How Bristol® MODBUS Master Works

By Robert Findley

OK, imagine this scenario. You are the process engineer on a standard station design integrating a basic control and monitoring scheme using a Bristol® RTU. There are a number of physical inputs and outputs and you are communicating this data to a HMI using our standard BSAP protocol. Your control application works great, you have solid communication back to the control room and you are just about to claim victory as everything is working just fine when ----- BAM !!! it hits you. Located on a wall across the room there is a 3rd party vendor device. You find out that the only access to that device is MODBUS communications and the control room needs data from it. You quickly think your day is not over. By the way it's Friday afternoon!

Well **don't panic** because fortunately you have a Bristol RTU from Emerson Process Management. Our communication ports can be configured to work with various products via their native communication language (known as Protocol). MODBUS is embedded in the firmware of every RTU, along with a long list of other protocols so you won't have to make another purchase. The simplicity of the MODBUS protocol makes it one of the most widely used among many different product vendors. Now let's talk about what you need to make MODBUS Master work!

1) Ask for the 3rd party device node address
MODBUS is a standard communications protocol (language) which has been around since around 1979. It is based upon a specific message format and a basic request and reply concept. By definition, the standard protocol provides for 1 master and up to 247 slave devices. The master is the only node which can initiate a message. Message framing sequence of messages and error response codes are defined within the protocol structure.

2) What is the "Starting Register Address" of the data you are trying to poll
MODBUS protocol can be sent to slave devices in two ways: Poll and respond or Broadcasting. Poll and respond is very simple. An addressed message is sent to the entire network and only the addressee responds. The message contains what is being requested by signifying where it resides in the slave unit by memory location (called a Register). Also, the message defines how many registers to deliver from the initial register requested. The slave then responds to the message with the appropriate information. Broadcasting allows points in the entire network to be forced in the slave devices and does not require a response message. This is done by using the default address zero in the poll message. While this in not used as often, it is a concept allowed within the MODBUS protocol.

3) Will you be using RTU or ASCII message framing method?
MODBUS has two modes of transmission called ASCII or RTU framing. This is an important piece of information that is required for both the master and slave devices. ASCII message framing is a 7 bit hexadecimal representation of data while RTU framing is an 8 bit binary representation of data. Simply put, it is the method used to represent data in transmitting messages. These coding systems help to abbreviate large blocks of data into manageable pieces.

4) What type of data are you polling for: Analog or Digital?
This one sounds tricky, but it is actually quite simple once you understand the concept. Since data is stored in register locations in the slave device you can segment analog signals and logical data (on/off) into different areas. When polling for data from the master, the message must contain the type of the data you are trying to gather. Function codes, which are embedded within the message, signify what action (read or write) is being requested from a device. There are 19 standard function codes, but the basic function codes are as follows:

Function Code 01 – Read Coils (i.e. Read Digital Values
Function Code 02 – Read Inputs (i.e. Read Digital Inputs)

Function Code 03 – Read Registers (i.e. Read Analog/Variable Signals)

Function Code 04 – Read Input Registers (i.e. Read Analog Inputs or fixed signals)

Function Code 05 – Force Single Coil (i.e. Write to Single Digital Output)

Function Code 06 - Preset Single Register (i.e. Write to Single Analog Register)

There are many other function codes, which all Emerson products support, that are discussed in our manuals. Please reference these manuals for advanced communications and concepts.

(These links requires registration on our website. If you do not have a user name and login for this, obtain a login ID and password before following the link by going to http://www.emersonprocess.com/signup/index.htm)

For 33xx users – http://www.emersonprocess.com/services/documentation/manuals/D4066/Gouldmbs.pdf

For CW users - http://www.emersonprocess.com/services/documentation/manuals/D5125/Modbus.pdf

Now that you have the information you need, how do you make it work? For the purposes of this document, I will cover the basic steps. Please reference the manual for the exact details.

Step 1 – Configure the Physical Port

The physical hardware port must be told what protocol will be used to communicate a specific protocol. This is done in the communications section of an ACCOL load or in the flash memory configuration of a ControlWave product.

Step 2 – Configure the Custom Module Within the Application Load

As with any Emerson product using a protocol (other than our native BSAP protocol), you need to define the port which will be used. The Custom Function Block allows the programmer to select which protocol should be used, as well as other configuration information necessary for enabling the protocol.

The most important terminals on the Custom Block are:

Mode – code for the protocol being used

Custom List – List number in application load that defines the configuration parameters

Com Port – code for the communication port to send the protocol from

Timeout – Length of time to wait for a response from a slave

Status - THE MOST IMPORTANT SIGNAL as it will give you the necessary clues if the protocol fails to communicate with the device.

There are other terminals on the custom module, but at a basic layer, these are the most important

Step 3 – Define the Configuration List

This sets up a “mapping list” which will correlate to the set of MODBUS Register addresses (i.e. a coils list, a inputs list, a registers list etc...). Each list will have Bristol signals assigned to the terminals on the

list. They can be named by the user in any manner, which allow our users to give a register location number a defined signal name. This is a very useful feature. After the slave device is polled with a request from the master, the response message registers will be mapped into the corresponding terminal on the list. Once it becomes an "Emerson signal" it can be used as a normal variable within the application code.

Step 4 – Start Running

I know I have made it sound much simpler than it is, but the Emerson implementation of the Bristol MODBUS protocol has been very successful over the course of many years. Even though MODBUS is a standard, many vendors have strayed from the standard, or extended the standard to provide functionality not originally intended by that standard. The power of the Emerson implementation of MODBUS is that the user has the flexibility to adapt to these variations.

For more information, please contact our Technical Support group at bsupport@emersonprocess.com

Emerson Process Management
Remote Automation Solutions

Watertown, CT 06795 USA T 1 (860) 945-2200
Mississauga, ON 06795 Canada T 1 (905) 362-0880
Worcester WR3 8YB UK T 44 (1) 905-856950

Website: www.EmersonProcess.com/Remote

EMERSON™
Process Management