Serial Port Calculator

The Serial Port Calculator is a software tool designed for PLC programmers and engineers who require connectivity to serial ports (RS-232/RS-485), sensors and data acquisition systems. It provides practical utilities for industrial communication, including:

- **Hex-to-decimal and decimal-to-hex conversion**
- **2-byte and 4-byte IEEE (floating-point) value calculation and conversion**
- **Serial port communication** (send/receive)
- Support for **Modbus protocol** messaging and testing

Overview of the Modbus Protocol

Modbus is a widely used **industrial communication protocol** originally developed by **Modicon (now Schneider Electric)** in 1979. It is designed for reliable communication between **industrial electronic devices**, such as **PLCs, sensors, actuators, HMIs, and SCADA systems**.

Modbus is valued for its **simplicity, openness, and robustness**, making it one of the most commonly implemented protocols in industrial automation.

Modbus Communication Model

Modbus follows a **master–slave** (also referred to as **client–server**) architecture:

- The **Master (Client)** initiates all communication requests.

- The **Slave (Server)** devices respond to the master's requests.

- Slave devices never communicate with each other directly.

Each slave device has a **unique address**, allowing the master to communicate with multiple devices on the same network.

Modbus RTU

- Uses **binary data format**

- Communicates over **serial links** such as RS-485 or RS-232

- High efficiency and compact data frames

- Widely used in industrial environments

Modbus Data Model

Modbus organizes data into four basic data types:

- **Coils** (Read/Write – Digital Outputs)

- **Discrete Inputs** (Read-only – Digital Inputs)

- **Input Registers** (Read-only – Analog Inputs)

- **Holding Registers** (Read/Write – Analog Values and Parameters)

Each data type is accessed using specific **function codes** defined by the Modbus standard.

Modbus RTU Message Frame

A standard **Modbus RTU** frame consists of the following fields:

| Address | Function Code | Data | CRC |

Field Descriptions

1. Address (1 byte)

- Identifies the target slave device
- Valid range: **1–247**
- Address **0** is reserved for broadcast messages (no response)

2. Function Code (1 byte)

Defines the action requested by the master.

Function Code Description

01	Read Coils
02	Read Discrete Inputs
03	Read Holding Registers
04	Read Input Registers
05	Write Single Coil
06	Write Single Register
15 (0x0F)	Write Multiple Coils
16 (0x10)	Write Multiple Registers

3. Data Field (N bytes)

Contains parameters or returned values, depending on the function code.

4. CRC (2 bytes)

- Cyclic Redundancy Check for error detection
- Sent as **low byte first, then high byte**

Example:

This example is a real Modbus message from the ADAM-4017+ module manufactured by Advantech. The module is configured to operate using the Modbus protocol.

Send   → 10 04 00 00 00 08 F2 8D

Message Structure:

1. **Slave ID:** Identifies the target device.

   The first hexadecimal byte is 10, which represents the slave (node) address in hexadecimal (**17 in decimal**).

2. **Function Code:** Specifies the operation requested by the master (e.g., read or write).

   The second hexadecimal byte is 04, indicating the **Read Input Registers** function.

3. **Data**: Defines the registers to be accessed.
   - The third and fourth hexadecimal bytes (00 00) specify the **starting register address**.
   - The fifth and sixth hexadecimal bytes (00 08) indicate the **number of registers to be read**.

4. **CRC (Cyclic Redundancy Check)**: Used for error detection. The seventh and eighth hexadecimal bytes (F2 8D) represent the **CRC value** of the message.

Received   → 10 03 10 7F FF 80 02 7F FF 7F FF 7F FC 7F FE 7F FE 7F FC 54 82

Message Structure:

1. **Slave ID:** Identifies the responding device

   The first hexadecimal byte is 10, which represents the slave (node) address in hexadecimal (**17 in decimal**).

2. **Function Code:** Specifies the operation performed.

   The second hexadecimal byte is 03, indicating the **Read Holding Registers** function.

3. **Data**: Contains the returned register values.
   - The third hexadecimal byte (10) specifies the **byte count**, indicating **16 bytes** of data follow.
   - The fourth through nineteenth hexadecimal bytes (7F FF 80 02 7F FF 7F FF 7F FC 7F FE 7F FE 7F FC) represent **eight 2-byte register values** returned by the slave.

4. **CRC (Cyclic Redundancy Check)**: Used for error detection. The twentieth and twenty-first hexadecimal bytes (54 82) represent the **CRC value** of the message.

The message structure for other Modbus devices may be similar; however, there are differences in the details, including the representation of real data types, which typically use IEEE floating-point format.

Exception Response Format
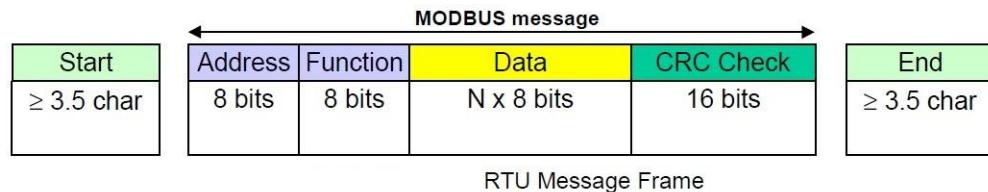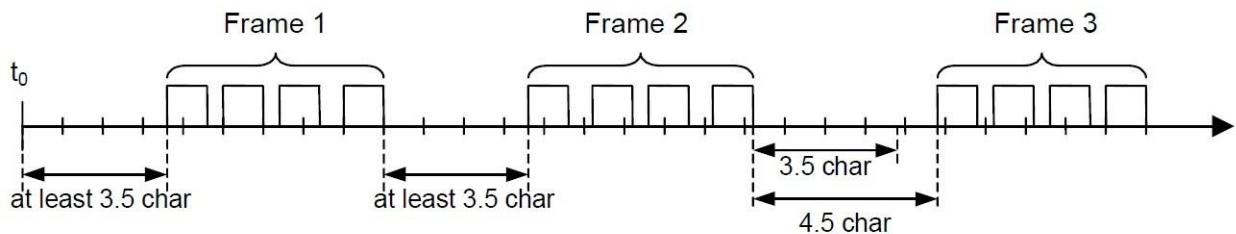
If an error occurs, the slave responds as follows:

| Address | Function Code + 0x80 | Exception Code | CRC |

Common Exception Codes:

- 01 – Illegal Function
- 02 – Illegal Data Address
- 03 – Illegal Data Value
- 04 – Slave Device Failure

Slave Address	Function Code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low CRC Hi

| Start	Address	Function	Data	CRC Check	End
≥ 3.5 char	8 bits	8 bits	N x 8 bits	16 bits	≥ 3.5 char

RTU Message Frame
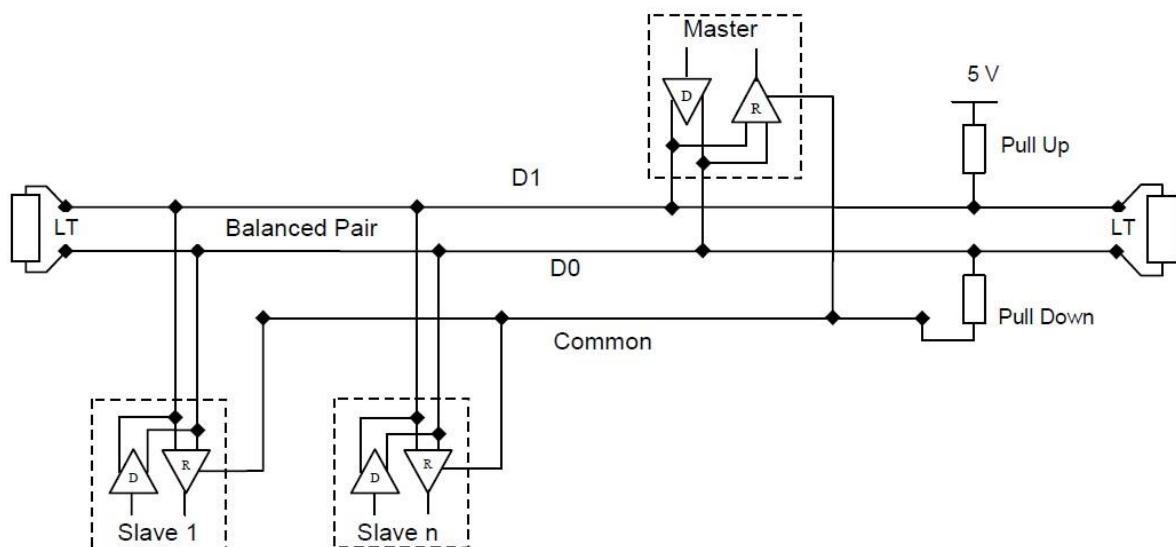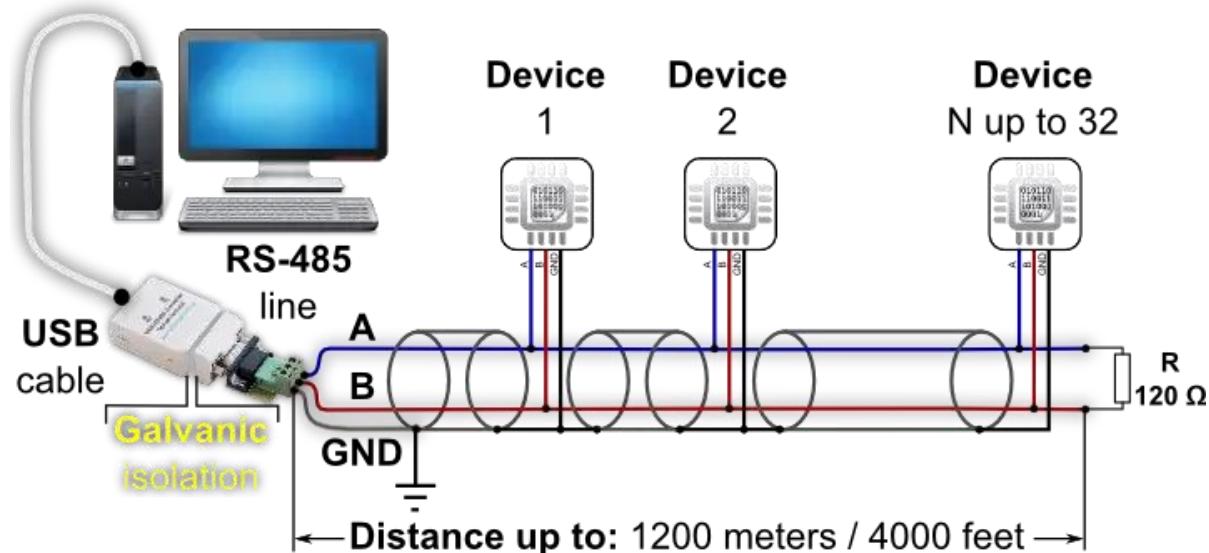
Advantages of Modbus

- Open and license-free protocol
- Easy to implement and troubleshoot
- Supported by a wide range of manufacturers
- Reliable communication over long distances (RS-485)

Typical Applications

- Industrial automation systems
- PLC-to-device communication
- Energy management systems
- Building automation
- Data acquisition and monitoring
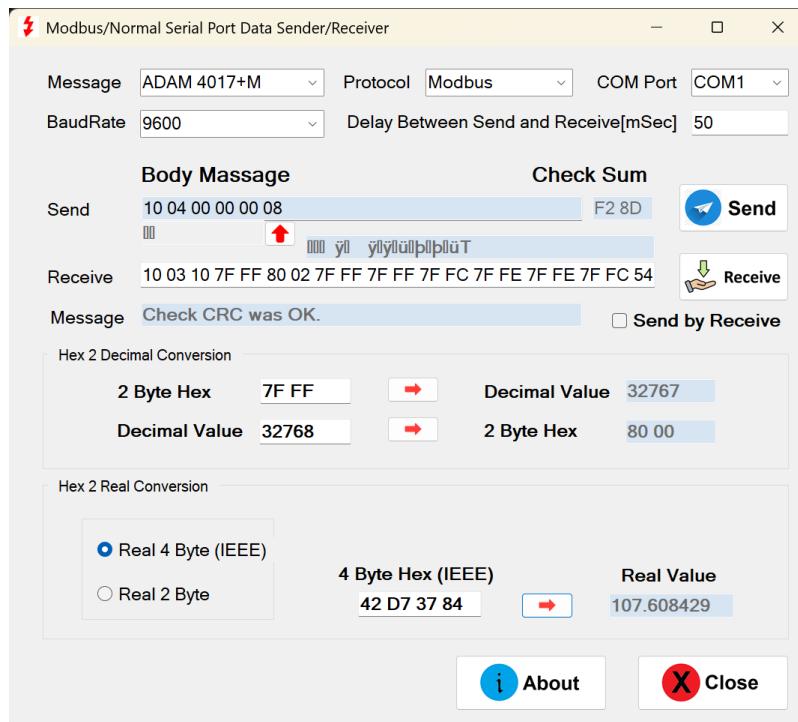
General 2-Wire Topology

## Software for Sending and Receiving Commands (Serial / Modbus)
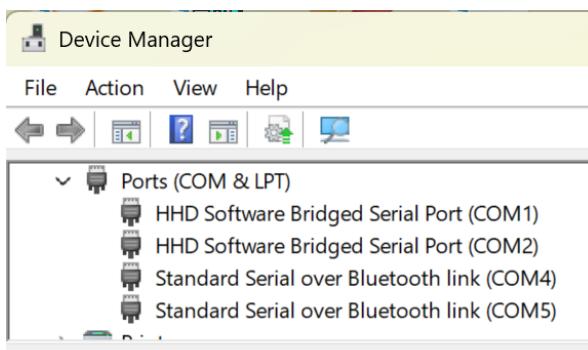
This software enables users to communicate with a device by sending and receiving commands. It allows the user to:

- Configure the serial port settings
- Select or build the desired command
- Calculate the Modbus checksum (CRC) according to the Modbus RTU standard.
- Send the command to the device
- Wait a predefined number of milliseconds and then read the response
- Convert hexadecimal values to decimal
- Convert 4-byte hexadecimal values (IEEE floating-point) to real values
- Convert 2-byte hexadecimal values to real values
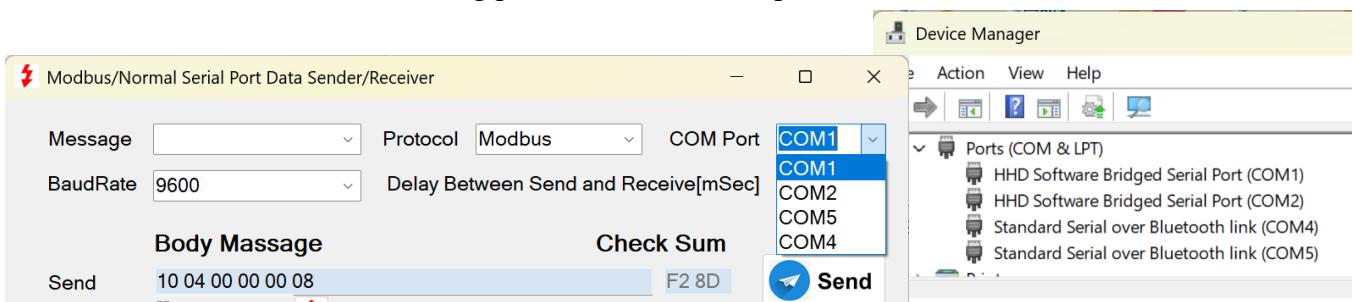
### How to Identify a Serial Port on Your Computer
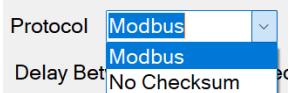
- Open the **Device Manager** application.
- Expand the **Ports (COM & LPT)** section.
- If a serial port is available, it will be listed there, as shown in the snapshot below.
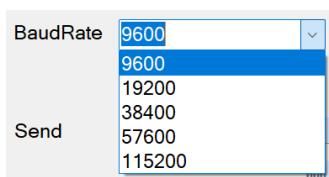
You can choose one of the existing ports as shown in the picture.

How to change the Protocol

How to change the Baud Rate

The latest settings are saved in the "**ModbusConver.ini**" file and are restored the next time the software is opened.

How to choose a message

The operator can configure up to 10 messages in the "**Message.ini**" file, which contains both the message protocol definition and the corresponding display name used by the software. The package includes predefined messages for three Advantech devices: "**ADAM-4017**" for reading eight voltage or current channels, "**ADAM-4015**" for reading six **PT100** temperature sensors, and "**ADAM-4018**" for reading eight **thermocouple** temperature sensors.

Sample "**Message.ini**" file content:

10 04 00 00 00 08;Modbus;ADAM 4017+;Advantech Adam 4017+ Modbus protocol for A/D

13 04 00 00 00 06;Modbus;ADAM 4015;Advantech Adam 4015 Modbus protocol for PT100 temperature Sensor

16 04 00 00 00 08;Modbus;ADAM 4018;Advantech Adam 4018 Modbus protocol for thermocouple Sensor

23 31 30 0D;No Checksum;ADAM 4017;Advantech Adam 4017 Advantech protocol for A/D

23 31 33 0D;No Checksum;ADAM 4015;Advantech Adam 4015 Advantech protocol for PT100 temperature Sensor

23 31 36 0D;No Checksum;ADAM 4018;Advantech Adam 4018 Advantech protocol for thermocouple Sensor
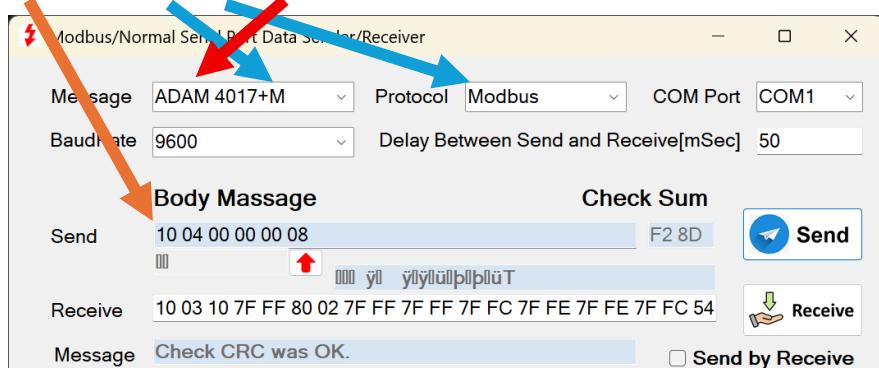
Each message is written on a single line, and each part is separated by a semicolon (;), as shown.

- The **first part** is the message in **hexadecimal format**. Each ASCII character is represented by **two hexadecimal digits** ranging from 00 to FF. For example, 10 in **hexadecimal** is equal to 16 in **decimal**. Each byte is separated by a space.

- The **second part** defines the **message protocol**. Two options are available: "**Modbus**" or "**No Checksum**". If "**Modbus**" is selected, the checksum (CRC) is calculated and automatically appended to the message before transmission.

- The **third part** is the **name of the device** to which the message will be sent.

- The **fourth part** is a **remark field**, used to provide additional information or comments.
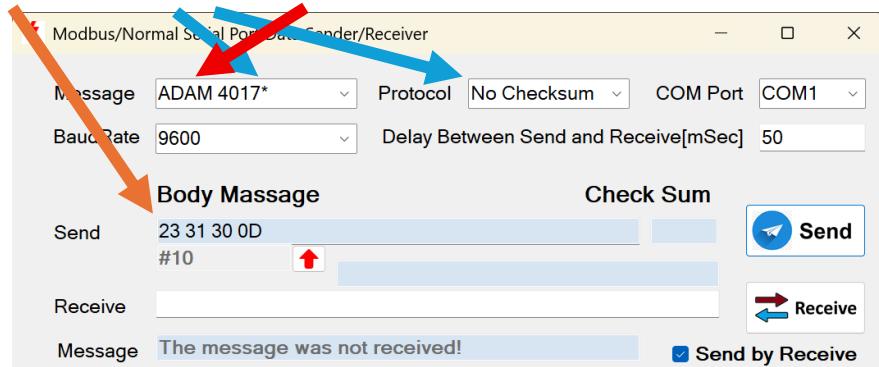
If "**Modbus**" is selected (in the 2nd part), the letter "**M**" is appended to the end of the device name (in the 3rd part), and if "**No Checksum**" is selected, an asterisk (**"*"**) is appended to the device name.

In our example, **Advantech** devices can also operate using a special Advantech message such as "**23 31 30**", which corresponds to the ASCII string "**#10**". In this case, the checksum is not applied.
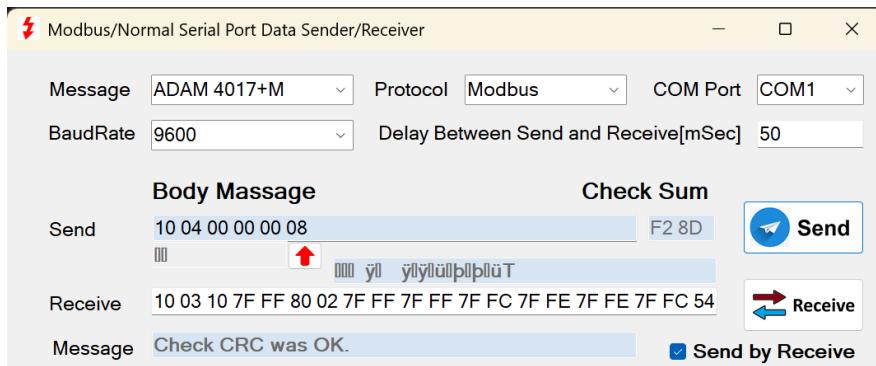
10 04 00 00 00 08;Modbus;ADAM 4017+;Advantech Adam 4017+ Modbus protocol for A/D

23 31 30 0D;No Checksum;ADAM 4017;Advantech Adam 4017 Advantech protocol for A/D

When a message is generated by pressing the **"Send"** button, if the selected protocol is **"Modbus"**, the checksum is automatically calculated and appended to the message before it is sent to the device. The software then waits for a predefined number of milliseconds and reads the response, which is displayed in the **"Receive"** section.

Another option is available through the **"Receive"** button. When this button is pressed, the configuration icons are disabled, and the software waits to receive a message. The current status is displayed in the **"Message"** section.

If **"Send by Receive"** is selected, the **"Receive"** button switches to a **two-way communication mode**. In this mode, after a message is automatically received, the main message is immediately sent back to the device. This feature is particularly useful for **loop-based operation** and **hardware simulation** scenarios.

All sent and received data are logged in the "**ModbusConver.txt**" file.

Sample content of the "**ModbusConver.txt**" file is shown below:

This text file was generated by ModbusConver.exe and developed by Babak Hajari. (Caution: Do not delete this line.)

2026/01/17   09:10
Send   => 10 04 00 00 00 08 F2 8D
Received => 10 03 10 7F FF 80 02 7F FF 7F FF 7F FC 7F FE 7F FE 7F FC 54 82
Message => Check CRC was OK.

2026/01/17   09:20
Send   => 10 04 00 00 00 08 F2 8D

2026/01/17   09:30
Send   => 10 04 00 00 00 08 F2 8D
Received => 10 03 10 7F FF 80 02 7F FF 7F FF 7F FC 7F FE 7F FE 7F FC 54 82 10 03 10 7F FF 80 02 7F FF 7F
FF 7F FC 7F FE 7F FE 7F FC 54 82
Message => Check CRC was not OK!

2026/01/17   09:40
Send M => 10 04 00 00 00 08 F2 8D
Message => The message was not received!

Calculation

Hex-to-Decimal Conversion

In this section, you can convert **2-byte hexadecimal values** to **decimal values**, and vice versa.

Hex 2 Decimal Conversion

2 Byte Hex	7F FF	➡	Decimal Value 32767
Decimal Value	32768	➡	2 Byte Hex 80 00

Hex-to-Real Conversion

In this section, you can convert **4-byte hexadecimal values (IEEE floating-point format)** to **real values**.

Hex 2 Real Conversion

- ⦿ Real 4 Byte (IEEE)
- ○ Real 2 Byte

4 Byte Hex (IEEE)   Real Value
42 D7 37 84   ➡   107.608429

In this section, you can convert **2-byte hexadecimal values** into **real values** using defined **minimum and maximum limits**. For example, if the minimum value is **–10 volts** and the maximum value is **10 volts**, the corresponding real value is calculated based on the hexadecimal input.

Hex 2 Real Conversion

- ○ Real 4 Byte (IEEE)
- ⦿ Real 2 Byte

Min Value -10   Max Value 10
2 Byte Hex   Real Value
7F FF   ➡   -.000153

For example, if the minimum value is **4 milli-amperes** and the maximum value is **20 milli-amperes**, the corresponding real value is calculated based on the hexadecimal input.

Hex 2 Real Conversion

- ○ Real 4 Byte (IEEE)
- ⦿ Real 2 Byte

Min Value 4   Max Value 20
2 Byte Hex   Real Value
7F FF   ➡   11.999878

By pressing the "**About**" button, information about the package and its developer is displayed in a new window, as shown below.