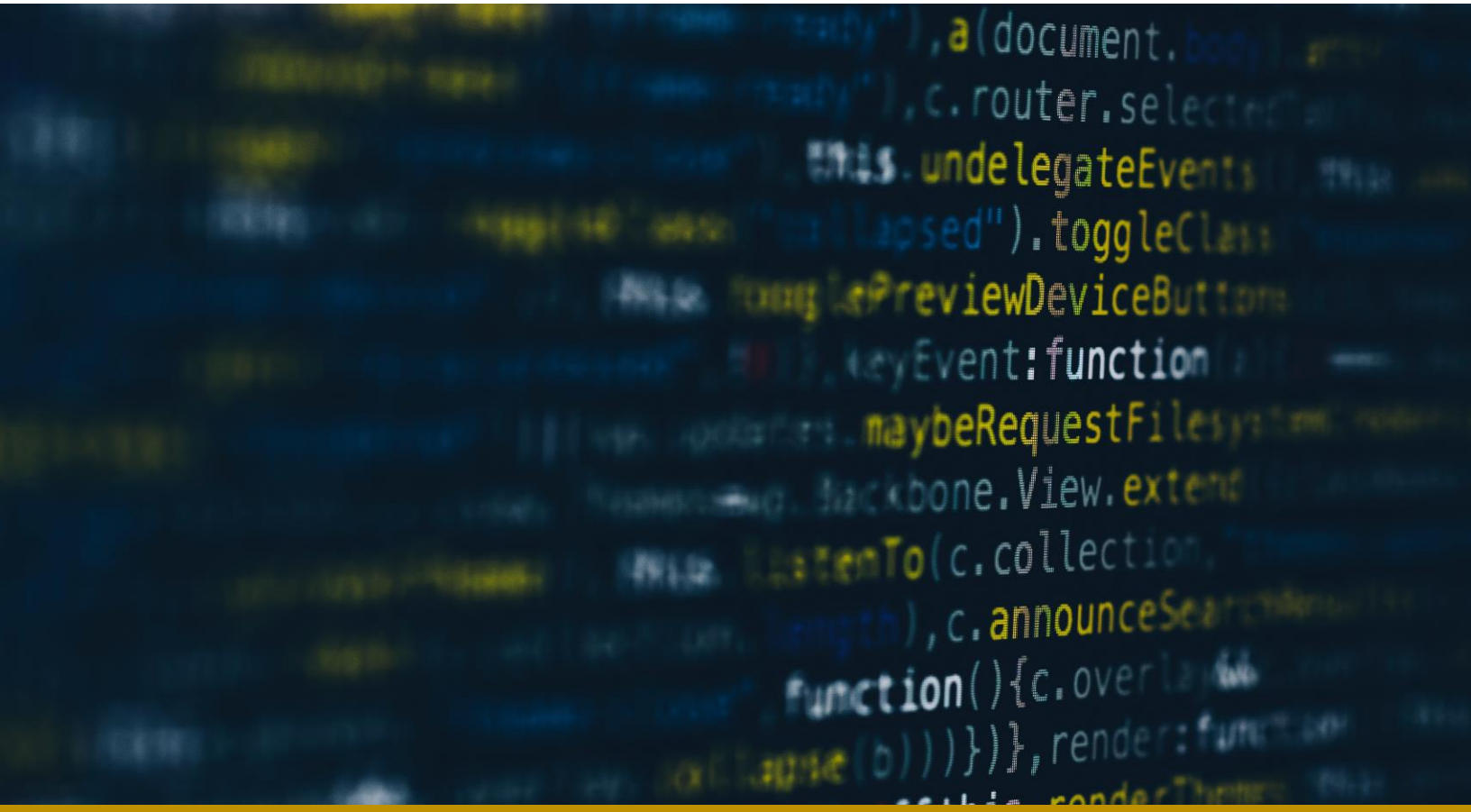
csstel



AN-862000

RS-485 Modbus Device Integration

Document No.: AN-862000, Rev 1.00
Applicable Products: ComView NX(L/X)
Contact: support@csstel.com
Web: www.csstel.com

Introduction

ComView provides two RS-485 ports to support up to 64 Modbus devices that range from simple sensors (such as temperature sensor) to more complex devices (such as smart power meter).

Modbus is a commonly used client-server messaging protocol that is implemented in a wide range of devices for data communication. To support Modbus, the device manufacturers/developers implement a Modbus register map and Modbus function codes specific to their own devices. Such information is provided to the users so that specific Modbus drivers can be developed to communicate and to read/write data with the device. Therefore, to support a Modbus device, customized software development is typically required which adds to project delay and costs.

ComView offers RS-485 app whose frontend is uniquely implemented to simplify the integration of Modbus devices with ComView solutions without having to customize or develop specific drivers to support such devices. This application note is intended to illustrate this Modbus implementation.

This application note does not provide detailed description of how to use ComView, its connectivity and configuration, and other supporting information, as these are beyond the scope of this document. Refer to other resources for more details.

References:

[1]. ComView - User Guide

[2]. Modbus Specifications - https://modbus.org/specs.php

Modbus Overview

Modbus is client-server messaging protocol for communication between devices. The device that initiates data request is called Modbus client (e.g., ComView) and the device that initiates the response is called Modbus server (e.g., power meter). In an RS-485 Modbus network, there is one client and up to 247 servers, each with a unique server ID/address from 1 to 247.

Modbus bases its data model on four primary tables to form a register map as shown below. Modbus function codes are used to access registers within these tables.

Modbus Register Map

Coil/Register Numbers	Data Addresses	Type	Table Name
1-9999	0000 to 270E	Read-Write	Discrete Output Coils (1-bit)
10001-19999	0000 to 270E	Read-Only	Discrete Input Contacts (1-bit)
30001-39999	0000 to 270E	Read-Only	Input Registers (2-byte)
40001-49999	0000 to 270E	Read-Write	Holding Registers (2-byte)

Modbus protocol defines three categories of function codes: public, user-defined, and reserved. A subset of public function codes most recognized by Modbus devices are shown in the table below:

Commonly Used Public Function Codes

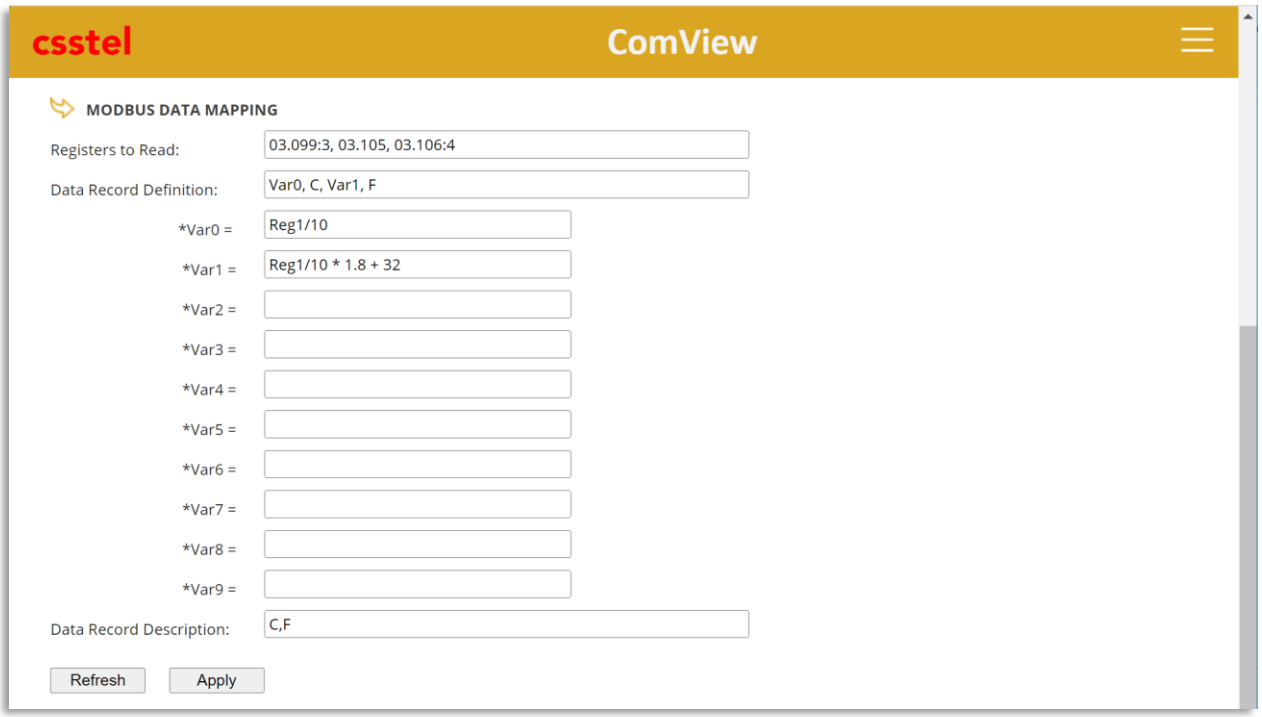
Function Code	Action	Table Name
01 (01 hex)	Read	Discrete Output Coils
05 (05 hex)	Write single	Discrete Output Coil
15 (0F hex)	Write multiple	Discrete Output Coils
02 (02 hex)	Read	Discrete Input Contacts
04 (04 hex)	Read	Input Registers
03 (03 hex)	Read	Holding Registers
06 (06 hex)	Write single	Holding Register
16 (10 hex)	Write multiple	Holding Registers

To access a Modbus register in a table, a function code designated for that table is used, e.g., function codes 01, 05, and 15 are used with Discrete Output table. To support a Modbus device, users would require a technical document that specifies the device register map and its supported function codes from the device manufacturer to develop a software driver for such device. ComView RS-485 app helps simplify this process.

Modbus Device Support

ComView RS-485 app offers users a unique implementation for Modbus device support. It simply lets users define the Modbus function code/register pairs that are required to read specific data from the device register map. Furthermore, RS-485 app also lets users define data record format and apply mathematical expressions to convert register values to meaningful data.

Users can conveniently set up Modbus device support by logging on ComView via its web interface and navigate to 'CONFIGURATION -> RS-485 APP'. A sample web page is shown below:



Registers to Read:

Description: To define a list of Modbus function code and register pairs for RS-485 app to read data from the Modbus device registers in accordance with the device technical document on Modbus

Usage: Enter CSV-string in 'FC.Reg0, FC.Reg1, …, FC.RegA:c, FC.RegN' format:

- FC: Modbus function code number in hex value (e.g., '03', function code 03 to read holding register)
- RegA: Modbus register address in decimal value (e.g., '40100', holding register 40100)
- RegA:c : consecutive Modbus registers, 'c' as total number of registers (e.g., '40100:3' represents register addresses 40100, 40101, and 40102)

Note: The sample entry '03.099:2, 03.105, 03.106:4' translates to a series of function code 03 to read 7 Modbus registers: 03.99, 03.100, 03.105, 03.106, 03.107, 03.108, 03.109. The values of these registers are mapped to 'Reg[0..N]' in consecutive sequence as follows:

- Reg0 = value of Modbus register 99
- Reg1 = value of Modbus register 100
- Reg2 = value of Modbus register 105
- Reg3 = value of Modbus register 106
- Reg4 = value of Modbus register 107
- Reg5 = value of Modbus register 108
- Reg6 = value of Modbus register 109

These register labels are then used in '*Var' expressions.

Data Record Definition:

Description: To define fields in CSV-formatted data record for RS-485 device

Usage: Enter multi-line strings in the following format:
Var0, UoM0, Var1, UoM1,…,Var9,UoM9
*Var0 = expression of (Reg[0..N])
…
*Var9 = expression of (Reg[0..N])

- Reg[0..N]: values of Modbus registers
- *Var[0..9]: string format representing expression of Reg[0..N] using arithmetic operators and/or IEEE-754 32-bit floating point conversion expression:
  - '+': addition
  - '-': subtraction
  - '*': multiplication
  - '/': division
  - '^': exponential
- 'f32(RegA,RegB)': IEEE-754 32-bit floating point conversion

Data Record Description:

Description: A placeholder intended for visual interpretation of data fields in CSV-formatted data record for RS-485 device. It is non-functional and is for references only.

Usage: Enter CSV-string in 'Label0, Label1, …, LabelN' format:

- Label[0..N]: user-definable text for data field [0..N]

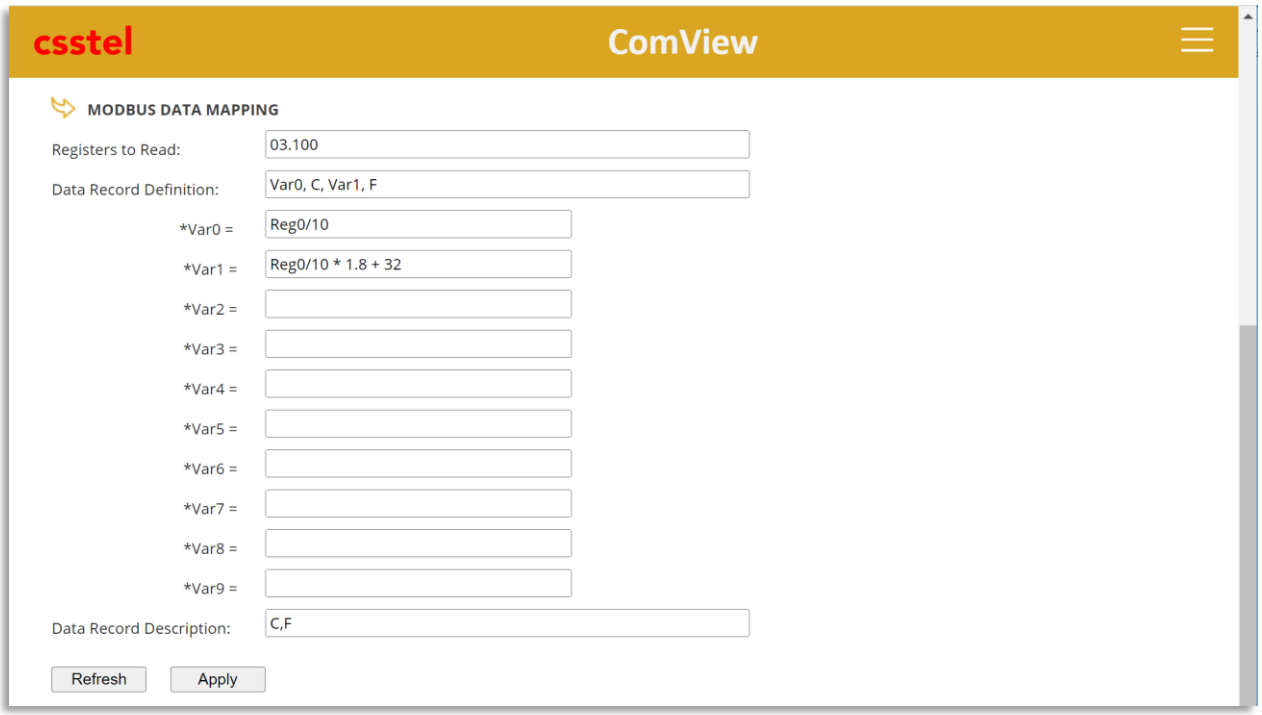
Example - Temperature Sensor Support

This section shows an example of how to configure ComView RS-485 app to read current temperature from a Modbus temperature sensor.

In this example, the sensor has the following subset of Modbus registers as extracted from its technical manual (with columns as registers, access type, function codes, and description):

5	read, write	0x03, 0x06, 0x10	**Communication protocol**[1] Allows the user to switch over to Spinel protocol. After sending the response, the device switches over to the desired protocol. (Each protocol is equipped with an instruction for switching between protocols.) Code for *Spinel: 0x0001 (default)* Code for Modbus RTU: 0x0002
99	read	0x03	**Temperature Status** 0x0000 … the value is valid others … the value is invalid
100	read	0x03	**Current Temperature** This value can be used to calculate the currently measured temperature: *temperature = value / 10* The increment of the resulting temperature is 0.1°C.
101	read	0x03	**RAW value** Value as it was received from the sensors.
105	read	0x03	**Sensor status ID** It can be of these values: 0x0000.....Error, ID is invalid. 0x0001.....ID is just being read. 0x00FF.....ID is valid
106 – 109	read	0x03	**Sensor ID** Temperature sensor unique identifier. Validity is in the previous register.

From the above, the register of interests is 100 for current temperature. It is in ºC only. To get actual current temperature in ºC, the value read must be divided by 10. To get temperature in ºF, calculation is required by using the formula "F=C*1.8 + 32". The following web page shows the user entries defined for this requirement.



Registers to Read:

Value: 03.100

Note: Use function code 03 to read Modbus register 100 and its value mapped to 'Reg0' for use in Var expression

Data Record Definition:

Value: Var0, C, Var1, F

Note:
```
*Var0 = Reg0/10          ; register 100 divided by 10 to get current temperature
*Var1 = Reg0/10*1.8 + 32 ; convert ºC to ºF
```

Data Record Description:

Value: C,F

Note: Values represent current temperature in ºC and in ºF

The above illustrates how ComView RS-485 app lets users define simple expressions to read Modbus register in ºC and convert to ºF to support a Modbus temperature sensor, such as for temperature monitoring requirements, without having to develop any software driver for it.

Example – Power Meter Support

This section shows an example of how to configure ComView RS-485 app to read various phase-1 power variables (line voltage, current, power, and frequency) from a Modbus power meter.

In this example, the power meter has the following subset of Modbus registers as extracted from its technical manual:

1.2 Input register

Input registers are used to indicate the present values of the measured and calculated electrical quantities. Each parameter is held in two consecutive16 bit register.The following table details the 3X register address, and the values of the address bytes within the message. A (*) in the column indicates that the parameter is valid for the particular wiring system. Any parameter with a cross(X) will return the value zero. Each parameter is held in the 3X registers. Modbus Protocol function code 04 is used to access all parameters.

For example, to request: Amps 1 Start address=0006
No. of registers =0002
Amps 2 Start address=0008
No. of registers=0002

Each request for data must be restricted to 40 parameters or less. Exceeding the 40 parameter limit will cause a Modbus Protocol exception code to be returned.

1.2.1 SDM630Modbus Input Registers

Address (Register)	Parameter Number	SDM630Modbus Input Register Parameter		Modbus Protocol Start Address Hex		3 Ø	3 Ø	1 Ø
		Description	Units	Hi Byte	Lo Byte	4 W	3 W	2 W
30001	1	Phase 1 line to neutral volts.	Volts	00	00	√	X	√
30003	2	Phase 2 line to neutral volts.	Volts	00	02	√	X	X
30005	3	Phase 3 line to neutral volts.	Volts	00	04	√	X	X
30007	4	Phase 1 current.	Amps	00	06	√	√	√
30009	5	Phase 2 current.	Amps	00	08	√	√	X
30011	6	Phase 3 current.	Amps	00	0A	√	√	X
30013	7	Phase 1 power.	Watts	00	0C	√	X	√
30015	8	Phase 2 power.	Watts	00	0E	√	X	√
30017	9	Phase 3 power.	Watts	00	10	√	X	X
30019	10	Phase 1 volt amps.	VA	00	12	√	X	√
30021	11	Phase 2 volt amps.	VA	00	14	√	X	X
30023	12	Phase 3 volt amps.	VA	00	16	√	X	X

30025	13	Phase 1 reactive power.	VAr	00	18	√	X	√
30027	14	Phase 2 reactive power.	VAr	00	1A	√	X	X
30029	15	Phase 3 reactive power.	VAr	00	1C	√	X	X
30031	16	Phase 1 power factor (1).	None	00	1E	√	X	√
30033	17	Phase 2 power factor (1).	None	00	20	√	X	X
30035	18	Phase 3 power factor (1).	None	00	22	√	X	X
30037	19	Phase 1 phase angle.	Degrees	00	24	√	X	√
30039	20	Phase 2 phase angle.	Degrees	00	26	√	X	X
30041	21	Phase 3 phase angle.	Degrees	00	28	√	X	X
30043	22	Average line to neutral volts.	Volts	00	2A	√	X	X
30047	24	Average line current.	Amps	00	2E	√	√	√
30049	25	Sum of line currents.	Amps	00	30	√	√	√
30053	27	Total system power.	Watts	00	34	√	√	√
30057	29	Total system volt amps.	VA	00	38	√	√	√
30061	31	Total system VAr.	VAr	00	3C	√	√	√
30063	32	Total system power factor (1).	None	00	3E	√	√	√
30067	34	Total system phase angle.	Degrees	00	42	√	√	√
30071	36	Frequency of supply voltages.	Hz	00	46	√	√	√
30073	37	Total Import kWh	kWh	00	48	√	√	√
30075	38	Total Export kWh.	kWh	00	4A	√	√	√
30077	39	Total Import kVArh .	kVArh	00	4C	√	√	√
30079	40	Total Export kVArh .	kVArh	00	4E	√	√	√
30081	41	Total VAh	kVAh	00	50	√	√	√

The format for each byte in RTU mode is:

Coding System:	8-bit per byte
Data Format:	4 bytes (2 registers) per parameter. Floating point format (to IEEE 754) Most significant register first (Default). The default may be changed if required -See Holding Register "Register Order" parameter.
Error Check Field:	2 byte Cyclical Redundancy Check (CRC)
Framing:	1 start bit 8 data bits, least significant bit sent first 1 bit for even/odd parity (or no parity) 1 stop bit if parity is used; 1 or 2 bits if no parity

From the above, the registers of interests are:

30001 Phase 1 line to neutral voltage (Volts), start address 0000, 2 registers (4-byte data)

30007 Phase 1 current (Amps), start address 0006 (0x0006), 2 registers (4-byte data)

30013 Phase 1 power (Watts), start address 0012 (0x000C), 2 registers (4-byte data)

30071 Frequency of supply voltages (Hz), start address 0070 (0x0046), 2 registers (4-byte data)

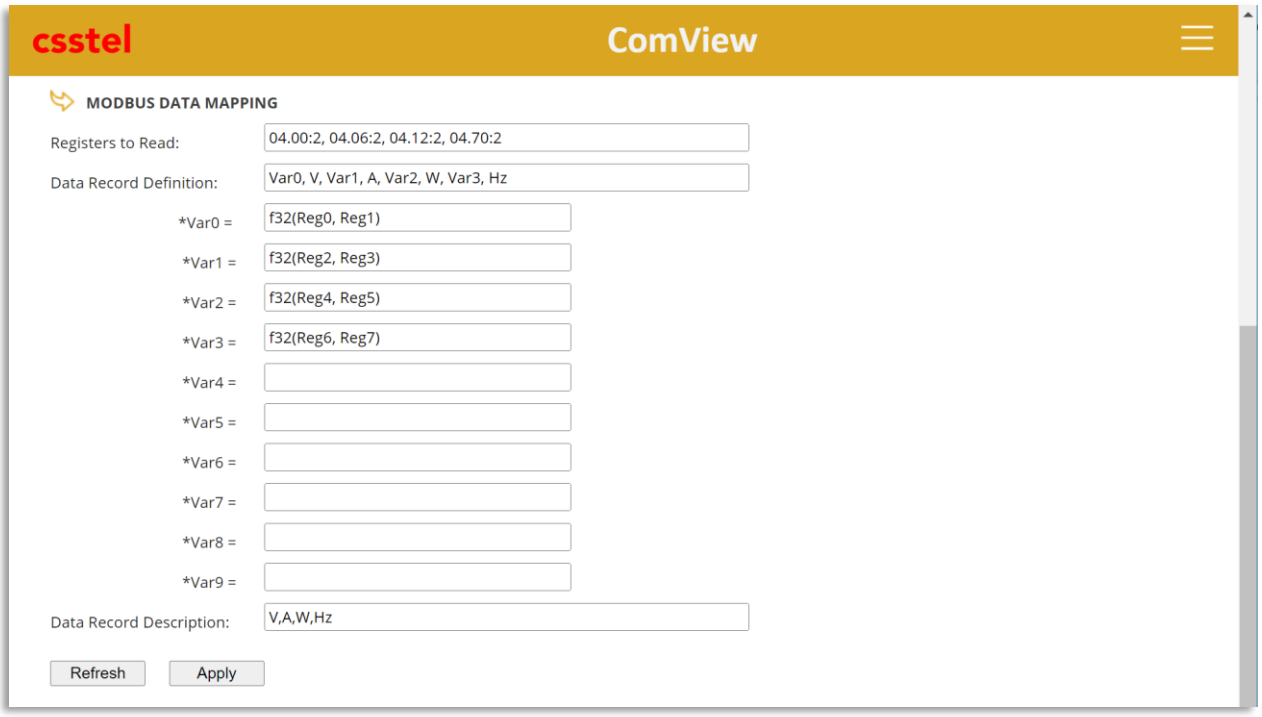
As specified in the device technical manual, Modbus function code 0x04 is used to read the above 3xxxx Input registers.

To read those registers, the Modbus function code and register pair definitions would be:

```
04.00:2     ; to read Phase 1 line to neutral voltage, register 30000 & 30001
04.06:2     ; to read Phase 1 current, register 30006 & 30007
04.12:2     ; to read Phase 1 power, register 30012 & 30013
04.70:2     ; to read frequency, register 30070 & 30071
```

Additionally, data format is in 4-byte IEEE 754 floating point and therefore they must be converted to decimal values.

The following web page shows the user entries defined for this requirement.



Registers to Read:

Value: 04.00:2, 04.06:2, 04.12:2, 04.70:2

Note: Use function code 04 to read Modbus registers and their values are mapped to 'Reg[0..N', as follows:

- Reg0 = value of Modbus register 30000
- Reg1 = value of Modbus register 30001
- Reg2 = value of Modbus register 30006
- Reg3 = value of Modbus register 30007
- Reg4 = value of Modbus register 30012
- Reg5 = value of Modbus register 30013
- Reg6 = value of Modbus register 30070
- Reg7 = value of Modbus register 30071

These register labels are then used in '*Var' expressions.

Data Record Definition:

Value:	Var0, V, Var1, A, Var2, W, Var3, Hz	
Note:	*Var0 = f32(Reg0, Reg1)	; IEEE-754 32-bit floating point conversion for V
	*Var1 = f32(Reg2, Reg3)	; IEEE-754 32-bit floating point conversion for A
	*Var2 = f32(Reg4, Reg5)	; IEEE-754 32-bit floating point conversion for W
	*Var3 = f32(Reg6, Reg7)	; IEEE-754 32-bit floating point conversion for Hz

Data Record Description:

Value:	V,A,W,Hz
Note:	Values represent V,A,W,Hz measurements

The above illustrates how ComView RS-485 app lets users define simple expressions to read Modbus registers with data in 4-byte IEEE 754 floating point and perform data conversions to decimal values to support a Modbus power meter, such as for power monitoring requirements, without having to develop any complex software driver for it.

Summary

This application note illustrates how ComView RS-485 app helps users quickly and easily define expressions to provide support for a Modbus device so that it can be integrated into ComView solutions to meet user operational requirements. Users can define Modbus function code/register pairs based on the Modbus device manufacturer specifications to read data from its Modbus registers and to define mathematical expressions to perform data conversions. With this unique implementation of Modbus support provided by ComView, most Modbus devices can be readily supported without the need for customizing or developing Modbus drivers, helping simplify device integration and reduce costs.

About CSSTEL

CSSTEL is a privately held developer and manufacturer of ComView hardware and software solutions for secure, remote infrastructure site management since 1997 with installations in over 30 countries around the world.

We offer ComView solutions that are scalable and customizable to monitor and manage virtually the entire spectrum of remote site infrastructure and site conditions.

We help telecom service providers, carriers, financial institutions, healthcare providers, government agencies, utilities, and other public and private sector organizations maintain constant visibility and control over their remote site infrastructure.

IMPORTANT:

Revision History

Revision	Date	Description
1.00	2023-01-08	Initial release

*** End of document ***