
# *C/C++ implementation of the Modbus interface*

*Enovasense Remote Control SDK*

# C/C++ Enovasense Modbus TCP client implementation

The C/C++ Enovasense SDK uses the *libmodbus* library (https://libmodbus.org/) as its only dependency. You will need to compile and link *libmodbus* to use this SDK. It was tested against *libmodbus* v3.1.7. *Libmodbus* is available for Windows (MSVC), Linux (g++) and MacOS. It is also possible to compile libmodbus on Windows with g++ using MinGW.

The SDK is basically a C++ class wrapping the Modbus read/write operations one would need to connect to an Enovasense sensor, select a calibration and start some measurements. The calibration process still needs to be done (at least partially) with the Enovasense Suite application.

To modify the implementation, please refer to the register mapping which is available in the Enovasense Modbus Specifications document (kept here as an annex for now).

## Overview

The SDK is composed of two files: “enovasensemodbusclient.cpp” and “enovasensemodbusclient.h”. It is possible to include the sources directly inside an existing project or to compile the library into the enovasensemodbusclient.dll file to link it dynamically. I would advise to use the sources directly.

The EnovasenseModbusClient class contains all the methods needed to read and write the registers in the most convenient way for the user.

EnovasenseAddressMapping is the namespace containing all of the addresses mapped to words. It also contains a “StatusBits” enumeration for STAT address bit states and a “ENO_ERRCODE” enumeration for error codes in the module control methods.

The namespace uses abbreviations which are described in the document "Modbus specifications” or in the Modbus registers mapping annex below.

The alias EAM will be used for EnovasenseAddressMapping namespace.

The following insight gives more details about each public method ready for use:

- Initialization of the client

**EnovasenseModbusClient() :**
It is the default constructor of the class. It only initializes some member variables.

**EnovasenseModbusClient(std::string IP, int port) :**
It is the constructor of the class which takes 2 parameters: the first one is the IP address of the Enovasense module connected to the user’s device (ex: the computer) and the later one is the port in which the server within the module is running. The user is invited to call this method at first in his code in the main.cpp (or in any other file in any other project that the user may set for his usage).
The constructor establishes the TCP socket to interact with the server and launches the client with open(std::string IP, int port). If no port is given as an argument, it will use the port 502 by default.

**int open(std::string IP, int port) :**
The user can call it independently from the constructor if the default constructor

EnovasenseModbusClient() has been used to create the default client object.
If no port is given as an argument, it will use the port 502 by default.

- Close the client

Handled by the destructor of the class. The destructor calls close() method to destroy the client.

- Get the last returned error

**int lastErrorCode() const :**
Returns the  last error code.

**std::string lastErrorString() const :**
Returns the last error description.

- Module Control commands

**int getCalibration() :**
Returns the current calibration used in the Enovasense module. The current calibration is contained in the CCAL address. Returns -1 if it can't read the register.

**int setCalibration(uint16_t value) :**
This method sets the calibration that the user wants to use in the address SCAL. "value" is a 16-bit integer between 1 and 999 corresponding to the Calibration number. If the calibration doesn't exist or can't be set EAM::ENO_INVALIDCALIBRATION error code is returned. Otherwise, the Calibration Trigger is called to load the calibration. If the calibration changed successfully EAM::ENO_NOERROR is returned otherwise EAM::ENO_TIMEOUT is returned.

**int startMeasurement() :**
This method is the one that triggers the measurement in the Enovasense module. It first checks if the STAT address has it's first bit (index 0) is set to 1 (meaning that the Enovasense device is ready for a measurement). This condition returns the error code EAM::ENO_NOTREADY if the bit is set to 0. Otherwise, the Measurement trigger is activated and if the measurement process is successful EAM::ENO_NOERROR error code is returned. If the device isn't measuring EAM::ENO_NOTMEASURING error code is returned.

**int isSoftwareLockEnabled() :**
The user can call this method to make sure of the state of the Software Lock in the address SLCK. Returns the bit value of "Laser Unlocked" in STAT address which is 1 if SLCK is set to 0xFF or 0 if SLCK is set to 0x00.

**int setSoftwareEnabled(bool enabled) :**
The user is urged to set the software lock before starting a measure to activate the laser properly. This method uses a Boolean argument, if "enabled" is "true" then it sets SLCK to 0xFF and if it is "false" then it sets SLCK to 0x00.
Returns the value set to SLCK if successful (i.e 0xFF or 0x00) or -1 if it couldn't set any value.

- Read calibration settings and the status of the last measurement

**int getLaserAmplitude() :**
Returns the laser amplitude being used for the measures when called. Returns -1 if it can't read the register.

**int getLaserThreshold() :**
Returns the laser threshold being used for the measures when called. Returns -1 if it can't read the register.

**int getCoolingTime() :**
Returns the cooling time of the laser when called. Returns -1 if it can't read the register.

**int getHeatingTime() :**
Returns the heating time of the laser when called. Returns -1 if it can't read the register.

**int getMeasuringTime () :**
Returns the time duration of each measurement when called. Returns -1 if it can't read the register.

**int getFrequenciesNumber() :**
Returns the number of frequencies used during the measurement. Returns -1 if it can't read the register.

**int getOutputsNumber() :**
Returns the number of outputs returned during the measurement. Returns -1 if it can't read the register.

**int getStatusBit(int index) :**
Returns the state of the bit indexed at "index" in STAT address:
-1 if it can't read the register
0 if the bit is cleared
1 if the bit is set
The argument is an integer "index" which corresponds to the index of the bit describing a specific status in the value stored by STAT.

**int isReady() :**
A simple method that calls getStatusBit(EAM::Ready). Returns the Status of the bit indexed at EAM::Ready (i.e 0) in the STAT address to check if the Enovasense device can start the measurement. Returns -1 if it can't read the register.

**int isMeasuring() :**
A simple method that calls getStatusBit(EAM::Measuring). Returns the Status of the bit indexed at EAM::Measuring (i.e 1) in the STAT address to check if the Enovasense device is currently measuring something. Returns -1 if it can't read the register.

**int isCalibrationSetSuccessfully() :**
A simple method that calls getStatusBit(EAM::CalibrationSetSuccessfully). Returns the Status of the bit indexed at EAM::CalibrationSetSuccessfully (i.e 2) in the STAT address to check if the Enovasense device changed the calibration successfully after the last calibration trigger. Returns -1 if it can't read the register.

**int isLaserUnlocked() :**
A simple method that calls getStatusBit(EAM::SoftwareUnlocked). Returns the Status of the bit indexed at EAM::SoftwareUnlocked (i.e 3) in the STAT address to check if the Enovasense device has the laser unlocked for the measure (and the SLCK is equal to 0xFF if it's the case). Returns -1 if it can't read the register.

**int isLaserOn() :**
A simple method that calls getStatusBit(EAM::LaserOn). Returns the Status of the bit indexed at EAM::LaserOn (i.e 4) in the STAT address to check if the laser in the Enovasense device is activated. Returns -1 if it can't read the register.

**int isAcquisitionBufferEmpty() :**
A simple method that calls getStatusBit(EAM::AcquisitionBufferEmpty). Returns the Status of the bit indexed at EAM::AcquisitionBufferEmpty (i.e 5) in the STAT address to check if the Acquisition Buffer is empty. Returns -1 if it can't read the register.

**int isModuleExternalLock() :**
A simple method that calls getStatusBit(EAM::ModuleExternalLock). Returns the Status of the bit indexed at EAM::ModuleExternalLock (i.e 6) in the STAT address to check if the module is remotely locked by another application. Returns -1 if it can't read the register.

**std::vector<uint16_t> getAllFrequencies() :**
Returns the vector of all the frequencies stored in the FRQX addresses used during the measurement.

**int getFrequencyAt(int index) :**
Returns the frequency stored at the address FRQX with X being the "index". It returns -1 if it fails to read the register.

- Read last measures – Calibration outputs

**std::vector<uint16_t> getAllOutputs() :**
Returns the vector of all the outputs stored in the OUTX addresses filled during the measurement.

**int getOutputAt(int index) :**
Returns the output stored at the address OUTX with X being the "index". It returns -1 if it fails the register.

- Read last measures – Raw data

**std::vector<uint16_t> getRawData() :**
Returns the vector of the phase/amplitude couple values stored in the PHIX and AMPX addresses filled during the measurement. The vector is filled with this order: [phase, amplitude, phase, amplitude, …]

## Modbus information

This document assumes that the reader has some knowledge about the Modbus protocol. More information about Modbus can be found here: https://modbus.org/specs.php

## Enovasense Modbus usage

Here is some basic information to help the user understand how Modbus is implemented in the Enovasense device:

- The Enovasense device can be accessed via Modbus TCP. It is possible to restrain the client addresses. This needs to be done via Enovasense as it requires firmware modifications (same as setting the Enovasense IP address).
- The registers used are holding registers. Hence the Modbus function codes supported are the function code 0x03 (read holding registers) for reading and the function code 0x06 (write holding register) and 0x10 (write holding registers) for writing.

## Enovasense measure trivia

Here is some basic information to help the user understand how certain parameters work. More information can be found in the manual of the Enovasense device.

- One measurement is separated in three phases: cooling, heating, and measuring. During the cooling, the laser is turned off to prevent the residual heating from previous measurements to affect the measurement process. During the heating, the laser is turned on but no data is acquired. This ensure that the system is in the right state of (pseudo-)equilibrium when measuring. During the measuring phase, data is acquired and processed to determine the thickness of the sample.
- The Enovasense device needs to be calibrated before the Modbus server can be used. The user must use a computer with the Enovasense Desktop Suite installed to create suitable calibrations to use, then load them on the device. The user will then be able to use the loaded calibration to measure sample via Modbus TCP.

## *Modbus registers map*

Below is the Modbus registers detail of the Enovasense device. Please note that everything is accessible from the holding registers.

| Address [DEC] and aera name | Description | | Symbol | Word counts | Unit |
|---|---|---|---|---|---|
| 0 to 9: Enovasense module control | Offset [DEC] | | | | |
| | +00 | Set calibration code | SCAL | 1 | |
| | +01 | Change calibration trigger | CTRG | 1 | *1 |
| | +02 | Start measurement trigger | MTRG | 1 | *1 |
| | +03 | Software enable. 0x00FF allows the laser to shoot. It is basically a software interlock. | SLCK | 1 | |
| | +04 | Reserved | | 1 | |
| | +05 | Reserved | | 1 | |
| | +06 | Reserved | | 1 | |
| | +07 | Reserved | | 1 | |
| | +08 | Reserved | | 1 | |
| | +09 | Reserved | | 1 | |
| 10 to 29: Enovasense module current settings and status | Offset [DEC] | | | | |
| | +00 | Current calibration code | CCAL | 1 | |
| | +01 | Laser amplitude | VAMP | 1 | mV |
| | +02 | Laser threshold | VTHR | 1 | mV |
| | +03 | Cooling time. | COOL | 1 | ms |
| | +04 | Heating time. | HEAT | 1 | ms |
| | +05 | Measuring time. | MEAS | 1 | ms |
| | +06 | Number of frequencies | NFRQ | 1 | |
| | +07 | Calibration number of outputs | NOUT | 1 | |
| | +08 | Current status of the Enovasense device (See below for bit mapping) | STAT | 1 | |
| | +09 | Reserved | | 1 | |
| | +10 | Frequency N°0 | FRQ0 | 1 | Hz |
| | +11 | Frequency N°1 | FRQ1 | 1 | Hz |
| | +12 | Frequency N°2 | FRQ2 | 1 | Hz |
| | +13 | Frequency N°3 | FRQ3 | 1 | Hz |
| | +14 | Frequency N°4 | FRQ4 | 1 | Hz |
| | +15 | Frequency N°5 | FRQ5 | 1 | Hz |
| | +16 | Frequency N°6 | FRQ6 | 1 | Hz |

| | | | | | |
|---|---|---|---|---|---|
| | +17 | Frequency N°7 | FRQ7 | 1 | Hz |
| | +18 | Frequency N°8 | FRQ8 | 1 | Hz |
| | +19 | Frequency N°9 | FRQ9 | 1 | Hz |
| 30 to 39: Last measure - Calibration outputs | Offset [DEC] | | | | |
| | +00 | Output N°0 | OUT0 | 1 | *2 |
| | +01 | Output N°1 | OUT1 | 1 | *2 |
| | +02 | Output N°2 | OUT2 | 1 | *2 |
| | +03 | Output N°3 | OUT3 | 1 | *2 |
| | +04 | Output N°4 | OUT4 | 1 | *2 |
| | +05 | Output N°5 | OUT5 | 1 | *2 |
| | +06 | Output N°6 | OUT6 | 1 | *2 |
| | +07 | Output N°7 | OUT7 | 1 | *2 |
| | +08 | Output N°8 | OUT8 | 1 | *2 |
| | +09 | Output N°9 | OUT9 | 1 | *2 |
| 40 to 59: Last measure - Raw data | Offset [DEC] | | | | |
| | +00 | Phase N°0 | PHI0 | 1 | 0.01° |
| | +01 | Amplitude N°0 | AMP0 | 1 | 0.001 |
| | +02 | Phase N°1 | PHI1 | 1 | 0.01° |
| | +03 | Amplitude N°1 | AMP1 | 1 | 0.001 |
| | +04 | Phase N°2 | PHI2 | 1 | 0.01° |
| | +05 | Amplitude N°2 | AMP2 | 1 | 0.001 |
| | … | … | … | … | … |
| | … | … | … | … | … |

*1: Setting the word to 0x00FF trigger the action. The user needs to reset the word to reuse the function.
*2: The data are unsigned integers encoded on 16 bits. The user shall create his calibration accordingly.

## Status word mapping

The STATS word (address: 18) contains the current status of the Enovasense device. The bits signification is described below:

| Bit index | Description |
|---|---|
| 0 | **Ready**: 1 when the Enovasense device can start a measurement, else 0. |
| 1 | **Measuring**: 1 if the Enovasense device is measuring, else 0. |
| 2 | **Calibration set successfully**: 1 if the last calibration trig led to a successfully calibration change, else 0. |
| 3 | **Laser unlocked**: 1 if SLCK is enabled, else 0. |
| 4 | **Laser ON**: 1 if the laser is turned on, else 0. This doesn't take the physical key or interlock into account. |
| 5 | **Acquisition buffer empty**: 1 if the acquisition buffer is empty, else 0. It should not be 1 after a measurement. If it is, then there may be a configuration or a firmware issue. |
| 6 | **Module external lock:** 1 if the module is remotely locked by another application, else 0. The module cannot be remote controlled via Modbus TCP when externally locked. Connecting to the module with the Enovasense Desktop application will lock the module until it is disconnected. |
| 7 | Reserved |
| 8 | Reserved |
| 9 | Reserved |
| 10 | Reserved |
| 11 | Reserved |
| 12 | Reserved |
| 13 | Reserved |
| 14 | Reserved |
| 15 | Reserved |

## Modbus specifications

EnovasenseAdrressMapping namespace:

```
namespace EnovasenseAddressMapping
{
// MACRO FOR THE ADDRESSES
uint16_t SCAL = 0;
uint16_t CTRG = 1;
uint16_t MTRG = 2;
uint16_t SLCK = 3;
uint16_t CCAL = 10;
uint16_t VAMP = 11;
uint16_t VTHR = 12;
uint16_t COOL = 13;
uint16_t HEAT = 14;
uint16_t MEAS = 15;
uint16_t NFRQ = 16;
uint16_t NOUT = 17;
uint16_t STAT = 18;
uint16_t FRQ0 = 20;
uint16_t FRQ1 = 21;
```

```
uint16_t FRQ2 = 22;
uint16_t FRQ3 = 23;
uint16_t FRQ4 = 24;
uint16_t FRQ5 = 25;
uint16_t FRQ6 = 26;
uint16_t FRQ7 = 27;
uint16_t FRQ8 = 28;
uint16_t FRQ9 = 29;
uint16_t OUT0 = 30;
uint16_t OUT1 = 31;
uint16_t OUT2 = 32;
uint16_t OUT3 = 33;
uint16_t OUT4 = 34;
uint16_t OUT5 = 35;
uint16_t OUT6 = 36;
uint16_t OUT7 = 37;
uint16_t OUT8 = 38;
uint16_t OUT9 = 39;
uint16_t PHI0 = 40;
uint16_t AMP0 = 41;
uint16_t PHI1 = 42;
uint16_t AMP1 = 43;
uint16_t PHI2 = 44;
uint16_t AMP2 = 45;

// different bit values contained in STAT address
enum StatusBits {
    Ready = 0,
    Measuring,
    CalibrationSetSuccessfully,
    SoftwareUnlocked,
    LaserOn,
    AcquisitionBufferEmpty,
    ModuleExternalLock
};

// MACRO of error codes used in the SDK
enum ENO_ERRCODE {
    ENO_INVALIDCALIBRATION = -4,
    ENO_NOTREADY,
    ENO_TIMEOUT,
    ENO_NOTMEASURING,
    ENO_NOERROR
};
}
```