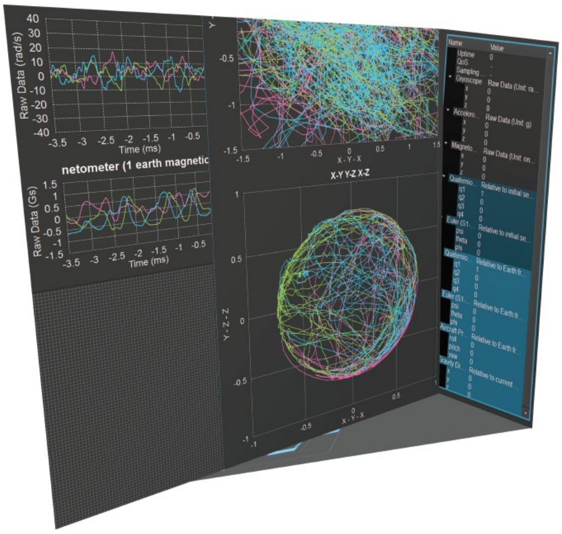
TransducerM User Guide

For product series: TM100, TM200

TransducerM is an attitude and heading reference system (AHRS) with 9-axis IMU



Version	Date	Revision Info
V1.2.8 (D)	Mar 01, 2019	Derived from UserGuide V1.2.7 (R) general version.
V1.2.9 (R)	Mar 14, 2019	Release version.
V1.3.1 (R)	Apr 16, 2019	Minor update. Release version.

** 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

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 13 and section 'ImuAssistant – Version Number' on page 13.

Firmware Version

Version Number	Comments
V15.3.2 (15)	More features are supported as the version number increases. The version number is fixed and determined by a particular TransducerM model. 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.
V15.3.x (15) (x > 2)	
V25.3.2 (15)	
V25.3.x (15) (x > 2)	

ImuAssistant Version

Version Number	Comments
V 3.8.2	ImuAssistant V3.8.2 and V3.8.x (x>2) are designed to be used with all the above mentioned firmware versions. 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 > 2)	

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.

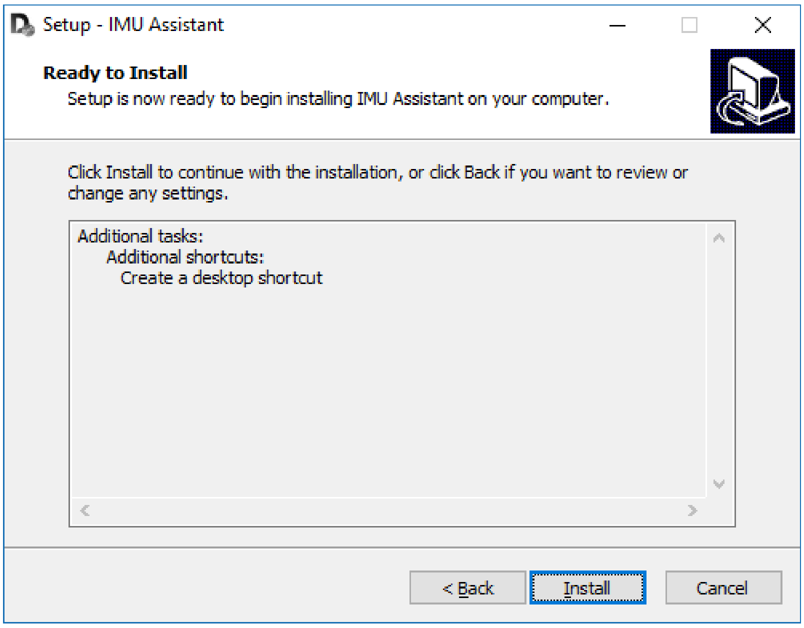


Figure 1: Software installer

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.

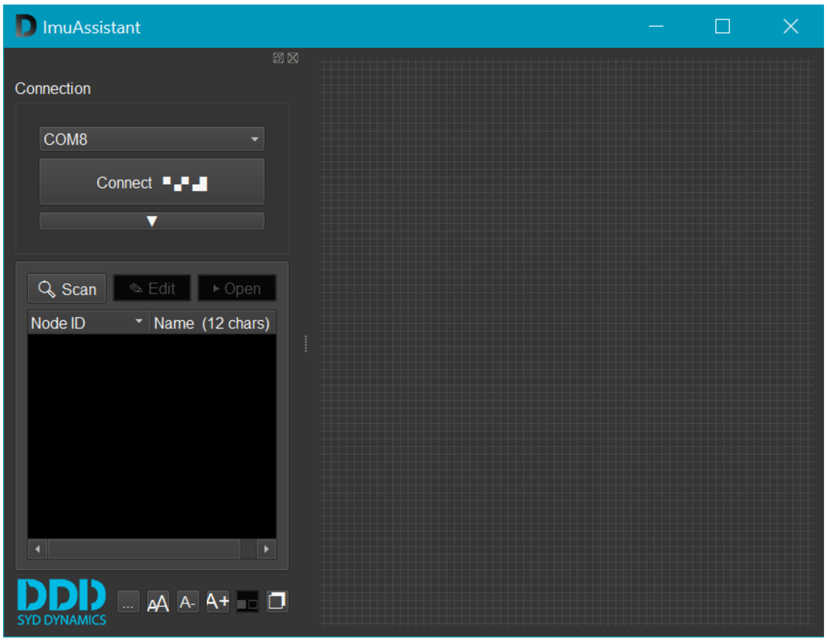


Figure 3: GUI window

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.

Please refer to the document 'TransducerM_* _Datasheet_EN_Vx-x-x.pdf' which comes along with this document for pin-to-pin definition.

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

Install TransducerM

On TransducerM, you should see axis definition similar to the one in Figure 4, 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.

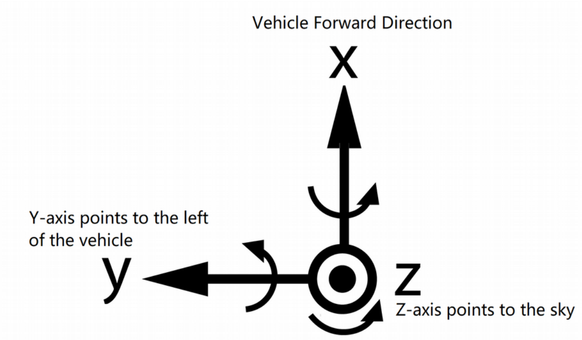


Figure 4: TransducerM Axis Definition

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 (only for TransducerM TM200 series) as illustrated in section 'Step 1 – Restore Default Setting' on page 10 to allow the TransducerM compensate for the disturbance.

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 15 seconds during and after power on. This is essential for the module to initialize and find the earth frame.

This also apples 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 TM100 PCBA Module:

The orange LED flashing indicates that the device is initializing and is not ready for normal output. After the orange LED has turned off and the green LED starts flashing, the module is then ready to output through its physical port.

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:

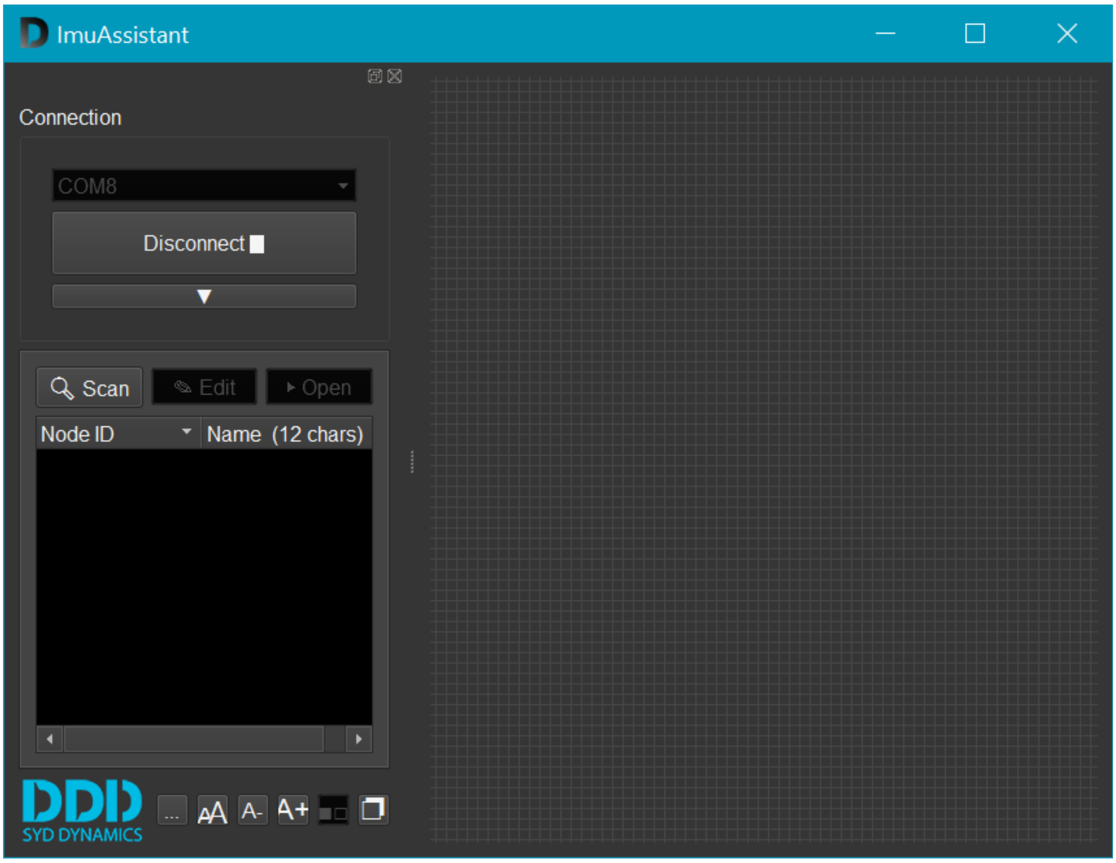


Figure 5: GUI after detecting TransducerM. Note that in this example, COM8 as auto-detected serial port

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 

button placed below the big '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.

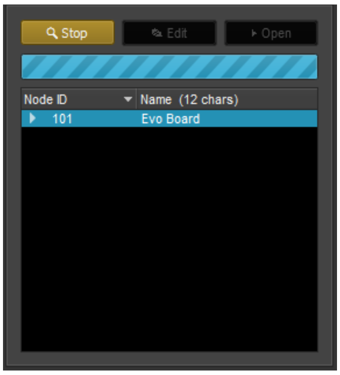


Figure 6: GUI appearance during port scan

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 7, 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 15 seconds after its power on, as the module may ignore the command from GUI during its initialization process.

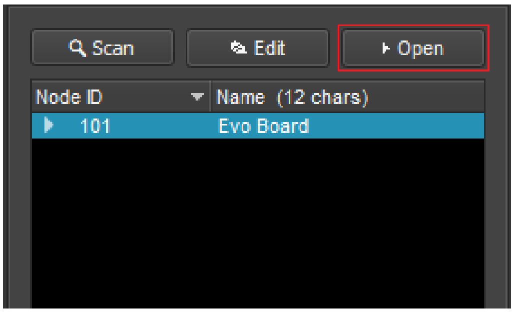


Figure 7: Opening a device

GUI Explanation

Once the device is opened, the data-manipulation portion of the user interface is visible. Shown in Figure 8 (section 1), which is a part of GUI that allows you to fully customize the data path inside your TransducerM.

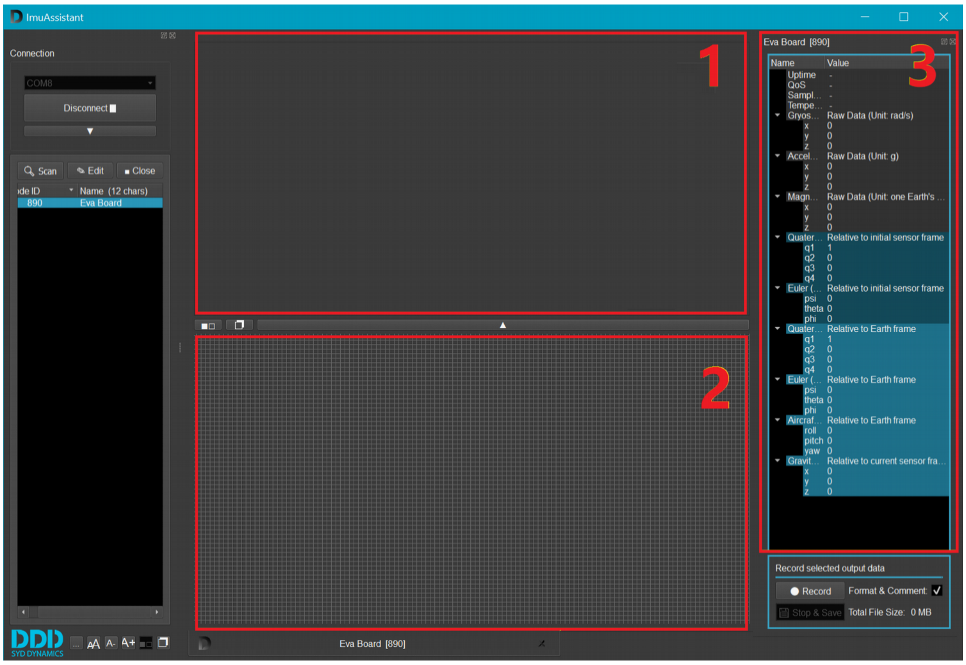


Figure 8: Fully expanded GUI
(Depending on the TransducerM model number and firmware version, you will see the panel configuration slight differently.)

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 8, while corresponding numerical data is still shown under corresponding title in section number 3 in Figure 8.

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 9). Red curve indicates X-Axis data, green for Y-Axis and blue for Z-Axis.

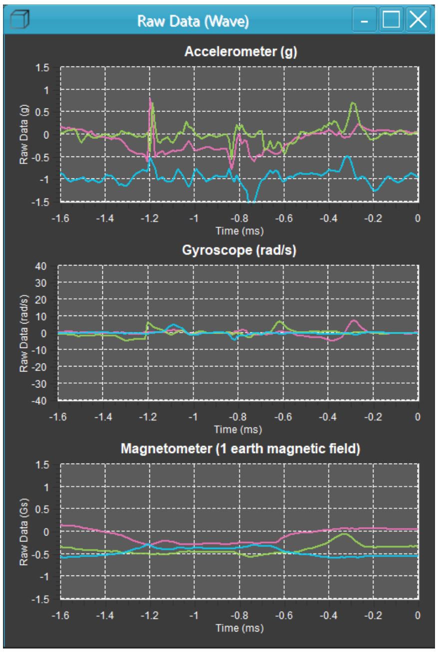


Figure 9: Raw data visualization

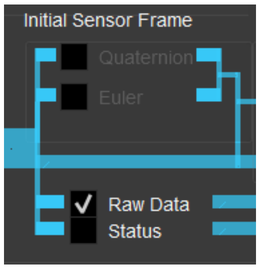


Figure 10: Output data selection

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 12). Rotating the module around any of its principal axes also changes the orientation of the cube in its 3D space (Figure 11).

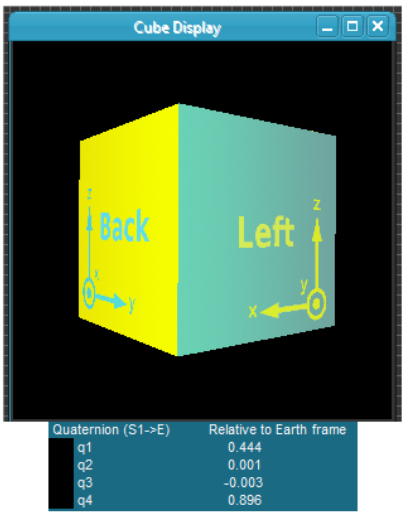


Figure 12: Initial acquired state visualization (relative to Earth's frame)

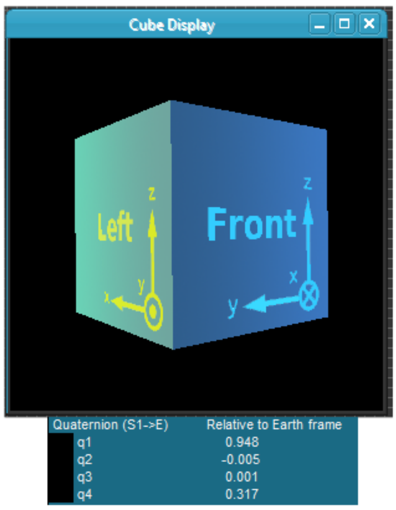


Figure 11: State visualization after rotating 90° CW (looking from top) around Z-axis

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 13. Widget on the left (Figure 13) 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 18, section 3) of GUI, under 'Aircraft Principal Axes'.

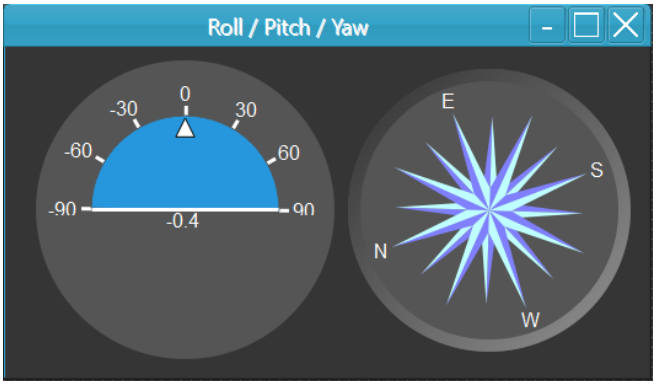


Figure 13: Roll, Pitch, Yaw visualization

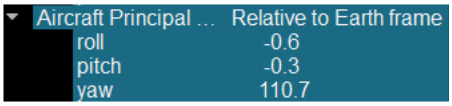


Figure 14: Roll, Pitch, Yaw numerical data

The TransducerM 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.

Save Setting

It is, however, worth mentioning that all the settings, except calibration button, selected in Figure 8 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.

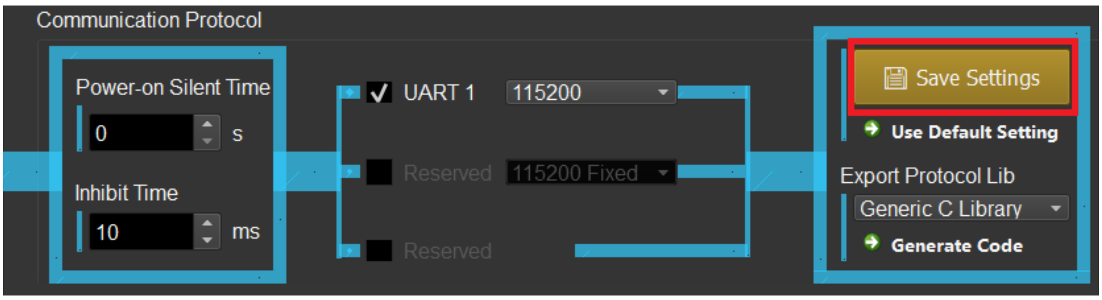
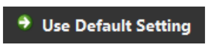


Figure 15: Configuring communication protocol

You can use the 

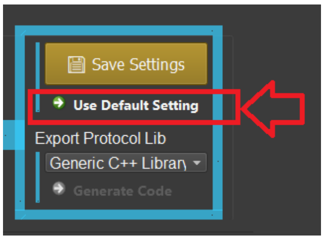
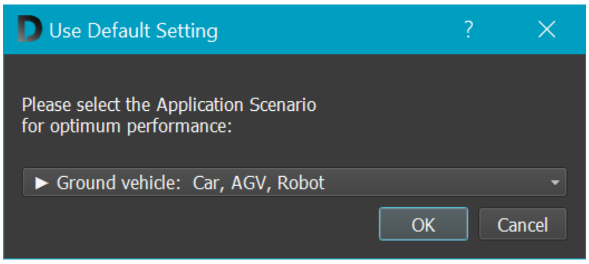
button (as shown in Figure 15) to reset the setting to its default, and then click 'Save Settings' to save the settings into the TransducerM flash memory.

Standard Setup and Test Procedure

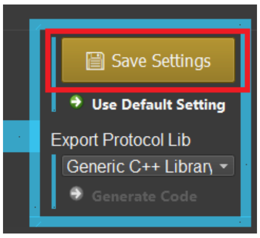
In this section, a standard procedure for testing the performance of TransducerM is described, during which you will also become familiar with the ImuAssistant build-in data recorder function.

Step 1 – Restore Default Setting

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

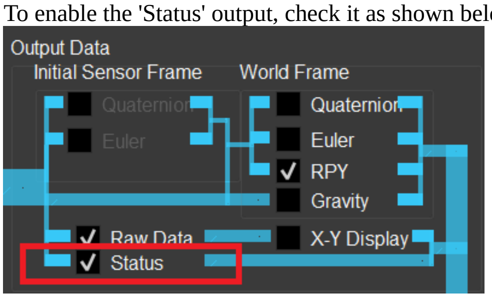
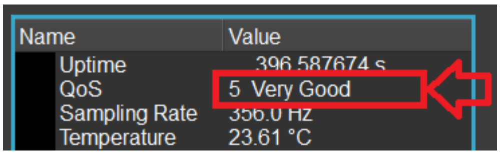
<table>
<tr><th>Procedure</th><th>Instruction</th><th>Comment</th></tr>
<tr><td>1</td><td>TransducerM power on</td><td></td></tr>
<tr><td>2</td><td>Wait 15 seconds</td><td>To ensure the initialization procedure finishes.</td></tr>
<tr><td>3</td><td>Open ImuAssistant, scan and open TransducerM</td><td></td></tr>
<tr><td>4</td><td>In ImuAssistant, Click 'Use Default Setting'</td><td>This button can be found in section number 3 of Figure 8

</td></tr>
<tr><td>5</td><td>Select the default settings for a particular user scenario.
(Only for TransducerM TM200 series)</td><td>For TM100 series, the default setting will be loaded after clicking the ‘Use Default Setting’ button.
For TM200 series, a pop-up window will show up, asking which scenario best describes the intended user case.


<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></td></tr>
</table>

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

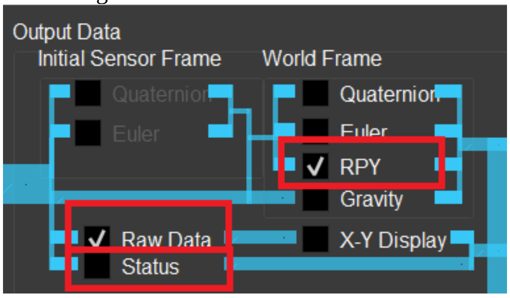
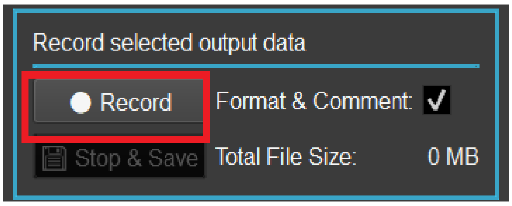
Step 2 – Heat Up

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	To enable the 'Status' output, check it as shown below: 
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.	QoS information can be found in the top-right corner of ImuAssistant: 
7	Keep TransducerM and ImuAssistant running, and perform the next step.	⚠ 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.

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.

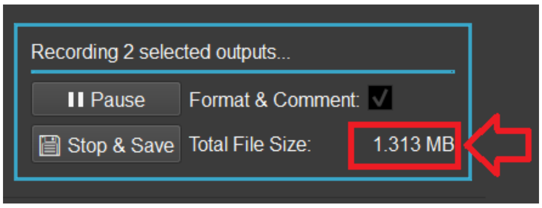


Figure 16: Data recorder panel during the test

Step 5 – Save the Recorded Data and Finish

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

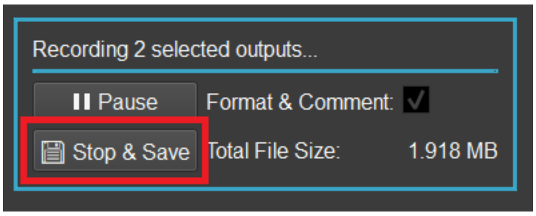


Figure 17: Data recorder saves and closes files

For more details regarding the data recorder, refer also to section ‘Data Recorder’ on page 20.

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 18, 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.

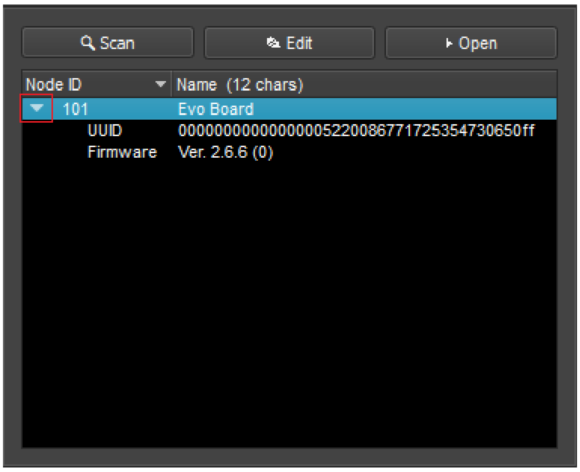


Figure 18: Device information

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 19.

Figure 19: ImuAssistant about button

A pop-up window will show up. Figure 20 is an example, saying ImuAssistant version number is V3.8.1.

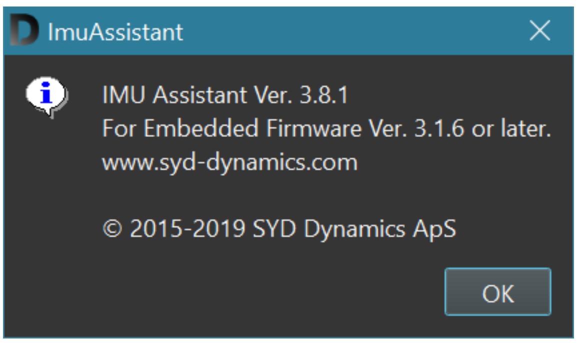


Figure 20: ImuAssistant version number

Enable and Disable Sensors

In section number 1 of the data-manipulation portion of the user interface, shown in Figure 8, 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'.

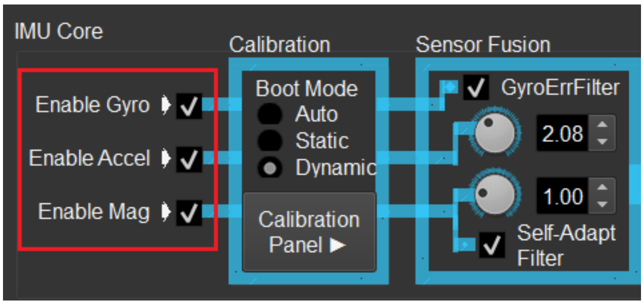


Figure 21: Data path (partial, enable/disable sensors)

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 22, within the Calibration box.

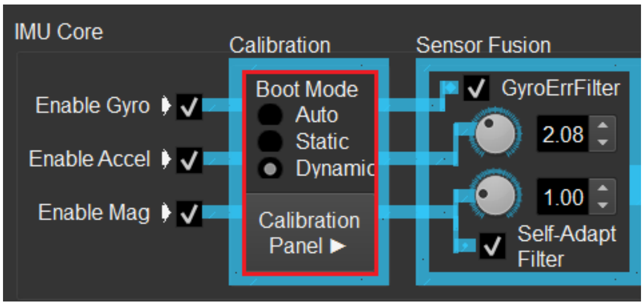


Figure 22: Data path (partial, boot mode and calibration panel)

If '**Static** Mode' is chosen, you have to keep the module stationary for at least 15 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 15 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℃ outdoor to 28℃ 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 23. It will then pop-up in section 3 shown in Figure 8.

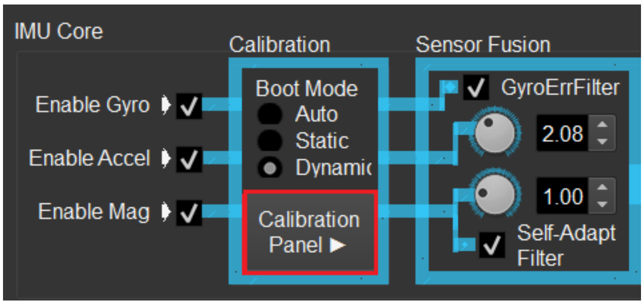


Figure 23: Button to open Calibration Panel

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 24. The use case of the 'CalibB' has been described in section 'Boot Mode' on page 14. 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.

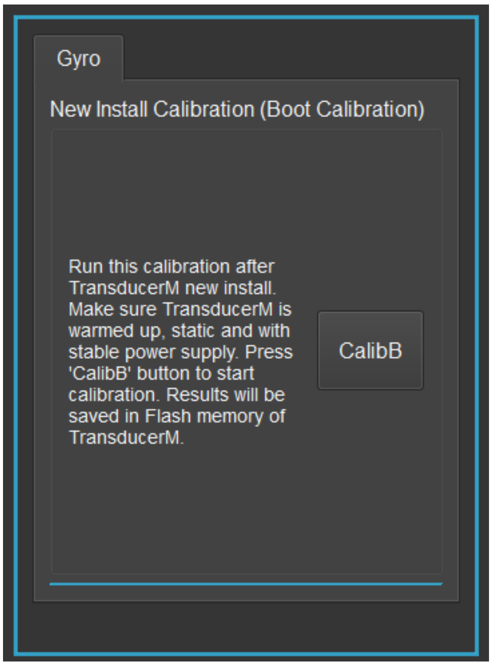


Figure 24: Calibration Panel minimum setup

Sensor Fusion

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

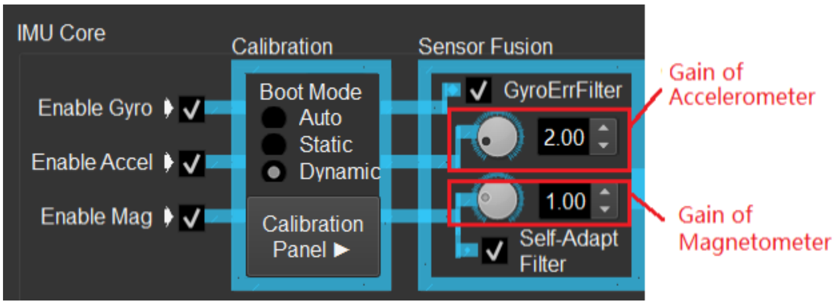


Figure 25: Data path (Partial, Sensor Fusion)
Please note that for TM100 series, the Gain setting for Accelerometer is not available

Inside the 'Sensor fusion' block, there are:

- '**GyroErrFilter**': a functionality that 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 10).

  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 lest 2.5.

  Please note that for TM100 series, the Gain setting for Accelerometer is not available.
- 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.

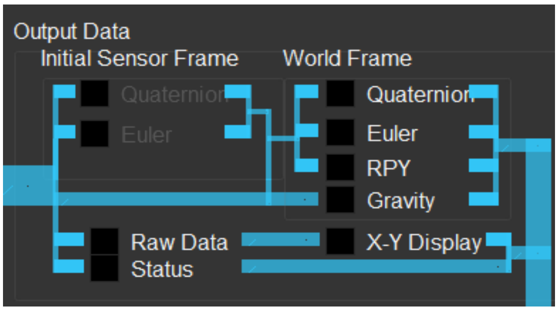


Figure 26: Configuring output of the module

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).

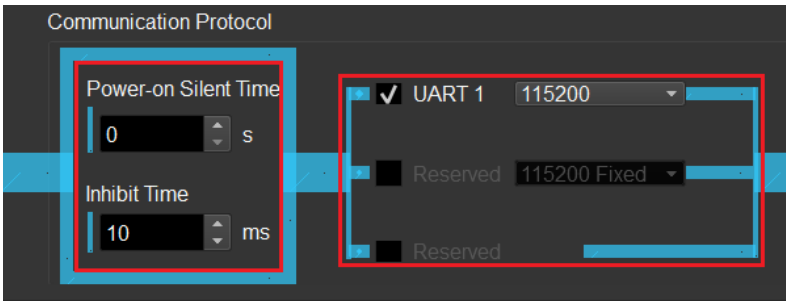


Figure 27: Configuring Communication Protocol

Here, you’re able to set the following:

- ‘**Power-on Silent Time**’: 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 responses 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 Windows 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** (For TM100 and TM200 series, only UART is available) and their respective baud rate (data transfer rate). For details regarding changing of UART baudrate, please refer to section ‘Change UART Baudrate’ on page 17.

Change UART Baudrate

For TM100 series, the data rate of UART1 is fixed to 115200 bps and cannot be changed.

For TM200 series, the data rate of UART1 can be changed by selecting the options as shown in Figure 28.

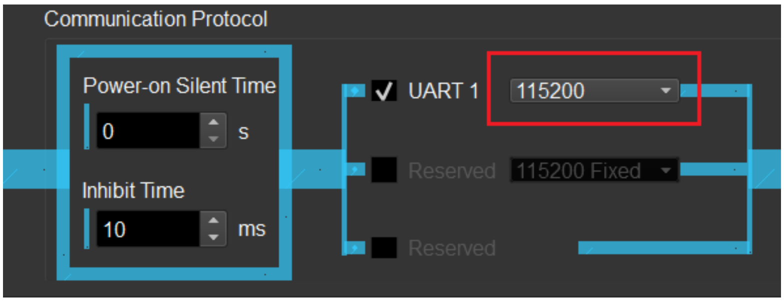


Figure 28: Configuring UART 1 baudrate

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 

as described in 'Finding Your Device' on page 6 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:

- For TM200 series, 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 17 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 17.

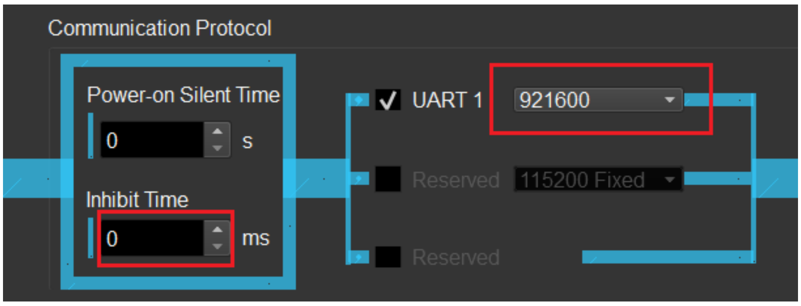


Figure 29 Example Setting to Increase Output Rate

Figure 29 shows a typical setting where around 150-200 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 24.

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 200Hz total output rate, with only 'Roll/Pitch/Yaw' data type selected for output, you get full 200Hz; if you both 'Roll/Pitch/Yaw' and 'Raw Data' selected, you get 100 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 problem.

Roll Pitch Yaw Display

Please refer to section 'Roll Pitch Yaw' on page 9.

Quaternion Display

Please refer to section 'Quaternion' on page 8.

Raw Data Display

Please refer to section 'Raw Data' on page 8.

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 Magnetometer 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 31. The red curve indicates X-Axis data, green for Y-Axis and blue for Z-Axis.

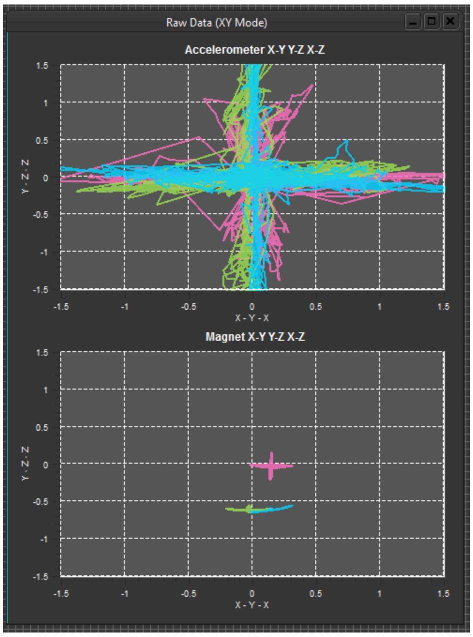


Figure 30: RAW data 'X-Y Display' visualization

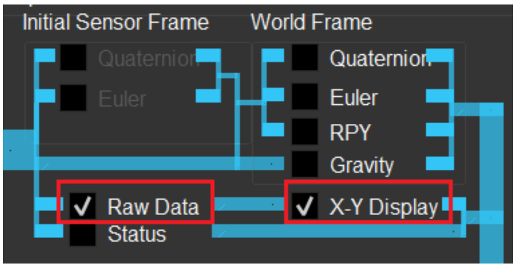


Figure 31: X-Y display output configuration

Save the Setting

Once you are satisfied with the configurations, click the 'Save Settings' button as shown in Figure 8 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 9.

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 select the library type and click 'Generate Code' as shown in Figure 32, or extract the downloaded zip file comes together with this document.

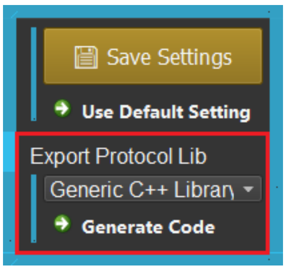


Figure 32: Save and Export

Please refer to the section 'Use SYD Dynamics Communication Library' on page 22 for instructions on how to use the communication library.

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 33 (the data types within the confine of the red polygon).

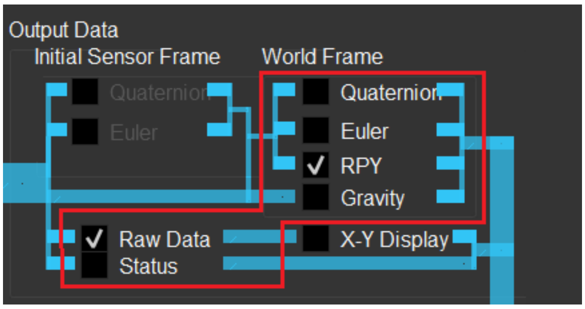


Figure 33: Data recorder step 1 - select the data types to be logged

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 8). If not, please check the procedures described in section 'Finding Your Device' on page 6, and 'Communication Protocol' on page 17.

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 34, 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 34. 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.

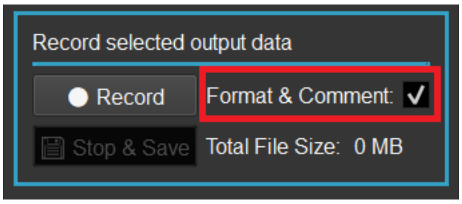


Figure 34: Data recorder step 2 - settings

Stop and Save Log Files

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

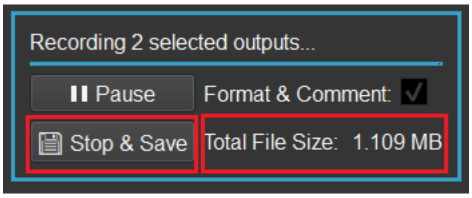


Figure 35: Data recorder step 3 - Stop & Save

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 36, which indicates you are safe to close the ImuAssistant without losing any data.

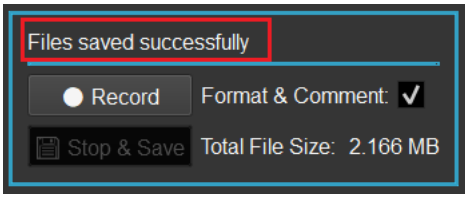


Figure 36: Data recorder step 3 - files saved successfully

The log data is saved in the file path as specified in section 'Configure Settings and Start to Record' on page 20.

Figure 37 shows example data log files when RawData and Roll/Pitch/Yaw data types are selected for recording.

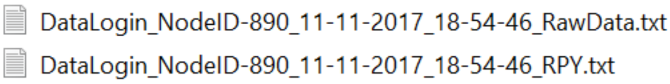


Figure 37: Data recorder - example of the recorded data files

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, time saving and can also be easily upgraded to future versions whenever new APIs are released.

The C++ library can be exported from the ImuAssistant GUI (refer to section ‘Export Communication Library’ on page 19).

Figure 38 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 38: 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:

```
// Step 1, 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.
    case EP_CMD_ACK_:{                             //          We can use a switch() as demonstrated here to do different
      Ep_Ack ep_Ack;                               //          tasks for different types of data.
      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 or can be exported from the ImuAssistant GUI (refer to section 'Export Communication Library' on page 19).

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 functionalists 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 18 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 22 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 22:
     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 18,
     // 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 C++ nor 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 39. 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.

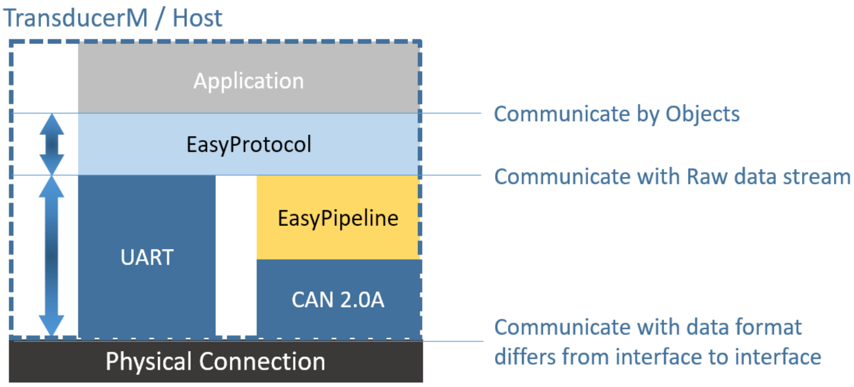


Figure 39: Communication Protocol Layers

Figure 39 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.

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 40. A Data Package is a Raw data stream representation of an Object. Both are carrying the same main information.

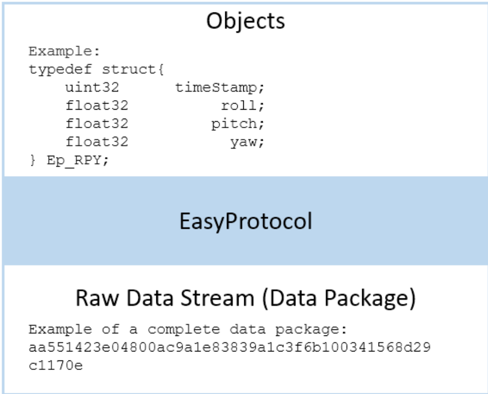


Figure 40: EasyProtocol Layer - Input Output Illustration

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 13.
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 28.

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 32 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_s1_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_s1_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_2)	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  timeStamp; // Timestamp when the Gravity Vector is calculated (Unit: uS)
    float32 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)

<table>
<tr><th>Bytes</th><th>Message</th><th>Data Type</th><th>Unit</th><th>Description</th></tr>
<tr><td>0-3</td><td>Time Stamp</td><td>32-bit unsigned integer</td><td>Micro-seconds (uS)</td><td>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.</td></tr>
<tr><td>4-7</td><td>Temperature</td><td>32-bit float</td><td>Celsius</td><td>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.)</td></tr>
<tr><td>8-9</td><td>Internal Update Rate</td><td>16-bit unsigned integer</td><td>Hz</td><td>TransducerM internal update rate for sensor fusion.</td></tr>
<tr><td>10-11</td><td>System Status bit fields</td><td>16-bit unsigned integer</td><td>-</td><td>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.
<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>
bit endianness: Little-endian

QoS[2..0] tells the Quality-of-Service of the system, which is an indication of how much performance the TransducerM delivers.

Possible values of QoS[2..0] are
<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>
</td></tr>
</table>

				<table><tr><td></td><td>fine performance. The TransducerM technical specification applies when the system reaches at least at this service level.</td></tr><tr><td>5</td><td>Very Good Service – All functions available and provide very good performance.</td></tr></table> For example, for a complete Status Object data package received, such as 'aa551016ec41007e405b5c080b264233030**500**4062', 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 **101**0 0000 (little-endian), which means QoS[2..0] = 101 in binary format, which means QoS=**5** (very good quality of service).

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	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. 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. 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 TransducrM at such frequency. An example request command in Raw data stream is '**aa55080c08000016000000e0ed**', which means it is a broad cast 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. 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
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;
```

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 26 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 28.
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-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 28, 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	**aa552c29ec41004029706b9d66383a508faab9da33c0b9a6e74d3c0267b9bb030480bf25bfac3d4fd40fbdcd4c4a3f5dfb**	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.**