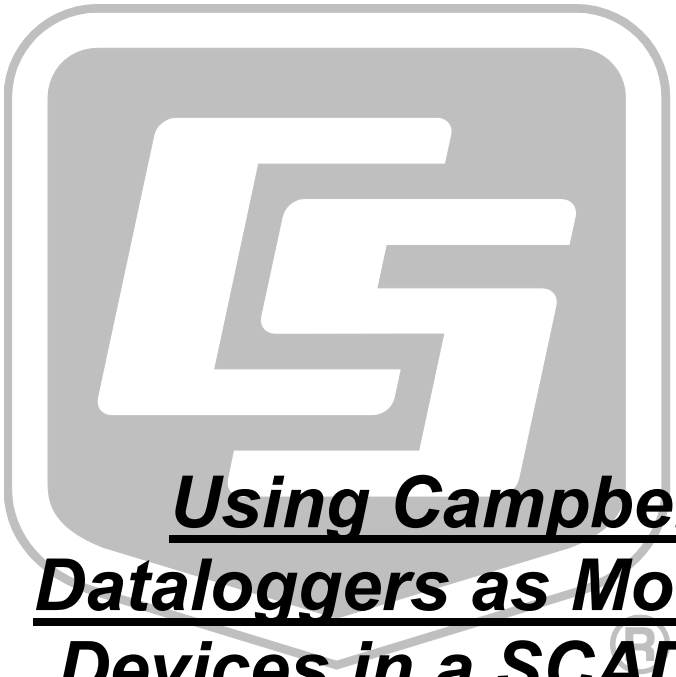
App. Note Code: 1D-X

APPLICATION NOTE



Using Campbell Scientific Dataloggers as Modbus Slave Devices in a SCADA Network

6/16

Table of Contents

PDF viewers: These page numbers refer to the printed version of this document. Use the PDF reader bookmarks tab for links to specific sections.

Figures

Table

CRBasic Examples

Using Campbell Scientific Dataloggers as Modbus Slave Devices in a SCADA Network

1. Introduction

Most Campbell Scientific dataloggers are capable of acting as Modbus slave devices. On serial connections, they will use the standard Modbus RTU protocol. IP enabled dataloggers will use the Modbus TCP protocol on IP connections. In general, any of the available communication ports could be configured for Modbus communication. Refer to the specification sheet for the individual datalogger model to view its capabilities. Due to the flexible nature of the hardware, the Modbus capability of the datalogger must be enabled through configuration or programming. The aim of this document is to help you quickly enable the Modbus functionality and configure it to be compatible with your SCADA system.

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

At a minimum, *Short Cut* and *Device Configuration Utility* are needed to configure a Campbell Scientific datalogger for use as a Modbus slave. Both may be downloaded or updated at *www.campbellsci.com/downloads*. For more complex applications, the *CRBasic Editor* is needed for programming. The *CRBasic Editor* is part of *LoggerNet* and *PC400*.

3. Physical Connections

Common physical connections are RS-232, TTL-level RS-232, RS-485, and Ethernet. Additionally, wireless options such as Wi-Fi are available. Consideration should be given to proper surge protection of any cabled connection. Between systems of significantly different ground potential, optical isolation may be appropriate. Third party optical isolators are available for RS-232, RS-485, and Ethernet.

Default parameters of serial connections are 115200 baud, no parity, and 1 stop bit. Other values may be used by changing the datalogger configuration.

The datalogger will need a stable DC power supply. For maximum reliability, it is recommended to use a battery-backed power supply such as the PS150. Refer to the datalogger specifications for power supply requirements.

Dataloggers are most commonly mounted on a 1-inch grid backplate inside an enclosure purchased from Campbell Scientific. Contact Campbell Scientific if you need a scaled footprint drawing for mounting in a third party enclosure.

4. Supported Function Codes

Supported Modbus functions are **01**, **02**, **03**, **04**, **05**, **15**, and **16**. For an explanation of these function codes, refer to the Modbus protocol standard.

5. Register Mapping

With their multipurpose nature, Campbell Scientific dataloggers do not have a fixed Modbus register map. The mapping of values to registers is done by the user with the datalogger program. The register mapping process is simple to do with *Short Cut*. By default, all variables are 32-bit floating point numbers in CDAB byte order. Other data types and byte orders are possible.

6. Comprehensive Example

This comprehensive example shows how to configure a CR1000-based weather station to be a Modbus slave device. This information is also directly applicable to several other models of dataloggers.

6.1 Entering IP Settings

If using an IP connection for communication, the datalogger should first be configured for the network. Once connected with *Device Configuration Utility*, the IP settings may be changed on the **Deployment** | **Ethernet** tab as shown in FIGURE 6-1. If using a Campbell Scientific network link interface on the **CS I/O** port, settings need to be entered on the **CS I/O** | **IP** tab. If unsure what to enter for any of these settings, contact your network administrator. Refer to the datalogger or the network link interface manual for more information.

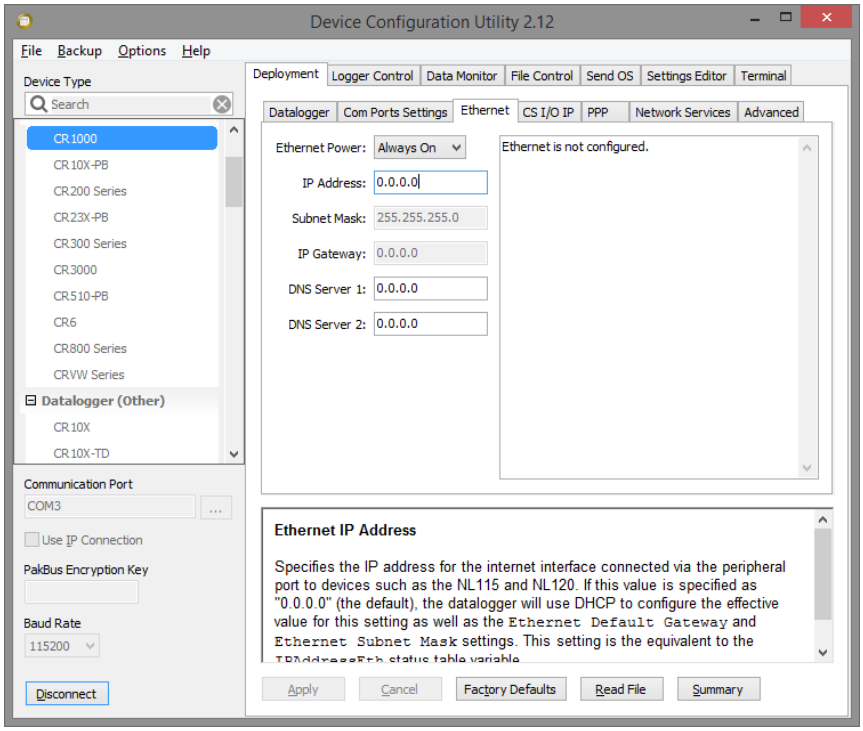
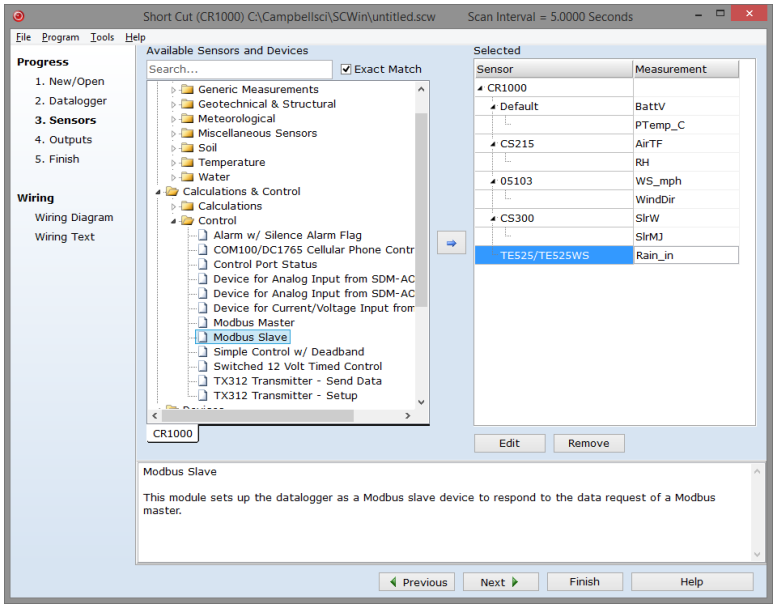


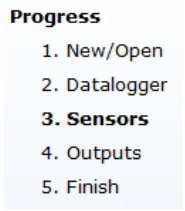
FIGURE 6-1. Entering IP settings in Device Configuration Utility

6.2 Adding Modbus Slave Functionality with *Short Cut*

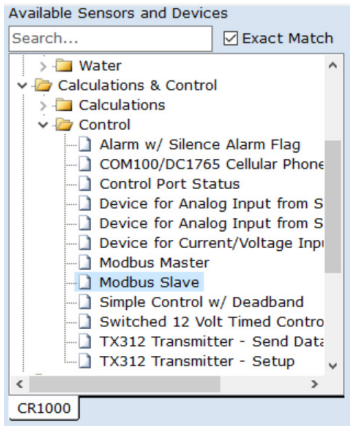
Create a datalogger program in *Short Cut*. For a *Short Cut* tutorial, see the datalogger manual or visit *www.campbellsci.com/videos*.



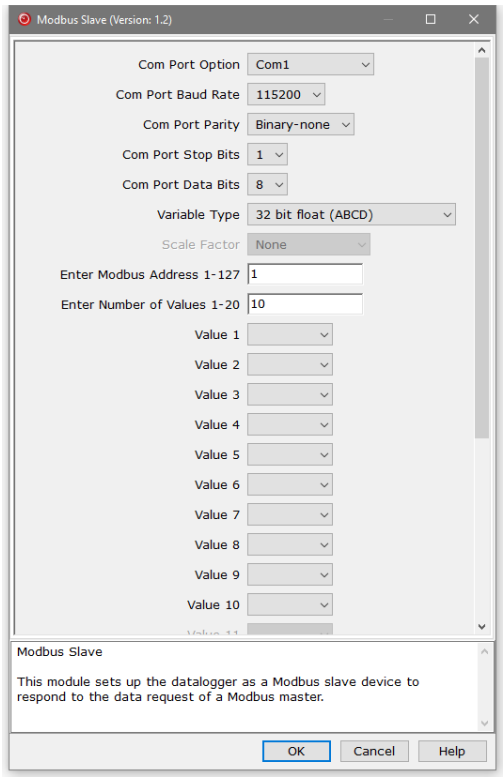
Click **Sensors** in the **Progress** list.



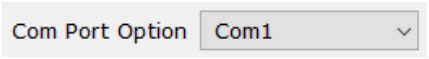
The Modbus slave functionality is added in *Short Cut* similar to adding another sensor. In the **Calculations & Control** | **Control** subfolder, double-click Modbus Slave.



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



Select the **Com Port Option** to use for Modbus communications.

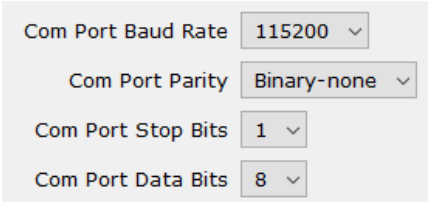


It is important to note that the PakBus® protocol will not work on the selected **Com Port Option** after loading the program. For example, if **ComRS232** is selected, *Device Configuration Utility* will no longer be able to communicate with the datalogger through its **RS-232** port.

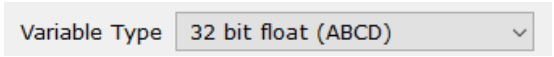
Communication can be recovered by sending a new operating system to the datalogger. This process will do a full reset of the datalogger. To learn how to send an operating system to a datalogger, watch the tutorial video at *www.campbellsci.com/videos?video=76*.

Select the **Com Port Baud Rate**, **Parity**, **Stop Bits**, and **Data Bits** to match the system the datalogger 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.



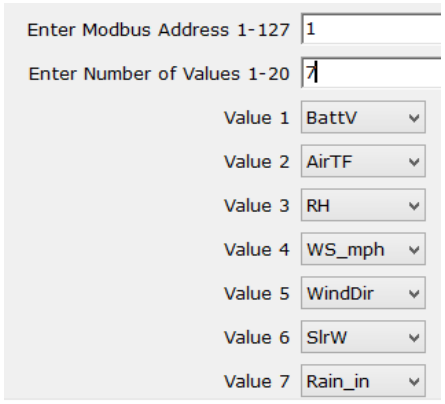
The **Variable Type** field selects the binary data type the datalogger will use for placing data in registers. All the registers on the datalogger will use the same type. Choose a **Variable Type** that is supported by the master. Set the master to the same value. **Variable Type** on other equipment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



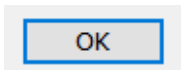
If applicable and necessary, enter the **Scale Factor**. Refer to Section 7.2, *Additional Variable Types* (p. 9), for more information.

Enter the **Modbus Address** to be assigned to the datalogger. Some manufacturers call this the **Modbus ID** or **Modbus Slave ID**.

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 to particular registers.



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



Send the program to the datalogger. Once the program is loaded, the datalogger will be able to respond to Modbus requests from a master device.

Note that 32-bit values use two registers each. This example will have the register map shown in TABLE 6-1.

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 shows *Modbus Poll* software reading holding registers from a CR1000 running the program created by *Short Cut*. The wind speed value, **WS_mph**, of **1.478** miles/hour is seen starting at register 7.

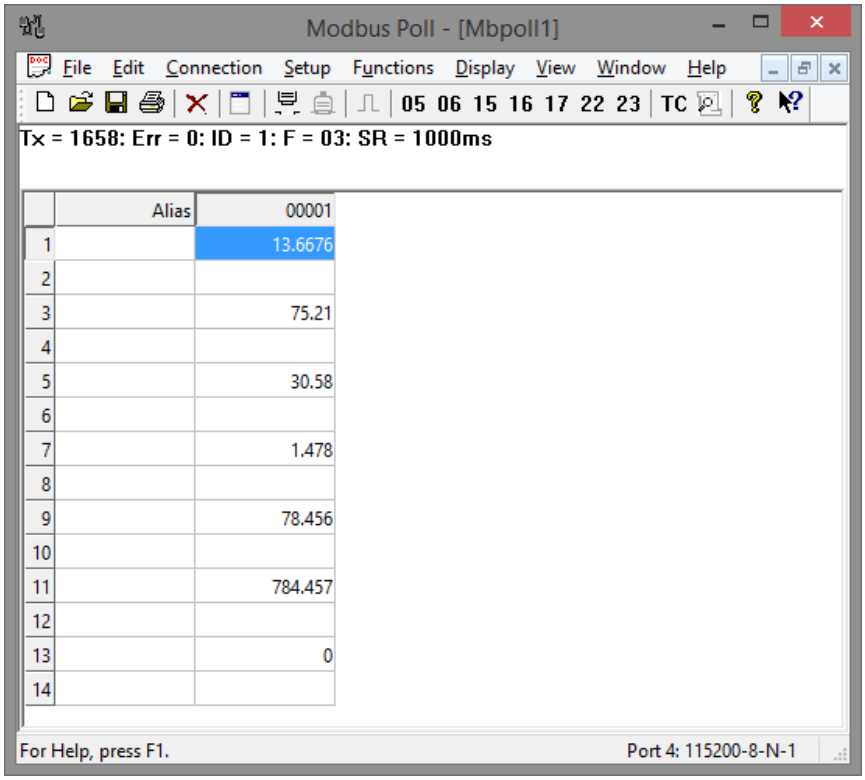


FIGURE 6-2. Modbus Poll *software reading CR1000 holding registers*

7. Advanced Topics

The Modbus functionality accessible by programming with *Short Cut* meets many needs, but it is limited. To access more functionality, the datalogger program must be customized with the *CRBasic Editor*. Using the *CRBasic Editor*, more variables may be mapped to registers, additional variable types are available, and variable types may be mixed within the register map.

7.1 Mapping More Than 20 Values

CRBasic Example 7-1 is the entire program created by *Short Cut* in the previous example. In addition to the needed lines for Modbus communication, it contains measurements and data storage tables. As this document proceeds, abbreviated examples will be shown with only the relevant pieces of the program.

CRBasic Example 7-1. Modbus Program Generated by *Short Cut*

```
'CR1000
'Created by Short Cut (3.2)

'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(Table2,True,-1)
  DataInterval(0,1440,Min,10)
  Minimum(1,BattV,FP2,False,False)
EndTable

'Main Program
BeginProg
  'Use SerialOpen to set RS232 options for Modbus Slave Instruction
  SerialOpen(COM1,115200,3,0,1000)
  'Modbus Slave Instruction
  ModbusSlave(COM1,115200,1,Modbus(),ModbusCoil(),2)

  'Main Scan
  Scan(5,Sec,1,0)
    'Default CR1000 Datalogger Battery Voltage measurement 'BattV'
    Battery(BattV)
    'Default CR1000 Datalogger Wiring Panel Temperature measurement 'PTemp_C'
```

```
    PanelTemp(PTemp_C,_60Hz)
    'CS215 Temperature & Relative Humidity Sensor measurements 'AirTF' and 'RH'
    SDI12Recorder(TRHData(),7,"0","M!",1,0)
    AirTF=AirTF*1.8+32
    '05103 Wind Speed & Direction Sensor measurements 'WS_mph' and 'WindDir'
    PulseCount(WS_mph,1,1,1,1,0.2192,0)
    BrHalf(WindDir,1,mV2500,1,1,1,2500,True,20000,_60Hz,355,0)
    If WindDir>=360 Or WindDir<0 Then WindDir=0
    'CS300 Pyranometer measurements 'SlrMJ' and 'SlrW'
    VoltSE(SlrW,1,mV250,2,1,0,_60Hz,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,1,2,2,0,0.01,0)
    'Call Data Tables and Store Data
    CallTable Table2
    '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 in a single array. The **ModbusSlave()** instruction references that array. To map more values, extend the array and copy in more values.

Variables are declared at the beginning of the program. The **Modbus** array currently is sized to hold up to 7 values. It is declared with this line:

```
Public Modbus(7)
```

If we want to write up to 30 values to the array, we just change the declaration like this:

```
Public Modbus(30)
```

Once the array is sized larger, more values can be copied to the array. In almost all cases, data values should be copied to the array within the scan. The scan is contained between the **Scan()** and **NextScan()** instructions. It makes sense to add lines in the same area where *Short Cut* is already copying measurement in values to the array.

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

Additional measurement values are assigned using the same assignment syntax as shown above.

7.2 Additional Variable Types

The last parameter of the **ModbusSlave()** instruction allows specifying the variable type to be applied to all registers.

```
ModbusSlave(COM1,115200,1,Modbus(),ModbusCoil(),2)
```

The default internal type for variables is 32-bit floating point. To use an integer type for the Modbus registers, the **Modbus** array used to hold the data needs to be set to the **Long** data type. The **Long** internal type is a 32-bit signed integer. The array is set to the **Long** type by adding **as Long** to the declaration as shown above.

```
Public Modbus(7) as Long
```

Integer data types will truncate anything after the decimal point. To keep resolution on your measurements, you may need to scale the values. Scaling can be done by adding math functions to the assignment lines of code. For example, to keep a resolution of 0.1 degrees on the temperature measurement in CRBasic Example 7-1, multiply by 10:

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

The Modbus register will then hold tenths of a degree. The Modbus master device will receive a data value of **752** instead of the **75.21** original data value. In most cases, the master device could be configured to scale by 0.1 to get the value back into degrees (**75.2**). *Short Cut* allows selection of integer types and applying a scaling factor (**Scale Factor** field). *CRBasic Editor* is needed if scale factor should not apply to all registers.

7.3 Coils

The example program includes an array called **ModbusCoil**. It must be declared as type **Boolean**. This array is used for the ***BooleanVariable*** parameter of the **ModbusSlave()** instruction, which maps the coils.

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

The example program did not assign any values to the coils. A coil may be set within the program in the same way that values were assigned to registers. In the example below, **LineState** could be declared as a **Boolean**, **Float**, or **Long**. A zero value will result in a coil state of **0**, or not set. Any value other than zero will result in a coil state of **1**, or set.

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

Some functions and measurement instructions directly output a Boolean state. For example, the state of a control port will be read as either true or false. A coil can be used as the destination variable. In this example, the state of the **C1** terminal is directly saved in **ModbusCoil(1)**:

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

7.4 Mixing Variable Types Within a Register Map

It is possible to map some values to registers as integers and other values as floating point. To do this, the array of variables used for **ModbusSlave()**

should be set to type **Long**. Integer values can be assigned as shown previously. Floating point values are placed in registers by using the **MoveBytes()** instruction. The **MoveBytes()** instruction will do a binary copy without changing the format. CRBasic Example 7-2 demonstrates placing an integer in registers 1 and 2, with a floating point number in registers 3 and 4.

CRBasic Example 7-2. Mixing Variable Types

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

BeginProg
  'Use SerialOpen to set RS232 options for Modbus Slave Instruction
  SerialOpen(COM1,115200,3,0,1000)
  'Modbus Slave Instruction
  ModbusSlave(COM1,115200,1,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
```

7.5 Changing Byte Order

If you need ABCD byte order (big endian) in place of CDAB order, that can be changed simply with a parameter of **ModbusSlave()**. Going from ABCD byte order to DCBA byte order (little endian) requires using the **MoveBytes()** instruction. Some datalogger models internally use DCBA byte order and other models use ABCD order. The **MoveBytes()** entry in the *CRBasic Editor Help* lists whether a particular datalogger model is big endian or little endian.

This example comes from the help for **MoveBytes()**. It reverses the byte order of a 32-bit floating point number. The same code will work for reversing the byte order of 32-bit integers.

CRBasic Example 7-3. Changing Byte Order

```
Public big_endian_num
Public lit_endian_num
Dim m, n
BeginProg
    Scan (1,sec,0,0)
      n = 0
      For m=3 to 0 step -1
       MoveBytes (lit_endian_num,m,big_endian_num,n,1)
       n=n+1
      Next m
    NextScan
EndProg
```

Campbell Scientific Companies

Campbell Scientific, Inc.
815 West 1800 North
Logan, Utah 84321
UNITED STATES
www.campbellsci.com • info@campbellsci.com

Campbell Scientific Canada Corp.
14532 – 131 Avenue NW
Edmonton AB T5L 4X4
CANADA
www.campbellsci.ca • dataloggers@campbellsci.ca

Campbell Scientific Africa Pty. Ltd.
PO Box 2450
Somerset West 7129
SOUTH AFRICA
www.campbellsci.co.za • cleroux@csafrica.co.za

Campbell Scientific Centro Caribe S.A.
300 N Cementerio, Edificio Breller
Santo Domingo, Heredia 40305
COSTA RICA
www.campbellsci.cc • info@campbellsci.cc

Campbell Scientific Southeast Asia Co., Ltd.
877/22 Nirvana@Work, Rama 9 Road
Suan Luang Subdistrict, Suan Luang District
Bangkok 10250
THAILAND
www.campbellsci.asia • info@campbellsci.asia

Campbell Scientific Ltd.
Campbell Park
80 Hathern Road
Shepshed, Loughborough LE12 9GX
UNITED KINGDOM
www.campbellsci.co.uk • sales@campbellsci.co.uk

Campbell Scientific Australia Pty. Ltd.
PO Box 8108
Garbutt Post Shop QLD 4814
AUSTRALIA
www.campbellsci.com.au • info@campbellsci.com.au

Campbell Scientific Ltd.
3 Avenue de la Division Leclerc
92160 ANTONY
FRANCE
www.campbellsci.fr • info@campbellsci.fr

Campbell Scientific (Beijing) Co., Ltd.
8B16, Floor 8 Tower B, Hanwei Plaza
7 Guanghua Road
Chaoyang, Beijing 100004
P.R. CHINA
www.campbellsci.com • info@campbellsci.com.cn

Campbell Scientific Ltda.
Fahrenheitstraße 13
28359 Bremen
GERMANY
www.campbellsci.de • info@campbellsci.de

Campbell Scientific do Brasil Ltda.
Rua Apinagés, nbr. 2018 ─ Perdizes
CEP: 01258-00 ─ São Paulo ─ SP
BRASIL
www.campbellsci.com.br • vendas@campbellsci.com.br

Campbell Scientific Spain, S. L.
Avda. Pompeu Fabra 7-9, local 1
08024 Barcelona
SPAIN
www.campbellsci.es • info@campbellsci.es

Please visit www.campbellsci.com to obtain contact information for your local US or international representative.