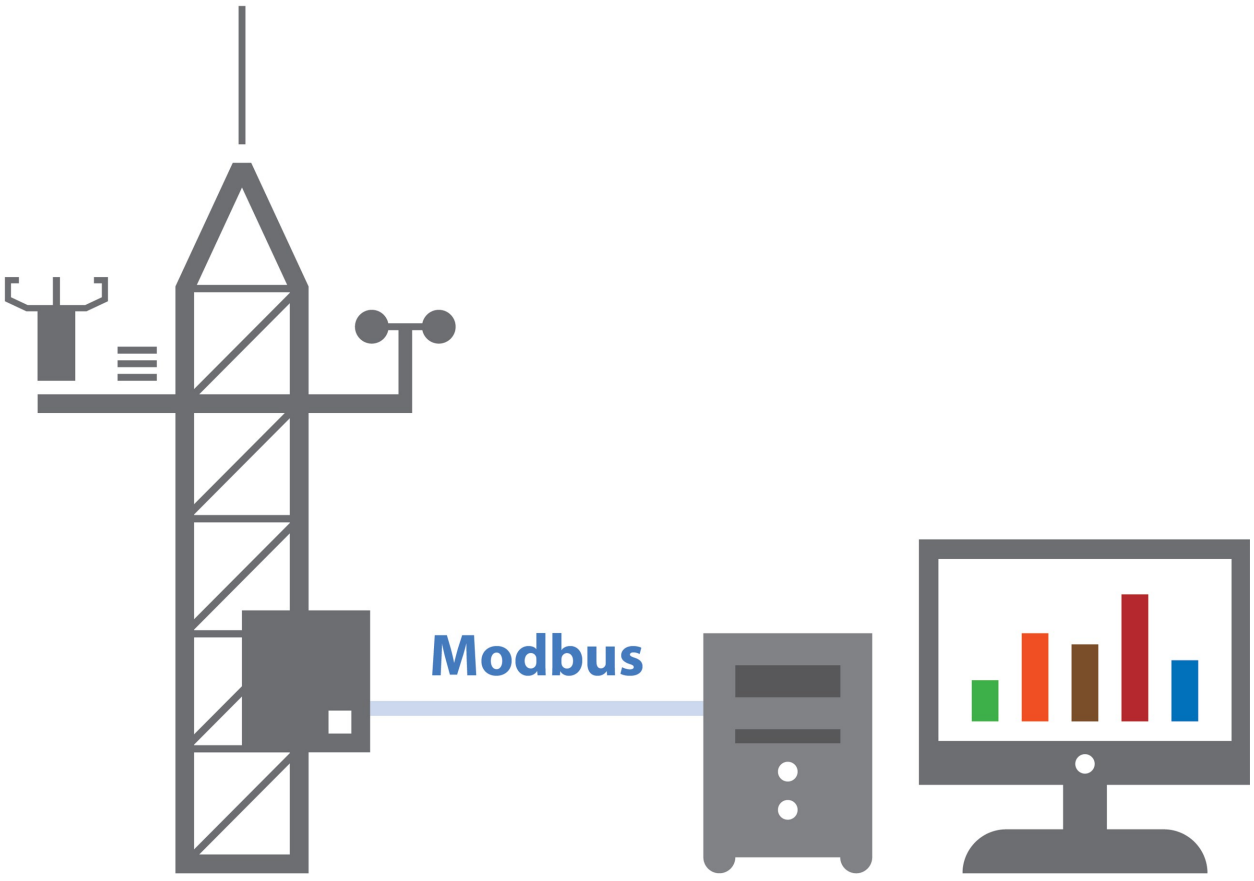
APPLICATION NOTE

Communications Device

Modbus

Troubleshooting Guide



Revision: 11/2024
Copyright © 2024
Campbell Scientific, Inc.

RELIABLE MONITORING SINCE 1974

Campbell SCIENTIFIC®

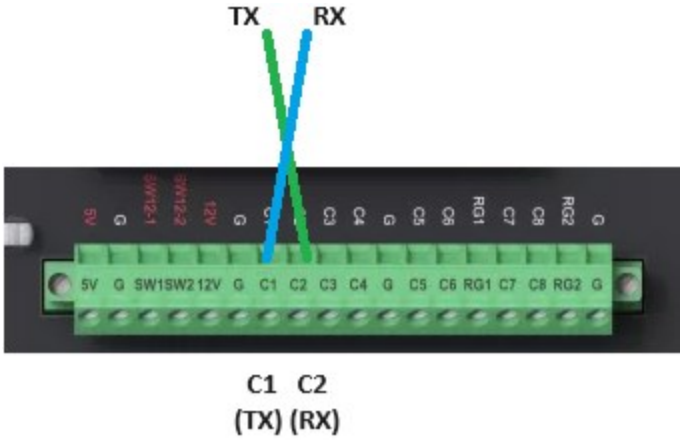
Table of contents

1. RS-232, RS-485, and Ethernet wiring

Modbus can run on multiple interfaces on Campbell Scientific devices, including RS-232, RS-485, and Ethernet. Ensure the correct cable is connected to the appropriate port on both the data logger and the Modbus device by tracing the cable and verifying the leads are securely connected to the designated ports on the data logger.

1.1 Connecting via RS-232

When connecting a Modbus device to your data logger using RS-232 over **C** or **U** ports, verify the Modbus **TX** wire is connected to the data logger **RX** port and the Modbus **RX** wire connects to the data logger **TX** port.



1.2 Connecting via RS-485

When connecting a Modbus device using RS-485 communication, ensure the data logger ports support RS-485. On a CR1000X data logger, **C5** to **C8** ports support RS-485 communication while **C1**to **C4** ports do not.

> NOTE:
> The CR1000Xe data logger supports RS-485 on C1 to C8 ports.

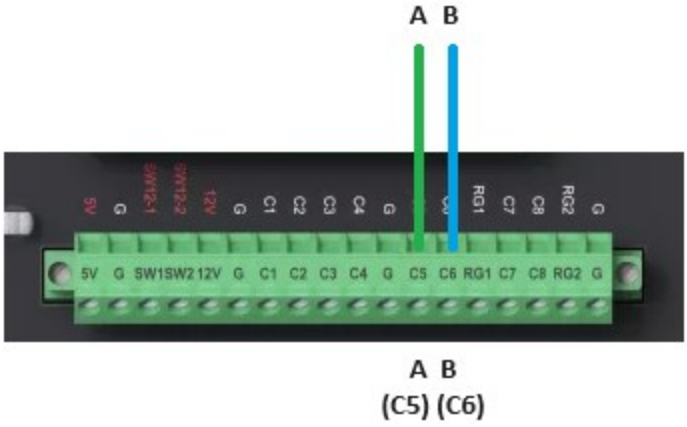
In RS-485 communications, data is transmitted using either two wires for half-duplex or four wires for full-duplex. These wires are labeled as: A (or A–) and B (or B+). In full-duplex setups, TD

A– specifically refers to the inverted terminal or wire used for transmitting data signals from a device, while TD B+ refers to the non-inverting wire used for transmit. The RD A– wire refers to the inverted receive wire, and the RD B+ refers to the non-inverted receive wire.

1.2.1 Half-duplex mode

When wiring your logger for RS-485 in half-duplex mode, ensure the A and B wires of the Modbus device connect to the designated **A** and **B** ports on the data logger.

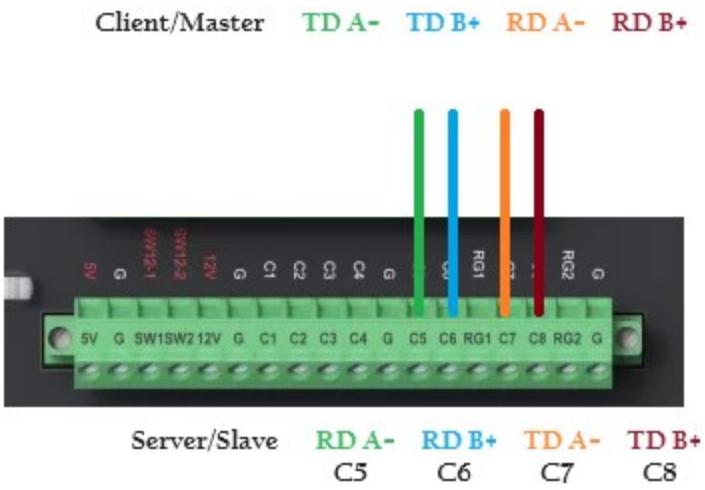
> **NOTE:**
> Some manufacturers don't adhere to the RS-485 standard and may label their A and B wires backwards. If your setup isn't operating correctly, you can try swapping the wires on the Modbus device. Often, a manufacturer labels the **A** port as negative (**–**) and the **B** port as positive (**+**).



1.2.2 Full-duplex server configuration

If wiring your data logger for RS-485 full-duplex mode in **Server** configuration, ensure the following connections:

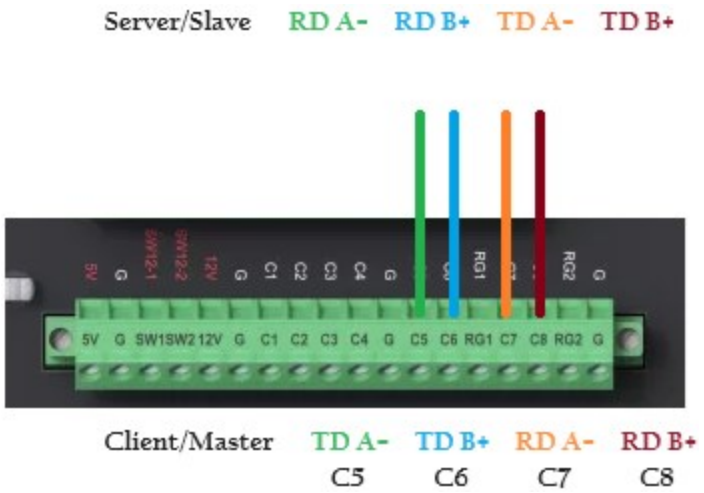
- Connect the Modbus TD A– wire to the data logger **RD A–** port.
- Connect the Modbus TD B+ wire to the data logger **RD B+** port.
- Connect the Modbus RD A– wire to the data logger **TD A–** port.
- Connect the Modbus RD B+ wire to the data logger **TD B+** port.



1.2.3 Full-duplex client configuration

If wiring your data logger for RS-485 in full duplex mode in **Client** configuration, ensure the following connections:

- Connect the Modbus RD A– wire to the **TD A–** port on the data logger.
- Connect the Modbus RD B+ wire to the **TD B+** port on the data logger.
- Connect the Modbus TD A- wire to the **RD A–** port on the data logger.
- Connect the Modbus TD B+ wire to the **RD B+** port on the data logger.



1.3 Connecting via Ethernet

When connecting your data logger directly to a Modbus device over Ethernet or to an Ethernet network, verify the Ethernet cable is connected to the **RJ45 Ethernet** port (not the **RJ45 CPI** port).

2. Power considerations for data logger and Modbus device

To verify the data logger has sufficient power, connect to it and check the battery voltage in the Status table. A healthy voltage typically exceeds 12 V; for example, a reading of 13.5 V is acceptable. However, lower readings, such as 11 V or 10.5 V, may impair or completely disrupt communications. You can check the battery voltage using the following path: ***Device Configuration Utility*** > **Data Monitor [tab]** > **Status** > **Battery**.

If voltage is low, check the 12 V power input to the data logger. Disconnect the cable from the data logger side, then check the charging regulator, solar panel, battery, and other power input to identify the source of the power issue.

See a video on **Measuring Data Logger Output Voltage With a Multimeter** here: www.campbellsci.com/videos/basic-troubleshooting-01.

3. Choosing the correct Modbus instruction: ModbusServer() or ModbusClient()

The terminology for Modbus instructions has been updated. If you're familiar with the previous terms "Modbus Slave" and "Modbus Master," here's a quick reference to the new terminology and their functions. It's crucial to use the appropriate instruction for your application.

- If the device you're communicating with is operating as a Modbus Server, you must configure your system as a Modbus Client.
- If the device is set up as a Modbus Client, your system must be configured as a Modbus Server.

> NOTE:
> The old instruction names are still supported in the ***CRBasic Editor*** and data logger operating systems, though they won't highlight blue in the ***CRBasic Editor***.

ModbusServer() = ModbusSlave()	Typically, it stores data and shares it with the Client or Master, primarily serving as a data provider.
ModbusClient() = ModbusMaster()	Generally, it requests and receives data from the Server or Slave, primarily functioning as a data recipient.

NOTE:
Your data logger can simultaneously function as both a ModbusServer(), sharing or providing data to one device, and a ModbusClient(), requesting data from another device over a separate connection.

4. ModbusServer() parameter details

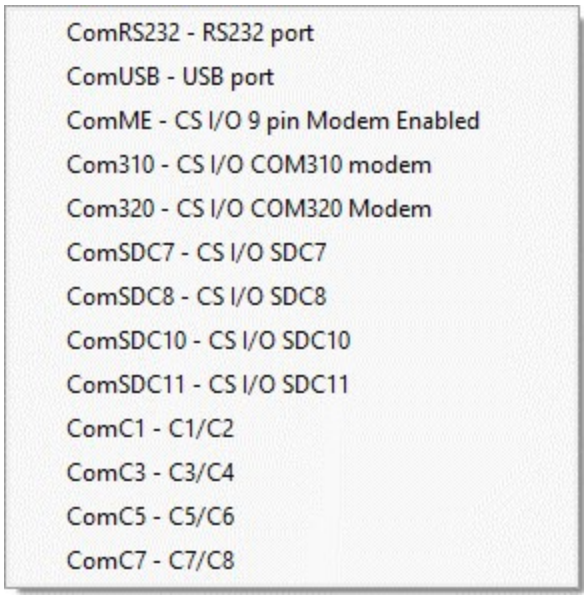
The ModbusServer() instruction enables a data logger configured as a Modbus Server to communicate with a device configured as a Modbus client. In the data logger program, the ModbusServer() instruction should be placed after the BeginProg instruction and before the Scan() instruction.

The ModbusServer() instruction has the following syntax:

```
ModbusServer (COMPortMB, BaudRate, MBAddr, ModbusVariable, ModbusBooleanVar,
ModbusOption)
```

Ensure each parameter is configured correctly:

1. ComPortMB: Ensure the COMPortMB setting matches the physical port the Modbus device is wired into. Valid settings depend on the data logger model. Examples include COMRS232, ComC1, ComC3, and ComC5. To view the full list of available options, right-click the blue-highlighted word ModbusServer in the *CRBasic Editor*.




```
ModbusServer (ComRS232,9600,1,ModIn(),0,0)
```

When using Modbus over TCP/IP, setting the **COMPort** parameter to 502 or higher designates the Modbus TCP/IP service port your data logger will monitor for incoming Modbus connections.

2. **BaudRate:** If using RS-232 or RS-485 for Modbus, check the Modbus Server baud rate matches the Modbus client baud rate. If they do not match, they will not be able to communicate with each other. This parameter is not relevant when using Modbus over a TCP/IP connection, and therefore will be ignored.

```
ModbusServer (ComRS232,9600,1,ModIn(),0,0)
```

3. **MBAddress:** Verify the Modbus address for the data logger is correct. Each device in a Modbus network must have a unique Modbus address. Record this address so other devices can be configured to connect to the data logger using the correct address.

```
ModbusServer (ComRS232,9600,1,ModIn(),0,0)
```

4. **Modbus Variable:** Ensure the Modbus variable is set and dimensioned to the correct size. The dimension size depends on the Modbus command. For instance, if the device is receiving three values, configure the **Public** variable array to three at the start of your program. For example: `Public ModIn(3)`

```
ModbusServer (ComRS232,9600,1,ModIn(),0,0)
```

The array values must be assigned within the program's Scan section, specifically between the Scan and NextScan statements. The following example demonstrates assigning three values:

```
ModIn(1) = Batt_volt
ModIn(2) = PTemp
ModIn(3) = ModResult
```

> **NOTE:**
> When values are retrieved from or sent to the Modbus Server, the data logger does not distinguish between holding and input registers. The primary difference lies in the address offset. The specified array is used for both input registers (with an address offset of 30000) and holding registers (with an address offset of 40000). Consequently, the first address for input registers is 30001, while the first address for holding registers is 40001.

> **NOTE:**
> Floating point variables take two Modbus registers.

5. **Modbus Boolean Variable:** This variable is used to hold the value of any discreet on/off commands sent to your Modbus Server. This parameter can be a variable or a variable array. If you're not using this function, set this value to 0. You may see a compile warning when using a 0; you can safely ignore the warning.

```
ModbusServer (ComRS232,9600,1,ModIn(),0,0)
```

6. **ModbusOption:** Verify the correct `ModbusOption` has been selected. Specifying a value for this parameter is optional. If you don't specify a value, `ModbusOption` will default to a 32-bit float or long with the reversed CDAB byte order.

```
ModbusServer (ComRS232,9600,1,ModIn(),0,0)
```

The options for this parameter in CRBasic are:

32-bit float or Long/reversal of byte order (CDAB)

16-bit signed integer

32-bit float or Long/no reversal of byte order (ABCD)

16-bit unsigned integer

Modbus ASCII 4 bytes (CDAB)

Modbus ASCII 2 bytes

Modbus ASCII 4 bytes (ABCD)

> NOTE:
> Record the parameters used in the `ModbusServer()` instruction for future reference when setting up a Modbus client.

5. Modbus client troubleshooting

Refer to the following when troubleshooting a data logger that is set up as a Modbus client:

5.1 Modbus client credentials checklist for third-party Modbus server device

To enable the data logger to communicate with the Modbus (Server) device, you must gather key information about the device. This information can be found in the manufacturer's documentation, the device configuration, or by consulting the person who programmed the device. Use the following checklist to ensure you have all the necessary details:

☐ Physical interface the Modbus (Server) device will use to connect to the data logger.

Interface: _____

☐ Baud rate of the Modbus (Server) device. This parameter is ignored if using an IP connection.

Baud rate: _____

☐ Modbus address of the Modbus Server your data logger will be communicating with.

Server address: _____ Client address: _____

☐ Starting register number for the Modbus data being retrieved from the Modbus Server.

Starting register: _____

☐ Determine the number of values to be received from the Modbus Server and the order in which they are received. Each value will have a specific address that you need to identify. In your CRBasic program, multiple `ModbusClient()` instructions are needed to retrieve data from non-sequential addresses.

Number of values: _____

☐ DataType of the Modbus data provided by the Modbus Server device. This includes understanding both the bit length and the byte order. The default is a 32-bit float or long with a reversed byte order of CDAB. The DataType will be one of the following:

32-bit float or Long/reversal of byte order (CDAB)
16-bit signed integer
32-bit float or Long/no reversal of byte order (ABCD)
16-bit unsigned integer
Modbus ASCII 4 bytes (CDAB)
Modbus ASCII 2 bytes
Modbus ASCII 4 bytes (ABCD)
Modbus Data Type: _____

☐ If making a connection over TCP/IP (Ethernet, Wi-Fi, or Cellular), you will need the IP address of the server.

Modbus Server IP address: _____

☐ When connecting over TCP/IP (Ethernet, Wi-Fi, or Cellular), you will need to know the port number the Modbus Server uses to provide the data. The default is 502, but you should confirm correct port number.

Modbus Server port number: _____

Once you have the necessary information, you can program your data logger using the `ModbusClient()` instruction to communicate with the Modbus Server device. The information provided in this check list serves as a guide for programming your Modbus device and is especially helpful for verifying parameters during troubleshooting. For more details on these parameters, refer to the ***CRBasic Editor*** help.

5.2 Setting up the communication port for Modbus communications

> **NOTE:**
> Skip this section if using Modbus over an IP connection.

To set up the data logger as a Modbus Server, you must configure the communication port for Modbus communication. The is typically done using the `SerialOpen()` instruction in CRBasic.

1. The **COMPort** parameter in your `SerialOpen()` instruction must match the physical port where you have wired your Modbus device. The example below uses COMRS232. Valid values include, but are not limited to: COMRS232, COMC1, COMC3, and COMC5. The COMPort options will vary depending on the data logger model you are working with. The full list of options available can be viewed by right clicking the **COMPort** parameter in the `SerialOpen()` instruction in the *CRBasic Editor*.

   ```
   SerialOpen (COMRS232,-9600,0,0,50)
   ```

2. The baud rate specified in the `SerialOpen()` instruction must match the other Modbus device baud rate. The example below uses –9600. A negative sign in front of the baud rate value indicates the interfaces will autobaud if the specified baud rate doesn't match. Autobaud is only compatible with the **RS-232** port. Right clicking the value in the *CRBasic Editor* will show you all the available baud rate options.

   ```
   SerialOpen (COMRS232,-9600,0,0,50)
   ```

3. The **SerialOpenFormat** parameter is only used for an RS-232 connection. The example below uses the default value of zero, which sets the format to **RS232 No Parity/one stop bit/8 data bits/no error checking with concurrent Pakbus over the port**. This format is used for most RS-232 connections. However, consult the manufacturer's documentation for your Modbus device to verify you are using the correct format.

   ```
   SerialOpen (COMRS232,-9600,0,0,50)
   ```

4. The **CommsMode** of the interface must match the serial protocol you intend to use. The example below uses code 0, which configures the port as RS232. Available modes include RS-232, TTL, RS-485 Half-Duplex Pakbus, RS-485 Half-Duplex transparent, and RS-485 Full-Duplex transparent. Note not all modes are applicable to every physical interface. For instance, an RS-232 interface typically cannot support RS-485 protocols. However, the C5/C6 terminal pair on a CR1000X can be configured for either RS-232 or RS-485 communications options.

```
SerialOpen (COMRS232,9600,0,0,50,0)
```

5.3 ModbusClient() parameter details

The ModbusClient() instruction should be placed in your program scan loop after the **Scan** statement and before the **NextScan** statement.

1. Declare and reference a variable to store the **ModbusResultCode**. In the example, the variable **Result** is used.. The ModbusResultCode indicates if the instruction failed and provides the reason for the failure (see Result code descriptions [p. 16]).

```
ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100)
```

2. The **COMPort** is the port the Modbus device is wired into. The following example uses RS232. Other valid ports include, but are not limited to, ComSDC7, ComC1, ComC2, ComC3, and ComC4.

```
ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100)
```

   You can also use an IP connection for communication by specifying a variable that contains an open IP socket to the remote device as the COMPort. For example, declare a variable named **TCPSocket** at the top of your program in the **Public** variables.

```
Public TCPSocket
```

   In your program, use the TCPOpen() instruction before the ModbusClient() instruction to establish a TCP/IP connection with the remote device. Specify a **TCPSocket** variable as the fifth parameter (ConnectHandle) in the TCPOpen() instruction as shown in the following example:

```
TCPOpen ("192.168.1.2",502,1,,TCPSocket)'In this example the optional 4th
parameter, IPTimeOut, is not used. Hence ,, between 1 and TCPSocket.
```

Using ConnectHandle with `ModbusClient()` is preferred because it automatically reconnects the socket if the connection is lost.

> **NOTE:**
> The first parameter in the `TCPOpen()` instruction is the IP address of the remote Modbus device. The second parameter is the port number the device is using to listen for Modbus traffic, typically port. The third parameter is a memory buffer that is not used for Modbus communications and should be set to 1 when performing Modbus communications. The fourth parameter is an optional IP time out.

The `ModbusClient()` instruction should specify the `TCPSocket`, as shown in the following example:

```
ModbusClient (Result,TCPSocket,115200,3,3,ModbusData(),1,10,3,100)
```

> **NOTE:**
> When using an IP Socket, the baud rate parameter is ignored.

If you are using Modbus over an IP connection, refer to the CR1000X example program for ModbusClient() over TCP/IP (Modbus TCP) (p. 30) for more information.

3. The baud rate must match the rate used on the other Modbus device. In the example, the baud rate is 115200 bps. If you are using Modbus over a TCP/IP connection, this parameter is ignored.

```
ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100)
```

4. The Modbus address must match the Modbus address of the other Modbus (Server) device. If unsure of the address of the other device, check the documentation, configuration, or programming of the Modbus device.

```
ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100)
```

5. For the `ModBusFunction`, the example uses Option 3, which reads the holding Registers from the Modbus Server. Option 3 is the most common option. Descriptions of all the options are provided in Table 5-1 (p. 13).

```
ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100)
```

Table 5-1: Modbus function options		
Option	Name	Description
1	Read Coil/Port Status	Reads the On/Off status of discrete output(s) in the ModbusServer
2	Read Input Status	Reads the On/Off status of discrete input(s) in the ModbusServer
3	Read Holding Registers	Reads the binary contents of holding register(s) in the ModbusServer
4	Read Input Registers	Reads the binary contents of input register(s) in the ModbusServer
5	Force Single Coil/Port	Forces a single Coil/Port in the ModbusServer to either On or Off
6	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

6. If your Modbus option is set to read data, the **ModbusVariable** stores the data received from the Modbus Server. If you are writing to the Modbus Server, this variable is the source of the data being sent.

   Declare the variable at the top of your CRBasic program along with the other **Public** variable declarations. The following is an example of the **Public** Modbus variable declaration, assuming 20 floating point Modbus values are being read from the ModbusServer() or held to be written to the Modbus Server:

   ```
   Public ModbusData(20)
   ```

   The type of variable you declare as the **Public** variable will affect its function. Be sure to declare the variable as the correct type based on the **ModbusFunction** and **ModbusOption** you are using. Refer to the **ModbusFunction**, which is the last parameter in the ModbusClient() instruction shown in the following example. If the values you are reading or writing are declared as four-byte floating-point (Float) data

types, and a register function code of 3, 4, or 16 is used, the data points will be mapped to a Modbus Holding or Input register as floating-point data.

This parameter can also be declared as Boolean data type, which is used to represent conditions or hardware that have only two states (true or false), such as flags and control ports. A register function code (coil) of 1, 2, 5, or 15 will be mapped to a Modbus coil in the other device.

```
Public ModbusData(2) As Boolean
```

If this parameter is declared as a Long data type (used for integer variables), and a register function code of 3, 4, or 16 is used, it will be mapped to Modbus Holding or Input registers as an integer. Longs are treated as signed integers, meaning the values will range from –32,768 to +32,767.

```
Public ModbusData(2) As Long
```

This is the parameter as it appears in the `ModbusClient()` instruction in your data logger program.

```
ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100)
```

7. Ensure you have specified the correct Modbus start register. This is the 16-bit address of the first register that will be requested (or acted upon). The start parameter is entered as a number in the range of 1 to 65535. The start parameter in the example is 1. The data address, or offset, in the Modbus frame sent to the server will be equal to "Start-1". For example, if you are reading the first starting register of a Modbus device, you should specify 1 for this parameter. For a Modbus function code 3, the manufacturer documentation might indicate the starting address is 40001, but you should specify 1 in your data logger. Setting the value to 40001 will tell your data logger to start reading 40,001 records into the holding register instead of the first one. This is a common mistake when using the `ModbusClient()` instruction.

   > NOTE:
   > Campbell Scientific data loggers use "Start-1" addressing to determine the address being used. Some manufacturers might start their addressing at 0 or 1, which can create confusion. If you specify the starting address, but do not receive correct values, it's a good practice to attempt reading a fixed value from the Modbus registers. Then, adjust the address incrementally until you receive the correct data from the other Modbus

device.

```
ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100)
```

8. Verify the **ModbusLength** parameter is correct. This is the number of CRBasic variables to act on with this instruction. When reading coils or using a 16-bit **ModbusOption** (Options 1, 3, 11, or 13), the **ModbusLength** will match the number of 16-bit register values to be requested or processed. When using a 32-bit or 4-byte **ModbusOption** (Options 0, 2, 10, or 12), the data logger will automatically double the **ModbusLength** value to request 32-bit data for each CRBasic variable. In summary, specify the number of values needed, and the data logger will handle the rest automatically, as long as the correct **ModbusOption** parameter is selected. See item 11 to verify the appropriate **ModbusOption** parameter.

```
ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100)
```

9. Verify the **ModbusTries** parameter. This value is the number of times the data logger will attempt to communicate with the Modbus Server before moving on to the next instruction in the program.

```
ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100)
```

10. The **ModbusTimeOut** parameter specifies the amount of time the data logger will wait for a response before considering the attempt a failure. This value is indicated in units of .01 seconds. Therefore, a value of 100 equals one second. If you are using Modbus over an IP connection with some latency, you may want to increase this value to a few seconds. For local serial connections, one second is typically sufficient.

```
ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100)
```

11. Verify the correct **ModbusOption** has been selected. If you don't apply a value for this parameter, it will default to 32-bit float or Long/reversal of byte order (CDAB). This parameter needs to match the other Modbus device your data logger is communicating with.

```
ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100,0)
```

Check the manufacturer manual for the other Modbus device or consult the person who programmed it to verify the correct format.Manufacturers may not always provide clear instructions on the format they use. Some may only specify the bit size, such as 16-bit or 32-bit. Others might mention "reverse byte order," "reverse endian," or simply "CDAB." If

the format is not listed, you may need to contact the person who programmed the system to determine the byte order used. Alternatively, you can try different formats until it works. For more information on big-endian and little-endian formats in communications, see Serial formatting (p. 34).

The options for this parameter in CRBasic are:

32-bit float or Long/reversal of byte order (CDAB)

16-bit signed integer

32-bit float or Long/no reversal of byte order (ABCD)

16-bit unsigned integer

Modbus ASCII 4 bytes (CDAB)

Modbus ASCII 2 bytes

Modbus ASCII 4 bytes (ABCD)

5.4 Result code descriptions

Table 5-2 (p. 16) describes the result codes provided by the `ModbusClient()` instruction. When Modbus communications fail, the result code often provides an indication of what is wrong with your Modbus communications. A similar table is available in the ***CRBasic Editor*** help.

Table 5-2: Result code descriptions

Code	Descriptions
–1	Illegal function. The function code received in the query is not an allowable action for the device. The device may not support the function, or it may not be in a state to process the request. Check your `ModbusFunction` in the Modbus client troubleshooting (p. 8)
–2	Illegal data address. The data address received in the query is not an allowable address for the device. The combination of the reference number and transfer length may be invalid. Check the manufacturer documentation of your Modbus Server and match the setting in the Modbus client troubleshooting (p. 8) section, item 7.
–3	Illegal data value. The value contained in the query data field is not an allowable value for the device.
–4	Server device failure. An unrecoverable error occurred while the device was attempting to perform the requested action.

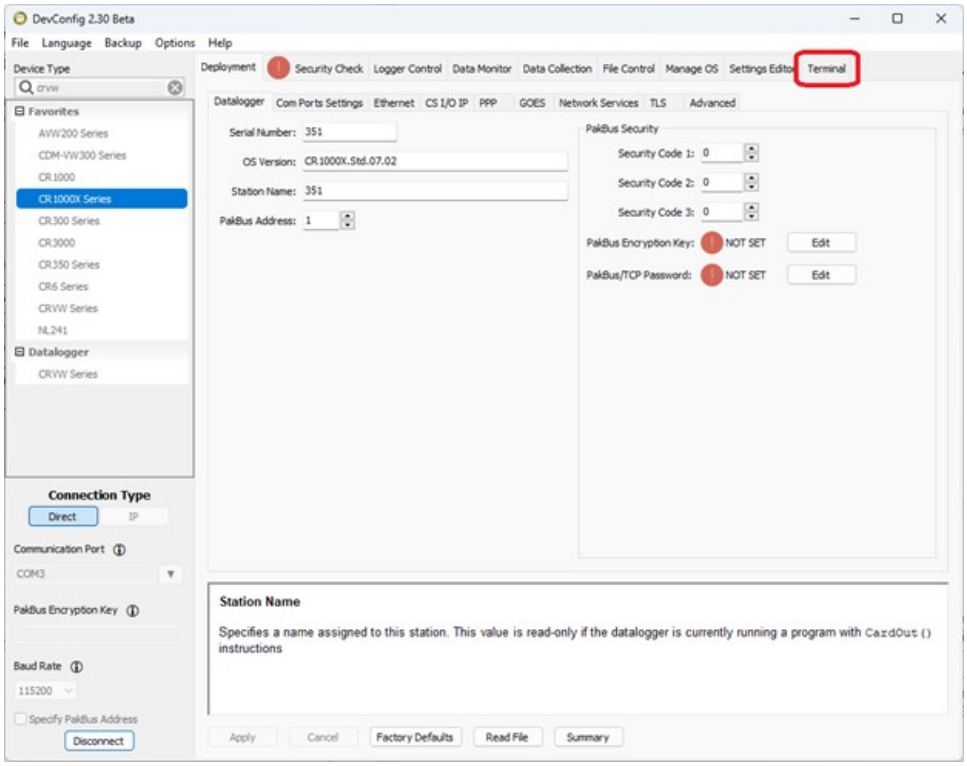
Table 5-2: Result code descriptions	
Code	Descriptions
–5	Acknowledge. The device has accepted the request and is processing it, but a long duration of time will be required to do so. This is a specialized function used in conjunction with programming commands.
–6	Server device busy. The device is engaged in processing a long-duration program command.
–8	Memory parity error. The device attempted to read a record file but detected a parity error in the memory. Used in conjunction with function codes 20 and 21.
–9	Gateway path unavailable. Indicates the gateway was unable to allocate an internal communications path from the input port to the output port for processing the request.
–10	Modbus Client error. Received an unexpected function code response from server.
–11	The specified ComPort (or TCP socket) is not opened. There is no connection to the Modbus Server. Verify the correct parameter is specified in the Modbus client troubleshooting (p. 8) section, item 2.
–16	ModbusClient error. Out of comms memory.
–20	ModbusClient error. Variable not dimensioned large enough to store results from server.
1..2..n	Communications with the Modbus Server has failed. This can be due to incorrect serial communications settings or incompatible data types. See Setting up the communication port for Modbus communications (p. 10) and Modbus client troubleshooting (p. 8).

6. Watch Modbus RTU traffic from the data logger terminal mode

You can verify your data logger and the other Modbus device are exchanging Modbus traffic. This procedure uses the data logger terminal mode to watch the outgoing and incoming

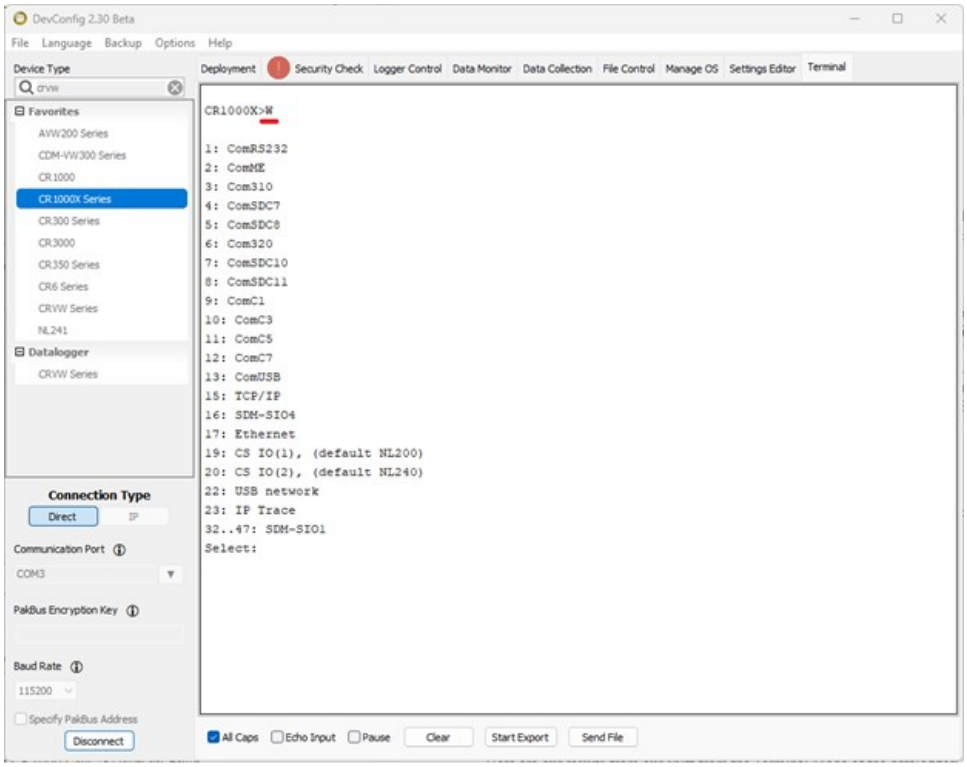
Modbus traffic from a Campbell Scientific data logger. Although the example shows a Modbus Client, the steps are the same for a Modbus Server.

1. Open ***Device Configuration Utility*** and connect to your data logger.
2. Click the **Terminal** tab.



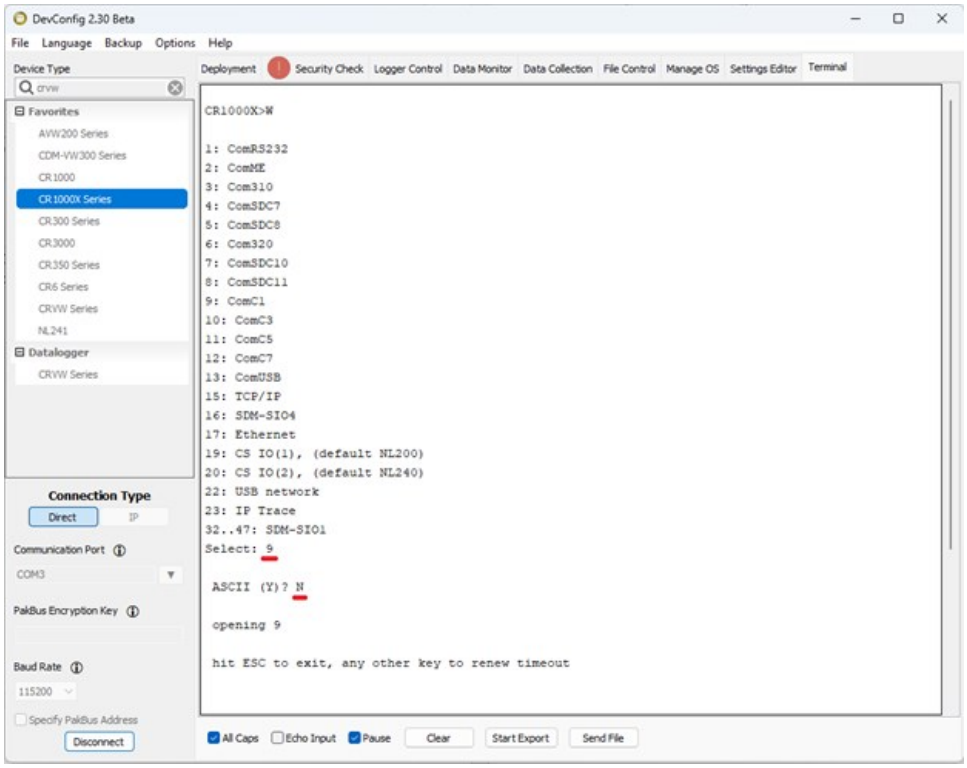
3. Press **Enter** until you see the data logger you are using. In this example, the CR1000X was used.

4. Type the capital letter **W** and press **Enter**.



5. Enter the number that corresponds to the COM port you are using and press **Enter**. If you are using IP for Modbus communications, go to Watch Modbus TCP traffic from the data logger terminal mode (p. 22).

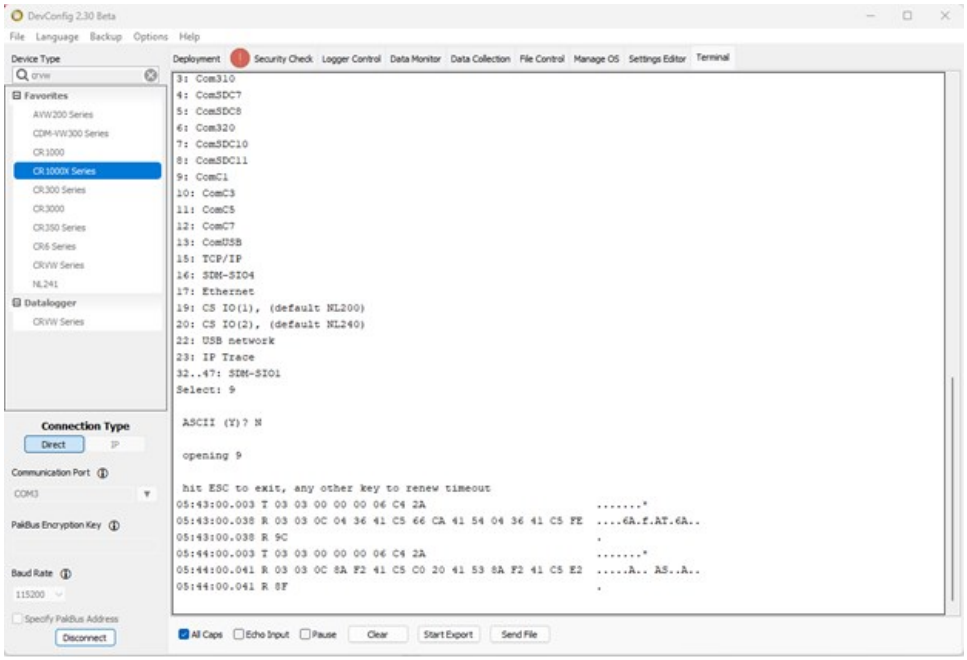
6. You will be prompted for **ASCII (Y)?**; answer with a capital letter **N** and press **Enter**.



7. You should now see a message that says **opening** followed by the port you specified. After that you will see the **hit ESC to exit, any other key to renew timeout** message. Now wait for your Modbus instruction to trigger or manually trigger it in your program. If the results begin to fill up the screen quickly, clicking the **Pause** check box at the bottom of the ***Device Configuration Utility*** window will pause the display, allowing you to take your time reading what's already been displayed.

   To export the results for later analysis, click the **Start Export** button. In the **Choose an export file** window, select a location to save the results of the sniff. Enter a file name (e.g., *myModbussniff.txt*) in the **File name** field at the bottom of the window, and then click **Save**.

Here is a screenshot of a sniff from the **Terminal** mode of the data logger:



This sniff shows the data logger we are monitoring (a Modbus Client) has sent out a transmission, as indicated by the **T** in the results.

```
 hit ESC to exit, any other key to renew timeout
05:43:00.003 T 03 03 00 00 00 06 C4 2A                          .......*
05:43:00.038 R 03 03 0C 04 36 41 C5 66 CA 41 54 04 36 41 C5 FE  ....6A.f.AT.6A..
05:43:00.038 R 9C                                               .
05:44:00.003 T 03 03 00 00 00 06 C4 2A                          .......*
05:44:00.041 R 03 03 0C 8A F2 41 C5 C0 20 41 53 8A F2 41 C5 E2  .....A.. AS..A..
05:44:00.041 R 8F                                               .
```

In the next screenshot, a serial device has responded to the transmission request sent by the data logger running Modbus. This response is indicated by the **R** or received serial traffic. In this case, it represents the Modbus Server response to the Modbus Client request.

```
 hit ESC to exit, any other key to renew timeout
05:43:00.003 T 03 03 00 00 00 06 C4 2A                          .......*
05:43:00.038 R 03 03 0C 04 36 41 C5 66 CA 41 54 04 36 41 C5 FE  ....6A.f.AT.6A..
05:43:00.038 R 9C                                               .
05:44:00.003 T 03 03 00 00 00 06 C4 2A                          .......*
05:44:00.041 R 03 03 0C 8A F2 41 C5 C0 20 41 53 8A F2 41 C5 E2  .....A.. AS..A..
05:44:00.041 R 8F                                               .
```

Now that we have confirmed that traffic is being sent and received from our data logger operating as a Modbus Client, and we are receiving responses, how can we verify this traffic is Modbus traffic?

The easiest way to recognize Modbus serial traffic (Modbus RTU) is to look for transmissions that start with the Modbus Server address and function code. Similarly, the server response will also begin with its address and function code. Since we are monitoring the Modbus Client, notice that immediately after the initial **T**, the Modbus address of the server (03 or "3") is followed by the function code (03 or "3"). This confirms the traffic we are observing is Modbus traffic.

```
 hit ESC to exit, any other key to renew timeout
05:43:00.003 T 03 03 00 00 00 06 C4 2A                          .......*
05:43:00.038 R 03 03 0C 04 36 41 C5 66 CA 41 54 04 36 41 C5 FE  .... 6A.f.AT.6A..
05:43:00.038 R 9C                                               .
05:44:00.003 T 03 03 00 00 00 06 C4 2A                          .......*
05:44:00.041 R 03 03 0C 8A F2 41 C5 C0 20 41 53 8A F2 41 C5 E2  .....A.. AS..A..
05:44:00.041 R 8F
```

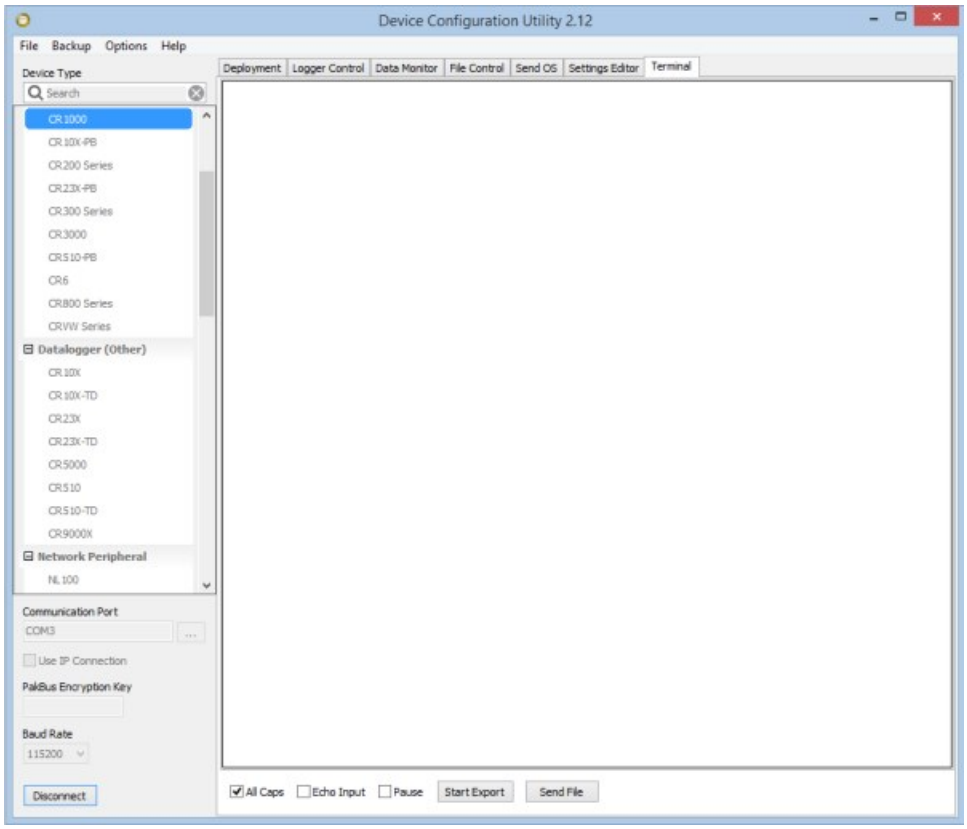
With Modbus Client, if you are not seeing responses from the Modbus Server, verify your configuration by following the first seven steps of this document and ensure the Modbus Server is configured correctly. This process can also be applied to an IP connection.

7. Watch Modbus TCP traffic from the data logger terminal mode

This example uses a CR1000 data logger operating as a Modbus Server. The process is the same for a Modbus Client operating on an IP connection.

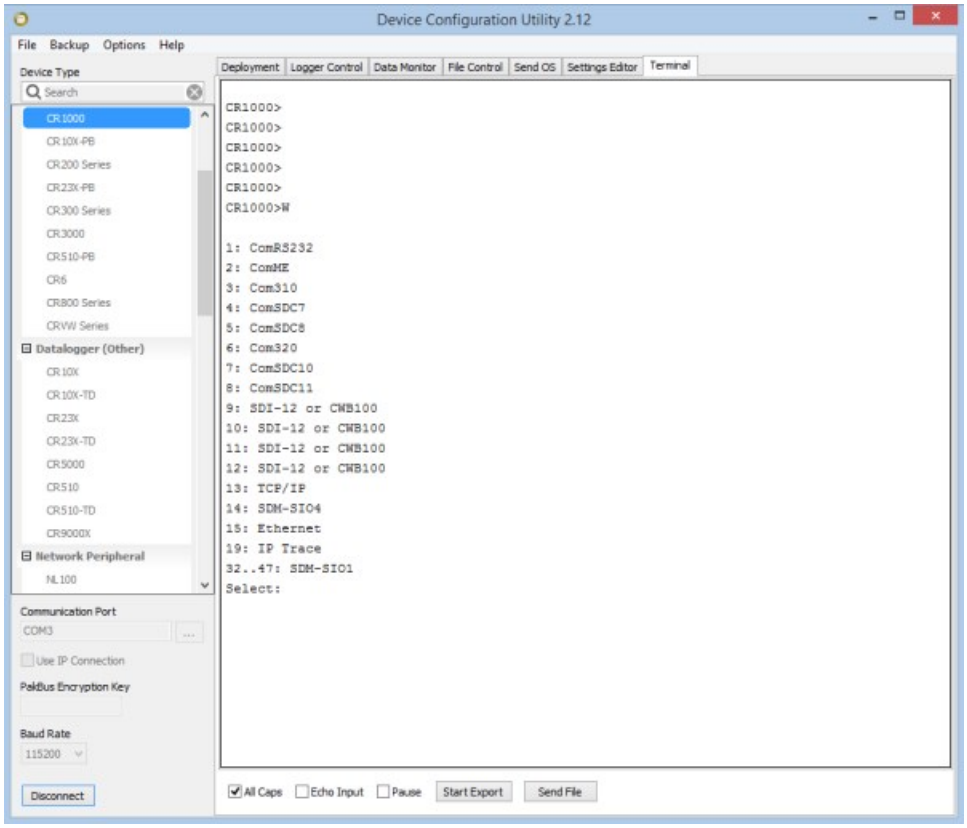
Follow these steps to use ***Device Configuration Utility*** to see the Modbus polls over your IP connection:

1. Open ***Device Configuration Utility*** and connect to your data logger.
2. Click the **Terminal** tab.



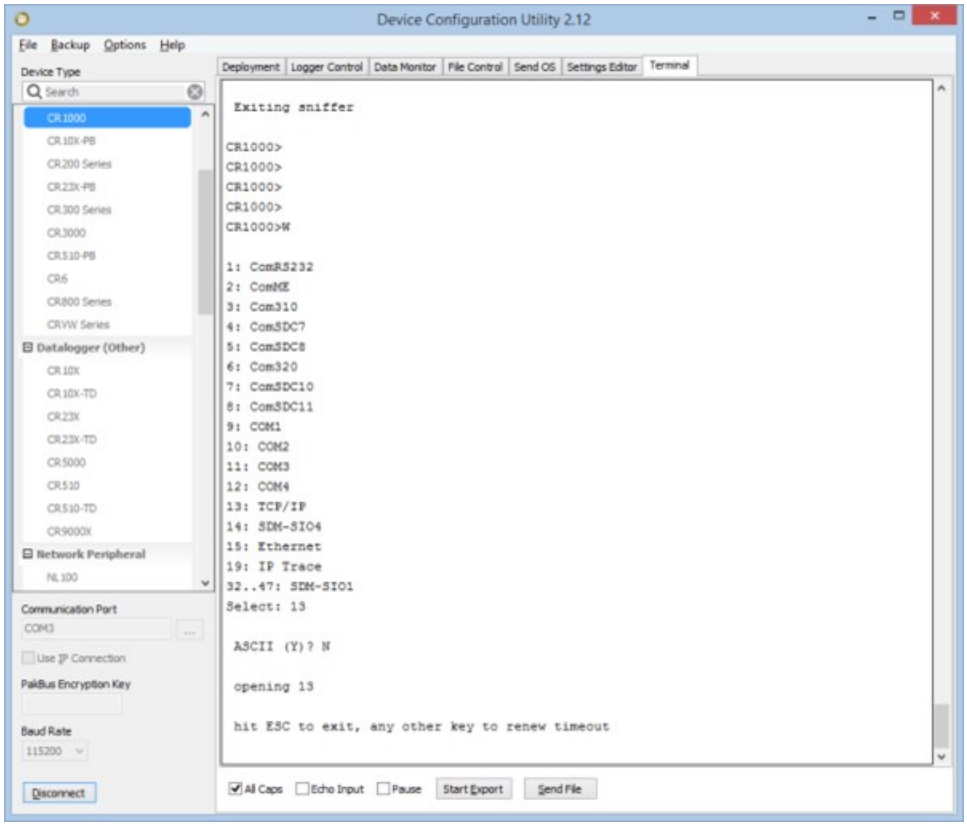
3. Press **Enter** on your keyboard until you see a prompt on your screen.

4. Type the capital letter **W** and press **Enter**.



5. Select **TCP/IP** and press the **Enter** key.
6. You will be prompted for `ASCII (Y)?`; answer with a capital letter **N** and press **Enter**.

7. If your data logger is not receiving any Modbus polls, your screen will likely look something like this:

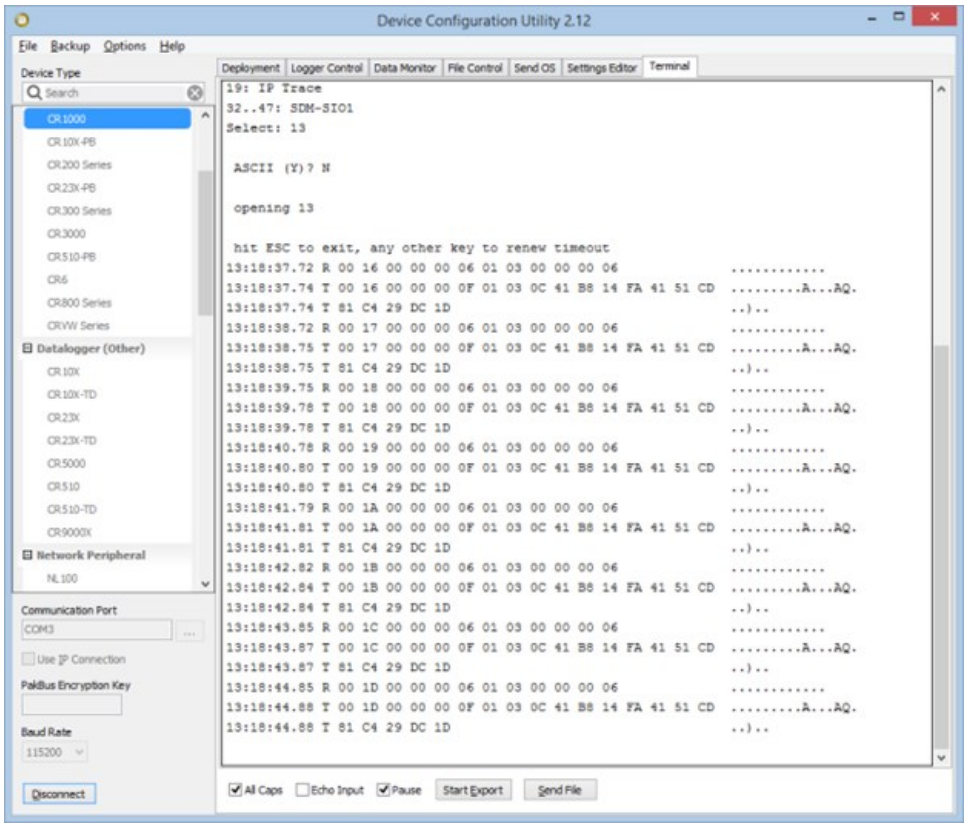


If no Modbus traffic is detected over TCP/IP and the only message on the screen is `hit ESC to exit, any other key to renew timeout`, this situation could indicate one or more of the following conditions:

- The SCADA system or other Modbus device (Modbus Client) is not polling the data logger (Modbus Server), or the data logger (Modbus Server) is not responding to the Modbus Client's requests.
- The SCADA system or other Modbus device is polling a different IP address.
- The data logger has been assigned an incorrect IP address.
- A cable is not plugged in.
- Modbus traffic is being blocked by the network.

Even if your data logger is set up and programmed correctly, it will not transmit data as expected if it is not receiving polling requests from the SCADA client. In this case, direct your troubleshooting efforts toward the SCADA network, client configuration, and other factors external to the data logger.You might observe TCP/IP traffic that is not related to Modbus. For

example, you may see PakBus traffic from a ***LoggerNet*** Server if there is a ***LoggerNet***-to-data-logger connection on the network. Once network or SCADA system problems are resolved, a successful trace will appear as follows:



The easiest way to identify Modbus TCP traffic and differentiate it from other protocols is by noting that transmissions from the client always begin with an identifier in the first two bytes. For example, in the trace, the first recognized poll started with **00 16**, and the data logger responded with this same unique identifier (**00 16**). On the next poll, the client used an identifier of **00 17**, and the data logger responded with **00 17**.

> **NOTE:**
> There is a difference between Modbus TCP traffic and Modbus RTU traffic. The easiest way to recognize Modbus RTU traffic is to look for a transmission from the client that starts with the Modbus Server address and function code. The server response will also start with its address and function code.

Modbus polls marked as (T) originate from the data logger you are running the terminal connection on, while polls marked as (R) come from the other device. If your data logger is acting as a Modbus Server and you are not seeing communication from it, there may be an error in your setup, such as:

- The data logger is not programmed as a Modbus server.
- The data logger has been given a Modbus server ID that does not match what the client is polling.

At this point, you'll need to examine your data logger setup for further troubleshooting.

> NOTE:
> This section is derived from a blog post written by Paul Smart and is available on the Campbell Scientific website here: www.campbellsci.com/blog/diagnose-modbus-communication.

8. Troubleshooting IP connectivity for Modbus TCP communications

Work with your IT department to confirm that your Modbus devices, including your data logger, are configured correctly and communicating on the network. The following is a list of common troubleshooting steps to help determine network connectivity issues.

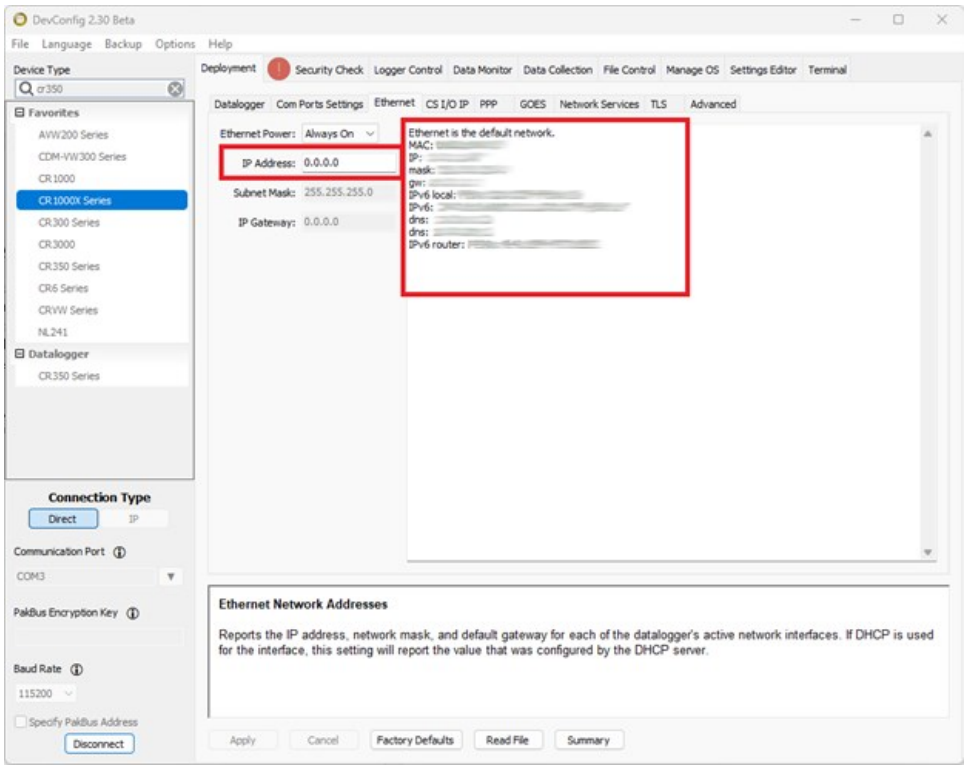
1. Verify the data logger IP address, subnet Mask, and gateway address are correct.
2. If the data logger is polling a Modbus Server, verify the Modbus Server address matches the Modbus address specified in your data logger program.
3. Confirm the IP address, subnet mask, and gateway address settings on other Modbus devices.
4. Examine the Ethernet cable to ensure it is plugged in, and confirm that Wi-Fi connections on both devices are properly established, if applicable.
5. Look at the Ethernet LEDs near the data logger Ethernet port. A green light and a flashing amber light indicate traffic is passing between the device and the network.

6. If the data logger is directly connected to the other Modbus device using Ethernet, set the Ethernet **Power** setting to **Always on** to keep the interface from going to sleep. This setting is found here: ***Device Configuration Utility*** > **Deployment** > **Ethernet** > **Ethernet Power**.
7. To check if the data logger and the other Modbus device are communicating, connect a computer to the same network subnet and try to ping the IP address of both devices from the **Command Prompt** of your computer.

   > **NOTE:**
   > For security, the data logger ping function is turned off by default. This setting is found here: ***Device Configuration Utility*** > **Deployment** > **Network Services** > **Ping (ICMP) Enabled**.

8. If the data logger is connected to the network, set its IP address in the **Ethernet** tab (or relevant IP interface) to **0.0.0.0** and apply the change. Setting 0.0.0.0 as the IP address enables DHCP on the data logger. If the network is assigning addresses using DHCP, the data logger will receive an IP address, which will appear in the status area of the **Ethernet** tab (or the corresponding IP interface) in the ***Device Configuration Utility***. When the data logger receives an address from the network, compare it with your existing address, subnet mask, and gateway address. The subnet mask and gateway address should match exactly, and the IP address should be very similar.

   > **NOTE:**
   > If the data logger receives an IP address that starts with 169.254, this indicates the network is not assigning addresses using DHCP. This may also mean the data logger and computer are directly connected to each other.



Example results follow:

Expected results:

IP Address: 172.91.23.45

Subnet Mask: 255.255.0.0

Gateway Address: 172.91.25.100

Results from **Status** area of **Ethernet** tab:

IP: 192.168.1.2

Mask: 255.255.255.0

GW: 192.168.1.1

The addresses 192.168.1.2 and 172.91.23.45 are from different networks and cannot communicate with each other. If your data logger receives an address like this, contact your IT department to request a new IP address and inform them the network configuration is incorrect.

Appendix A. CR1000X example program for ModbusClient() over TCP/IP (Modbus TCP)

This program reads in two floating point values from the Modbus Server at IP Address *192.168.1.1* every 60 seconds and writes them to the ModbusData floating point variable. The ModbusClient() instruction is placed within a SlowSequenceto ensure that measurement instructions continue to operate in the scan without being slowed down.

```
Public Temp, batt_volt, ModResult
Public ModbusData(2)
Public TCPSocket

'Main Program
BeginProg
  Scan (2,Sec,0,0)
    Battery(batt_volt)  'Simple diagnostic instructions to record
    PanelTemp(Temp)

  NextScan
  SlowSequence
  Scan (60,Sec,3,0)
    If TCPSocket = 0 Then
      TCPOpen ("192.168.1.1",502,1,,TCPSocket)
      'Check every time if the socket is still open. If not, reopen the socket and
      'assign the handle to a variable called TCPSocket.
    EndIf
    'Utilize the handle from the TCPOpen instruction as the COM port and read in
    'values from the Modbus server. These values are stored in the ModbusData
    'variable.
    ModbusClient (ModResult,TCPSocket,-9600,3,3,ModbusData(),1,2,3,200,0)
  NextScan
EndProg
```

Appendix B. CR1000X example programs for ModbusServer()

Following are two examples of the ModbusServer() instruction.

Example 1, FP and Coils not declared

```
Public PTemp, batt_volt, ModResult
Public ModIn(3)

'Main Program
BeginProg
ModbusServer (ComRS232,9600,1,ModIn(),0)
  Scan (2,Sec,0,0)
    Battery(batt_volt) 'Diagnostic instruction to record data logger battery
                       'voltage
    PanelTemp(PTemp,1500) 'Diagnostic instruction to record data logger panel
                          'temperature

    ModIn(1) = batt_volt
    ModIn(2) = PTemp
    ModIn(3) = ModResult

  NextScan
EndProg
```

Example 2, Integer, and coils defined as Boolean array (Coils map to C1 to C8): :

```
Public PTemp, batt_volt, Counter
Public Port(8) as Boolean
Public ModIn(10) as Long 'Long provides integer conversion
'Main Program
BeginProg
ModbusServer (ComRS232,9600,1,ModIn(),Port())
  Scan (2,Sec,0,0)
    Battery(batt_volt) 'Diagnostic instruction to record data logger battery
                       'voltage
    PanelTemp(PTemp,1500) 'Diagnostic instruction to record data logger panel
                          'temperature
    Counter=Counter+1

    ModIn(1) = batt_volt
    ModIn(2) = PTemp
    ModIn(3) = Counter

    PortSet(C1,Port(1))
    PortSet(C2,Port(2))
    PortSet(C3,Port(3))
    PortSet(C4,Port(4))
    PortSet(C5,Port(5))
    PortSet(C6,Port(6))
    PortSet(C7,Port(7))
    PortSet(C8,Port(8))

  NextScan
EndProg
```

Appendix C. CR1000X example program for ModbusClient() over serial (Modbus RTU)

This program uses SerialOpen to specify serial configurations and ModbusClient() to query the Modbus Server over RS-232.

```
'Declare Public Variables
Public PTemp, batt_volt,ModbusData(20),Result

'Define Data Tables
DataTable (Test,1,-1)
  DataInterval (0,15,Sec,10)
  Minimum (1,batt_volt,FP2,0,False)
  Sample (1,PTemp,FP2)
  Sample (20,ModbusData(),IEEE4)
EndTable

'Main Program
BeginProg

  SerialOpen (COMRS232,115200,1,0,50)

  Scan (1,Sec,0,0)
    PanelTemp (PTemp,15000)
    Battery (Batt_volt)

    'Retrieve Modbus Data
    ModbusClient (Result,COMRS232,115200,3,3,ModbusData(),1,10,3,100)

    'Enter other measurement instructions
    'Call Output Tables
    CallTable Test
  NextScan
EndProg
```

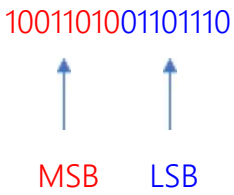
Appendix D. Serial formatting

Communication ports and Modbus configurations can vary between manufacturers. This section outlines the key settings to check when reading data from a third-party sensor using the `ModbusClient()` instruction.

1. **Big-endian vs. little-endian**

   In terms of byte order, little-endian places the most significant byte on the right side, while big-endian places the most significant byte on the left side. For example, the decimal number 39,534, when written in big-endian binary, has the byte that most significantly impacts the decimal value placed on the left side.

   Big-endian

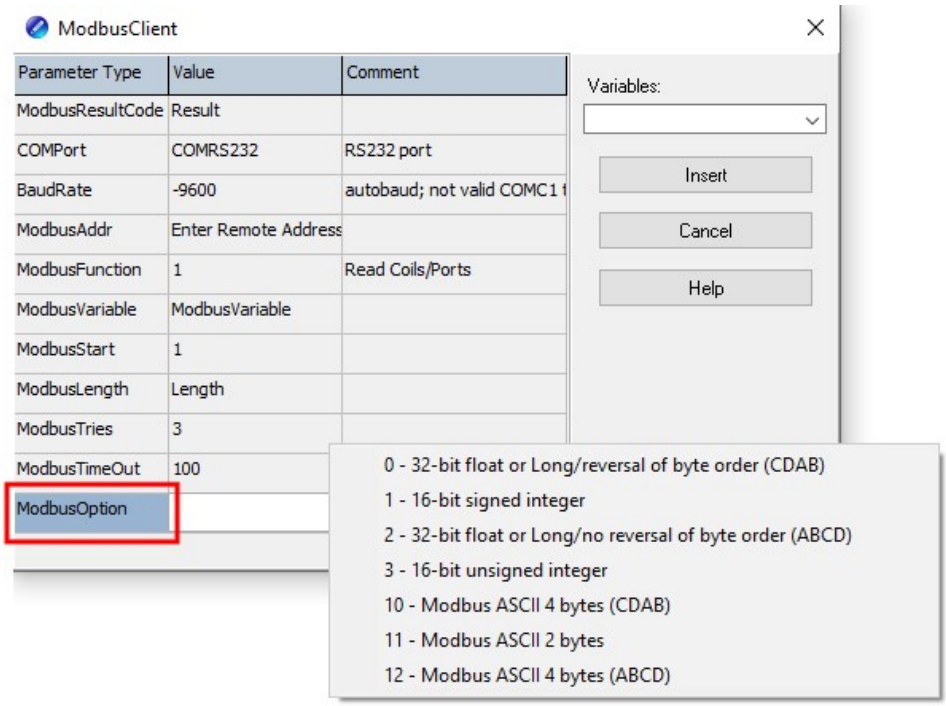
   

   Little-endian flips this order with the leftmost byte being the least significant and the rightmost byte being the most significant.

   Little-endian

   1001101001101110

   Modbus follows the same concept of endianness when transmitting data. This can be seen in the `ModbusOption` parameter within the `ModbusClient()` instruction.



Modbus register data is transmitted in hexadecimal, with each hex digit representing a 4-bit pattern. Therefore, two hex digits equal 1 byte. In little-endian format, this is represented as CDAB, while in big-endian format, it is represented as ABCD. The default for most Modbus servers is little-endian format.

2. Logic, parity, data bits, and stop bits

   Modbus has several layers of communication parameters that must be configured correctly. At the serial communication level, these include settings for logic, parity, data bits, and stop bits.

   a. Logic

      The logic level determines whether a binary 0 is represented as a low or high voltage. This is determined with either a pull-up or pull-down resistor of some kind. This resistor ensures, when there is no signal, a floating value is either connected to a low or high voltage source at all points in time to reduce errors. In simple terms, logic 1 low means that a 0 is formed from a low voltage level and logic 1 high means that 0 is formed from a high voltage level. True RS-232 typically uses logic 1 low; TTL typically uses logic 1 high.

b. **Parity**

Parity is a form of error checking for frames that come in. Most devices do not use parity; however, some utilize either odd or even parity. If parity is used, both the transmitter and receiver count the amount of binary 1s in the frame. Odd parity will contain an odd number of 1s in the frame. Even parity will contain an even number of 1s in the frame. An additional bit will be added to ensure the “even” or “odd” parameter is met.

Example:

Parity	Original byte	Modified byte
Odd	00101101 (four 1s)	001011011 (five 1s)
Even	00101101 (four 1s)	00101101 (four 1s)

c. **Data bits**

Data bits determine how many bits there are in a singular frame without parity and without the start and stop bits. Usually, this value is 8.

d. **Stop bit**

Stop bits signal the end of a data frame in serial communication. They help differentiate one packet from the next by marking the frame's start with a "low" logic level (start bit) and its end with a "high" logic level (stop bit). The change in logic levels indicates the beginning and end of a frame. Typically, 1 stop bit is used in serial communications.

3. **Register types**

Register types are determined by their address within the register map. This is denoted as the Modicon notation.

Coil = 00001 to 09999

Discrete input = 10001 to 19999

Input register = 30001 to 39999

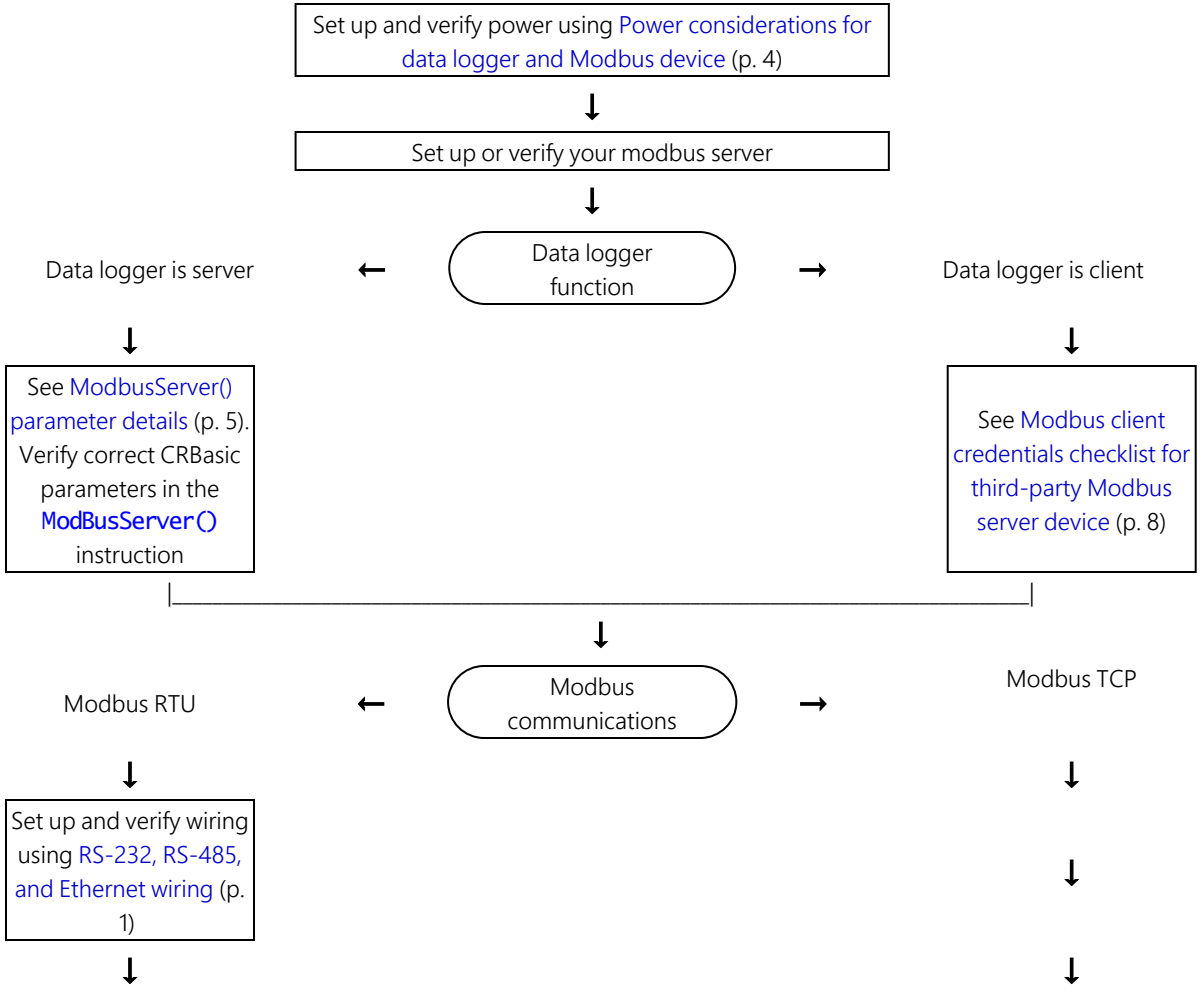
Holding register = 40001 to 49999

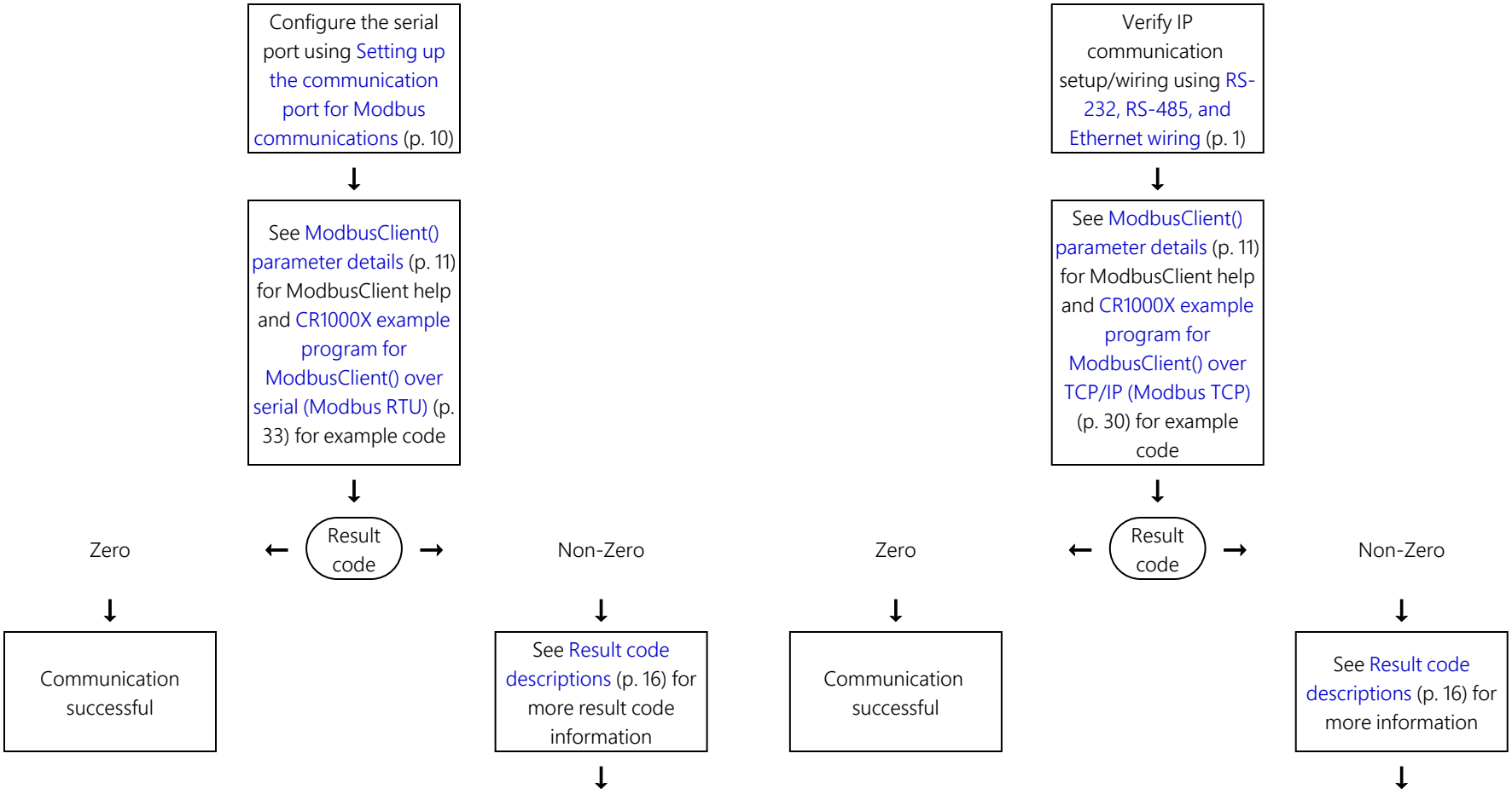
Holding registers are the most common if you are looking to read values from the server.

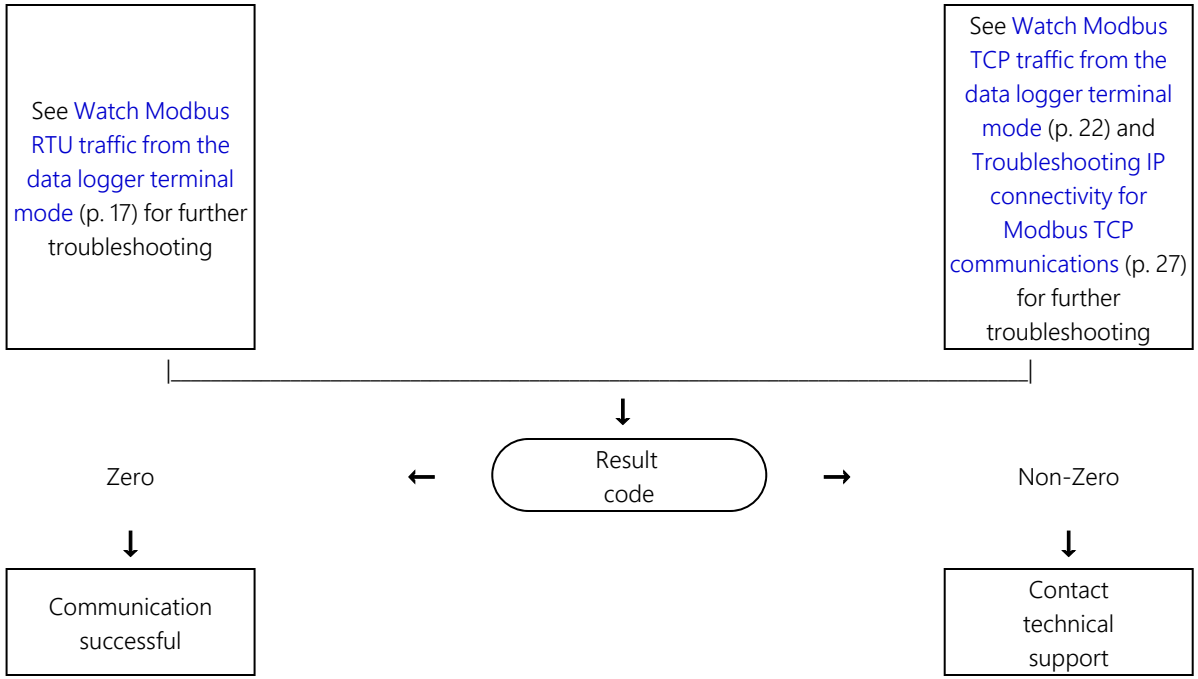
4. **Modbus address of the Modbus device**

   Modbus addresses can range from 0 to 65,535 and should be listed on your Modbus device documentation. The address must be entered in the fourth parameter of the `ModbusClient()` instruction.

Appendix E. Modbus setup flowchart




Configure the serial port using Setting up the communication port for Modbus communications (p. 10)
See ModbusClient() parameter details (p. 11) for ModbusClient help and CR1000X example program for ModbusClient() over serial (Modbus RTU) (p. 33) for example code
Result code
Zero
Communication successful
Non-Zero
See Result code descriptions (p. 16) for more result code information
Verify IP communication setup/wiring using RS-232, RS-485, and Ethernet wiring (p. 1)
See ModbusClient() parameter details (p. 11) for ModbusClient help and CR1000X example program for ModbusClient() over TCP/IP (Modbus TCP) (p. 30) for example code
Result code
Zero
Communication successful
Non-Zero
See Result code descriptions (p. 16) for more information


See Watch Modbus RTU traffic from the data logger terminal mode (p. 17) for further troubleshooting
See Watch Modbus TCP traffic from the data logger terminal mode (p. 22) and Troubleshooting IP connectivity for Modbus TCP communications (p. 27) for further troubleshooting
Result code
Zero
Non-Zero
Communication successful
Contact technical support

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