APPLICATION NOTE

Data Logger

Using Campbell Scientific Data Loggers as Modbus Servers in a SCADA Network

Revision: 06/2025

RELIABLE MONITORING SINCE 1974

Campbell SCIENTIFIC®

Table of Contents

1. Introduction

> NOTE:
> `ModbusServer()` was formerly `ModbusSlave()`. Campbell Scientific, in conjunction with the Modbus Organization, is now using "client-server" to describe Modbus communications. The Modbus client(s) initiates communications and makes requests of server device(s). Server devices process requests and return an appropriate response (or error message). For more information, see: www.modbus.org. Older data logger programs with the old terminology will still compile and run in the data logger.

This document provides quick guidance for enabling and configuring Modbus communications with your SCADA system. Because of the flexible nature of the hardware, Modbus functionality in the data logger must be enabled through configuration or programming.

Most Campbell Scientific data loggers can function as Modbus servers. For serial connections, they use the Modbus RTU protocol; for IP connections, they use Modbus TCP. In general, any available communications port can be configured for Modbus communications. For specific capabilities, refer to the specification sheet for the data logger model.

Campbell Scientific designs products to be compliant with the official Modbus specifications. The specifications may be downloaded from www.modbus.org. Functionality is tested with ***Modbus Poll*** by Witte Software.

2. Necessary software

This guide describes how to add Modbus server functionality to a data logger using a program generated by ***Short Cut***, *version 4.8*. Earlier versions of ***Short Cut*** may not support Modbus functionality for current data logger models. Communications settings are configured using ***Device Configuration Utility***. For more complex applications, you can customize the ***Short Cut***–generated program using the ***CRBasic Editor***, which is included with ***LoggerNet*** and ***PC400***. All Campbell Scientific software can be downloaded or updated from: www.campbellsci.com/downloads.

3. Physical connections

Common physical connection options include RS-232, TTL-level RS-232, RS-485, and Ethernet. Wireless options such as Wi-Fi are also available. Ensure proper surge protection for all cabled connections. When connecting systems with significantly different ground potentials, optical isolation may be necessary. Third-party optical isolators are available for RS-232, RS-485, and Ethernet. For more information on maximum distances for RS-232 and RS-485, see Maximum Distances for RS-232 and RS-485.

For information on proper grounding, see these videos:

- Understanding the Importance of Grounding
- Connecting a Data Acquisition System to Earth Ground
- Avoiding Ground Loops

By default, serial connections use 115200 baud, no parity, and 1 stop bit. These settings can be changed through data logger configuration as needed.

The data logger requires a stable DC power supply. For maximum reliability, a battery-backed supply such as the PS150 is recommended. Refer to the data logger specifications for detailed power requirements. Also see: Power Supplies.

Data loggers are typically mounted on a 1-inch grid backplate inside a Campbell Scientific enclosure. For third-party enclosures, contact Campbell Scientific for a scaled footprint drawing.

4. Supported function codes

As a Modbus server, the data logger responds to data requests from a Modbus client. Function codes define the type of data exchanged. The supported Modbus function codes are 01, 02, 03, 04, 05, 06, 15, and 16. For more details, refer to the `ModbusClient()` instruction in the CRBasic Help.

Table 4-1: Modbus function codes

Code	Name	Description
01	Read Coil/Port Status	Reads the On/Off status of discrete output(s) in the ModbusServer
02	Read Input Status	Reads the On/Off status of discrete input(s) in the ModbusServer
03	Read Holding Registers	Reads the binary contents of holding register(s) in the ModbusServer
04	Read Input Registers	Reads the binary contents of input register(s) in the ModbusServer
05	Force Single Coil/Port	Forces a single Coil/Port in the ModbusServer to either On or Off
06	Write Single Register	Writes a single register value (16-bit long) to a ModbusServer. (ModbusOption must be set to 1)
15	Force Multiple Coils/Ports	Forces multiple Coils/Ports in the ModbusServer to either On or Off
16	Write Multiple Registers	Writes values into a series of holding registers in the ModbusServer

5. Register mapping

Due to their multipurpose design, Campbell Scientific data loggers do not use a fixed Modbus register map. Instead, the user defines the register mapping in the data logger program. This process is straightforward using ***Short Cut***. By default, all variables are 32-bit floating-point numbers in CDAB byte order, though other data types and byte orders can be used.

6. Comprehensive example

This comprehensive example shows how to configure a CR1000Xe-based weather station to be a Modbus server device. The same approach applies to several other Campbell Scientific data logger models.

6.1 Entering IP settings using *Device Configuration Utility*

If using an IP connection for communications, configure the data logger for the network before connecting. Once connected with ***Device Configuration Utility***, the IP settings may be changed on the **Deployment** > **Ethernet** tab as shown in Figure 6-1 (p. 4).

If using a Campbell Scientific network link interface—such as an NL201 or NL241—connected to the **CS I/O** port, enter the IP settings on the **CS I/O IP** tab instead.

If you're unsure what to enter, contact your network administrator. For more information, refer to the data logger or network interface manual.

Also see: IP Networking and Data Loggers Part 1 and IP Networking a Data Loggers Part 2.

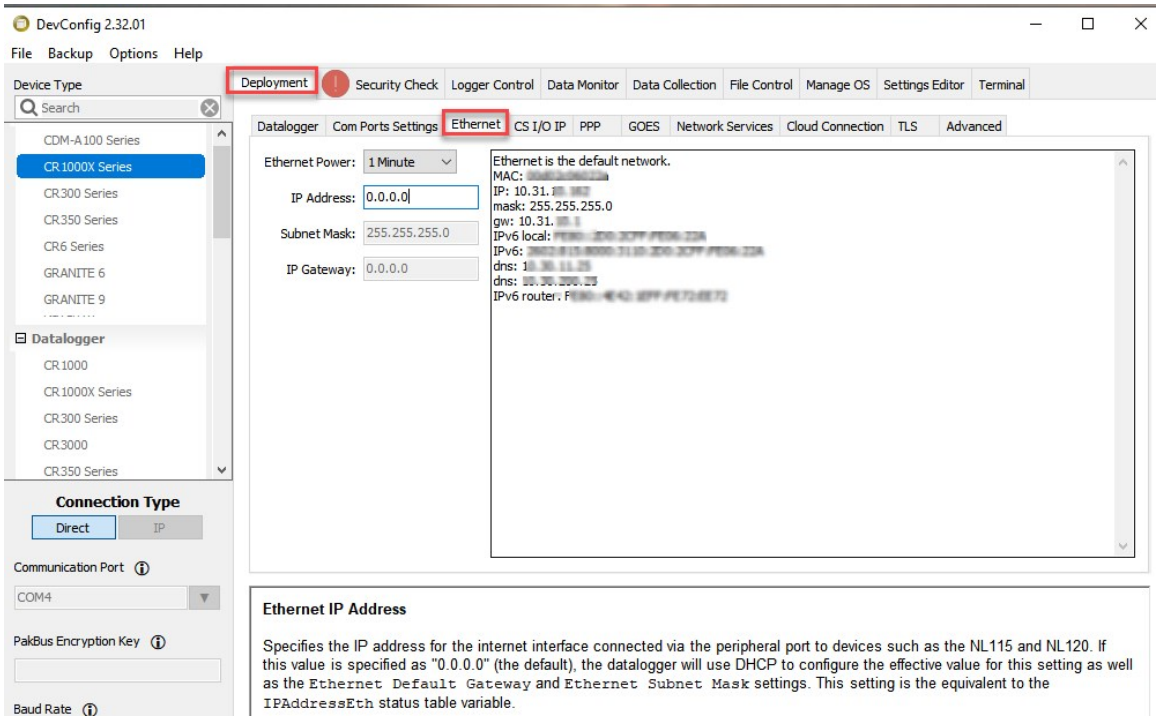
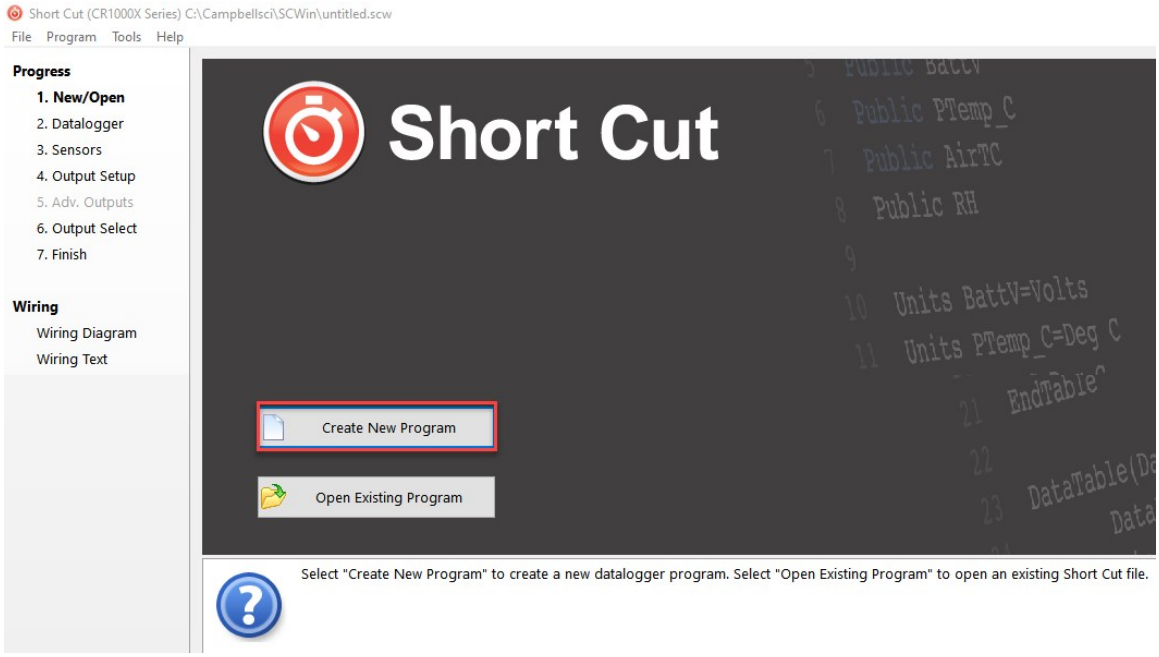


*Figure 6-1. Entering IP settings in **Device Configuration Utility***

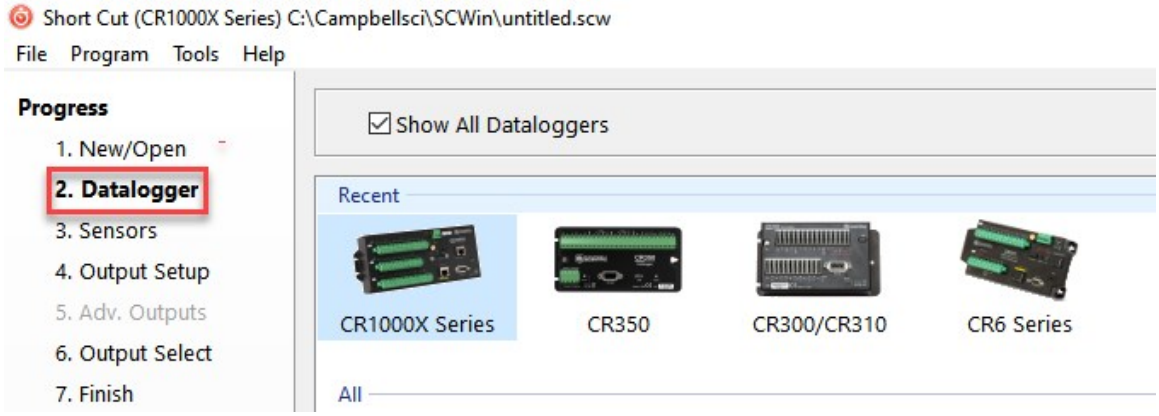
6.2 Adding Modbus server functionality with *Short Cut*

This section demonstrates creating a data logger program in ***Short Cut***. For a ***Short Cut*** tutorial, see the data logger manual or visit www.campbellsci.com/videos.

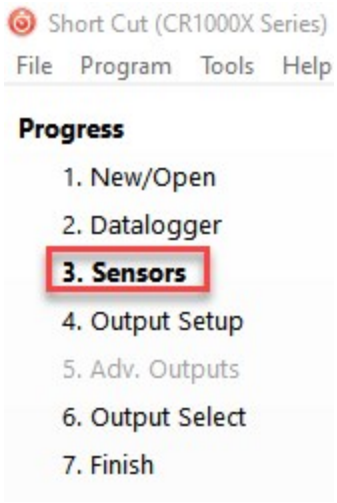
1. Open ***Short Cut*** and select **Create New Program**



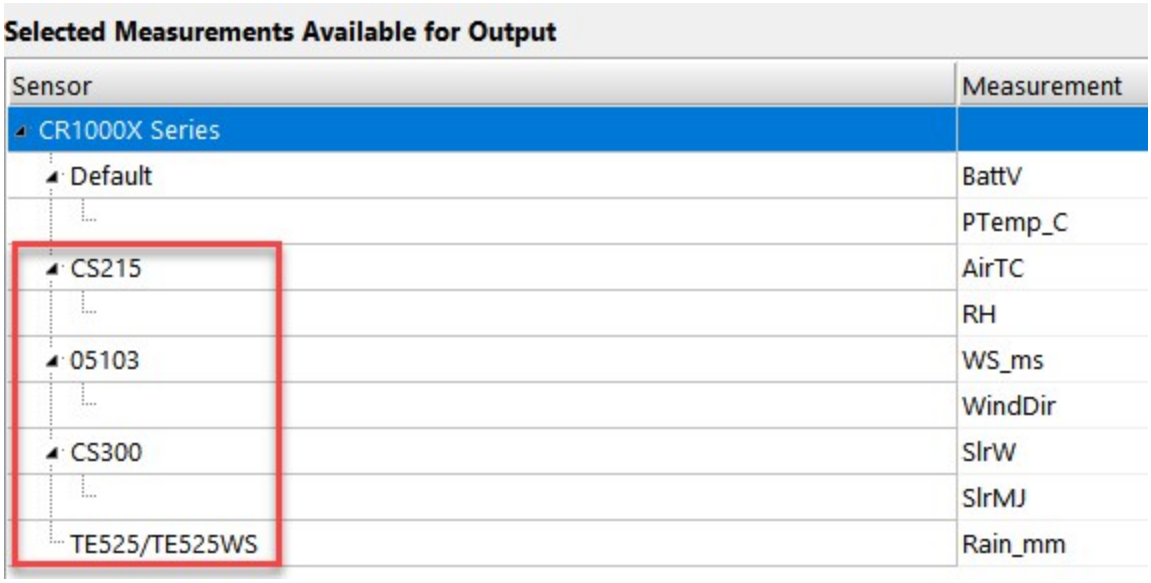
2. Select your specific data logger model.



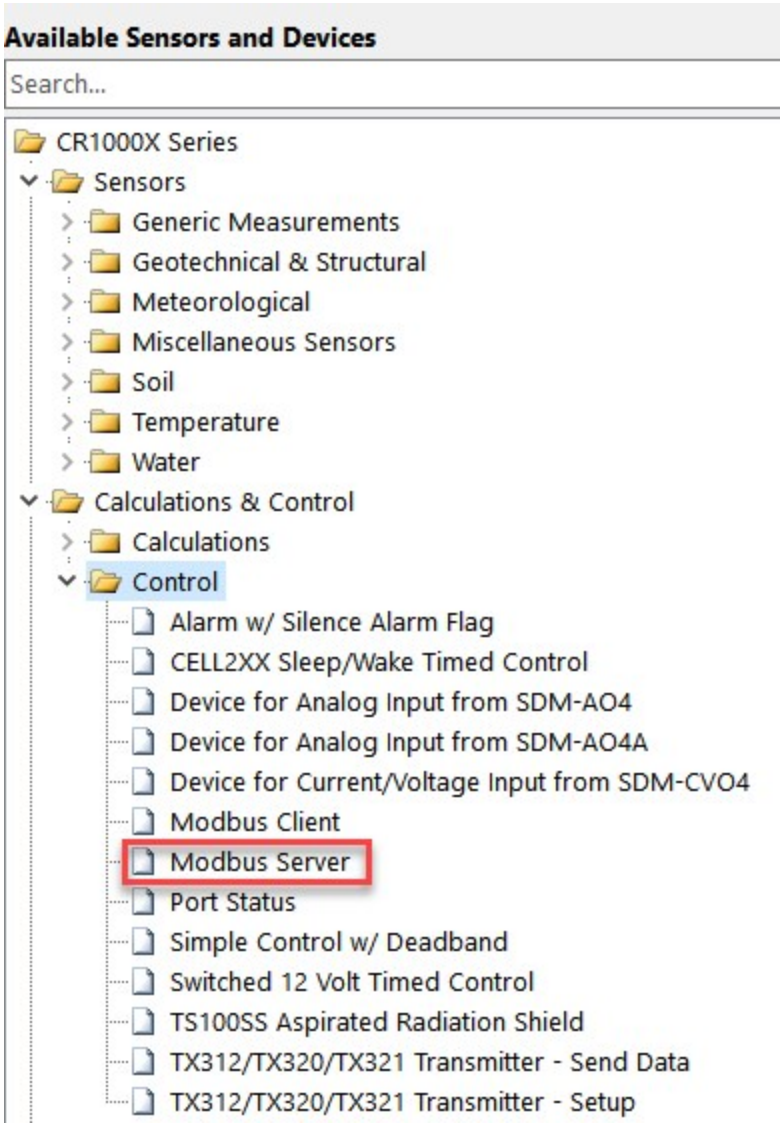
3. Click **Sensors** in the **Progress** list.



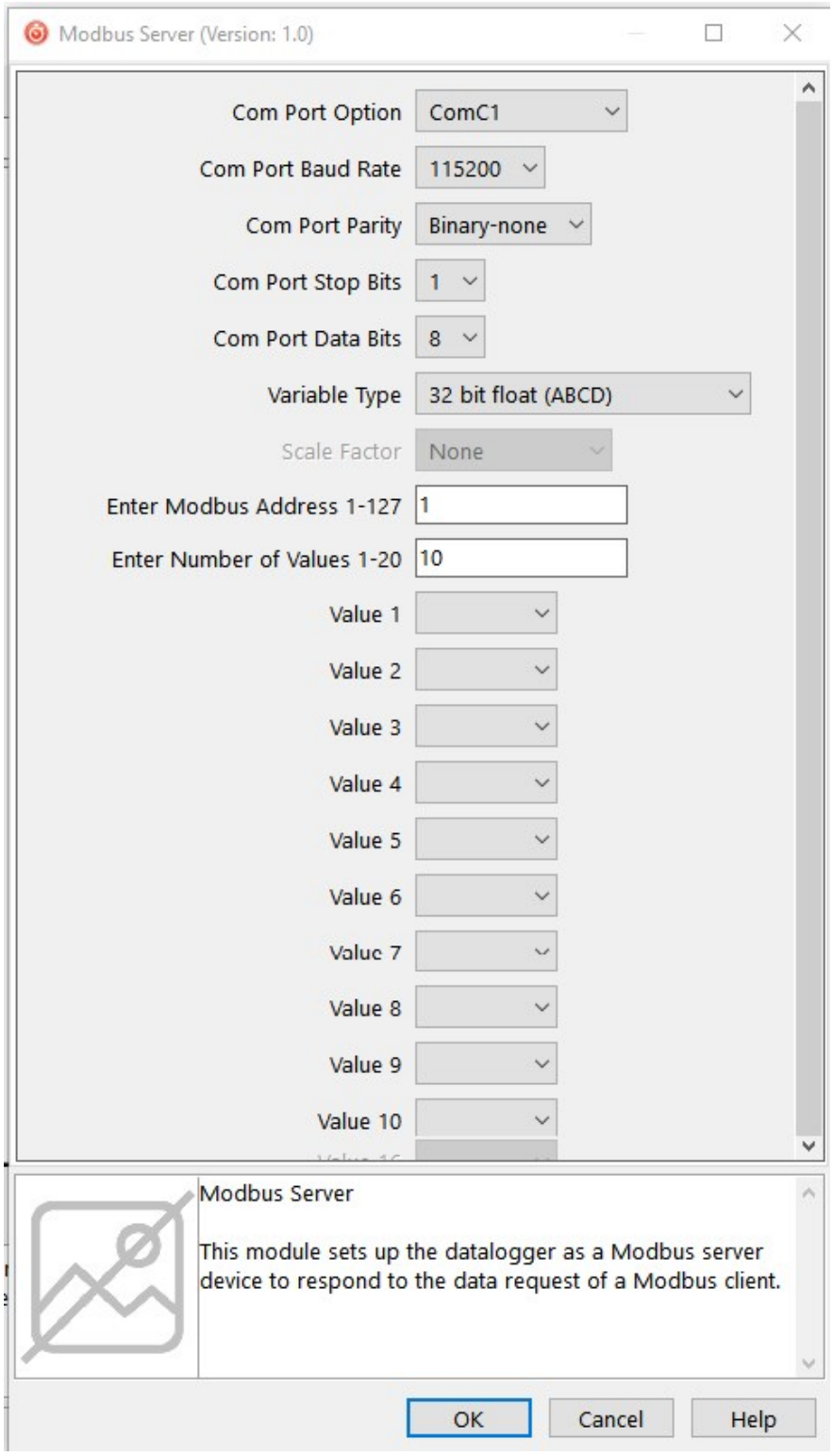
4. Add the sensors that will be measured in your program. The following sensors are used in this example.



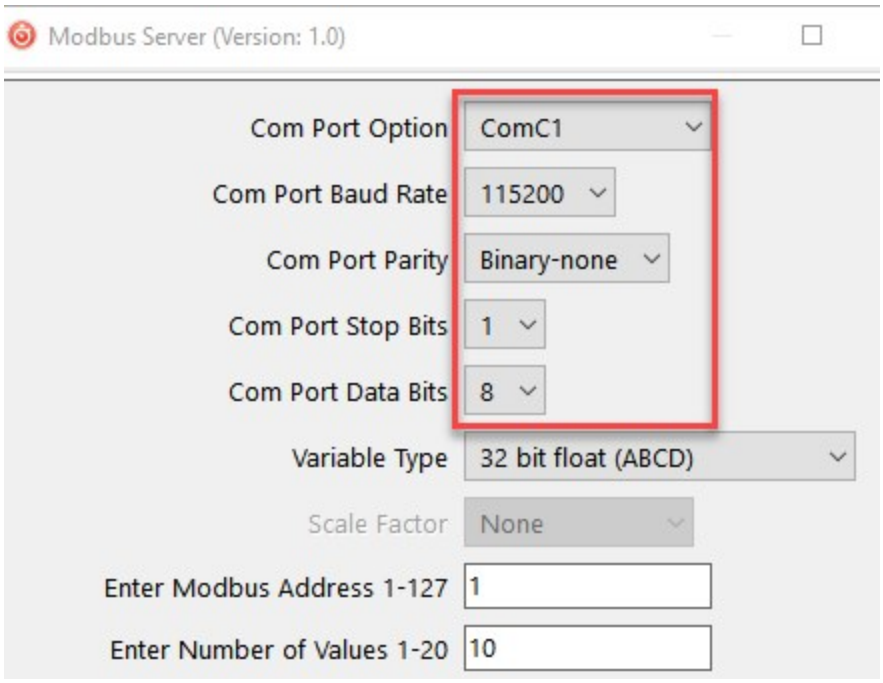
5. Modbus server functionality is added to the ***Short Cut*** program in a similar manner to adding another sensor. In the **Calculations & Control** > **Control** subfolder, double-click **Modbus Server**.



6. A dialog window is presented with several fields and options. These options are discussed in the following steps.



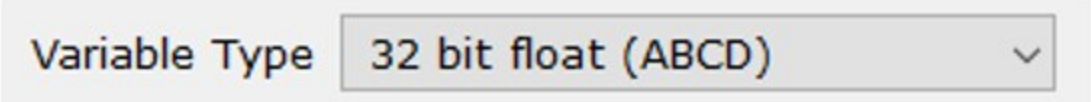
7. Select the **Com Port**, **Com Port Baud Rate**, **Parity**, **Stop Bits**, and **Data Bits** to match the Modbus client the data logger will be connected to. If **Modbus TCP/IP** is selected as the **Com Port Option**, IP port 502 will be used. Baud rate, parity, stop bits, and data bits parameters do not affect Modbus TCP/IP.



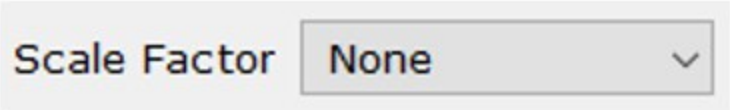
> **NOTE:**
> It is important to note that the PakBus® protocol will not work on the selected **Com Port Option** after loading the program. For example, if **ComRS232** is selected, ***Device Configuration Utility*** will no longer be able to communicate with the data logger through its **RS-232** port.

Communications can be recovered by sending a new operating system to the data logger. This process will do a full reset of the data logger. To learn how to send an operating system to a data logger, watch the tutorial video at Sending an OS to a Local Datalogger.

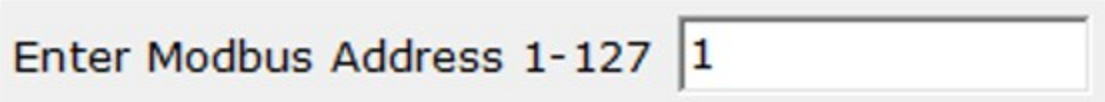
8. The **Variable Type** field selects the binary data type the data logger will use for placing data in registers. All the registers on the data logger will use the same type. Choose a **Variable Type** that is supported by the master. Set the master to the same value. **Variable Type** on other equipment and software is sometimes labeled as **Data Type**. Master devices typically allow selection of several different data types.



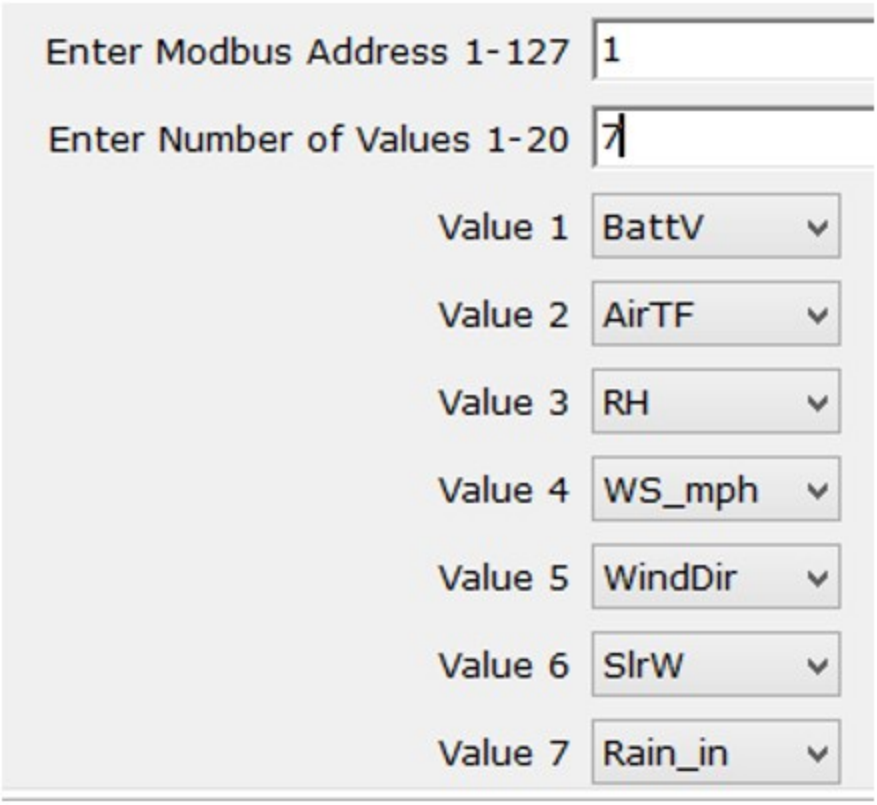
9. If applicable and necessary, enter the **Scale Factor**. Refer to Additional variable types (p. 15) for more information.



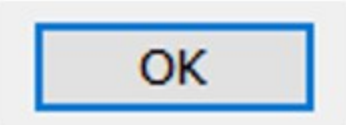
10. Enter the **Modbus Address** to be assigned to the data logger. Some manufacturers call this the **Modbus ID** or **Modbus Server ID**.



11. The **Number of Values** parameter is the number of values that will be mapped to Modbus registers. The boxes below are used to assign available variables in the data logger program to particular registers.



12. Once all parameters have been filled in within the **Modbus Server** dialog, click **OK** to add it to the data logger program. Finish the remaining steps to complete program.



Send the program to the data logger. Once the program is loaded, the data logger will be able to respond to Modbus requests from a client device.

Note that 32-bit values use two registers each. This example will have the register map shown in Table 6-1 (p. 11).

Table 6-1: Example register map

Register	Value
1,2	BattV
3,4	AirTF
5,6	RH
7,8	WS_mph
9,10	WindDir
11,12	SlrW
13,14	Rain_in

Mapped registers are available as both input and holding registers. The offset for holding registers is 40,000. Thus, to poll these 7 values with function code 04, request 14 registers starting at 40,001. Most devices do not expect the offset, so the starting register of 1 can be used.

Figure 6-2 (p. 12) shows ***Modbus Poll*** software reading holding registers from a CR1000X running the program created by ***Short Cut***. The wind speed value, **WS_mph**, of **1.478** miles/hour is seen starting at register 7.

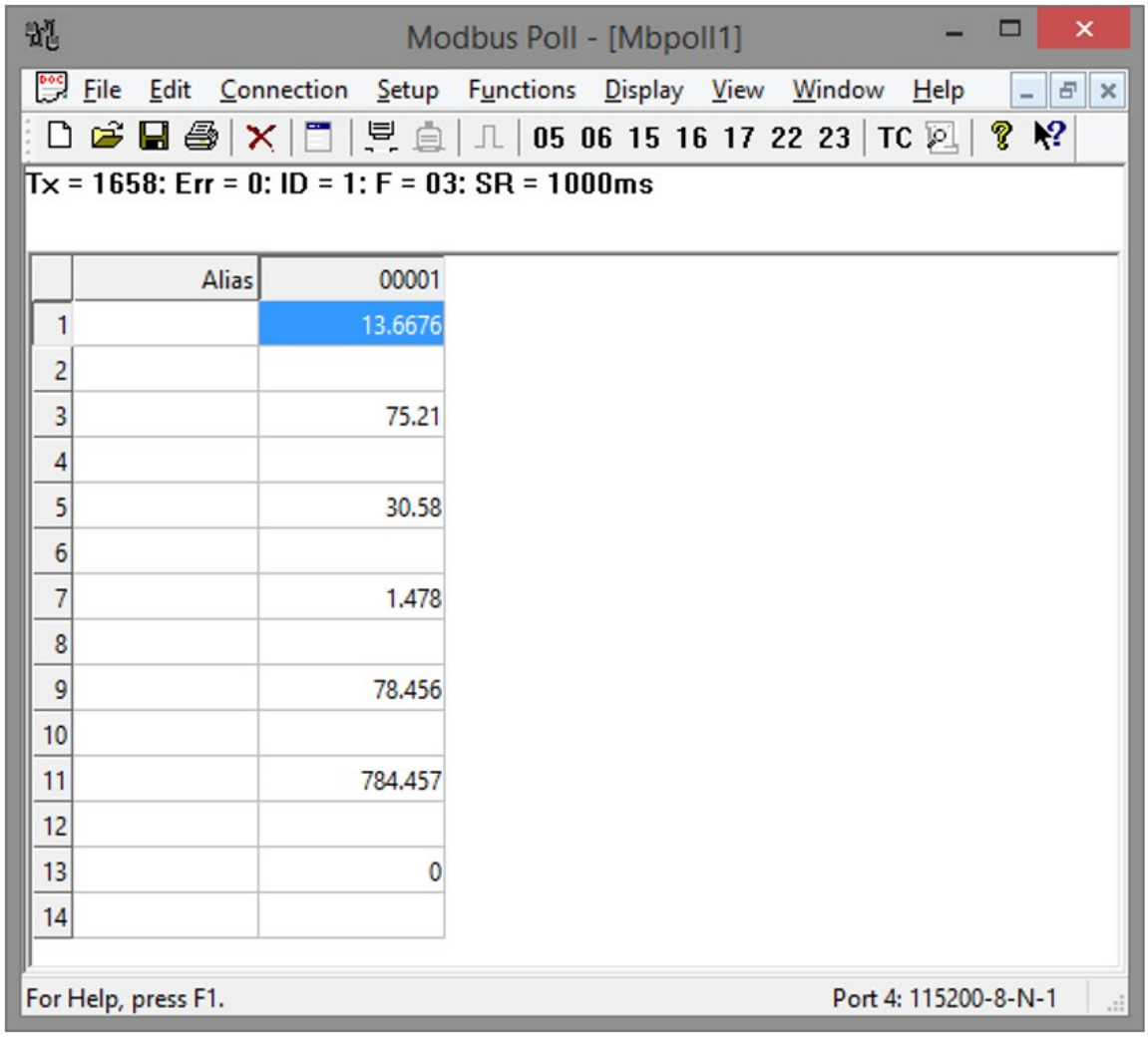


Figure 6-2. Modbus Poll software reading CR1000X series holding registers

7. Advanced topics

The Modbus functionality available through ***Short Cut*** *version 4.8* meets many common needs, but is limited. For greater flexibility, use the ***CRBasic Editor*** to customize the data logger program created by ***Short Cut***. The ***CRBasic Editor*** allows you to map more variables to registers, use additional variable types, and mix variable types within the register map.

7.1 Mapping more than 20 values

CRBasic Example 1 (p. 13) shows the entire program created by ***Short Cut*** in the previous example. It includes the necessary lines for Modbus communications, along with measurement instructions and data storage tables.

CRBasic Example 1: Modbus program generated by *Short Cut*

```
'CR1000X Series
'Created by Short Cut (4.8)

'Declare Variables and Units
Public BattV
Public PTemp_C
Public TRHData(2)
Public WS_mph
Public WindDir
Public SlrW
Public SlrMJ
Public Rain_in
Public Modbus(7)
Public ModbusCoil(8) As Boolean

Alias TRHData(1)=AirTF
Alias TRHData(2)=RH

Units BattV=Volts
Units PTemp_C=Deg C
Units WS_mph=miles/hour
Units WindDir=degrees
Units SlrW=W/m^2
Units SlrMJ=MJ/m^2
Units Rain_in=inch
Units AirTF=Deg F
Units RH=%

'Define Data Tables
DataTable(Min_BattV,True,-1)
  DataInterval(0,1440,Min,10)
  Minimum(1,BattV,FP2,False,False)
EndTable

'Main Program
BeginProg
  'Use SerialOpen to set RS232 options for ModbusServer Instruction
  SerialOpen(COMC1,115200,3,0,1000)
  'Modbus Server Instruction
```

CRBasic Example 1: Modbus program generated by *Short Cut*

```
ModbusServer(COMC1,115200,1,Modbus(),ModbusCoil(),2)

'Main Scan
Scan(5,Sec,1,0)
  'Default CR1000X Data logger Battery Voltage measurement 'BattV'
  Battery(BattV)
  'Default CR1000 Data logger Wiring Panel Temperature measurement 'PTemp_C'
  PanelTemp(PTemp_C,_60)
  'CS215 Temperature & Relative Humidity Sensor measurements 'AirTF' and 'RH'
  SDI12Recorder(TRHData(),C7,"0","M!",1,0,-1)
  AirTF=AirTF*1.8+32
  '05103 Wind Speed & Direction Sensor measurements 'WS_mph' and 'WindDir'
  PulseCount(WS_mph,1,P1,5,1,0.2192,0)
  BrHalf(WindDir,1,mV5000,1,VX1,1,2500,True,20000,60,355,0)
  If WindDir>=360 Or WindDir<0 Then WindDir=0
  'CS300 Pyranometer measurements 'SlrMJ' and 'SlrW'
  VoltSE(SlrW,1,mV1000,2,1,0,60,1,0)
  If SlrW<0 Then SlrW=0
  SlrMJ=SlrW*2.5E-05
  SlrW=SlrW*5
  'TE525/TE525WS Rain Gauge measurement 'Rain_in'
  PulseCount(Rain_in,1,P2,1,0,0.01,0)
  'Call Data Tables and Store Data
  CallTable Min_BattV
  'Copy values/measurements to Modbus Array
  Modbus(1)=BattV
  Modbus(2)=AirTF
  Modbus(3)=RH
  Modbus(4)=WS_mph
  Modbus(5)=WindDir
  Modbus(6)=SlrW
  Modbus(7)=Rain_in
NextScan
EndProg
```

Near the end of the program are several lines that copy values from measurements to values within an array. For example, this line copies the air temperature measurement into the second value of an array called Modbus:

```
Modbus(2)=AirTF
```

All the values mapped to registers must be stored in a single array. The ModbusServer() instruction references that array. To map additional values, increase the size of the array and assign more values to it.

Variables are declared at the beginning of the program. In the current example, the `Modbus()` array is defined to hold up to 7 values:

```
Public Modbus(7)
```

To support up to 30 values, simply change the declaration to:

```
Public Modbus(30)
```

Once the array is resized, you can assign additional values to it. In most cases, these assignments should be made within the scan loop, which is defined between the `Scan()` and `NextScan()` instructions.

It's best to add new assignments in the same section where *Short Cut* already copies measurement values into the array. For example:

```
Modbus(6)=SlrW
Modbus(7)=Rain_in
Modbus(28)=NextMeasurement
```

Additional measurement values are assigned using the same syntax.

In the following sections, abbreviated code examples will be used to highlight only the relevant portions of the program.

7.2 Additional variable types

The last parameter of the `ModbusServer()` instruction is an optional parameter called `ModbusOption`. This parameter specifies the variable type to apply to all registers.

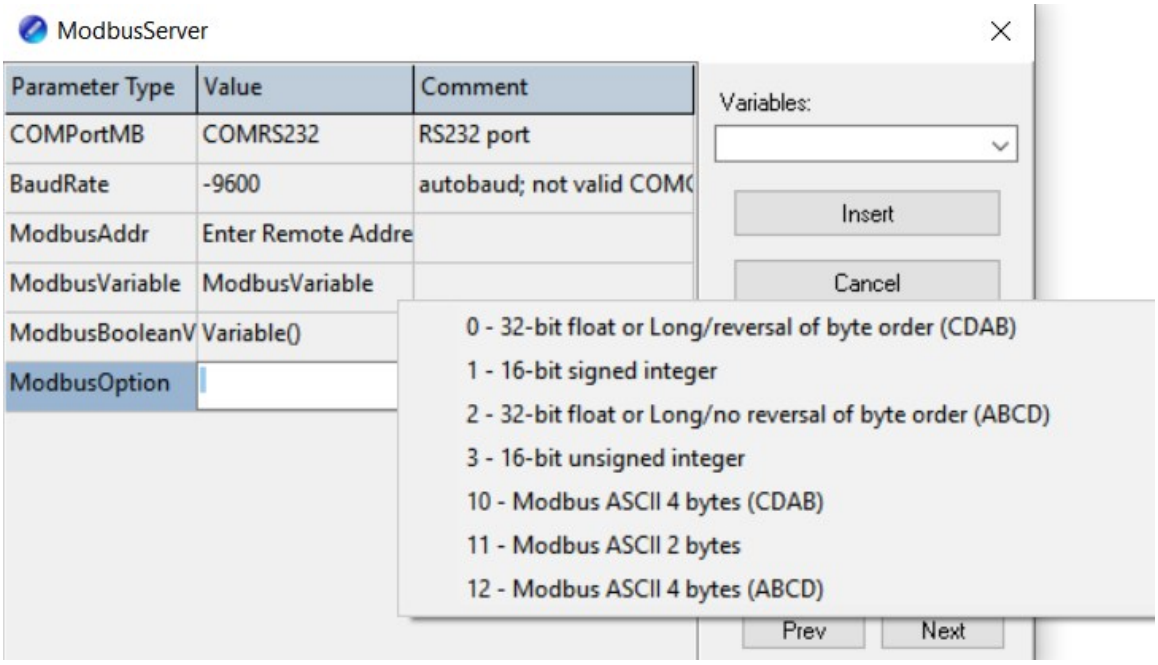


Figure 7-1. ModbusOption variable type choices

In the example ***Short Cut*** program, the **ModbusOption** is set to 2, which indicates 32-bit Float or Long/no reversal of byte order (ABCD).

```
ModbusServer(COMC1,115200,1,Modbus(),ModbusCoil(),2)
```

If the **ModbusOption** is empty or 0, then the default variable type is 32-bit Float or Long with reversal of byte order (CDAB).

To use an integer variable type, declare the Modbus array as the **Long** data type, as shown below:

```
Public Modbus(7) As Long
```

7.3 Scaling values

Integer data types truncate any digits after the decimal point. To preserve measurement resolution, you may need to scale the values.

Short Cut allows you to select integer data types and apply a single scaling factor using the **Scale Factor** field. This factor is applied to all registers.

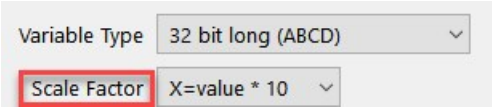


*Figure 7-2. **Scale Factor** in **Short Cut***

If you need different scaling for individual registers, use the ***CRBasic Editor*** to apply math operations directly in the assignment lines of code. For example, to maintain a resolution of 0.1 degrees for the temperature measurement in CRBasic Example 1 (p. 13), multiply the value by 10, as shown below for `AirTF`.

```
Modbus(2)=AirTF * 10
```

The Modbus register will then hold tenths of a degree. In this example, the Modbus client receives **752** instead of the original **75.21**.

Most Modbus clients can be configured to apply a scaling factor (such as 0.1) to convert the value back to degrees (e.g., **75.2**).

7.4 Coils

In Modbus terminology, coils are binary (on/off) values that represent digital outputs. Each coil is a single bit, typically used to control or reflect the state of a device like a relay, valve, or light.

The example program includes an array named `ModbusCoil()`, which must be declared as type `Boolean`.

```
Public ModbusCoil(8) As Boolean
```

This array is used for the `ModbusBooleanVar` parameter of the `ModbusServer()` instruction, which maps the Modbus coils.

ModbusServer

Parameter Type	Value	Comment
COMPortMB	ComC1	C1/C2
BaudRate	115200	fixed baud rate; valid or
ModbusAddr	1	
ModbusVariable	Modbus()	
ModbusBooleanVar	ModbusCoil()	
ModbusOption	2	32-bit float or Long/no

The example program did not assign any values to the coils. Coils can be set within the program similarly to how values are assigned to registers. In the example below, `LineState` can be declared as a `Boolean`, `Float`, or `Long`. A value of zero sets the coil state to (off), while any non-zero value sets the coil state to 1 (on):

```
ModbusCoil(2) = LineState
```

Some functions and measurement instructions directly output a Boolean value. For example, the state of a control port is returned as either true or false. In such cases, a coil can be used as the destination variable. In this example, the `PortGet()` instruction reads the state of the C1 terminal and stores it directly in `ModbusCoil(1)`:

```
PortGet (ModbusCoil(1),1)
```

7.5 Mixing variable types within a register map

It is possible to map some values to registers as integers and other as floating point. To do this, declare the variable array used in the `ModbusServer()` instruction as type `Long`. Assign Integer values directly, as shown earlier. To store floating point values in the register array, use the `MoveBytes()` instruction. This instruction performs a binary copy without altering the data format.

CRBasic Example 2 (p. 19) demonstrates this approach by placing an integer in registers 1 and 2, and a floating point number in registers 3 and 4.

CRBasic Example 2: Mixing variable types

```
Public FloatingPoint
Public Modbus(2) As Long
Public ModbusCoil(8) As Boolean

BeginProg
  'Use SerialOpen to set RS232 options for ModbusServer Instruction
  SeriaOpen(COMC1,115200,3,0,1000)
  ModbusServer(COMC1,115200,1,Modbus(),ModbusCoil(),2)

  Scan(5,Sec,1,0)
    FloatingPoint = 123.456

    'Put floating point value into register 1 and 2 as a scaled integer
    Modbus(1)=FloatingPoint * 1000
    'Put floating point value into registers 3 and 4 as a 32-bit floating point
    MoveBytes (Modbus(2),0,FloatingPoint,0,4)

  NextScan
EndProg
```

7.6 Changing byte order

Byte order (also known as endianness) refers to how multi-byte data—such as a 32-bit value— is arranged in memory.

In little-endian format, the least significant byte (the "little end") comes first. In big-endian format, the most significant byte (the "big end") comes first.

If you need ABCD byte order (big endian) in place of CDAB order, that can be changed simply with the **ModbusOption** parameter of the **ModbusServer()** instruction.

> NOTE:
> CDAB = is a mixed-endian format, often used in Modbus communications where 16-bit words are swapped within a 32-bit value.

However, converting from ABCD (big endian) to DCBA (little endian) byte order requires the use of the **MoveBytes()** instruction.

The following example, taken from the CR1000X **MoveBytes()** help, shows how to reverse the byte order of a big-endian number to a little-endian number using the **Transfer** parameter of the **MoveBytes()** instruction. The same method can also be used to reverse the byte order of a 32-bit floating point value.

CRBasic Example 3: Changing byte order

```
Public big_endian_num as Long
Public lit_endian_num as Long
Dim m

BeginProg
  Scan (1,sec,0,0)
    'Use optional Transfer parameter
    MoveBytes (lit_endian_num,m,big_endian_num,0,4,2) 'option 2 =
    'little-endian 4 byte
  NextScan
EndProg
```

7.7 Troubleshooting and additional resources

For troubleshooting tips, see the Modbus Troubleshooting Guide.

For additional information about using the Modbus protocol with Campbell Scientific data loggers, see Modbus communications in the data logger manual and these blog articles:

- Why Modbus Matters: An Introduction
- How to Access Live Measurement Data Using Modbus

Global Sales and Support Network

A worldwide network to help meet your needs

Campbell Scientific Regional Offices

Australia

Location:	Garbutt, QLD Australia
Phone:	61.7.4401.7700
Email:	info@campbellsci.com.au
Website:	www.campbellsci.com.au

Brazil

Location:	São Paulo, SP Brazil
Phone:	11.3732.3399
Email:	vendas@campbellsci.com.br
Website:	www.campbellsci.com.br

Canada

Location:	Edmonton, AB Canada
Phone:	780.454.2505
Email:	dataloggers@campbellsci.ca
Website:	www.campbellsci.ca

China

Location:	Beijing, P. R. China
Phone:	86.10.6561.0080
Email:	info@campbellsci.com.cn
Website:	www.campbellsci.com.cn

Costa Rica

Location:	San Pedro, Costa Rica
Phone:	506.2280.1564
Email:	info@campbellsci.cc
Website:	www.campbellsci.cc

France

Location:	Montrouge, France
Phone:	0033.0.1.56.45.15.20
Email:	info@campbellsci.fr
Website:	www.campbellsci.fr

Germany

Location:	Bremen, Germany
Phone:	49.0.421.460974.0
Email:	info@campbellsci.de
Website:	www.campbellsci.de

India

Location:	New Delhi, DL India
Phone:	91.11.46500481.482
Email:	info@campbellsci.in
Website:	www.campbellsci.in

Japan

Location:	Kawagishi, Toda City, Japan
Phone:	048.400.5001
Email:	jp-info@campbellsci.com
Website:	www.campbellsci.co.jp

South Africa

Location:	Stellenbosch, South Africa
Phone:	27.21.8809960
Email:	sales@campbellsci.co.za
Website:	www.campbellsci.co.za

Spain

Location:	Barcelona, Spain
Phone:	34.93.2323938
Email:	info@campbellsci.es
Website:	www.campbellsci.es

Thailand

Location:	Bangkok, Thailand
Phone:	66.2.719.3399
Email:	info@campbellsci.asia
Website:	www.campbellsci.asia

UK

Location:	Shepshed, Loughborough, UK
Phone:	44.0.1509.601141
Email:	sales@campbellsci.co.uk
Website:	www.campbellsci.co.uk

USA

Location:	Logan, UT USA
Phone:	435.227.9120
Email:	info@campbellsci.com
Website:	www.campbellsci.com