Modbus Protocol
for
SISGEO digitized instruments

-SPECIFICATION-

Introduction on SISGEO digitized instruments

The SISGEO digitized instruments use RS-485 serial communication and as such they can be chained together and connected to a Master's MODBUS device (datalogger, PC, etc.).
Normally each sensor is configured at the factory by SISGEO to have a unique RS-485 address. This configuration allows a single datalogger to take readings from multiple sensors in a chain.
At the start of each scan, the power supply output of the datalogger should be turned on and then turned off after reading from all the sensors in the chain.

Each sensor is set at the factory also for the power supply mode: always on or timed (called 'SISGEO Communications Window Feature') used to conserve power.

Always On (standard setup)

All the sensors in the chain are simultaneously powered and "awake" (maximum power consumption state): each sensor is ready to respond to the commands of the Master's MODBUS device (i.e. a datalogger).

Typical consumption (for EACH sensor): ~7mA @ 24Vdc, ~12mA @ 12Vdc

Timed (only on request) - called 'SISGEO Communications Window Feature'

The feature uses the Warming Delay and Address Delay parameters stored in the electronic non-volatile memory to keep the sensor in a low power state while reading from other sensors in the chain.
The Master's Modbus device (i.e. a datalogger or a PC) should wait the appropriate amount time before attempting to read from a particular sensor in the chain. The Master's Modbus device will turn on the power supply output at the start of the scan; each sensor will stay in its lower power state according to the Warming Delay and Address Delay times. Once a sensor is "awake" it will be in its maximum power consumption state.
The sensors have a command to enter "Stop Mode" to return to a low power state. The Master's Modbus device should send the "Switch Off" command to put each sensor into "Stop Mode" just after reading its values.
Once a sensor is in "Stop Mode" the only way to wake it up again is to remove its external power and restore it.
Each sensor will be turned on after a time equal to its RS-485 address:

Address Delay = RS-485 address * delay

and will start measuring for a Warming Delay.
It will be possible for the Master's Modbus device read the measurement after a timeframe given by:

Waiting time = Address Delay + Warming Delay

NOTE: The Warming Delay and Address Delay parameters must be the same for all the sensors in the chain

Typical consumption (only for a consumption estimation, for more details please refers to the specific technical data sheets):

Address Delay:	~ 1mA@24Vdc, ~ 2mA@12Vdc
Warming Delay:	~ 5mA@24Vdc, ~ 7mA@12Vdc
Ready:	~ 7mA@24Vdc, ~ 12mA@12Vdc
Stop mode:	~ 0.3mA@24Vdc, ~ 0.4mA@12Vdc

Power range :	9÷28Vdc with SMPS
Advised power supply:	24Vdc

Other info could be find in each instrument manual and datasheet; also FAQs can be useful.

Modbus interface

The Modbus interface uses the following communication parameters (not modifiables):

Baud rate : 9600
Data bits : 8
Stop bits : 1
Parity : None

The instrument address (default 1) may be set in the configuration registers: the instrument will always answer a message with a target address of 255.
Access to the instrument may be limited to a given time window ('SISGEO Communication Window Feature') as programmed in the options configuration register (Holding Register 0x0102).

Input registers

These registers are read via the Modbus Read Input Register command (0x04).

Address	Name	Description
0x0100	COUNT	Number of readings completed
0x0101	TYPE	With the FW ver. 2.2 the reading of this register shows the value of the holding register 0x13F (for the description please refers to the relative section of this manual)
0x0110	RawXH	Most significant part of raw AD reading for X instrument
0x0111	RawXL	Least significant part of raw AD reading for X instrument
0x0112	RawYH	Most significant part of raw AD reading for Y instrument
0x0113	RawYL	Least significant part of raw AD reading for Y instrument
0x0114	RawTH	Most significant part of raw AD reading for temperature
0x0115	RawTL	Least significant part of raw AD reading for temperature
0x0120	SinXH	Integer part of Amplitude * sin(α) reading for X inclinometer
0x0121	SinXL	Fractional part of Amplitude * sin(α) reading for X inclinometer
0x0122	SinYH	Integer part of Amplitude * sin(α) reading for Y inclinometer
0x0123	SinYL	Fractional part of Amplitude * sin(α) reading for Y inclinometer
0x0124	TemperatureH	Most significant part of temperature (ºC) reading for inclinometer
0x0125	TemperatureL	Least significant part of temperature (ºC) reading for inclinometer

When the first part of a two register value is read, the high part is latched; for example, reading SinXH latches SinXL.

Raw values may be converted to long with the following code:

```
unsigned high = RawXH;
unsigned low  = RawXL;
long RawX     = (long)((unsigned long)high << 16 | (unsigned long)low);
```

Integer/Fractional values may be converted to their float equivalent with the following code:

```
unsigned high = SinXH;
unsigned low  = SinXL;
long SinX    = (long)((unsigned long)high << 16 | (unsigned long)low);
float SinXF  = SinX/65536.0;
```

Switch off register

This is a special register used to switch the unit off. This register is written with the Modbus Write Multiple Registers command (0x10).

REG	Raw default	Default	Description
0x0010			Write: • 0xFF to this register to switch the instrument off • 0xEE to reset the instrument (the instrument should be reset after changing the configuration).

Configuration registers

These registers are read via the Modbus Read Holding Register command (0x03).

These registers are written with the following sequence using the Modbus Write Multiple Registers command (0x10):

1. Write the configuration size to register 0x0100.
2. Write 16 registers with 0x10 command starting at register 0x0101
3. Write 16 registers with 0x10 command starting at register 0x0111
4. Write 16 registers with 0x10 command starting at register 0x0121
5. Write 16 registers with 0x10 command starting at register 0x0131
6. Write 1 register with 0x10 command starting at register 0x0141

Registers hold data of different nature:

Short Integer

Short integers are 16 bit values (for example register 0x0106, Number of readings to average).
They are transmitted with the normal Modbus convention

Long integers

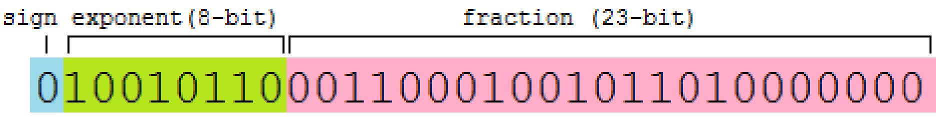
Long integers are 32 bit values (for example registers 0x0107 and 0x0108 for the calibration date).
Each register, containing 16 bits word of data is transmitted with the normal Modbus convention, but the least significant word uses the register with the lower address, so that the whole value is reconstructed, using C-like notation, as: (REG[0x0108]<<16) | REG[0x107].

Floats

Floats are stored in a manner similar to long integers, but the resulting bitmap must be interpreted according to IEEE 754.
Thus reading SArated:

register 0x0109 = 0x9680
register 0x010A = 0x4B18

The float value bitmap is 0x4B189680.



The resulting value is 10000000.0=1.0e7.

Options

The Options register (address 0x0102) is used to configure the 'SISGEO Communication Window Feature'.
When this register is set to 0 (the default value) this feature is disabled and the instrument is always on and ready to serve Modbus commands.
When a different value is set, the most significant byte is the Address Delay parameter and the least significant byte is the Warming Delay parameter.
With a value of a register value of 0x050A, we would have:

- Address Delay = 0x05 = 5
- Warming Delay = 0x0A = 10

The instrument will switch on after a time in seconds of:

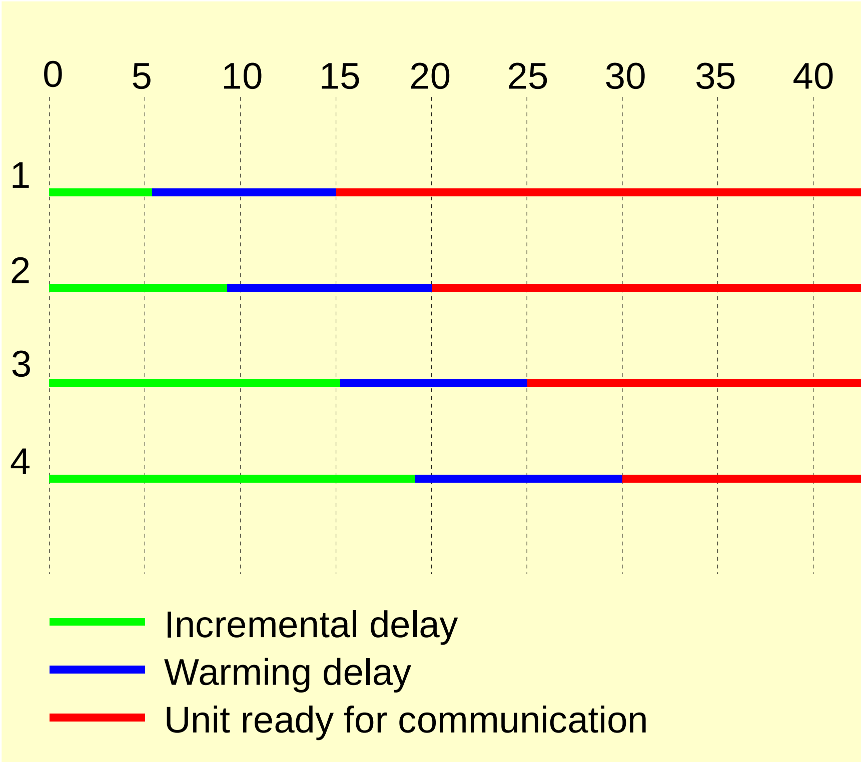
$$incrementaldelay = addressdelay * Modbusaddress$$

and will start measuring the instrument for an Warming Delay time; the measure will be available for reading on the Modbus after a total time of:

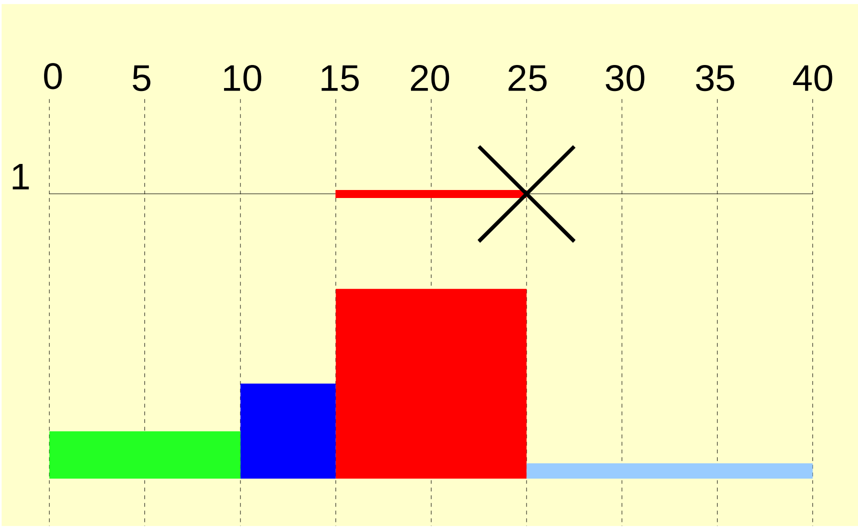
$$delay = addressdelay * Modbusaddress + warmingdelay$$

So instruments with addresses 1, 2, 3 and 4 would enable their communication interface after a delay of 15, 20, 25 and 30 when using the sample values below

The Modbus master should wait till the instrument come online, read the required values and switch them off when they are no more needed. The only way to switch on again a unit, after it has been switched off, is to cycle power off and on again.



During the delay time, instrument and AD are powered, so that this accounts as settling time for the measure. It is expected that the Modbus master will send a switch off command after reading the instrument so that power drain will switch to a minimum.



In the graph above power use is estimated assuming that at time 25 the Modbus master has sent a switch-off command.

Following the component switched ON and OFF during the 4 power state:

	Incremental Delay	Warming Delay	Communication	Switch-off Mode
Sensors and ADCs	OFF	ON	ON	OFF
RS485 interface	OFF	OFF	ON	OFF
Microcontroller	RUNNING	RUNNING	RUNNING	STOP MODE

Base configuration

REG	Raw default	Default	Description
0x0101	0x5AA5		Validity signature
0x0102	0x0000	0	Options
0x0103	0x0001	1	Modbus Address
0x0104	0x0053	1	Serial number: S000001 The serial number S123456 would be stored in registers as: 0x104: 1253 (ASCII code of S is 0x53) 0x105: 5634
0x0105	0x0100		
0x0106	0x000A	30	Number of readings averaged to obtain an instrument reading.
0x0107	0x0000	521547589	Calibration date in second since (format: 1/1/2000 00:00:00)
0x0108	0x0000		
0x0109	0xF8C0	1.612E7	SArated
0x010A	0x4B75		
0x010B	0xF8C0	1.612E7	SBrated
0x010C	0x4B75		
0x010D	0x0000	0.0	EA
0x010E	0x0000		
0x010F	0x0000	0.0	EB
0x0110	0x0000		
0x0111	0x0000	0.0	MArated
0x0112	0x0000		
0x0113	0x0000	0.0	MBrated
0x0114	0x0000		
0x0115	0x0000	0.0	XA
0x0116	0x0000		
0x0117	0x0000	0.0	XB
0x0118	0x0000		
0x0119	0x0000	0.0	T0
0x011A	0x0000		
0x011B	0x0000	0.0	T1
0x011C	0x0000		
0x011D	0x0000	0.0	T2
0x011E	0x0000		

Further content for first generation sensors

REG	Raw default	Default	Description
0x011F	0xE8D5	4.08e-4	SA0
0x0120	0x39D5		
0x0121	0x8CA4	2.2e-5	SA1
0x0122	0x37B8		
0x0123	0xA3B6	-1.1e-6	SA2
0x0124	0xB593		
0x0125	0x0000	0.0	SA3
0x0126	0x0000		
0x0127	0xE8D5	4.08e-4	SB0
0x0128	0x39D5		
0x0129	0x8CA4	2.2e-5	SB1
0x012A	0x37B8		
0x012B	0xA3B6	-1.1e-6	SB2
0x012C	0xB593		
0x012D	0x0000	0.0	SB3
0x012E	0x0000		
0x012F	0xCFAB	-0.0522/4	MA0
0x0130	0xBC55		
0x0131	0x9724	-0.0039/4	MA1
0x0132	0xBA7F		
0x0133	0xB717	0.0001/4	MA2
0x0134	0x37D1		
0x0135	0x07B0	-6.0e-7/4	MA3
0x0136	0xB421		
0x0137	0xCFAB	-0.0522/4	MB0
0x0138	0xBC55		
0x0139	0x9724	-0.0039/4	MB1
0x013A	0xBA7F		
0x013B	0xB717	0.0001/4	MB2
0x013C	0x37D1		
0x013D	0x07B0	-6.0e-7/4	MB3
0x013E	0xB421		

Further content for second (actually last) generation sensors

REG	Raw default	Default	Description
0x011F	0x877F	-3.1e-4	SA0
0x0120	0xB9A2		
0x0121	0x37BD	+3.2e-5	SA1
0x0122	0x3806		
0x0123	0x37BD	-5.0e-7	SA2
0x0124	0xB506		
0x0125	0xCC77	-5.0e-9	SA3
0x0126	0xB1AB		
0x0127	0x877F	-3.1e-4	SB0
0x0128	0xB9A2		
0x0129	0x37BD	3.2e-5	SB1
0x012A	0x3806		
0x012B	0x37BD	-5.0e-7	SB2
0x012C	0xB506		
0x012D	0xCC77	-5.0e-9	SB3
0x012E	0xB1AB		
0x012F	0x0000	0.0	MA0
0x0130	0x0000		
0x0131	0x0000	0.0	MA1
0x0132	0x0000		
0x0133	0x0000	0.0	MA2
0x0134	0x0000		
0x0135	0x0000	0.0	MA3
0x0136	0x0000		
0x0137	0x0000	0.0	MB0
0x0138	0x0000		
0x0139	0x0000	0.0	MB1
0x013A	0x0000		
0x013B	0x0000	0.0	MB2
0x013C	0x0000		
0x013D	0x0000	0.0	MB3
0x013E	0x0000		

This data (conversion parameter) are set to 0 by factory default.

REG	Raw default	Default	Description
0x013F	0x0002	2	FLAGS = Channel numbers (1 / 2)
0x0140	0x4000	20000	Amplitude used to multiply sin alfa value (in case of inclinometers)
0x0141	0x469C		

Calibration and measure

The following symbols will be used:

Raw input values	
T_{RAW}	Raw temperature measure
A_{RAW}	Raw X (AD counts)
B_{RAW}	Raw Y (AD counts)
Calibration parameters	
T_0, T_1, T_2	Correction coefficients for A axis temperature reading
SA_0, SA_1,SA_2,SA_3	Correction coefficients for A axis sensitivity
SB_0, SB_1,SB_2,SB_3	Correction coefficients for B axis sensitivity
MA_0,MA_1,MA_2,MA_3	Correction coefficients for A axis mechanical offset
MB_0,MB_1,MB_2,MB_3	Correction coefficients for B axis mechanical offset
SA_{rated}, SB_{rated}	Rated sensitivity at 20°C for A and B channels
EA,EB	Electrical offset for A and B channels
MA_{rated}, MB_{rated}	Rated mechanical offset at 20°C for A and B channels
XA,XB	Cross-axis coefficients
Intermediate results	
TA	Temperature measure in °C
SA,SB	Sensitivity at current temperature for A and B channels
MA,MB	Mechanical offset at current temperature for A and B channels
Final results	
readingA, readingB	Readings in sinα for A and B axis
final $reading_A$ final $reading_B$	Final readings corrected for cross axis

Temperature conversion

Temperature reading are corrected according to the following formulas:

$$T = T_0 + T_1 * T_{RAW} + T_2 * T_{RAW}^2$$

Temperature compensation of parameters

The parameters used for linearization are compensated for temperature according to the following formulas.

The sensitivity correction factors are calculated as follows. First correction coefficients for sensitivity are calculated:

$$S_{Acorr} = SA_0 + SA_1 * TA + SA_2 * TA^2 + SA_3 * TA^3$$

$$S_{Bcorr} = SB_0 + SB_1 * TB + SB_2 * TB^2 + SB_3 * TB^3$$

Then sensitivity is corrected as:

$$S_A = S_{Arated} * (1 + S_{Acorr})$$

$$S_B = S_{Brated} * (1 + S_{Bcorr})$$

The correction for mechanical offset is as follows:

$$M_{Acorr} = MA_0 + MA_1 * T + MA_2 * T^2 + MA_3 * T^3$$

$$M_{Bcorr} = MB_0 + MB_1 * T + MB_2 * T^2 + MB_3 * T^3$$

The mechanical offset is corrected as:

$$M_A = M_{Arated} - M_{Acorr}$$

$$M_B = M_{Brated} - M_{Bcorr}$$

Linearization and cross-axis

The readings are linearized as follows:

$$reading_A = sin(arcsin(A_{raw}/S_A - E_A) - M_A)$$
$$reading_B = sin(arcsin(B_{raw}/S_B - E_B) - M_B)$$
$$finalreading_A = reading_A - (reading_B * X_A)$$
$$finalreading_B = reading_B - (reading_A * X_B)$$

Appendix A – Programming Tips

Following you can find examples of main operation. The following examples are coded in C# and uses the nModbus library.

Reading values

To read value from inclinometers the ReadInputRegister (0x04) Modbus function must be used. There are 3 measured value (SinX,SinY and T); for each of this value 2 InputRegister are used for a total of 6 input registers.

Example 1 – Reading values

```
private byte Address485;    // 485 address of sensor to read
private Int16 SinXH;                  // Higher part of sin X
private UInt16 SinXL;       // Lower part of sin X
private Int16 SinYH;                  // Higher part of sin Y
private UInt16 SinYL;       // Lower part of sin Y
private Int16 CalTH;                  // Higher part of temperature
private UInt16 CalTL;       // Lower part of temperature
...

/// <summary>
/// Read 3 measured value and store raw value to SinXH,SinXL,SinYH,SinYL,CalTh and CalTL
/// </summary>
/// <param name="mbsm">
/// nModbus master
/// </param>
/// <returns>
/// True if read ok otherwise false
///</returns>
public bool Read(Modbus.Device.IModbusSerialMaster mbsm)
{
        try {
                ushort[] I = mbsm.ReadInputRegisters((byte)Configuration.address, 0x0120, 6);
        if (I.Length != 6) return false;
        SinXH = (Int16)I[0];
        SinXL = I[1];
        SinYH = (Int16)I[2];
        SinYL = I[3];
        CalTH = (Int16)I[4];
        CalTL = I[5];
        }
    catch (Exception) {
        return false;
        }
    return true;
}
...
/// <summary>
/// Decode lower and higher part of Sin X in to float vale
/// </summary>
public float SinX
{
        get {
        int v = SinXH;
        v <<= 16;
        v |= SinXL;
        return v / 65536.0f;
```

```
        }
}

/// <summary>
/// Decode lower and higher part of Sin Y in to float vale
/// </summary>
public float SinY
{
        get {
                int v = SinYH;
        v <<= 16;
        v |= SinYL;
        return v / 65536.0f;
        }
}

/// <summary>
/// Decode lower and higher part of Temperature in Celsius to float vale
/// </summary>
public float Temperature
{
        get {
        int v = CalTH;
        v <<= 16;
        v |= CalTL;
        return v / 65536.0f;
        }
}
```

Note:

Once the sensor is powered up it starts to read, depending on configuration the reading process can take different time (usually with 15 averages the reading process takes 3 seconds). To ensure the read process is completed you can read Input Register at address 0x100. The value returned is the count of completed read.

SISGEO suggests to wait 3 complete reading before consider the reading valid.

Input Register 0x101 contain the number of axis of the sensor (1 means only Sin X is present, 2 means both Sin X and Sin Y are present). Temperature value is always present.

Configuration

For normal operation reading the configuration is not needed. Configuration is stored in Holding registers and must be read using Modbus function ReadHoldingRegister (0x03).

To read the configuration the following sequence must be done:

1) Read Single Holding Register (Modbus function 0x03) at address 0x0100. The read value is the size of configuration in bytes.

2) Read Holding Registers (Modbus function 0x03) starting at address 0x0101 for the size read by previous step divided by 2.

The read register must fill a byte array (two bytes for each registry) and decoded according to the Table present in the specification document (Configuration registers).

Note:

You can not read or write configuration partially. Reading configuration must read the correct size.

Example 2 – Reading configuration

```
public SensorConfiguration Configuration;      // Configuration decoding class
...
/// <summary>
/// Read configuration
/// </summary>
/// <param name="mbsm">
/// nModbus master
/// </param>
/// <returns>
/// True if configuration read ok otherwise false
/// </returns>
public bool ReadConfiguration(Modbus.Device.IModbusSerialMaster mbsm)
{
            try {
            ushort[] S=mbsm.ReadHoldingRegisters((byte)Configuration.address, 0x0100, 1);
                        if (S.Length != 1 || S[0] == 0) return false;
          ushort[] D = mbsm.ReadHoldingRegisters((byte)Configuration.address, 0x0101, (ushort)(S[0] / 2));
            if (D.Length != S[0] / 2) return false;
            Configuration = new SensorConfiguration(D);
     }
     catch (Exception) {
            return false;
     }
            return true;
}
```

Writing configuration is a 5 step procedure:

1) Read Single Holding Register (Modbus function 0x03) at address 0x0100. The read value is the size of configuration in bytes.

2) Read Multiple Holding Registers (Modbus function 0x03) starting at address 0x0101 for the size read by previous step divided by 2.

3) Change any required configuration parameter.

4) Write Single Holding Register (Modbus function 0x10) at address 0x100 with the size read in step 1.

5) Write Multiple Holding Registers (Modbus function 0x10) starting at address 0x101 with a maximum of 16

registers up to finish the size to write.

Note:

You can not write configuration partially. If 0x100 Holding register is not written with the correct size value the following write to multiple register will fail.

Example 3 – Writing configuration

```
public SensorConfiguration Configuration;      // Configuration decoding class
...
/// <summary>
/// Decode lower and higher part of Sin X in to float vale
/// </summary>
/// <param name="Address">
/// 485 address of sensor
/// </param>
/// <param name="mbsm">
/// nModbus master
/// </param>
/// <returns>
/// True if configuration writed ok otherwise false
/// </returns>
public bool WriteConfiguration(byte Address, Modbus.Device.IModbusSerialMaster mbsm)
{
            try {
            ushort[] S = new ushort[1];
          S[0] = SensorConfiguration.CONFIGURATION_SIZE;
          mbsm.WriteMultipleRegisters(Address, 0x0100, S); // First write Holding register 0x0100 with size of configuration
          ushort[] UD = Configuration.UData;
          ushort[] FD = new ushort[16];
          for (int k = 0; k < 16; k++) FD[k] = UD[k];
          mbsm.WriteMultipleRegisters(Address, 0x0101, FD); // write first 16 registers starting at address 0x0101
          FD = new ushort[16];
          for (int k = 0; k < 16; k++) FD[k] = UD[16 + k];
          mbsm.WriteMultipleRegisters(Address, 0x0111, FD);
          FD = new ushort[16];
          for (int k = 0; k < 16; k++) FD[k] = UD[32 + k];
          mbsm.WriteMultipleRegisters(Address, 0x0121, FD);
          FD = new ushort[16];
          for (int k = 0; k < 16; k++) FD[k] = UD[48 + k];
          mbsm.WriteMultipleRegisters(Address, 0x0131, FD);
          FD = new ushort[1];
          for (int k = 0; k < 1; k++) FD[k] = UD[64 + k];
          mbsm.WriteMultipleRegisters(Address, 0x0141, FD); // write spare registers
            }
     catch (Exception) {
            return false;
     }
     return true;
}
```

Note:

The Example 3 is built for configuration size of 65*2 bytes for different size the correct number of 16th unsigned int and remaining must be calculate and written.

CHANGES VERSION FW 2.2

With the new firmware version it is possible:

F1) To configure the actual reading in: Sen α, degrees, millimeters for meter or inch for feet.

F2) Conversion of the readings counts (Channel A and Channel B) in electrical units (mV) using two straight lines for each channel, one for the positive readings (0/+FS) and one for the negative ones (-FS/0). The result of these conversions will be inserted in a third degree polynomial (a different polynomial for each channel).

F3) To add the holding register 0X0000 from which it is possible to obtain the firmware version.

F4) Individually read the holding register containing the configuration (from 0x101 onwards).

With the new version of the "ILM" software (available only for SISGEO internal use) it is possible:

S1) To configure the different reading modes.

S2) For the different modes of the polynomial conversion, the following numbers of decimal points are established:

- With **Amp = 1 : 6 decimals** (ex. 0.123456) will be displayed
- With **Amp = 12 : 5 decimals** (ex. 0.12345) will be displayed
- With **Amp = 90** : **4 decimals** (ex. 12.1234) will be displayed
- With **Amp = 1000 : 3 decimals** (ex. 123.123) will be displayed
- With **Amp = 20000 : 2 decimals** (ex. 1234.12) will be displayed

S3) In polynomial mode a byte of configuration is used to show a unit of measure and choose the relative number of decimals. It is used the following chart unit/decimal:

UA/UB Value	Unit of Measurement	DECIMALS
1	mV	2
2	bar	5
3	mbar	3
4	atm	5
5	psi	4
6	Pa	0
7	kPa	3
8	MPa	6
9	mmH_2O	1
10	mH_2O	4
11	inH_2O	3

12	ftH_2O	4
13	mmHg	3
14	cmHg	4
15	inHg	4
16	Kg/cm^2	5
17	Kg/m^2	1
18	lb/in^2	4
19	lb/ft^2	2
20	N/cm^2	4
21	N/m^2	0
22	t/m^2	4
23	$t(UK)/ft^2$	5
24	$t(USA)/ft^2$	5

Firmware changes

The procedure of Modbus reading (Input Register) is not subjected to changes, while the configuration will be modified as follows.

It has been added an holding register 0x0000 which returns the firmware version: the MSB it the Major Version and the LSB is the Minor Version. The value configured for this version is 2.1.

The current register 0x013F, named FLAGS (Holding Register) now has the following form:

00000000 0000MMCC

where:

CC

01 = 1 channel

10 = 2 channels

MM

00 = A * Sen α

01 = Degrees

10 = Millimeters on Meter (A=1000) / Inch on Feet (A=12)

11 = mV + Polynomial

In case MM = 11 (mV +Polynomial) the following registers (configuration registers) have the following meaning:

REG	Raw default	Default	Description
0x0109	0x0000	0.0	APQ = constant term of straight line positive side channel A
0x010A	0x0000		
0x010B	0x0000	1.0	APM = slope of straight line positive side channel A
0x010C	0x3F80		
0x010D	0x0000	0.0	AZ = counts over which it is used the straight line positive side and under which it is used the straight line negative side for channel A
0x010E	0x0000		
0x010F	0x0000	0.0	ANQ = constant term of straight line negative side channel A
0x0110	0x0000		
0x0111	0x0000	1.0	ANM = slope of straight line negative side channel A
0x0112	0x3F80		
0x0113	0x0000	0.0	BPQ = constant term of straight line positive side channel B
0x0114	0x0000		
0x0115	0x0000	1.0	BPM = slope of straight line positive side channe B
0x0116	0x3F80		
0x0117	0x0000	0.0	BZ = counts over which it is used the straight line positive side and under which it is used the straight line negative side for channel B
0x0118	0x0000		
0x0119	0x8000	-273.0	T0 = constant term of temperature calculation correction
0x011A	0xC388		
0x011B	0x36dd	2.128738E-4	T1 = 1st degree coefficient for the correction of the temperature calculation
0x011C	0x395F		
0x011D	0x0000	0.0	T2 = 2nd degree coefficient for the correction of the temperature calculation
0x011E	0x0000		
0x011F	0x0000	0.0	BNQ = constant term of straight line negative side channel B
0x0120	0x0000		

0x0121	0x0000	1.0	BNM = slope of straight line negative side channel B
0x0122	0x3F80		
0x0123	0x0000	0.0	AX0 = constant term of polynomial correction channel A
0x0124	0x0000		
0x0125	0x0000	1.0	AX1 =1st degree coefficient for the polynomial correction channel A
0x0126	0x3F80		
0x0127	0x0000	0.0	AX2 = 2nd degree coefficient for the polynomial correction channel A
0x0128	0x0000		
0x0129	0x0000	0.0	AX3 = 3rd degree coefficient for the polynomial correction channel A
0x012A	0x0000		
0x012B	0x0000	0.0	BX0 = constant term of polynomial correction channel A B
x012C	0x0000		
0x012D	0x0000	1.0	BX1 = 1st degree coefficient for the polynomial correction channel B
0x012E	0x3F80		
0x012F	0x0000	0.0	BX2 = 2nd degree coefficient for the polynomial correction channel B
0x0130	0x0000		
0x0131	0x0000	0.0	BX3 = 3rd degree coefficient for the polynomial correction channel B
0x0132	0x0000		
0x0133	0x0000	0.0	AH = coefficient of temperature correction millivolt (mV) reading channel A
0x0134	0x0000		
0x0135	0x0000	1.0	AK = coefficient of temperature correction millivolt (mV) reading channel A
0x0136	0x3F80		
0x0137	0x0000	0.0	BH = coefficient of temperature correction millivolt (mV) reading channel B
0x0138	0x0000		
0x0139	0x0000	1.0	BK = coefficient of temperature correction millivolt (mV) reading channel B
0x013A	0x3F80		
0x013B	0x0000	0	UA/UB = Unit of measure see table (S3) (LSB = UA, MSB = UB)

0x013C	0x0000		
0x013D	0x0000	0.0	Not used
0x013E	0x0000		

The calculation is performed with the following procedure.

With AR and BR being respectively the readings in counts of the AD for channel A and channel B.

CHANGES VERSION FW 2.3

With the new firmware version has been reduced the time to switch off the sensor to 100 ms

CONVERSION FROM COUNTS TO ELECTRICAL UNIT (mV)

CHANNEL A

for AR >= AZ

AmV = AR * APM + APQ

for AR < AZ

AmV = AR * ANM + ANQ

CHANNEL B

for BR >= BZ

BmV = BR * BPM + BPQ

for BR < AZ

BmV = BR * BNM + BNQ

TEMPERATURE CONVERSION

Being T_{raw} the counts relative to the temperature, read from the A/D

$$T = T_0 + T_1 * T_{RAW} + T_2 * T_{RAW}^2$$

TEMPERATURE CORRECTION READING mV

AmV and BmV represent the readings in mV of both channels A and B.

AmVC = (AH * T + AK) * AmV

BmVC = (BH * T + BK) * BmV

where T is the temperature measured in Celsius (°C) degrees.

NOTE. If you set up AH and BH at 0 and AK and BK at 1, no temperature correction is performed.

CONVERSION FROM ELECTRICAL UNIT (mV) TO PHYSICS UNIT

AmVC and BmVC represent the readings in mV of channels A and B, corrected in temperature.

$$finalreading_A = AmVC^3 * AX3 + AmVC^2 * AX2 + AmVC * AX1 + AX0$$
$$finalreading_B = BmVC^3 * BX3 + BmVC^2 * BX2 + BmVC * BX1 + BX0$$

The calibration procedure is divided into two phases. The first phase calculate the correction in mV. Then, it will be possible both to insert the calibration parameters of the sensor and to perform its specific calibration.

It has been added Input Register for the numeric representation according to standard IEEE 754

The previous method of conversion with codification:

```
unsigned high = SinXH;
unsigned low  = SinXL;
long SinX    = (long)((unsigned long)high << 16 | (unsigned long)low);
float SinXF  = SinX/65536.0;
```

limited the decimal precision to 1/65535. Existing a polynomial conversion that could convert the value in any numeric representation (ex: the conversion in bar and sen alfa configuration with amp 1 requires 5 decimals), it has been introduced the conversion according to standard IEEE 754.
Here follows an example C# of the conversion:

```
private float Convert(Int16 SinXH, UInt16 SinXL) {

    if ((Configuration.Flags &
SensorConfiguration.CFG_SENSOR_FLAG_MEASURE_MODE_FLOAT) ==
            SensorConfiguration.CFG_SENSOR_FLAG_MEASURE_MODE_FLOAT) {
```

```
        byte[] bh = BitConverter.GetBytes(SinXH);
        byte[] bl = BitConverter.GetBytes(SinXL);
        byte[] bf = new byte[4];
        bf[0] = bl[0];
        bf[1] = bl[1];
        bf[2] = bh[0];
        bf[3] = bh[1];
        return BitConverter.ToSingle(bf, 0);
    }
    else {
        int v = h;
        v <<= 16;
        v |= l;
        return v / 65536.0f;
    }
}
```

If it is not required a decimal precision superior to 1/65535, you could use the previous codification.

To maintain the compatibility with the previous versions of the firmware, the following registers have been added:

REG	Content	Description
0x0126	Converted value X Float IEEE MSB	Value converted axis X according the codification Float IEEE 754
0x0127	Converted value X Float IEEE LSB	
0x0128	Converted value Y Float IEEE MSB	Value converted axis Y according the codification Float IEEE 754
0x0129	Converted value Y Float IEEE LSB	
0x012A	Converted value Temperature Float IEEE MSB	Value converted temperature according the codification Float IEEE 754
0x012B	Converted value Temperature Float IEEE LSB	

In these registers the reading in format IEEE 754 are saved.

Out of range and reading errors A/D

The controls for out of range and for reading errors of A/D have been introduced. Here follow the values returned from the input registers (see previous charts for the description of the single registers) according to the out of limits found:

Out of scale	*Register (Hex)*	*Value (uint16 Hex)*	*Value*
AD FAILURE	0x0110	7FFF	+2147483647
	0x0111	FFFF	
OVERFLOW	0x0110	7FFF	+2147483647
	0x0111	FFFF	
UNDERFLOW	0x0110	8000	-2147483648
	0x0111	0000	
AD FAILURE	0x0112	7FFF	+2147483647
	0x0113	FFFF	
OVERFLOW	0x0112	7FFF	+2147483647
	0x0113	FFFF	
UNDERFLOW	0x0112	8000	-2147483648
	0x0113	0000	
AD FAILURE	0x0114	7FFF	+2147483647
	0x0115	FFFF	
OVERFLOW	0x0114	7FFF	+2147483647
	0x0115	FFFF	
UNDERFLOW	0x0114	8000	-2147483648
	0x0115	0000	
AD FAILURE	0x0120	7FFF	+2147483647
	0x0121	FFFF	
OVERFLOW	0x0120	7FFF	+2147483647
	0x0121	FFFF	

Out of scale	*Register (Hex)*	*Value (uint16 Hex)*	*Value*
UNDERFLOW	0x0120	8000	-2147483648
	0x0121	0000	
AD FAILURE	0x0122	7FFF	+2147483647
	0x0123	FFFF	
OVERFLOW	0x0122	7FFF	+2147483647
	0x0123	FFFF	
UNDERFLOW	0x0122	8000	-2147483648
	0x0123	0000	
AD FAILURE	0x0124	7FFF	+2147483647
	0x0125	FFFF	
OVERFLOW	0x0124	7FFF	+2147483647
	0x0125	FFFF	
UNDERFLOW	0x0124	8000	-2147483648
	0x0125	0000	
AD FAILURE	0x0126	7FFF	NaN
	0x0127	FFFF	
OVERFLOW	0x0126	7F80	+Infinity
	0x0127	0000	
UNDERFLOW	0x0126	FF80	-Infinity
	0x0127	0000	
AD FAILURE	0x0128	7FFF	NaN
	0x0129	FFFF	
OVERFLOW	0x0128	7F80	+Infinity
	0x0129	0000	
UNDERFLOW	0x0128	FF80	-Infinity
	0x0129	0000	

Out of scale	*Register (Hex)*	*Value (uint16 Hex)*	*Value*
AD FAILURE	0x012A	7FFF	NaN
	0x012B	FFFF	
OVERFLOW	0x012A	7F80	+Infinity
	0x012B	0000	
UNDERFLOW	0x012A	FF80	-Infinity
	0x012B	0000	