User Guide

Babel Buster SPX Pro

Model SPX Pro V2
Modbus RTU/TCP, SNMP Gateway
Rev. 1.1 – Dec. 2016

© 2016 Control Solutions, Inc.

SPX Pro V2 User Guide Contents

1 Introduction
1.1 How to Use This Guide
1.2 Overview of Gateway Device
1.3 What is New in Pro Version of Gateway
1.4 Important Safety Notice
1.5 Warranty

2 Connecting Gateway for the First Time
2.1 Connectors and Indicators
2.2 Open Web User Interface

3 Configuring Local Registers
3.1 Creating Local Registers
3.2 Special Features of Local Registers
3.3 Local Register Calculate Rules
3.4 Local Register Copy Rules

4 Configuring Gateway as a Modbus RTU Master
4.1 Modbus RTU Device Configuration
4.2 Modbus RTU Master Read Maps
4.3 Modbus RTU Master Write Maps
4.4 Modbus RTU Master Data Displayed by Slave
4.5 Modbus RTU Errors
4.6 Character String Example

5 Configuring Gateway as a Modbus TCP Client
5.1 Modbus TCP Device Configuration
5.2 Modbus TCP Client Read Maps
5.3 Modbus TCP Client Write Maps
5.4 Modbus TCP Client Data Displayed by Server
5.5 Modbus TCP Errors

6 Configuring Gateway as a Modbus RTU Slave
6.1 Modbus RTU Device Configuration
6.2 Modbus RTU Slave Register Map
6.3 Modbus RTU Slave Diagnostic

7 Configuring Gateway as a Modbus TCP Server
7.1 Modbus TCP Device Configuration
7.2 Modbus TCP Server Register Map

8 Configuring Gateway as an SNMP Server
8.1 Creating Local SNMP MIB
8.2 Supported Data Formats, RFC 6340
8.3 Testing the SNMP Agent

9 Configuring Gateway as an SNMP Client
9.1 SNMP Device Configuration
9.2 SNMP Client Read Maps (Get)
9.3 SNMP Client Write Maps (Set)
9.4 SNMP Client Data Displayed by Agent
9.5 Supported Data Formats
9.6 SNMP Errors

10 Configuring SNMP Table Walker
10.1 Table Walk Methods
10.2 Configuring Table Walk Rules
10.3 Table Walk Errors

11 Configuring SNMP Trap Sender
11.1 SNMP Trap Destinations
11.2 SNMP Trap Triggers
11.3 SNMP Trap Summary
11.4 SNMP Identity
11.5 Testing the SNMP Trap Sender

12 Configuring SNMP Trap Receiver
12.1 Trap Receive Devices
12.2 Trap Receive Rules
12.3 Trap Receive Examples for SNMPv1
12.4 Trap Receive Example for SNMPv2

13 Programming with Script Basic
13.1 Creating a Program
13.2 Testing the Program
13.3 Setting the Program to Auto-Run on Startup
13.4 Special Functions for SNMP Access and Diagnostics
13.5 Example: Writing Alphanumeric Display
13.6 Example: Trap Receiver Test
13.7 Example: Trap Info to Alphanumeric Display
13.8 Example: Device and Rule Diagnostics
13.9 Example: SNMP Enabling a Serial Device
13.10 Example: Modbus Enabling a Serial Device

14 System Configuration and Resources
14.1 Configuration Files
14.2 Network Configuration
14.3 Resource Allocation
14.4 User Login Passwords

15 Trouble Shooting
15.1 Modbus RTU Trouble Shooting
15.2 Modbus TCP Trouble Shooting
15.3 SNMP Trouble Shooting
15.4 Modbus Reference Information

Appendix A Hardware Details
A.1 Wiring
A.2 Front Panel LED Indicators
A.3 RS-485 Line Termination and Bias

Appendix B Example Application: RFC 1628 UPS
B.1 MIB Query
B.2 Alarm Table Walker
B.3 Trap Receiver
B.4 Example Data

Appendix C Using Wireshark for Trouble Shooting
C.1 Hardware Requirements
C.2 Example of Using Wireshark

1. Introduction

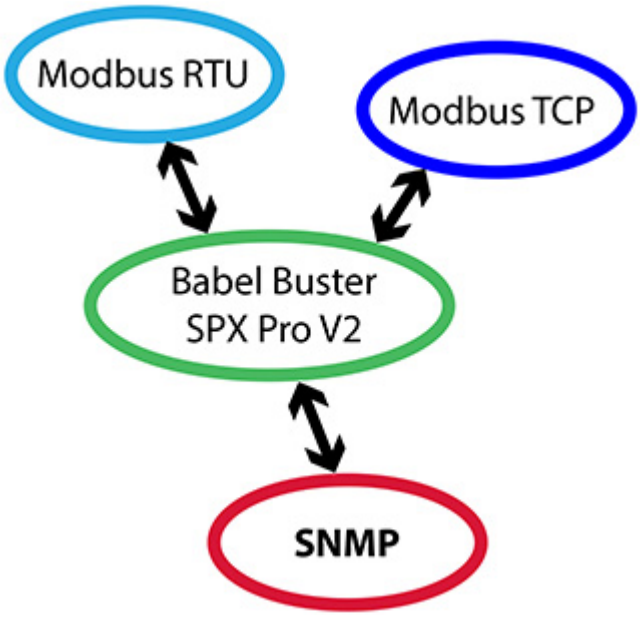
1.1 How to Use This Guide

This user guide provides background information on how the gateway works, and an overview of the configuration process. There are several sections for groups of tabs found in the web interface in the gateway which is accessed by opening a web browser and browsing to the IP address of the device.

You should at least read the overview sections to gain an understanding of how the gateway functions. You can use the remaining sections as reference material to look up as needed. There is a "Quick Help" section at the bottom of each web page in the gateway which is generally sufficient for quick reference in setting up the gateway.

1.2 Overview of Gateway Device

The Babel Buster SPX Pro V2 is a Modbus gateway used primarily to interface between Modbus devices and SNMP devices. The gateway may be used as Modbus TCP client and server, Modbus RTU master or slave, SNMP server (agent), and SNMP client (manager). The SPX Pro V2 includes additional special features allowing it to Walk SNMP tables, and receive traps from other SNMP devices. The SPX Pro V2 includes Script Basic for local programming to interpret trap messages that do not otherwise fall into a trap receive rule template.



The most common application for these gateways is interfacing a Modbus device to an SNMP network, or vice versa. Another common application is remapping registers from multiple slave devices to appear as a single device on the alternate network, mapping RTU to TCP or vice versa.

1.3 What is New in Pro Version of Gateway

The most significant enhancements in the Pro gateway are the addition of extremely powerful trap receiver capabilities, and the MIB Walk capability. The MIB walker can traverse straight contiguous tables or sparse tables with wildcard fields in the OID to act upon for capturing data.

The other feature in the Pro gateway that is a departure from the norm in Control Solutions Modbus gateways is the ability to let the user create Modbus registers for handling data ranging from 16-bit to 64-bit, integer or floating point. The local register map is created dynamically by the user. There is no preconfigured, fixed register map.

Another useful feature in the Pro gateway is the treatment of ASCII character strings as a series of registers. This series of registers is treated as a single data element by the gateway. This is particularly significant when using Script Basic, which has a rich library of character string manipulation functions. The Basic "getreg" and "setreg" functions can read and write entire character strings as a single register. When interfaced to an alphanumeric display device having a Modbus interface, generating custom display messages becomes straight forward.

This simple program:

```
setreg 21, "Hello from Basic"
setreg 41, "This is a demo"
setreg 61, "of writing display"
setreg 81, "from program."
```

Becomes this display message:

More detail about programming is found later in this User Guide.

1.4 Important Safety Notice

Proper system design is required for reliable and safe operation of distributed control systems incorporating any Control Solutions product. It is extremely important for the user and system designer to consider the effects of loss of power, loss of communications, and failure of components in the design of any monitoring or control application. This is especially important where the potential for property damage, personal injury, or loss of life may exist. By using ANY Control Solutions, Inc., product, the user has agreed to assume all risk and responsibility for proper system design as well as any consequence for improper system design.

1.5 Warranty

This software and documentation is provided "as is," without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of fitness or merchantability for a particular purpose. Control Solutions may make improvements and/or changes in this documentation or in the product(s) and/or the program(s) described in this documentation at any time. This product could include software bugs, technical inaccuracies, typographical errors, and the like. Changes are periodically made to the information herein; these changes may be incorporated in new editions of the software.

2. Connecting Gateway for the First Time

2.1 Connectors and Indicators

Follow these steps to make the initial connection to the SPX Pro V2.

(a) Connect power. Apply +12 to +24VDC or 24VAC to the terminal marked "POWER", and common or ground the the terminal marked "GND".

(b) Connect a CAT5 cable between the RJ-45 jack on the gateway, and your network switch or hub. You cannot connect directly to your PC unless you use a "crossover" cable.

(c) Apply power.

A green LED inside the case should light indicating power is present.

If the link LED on the RJ45 jack is not on, check your Ethernet cable connections. Both link and activity LEDs on the RJ45 jack will be on solid for a short time during boot-up. The entire bootup process will take 1-2 minutes, during which time you will not be able to connect with a browser.

Ethernet link LED is the yellow LED integrated into the CAT5 connector. Ethernet activity LED is the green LED integrated into the CAT5 connector.

Refer to Appendix A for additional detail pertaining to connections and indicators as well as optional internal jumper settings.

2.2 Open Web User Interface

The default IP address as shipped is 10.0.0.101. If your PC is not already on the 10.0.0.0 domain, you may need to add a route on your PC (XP only). Do this by opening a command prompt. First type “ipconfig” and note the IP address listed. This is your PC’s IP address. Now type the command

```
route add 10.0.0.0 mask 255.255.255.0 1.2.3.4
```

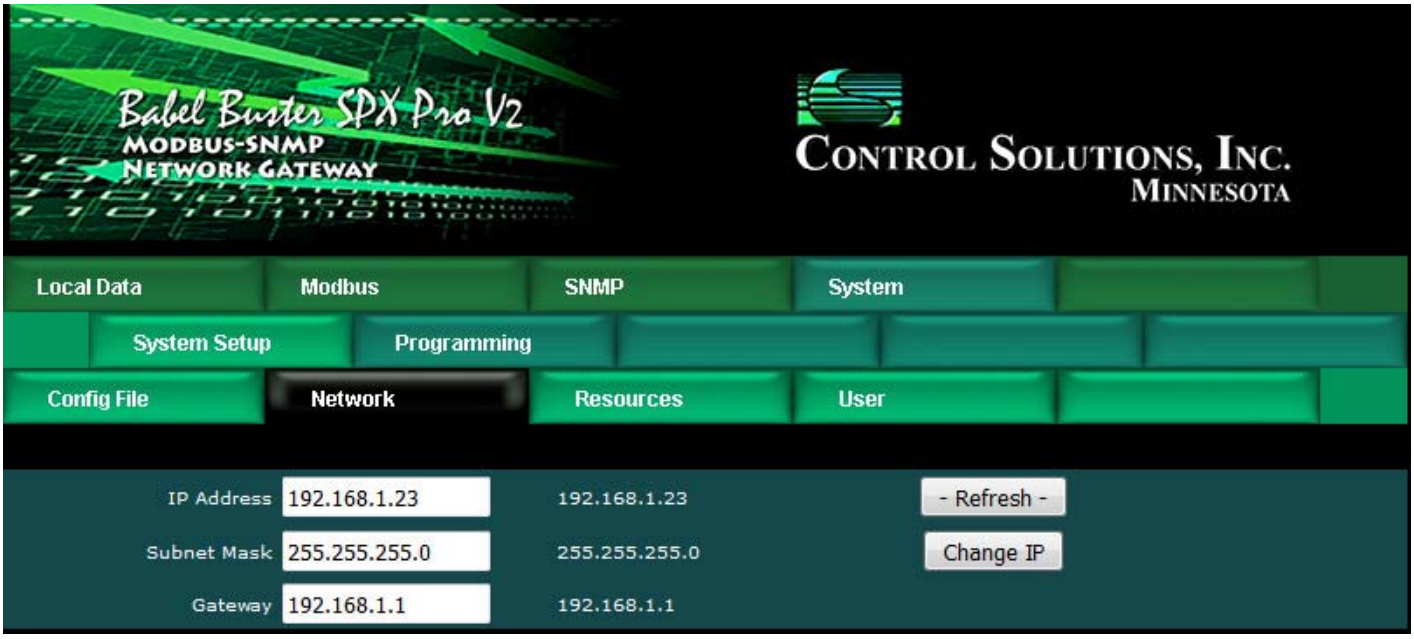
but substitute your PC’s IP address for 1.2.3.4.

This generally works, but if this fails, you will need to temporarily change your computer’s IP address to a fixed address that starts with 10.0.0. and ends with anything but 101.

Open your browser, and enter “http://10.0.0.101/” in the address window. You should see a page with the “Babel Buster SPX Pro V2” header shown above. From this point, you will find help on each page in the web site contained within the product.

When you click on any of the page tabs such as System, you will be asked for a user name and password. The default login is user name “system” with password “admin”. You can also log in as “root” using password “buster”. You must be logged in as the root user to add other users.

To change the IP address of the gateway, go to the Network page under System :: System Setup. The following page should appear (only top portion illustrated here). Change the IP address, and subnet mask and gateway if applicable. Click Change IP to save the changes. The process of programming this into Flash takes around half a minute. The new IP address only takes effect following the next system restart or power cycle.



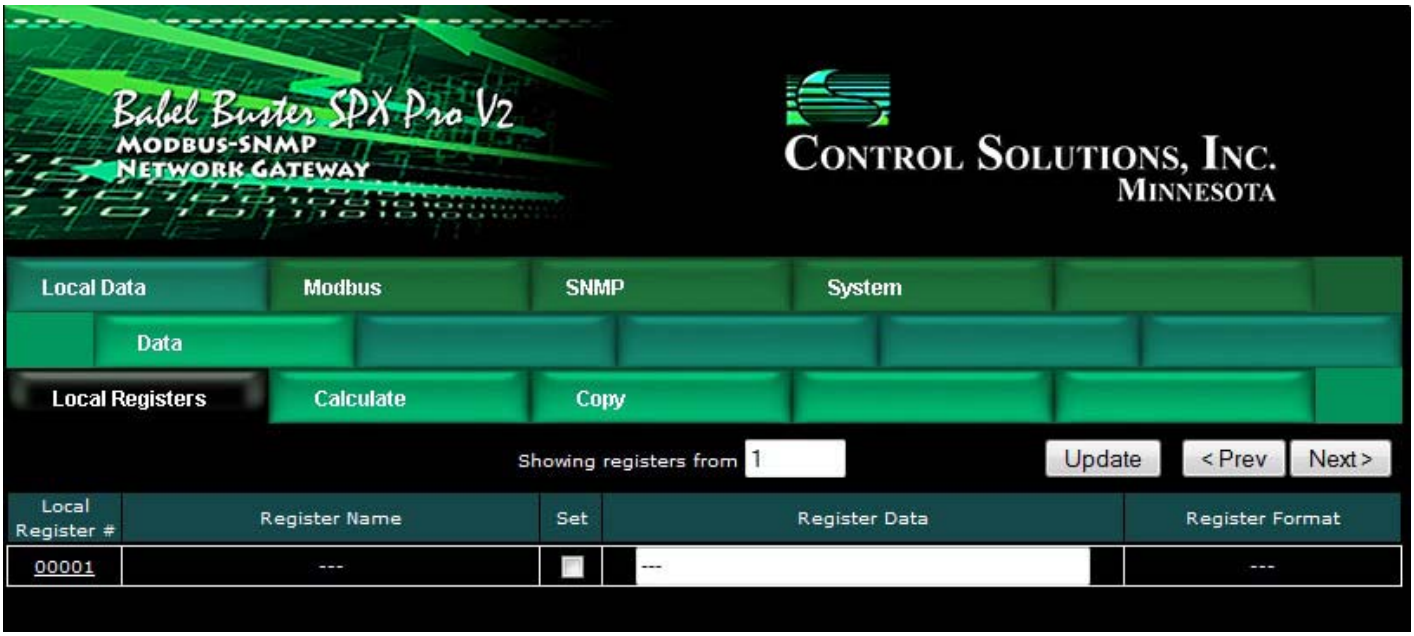
Most changes are stored in an XML configuration file in the device's Flash file system. Only a few are stored differently, and the IP address is one of those. Normally, clicking Update on any configuration page only stores that configuration information to a temporary RAM copy of the configuration file. To make your changes other than IP address permanent, you must click Save on the Config File page (System :: System Setup :: Config File). Refer also to section 14.1.

3. Configuring Local Registers

3.1 Creating Local Registers

Babel Buster gateways originally came with a predefined set of local registers that were accessible externally as Modbus registers. The Pro series takes a completely different approach to defining registers. When you first take the Babel Buster Pro out of the box, it has no registers. You get to create registers as you need them, and several data formats are supported, from 16-bit to 64-bit, both integer and floating point.

Your first visit to the Local Registers page will be the uneventful display illustrated below.

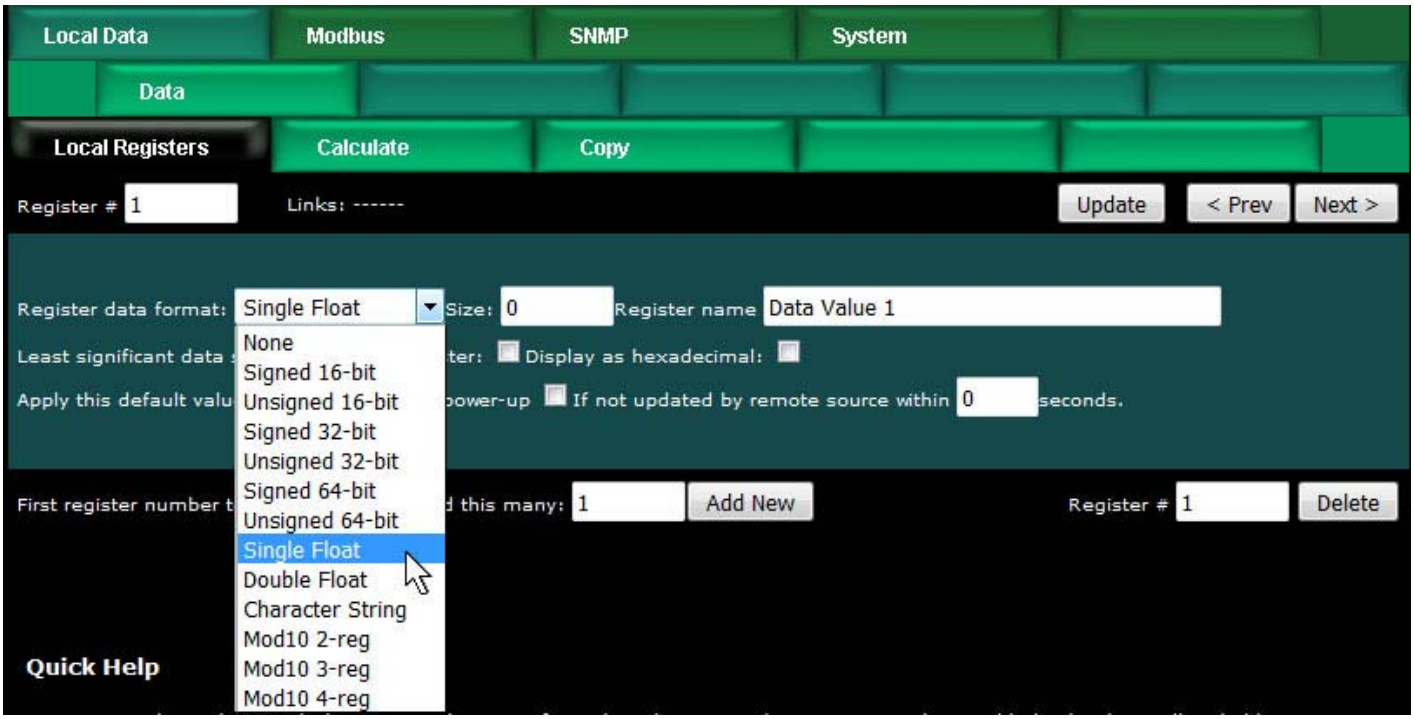


To begin the process of creating registers, click on the only register number available at this point. Later on, click on the register number at the end of the list to add more, or click anywhere in the middle of the list if there are gaps in the register number sequence that you would like to fill.

Upon clicking a register number, the register detail will be displayed. This can be either detailed configuration of an existing register, or detail about new registers you are about to add. The only time you can select the data format is when adding new registers. Once the registers are created, the data format cannot be changed because this impacts how many Modbus registers are actually used. If you had a set of 16-bit registers defined, and wanted to change one of them to 32-bit, it would cause all of the remaining registers to be renumbered if such a change was allowed. While this may seem harmless at first, it becomes a huge mess trying to keep track of where in all the rules the existing numbers needed to get changed. Therefore, such changes are prohibited.

Select the data format for the new registers to be created. The designations Signed and Unsigned refer to integers. Float refers to IEEE 754 floating point format. Character string refers to a series of registers with two ASCII characters per 16-bit register. Mod10 refers to a format unique to Schneider Electric power meters (refer to Schneider Electric documentation for a definition of those formats).

IMPORTANT: Modbus protocol knows holding registers (or input registers) to simply be a 16-bit piece of data. The protocol knows nothing about signed or unsigned integer - that interpretation is up to you. The 16 bits may even be a collection of 16 status flags. If a register is defined in the Babel Buster Pro as anything bigger than 16 bits, it is actually a pair (or series of 2 or more) 16-bit registers. Again, Modbus protocol does not know anything about floating point, character strings, etc. Modbus only knows how to send 16 bits of something at a time when function codes reference a holding register or input register. Modbus only knows how to send 1 bit at a time if referenced as a coil or discrete input. It is up to the Modbus master to be smart enough to ask for 2 registers at a time if it knows it wants to read a 32-bit value.

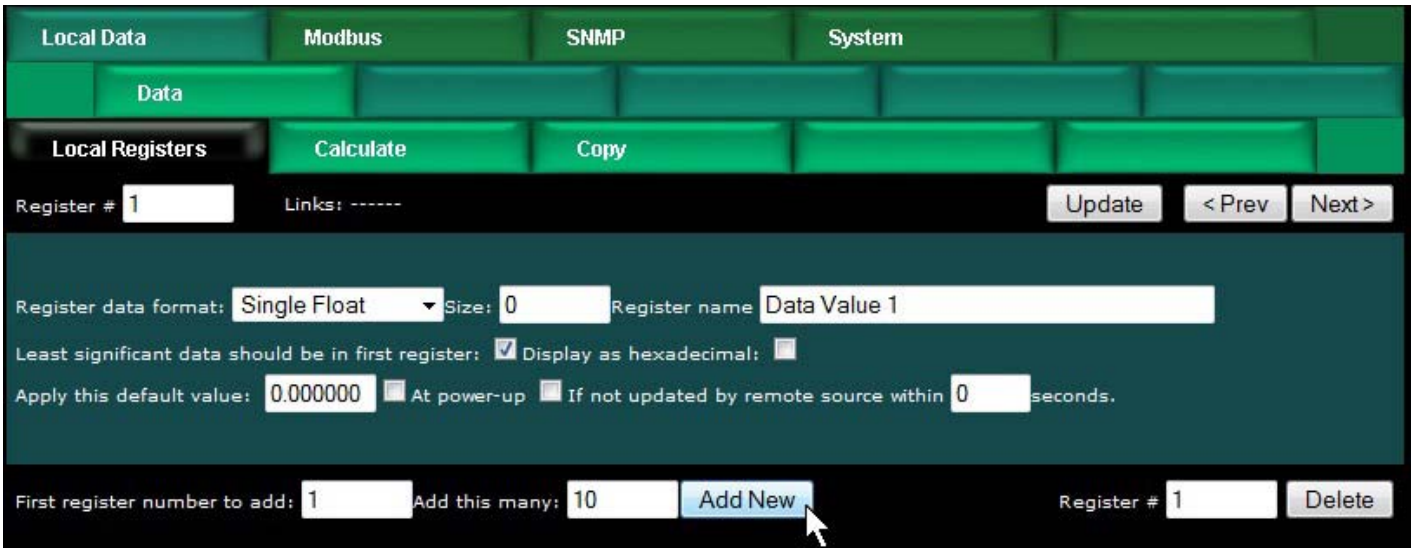


In addition to selecting data format for creating new registers, provide a temporary register name. You can change this later, but it is usually helpful to start with something for a name, and it is suggested that the name ends with a number. The reason why is illustrated shortly.

Modbus protocol is strict about 16-bit increments of data for holding registers. However, when register pairs (or quads) are used to hold a 32-bit (or 64-bit) value, the order in which those registers are interpreted is not defined by any standard. It is up to you to keep track of that. Babel Buster Pro supports interoperability with other Modbus devices by letting you specify what order should be used internally. If the least significant data should appear in the first (or lower numbered) register, then check the box that says "Least significant data should be in first register" either when creating the register, or later by reconfiguring the existing register.

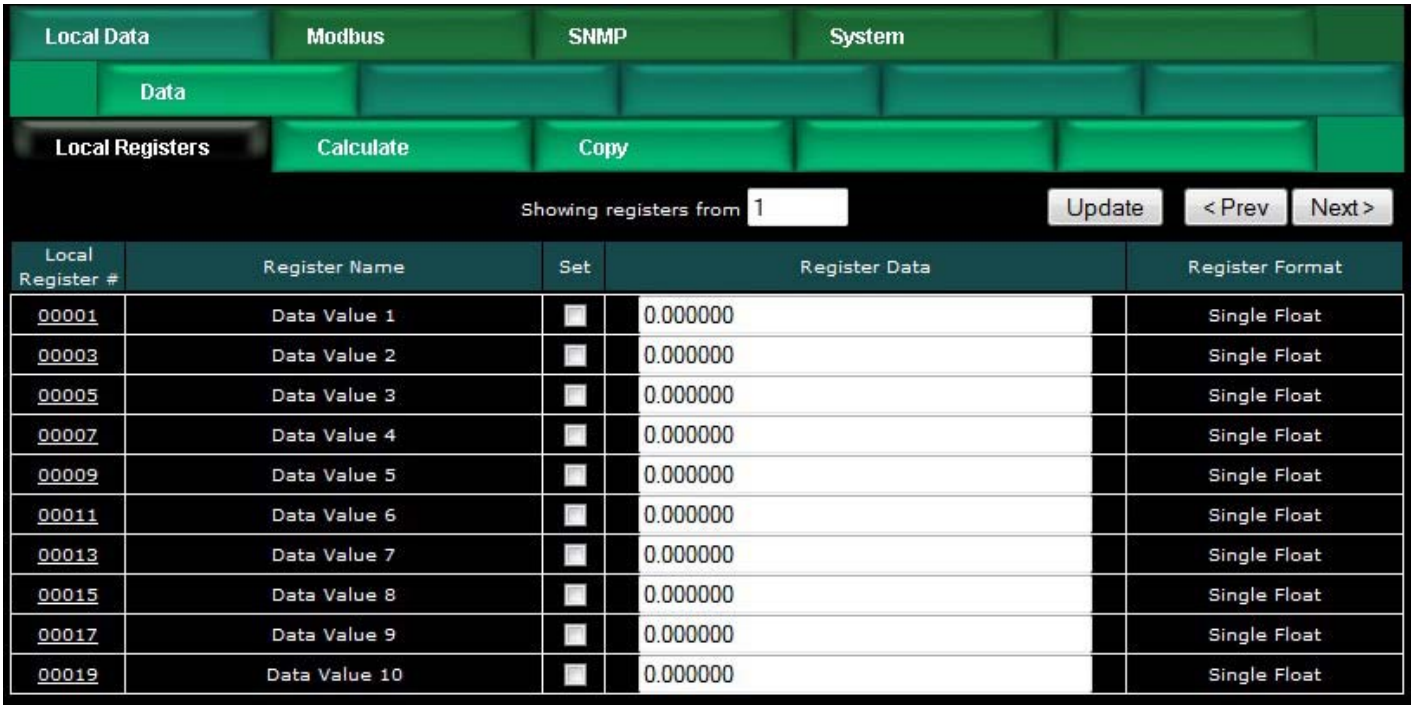
When first creating registers, you do not need to enter any of the default information on the last line. (You can, but don't have to.) The size only applies when creating character string variables (illustrated later, below).

The first register number to add, indicated at the bottom of the page, will be the first available register slot that is not yet assigned. You can enter some different number here. It is not required to create registers in contiguous order. You can jump around, as long as registers don't try to overlap. Select a number of registers to create, and click Add New. If you attempt to add registers that overlap existing registers, the allocation algorithm will find an available slot for you and fill that instead.

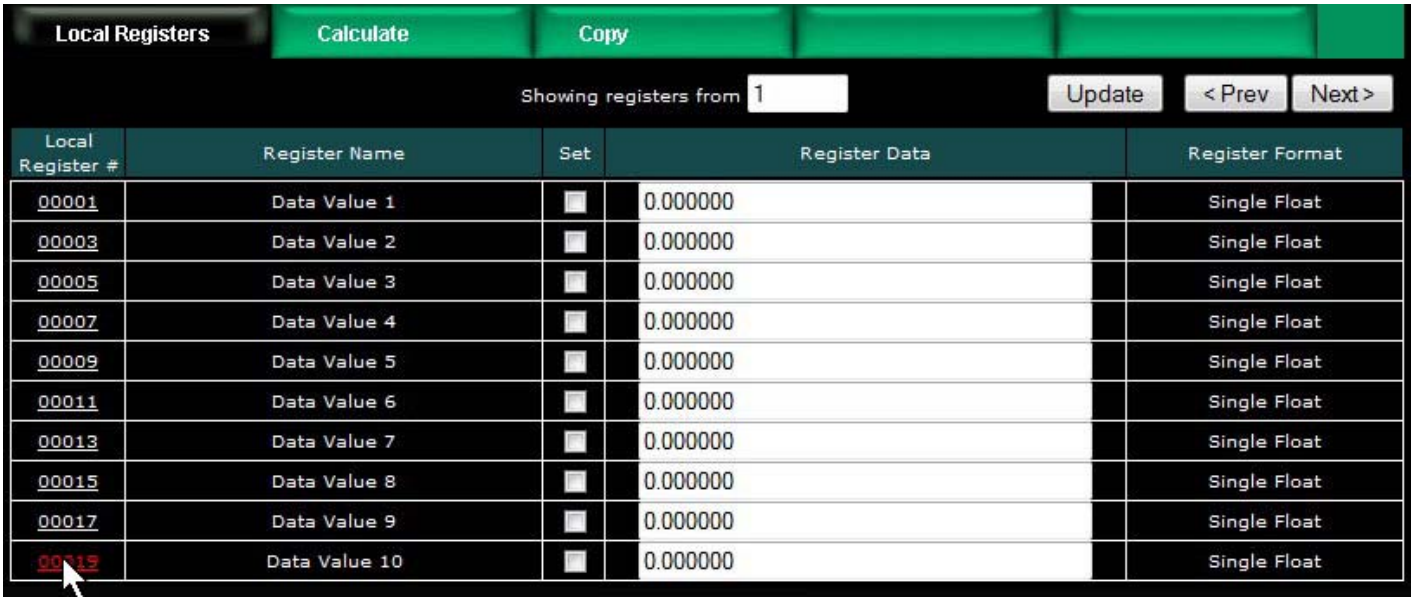


We have now created 10 new floating point registers. The important thing to observe here is the register numbers in the first column. Remember that Modbus protocol assigns register numbers (or addresses) in 16-bit increments. Since single precision floating point registers are 32-bit registers, each floating point value occupies 2 spots in the Modbus register map. The local register number assignments reflect this fact.

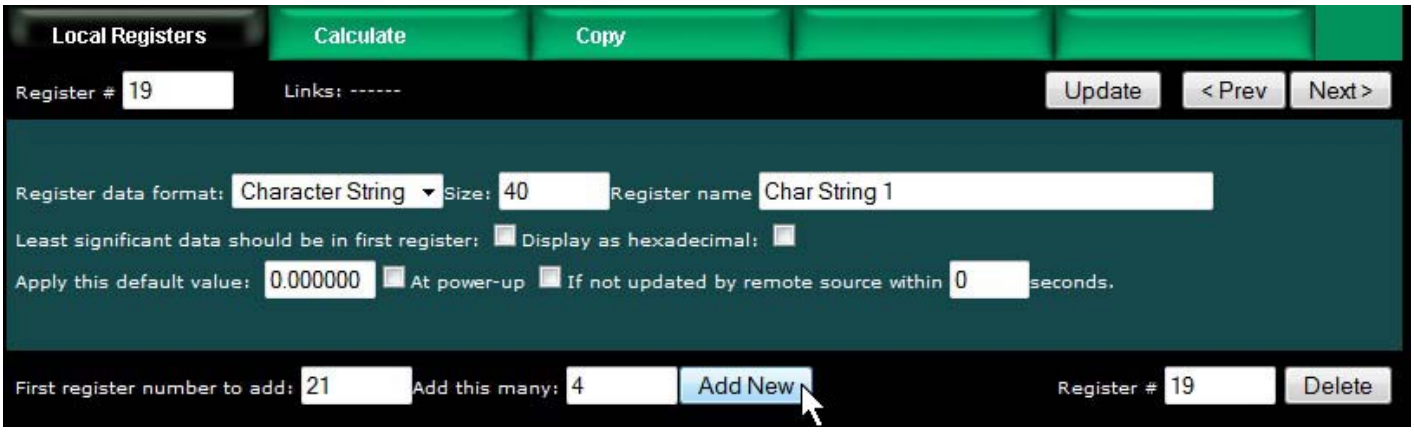
Note that the name given in the screen above was "Data Value 1" but our series of 10 new registers came up with 10 sequential names. If the last thing to appear in the name given when assigning new registers is a number, this value will be incremented by one in the name of each successive new register.



Now let's proceed to add some character strings. Click on the last register number assigned thus far to open the register detail page.



Select Character String for data format. This is the one time a size must be specified. In the example below, we are going to allocate 40-character strings. This means that each "register" will actually be a series of 20 Modbus registers (two ASCII characters per register). The character string processing assumes that any display device used in conjunction with these registers will display the high order byte on the left and low order byte on the right in any given 2-character portion of the display screen. This is consistent with display devices that have been tested with Babel Buster Pro.



After clicking Add New, we now see our character string registers in our register list (click on the Local Registers tab to get back to the register list). Note that the register numbers increment by 20 to accommodate our 40-character strings.

Local Data | Modbus | SNMP | System

Data

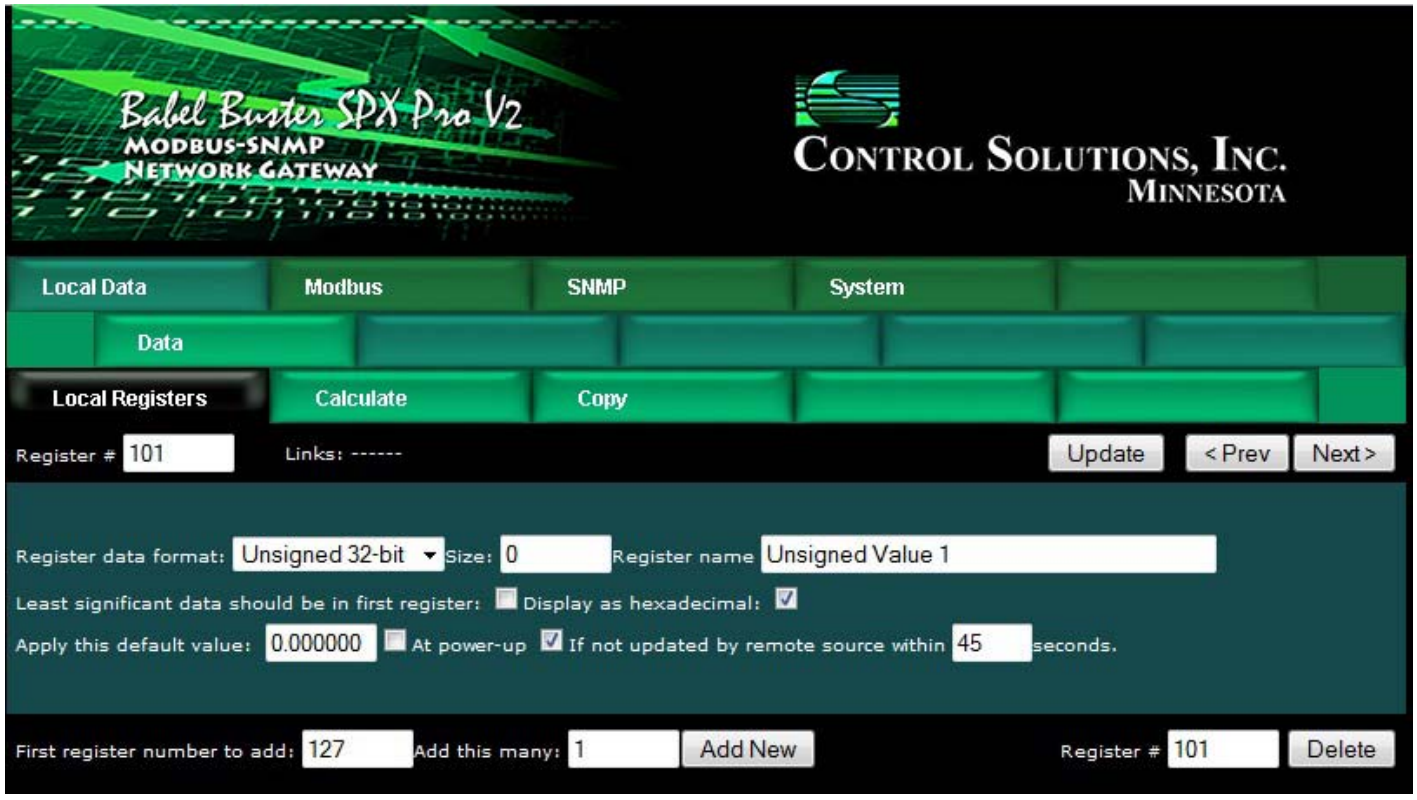
Local Registers | Calculate | Copy

Showing registers from 1 Update < Prev Next >

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	☐	0.000000	Single Float
00003	Data Value 2	☐	0.000000	Single Float
00005	Data Value 3	☐	0.000000	Single Float
00007	Data Value 4	☐	0.000000	Single Float
00009	Data Value 5	☐	0.000000	Single Float
00011	Data Value 6	☐	0.000000	Single Float
00013	Data Value 7	☐	0.000000	Single Float
00015	Data Value 8	☐	0.000000	Single Float
00017	Data Value 9	☐	0.000000	Single Float
00019	Data Value 10	☐	0.000000	Single Float
00021	Char String 1	☐		Char String[40]
00041	Char String 2	☐		Char String[40]
00061	Char String 3	☐		Char String[40]
00081	Char String 4	☐		Char String[40]

3.2 Special Features of Local Registers

There are a couple of features that you may go back and change at any time after registers are created.



You may select hexadecimal display of data. This only applies to display on the local web pages, and does not change what Modbus sees externally (Modbus only sees a collection of bits, and that fact never changes). Hexadecimal display of a floating point value is probably meaningless, but hexadecimal display of registers containing a set of status bits packed into a single register is often easier to interpret when displayed as hexadecimal.

You have the option of applying a default value under two conditions: (1) Automatically at power-up; (2) Any time this register is not updated by some remote source within the timeout given (this assumes the Pro gateway is acting as a slave or server).

If this register is being used to hold a value provided by some other device acting as master, and you want a way of externally detecting when that device has failed to communicate, you can cause a "flag" value to appear in this register when a new value has not been received within the given timeout period.

The other non-error use for setting the default value (typically zero) after a timeout is when this register is used to hold the results of certain types of SNMP table walks. This use is discussed in the section describing SNMP table walking.

Applying a default at power-up is potentially useful if you want this gateway to always write a certain value to some other Modbus slave any time the system wakes up.

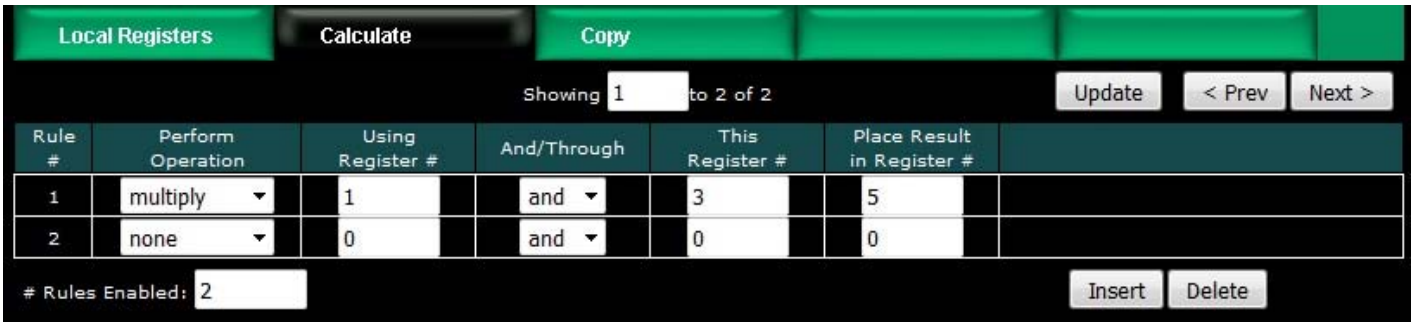
The other use for default at power-up is when you need to use a constant value in a calculate rule. Set aside a register whose only purpose is to hold this constant for later use.

3.3 Local Register Calculate Rules

The Babel Buster Pro includes the ability to do simple calculations based on simple template rules. Select the operation, one or two operands as applicable, and a register to place the result in. Operations like "multiply" will use registers A "and" B. Operations like "sum" can add up the contents of a series of registers by selecting "thru" instead of "and".

These template rules can be useful for doing minor data manipulation or testing for purposes of enabling rules, or for generating SNMP traps. If you need to do something more complex than what these simple rules will accomplish, you have the option of writing a program in Script Basic that can get as complicated as you like. Script Basic programs have the ability to both read and write local registers, which means your program can look at data values, and set registers that in turn cause things like SNMP traps to occur.

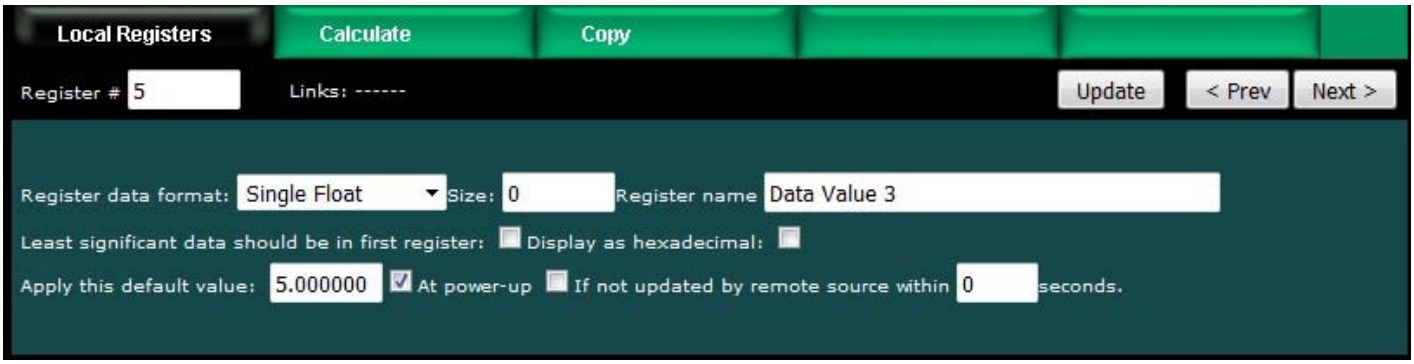
Here is an example of a template rule that multiplies register 1 by register 3 and places the result in register 5.



If registers 1 and 3 contain the values shown below, then the result shown in register 5 would be expected.

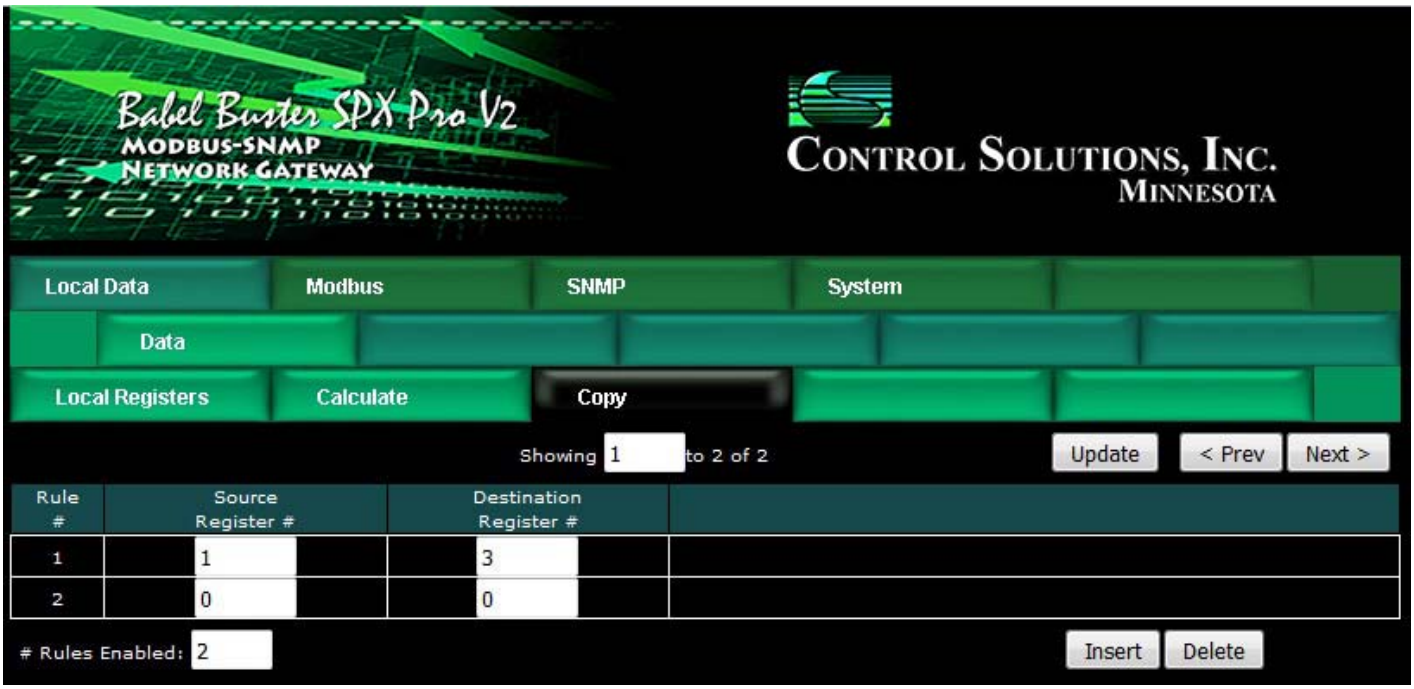
Constants may be introduced into the calculation by reserving a register to hold that constant, and then configuring it to apply a default value at power-up. This default value should be the constant you wish to include in a calculation.

The tests available are only compared to zero. If you need to compare to some other number, create a constant register, perform a subtraction of the test register minus the constant register, storing the result in some other result register. Then use the test for greater than or less than zero applied to the result register to determine how the test register compared to the constant. (Internally, computers can only test positive or negative, so these are the steps the computer is going through any time your program compares a variable to a constant. In higher level programming languages, these explicit steps are simply hidden from you.)



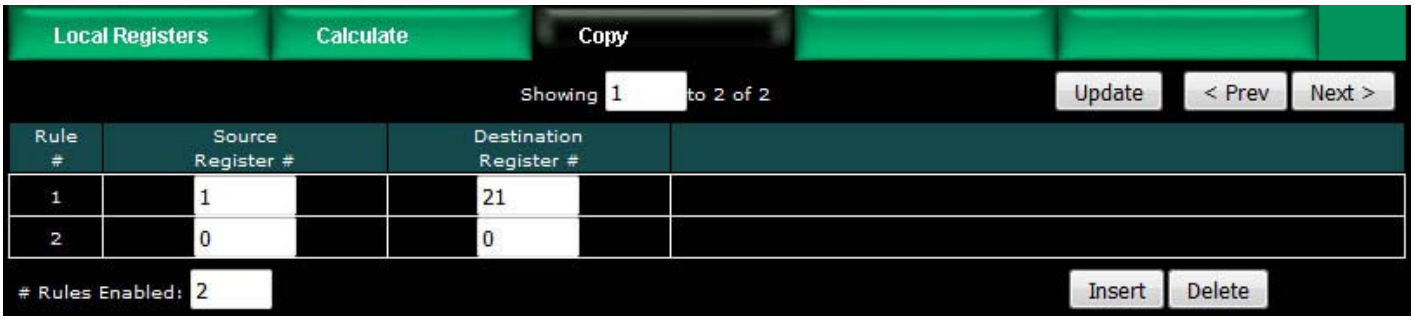
3.4 Local Register Copy Rules

The copy rules provide a means of simply copying the content of one register to another.



The above rule would cause the following data copy to occur:

Here, however, is a much more interesting use of the copy rule:



The above rule would cause the following copy to happen. Note that "copy" also means data reformatting. Therefore, if you need to convert a number to a character string (or vice versa), simply copy it from one register to the other. In this example, our copy rule is converting floating point to an ASCII string ready to be sent to a display device.

String conversion is not the only conversion you can do. If you need to convert floating point to integer, or vice versa, the copy rule will also do that. Note, however, that if you need to read an integer from a remote Modbus slave but want the result stored locally as floating point, you can do that conversion as part of the read map and do not need a separate copy rule to accomplish that conversion.

One more note about string conversion: If you do not like the appearance of numeric strings provided by the automatic conversion, you have full control over formatting if you use Script Basic to do the string conversion. Script Basic would also allow you to add text to the result, so you may end up with something like "Tank Level: 46 ft." instead of the "45.980000" in this example.

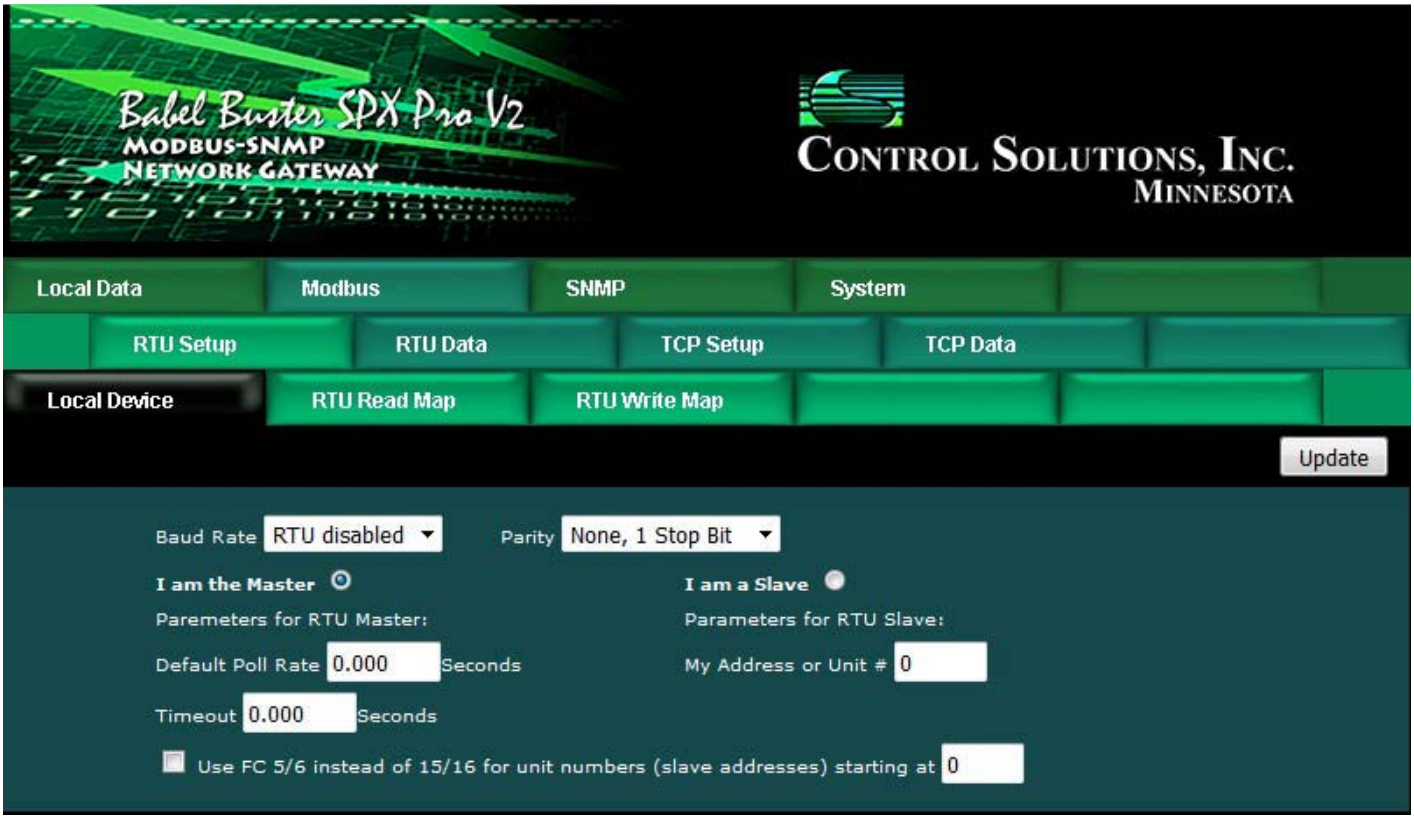
4. Configuring Gateway as a Modbus RTU Master

The SPX Pro V2 can be a Modbus RTU master or slave. As a master you can read Modbus data from, or write Modbus data to, other Modbus slaves. The gateway will periodically poll the other Modbus devices according to register maps you set up. To read from a remote Modbus device, configure a Read Map. To write to a remote Modbus device, configure a Write Map.

Data read from a remote device is stored in a local register when received. Data written to a remote device is taken from a local register when sent. The local registers are the same collection of registers that are accessible to other masters when operating as slave, and accessible to other Modbus TCP devices as a collection of holding registers. The local registers are also accessible as SNMP MIB variables.

4.1 Modbus RTU Device Configuration

Modbus device configuration for RTU really consists of just port configuration. When brand new out of the box, the RTU port defaults to disabled. When disabled, Script Basic has access to the RS485 port for serial communication with non-Modbus devices.

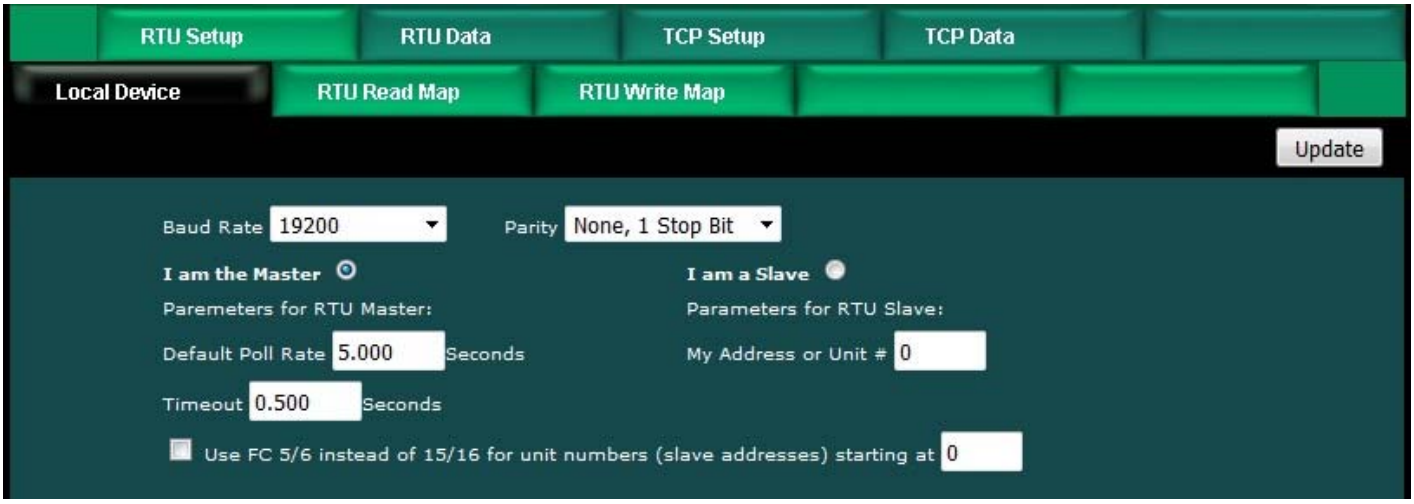


Select baud rate and parity from the drop down list. Click either Master or Slave buttons to select type of operation. Enter timing parameters or address as applicable. Click update to register your changes.

The default poll rate entered here will be used for all Modbus RTU Read and Write maps unless a different number is entered in the expanded view of the map.

IMPORTANT: Set timeout to something long enough for the device. If too short, the gateway will not wait long enough for a response from the Modbus slave device, and the result will be a lot of "no response" errors from the device even though the device is perfectly functional.

If your slave/server device only supports function codes 5 and 6 for writing coils and holding registers, check the Use FC 5/6 box. The default function codes are 15 and 16, which are most widely used. If you check the box, you should also enter a "starting at" unit # or slave address. This allows supporting both types of devices at the same time provided you assign slave addresses in two non-overlapping groups. (These settings do not apply if the gateway is the slave.)

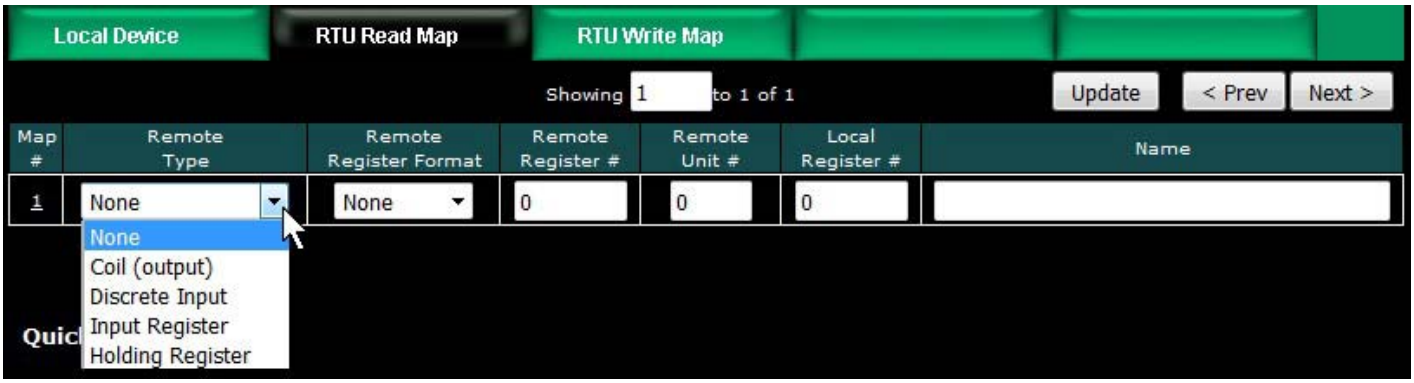


4.2 Modbus RTU Master Read Maps

Getting the gateway to read registers from another Modbus device requires setting up a "Read Map" as shown here.

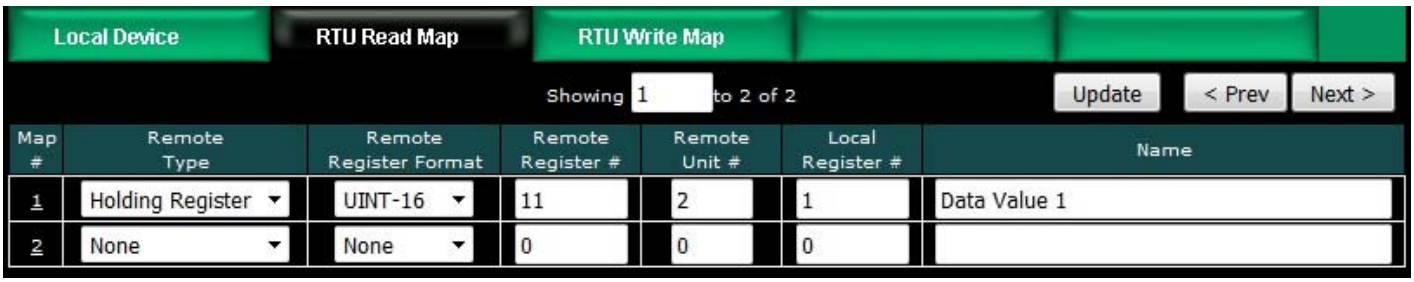
Rule number simply tells you where you're at on the list of register maps. Click "next" and "prev" to scroll through the list. To advance directly to a specific map, enter the desired number in the "Showing" box, then click Update.

Rules entered on this page only read data from remote devices. Go to the RTU Write Map to write data to those devices. The full parameter set is different for read versus write.



To create a Read Map, start by selecting the Modbus register type to read from the drop-down list.

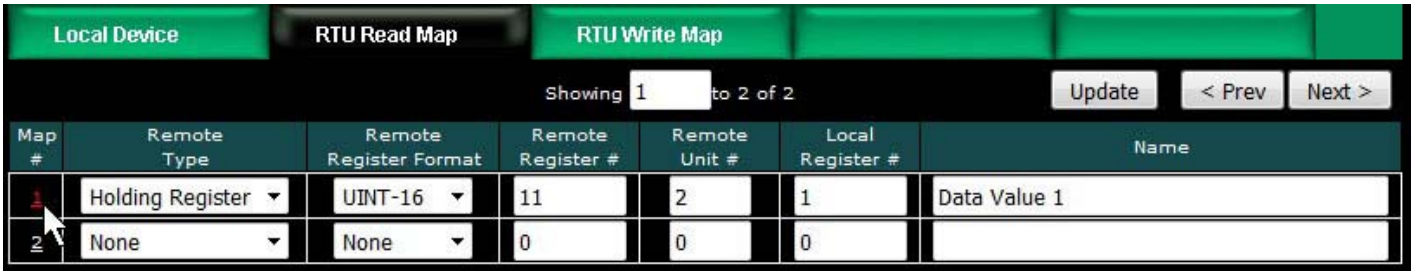
Select the data format expected in the remote Modbus register. The abbreviation INT under Format means signed integer, while UINT represents unsigned integer. The INT and UINT are followed by the number of bits to be read (which translates into 1, 2, or 4 consecutive holding registers). The FLOAT format refers to 32-bit IEEE 754 format while DOUBLE refers to 64-bit IEEE 754 floating point. The MOD10 format is unique to Schneider Electric power meters, and is supported in 2, 3, and 4-register formats. (Note: Use INT-16 or UINT-16 for coils or discrete inputs - in this case format only affects local register data conversion.)



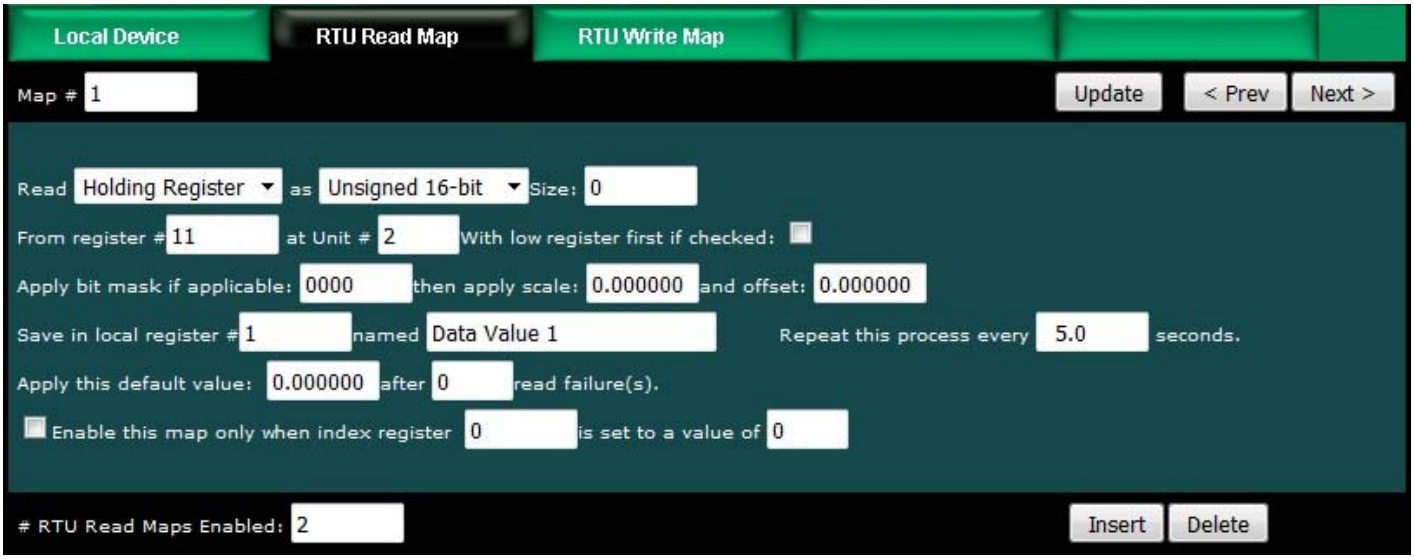
Enter the register number to read from the remote RTU slave and slave address of that RTU slave. Do not use Modicon numbers here. In other words, if your slave

device's documentation says read register 40001, that is short hand (Modicon notation) for saying read holding register 1. Refer to the Modbus Reference Information section of this user guide for more discussion about register numbers like 40001. If you enter 40001 here to read the first holding register, you will get an exception error since the actual register number is not 40001.

The Local Register is where data read from the remote Modbus RTU slave will be stored locally in the Babel Buster Pro. If the local register data format does not match what you are reading from the Modbus slave, the data will be converted automatically when it is read.



Click on the Map number in the first column to access the expanded view of the Read Map.



For each remote register to be read, select the register type, format, number, and remote unit (slave address). The optional bit mask and scaling are discussed with examples below.

Modbus protocol treats all input registers or holding registers as strictly 16-bit registers. To accommodate 32-bit or longer data, Modbus devices use multiple consecutive "registers" to hold the data. There is no standardization of whether the least significant part of the data comes first or last. Therefore, Babel Buster lets you set that according to whatever the slave device requires. If the least significant data is found in the first (or lower numbered) register in your slave device, then check the box after "With low register first".

The poll rate ("Repeat this process...") determines how often the remote register will be read. If zero is entered here, the rate will become the default poll rate given on the Devices page for the Modbus RTU port.

The default value will be stored into the local register after the given number of read failures if the fail count is non-zero. Setting the count to zero will disable the default, and the object will retain the most recent value obtained. If the default value does take effect, the actual data value read will be retained when communications are restored.

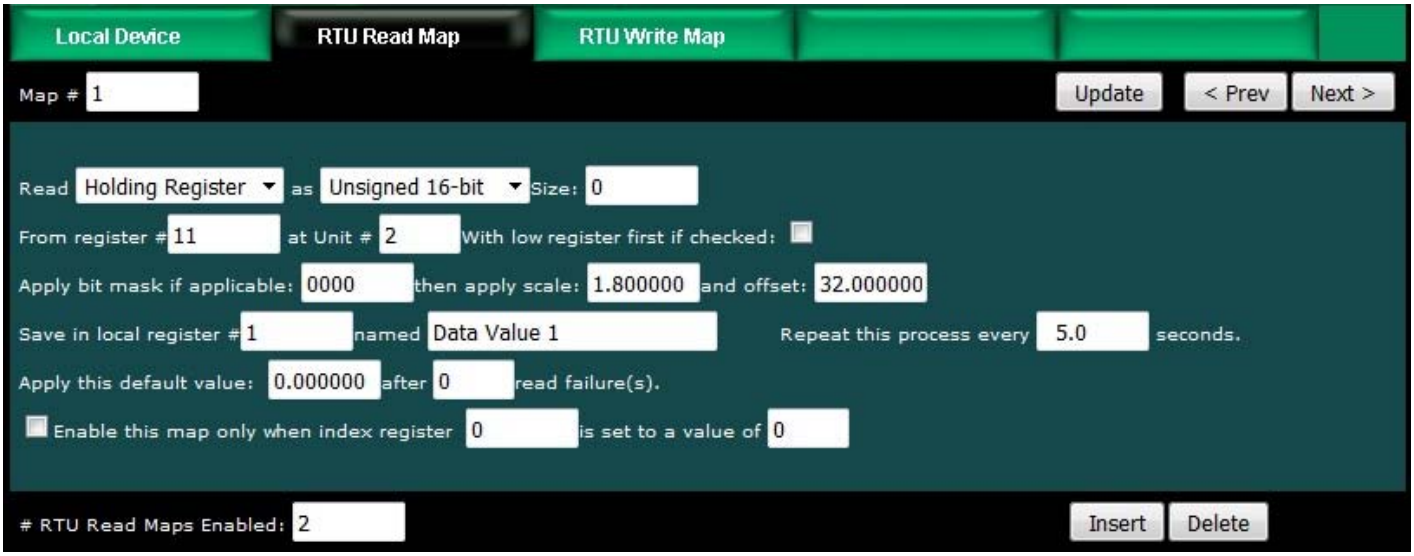
You have the option of making this Read Map conditional. If an index register number is provided and the Enable box is checked, then this read map will only be executed when the index register (local register) contains the value given. This allows multiple read maps to supply data to the same local register based on the value of the index register. It also allows reading to simply be suspended if a single read map supplies data to the local register. In a more sophisticated scenario, you could potentially suspend reading of the slave if you know the slave is powered down.

Rule number simply tells you where you're at on the list of register maps. Click "next" and "prev" to scroll through the list. To advance directly to a specific map, enter the desired number in the "Map #" box, then click Update.

Delete will remove the rule number shown in the "Map #" box. Insert will insert a new rule before the rule number shown, and is used for placing rules between existing rules. It is not necessary to use Insert to add rules to the bottom of the list or to define any rule presently having zero for a source object or "none" for remote type.

Selecting "none" for remote type effectively deletes the rule even though it will still appear in the list until deleted. Unused rules at the end of the list will always show none as the type. If you wish to prevent these from being displayed, reduce the number of rules enabled.

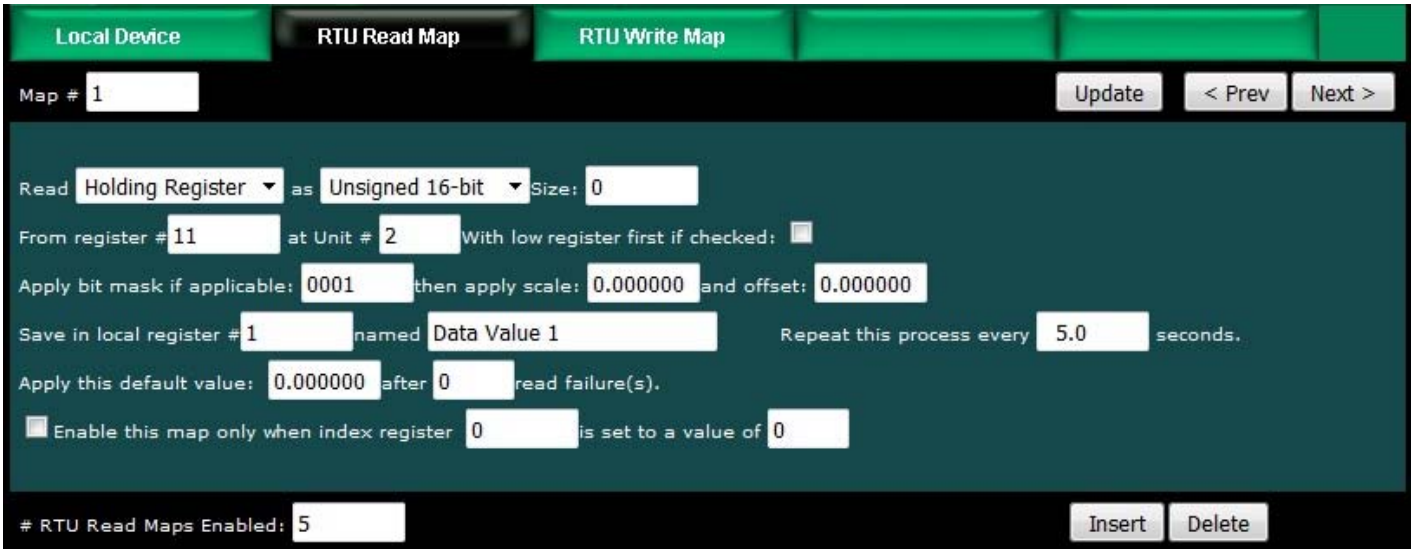
The number of rules enabled simply limits the scope of rule review so that you do not have to review a lot of unused rules. If the displayed rules are used up and you need more, increase the enabled number.



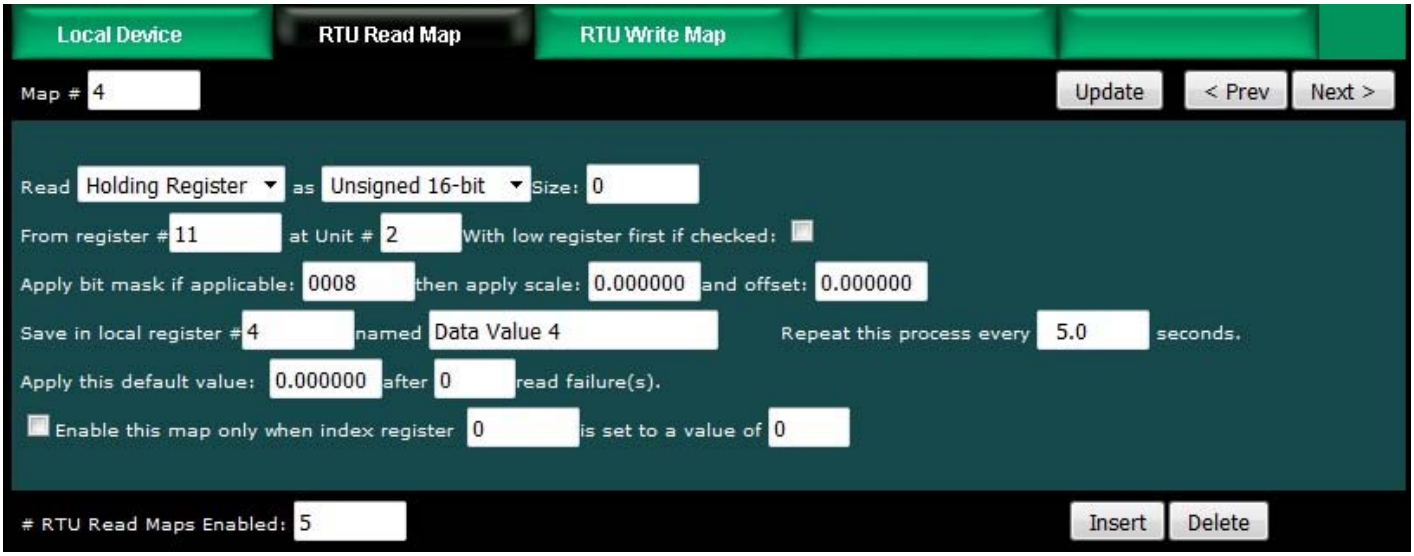
You have the option of providing a scale and offset. A scale of zero will cause scale and offset to be ignored. If provided, the Modbus data will be treated as raw data. When the Modbus data is received, it will be multipled by scale, then added to offset, and then stored in the local register. If the Modbus slave was providing degrees Celsius, and the scale factors illustrated above were used, then a Modbus value of 25 would result in the local register receiving a value of 77 (degrees Fahrenheit).

It is common for Modbus devices to pack a number of status bits into a single holding register. In order to do meaningful things based on a single bit, such as generate an SNMP trap upon alarm indicated by one of those bits, it is necessary to split that register into multiple local registers. Babel Buster supports this requirement by providing an optional bit mask.

If a bit mask is entered (in hexadecimal), and the remote register type is signed or unsigned integer (16-bit or 32-bit data), the mask will be bit-wise logical AND-ed with the Modbus data, and the retained bits will be right justified in the result stored locally.



Reading bit 0 is illustrated above while reading bit 3 is illustrated below (bit 3 mask in binary would be 1000 which is hexadecimal 8 as entered in the configuration page). Refer to the Modbus Reference section in this user guide for a list of all possible mask values.



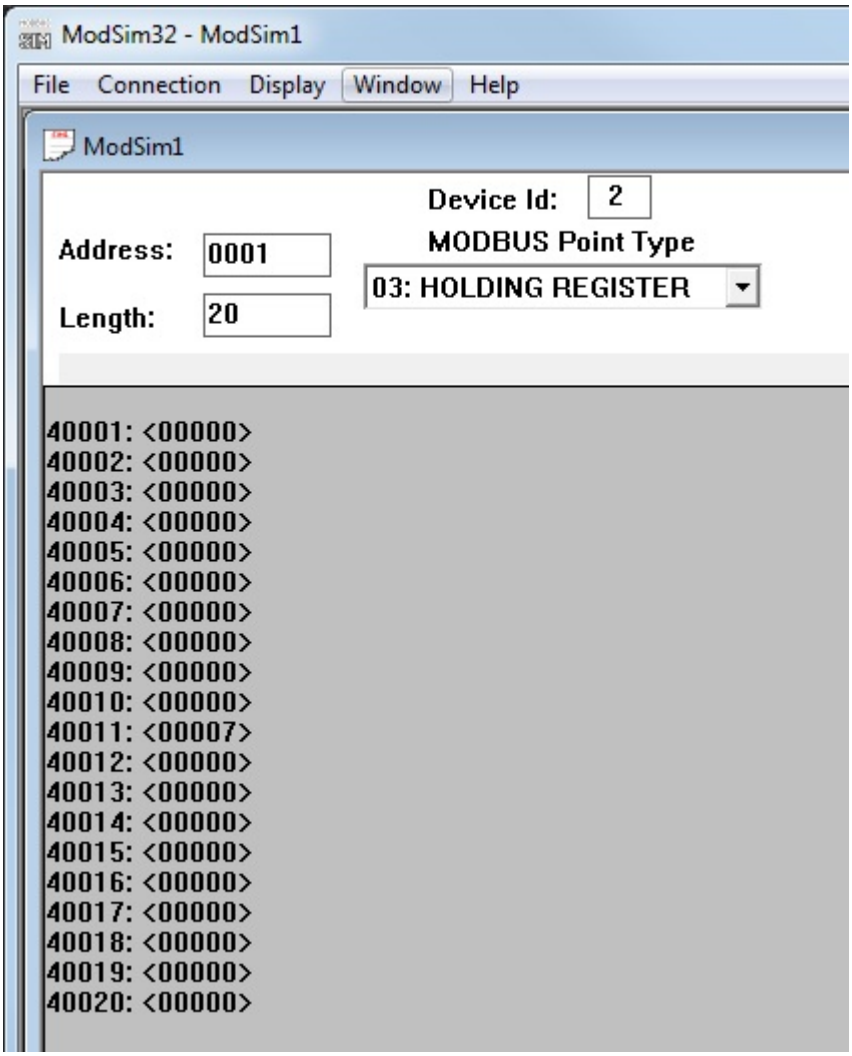
If the read maps referencing the same remote register are created in sequential contiguous order, the Babel Buster will optimize the RTU activity by reading the remote register once and then sharing the data with all of the read maps in the group. In the example illustrated here, four consecutive read maps reference the same remote register, each selecting a different bit. The first rule is selecting bit 0, the second selecting bit 1, and so on.

Local Device | RTU Read Map | RTU Write Map

Showing 1 to 5 of 5 Update < Prev Next >

Map #	Remote Type	Remote Register Format	Remote Register #	Remote Unit #	Local Register #	Name
1	Holding Register	UINT-16	11	2	1	Data Value 1
2	Holding Register	UINT-16	11	2	2	Data Value 2
3	Holding Register	UINT-16	11	2	3	Data Value 3
4	Holding Register	UINT-16	11	2	4	Data Value 4
5	None	None	0	0	0	

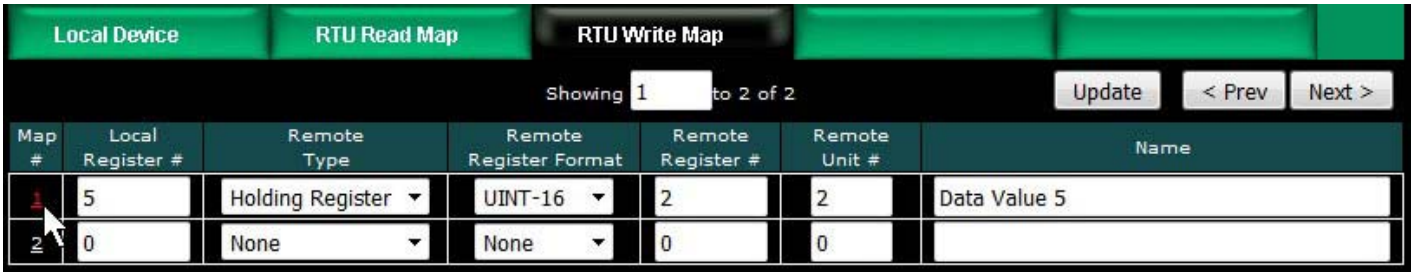
To try out these read maps, we have set up ModSim with the remote register 11 containing a value of 7.



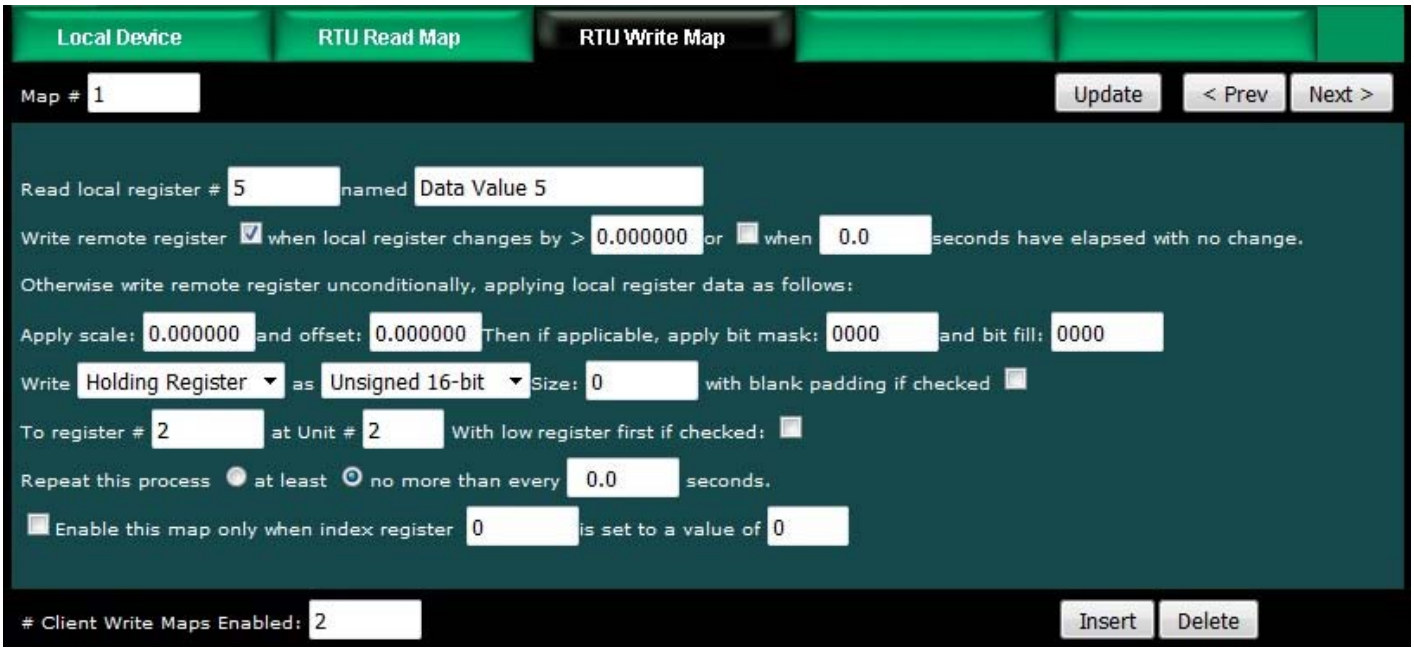
The holding register value of 7 translates into the following local register values when the bit mask option is used as illustrated.

4.3 Modbus RTU Master Write Maps

Getting the gateway to write registers to another Modbus device requires setting up a "Write Map" as shown here. Much of the Write Map is configured the same as a Read Map.



The data direction is reversed but the same selections are still made. Select the local register that will be the source of data to write to the remote Modbus RTU slave. Select the register type, data format, and register number to be written to in that slave. Enter the slave address as Remote Unit. Click on the map number in the first column to access additional optional configuratino parameters.



The local register data may be written to the remote slave periodically, or when it changes, or both. To send upon change (send on delta), check the first box and enter the amount by which the local register must change before being written to the remote device. To guarantee that the remote register will be written at least occasionally even if the data does not change, check the second box and enter some amount of time. This time period will be referred to as the "maximum quiet time".

Data from the local register may be manipulated before being written to the remote register. The local data is first multiplied by the scale factor. The offset is then added to it. If a bit mask is entered, and the remote register type is signed or unsigned integer (16-bit or 32-bit data), the mask will be bit-wise logical AND-ed with the data. The mask is right justified, then AND-ed with the data. The result is then left shifted back to the original position of the mask. In other words, the least significant bits of the original data will be stuffed at the position marked by the mask.

After the scaling and masking, the bit fill will be logically OR-ed into the result, but only if the mask was nonzero and was used. Both mask and fill are entered in hexadecimal. The effect of "fill" is that certain bits will always be set to 1 in the data written to the remote Modbus device.

Multiple local registers may be packed into a single remote register. To accomplish this, define two or more rules in sequence with the same remote destination. If the destination is the same, data types are 16 or 32-bit integer (signed or unsigned), bit masks are nonzero, and the rules are sequential, the results of all qualifying rules will be OR-ed together before being sent to the remote destination.

For the remote register to be written, select the register type, format, number, and remote unit (slave address). Data formats are the same as described above for Read Maps. Size is only specified for character strings. Use INT-16 or UINT-16 data format for coils - in this case format only affects local register data conversion.

Modbus protocol treats all holding registers as strictly 16-bit registers. To accommodate 32-bit or longer data, Modbus devices use multiple consecutive "registers" to hold the data. There is no standardization of whether the least significant part of the data comes first or last. Therefore, Babel Buster lets you set that according to whatever the slave device requires. If the least significant data is found in the first (or lower numbered) register in your slave device, then check the box after "With low register first".

The repeat time may determine how often the remote register will be written. If send on delta and maximum quiet time are not checked above, clicking the "at least" button will establish a periodic update time. If send on delta is used and you wish to limit the network traffic when changes are too frequent, click the "no more than" button and enter the minumum time that should elapse before allowing another write to the remote device. It is valid to select "no more than every 0.0 seconds" if you want all changes to be sent, but no periodic writes.

You have the option of making this Write Map conditional. If an index register number

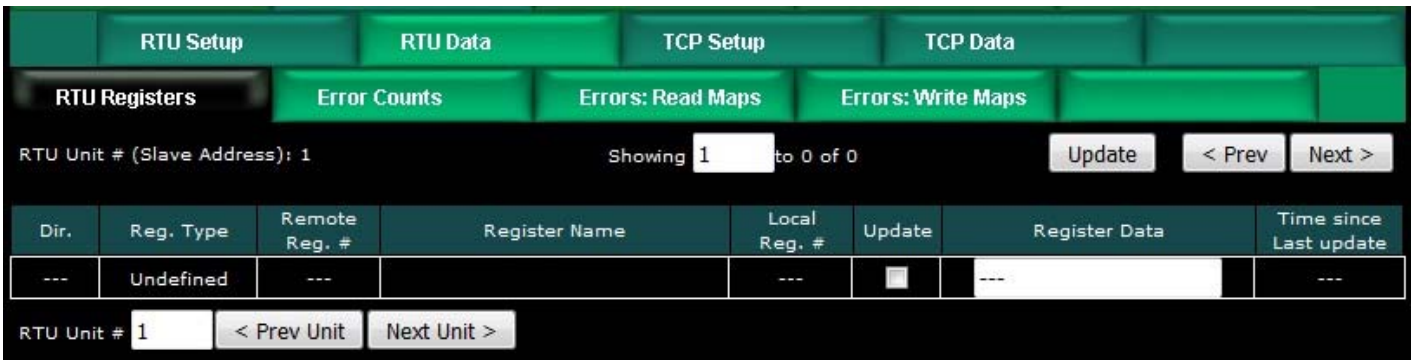
is provided and the Enable box is checked, then this write map will only be executed when the index register (local register) contains the value given. This allows multiple write maps to supply data to the same remote register based on the value of the local index register. It also allows writing to simply be suspended if a single write map supplies data to the remote register. In a more sophisticated scenario, you could potentially suspend writing of the slave if you know the slave is powered down.

Delete will remove the rule number shown in the "Map #" box. Insert will insert a new rule before the rule number shown, and is used for placing rules between existing rules. It is not necessary to use Insert to add rules to the bottom of the list or to define any rule presently having zero for a source register or "none" for remote type.

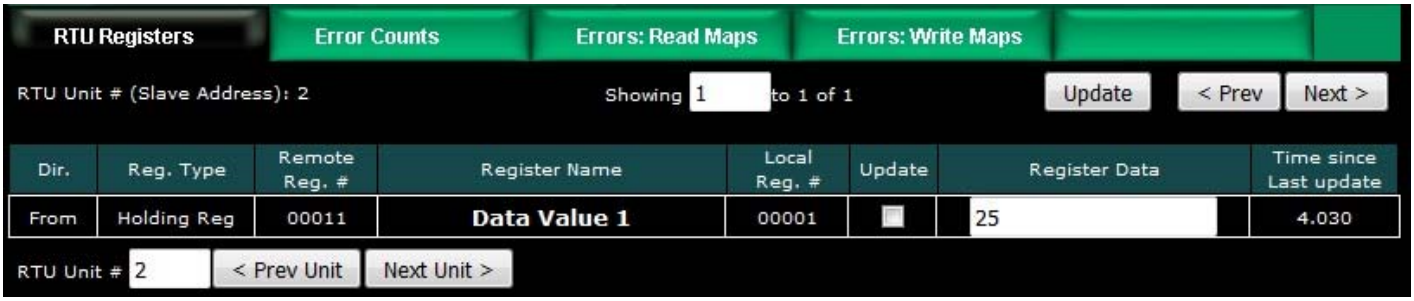
Selecting "none" for remote type effectively deletes the rule even though it will still appear in the list until deleted. Unused rules at the end of the list will always show none as the type. If you wish to prevent these from being displayed, reduce the number of rules enabled.

4.4 Modbus RTU Master Data Displayed by Slave

The RTU Registers page shows a list of local registers mapped to RTU slave devices. The page will show only one device at a time, and may have no entries as illustrated below.

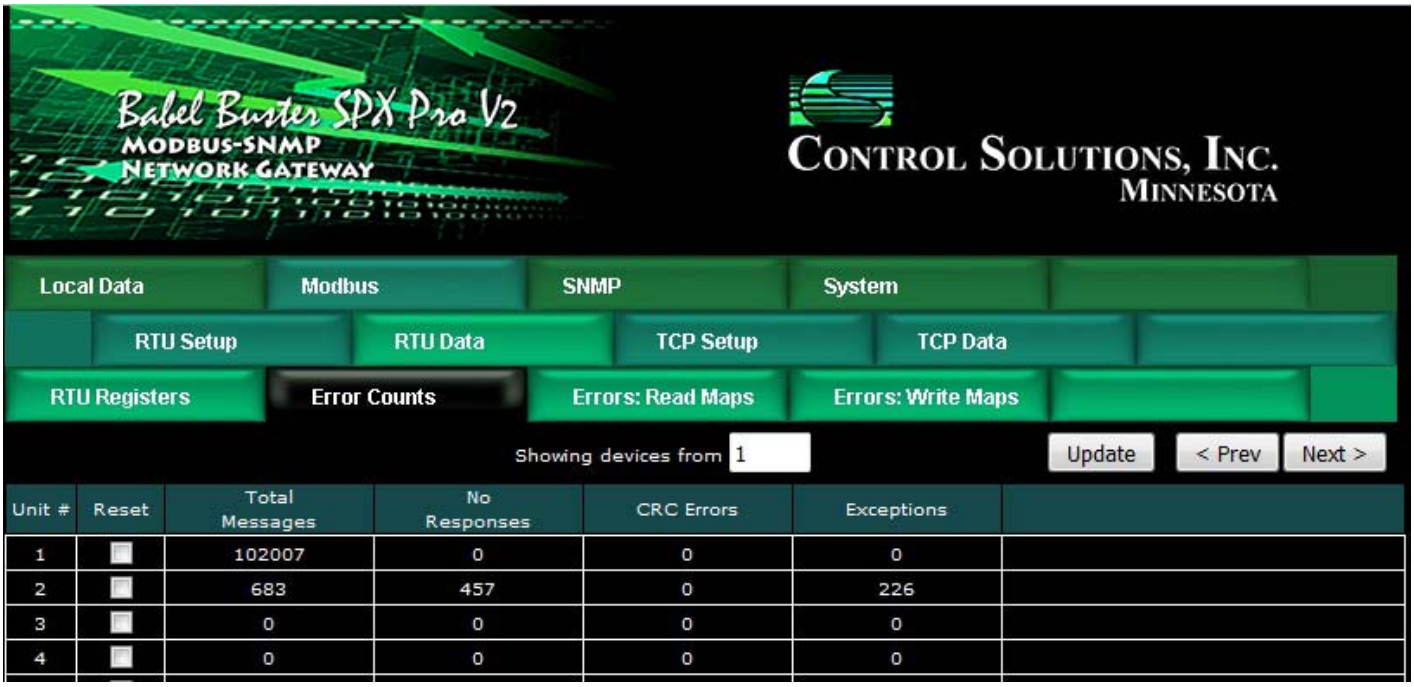


Click the Next Unit button to go to the next RTU slave, or simply enter a number in the RTU Unit # window and click Update. Registers for that slave will now be displayed. In addition to a summary of the map (both read and write maps are shown), the time since last update is displayed. This time should generally be less than the poll time. If the last update time is large, it means there is an error preventing the update.

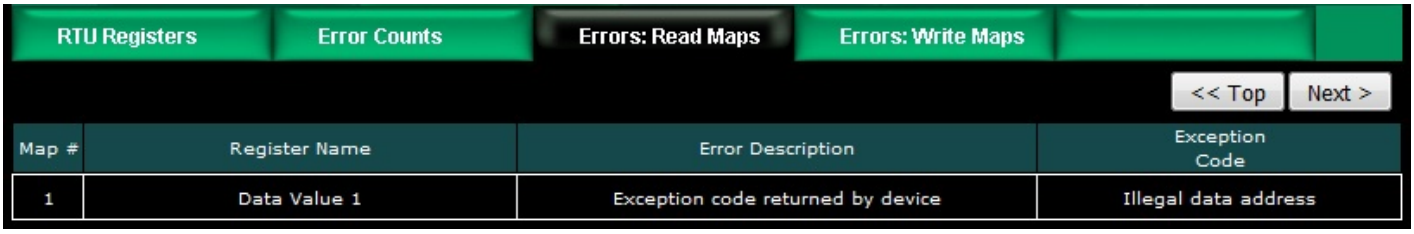


4.5 Modbus RTU Errors

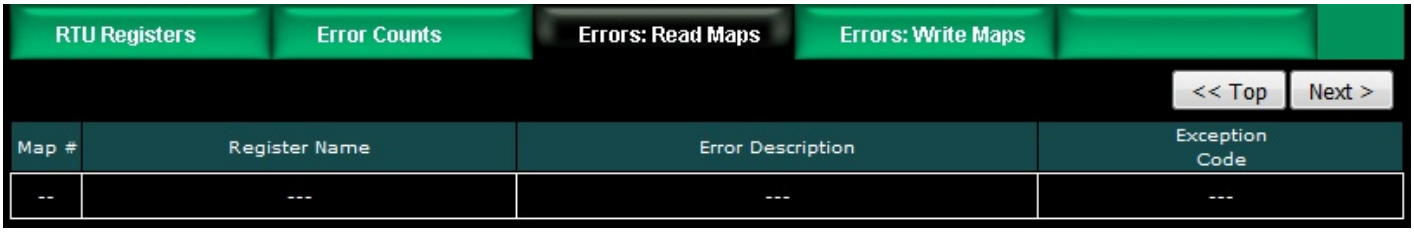
The Error Counts page shows a tabulation, by RTU slave, of all errors observed. In the example below, we can see that the RTU device at slave address 1 is running flawlessly while slave #2 has had some issues.



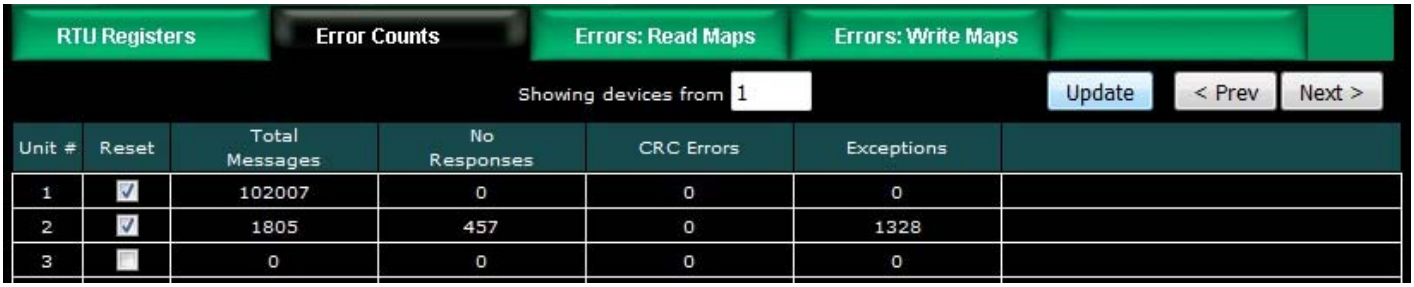
If the counts show some problems, we can look for more detail on the Errors: Read Maps (or Errors: Write Maps) pages. These pages will tell us exactly which Read Map (or Write Map) the problem is occurring on, and what the error is, as illustrated below.



When the problem is resolved, it will be removed from this list.

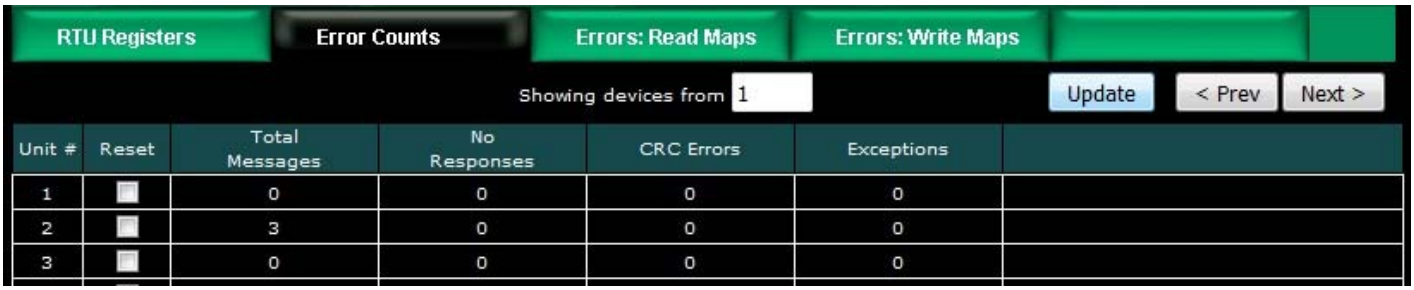


Once you have resolved problems, you can reset the error counts by checking the box in the Reset column and then clicking Update.



Unit #	Reset	Total Messages	No Responses	CRC Errors	Exceptions
1	☑	102007	0	0	0
2	☑	1805	457	0	1328
3	☐	0	0	0	0

The counts will reset to zero, but in most cases, at least "Total Messages" will start incrementing again.

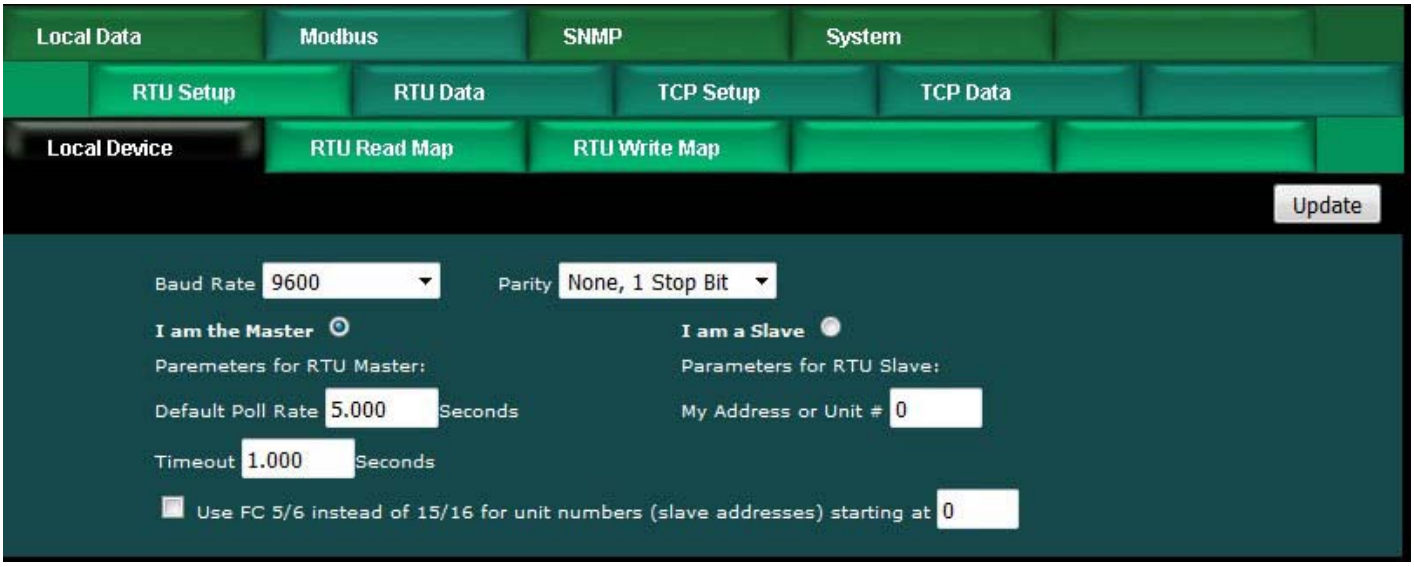


Unit #	Reset	Total Messages	No Responses	CRC Errors	Exceptions
1	☐	0	0	0	0
2	☐	3	0	0	0
3	☐	0	0	0	0

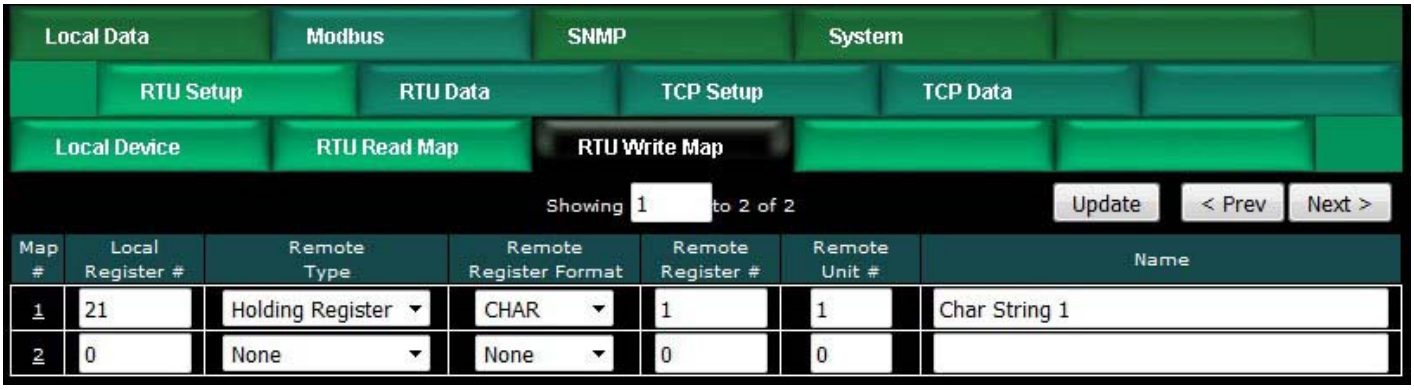
4.6 Character String Example

An SNMP Trap Receiver example is shown later in this user guide which receives a trap message into a character string. Here we will take a close look at configuring a Write Map to send that trap message to an alphanumeric display.

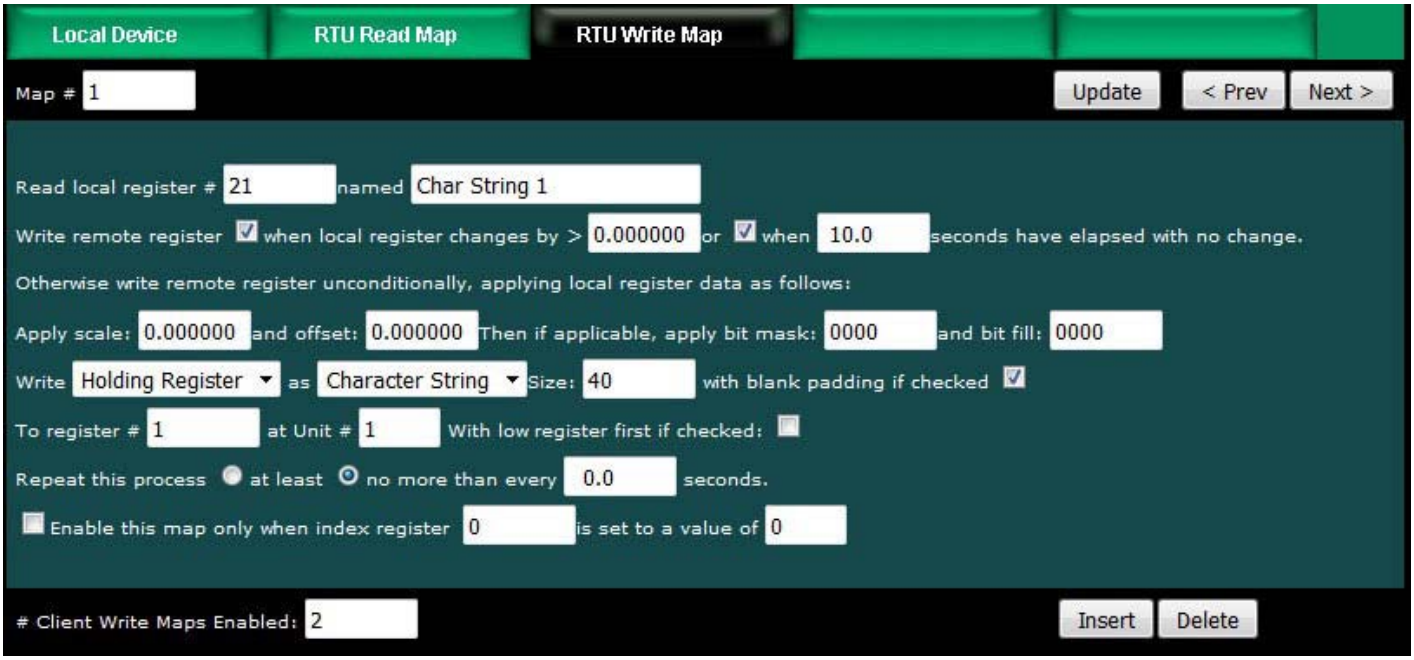
Start by configuring the local device to be Modbus RTU master, at whatever baud rate matches the display device.



Create the Write Map. Select the local register to be written. Select the remote register type and number to be written. For an alphanumeric display, this will be a holding register, and the display device's documentation should tell you what register number represents the start of each line.



You need to click on map number in the first column to get to the expanded view of the Write Map so that you can enter the number of characters to be written. Note that this is character count, not register count. There will two characters per Modbus register, with the left most character in the upper 8 bits of the 16-bit register. The example below will write 40 characters, or 20 Modbus holding registers.



The SNMP trap receiver has received the message illustrated below.

Local Registers | Calculate | Copy

Showing registers from 1 Update < Prev Next >

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	☐	0.000000	Single Float
00003	Data Value 2	☐	0.000000	Single Float
00005	Data Value 3	☐	0.000000	Single Float
00007	Data Value 4	☐	0.000000	Single Float
00009	Data Value 5	☐	0.000000	Single Float
00011	Data Value 6	☐	0.000000	Single Float
00013	Data Value 7	☐	0.000000	Single Float
00015	Data Value 8	☐	0.000000	Single Float
00017	Data Value 9	☐	0.000000	Single Float
00019	Data Value 10	☐	0.000000	Single Float
00021	Char String 1	☐	UPS: No longer on battery power.	Char String[40]
00041	Char String 2	☐		Char String[40]
00061	Char String 3	☐		Char String[40]
00081	Char String 4	☐		Char String[40]
00101	Unsigned Value 1	☐	00000000	Unsigned 32-bit

The RTU Write Map causes the display to be updated as illustrated here.

5. Configuring Gateway as a Modbus TCP Client

The SPX Pro V2 can be a Modbus TCP client and server. The terms client and server are more often used with Ethernet network devices, but for Modbus purposes, they still mean master and slave respectively. You must choose one or the other between master and slave for Modbus RTU, but Modbus TCP can be both simultaneously thanks to Ethernet.

As a master (client) you can read Modbus data from, or write Modbus data to, other Modbus TCP devices. The gateway will periodically poll the other Modbus devices according to register maps you set up. To read from a remote Modbus device, configure a Read Map. To write to a remote Modbus device, configure a Write Map.

Data read from a remote device is stored in a local register when received. Data written to a remote device is taken from a local register when sent. The local registers are the same collection of registers that are accessible to other Modbus TCP clients as a collection of holding registers. The local registers are also accessible as SNMP MIB variables.

5.1 Modbus TCP Device Configuration

The Modbus TCP client is the Ethernet version of Modbus master. Modbus RTU requires a slave address. Modbus TCP also requires, in effect, a slave address. In the case of TCP, that slave address is an IP address. Since entering the IP address requires more effort than one simple slave number, and because internally a network socket is required per IP address, the TCP devices are set up in their own table.

For each Modbus TCP device that you wish to read and write, enter its IP address on the TCP Setup Devices page. The port is normally going to be 502 (the standard Modbus TCP port), but if it is different, enter that number here. You have the option of referring to a Modbus TCP device by domain name. If you use a domain name, be sure that domain can be found at the DNS servers provided on the Network setup page.

A unit number is always included in each Modbus TCP packet. This is the equivalent of the RTU slave address. Some TCP devices pay no attention to unit and simply echo back whatever you had sent. However, if you are accessing RTU devices on the other side of a TCP to RTU router (gateway), then the unit does become the RTU slave address on the RTU side of the gateway and multiple RTU devices are accessed at the same TCP IP address. (Control Solutions BB2-6010-GW or any other model with the -GW suffix would provide this type of TCP to RTU routing.)

If "Unit" on the devices page is left at zero, then unit #1 is used by default. Otherwise the number entered here becomes the default. However, each individual read and write map can have different unit numbers, as would be required for accessing multiple RTU devices via a gateway at one IP address. When a non-zero unit number is placed in the read or write map, it will override the default on the Devices page.

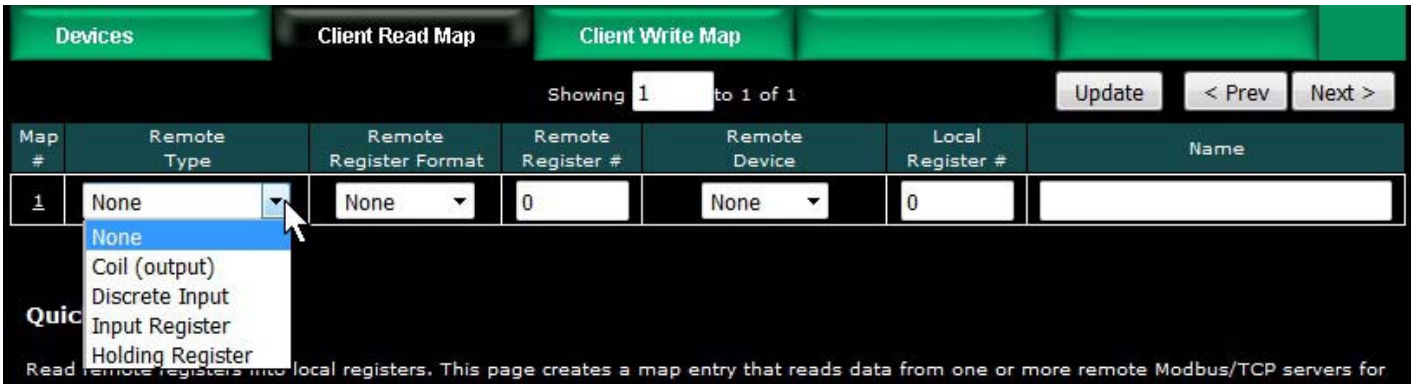
The default poll rate entered here will be used for all Modbus TCP Read and Write maps unless a different number is entered in the expanded view of the map.

5.2 Modbus TCP Client Read Maps

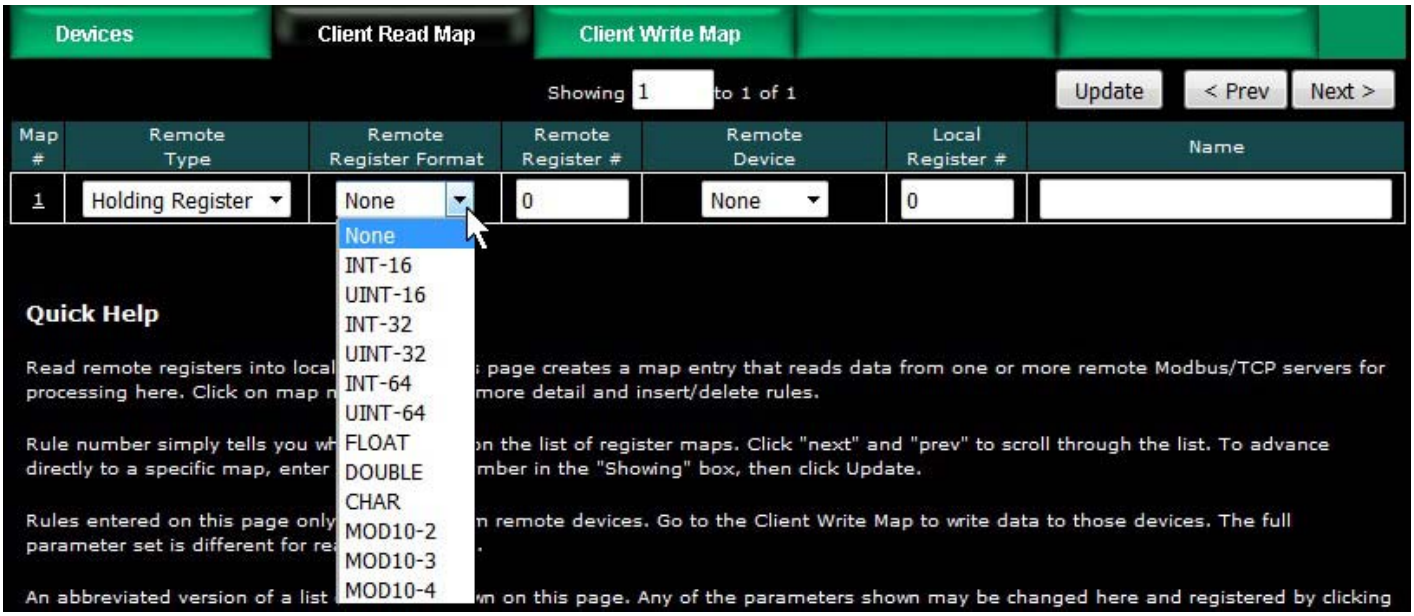
Getting the gateway to read registers from another Modbus device requires setting up a "Read Map" as shown here.

Rule number simply tells you where you're at on the list of register maps. Click "next" and "prev" to scroll through the list. To advance directly to a specific map, enter the desired number in the "Showing" box, then click Update.

Rules entered on this page only read data from remote devices. Go to the TCP Write Map to write data to those devices. The full parameter set is different for read versus write.



To create a Read Map, start by selecting the Modbus register type to read from the drop-down list.

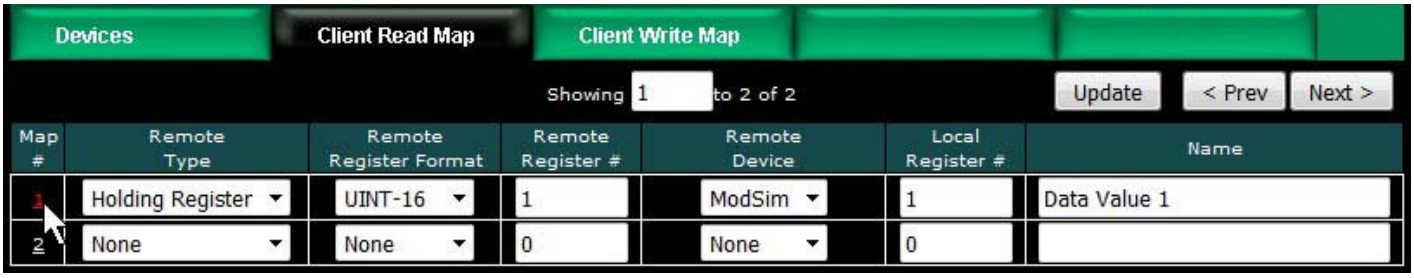


Select the data format expected in the remote Modbus register. The abbreviation INT under Format means signed integer, while UINT represents unsigned integer. The INT and UINT are followed by the number of bits to be read (which translates into 1, 2, or 4 consecutive holding registers). The FLOAT format refers to 32-bit IEEE 754 format while DOUBLE refers to 64-bit IEEE 754 floating point. The MOD10 format is unique to Schneider Electric power meters, and is supported in 2, 3, and 4-register formats. (Note: Use INT-16 or UINT-16 for coils or discrete inputs - in this case format only affects local register data conversion.)

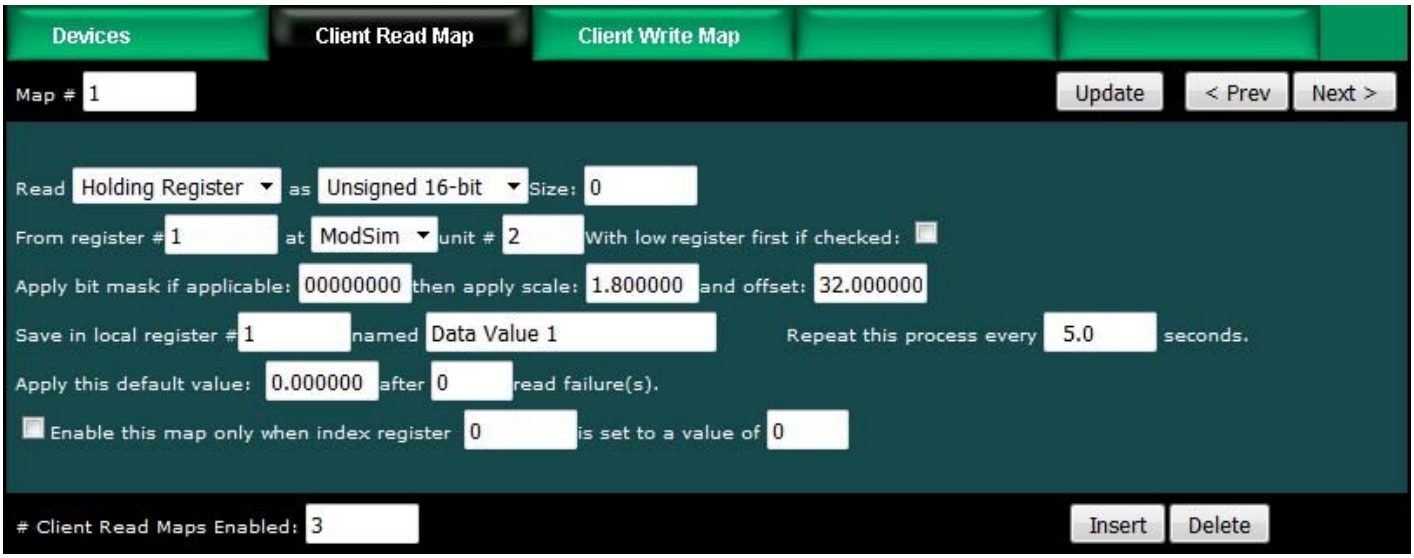
Enter the register number to read from the remote TCP server. Do not use Modicon numbers here. In other words, if your device's documentation says read register 40001, that is short hand (Modicon notation) for saying read holding register 1. Refer to the Modbus Reference Information section of this user guide for more discussion about register numbers like 40001. If you enter 40001 here to read the first holding register, you will get an exception error since the actual register number is not 40001.

Select a TCP device from the list that this register should be read from. Only devices entered on the Devices page will appear in the list.

The Local Register is where data read from the remote Modbus TCP server will be stored locally in the Babel Buster Pro. If the local register data format does not match what you are reading from the Modbus device, the data will be converted automatically when it is read.



Click on the Map number in the first column to access the expanded view of the Read Map.



For each remote register to be read, select the register type, format, number. Select a TCP server device from the list to read from. Only devices entered on the Devices page will appear here. If a unit number other than the default unit entered for this TCP server on the Devices page should be used, enter that unit number here.

The optional bit mask and scaling are discussed with examples below.

Modbus protocol treats all input registers or holding registers as strictly 16-bit registers. To accommodate 32-bit or longer data, Modbus devices use multiple consecutive "registers" to hold the data. There is no standardization of whether the least significant part of the data comes first or last. Therefore, Babel Buster lets you set that according to whatever the remote Modbus device requires. If the least significant data is found in the first (or lower numbered) register in your Modbus device, then check the box after "With low register first".

The poll rate ("Repeat this process...") determines how often the remote register will be read. If zero is entered here, the rate will become the default poll rate given on the Devices page for the Modbus TCP device selected.

The default value will be stored into the local register after the given number of read failures if the fail count is non-zero. Setting the count to zero will disable the default, and the object will retain the most recent value obtained. If the default value does take effect, the actual data value read will be retained when communications are

restored.

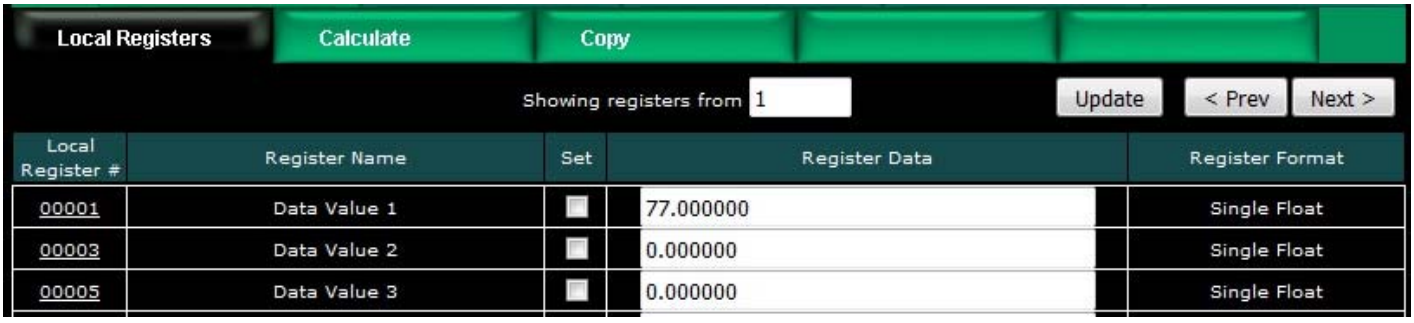
You have the option of making this Read Map conditional. If an index register number is provided and the Enable box is checked, then this read map will only be executed when the index register (local register) contains the value given. This allows multiple read maps to supply data to the same local register based on the value of the index register. It also allows reading to simply be suspended if a single read map supplies data to the local register. In a more sophisticated scenario, you could potentially suspend reading of the remote Modbus device if you know the device is powered down.

Rule number simply tells you where you're at on the list of register maps. Click "next" and "prev" to scroll through the list. To advance directly to a specific map, enter the desired number in the "Map #" box, then click Update.

Delete will remove the rule number shown in the "Map #" box. Insert will insert a new rule before the rule number shown, and is used for placing rules between existing rules. It is not necessary to use Insert to add rules to the bottom of the list or to define any rule presently having zero for a source object or "none" for remote type.

Selecting "none" for remote type effectively deletes the rule even though it will still appear in the list until deleted. Unused rules at the end of the list will always show none as the type. If you wish to prevent these from being displayed, reduce the number of rules enabled.

The number of rules enabled simply limits the scope of rule review so that you do not have to review a lot of unused rules. If the displayed rules are used up and you need more, increase the enabled number.



You have the option of providing a scale and offset. A scale of zero will cause scale and offset to be ignored. If provided, the Modbus data will be treated as raw data. When the Modbus data is received, it will be multipled by scale, then added to offset, and then stored in the local register. If the Modbus device was providing degrees Celsius, and the scale factors illustrated above were used, then a Modbus value of 25 would result in the local register receiving a value of 77 (degrees Fahrenheit).

It is common for Modbus devices to pack a number of status bits into a single holding register. In order to do meaningful things based on a single bit, such as generate an SNMP trap upon alarm indicated by one of those bits, it is necessary to split that

register into multiple local registers. Babel Buster supports this requirement by providing an optional bit mask.

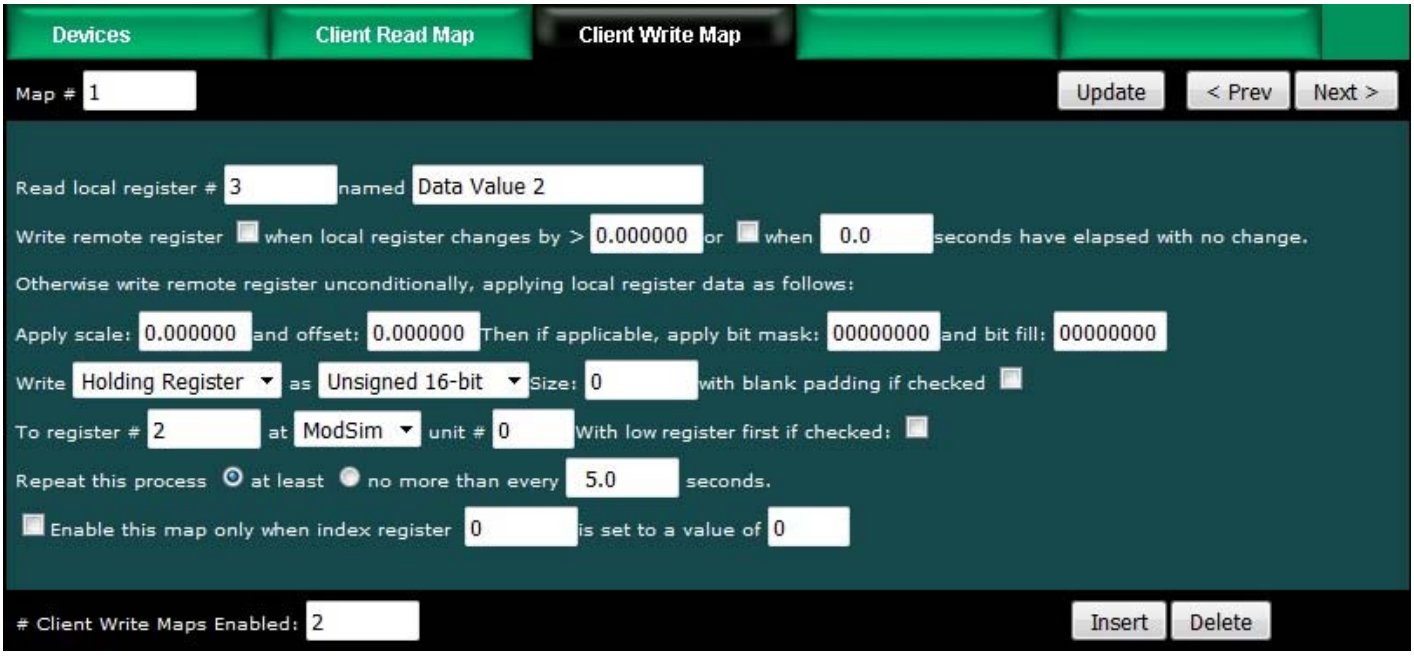
If a bit mask is entered (in hexadecimal), and the remote register type is signed or unsigned integer (16-bit or 32-bit data), the mask will be bit-wise logical AND-ed with the Modbus data, and the retained bits will be right justified in the result stored locally. Refer to the Modbus Reference section in this user guide for a list of all possible mask values.

If the read maps referencing the same remote register are created in sequential contiguous order, the Babel Buster will optimize the TCP activity by reading the remote register once and then sharing the data with all of the read maps in the group. The example illustrated for Modbus RTU in section 4.2 works exactly the same for TCP. In that example, four consecutive read maps reference the same remote register, each selecting a different bit. The first rule is selecting bit 0, the second selecting bit 1, and so on.

5.3 Modbus TCP Client Write Maps

Getting the gateway to write registers to another Modbus device requires setting up a "Write Map" as shown here. Much of the Write Map is configured the same as a Read Map.

The data direction is reversed but the same selections are still made. Select the local register that will be the source of data to write to the remote Modbus TCP device. Select the register type, data format, and register number to be written to in that device. Select a TCP server device from the list to write to. Only devices entered on the Devices page will appear here. If a unit number other than the default unit entered for this TCP server on the Devices page should be used, enter that unit number here. Click on the map number in the first column to access additional optional configuration parameters.



The local register data may be written to the remote device periodically, or when it changes, or both. To send upon change (send on delta), check the first box and enter the amount by which the local register must change before being written to the remote device. To guarantee that the remote register will be written at least occasionally even if the data does not change, check the second box and enter some amount of time. This time period will be referred to as the "maximum quiet time".

Data from the local register may be manipulated before being written to the remote register. The local data is first multiplied by the scale factor. The offset is then added to it. If a bit mask is entered, and the remote register type is signed or unsigned integer (16-bit or 32-bit data), the mask will be bit-wise logical AND-ed with the data. The mask is right justified, then AND-ed with the data. The result is then left shifted back to the original position of the mask. In other words, the least significant bits of the original data will be stuffed at the position marked by the mask.

After the scaling and masking, the bit fill will be logically OR-ed into the result, but only if the mask was nonzero and was used. Both mask and fill are entered in hexadecimal. The effect of "fill" is that certain bits will always be set to 1 in the data written to the remote Modbus device.

Multiple local registers may be packed into a single remote register. To accomplish this, define two or more rules in sequence with the same remote destination. If the destination is the same, data types are 16 or 32-bit integer (signed or unsigned), bit masks are nonzero, and the rules are sequential, the results of all qualifying rules will be OR-ed together before being sent to the remote destination.

For the remote register to be written, select the register type, format, number, select a remote TCP device from the list, and enter a unit number if the default unit number for that device should not be used. Data formats are the same as described above for Read Maps. Size is only specified for character strings. Use INT-16 or UINT-16 data

format for coils - in this case format only affects local register data conversion.

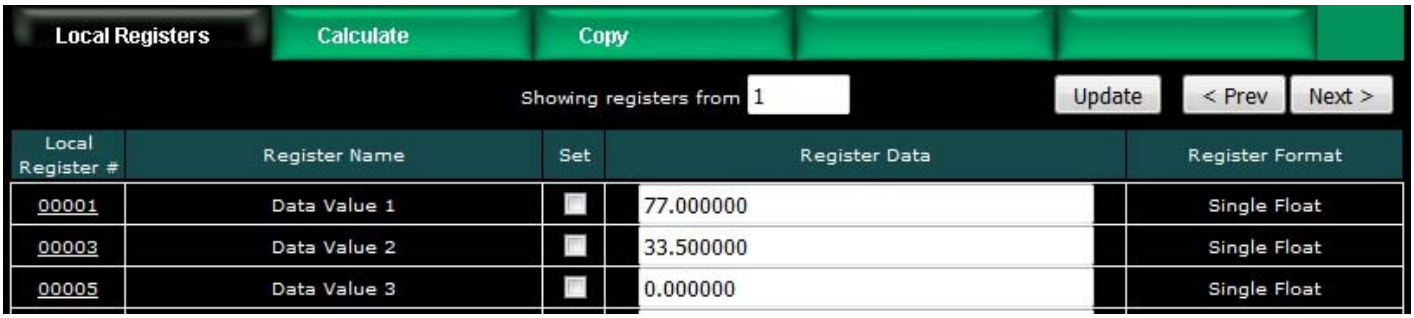
Modbus protocol treats all holding registers as strictly 16-bit registers. To accommodate 32-bit or longer data, Modbus devices use multiple consecutive "registers" to hold the data. There is no standardization of whether the least significant part of the data comes first or last. Therefore, Babel Buster lets you set that according to whatever the remote Modbus device requires. If the least significant data is found in the first (or lower numbered) register in your Modbus device, then check the box after "With low register first".

The repeat time may determine how often the remote register will be written. If send on delta and maximum quiet time are not checked above, clicking the "at least" button will establish a periodic update time. If send on delta is used and you wish to limit the network traffic in the event changes are frequent, click the "no more than" button and enter the minumum time that should elapse before another write to the remote device. It is valid to select "no more than every 0.0 seconds" if you want all changes to be sent, but no periodic writes.

You have the option of making this Write Map conditional. If an index register number is provided and the Enable box is checked, then this write map will only be executed when the index register (local register) contains the value given. This allows multiple write maps to supply data to the same remote register based on the value of the local index register. It also allows writing to simply be suspended if a single write map supplies data to the remote register. In a more sophisticated scenario, you could potentially suspend writing of the remote device if you know the device is powered down.

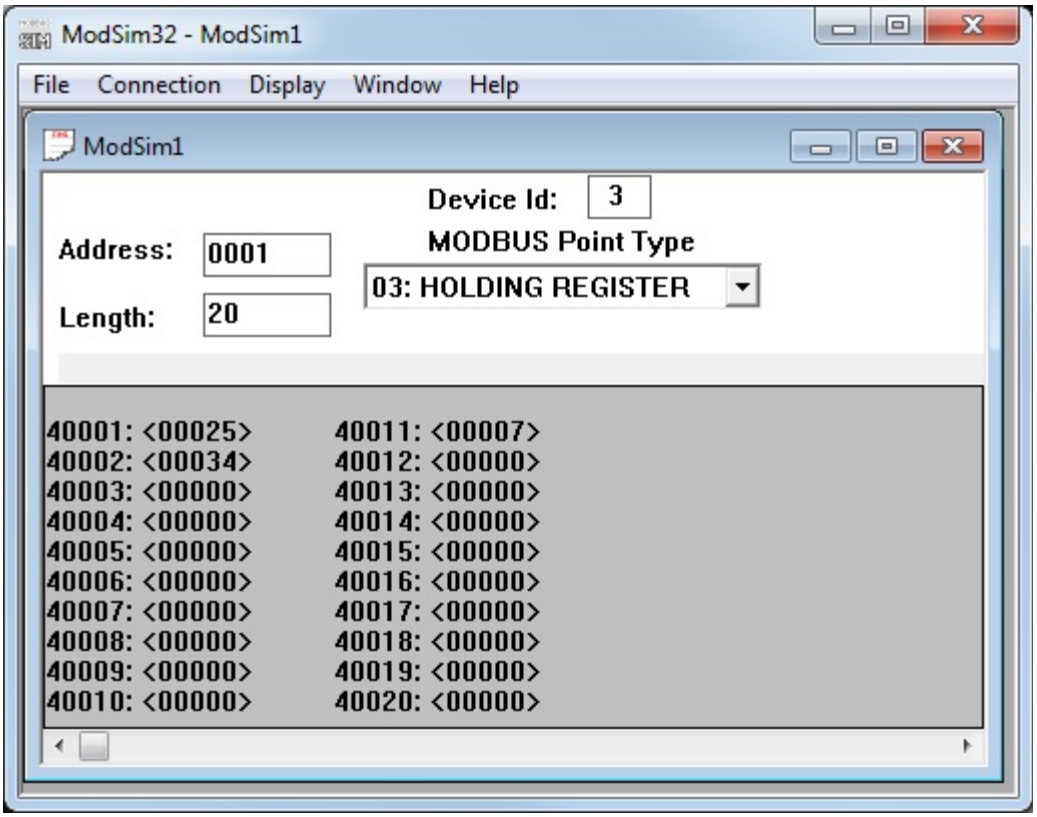
Delete will remove the rule number shown in the "Map #" box. Insert will insert a new rule before the rule number shown, and is used for placing rules between existing rules. It is not necessary to use Insert to add rules to the bottom of the list or to define any rule presently having zero for a source register or "none" for remote type.

Selecting "none" for remote type effectively deletes the rule even though it will still appear in the list until deleted. Unused rules at the end of the list will always show none as the type. If you wish to prevent these from being displayed, reduce the number of rules enabled.



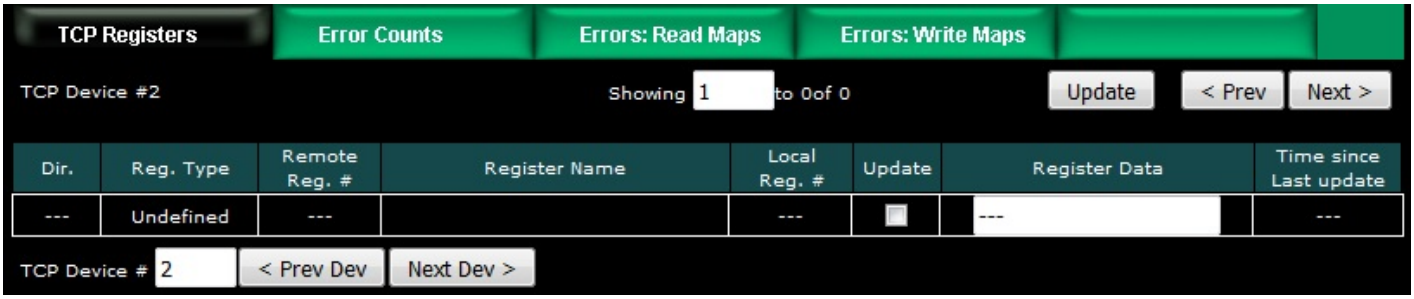
If the local register and remote register are not the same format, then data is converted automatically when written. In the example above, the Write Map is writing

register 3, a floating point value, to remote register 2, an unsigned 16-bit value. The data is converted and rounded up, as illustrated below by ModSim acting as our Modbus TCP server here.

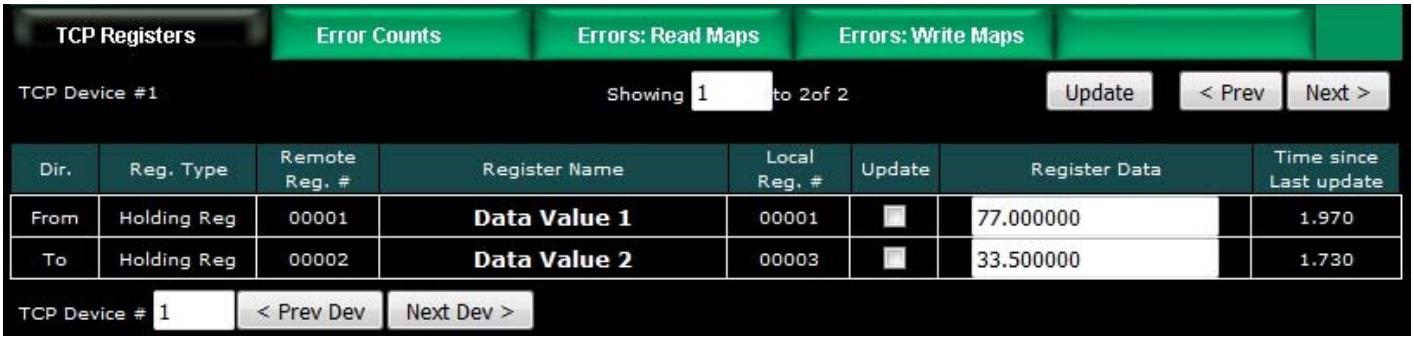


5.4 Modbus TCP Client Data Displayed by Server

The TCP Registers page shows a list of local registers mapped to TCP server devices. The page will show only one device at a time, and may have no entries if there are no maps for that device.

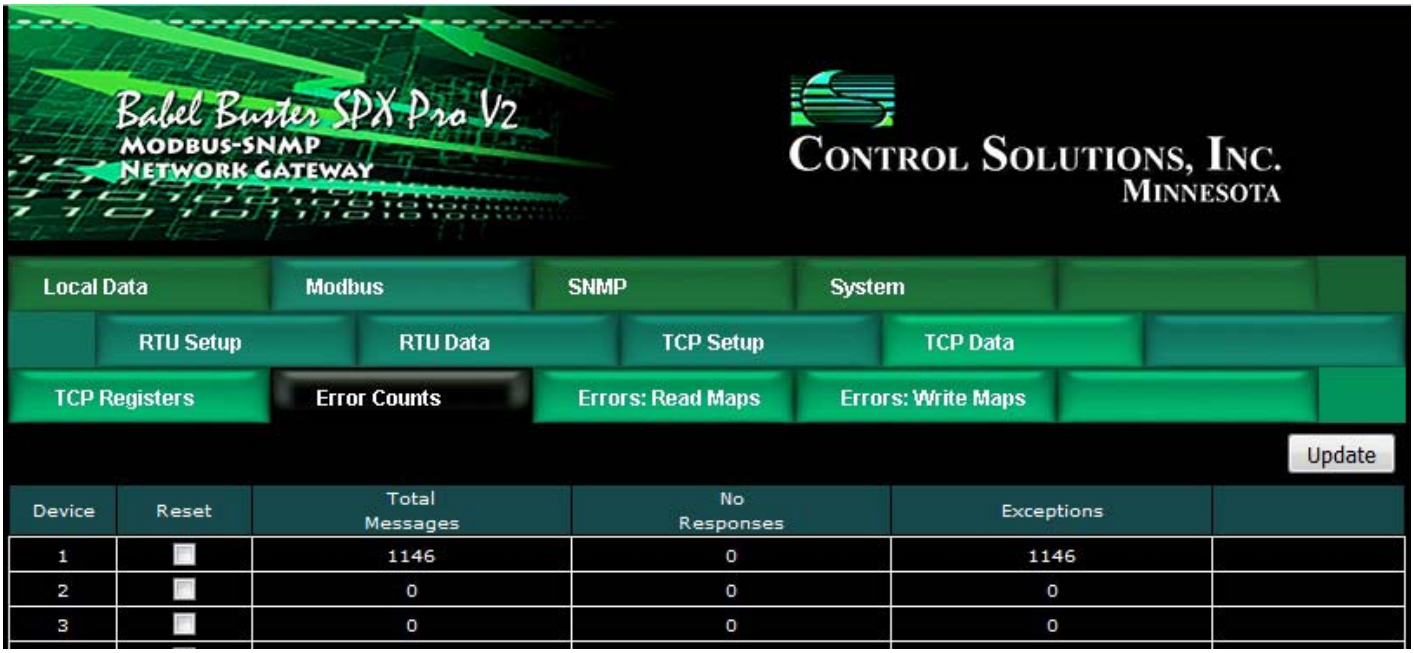


Click the Next Dev or Prev Dev buttons to go to the next or previous TCP device, or simply enter a number in the TCP Device # window and click Update. Registers for that server will now be displayed. In addition to a summary of the map (both read and write maps are shown), the time since last update is displayed. This time should generally be less than the poll time. If the last update time is large, it may mean there is an error preventing the update.

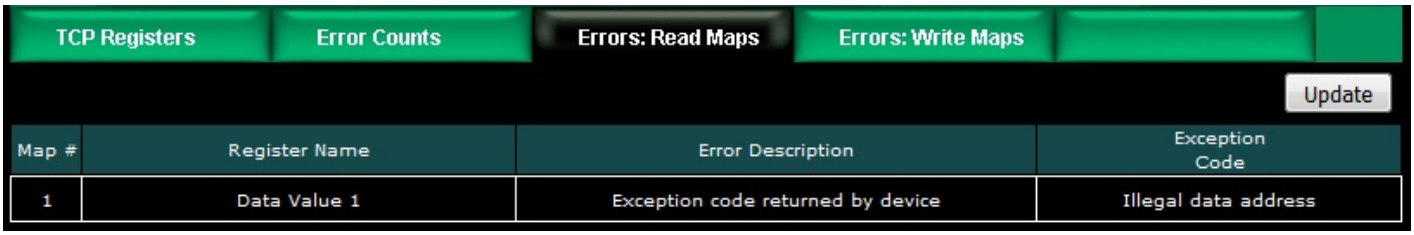


5.5 Modbus TCP Errors

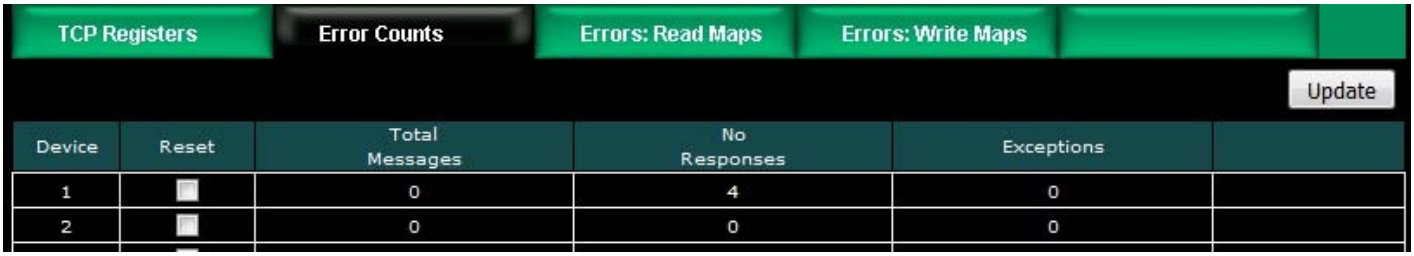
The Error Counts page shows a tabulation, by TCP device, of all errors observed. In the example below, we can see that TCP device #1 is apparently not configured correctly as it is getting as many excpetion errors as messages sent.



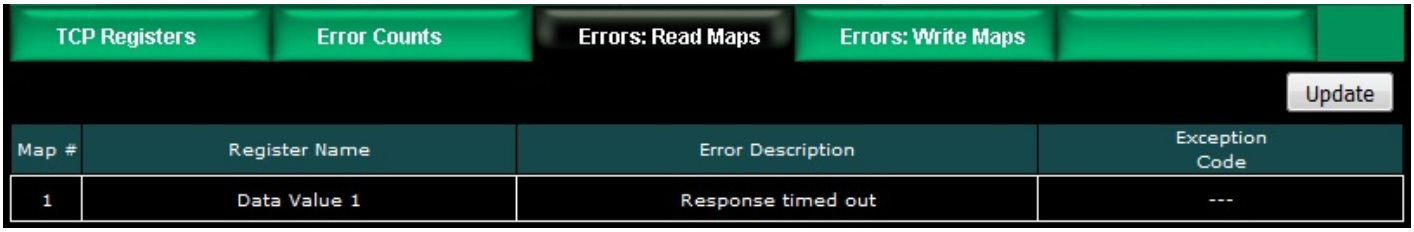
If the counts show some problems, we can look for more detail on the Errors: Read Maps (or Errors: Write Maps) pages. These pages will tell us exactly which Read Map (or Write Map) the problem is occurring on, and what the error is, as illustrated below.



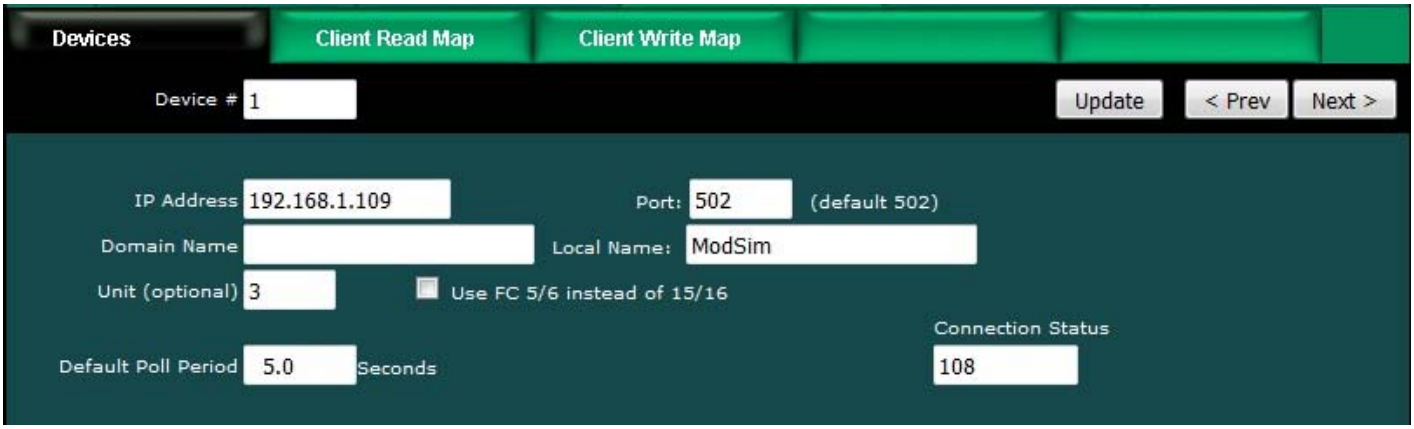
If you see total messages of zero and a "no responses" count greater than zero, it means the Babel Buster was not able to connect to the IP address of the TCP server. Without being able to connect at all, there was never an attempt to send a message, and hence zero total messages while "no responses" continues to increment.



When "no responses" is indicated, the Errors: Read Maps (or Write Maps as applicable) page will show that the response timed out, but this is typically a foregone conclusion when you see the "no responses" count for a TCP device.



When you get any type of connection related problem with a TCP device, the connection status will typically give you some clues.



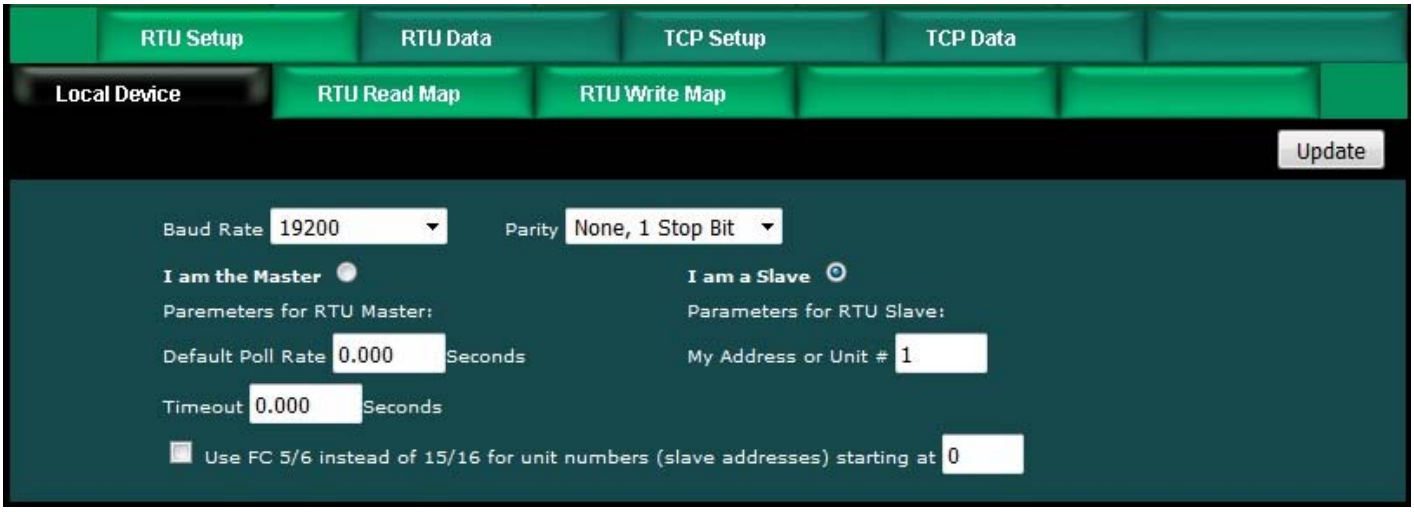
Connection status codes you may see include:

5 = Connection failed, unable to bind (usually means remote device not connected or not reachable)
81 = Connection in progress (means unsuccessful connect attempt, still trying)
95 = Network is unreachable
97 = Connection aborted
98 = Connection reset by peer
103 = Connection timed out
104 = Connection refused
107 = Host is unreachable
108 = Host is reachable, but expected port cannot be opened

6. Configuring Gateway as a Modbus RTU Slave

6.1 Modbus RTU Device Configuration

Device configuration for RTU means configuring the serial port. Select the baud rate and parity as applicable. Select "I am a Slave". It is important to provide an address or unit number that is not used by any other slave on the RTU network. The poll rate, timeout, and "Use FC 5/6..." only apply when RTU is master.



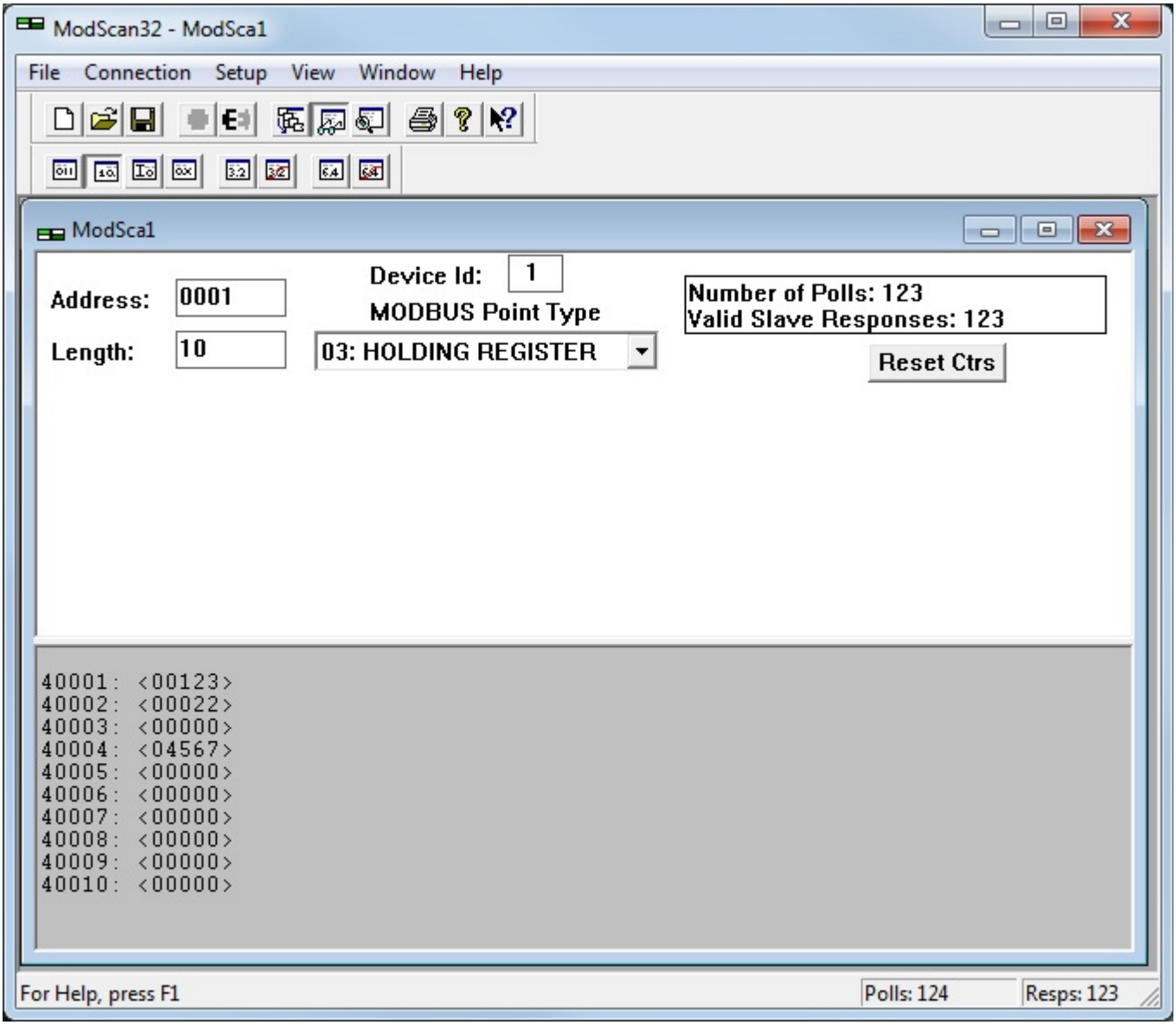
6.2 Modbus RTU Slave Register Map

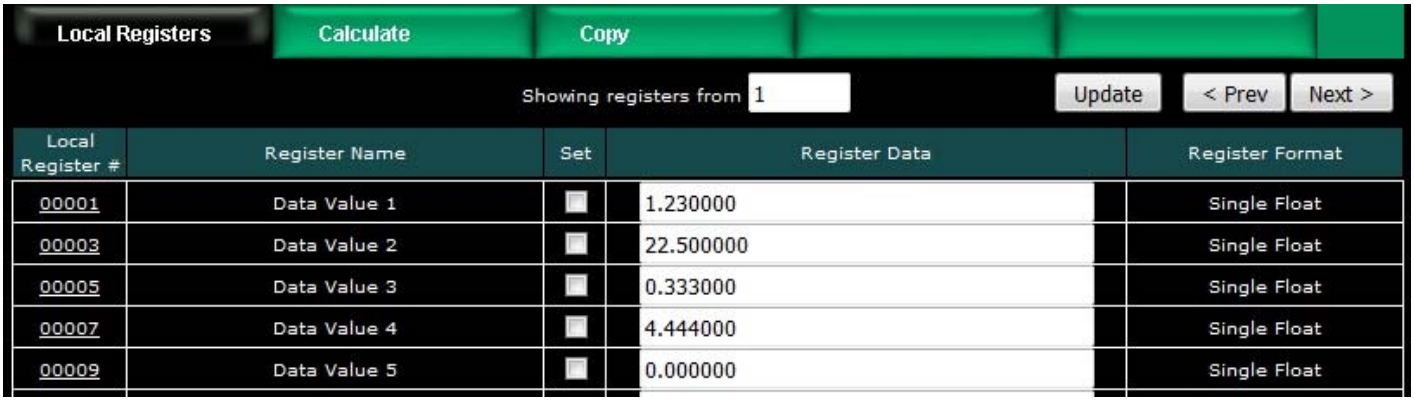
The local registers in the Babel Buster Pro will be most often accessed as holding registers, but can also be accessed as input registers (for reading but input registers cannot be written to). If the local register is defined as 16-bit signed or unsigned, then it can also be accessed as a coil or discrete input (for reading). When accessed as a single bit Modbus register, the value read by the Modbus master will be 0, or 1 if the local register contains 1 or any other non-zero value. Of course the remote master can only write 0 or 1 to a coil. Note also that a local register defined as something bigger than 16-bit cannot be accessed as a coil or discrete input.

The register numbers that the remote Modbus RTU master should read or write are simply those shown in the first column on the Local Registers page.

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	☐	123	Unsigned 16-bit
00002	Data Value 2	☐	22	Unsigned 16-bit
00003	Data Value 3	☐	0	Unsigned 16-bit
00004	Data Value 4	☐	4567	Unsigned 16-bit
00005	Data Value 5	☐	0	Unsigned 16-bit
00006	Data Value 6	☐	0	Unsigned 16-bit

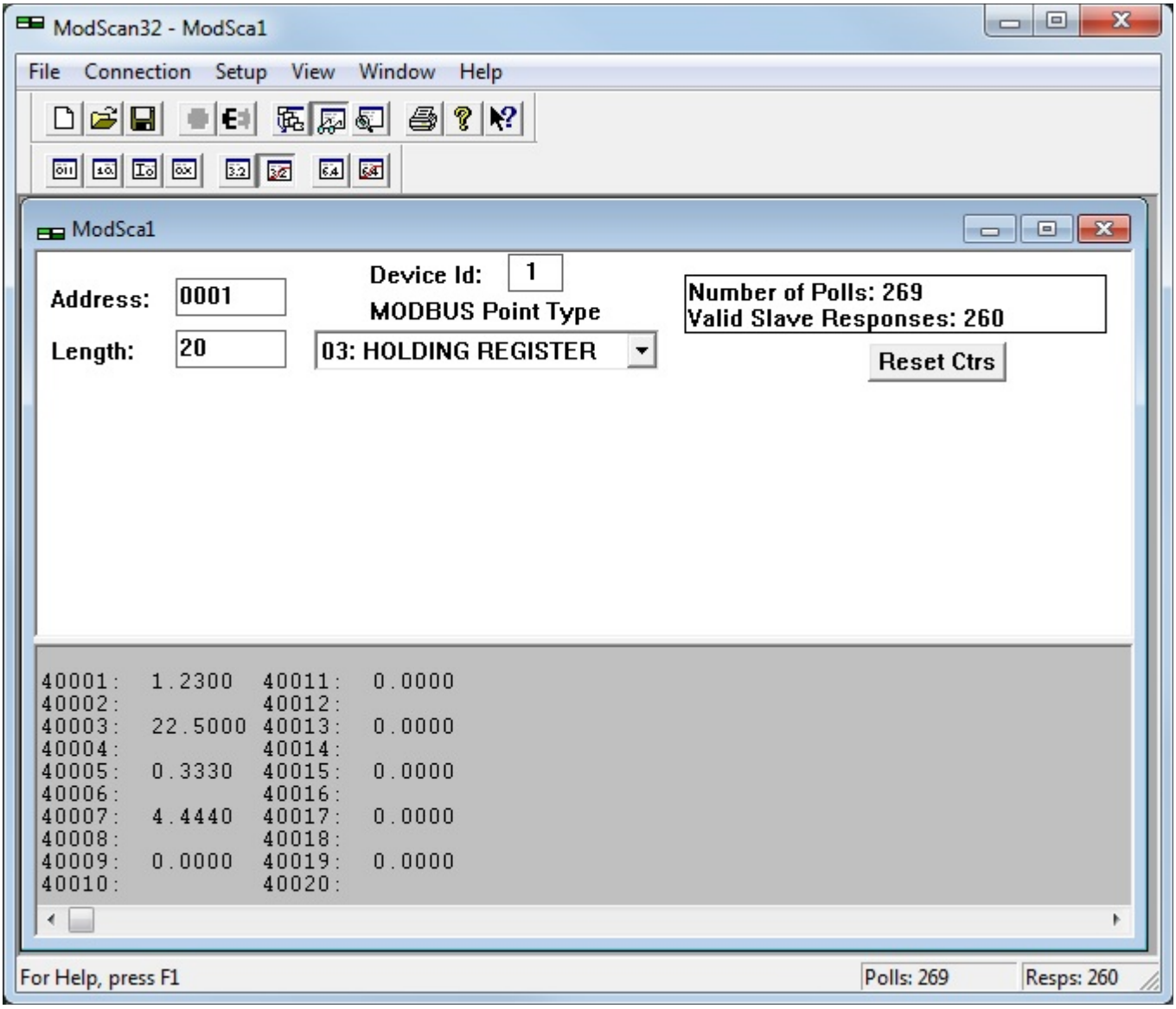
If the local registers are defined as unsigned 16-bit and contain the data illustrated above, then ModScan will read them as illustrated below.





Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	☐	1.230000	Single Float
00003	Data Value 2	☐	22.500000	Single Float
00005	Data Value 3	☐	0.333000	Single Float
00007	Data Value 4	☐	4.444000	Single Float
00009	Data Value 5	☐	0.000000	Single Float

If the local registers are defined as floating point and contain the data illustrated above, then ModScan will read them as illustrated below. Note that registers defined as having data formats greater than 16 bits must be read on the proper boundaries. You are not allowed to read half of a floating point number or just part of any other larger data value. If you were to attempt to read regster #2 in the this example, you would get an exception error. This also means you cannot read just 2 characters in the middle of a larger character string. When the register is defined as character string, you must access the entire character string as a collective set of registers (i.e. read/write multiple holding registers in a single request).



6.2 Modbus RTU Slave Diagnostic

The Babel Buster SPX Pro V2 brand new out of the box will have no registers configured, and will be configured as Modbus RTU slave, 9600 baud, slave address 1. Although no registers are configured, there will be a single holding register accessible for diagnostic purposes, at register number 8801 (or 48801 if using Modicon notation). The content of this register will be firmware revision expressed as a 3-digit number "abbcc" where "a" is the major revision, "bb" is minor revision, and "cc" is build iteration. This should correspond to the firmware revision displayed on the home page (index.html) of the web user interface for the device, which is displayed as "a.bb.c".

7. Configuring Gateway as a Modbus TCP Server

7.1 Modbus TCP Device Configuration

There is really little to do to configure the Babel Buster Pro to be a Modbus TCP server. The gateway needs an IP address and you have most likely already set that. The only other thing is to verify that the Modbus Port number is set to a non-zero number. Port 502 is the port set aside for standard Modbus TCP use and should be used unless you have a specific reason not to.

If you will not be using Modbus TCP and wish to disable it, enter zero for Modbus Port, and click Set Ports. Following the next restart, you will be unable to connect via Modbus TCP with port set to zero.

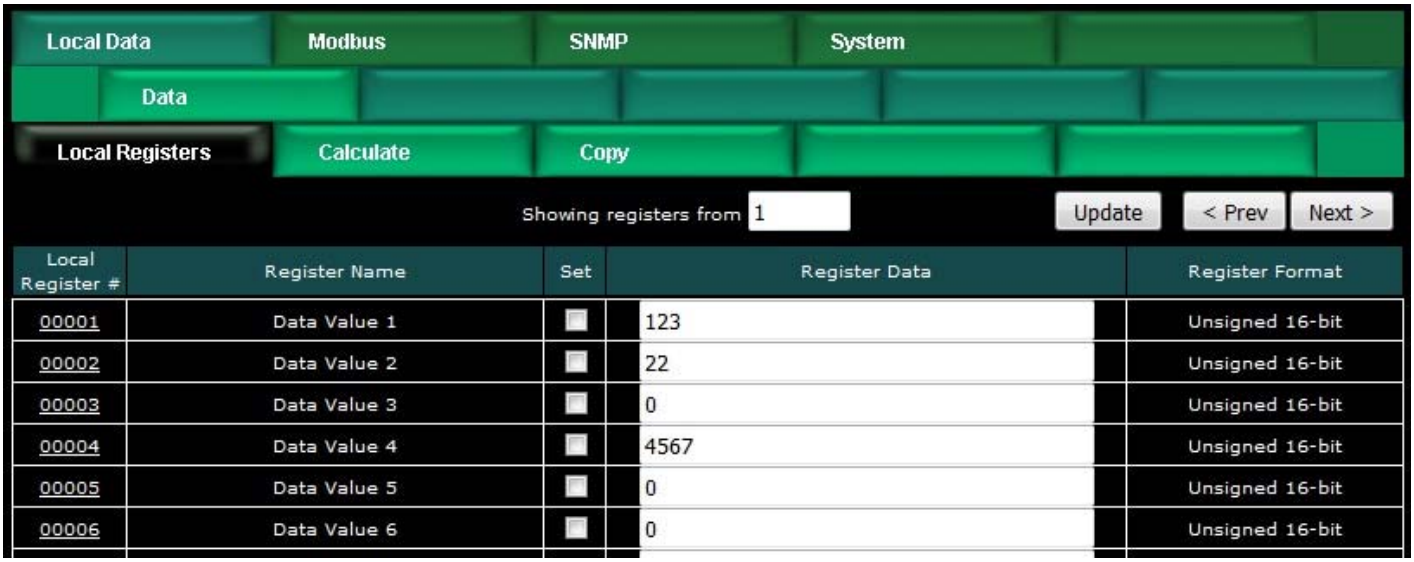
IMPORTANT: The Modbus port will be initially set to zero as shipped from the factory. You will need to change it to 502 and restart before connecting via Modbus TCP for the first time.

7.2 Modbus TCP Server Register Map

The local registers in the Babel Buster Pro will be most often accessed as holding registers, but can also be accessed as input registers (for reading but input registers cannot be written to). If the local register is defined as 16-bit signed or unsigned, then it can also be accessed as a coil or discrete input (for reading). When accessed as a single bit Modbus register, the value read by the Modbus master will be 0, or 1 if the local register contains 1 or any other non-zero value. Of course the remote master can

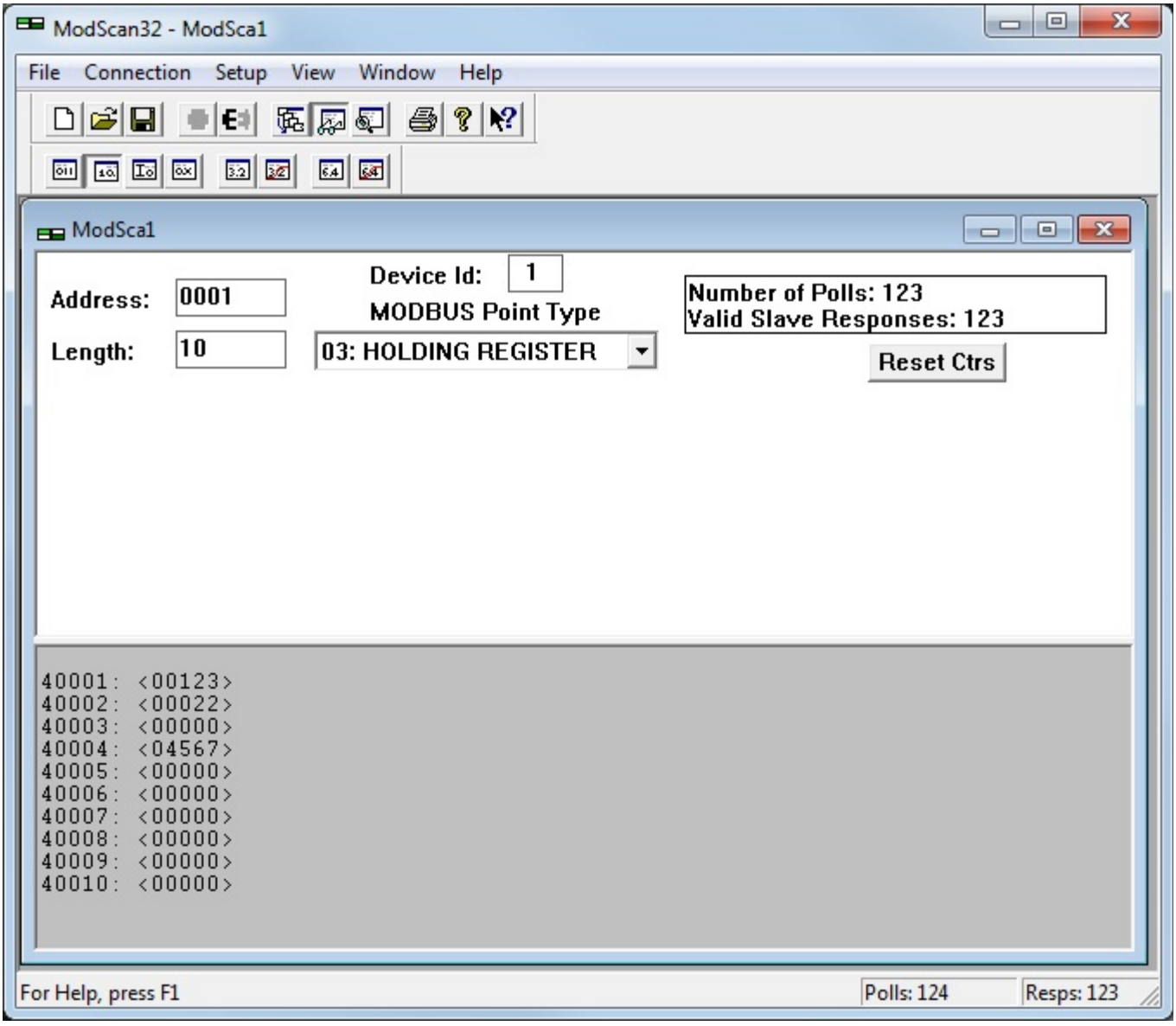
only write 0 or 1 to a coil. Note also that a local register defined as something bigger than 16-bit cannot be accessed as a coil or discrete input.

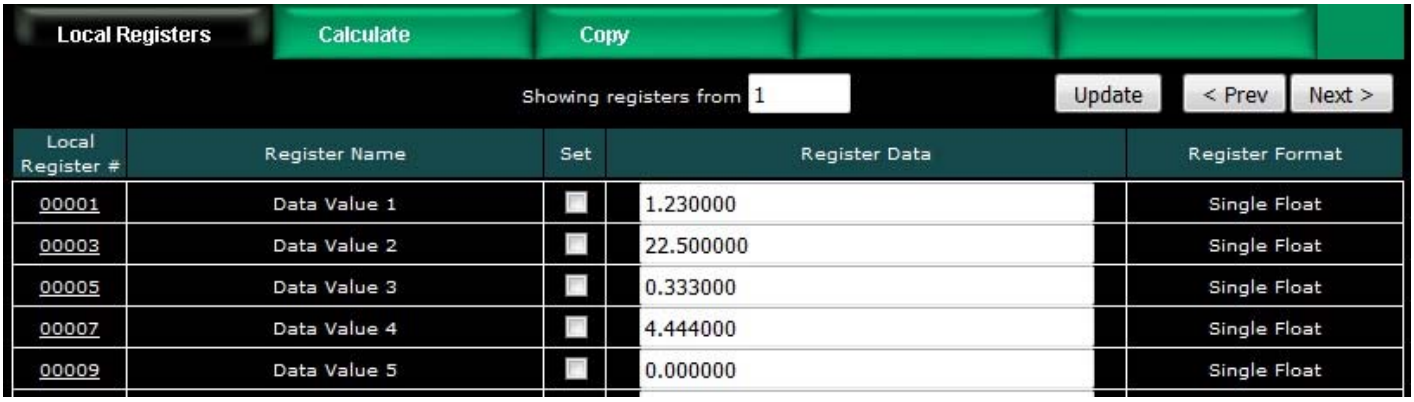
The register numbers that the remote Modbus TCP client should read or write are simply those shown in the first column on the Local Registers page.



Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	☐	123	Unsigned 16-bit
00002	Data Value 2	☐	22	Unsigned 16-bit
00003	Data Value 3	☐	0	Unsigned 16-bit
00004	Data Value 4	☐	4567	Unsigned 16-bit
00005	Data Value 5	☐	0	Unsigned 16-bit
00006	Data Value 6	☐	0	Unsigned 16-bit

If the local registers are defined as unsigned 16-bit and contain the data illustrated above, then ModScan will read them as illustrated below.

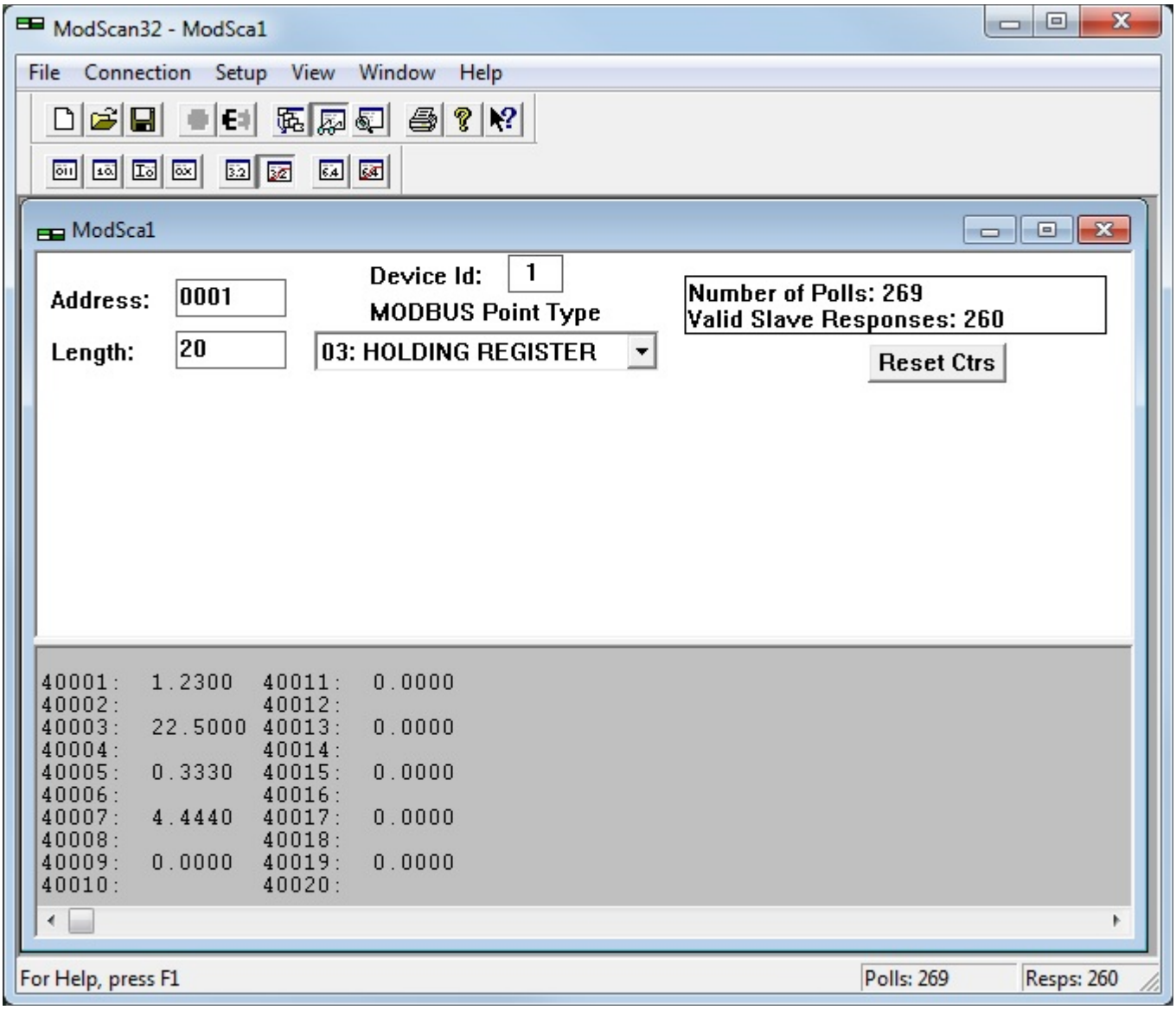




Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	☐	1.230000	Single Float
00003	Data Value 2	☐	22.500000	Single Float
00005	Data Value 3	☐	0.333000	Single Float
00007	Data Value 4	☐	4.444000	Single Float
00009	Data Value 5	☐	0.000000	Single Float

If the local registers are defined as floating point and contain the data illustrated above, then ModScan will read them as illustrated below. Note that registers defined as having data formats greater than 16 bits must be read on the proper boundaries. You are not allowed to read half of a floating point number or just part of any other larger data value. If you were to attempt to read regster #2 in the this example, you would get an exception error. This also means you cannot read just 2 characters in the

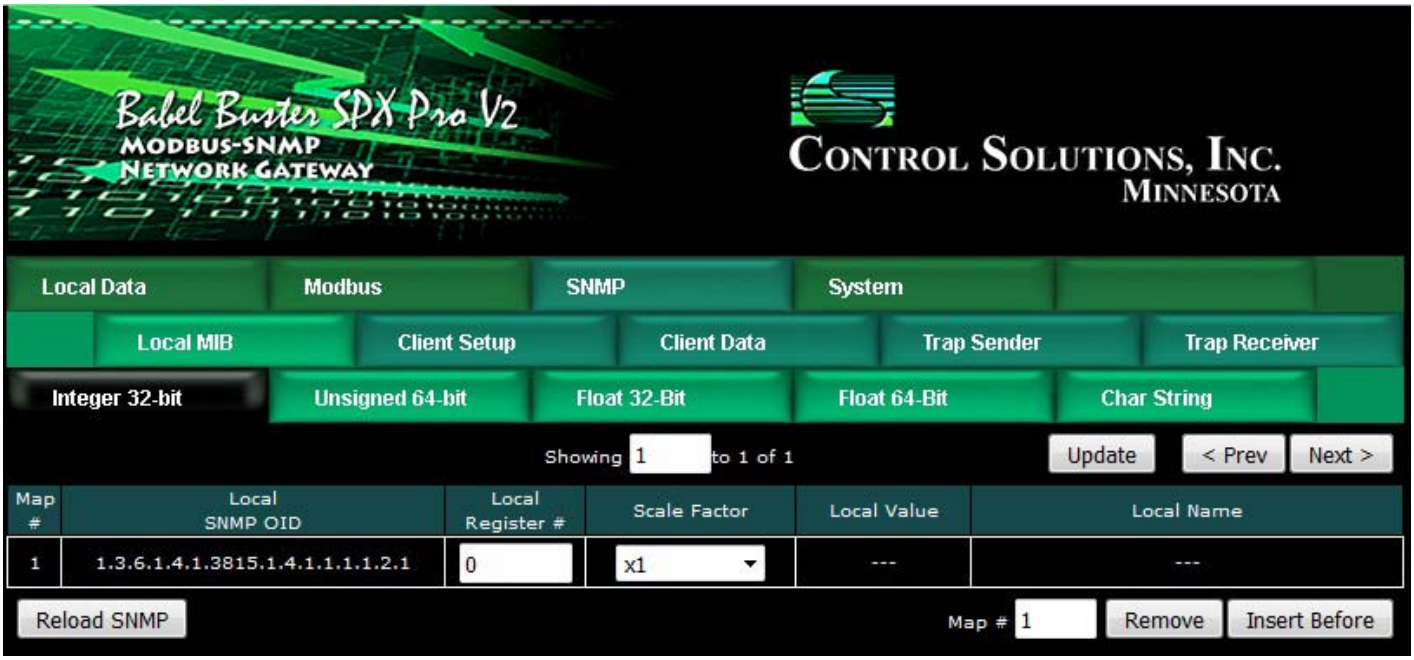
middle of a larger character string. When the register is defined as character string, you must access the entire character string as a collective set of registers (i.e. read/write multiple holding registers in a single request).



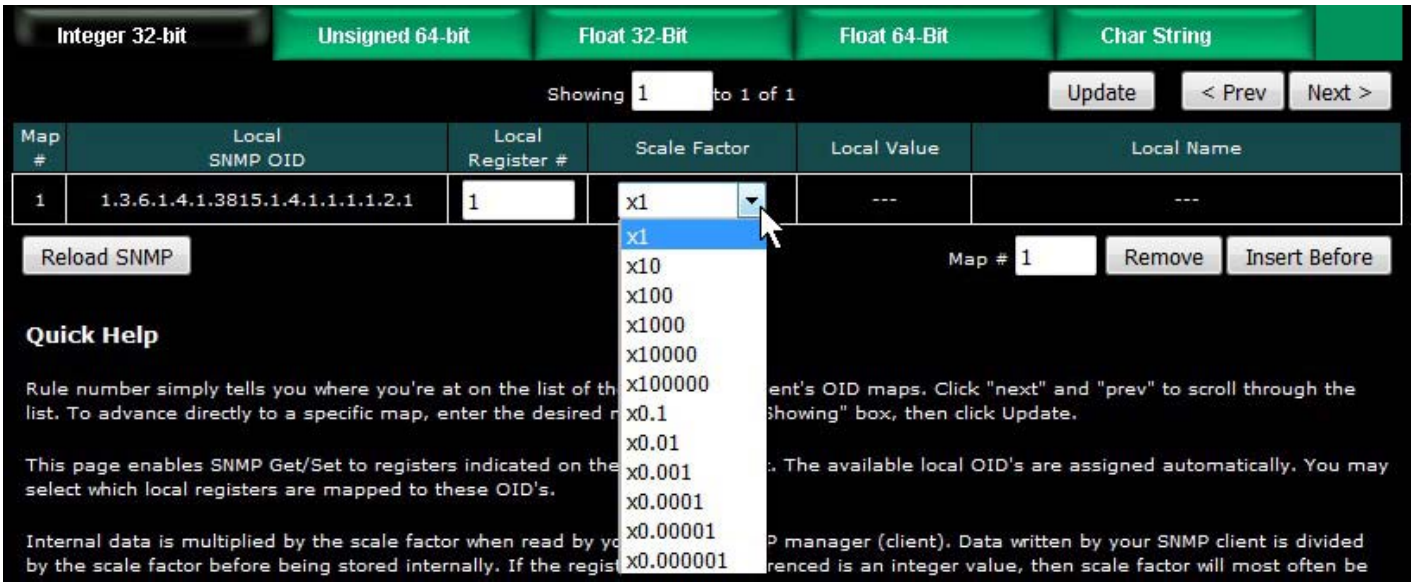
8. Configuring Gateway as an SNMP Server

8.1 Creating Local SNMP MIB

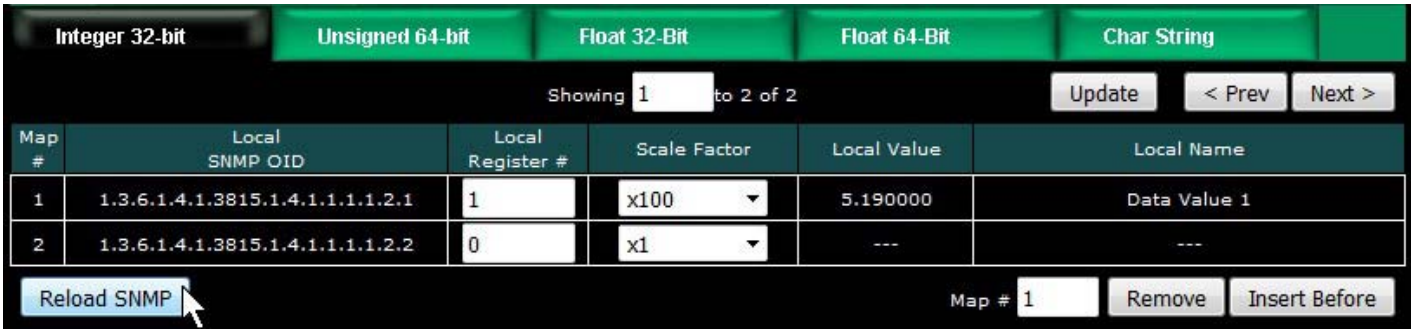
The Babel Buster Pro starts out with no variables in its MIB just like it starts out with no local registers. When you do create local registers, you then have a choice of where to make them show up in the MIB. There are five branches in the Pro MIB, although only the Integer branch is guaranteed to be universally accessible to all other SNMP devices. It is up to you to select which branch of the MIB to place each local register in. You do not need to place all local registers in the MIB, only those that you want externally accessible via SNMP. You must also place local registers in the MIB in order to generate SNMP traps related to those local registers.



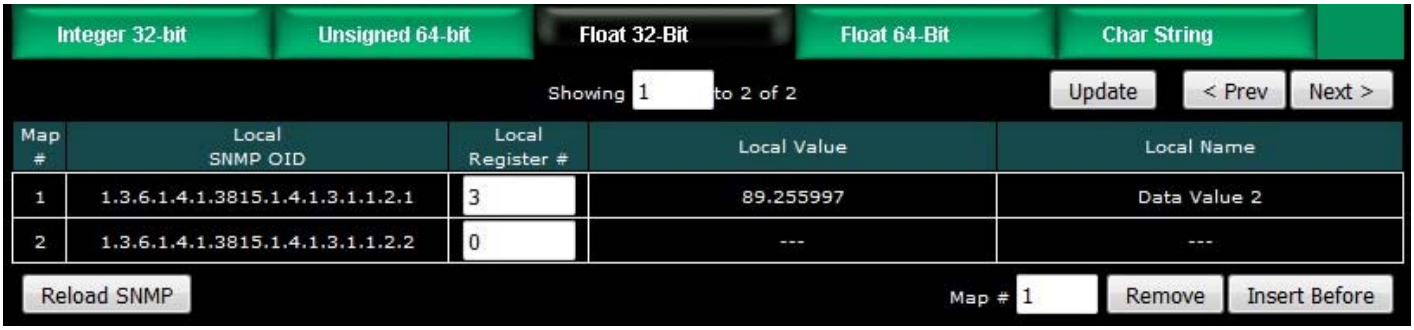
To add a local register to a MIB branch, simply enter the local register number at the next available OID which will always automatically be at the bottom of the list. When placing registers in the Integer branch, you also have the option of applying a scale factor. Scaled integer is the most universally recognized means of transmitting non-integer data.



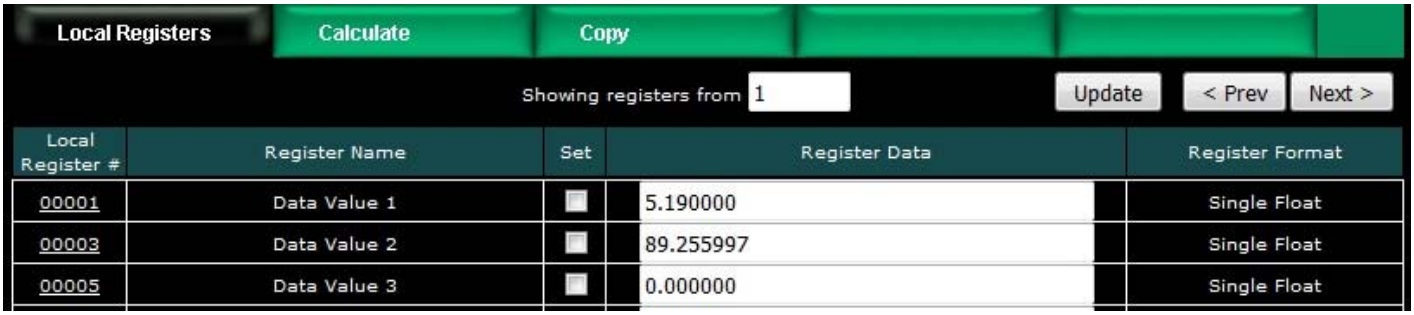
In this example, we have selected a scale factor fo x100. That means that our local value of 5.19 will be transmitted (in a Get response) as integer 519 and it is up to the recipient to know that this is scaled x100.

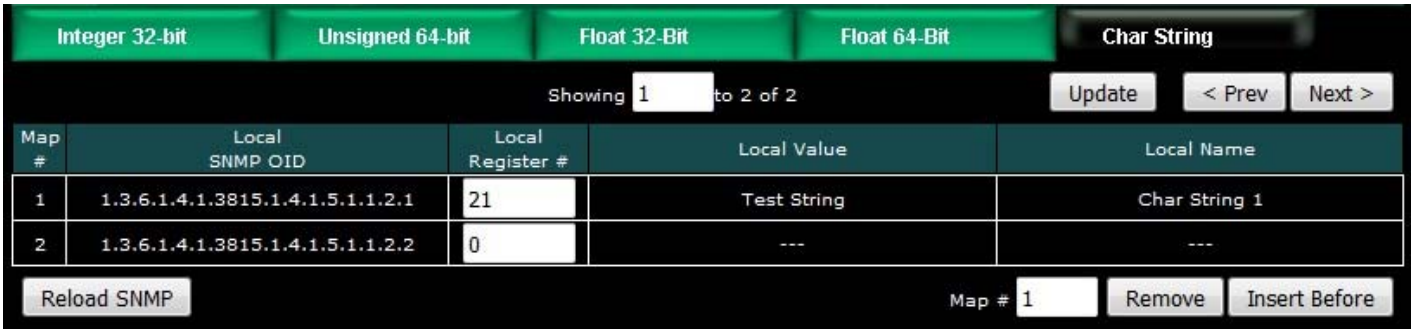


After adding new members to the MIB, it is necessary to click Reload SNMP before they will become accessible to an external SNMP manager's Get request.

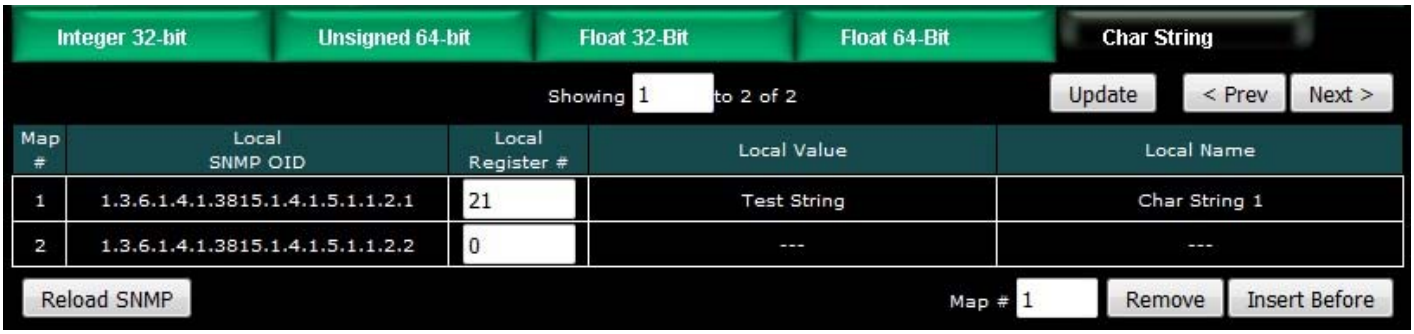


Local registers added to the Float 32-Bit MIB branch will be provided in IEEE 754 format per RFC 6340.





Local registers defined as character strings may be added to the Char String branch of the MIB. Registers in this branch will be provided as an Octet String.



8.2 Supported Data Formats, RFC 6340

SNMP does not have a universally accepted representation for floating point. The one universally known data type is INTEGER. A commonly recomended means of transmitting floating point data is either as a scaled integer or as an ASCII character string. There is an RFC 6340 for representation of floating point based on IEEE 754 encoding. The "Float 32-bit" and "Float 64-bit" data types in the Babel Buster Pro refer to RFC 6340 encoding.

Specifically, the data types found in the Babel Buster Pro MIB are encoded with ASN types as follows:

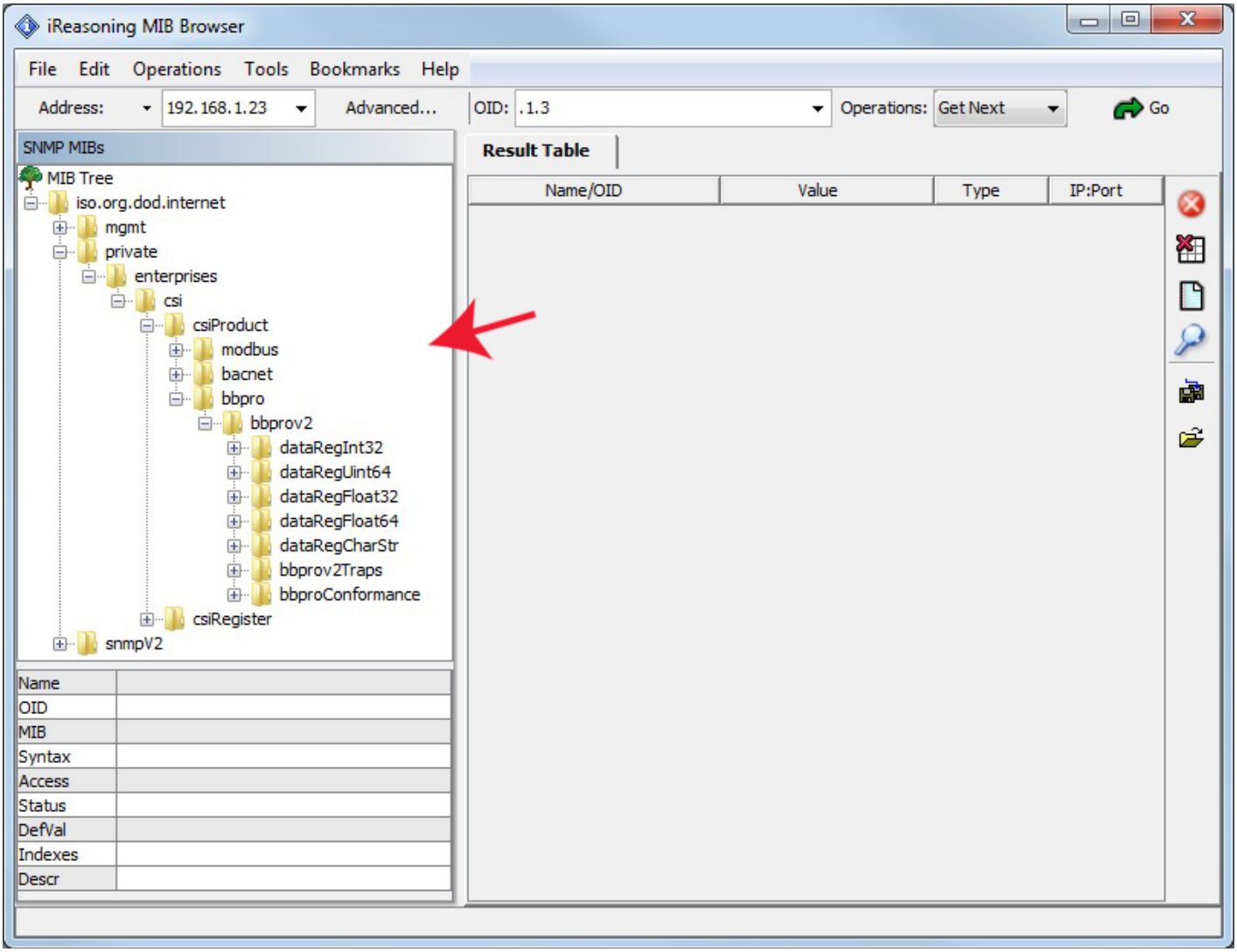
Integer 32-Bit	INTEGER	ASN_INTEGER
Unsigned 64-Bit	COUNTER64	(ASN_APPLICATION | 6)
Float 32-bit	OCTET STRING	ASN_OCTET_STR (length 4)

Float 64-bit	OCTET STRING	ASN_OCTET_STR (length 8)
Char String	OCTET STRING	ASN_OCTET_STR (length variable)

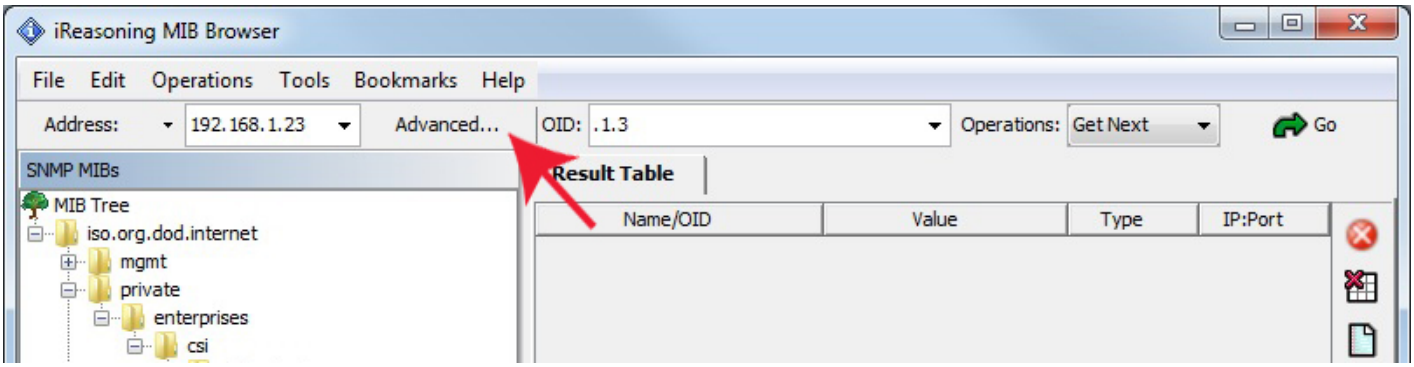
8.3 Testing the SNMP Agent

A variety of tools are available for browsing an SNMP MIB and receiving SNMP Traps. The tool used in the following examples is the iReasoning MIB Browser. Refer to the Tools section under Support at csimn.com for more information about SNMP tools.

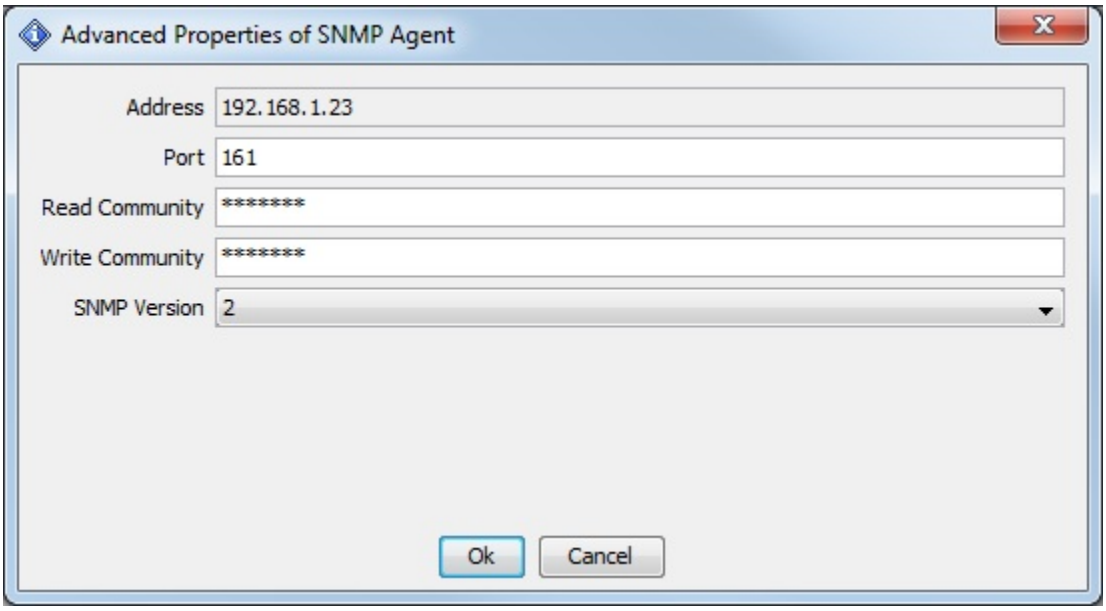
The MIB browser allows you to view the MIB variables in the Babel Buster Pro. Before you can browse the MIB in a meaningful way, you need to load the MIB files that tell the browser what it needs to know about the Pro's MIB. There are three files that need to be loaded, in order: CSIreg.mib, FLOAT-TC-MIB.mib, and CSIbbprov2.mib. Once you have loaded these files as illustrated below, you can view the tree structure of the MIB in the browser.



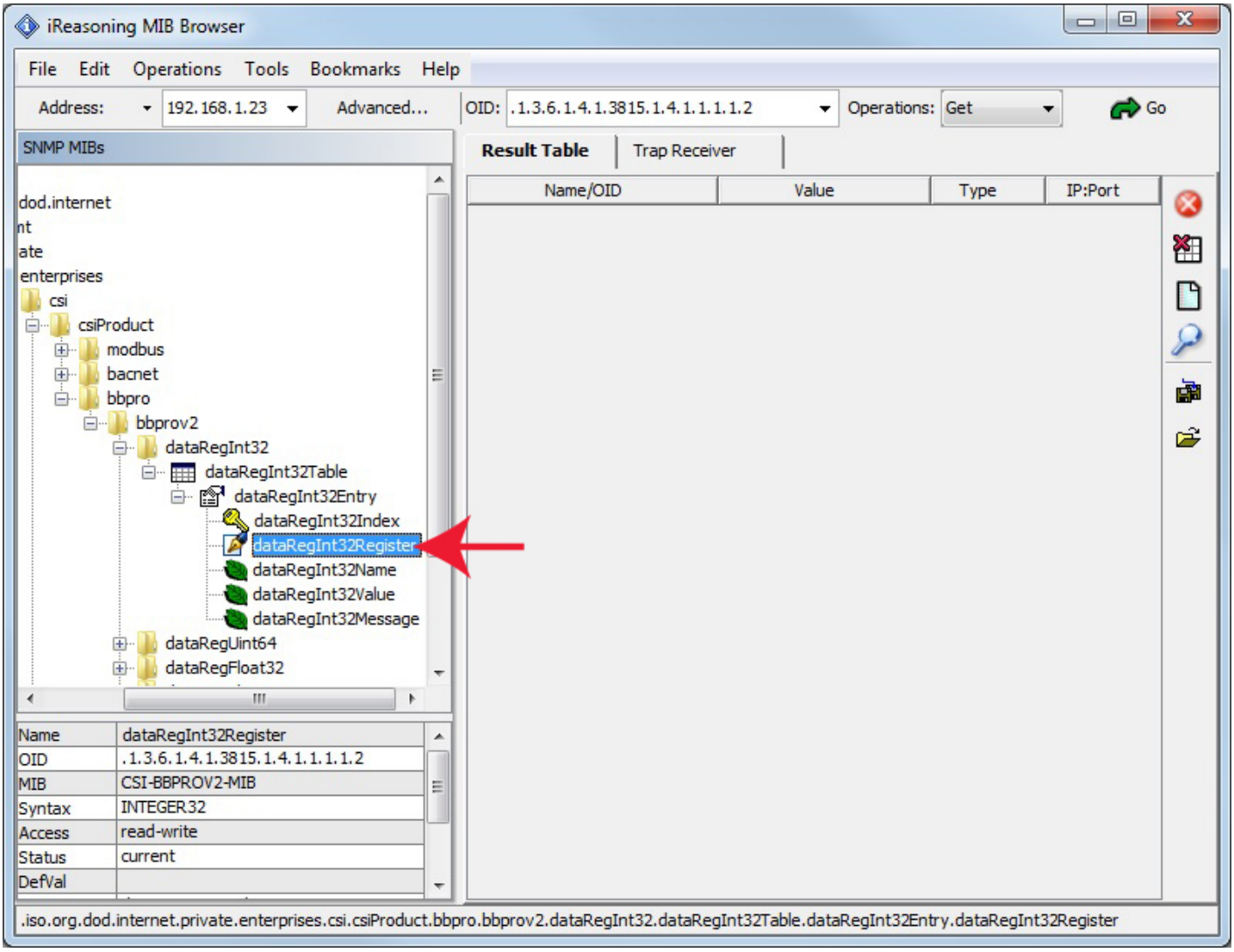
Enter the IP address of the Babel Buster Pro in the Address window. In addition. click on Advanced... to set access parameters.



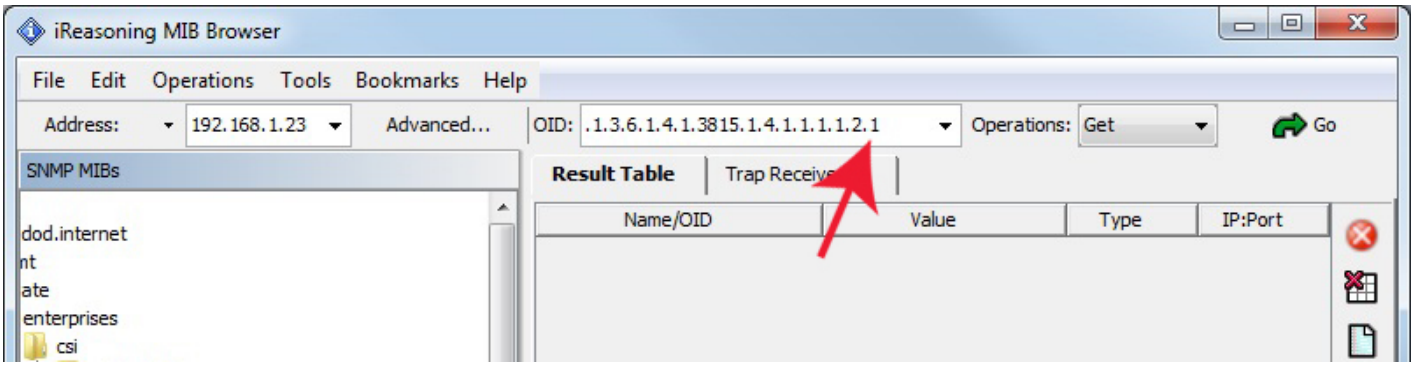
Click Advanced... to open the dialog that lets you enter the community string for the Babel Buster Pro. This work ssomewhat like a password. If the community string you provide here does not match the community string set in the Babel Buster Pro, you will be unable to access MIB variables in the Babel Buster. You also need to select the correct SNMP version number in order to get the correct result. Babel Buster Pro MIB is version 2. (Babel Buster Pro can send traps as either v1 or v2c, but Get/Set must be v2.)



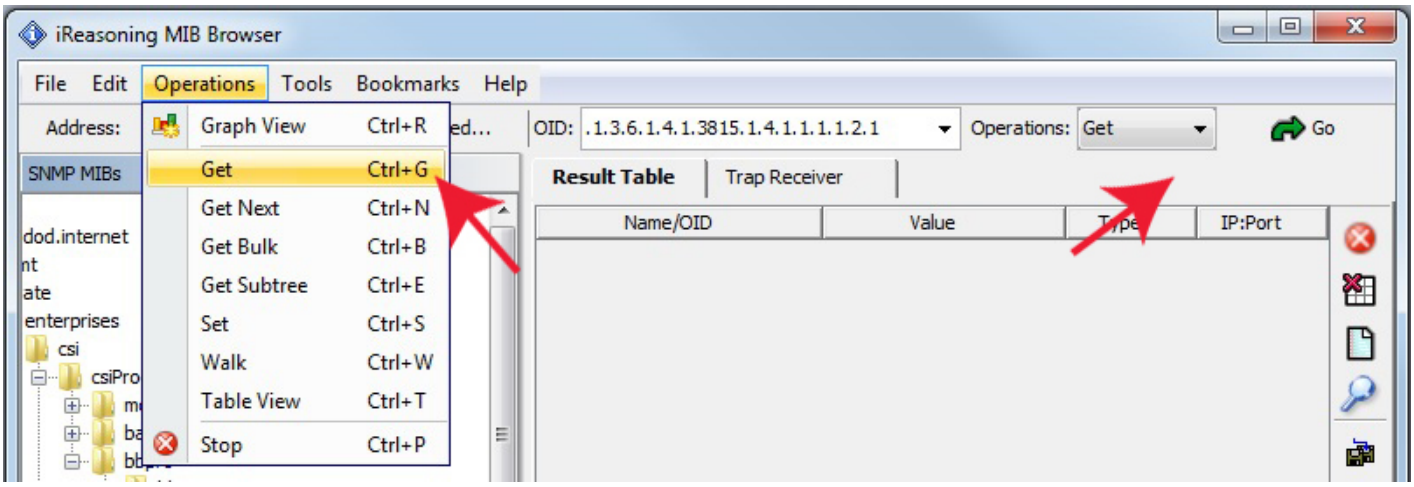
To read a MIB variable from the Babel Buster Pro, start by selecting the register member of the data table. In this case, we are selecting the 32-Bit Integer branch of the MIB.



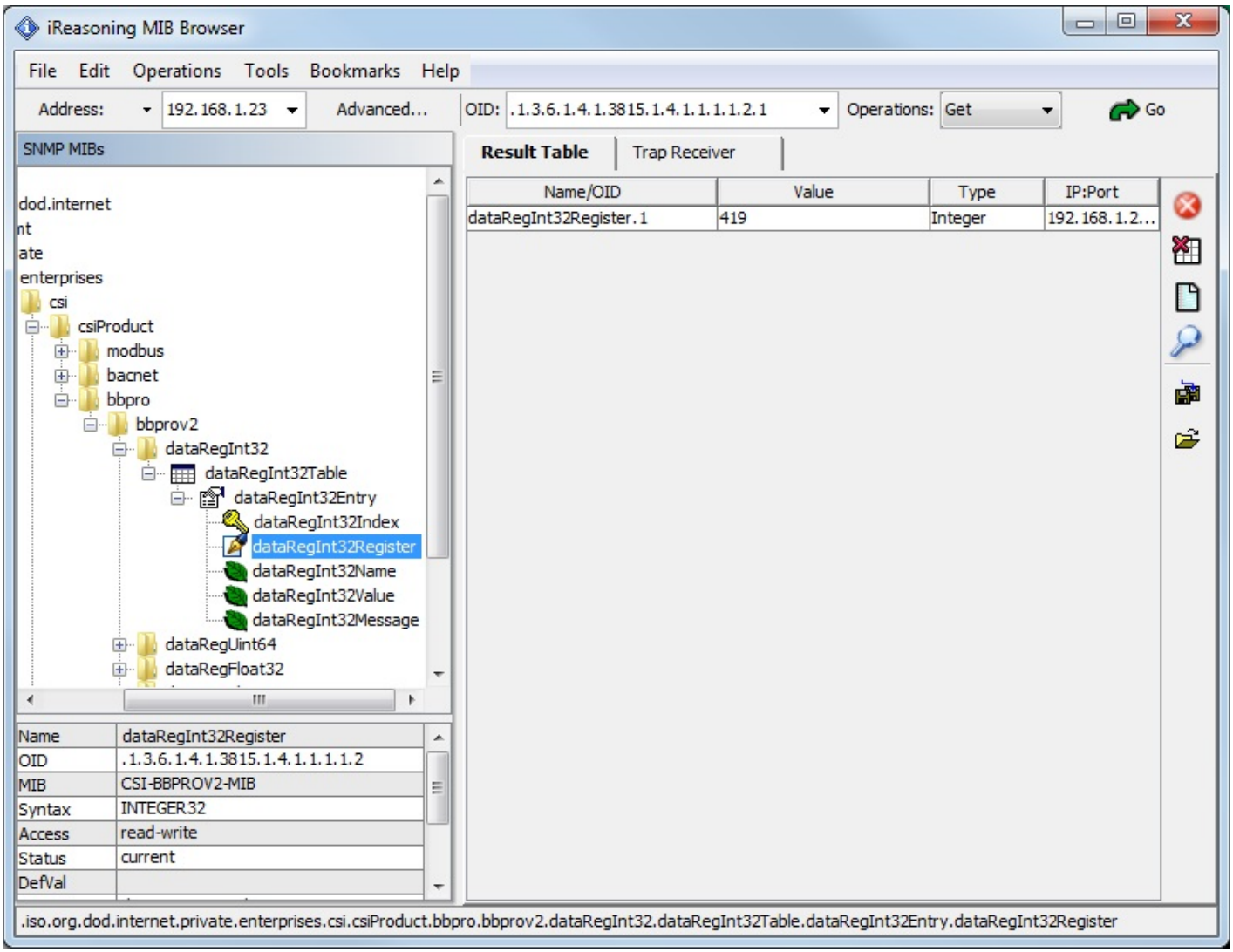
You will need to specify which row in the table you want to read. Do this by appending (by typing) a number to the OID that appeared in the OID window when you clicked on the table entry in the MIB tree view.



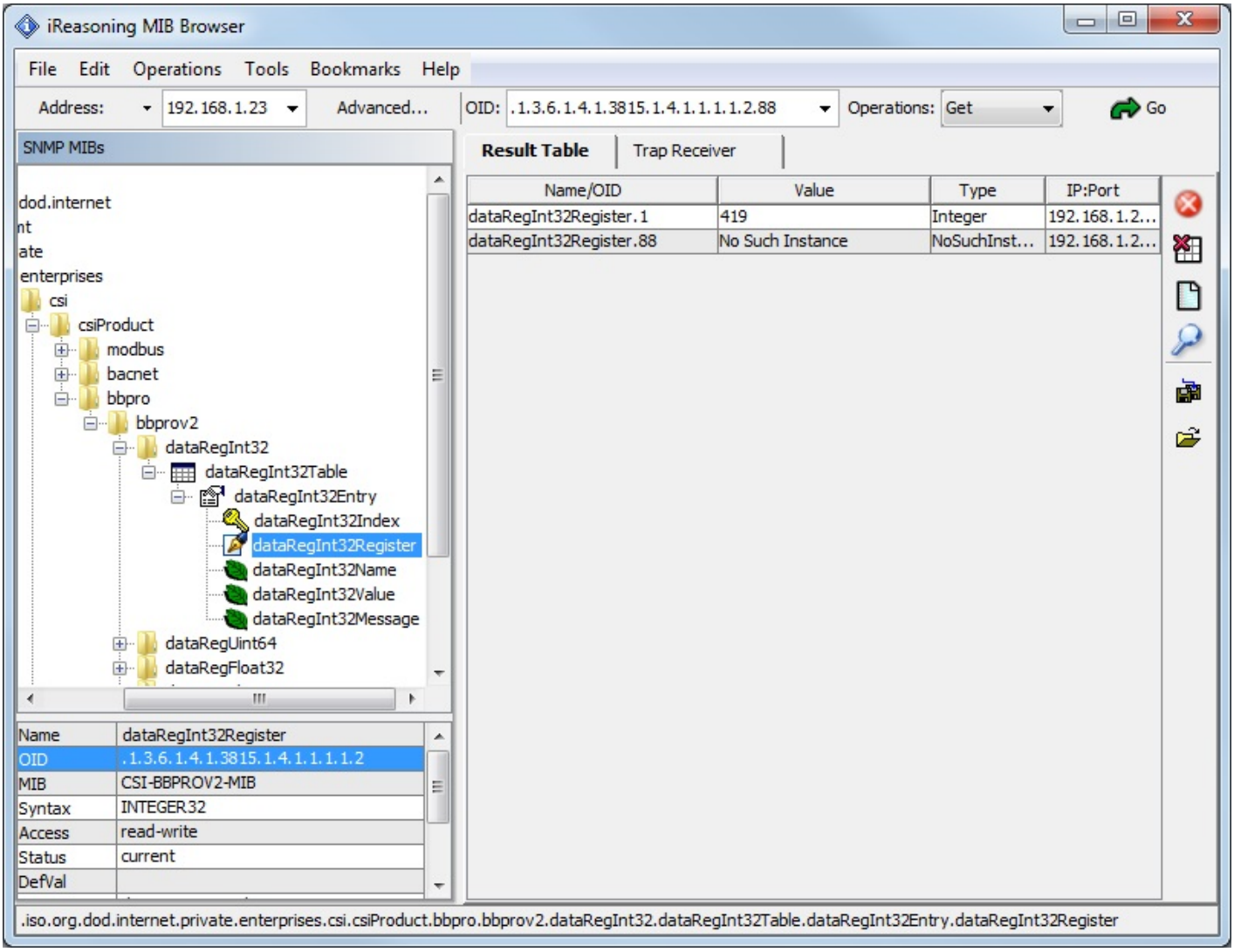
To cause the MIB browser to Get a value from the Babel Buster Pro, select Get from the Operations menu. You can select Get from either menu, left or right, in the iReasoning browser. Once Get is selected on the right, you only need to click the Go button to repeat the Get.



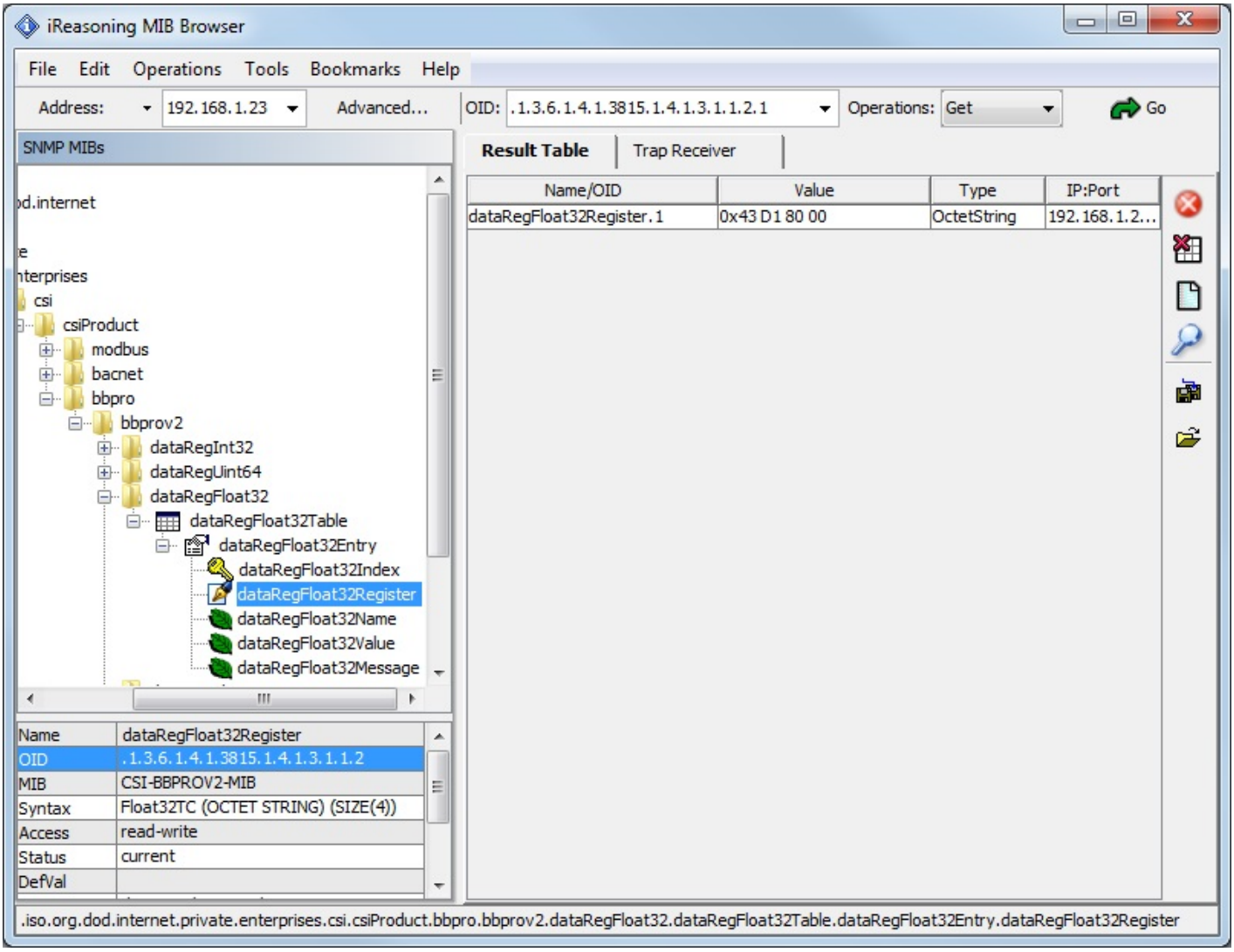
Upon successfully Getting a value, the display will appear as illustrated below.



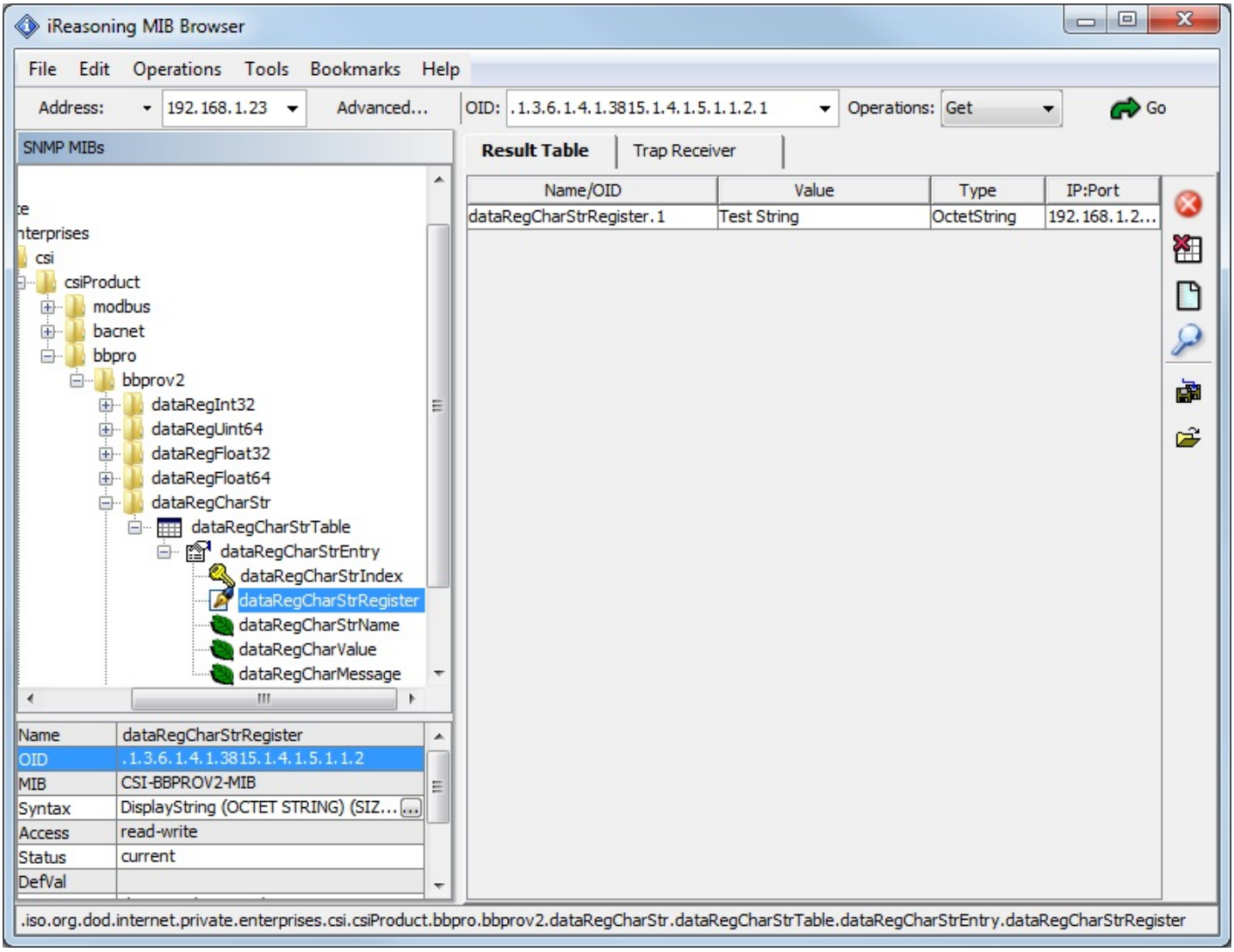
If you attempt to Get a variable that does not exist, you will get the error message illustrated below. This will also happen if the variable has been added to the MIB but you forgot to click Reload SNMP after adding the local register to the MIB. This will also happen if you did not select the right variable from the MIB tree shown, or forgot to add the index to the end of the OID in the Object ID window.



Getting our floating point value will appear as follows. The bytes "43 D1 80 00" translate to the floating point version of 419.00. You can use Google to locate conversion tools to go from IEEE 754 hexadecimal representation (as shown in the browser) to decimal.



The SNMP Get of our character string is illustrated below. Although encoded generically as an Octet String, the MIB browser is kind enough to display it as an ASCII string since it found the string to be all printable characters.


iReasoning MIB Browser
File Edit Operations Tools Bookmarks Help
Address: 192.168.1.23 Advanced... OID: .1.3.6.1.4.1.3815.1.4.1.5.1.1.2.1 Operations: Get Go
SNMP MIBs
nterprises
csi
csiProduct
modbus
bacnet
bbpro
bbprov2
dataRegInt32
dataRegUint64
dataRegFloat32
dataRegFloat64
dataRegCharStr
dataRegCharStrTable
dataRegCharStrEntry
dataRegCharStrIndex
dataRegCharStrRegister
dataRegCharStrName
dataRegCharValue
dataRegCharMessage
Result Table
Trap Receiver
Name/OID
Value
Type
IP:Port
dataRegCharStrRegister.1
Test String
OctetString
192.168.1.2...
Name dataRegCharStrRegister
OID .1.3.6.1.4.1.3815.1.4.1.5.1.1.2
MIB CSI-BBPROV2-MIB
Syntax DisplayString (OCTET STRING) (SIZ...
Access read-write
Status current
DefVal
.iso.org.dod.internet.private.enterprises.csi.csiProduct.bbpro.bbprov2.dataRegCharStr.dataRegCharStrTable.dataRegCharStrEntry.dataRegCharStrRegister

9. Configuring Gateway as an SNMP Client

The SNMP Client is used to read and write (Get and Set) data in other SNMP devices. Data read from a remote SNMP device is stored in a local register when received. Data written to a remote SNMP device is taken from a local register when sent. The local registers are the same collection of registers that are accessible to Modbus or to other SNMP devices via the Babel Buster Pro's own MIB.

9.1 SNMP Device Configuration

The list of other SNMP devices that will be accessed by the Babel Buster Pro are entered on the Devices page under SNMP Client Setup.

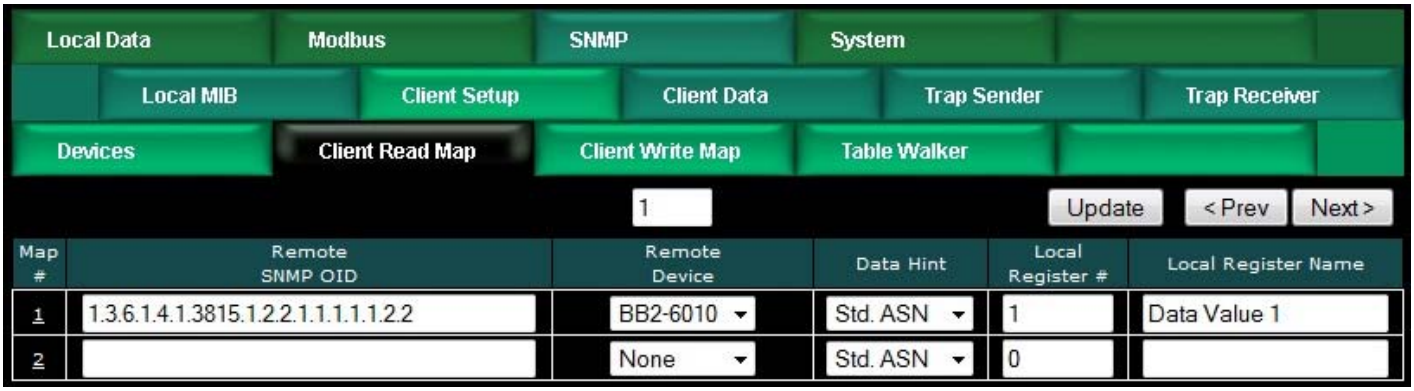
Enter the IP address of the SNMP agent that the Babel Buster Pro will send Get or Set requests to. Provide a local name for this device. This local name will appear in the device list when setting up Client Read Maps and Client Write Maps. Select whether v1 or v2 requests should be sent to this device (v3 is not supported by this model gateway). Your Get or Set request will not be honored by the remote SNMP device if you do not provide the correct community string. Enter the community here. You will

need to obtain that from the device you will be Getting or Setting.

Enter a default poll period. This sets the rate at which the Babel Buster Pro will periodically Get or Set the variables identified in the Read and Write Maps.

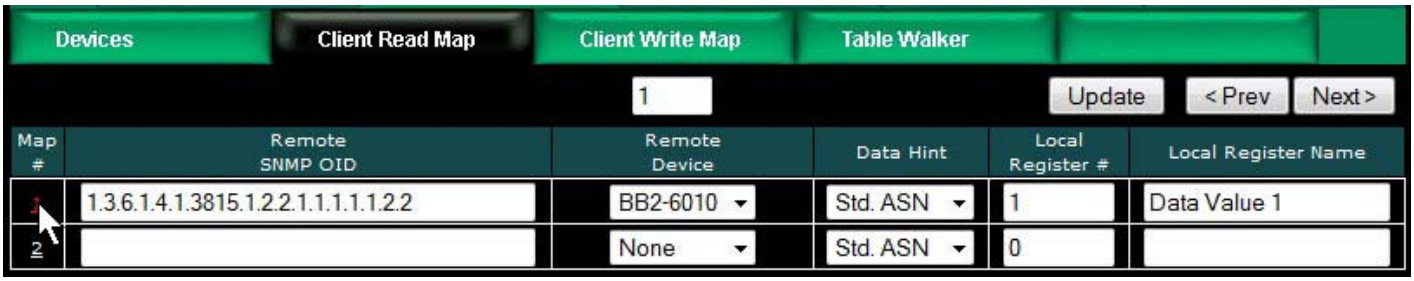
9.2 SNMP Client Read Maps (Get)

Reading or Getting data from another SNMP device requires setting up a Read Map.

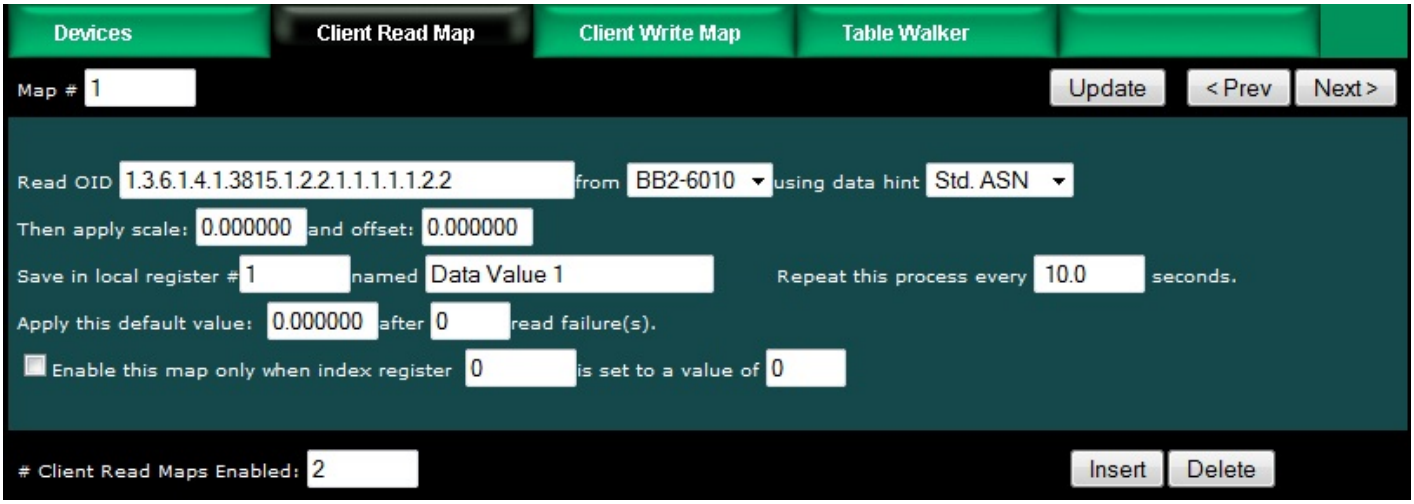


To configure a Read Map, enter the OID of the variable you wish to Get, select the SNMP device from the device list, and provide a local register number where the received data should be placed. If the remote device will be providing floating point as RFC 6340, select that in the Data Hint list, otherwise leave the selection as "Std. ASN" which means stanard ASN encoding.

Data hint helps the data parser figure out what the varible is. If ASN encoding is recognized as anything other than an octet string, the hint will be disregarded. If an octet string is found, then the parser needs to know if it should be treated as RFC 6340 floating point. If no hint is given (standard ASN), then the octet string will be treated as an ASCII character string, in which case ASCII to numeric conversion will be attempted automatically if the local register is numeric (string will simply be copied if result register is a character string type register). Note that standard, well-known ASN types are recognized as well as the NetSnmp ASN type for opaque float.



Click on the Map number in the first column to access the expanded view of the Read Map where additional optional parameters may be entered.



The optional scale and offset may be used if the value being read is scaled, or you may simply use scale and offset to perform units conversions. If non-zero, the received data will be multiplied by scale, then added to offset, before being placed in the local register.

If the remote SNMP device should be read at some rate other than the default poll time given on the Devices page, enter that here.

If you wish to have a specific default value set in the local register in the event the Get fails, enter that default value and some non-zero number of read failures. The default value will be set after this number of failed attempts.

You have the option of making this Read Map conditional. If an index register number is provided and the Enable box is checked, then this read map will only be executed when the index register (local register) contains the value given. This allows multiple read maps to supply data to the same local register based on the value of the index register. It also allows reading to simply be suspended if a single read map supplies data to the local register. In a more sophisticated scenario, you could potentially suspend reading of the remote SNMP device if you know the device is powered down.

Rule number simply tells you where you're at on the list of Read Maps. Click "next" and "prev" to scroll through the list. To advance directly to a specific map, enter the desired number in the "Map #" box, then click Update.

Delete will remove the rule number shown in the "Map #" box. Insert will insert a new rule before the rule number shown, and is used for placing rules between existing rules. It is not necessary to use Insert to add rules to the bottom of the list or to define any rule presently having zero for a source object or "none" for remote type.

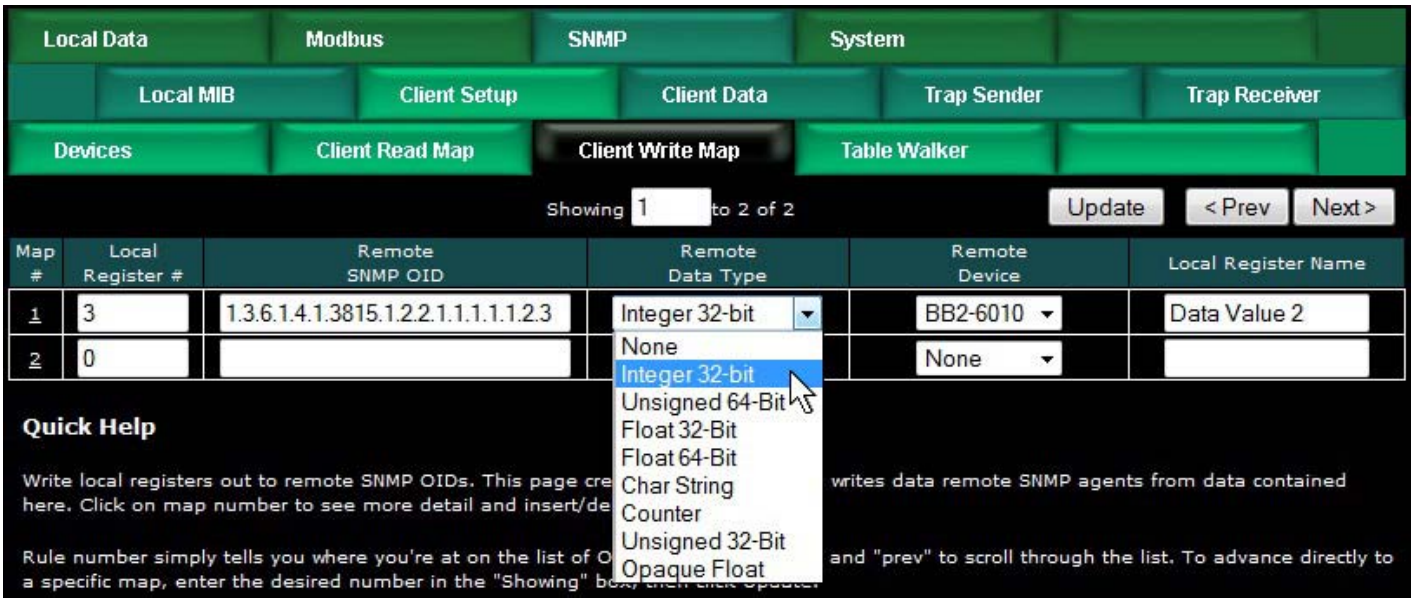
Selecting "none" for remote type effectively deletes the rule even though it will still appear in the list until deleted. Unused rules at the end of the list will always show none as the type. If you wish to prevent these from being displayed, reduce the number of rules enabled.

The number of rules enabled simply limits the scope of rule review so that you do not

have to review a lot of unused rules. If the displayed rules are used up and you need more, increase the enabled number.

9.3 SNMP Client Write Maps (Set)

Writing or Setting data to another SNMP device requires setting up a Write Map.



Start by selecting the local register that will be the source of data to Set in the remote device. Enter the OID to write in that device. Select the data format that will be expected by that device at the OID given. Select a device from the device list. Only devices entered on the Devices tab will appear in this list.

Click on the Map number in the first column to access the expanded view of the Write Map where additional optional parameters may be entered.

The local register data may be written to the remote SNMP device periodically, or

when it changes, or both. To send upon change (send on delta), check the first box and enter the amount by which the local register must change before being written to the remote device. To guarantee that the remote OID will be written at least occasionally even if the data does not change, check the second box and enter some amount of time. This time period will be referred to as the "maximum quiet time".

Data from the local register may be manipulated before being written to the remote OID. The local data is first multiplied by the scale factor. The offset is then added to it.

Enter the OID to write in that device. Select the data format that will be expected by the selected device at the OID given. Select a device from the device list. Only devices entered on the Devices tab will appear in this list.

The repeat time may determine how often the remote OID will be written. If send on delta and maximum quiet time are not checked above, clicking the "at least" button will establish a periodic update time. If send on delta is used and you wish to limit the network traffic in the event changes are frequent, click the "no more than" button and enter the minumum time that should elapse before another write to the remote device. It is valid to select "no more than every 0.0 seconds" if you want all changes to be sent, but no periodic writes.

You have the option of making this Write Map conditional. If an index register number is provided and the Enable box is checked, then this write map will only be executed when the index register (local register) contains the value given. This allows multiple write maps to supply data to the same remote OID based on the value of the local index register. It also allows writing to simply be suspended if a single write map supplies data to the remote OID. In a more sophisticated scenario, you could potentially suspend writing of the remote device if you know the device is powered down.

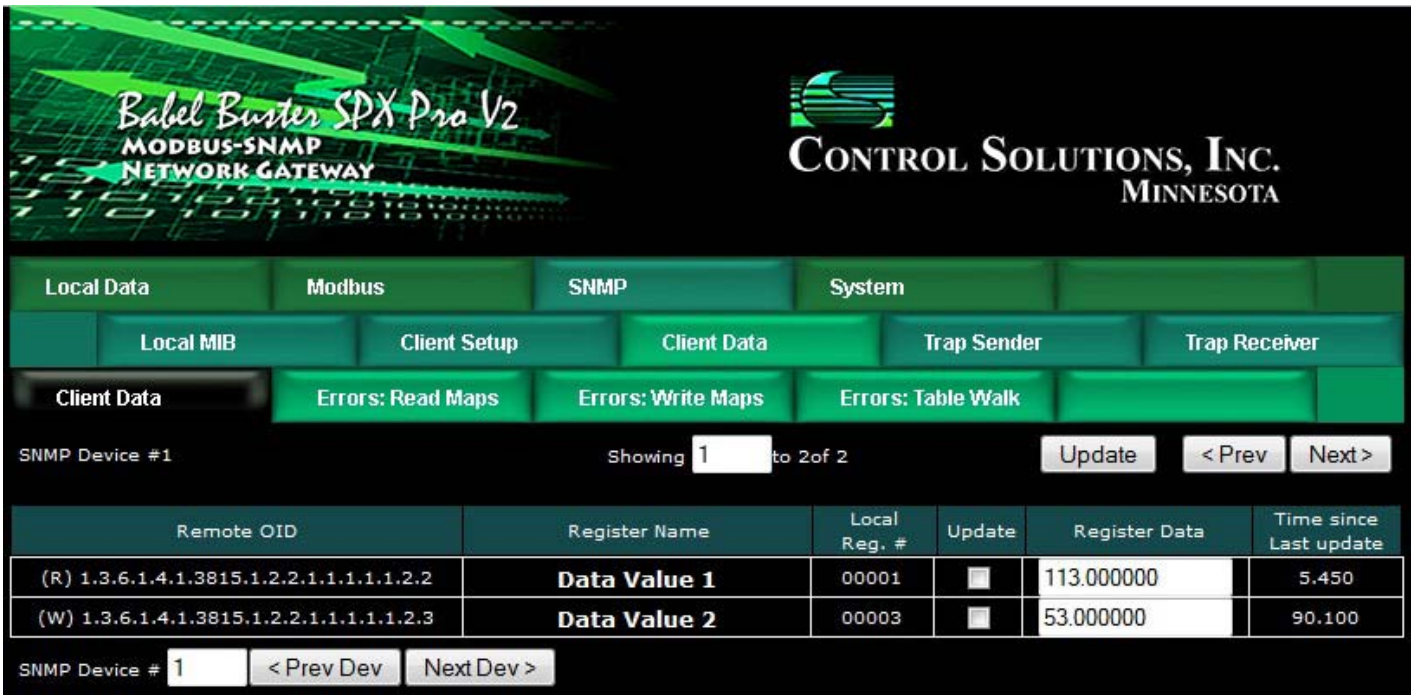
Delete will remove the rule number shown in the "Map #" box. Insert will insert a new rule before the rule number shown, and is used for placing rules between existing rules. It is not necessary to use Insert to add rules to the bottom of the list or to define any rule presently having zero for a source register or "none" for remote type.

Selecting "none" for remote type effectively deletes the rule even though it will still appear in the list until deleted. Unused rules at the end of the list will always show none as the type. If you wish to prevent these from being displayed, reduce the number of rules enabled.

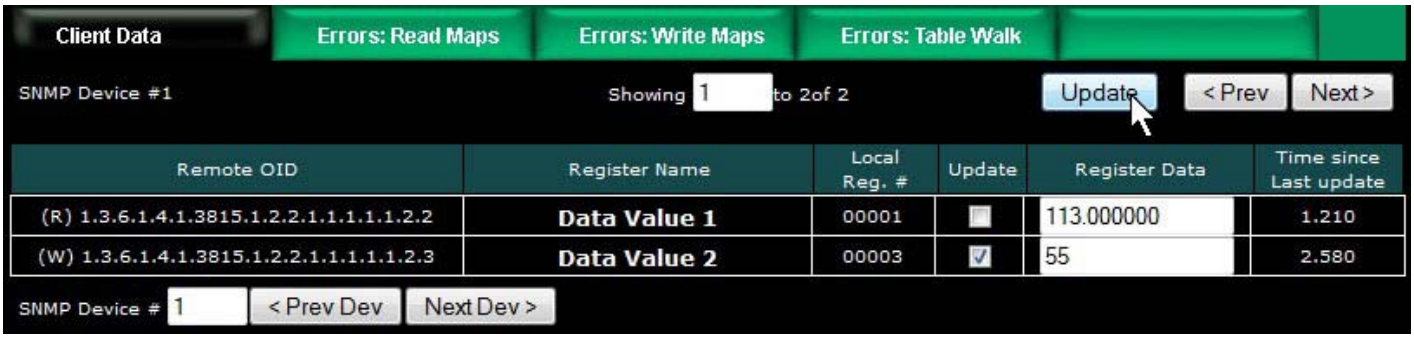
To test Setting the mapped OID, you can go to the Local Registers page, enter a new value for the local register, check Set, and then click Update. If monitoring traffic in Wireshark, you should see the request go out.

9.4 SNMP Client Data Displayed by Agent

The Client Data page shows the list of local registers mapped to a remote SNMP device.



Click the Next Dev or Prev Dev buttons to go to the next or previous SNMP device, or simply enter a number in the SNMP Device # window and click Update. Maps for that device will now be displayed. In addition to a summary of the map (both read and write maps are shown), the time since last update is displayed. This time should generally be less than the poll time. If the last update time is large, it may mean there is an error preventing the update.



To test sending a Set request to the remote SNMP device, enter a new value for Register Data, check the Update box, and then click the Update button. This can also be done from the Local Registers page.

9.5 Supported Data Formats

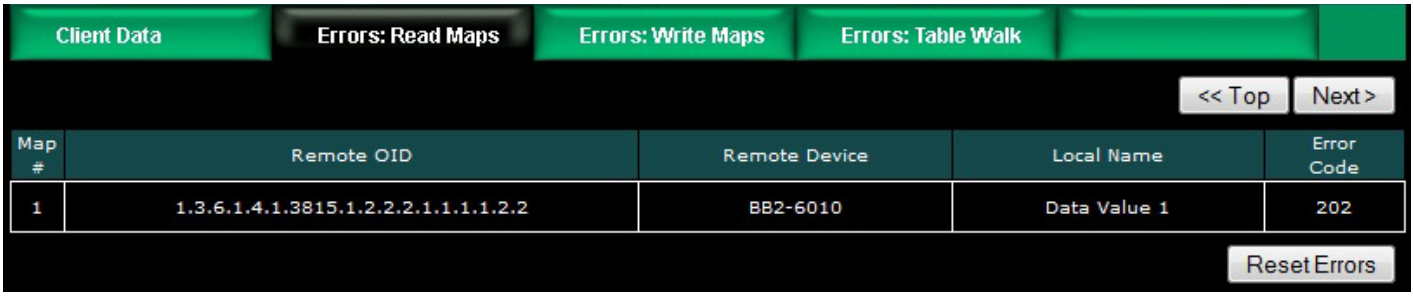
SNMP does not have a universally accepted representation for floating point. The one universally known data type is INTEGER. A commonly recomended means of transmitting floating point data is either as a scaled integer or as an ASCII character string. A well known but application specific implementation (NetSNMP) uses ASN OPAQUE FLOAT. There does exist an RFC 6340 for representation of floating point. Both the NetSNMP and RFC 6340 versions are based on IEEE 754 encoding. The "Float 32-bit" and "Float 64-bit" data types in the Babel Buster Pro's own MIB refer to RFC 6340 encoding.

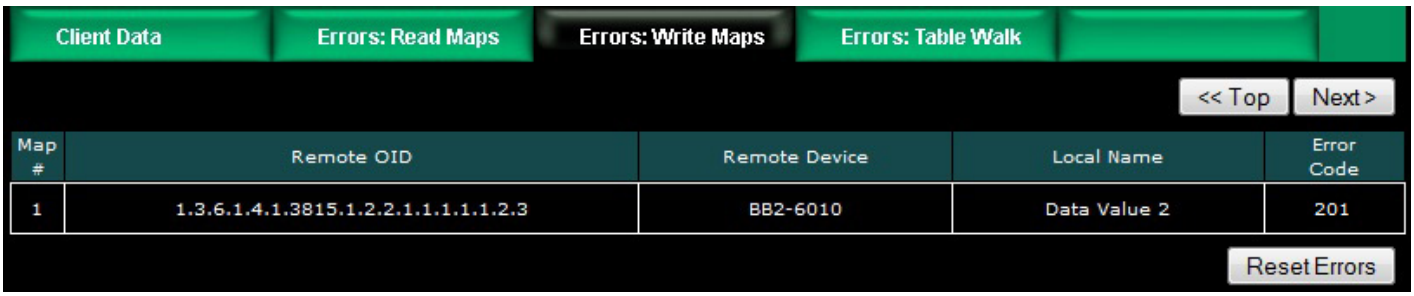
Data types recognized by Babel Buster Pro when Getting or Setting SNMP data in other devices are listed in the table below. Specifically, the data types are encoded with ASN types as follows:

Integer 32-Bit	INTEGER	ASN_INTEGER
Unsigned 64-Bit	COUNTER64	(ASN_APPLICATION | 6)
Float 32-bit	OCTET STRING	ASN_OCTET_STR (length 4)
Float 64-bit	OCTET STRING	ASN_OCTET_STR (length 8)
Char String	OCTET STRING	ASN_OCTET_STR (length variable)
Counter	COUNTER	(ASN_APPLICATION | 1)
Unsigned 32-bit	UNSIGNED	(ASN_APPLICATION | 2)
Float-Opaque	OPAQUE FLOAT	(ASN_APPLICATION | 8)

9.6 SNMP Errors

The errors pages will show Read Maps and Write Maps that currently have an error status.





Error codes that may be returned by the SNMP client are as follows:

Standard SNMP error codes returned by remote Agent:
1 = SNMP_ERROR_tooBig
2 = SNMP_ERROR_noSuchName
3 = SNMP_ERROR_badValue
4 = SNMP_ERROR_readOnly
5 = SNMP_ERROR_genErr
6 = SNMP_ERROR_noAccess
7 = SNMP_ERROR_wrongType
8 = SNMP_ERROR_wrongLength
9 = SNMP_ERROR_wrongEncoding
10 = SNMP_ERROR_wrongValue
11 = SNMP_ERROR_noCreation
12 = SNMP_ERROR_inconsistentValue
13 = SNMP_ERROR_resourceUnavailable
14 = SNMP_ERROR_commitFailed
15 = SNMP_ERROR_undoFailed
16 = SNMP_ERROR_authorizationError
17 = SNMP_ERROR_notWritable
18 = SNMP_ERROR_inconsistentName

Client generated errors:
201 = No response from remote Agent (server)
202 = No such name (implicit)
203 = Unable to interpret application data
204 = Reply does not match request
205 = There is a problem with the rule configuration
206 = Unable to build or parse SNMP PDU
207 = Table walk came up short of expected count

If the error code indicates no response, the device status (Client Setup Devices page) will provide an additional indication of a connection related error.

Device status may indicate the following application level error conditions:
10 = could not bind socket
11 = response timed out (may mean IP address given is not reachable)

Device status may also indicate error codes from 80 and up are IP stack errors. Since SNMP uses UDP, which is connection-less, the only likely errors are:

95 = Network is unreachable
107 = Host is unreachable
108 = Host is reachable, but expected port cannot be opened

10. Configuring SNMP Table Walker

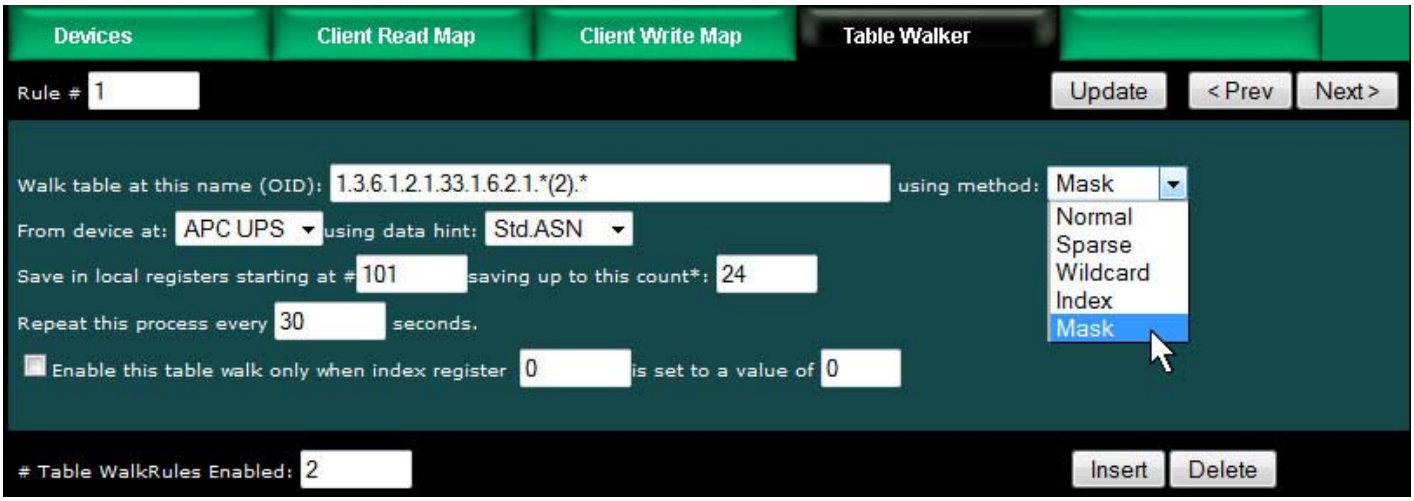
10.1 Table Walk Methods

Visualize the SNMP table as a spread sheet. Although there can technically be any number of dimensions to an SNMP table, this table walker is limited to 2-dimensional tables, just like a spread sheet. The table consists of some number of columns each of which are identified by an OID. The table contains any number of rows, which are identified by a table index. The table index, or row number, simply appears as the last field in the OID (or in other words, append the row number to the OID that identifies the column).

The Babel Buster table walker will scan the table, picking out one column per walk rule. For each row in the table, that column will be retrieved and its data placed into the local register provided. The first data value retrieved will be placed into the starting local register, and the walk will continue until the number of registers specified by "count" are filled.

The table walk method will default to "Normal" if you use only the quick entry of a simple table walk rule. This means you enter a starting OID, select a device, enter a starting register and number of local registers to fill with table content, and set a poll rate (meaning how often to periodically walk the table).

To select any table walk method other than Normal, you need to click on the table walk rule number in the first column, and select the method on the expanded view of the walk rule.



The table walking process can become complicated by the fact that tables are allowed to have missing rows, or rows may have intermittent missing columns. In some applications, rows will be populated only temporarily and then they will disappear. The table walk method identified as "Normal" expects the given column to exist in all rows through the range of rows defined by the starting OID and the count. The remaining methods accommodate the other complications permitted by SNMP.

Method "**Normal**" will simply produce a 1 to 1 correlation between table entries and register numbers, placing successive values in successive registers. Data will be interpreted according to the data hint if an octet string is returned, otherwise the ASN encoding will take precedence. If the Get-Next sequence fails to return enough OIDs to fill the 'count' criteria, an error code is set for the device indicating that the table came up short on data.

Method "**Sparse**" is the same as Normal, except missing OIDs in the sequence is anticipated, and the corresponding local registers in the sequence are skipped over if the respective OID is not included in the Get-Next sequence. No error is flagged for table being short on data.

Method "**Wildcard**" allows wildcard fields in the table OID. The walk does not care about order of OIDs returned by Get-Next as long as they match the OID given after discounting wildcard fields. No attempt is made to sequentially pair OIDs with registers. The next 'count' OIDs that successfully match the OID with wildcards will fill the next 'count' of registers beginning with the starting local register number. The OID sent out in the first Get-Next request will have zero in any wildcard fields, and each successive Get-Next will send out the OID from the response to the previous Get-Next.

Method "**Index**" will walk the table, but expect to find that values are OIDs. In other words, the table name is an OID, but the contents of the variable at that name will be another OID. The result is that the OID index (last field of OID) from the value will be used as the offset to calculate which register number is to be affected, and that register will be set to 1 indicating this OID is present in the table. This seemingly odd means of table walking is requiired in order to translate the alarm table from RFC 1628 for UPS systems into indexable Modbus registers that indicate the presence or

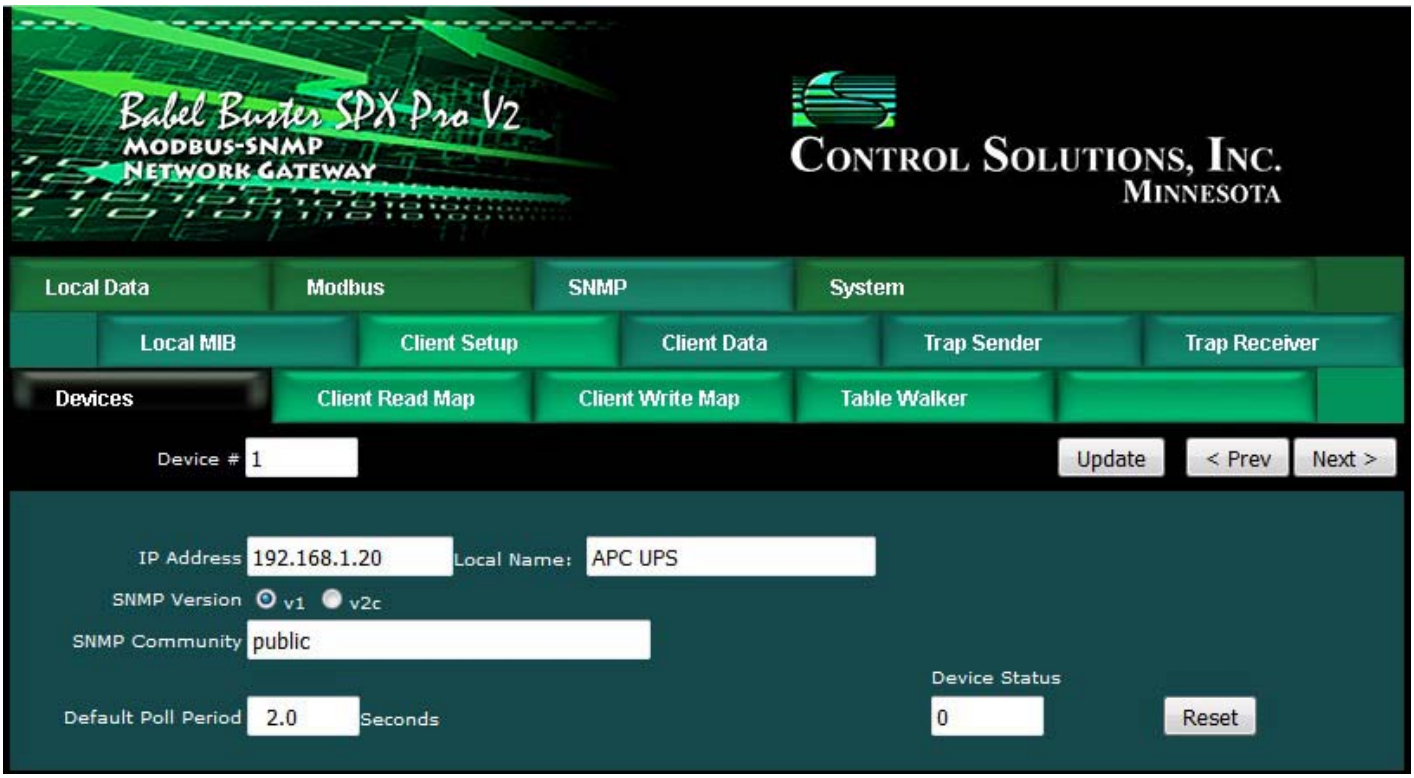
absence of alarms defined in RFC 1628. (Note that the alarm table in RFC 1628 is "sparse" meaning its table entries are only present if an alarm is present, and the table is empty if there are no alarms. One cannot simply query OIDs to determine presence of an alarm. Alarms are implied by presence of a table entry that only exists while the alarm is present. Furthermore, the alarm table is simply a circular buffer of alarm entries, and the table index means nothing.)

The register set to 1 by the Index method will not be reset to zero by anything in the table walk rule. Use the timeout feature available in the definition of the register itself to set it to a default value after some amount of time, typically a time longer than the rate at which this table is walked. The result will be a register that is set to 1 when the corresponding alarm exists, and automatically reset to zero sometime after the alarm no longer is found in the alarm table.

Method "**Mask**" is a modified version of "Index". It will walk the table in the same manner, but set bits within the same single register corresponding to the OID index field. OID index .1 will be bit 0, index .2 will be bit 1, and so on. In this case, starting register is the only register number affected, and "count" is the number of bits that will be affected in that one register. If the Index method is used and one wishes to recognize the full range of alarms known to RFC 1628, then 24 registers will be consumed. Using the Mask method, only one 32-bit register is used (technically 2 Modbus 16-bit registers). Furthermore, non-present alarms will have their corresponding bits cleared without any timeout default value. (Timeout and default should not be used in this case since the single register represents as many as 24 individual states.)

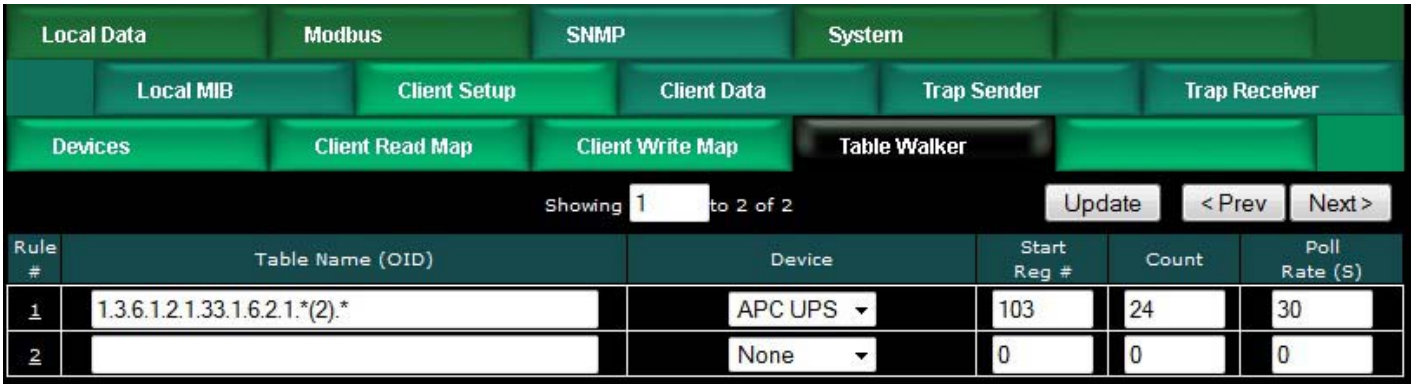
10.2 Configuring Table Walk Rules

The table walk rules apply to a specific SNMP device whose table is to be walked. The first step in creating a table walk is to enter the device information on the Devices tab. The same set of devices are used for SNMP client read and write rules.



Enter the IP address of the SNMP device, and provide a local name. This local name is only used to populate the device list on the rules pages. The SNMP version will be either v1 or v2c for this gateway (v3 is not supported here - v3 requires a different model). This determines which version is indicated in the Get-Next requests generated by the tabel walker. The community should match whatever the SNMP device to be queried is expecting to see. The default poll rate is used whenever the rate in the individual rule is left at zero.

Next, proceed to the Table Walker page. The page that opens initially is a tabular list of multiple rules. You can enter everything necessary for a "Normal" method table walk using just this tabular form. However, if you will be using wildcards in the OID or any method other than Normal, you must click the rule number in the first column and set up the rule using the expanded form. Upon clicking a rule number, the expanded form of the rule is displayed. To return to the tabular list, click on the Table Walker tab again.



The parameters for a table walk rule are as follows. Only the table name, device,

starting register, count, and poll rate are entered on the tabular list of rules. To enter the additional parameters or use a method other than "Normal", the expanded view must be used.

Table Name or OID: The walk will begin at the variable name or OID given. It is not necessary to start at the beginning of a table. To walk a section of the table, enter an OID whose last field is the desired first index, minus one. The count will limit the scope of the walk.

The table name OID includes optional wildcard fields when the walk method is Index or Mask. For example, to walk the alarm table of a UPS using RFC 1628, the OID would be 1.3.6.1.2.1.33.1.6.2.1.*(2).*. The *(2) means allow any value in the second to last field but only act on the value when this field is 2. The last asterisk means disregard the last field entirely (which for RFC 1628 increments with each new alarm until reaching some rollover point, then starts back at 1 again).

Method: Select method as defined above in section 10.1.

Device: Select an SNMP device from the list created in the Devices page.

Data Hint: Data hint provides instruction for interpretation of data in the event the value is an Octet String. If the data is an ASN type that is clearly recognizable, the ASN encoding will take precedence. If the Octet String should be interpreted per RFC 6340 for floating point, select RFC 6340. Otherwise, leave the "Std. ASN" selection as is. An Octet String will be treated as a character string if the destination local register is defined as a character string. Otherwise, an attempt will be made to convert the Octet String as if it were an ASCII representation of some number. If the destination is not a character string register and the octet string contains no numeric information, the result will be zero.

Starting Local Register and Count: The "local registers starting at #" is the first Modbus register to be filled by this table walk, and "count" is the number of registers in the sequence that are to be affected. Count is a number of register instances, not strictly 16-bit address space increments. Therefore, if the starting register number points to the first of several floating point registers, and count is 2, a total of four 16-bit official Modbus registers (2 register pairs, or 2 floating point values) will be filled by the table walk. The count also determines how many OIDs in the table will be read since there is a 1 to 1 correspondence between table entries and local register values saved.

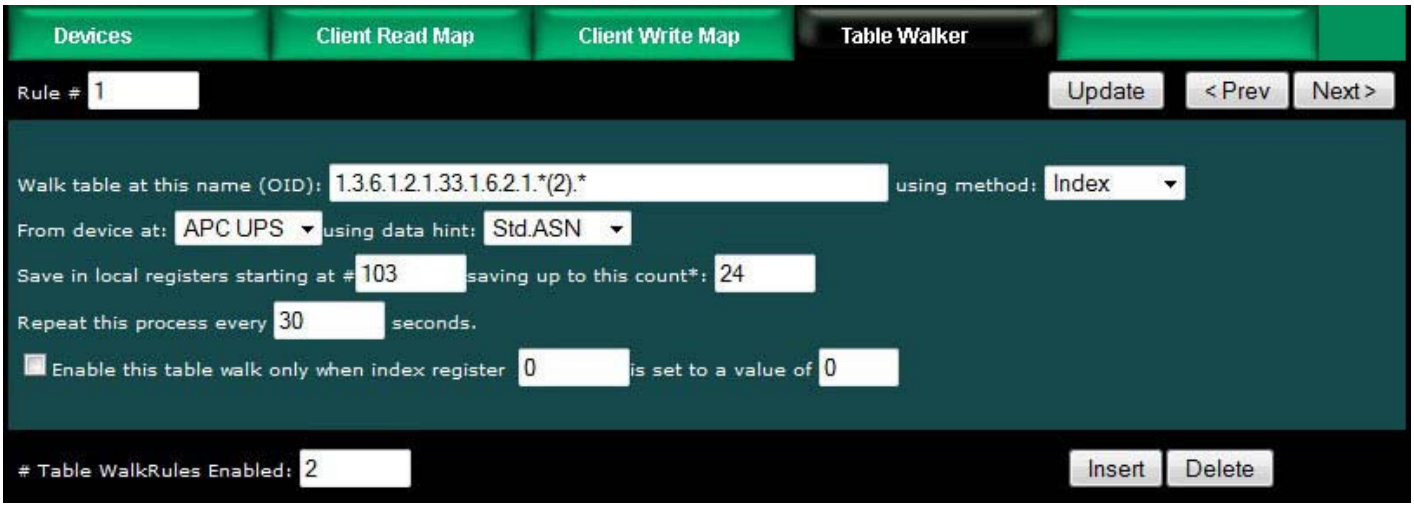
* Count has a different meaning when method is "Mask". In this case, the starting register is the only register affected, and count is the number of bit positions in that register that will be processed. It also implies that this many OIDs in the table will be read.

Poll Rate (Repeat...): This sets the rate at which the table should be walked. The amount of time to wait between table walks is entered. Be prudent with poll rate - this will generate a lot of network traffic.

Enable Index Register: You have the option of enabling this rule only when a selected register contains a given value. Any local register may be used as the index register. As the name implies, you could have the same local registers contain different values based on different rules as indexed by the index register.

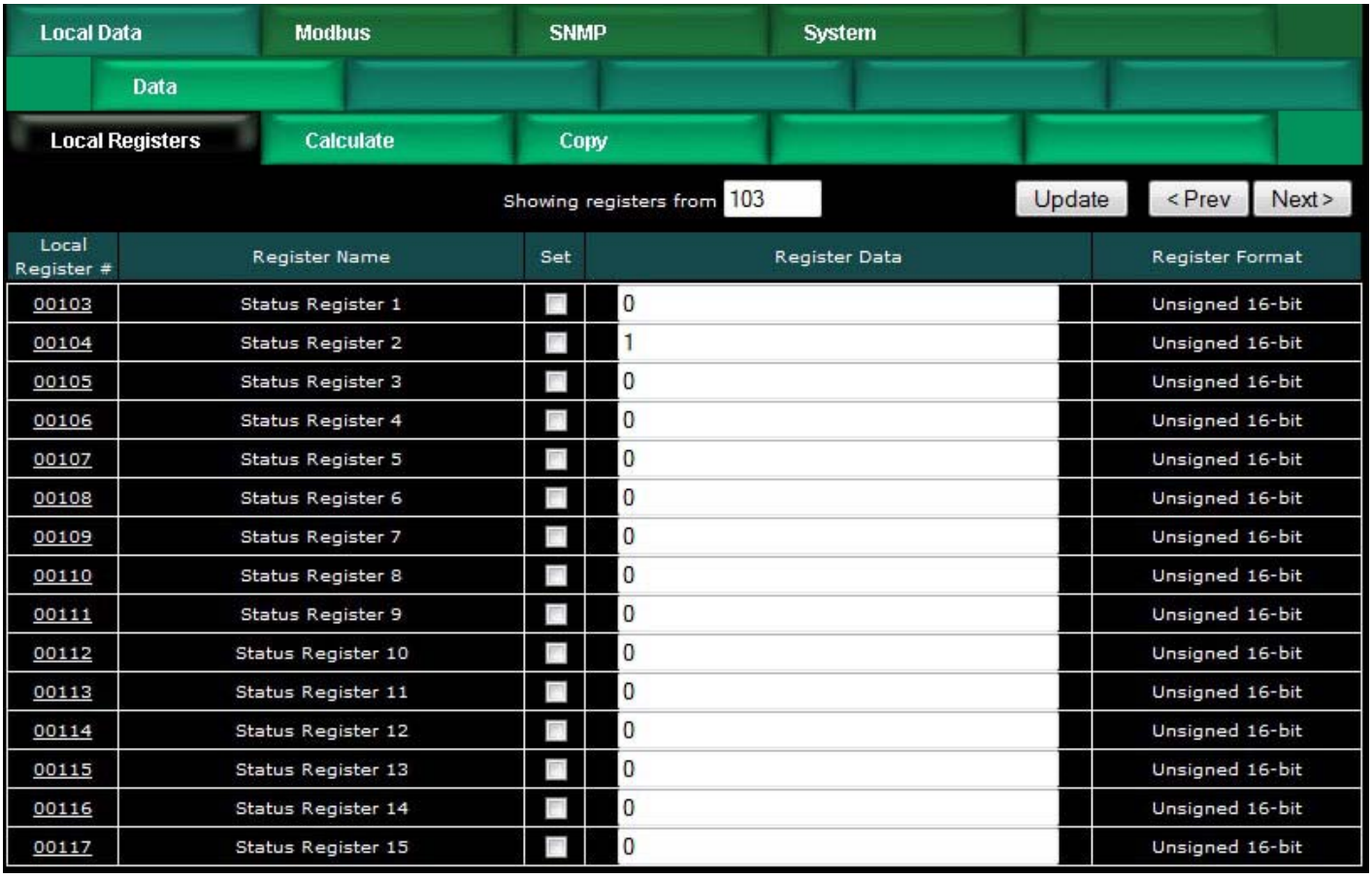
Click Update to register your changes. Don't forget to save all changes in the configuration file when done. To insert a rule before an existing rule, enter the existing rule number in the "Rule #" window at the top, and click Insert. To delete an existing rule, enter the rule number in the "Rule #" window and click Delete.

Table Walk Example 1:



The table walk rule illustrated above will walk the alarm table in a UPS that implements RFC 1628. Using the Index method, registers named "Status Register 1" through "Status Register 24" in our test case will be set to a value of 1 when the corresponding alarm condition is present. RFC 1628 defines 24 "well known" alarm conditions. Alarm condition 2 is defined to mean "UPS is on battery". The screen shot below illustrates the UPS being on battery, with no other alarms currently present.

Note that while the minimum register size you can allocate in the Babel Buster Pro is a 16-bit register, you can read the registers as Modbus coils if the gateway is operating as Modbus slave. In the example below, holding register 104 will return a value of 1, but coil 104 will also return a status of "on" or 1. Likewise, you can read these status registers as discrete inputs or input registers from Modbus. (If reading a value greater than 1 as a coil or discrete input, any non-zero value will still return just "on" or 1.)



The Index method of table walk will set registers corresponding to OIDs found that reference the "well known alarms" defined in RFC 1628. Registers are skipped if no corresponding OID is found in the table. That means the Index walk will set registers but never clear them. To get the registers to reset after being set to indicate an alarm, the timeout feature of the register itself must be used. As illustrated below, the status register will be set to a default value of zero "if not updated..." within 45 seconds. The table walk in this case is set to repeat every 30 seconds. If the alarm is still present, it will be set once again. The timer is reset every time the table walk writes to this register. But this periodic setting of the register will stop if the alarm is no longer present, and then the 45 second timeout will clear or reset the register. The "updated by remote source" can be any of several things, but in this case, it will be the table walk that is considered to be the remote source.

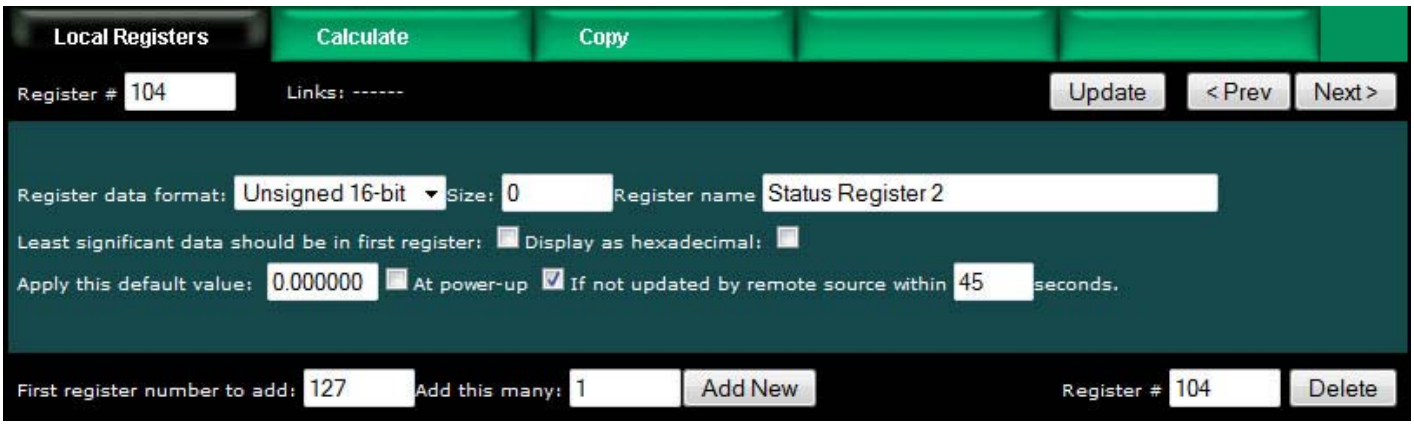
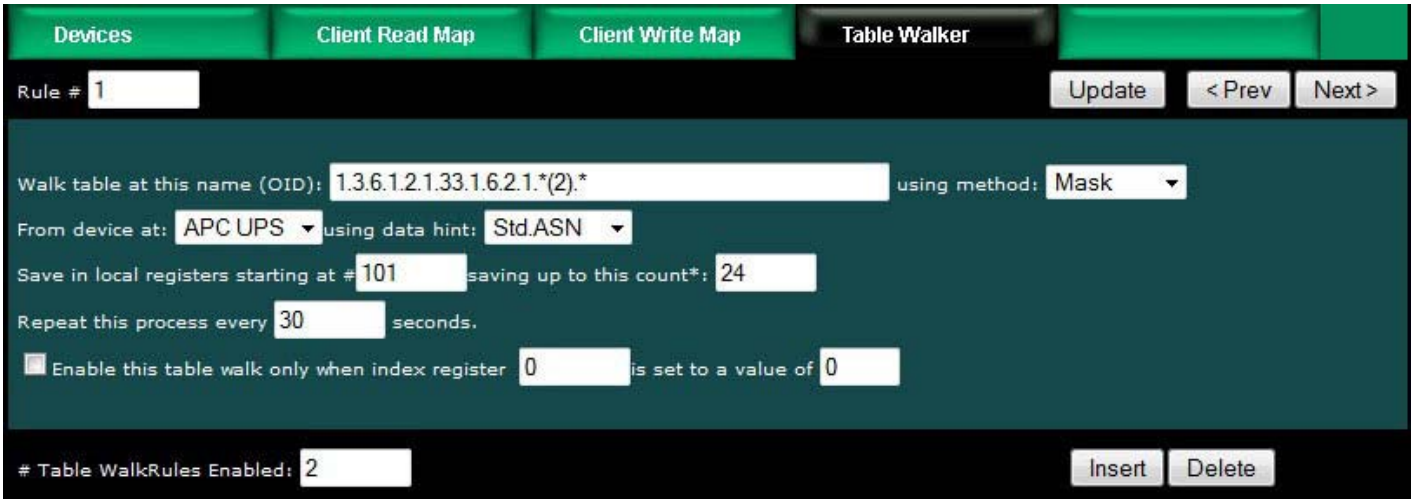
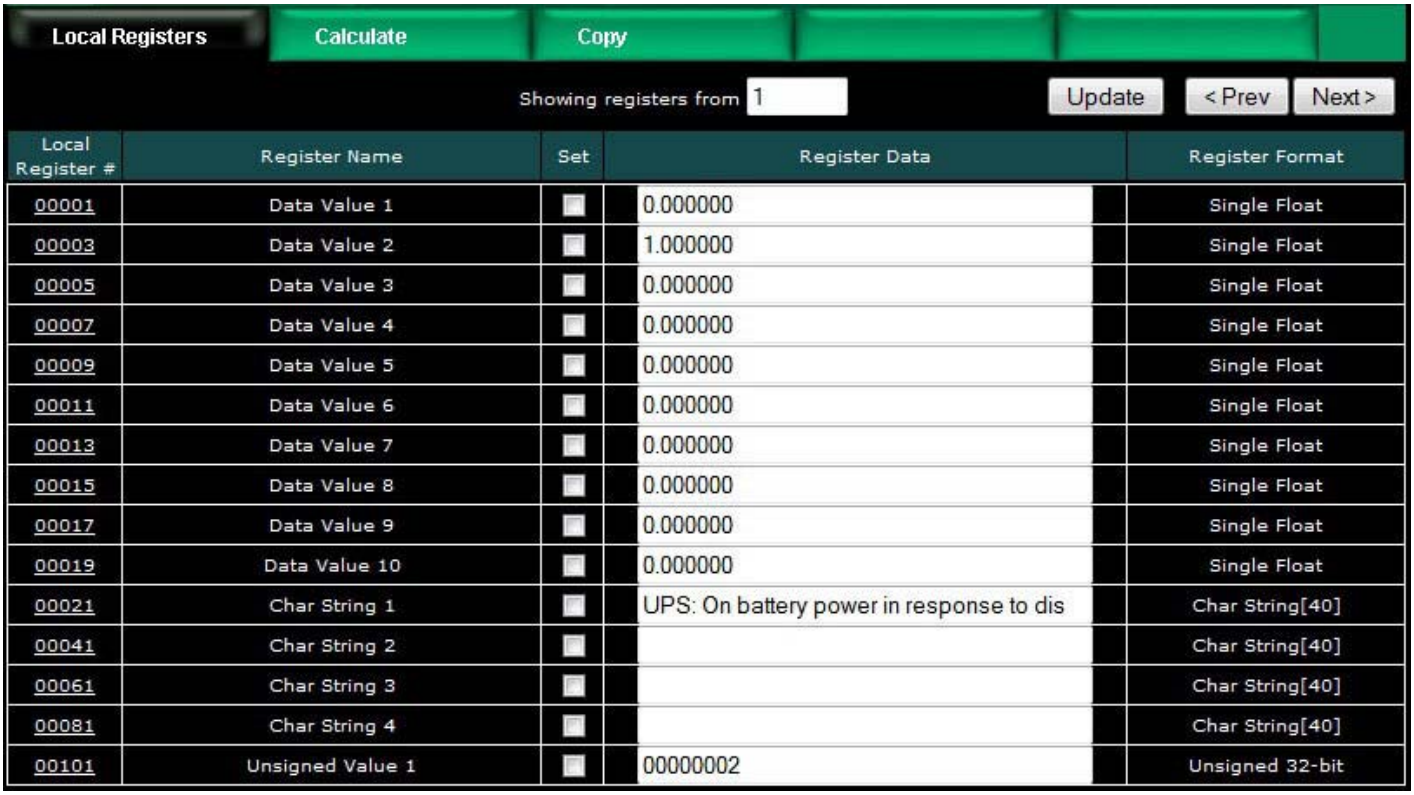


Table Walk Example 2:

The walk rule in this example is identical to the first example, with one exception: The method is Mask. Instead of using 24 different registers to reflect the states of the 24 well known alarms, a single register is used with one bit per alarm packed into this register. The single register referenced should be locally defined as a 32-bit unsigned integer register.

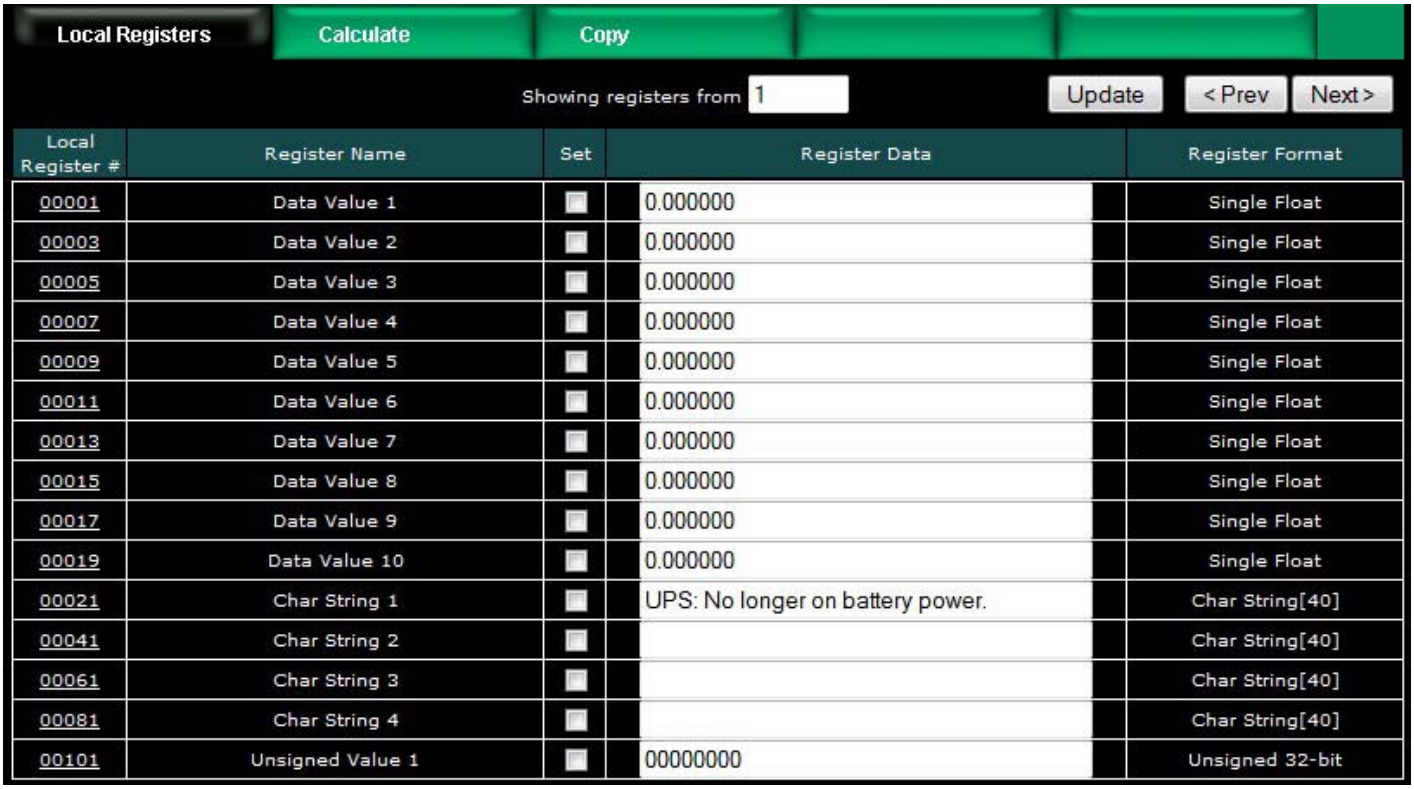


The resulting "UPS on battery" alarm, being alarm #2, sets bit 1 in the packed bit register 101 in our example. Alarm #1 sets the least significant bit, alarm #2 the next bit, and so on.



As noted in the discussion of methods, the Mask method does not require any timeout default value to be used to clear the packed bit (or bit mask) register. Each time the

table is walked, the content of the register is replaced with entirely new content, which will result in bits getting cleared if the respective alarm is no longer active.



Local Registers | Calculate | Copy

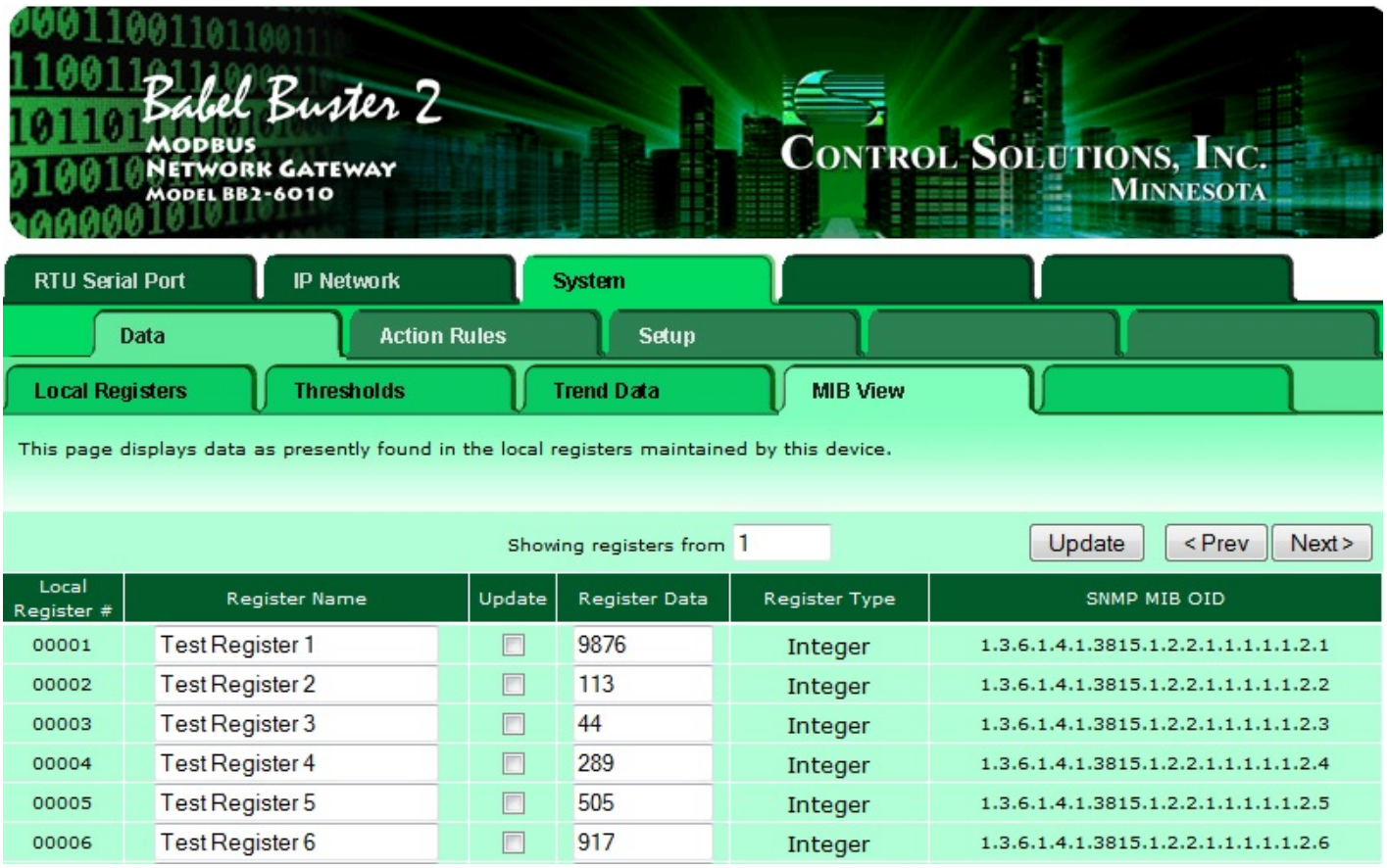
Showing registers from 1 Update < Prev Next >

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1		0.000000	Single Float
00003	Data Value 2		0.000000	Single Float
00005	Data Value 3		0.000000	Single Float
00007	Data Value 4		0.000000	Single Float
00009	Data Value 5		0.000000	Single Float
00011	Data Value 6		0.000000	Single Float
00013	Data Value 7		0.000000	Single Float
00015	Data Value 8		0.000000	Single Float
00017	Data Value 9		0.000000	Single Float
00019	Data Value 10		0.000000	Single Float
00021	Char String 1		UPS: No longer on battery power.	Char String[40]
00041	Char String 2			Char String[40]
00061	Char String 3			Char String[40]
00081	Char String 4			Char String[40]
00101	Unsigned Value 1		00000000	Unsigned 32-bit

Table Walk Example 3:

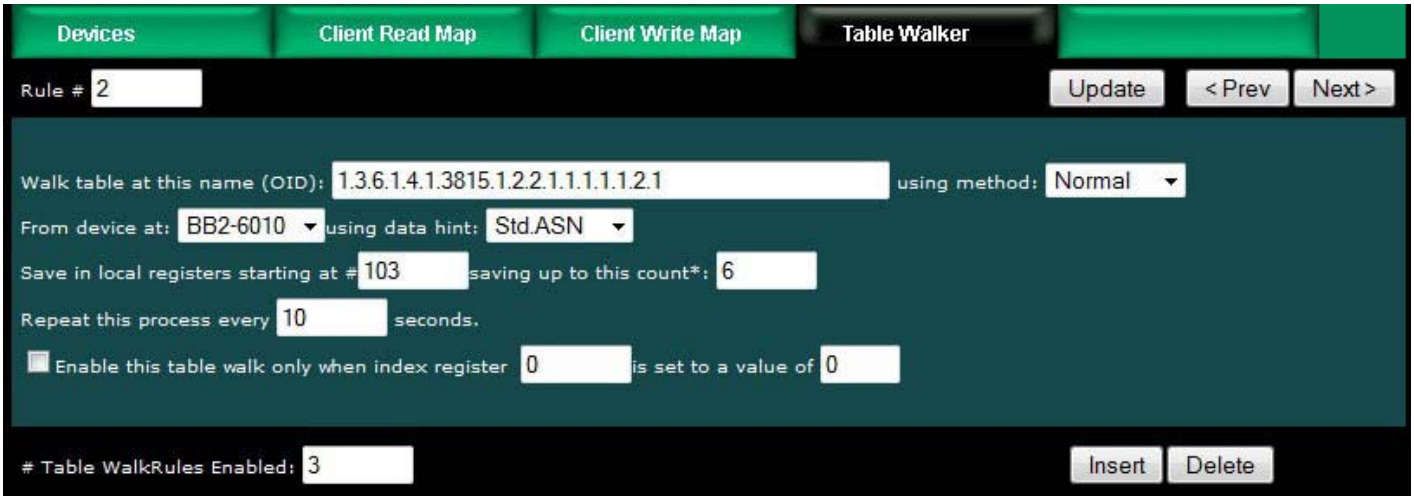
The next example is a simple table walk using the Normal method, included here for completeness of discussion. The real power of the Babel Buster Pro is in its ability to walk the types of tables that require more complex methods. The Normal table walk illustrated here does not need to be a table walk at all. It could just as easily be a set of SNMP client read map rules. The client read rules would periodically Get the values named. Likewise, the Normal table walk periodically Gets the values named. The problem with relying solely upon periodic polling is that in applications like RFC 1628, alarm OIDs only exist while the alarm is active and any attempt to do a direct Get at other times results in an error. Such a problem is solved by the other methods available for table walking in the Babel Buster Pro.

To illustrate the Normal table walk, we will get values from the contiguous full table provided by another gateway, the BB2-6010. A screen shot of its MIB View shows the OIDs we will retreive with our table walk.



The screen shot of the walk rule list shows the previous example plus this example added as a second rule. Because this is a simple "Normal" table walk, everything needed can be entered on the tabular rule list. The first OID to be read is given as the table name. Select the device, starting local register and count, and poll rate. That is all that is needed. (Note: The device in this example has already been set up over on the devices page, first tab.)

If you were to click the rule number (2) to look at the expanded form of this rule, here is what you would see. No additional entries are required to make this table walk functional.



Given the screen shot of the BB2-6010 MIB View, here is what we expect to see in the Babel Buster Pro when the table walk starts running.

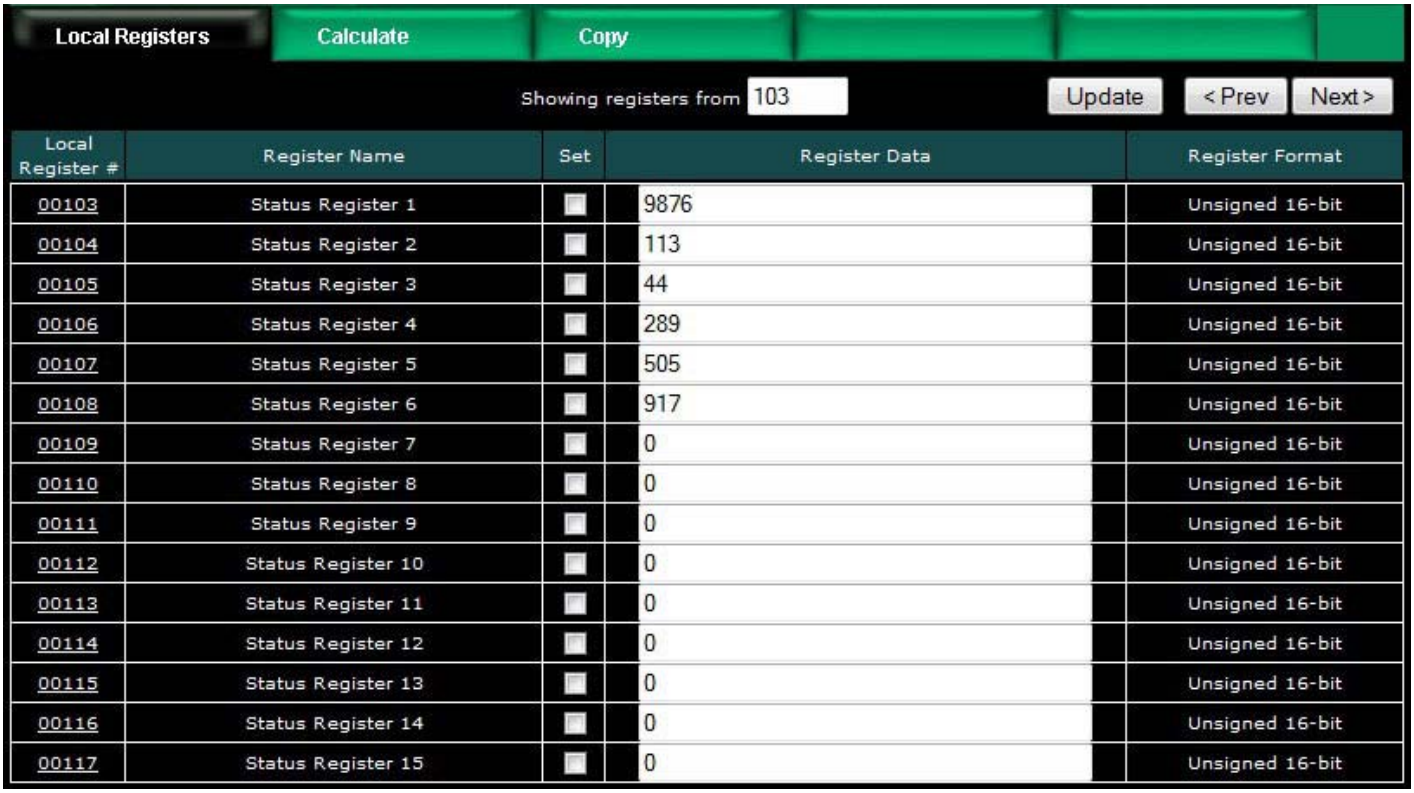
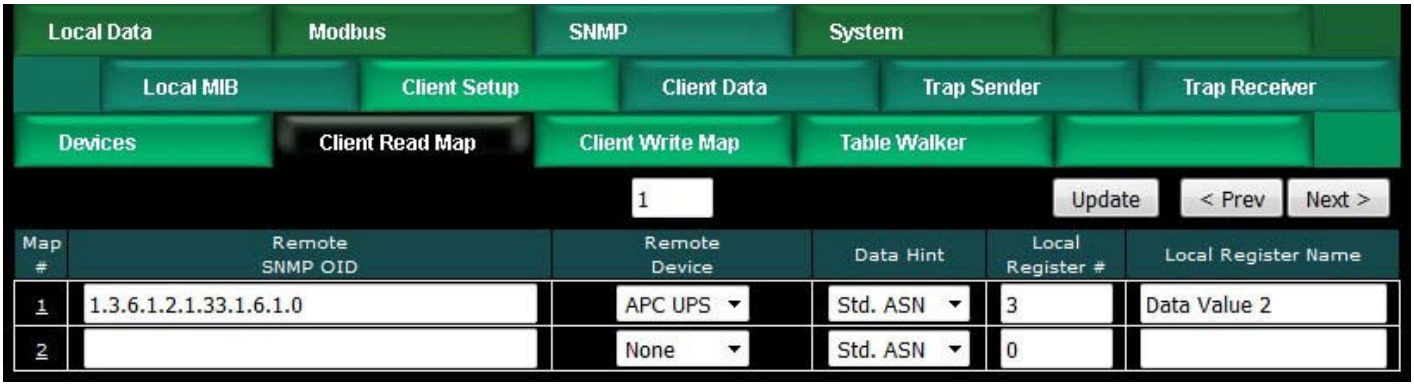


Table Walk Example 4:

The following example elaborates on the RFC 1628 alarm table walk originally illustrated above. We are now going to walk the alarm table conditionally. Namely, we will only walk the table when we know there are alarms in the table. There is a variable in the RFC 1628 MIB that tells us how many alarm entries are currently in the table. This is found at the OID illustrated below, and we have set up an SNMP client read map to retrieve that count, and place it in local register 3.



The index register in the table walk rule wants to see a specific number. Since there may be zero to many alarms, we need a way of simply reducing that to an effective "yes or no" to the walk rule. We accomplish this by simply testing to see if the alarm count is greater than zero by using a Calculate rule. The result of the test will be 0 (false) or 1 (true) with the result placed in local register 5.

Now that we have a register whose value will be 1 only when one or more alarms are in the RFC 1628 alarm table, we can enable the index register in the table walk rule as illustrated below. We have now reduced network traffic by only walking the alarm table when we know alarms are present.

This means of limiting when the table is walked can apply to either of examples 1 or 2 above. What is pictured here is a continuation of example 2. The Mask method does

not normally require the default timeout for clearing the alarm bits in the results register. However, if the table walk is halted as soon as there are no alarms, then there needs to be another means of resetting all of the alarm bits. Do this by setting the default timeout as illustrated in example 1.

10.3 Table Walk Errors

If the table walk is unsuccessful, the table OID, device, and an error code are displayed here for the walk rule that ran into trouble.

The following errors are possible. Except for error code 207, all of these error codes apply to any SNMP activity includiing client Get or Set activity defined by the SNMP client read and write rules. Error 207 applies only to table walking, and means that the Normal method of table walking did not find enough OIDs in the table to satisfy the requested count of local registers to fill.

Standard SNMP error codes returned by remote Agent:
1 = SNMP_ERROR_tooBig
2 = SNMP_ERROR_noSuchName
3 = SNMP_ERROR_badValue
4 = SNMP_ERROR_readOnly
5 = SNMP_ERROR_genErr
6 = SNMP_ERROR_noAccess
7 = SNMP_ERROR_wrongType
8 = SNMP_ERROR_wrongLength
9 = SNMP_ERROR_wrongEncoding
10 = SNMP_ERROR_wrongValue
11 = SNMP_ERROR_noCreation
12 = SNMP_ERROR_inconsistentValue
13 = SNMP_ERROR_resourceUnavailable
14 = SNMP_ERROR_commitFailed
15 = SNMP_ERROR_undoFailed
16 = SNMP_ERROR_authorizationError
17 = SNMP_ERROR_notWritable
18 = SNMP_ERROR_inconsistentName

Client generated errors:
201 = No response from remote Agent (server)
202 = No such name (implicit)
203 = Unable to interpret application data
204 = Reply does not match request
205 = There is a problem with the rule configuration
206 = Unable to build or parse SNMP PDU
207 = Table walk came up short of expected count

If the error code indicates no response, the device status (SNMP Client Setup Devices page) will provide an additional indication of a connection related error.

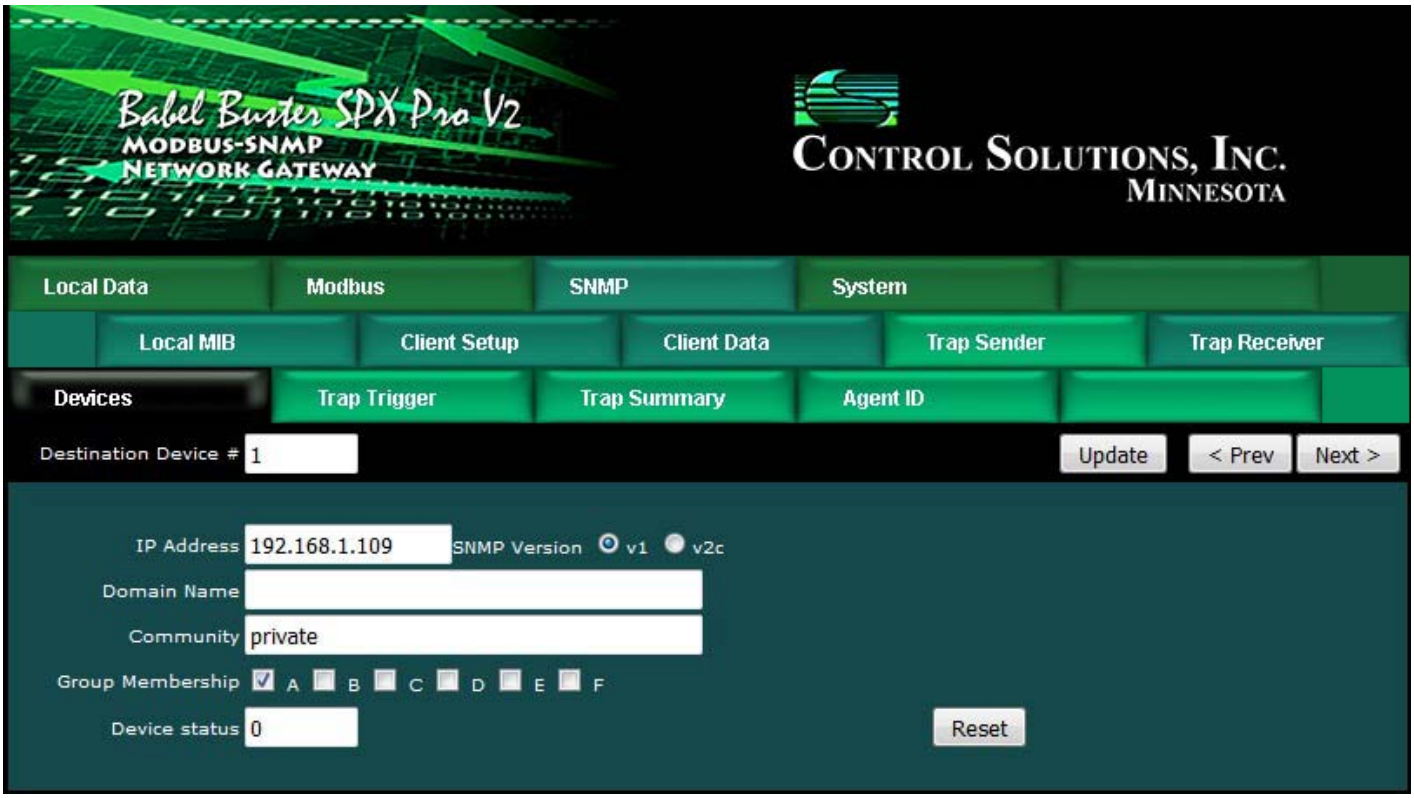
11. Configuring SNMP Trap Sender

11.1 SNMP Trap Destinations

The first step in sending traps is to tell Babel Buster Pro where to send them. This is done on the Devices page of the Trap Sender (which is not the same list of devices used by the Trap Receiver or SNMP Client).

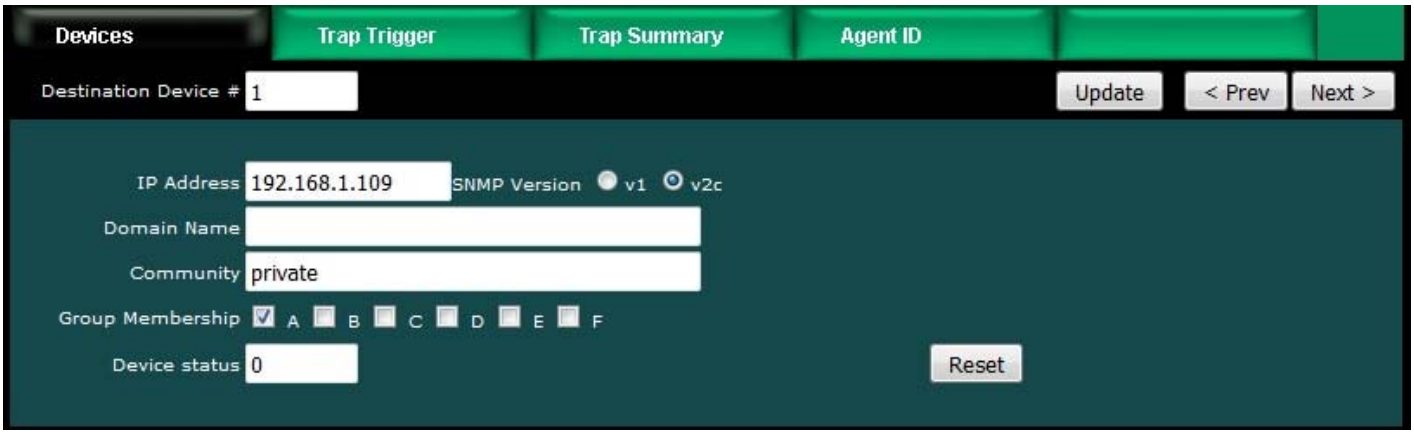
For each destination that traps should be sent to, enter the IP address, select v1 or v2, and provide the community string that should be used when sending traps to that device.

Be sure to select one or more groups for membership for this device. The device must be a member of the group before traps assigned to that group will be sent to this device. This membership method allows different traps to be sent to different devices if desired.



The structure of a v2 trap is noticeably different than a v1 trap, but the only difference you need to take note of here is simply selecting one or the other for the given IP

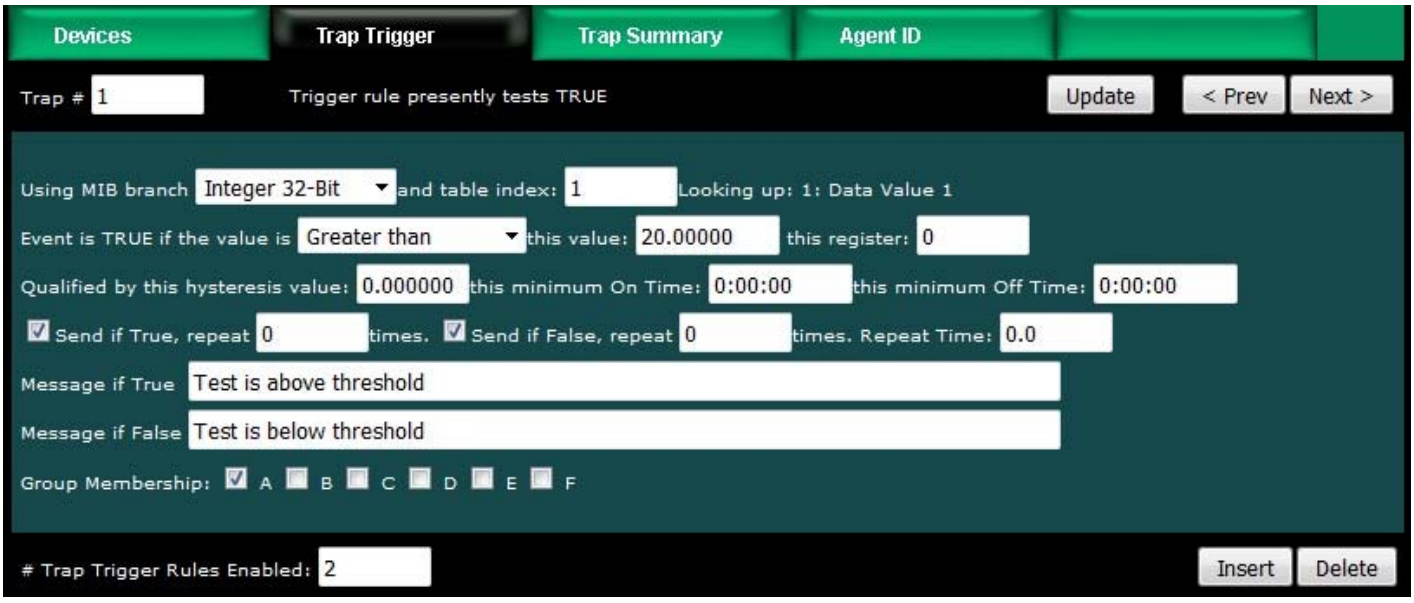
address.



11.2 SNMP Trap Triggers

Create a Trap Trigger for each trap that should be sent.

A trap must reference a variable in the local MIB. Therefore, before you can create a trap trigger, you must assign the local register of interest to a spot in the local MIB. (See "Configuring Gateway as an SNMP Server".) Once the local register has a home in the MIB, select the branch and MIB index in the trap trigger rule. Do not enter local register number here. Enter the MIB table index from your local MIB configuration pages.



Once the variable of interest is selected, define the threshold. This will be a test such as "greater than" or "less than", etc. You can provide a fixed value for the threshold, or you may reference a local register that will hold a threshold that may change from time to time. The trap trigger will be "true" when the MIB variable meets the criteria given.

You have the option of specifying a hysteresis value. If non-zero, this means that the

referenced MIB variable must fall away from the threshold by this amount before a "true" result will return to "false". You also have the option of specifying a minimum On and Off time. The "On" time means the rule must test true for this amount of time before the status will actually be set true and the trap will actually be sent. The "Off" time means the rule must test false for this amount of time before its status will actually be returned to false.

Check "Send if True" to send traps when the trap trigger rule meets all critera for "true". Check "Send if False" if you would also like to send a trap when the trigger rule meets all criteria for "false".

Normally traps will be sent just once. If you would like each trap to be sent more than once, enter a non-zero repeat count and a repeat time in seconds. If repeat count is 1, then the trap would be sent twice (provided repeat time is also non-zero).

11.3 SNMP Trap Summary

The trap summary provides a page where you may see the present status of all of your trap trigger rules. Here we see the normal "false" status of the above trigger.

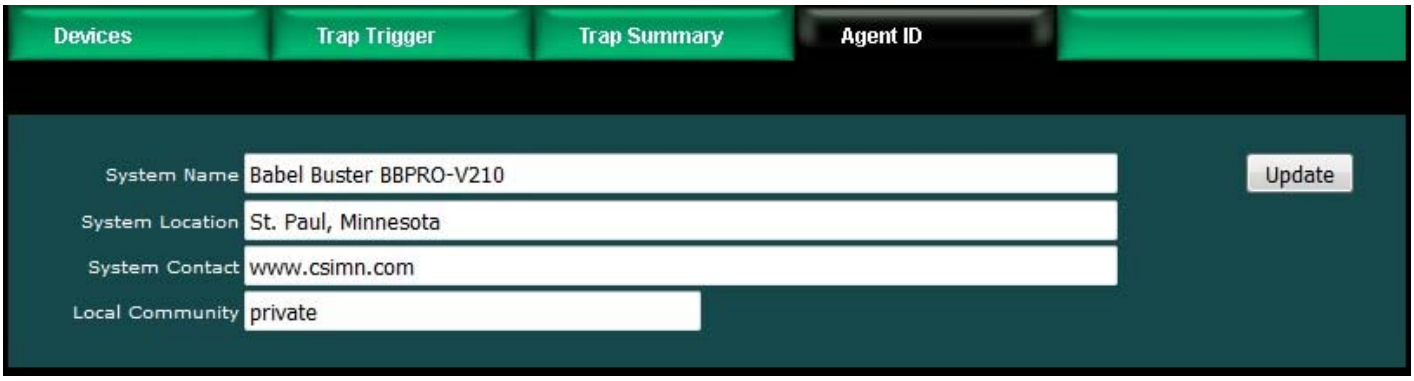
Trap #	Local MIB Variable	Local Register #	Test Result	Local Value	Local Name
1	Integer.1	1	False	5.190000	Data Value 1
2	---	0	False	---	---

The thresohld has been exceeded in the above rule, and therefore a trap has just been sent in the illustration below.

Trap #	Local MIB Variable	Local Register #	Test Result	Local Value	Local Name
1	Integer.1	1	True	33.889999	Data Value 1
2	---	0	False	---	---

11.4 SNMP Identity

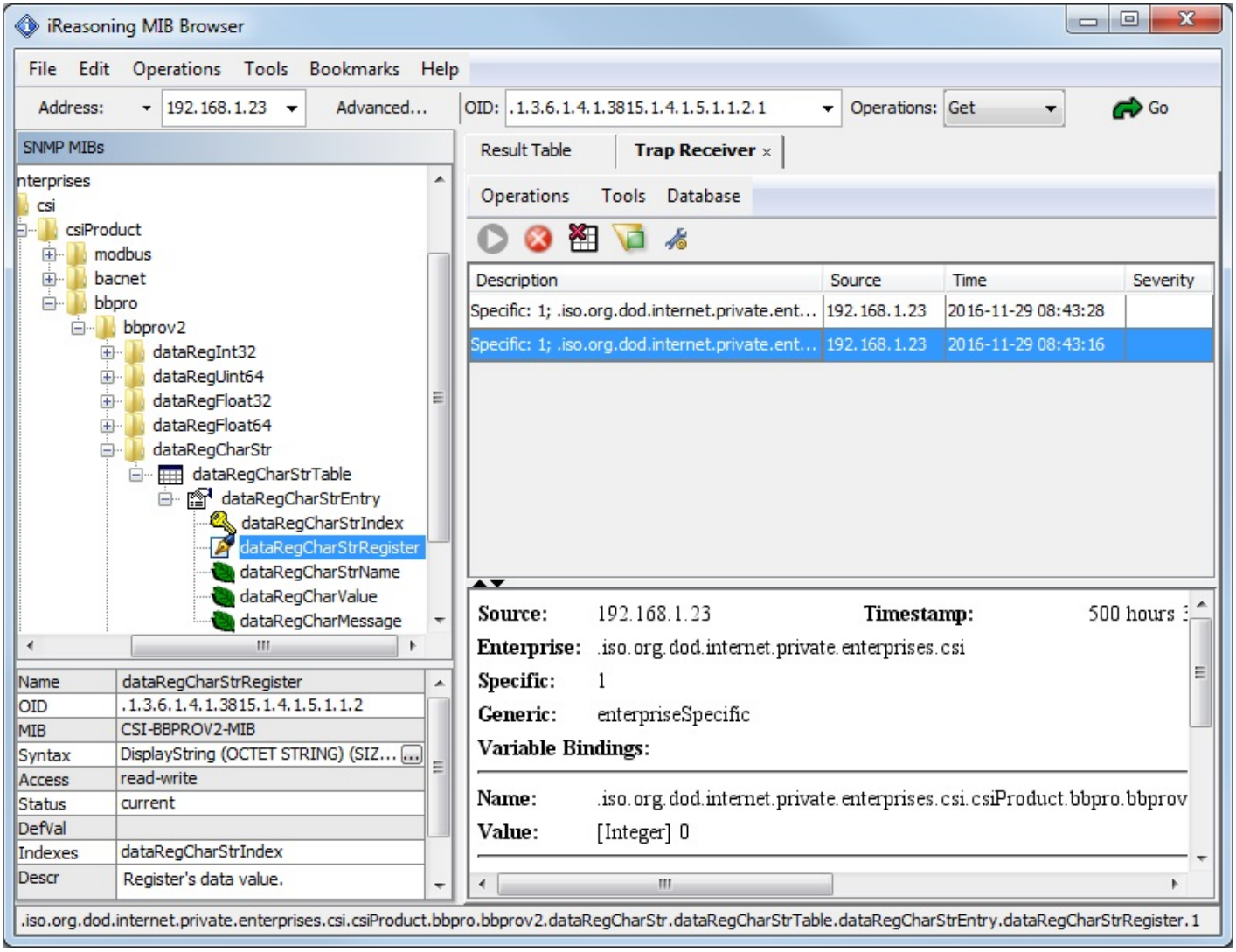
Sending traps requires setting up the local MIB. Part of setting up a local MIB includes setting the identity of the local device. This is done on the Agent ID page. Local community is also accessible on the Network setup page, but the remaining parameters for local identity are only accessible here.



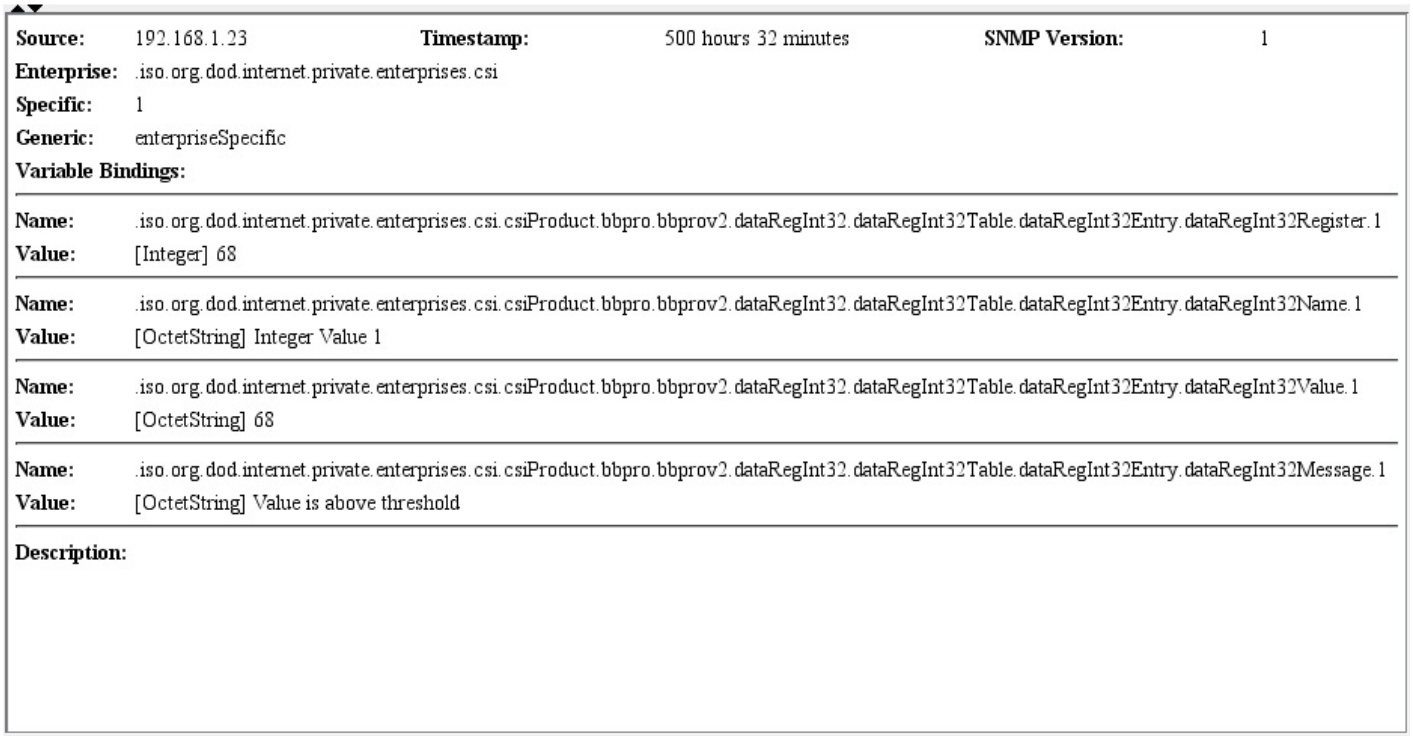
11.5 Testing the SNMP Trap Sender

A variety of tools are available for browsing an SNMP MIB and receiving SNMP Traps. The tool used in the following examples is the iReasoning MIB Browser. Refer to the Tools section under Support at csimn.com for more information about SNMP tools.

The iReasoning MIB Browser allows you to browse the MIB, do table walks, etc., but of primary interest here, also allows you to test traps. When you open the iReasoning browser, under Tools, select Trap Receiver. The screen illustrated below shows the trap viewer after receiving a couple of v1 traps from our Babel Buster Pro.



If you expand or scroll the window in the lower right, you will see the full view of the trap as illustrated below.



Source: 192.168.1.23 **Timestamp:** 500 hours 32 minutes **SNMP Version:** 1
Enterprise: .iso.org.dod.internet.private.enterprises.csi
Specific: 1
Generic: enterpriseSpecific
Variable Bindings:

Name: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bbpro.bbprov2.dataRegInt32.dataRegInt32Table.dataRegInt32Entry.dataRegInt32Register.1
Value: [Integer] 68

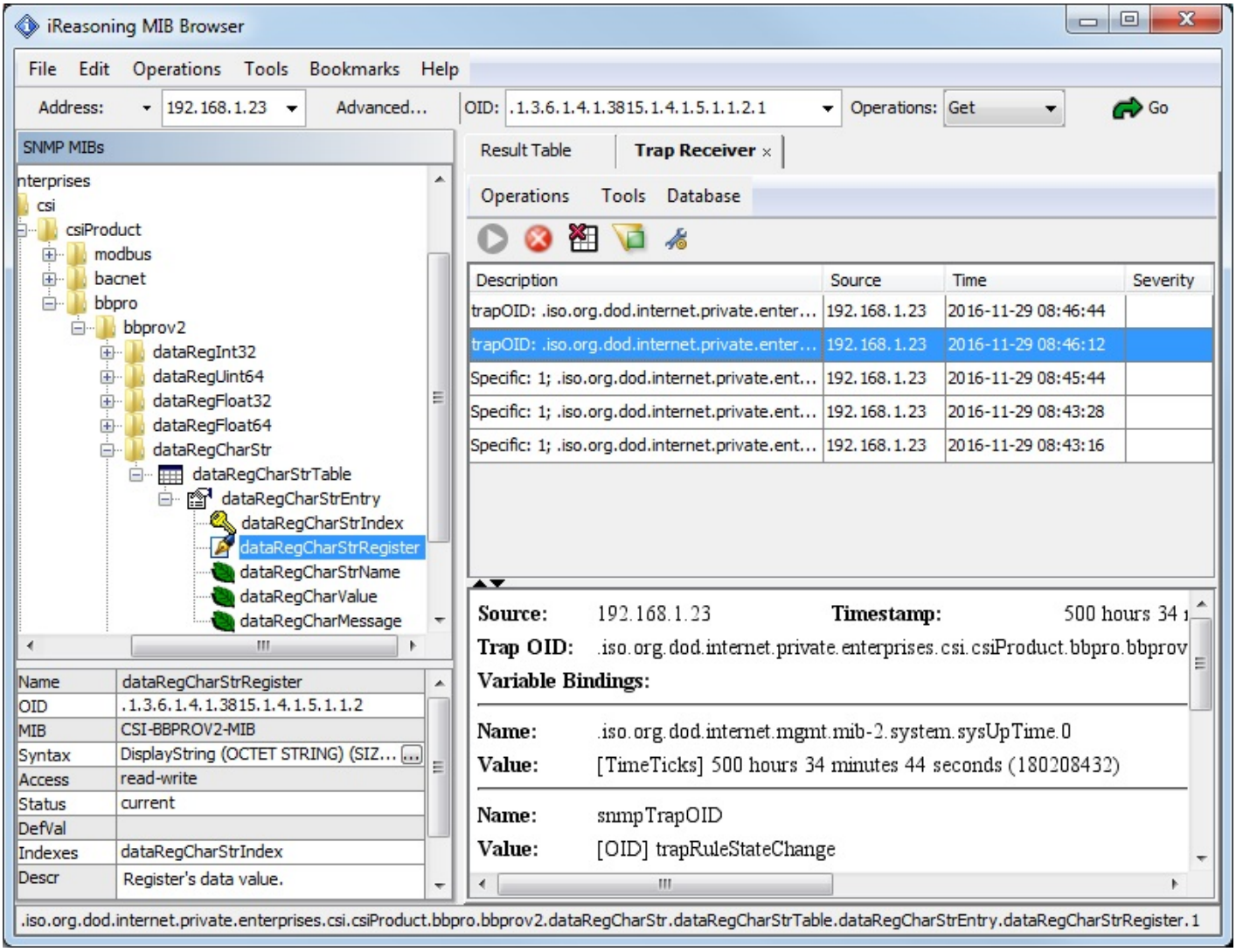
Name: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bbpro.bbprov2.dataRegInt32.dataRegInt32Table.dataRegInt32Entry.dataRegInt32Name.1
Value: [OctetString] Integer Value 1

Name: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bbpro.bbprov2.dataRegInt32.dataRegInt32Table.dataRegInt32Entry.dataRegInt32Value.1
Value: [OctetString] 68

Name: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bbpro.bbprov2.dataRegInt32.dataRegInt32Table.dataRegInt32Entry.dataRegInt32Message.1
Value: [OctetString] Value is above threshold

Description:

The trap viewer, after reconfiguring the Babel Buster Pro to send v2c traps instead, is illustrated below.



The similar but different content of the v2c version of the same trap (with slightly different value) is illustrated below.

Source: 192.168.1.23 **Timestamp:** 500 hours 34 minutes 44 seconds **SNMP Version:** 2

Trap OID: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bbpro.bbprov2.bbprov2Traps.trapRuleStateChange

Variable Bindings:

Name: .iso.org.dod.internet.mgmt.mib-2.system.sysUpTime.0
Value: [TimeTicks] 500 hours 34 minutes 44 seconds (180208432)

Name: snmpTrapOID
Value: [OID] trapRuleStateChange

Name: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bbpro.bbprov2.dataRegInt32.dataRegInt32Table.dataRegInt32Entry.dataRegInt32Register.1
Value: [Integer] 81

Name: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bbpro.bbprov2.dataRegInt32.dataRegInt32Table.dataRegInt32Entry.dataRegInt32Name.1
Value: [OctetString] Integer Value 1

Name: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bbpro.bbprov2.dataRegInt32.dataRegInt32Table.dataRegInt32Entry.dataRegInt32Value.1
Value: [OctetString] 81

Name: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bbpro.bbprov2.dataRegInt32.dataRegInt32Table.dataRegInt32Entry.dataRegInt32Message.1
Value: [OctetString] Value is above threshold

Description:

12. Configuring SNMP Trap Receiver

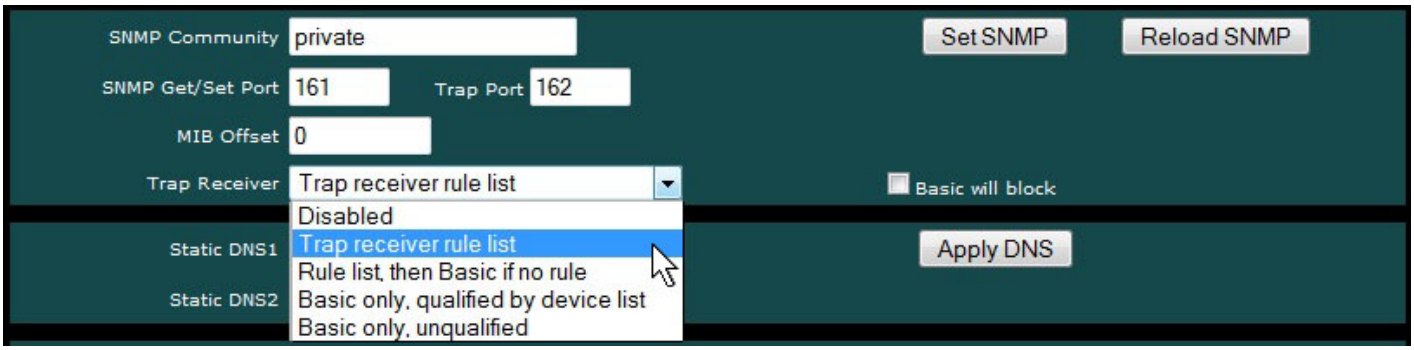
12.1 Trap Receive Devices

This Devices page under Trap Receiver provides IP addresses of devices that will be sending traps to this Babel Buster. Traps will not be recognized by the trap rules if the IP address is not recognized in the list provided here.

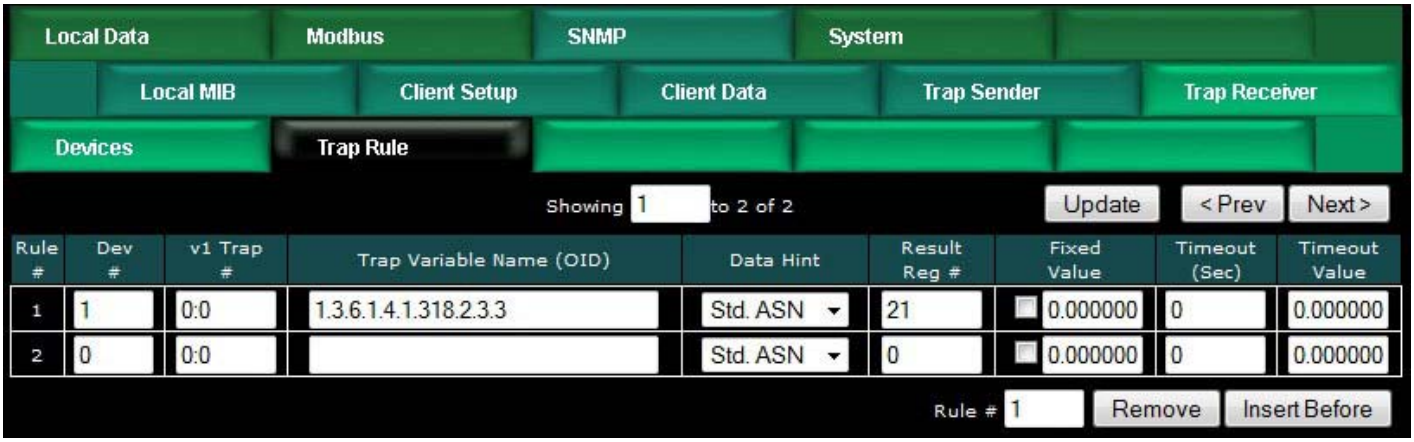
The number of traps received from this device is counted. The counts reflect the number of traps received from this IP address since the last reset (or power up). Click Reset to reset the count. Both total traps and traps recognized by rules in the table are tabulated. Traps processed by Script Basic are not counted in the "recognized" tabulation since this counter has no way of knowing what Script Basic might have done with the trap message.

In addition to entering the IP addresses of devices that will be sending traps to this Babel Buster, you need to enable trap receiving on the Network setup page. If disabled, nothing entered on the Trap Receiver pages will be utilized. If only trap rules will be used, select "Trap receiver rule list". If Script Basic will be used, select the applicable option. Rules entered on the Trap Rule page will be ignored if either of the "Basic only" options are selected.

The "Basic only, unqualified" option is the one instance where you do not need to enter any IP addresses on the Devices page. In this case, there is no filtering by IP address and all traps regardless of source are then sent to Script Basic. It then becomes your program's job to figure out where they came from and what if anything to do about the trap.



12.2 Trap Receive Rules



Rules for recognizing received traps are defined on this page. You have the option, on the System Setup Network page, to declare that trap receiving is disabled, or if enabled, processed by these rules and/or Script Basic programming.

Rule number simply tells you where you're at on the list of Trap receiver rules. Click "next" and "prev" to scroll through the list. To advance directly to a specific rule, enter the desired number in the "Showing" box, then click Update.

Device number specifies which device on the SNMP Client Devices list this trap will be received from. If traps are received from devices not listed on the Devices page, they will be discarded unless Script Basic has been selected to receive unqualified traps (see Systen Setup Network page). Setting device number to zero will result in the rule being removed from the list the next time the configuration file is saved.

If the trap received is a v1 (version 1) trap, you can specify the trap number expected as "a:b" where "a" is the generic trap number (for example, 6 for private enterprise) and "b" is the specific trap number as found in the device's MIB file. Leave this set to 0:0 if no v1 trap number is expected or should be disregarded. (Note: SNMP v2c traps

will not have this number, and if trap type is detected as v2c, then this entry will be disregarded.)

The variable name that is expected to be found in the trap should be given as an OID for v2 traps, and optionally for v1 traps. If this OID is not found in the trap message, no further action is taken for this trap rule. If no OID is given, especially for v1 traps, then the trap number alone is enough to qualify this trap, and you should use the Fixed Value as the value to be placed into the result register.

Special case use of Trap Name OID: If the trap is v1 and trap numbers are given, then this OID, if provided, will be used to check the enterprise OID of the trap message received. If the OID is provided in the rule, then it must match the v1 trap enterprise OID before the rule will be considered successful.

Data hint helps the data parser figure out what the varible is. If ASN encoding is recognized as other than an octet string, the hint will be disregarded. If an octet string is found, then the parser needs to know if it should be treated as RFC 6340 floating point. If no hint is given (standard ASN), then the octet string will be treated as an ASCII character string, in which case ASCII to numeric conversion will be attempted automatically if the result register is numeric (string will simply be copied if result register is a character string type register). Note that standard, well-known ASN types are recognized as well as the NetSnmp ASN type for opaque float.

If a matching variable name is found, and the data parser succeeds in parsing the data value, then it will be placed into the register identified as "Result Reg #". However, if Fixed Value is checked, then whatever value is entered next to the check box will be placed into the result register just because the variable name was found. In the case of v1 traps, if no OID is given, then the trap number alone is enough to casue the fixed value to be placed into the result register.

Timeout is optional, and will be in effect if a non-zero time in seconds is provided. If given, then the timeout value will be placed into the result register after this amount of time has elapsed since receiving the trap. If another of the same trap is received, the timer will be reset. Only after this time has expired without receiving the trap, but after receiving it at least once, the timeout value is placed into the result register.

Click Update when all entries have been made. After all configuration has been entered, be sure to go to the Config File page and click Save so that changes are retained through power cycles.

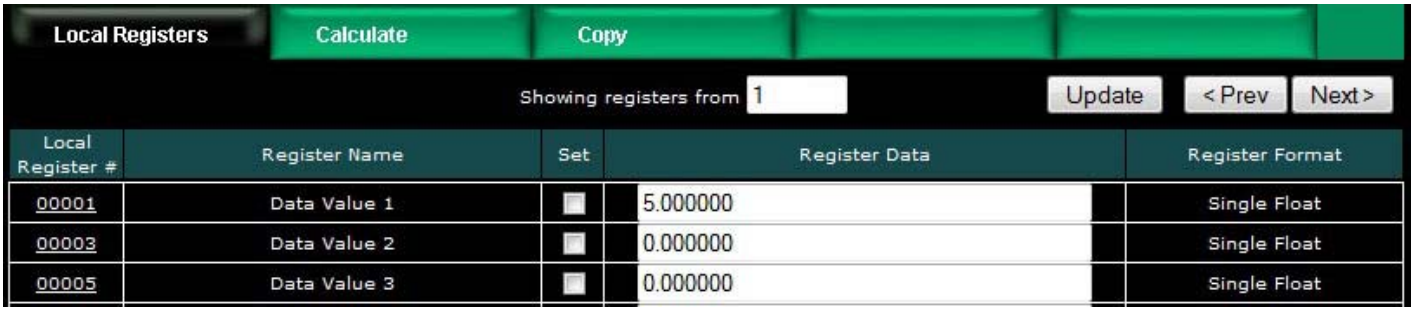
To insert or delete trap rules in the table, enter the trap number and click the appropriate button. To add a new rule to the end of the list when already at the end, simply click the Next button at the top.

12.3 Trap Receive Examples for SNMPv1

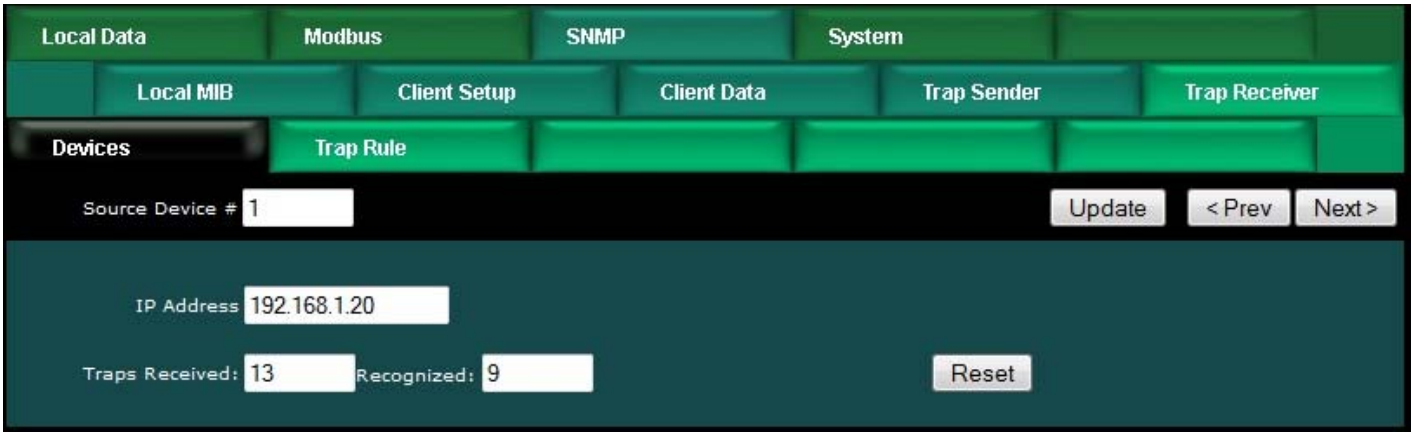
An APC UPS includes both RFC 1628 and their own private enterprise MIB which starts

at 1.3.6.1.4.1.318. The APC MIB sends enterprise specific trap number 5 when the UPS switches to battery power, and trap number 9 when the UPS switches off battery power (utility power restored). The following pair of trap receive rules set local register 1 to a value of 5 when the UPS goes on battery, and 0 when off battery. The choice of the value 5 was purely arbitrary for illustration.

The resulting local register when utility power to the UPS is cut off in this example appears as follows:



You can go back to the Devices page and observe how many traps have been received and how many were recognized by rules. In this instance, both RFC 1628 and the APC private MIB are sending traps, but we are only configured to recognize the APC private traps. Therefore the number received is greater than the number recognized.



The trap receive rules illustrated in this next example will set register 1 to a value of

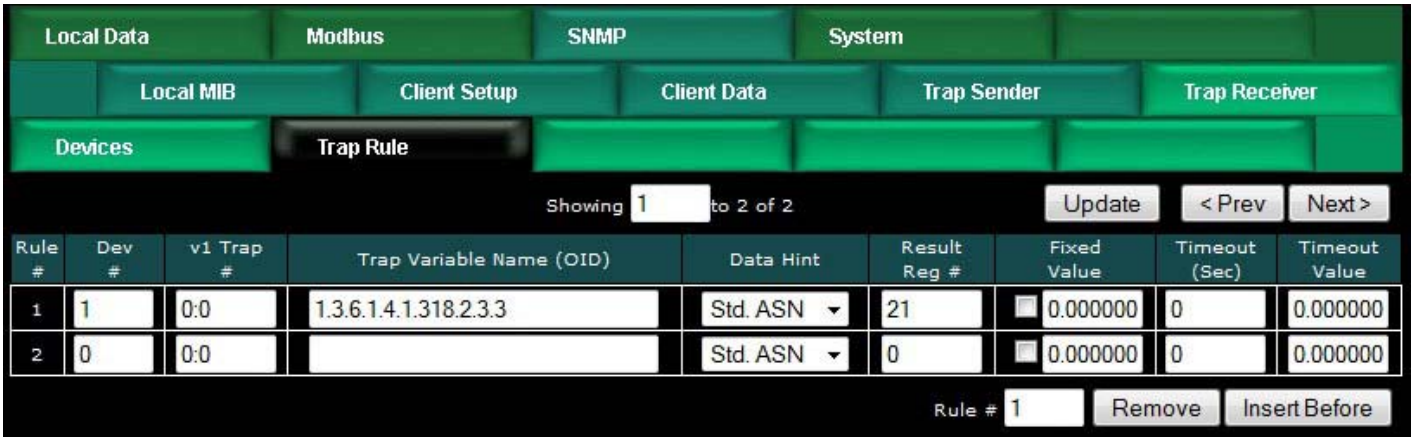
99 (again a purely arbitrary choice) when the "on battery" trap is received, this time from RFC 1628. This trap (per RFC 1628) will repeat every 60 seconds until power is restored. Therefore, we have set a timeout for just longer than 60 seconds so that our local register returns to zero after the alarm traps stop ocurring.

In addition to the trap number itself, the trap includes some varbinds (data pairs consisting of an OID and the data value identified by that OID). Rule #2 in this example will capture the remaining minutes of battery power sent by the UPS with each of those trap messages sent every 60 seconds. Although this is still a v1 trap, it is important to leave the trap number set to 0:0 so that the varbind is processed by the rule rather than the trap number.

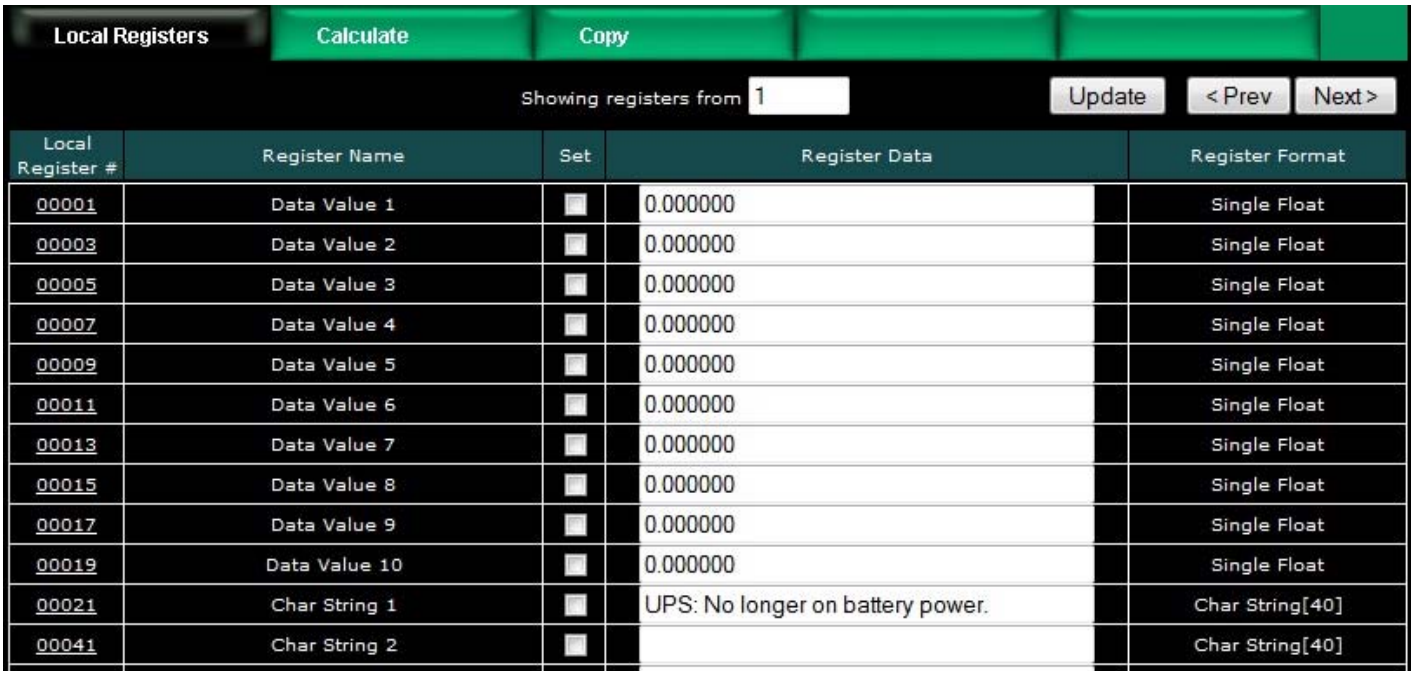
Immediately upon switching to battery power (utility power cut off), registers 1 and 2 are populated as illustrated below by the traps received from our test UPS. Data Value 1 is simply the fixed value flagging the fact that we are on battery power, and Data Value 2 is the number of minutes of remaining battery power provided by the UPS.

Upon restoring utility power, Data Value 1 returns to zero indicating we are no longer on battery power. Data Value 2 will continue to contain the last received "minutes remaining" value.

Another option we have with the v1 traps sent by the APC private MIB is to capture the character string message included in the trap. The trap rule illustrated below will take the string from this varbind and place it into a character register.

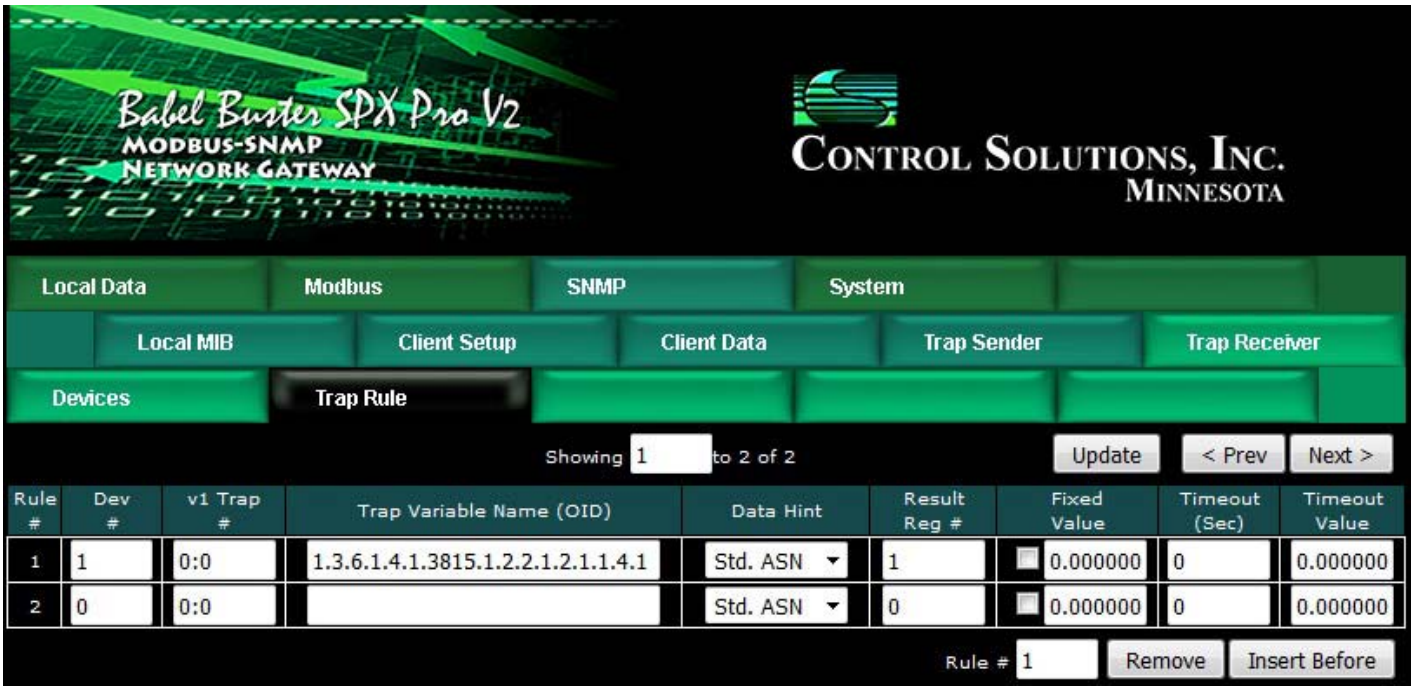


The screen shot illustrated below shows the message received in the trap message that was sent when utility power was restored.

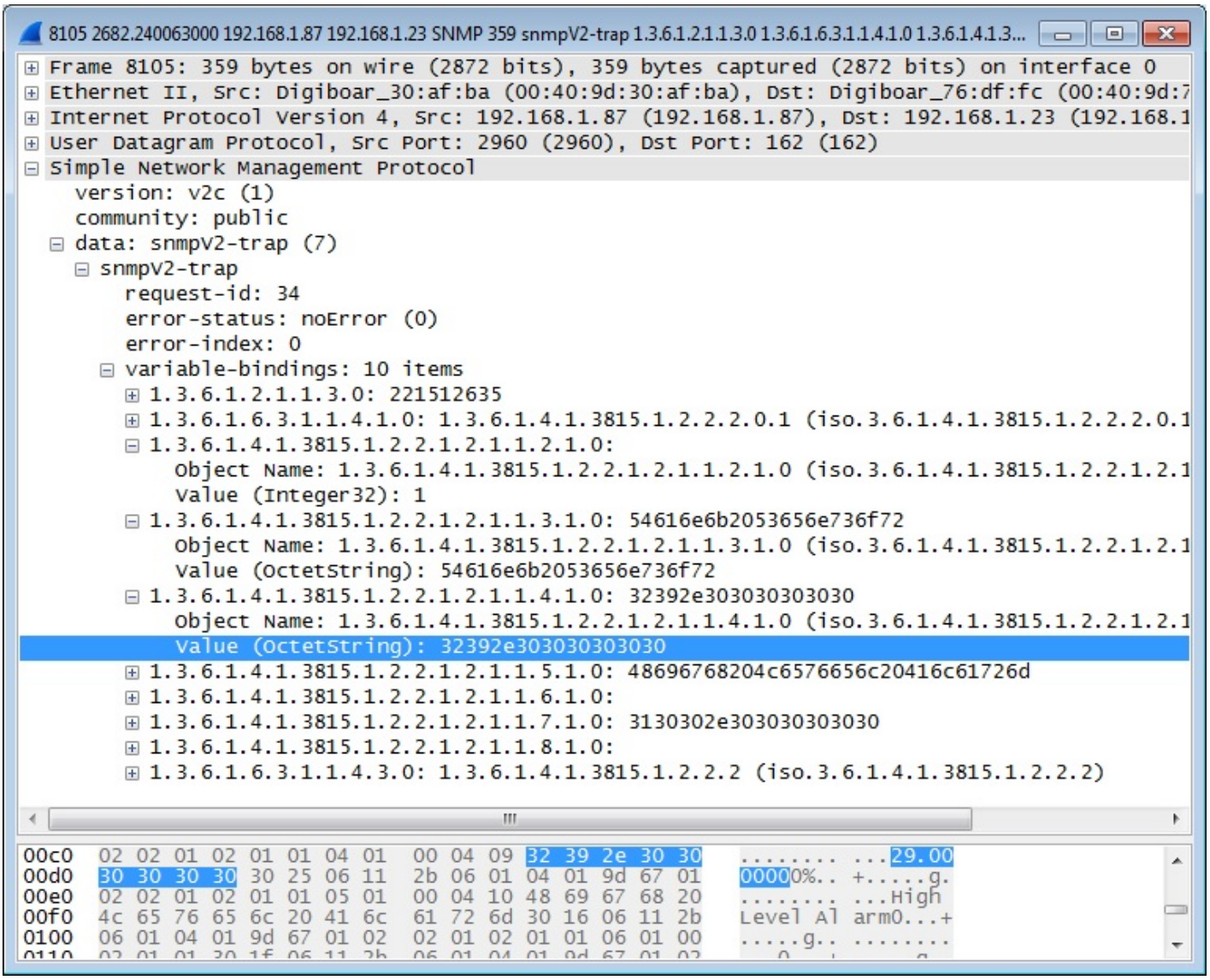


12.4 Trap Receive Example for SNMPv2

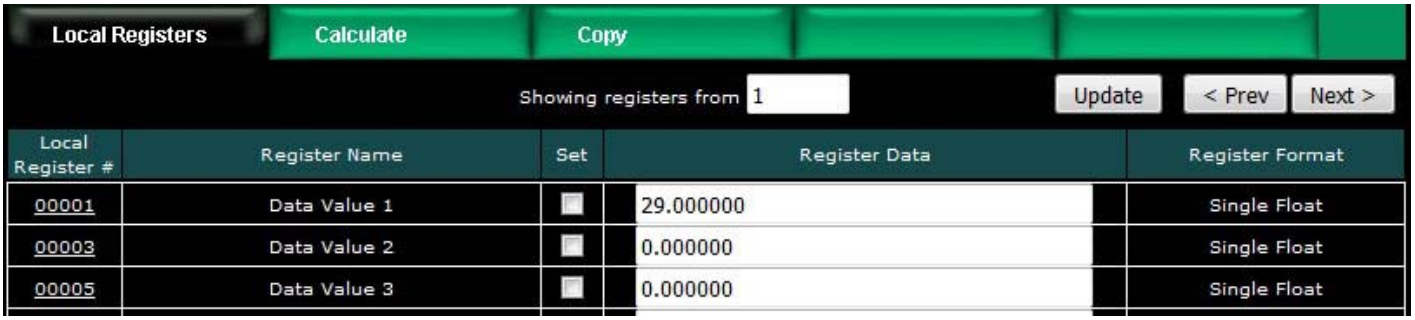
Most of what is discussed for SNMPv1 trap receive rules also applies to SNMPv2. The only real difference is that SNMPv2 does not send trap numbers, therefore you cannot use the v1 trap "a:b" number for v2 traps. Anything that will be recognized as a v2 trap will require a Trap Variable Name (OID). Other than trap number, everything else about configuring the v2 trap receive rule works the same as for v1.



The Wireshark capture of a trap from this v2 device is illustrated below. In this case, the OID in the trap rule identifies a numeric value that is sent as an ASCII string.



Numeric conversion of the Octet String is attempted automatically since the data hint is "Std. ASN" (standard ASN). The result is the correct value placed in a floating point local register.



13. Programming with Script Basic

13.1 Creating a Program

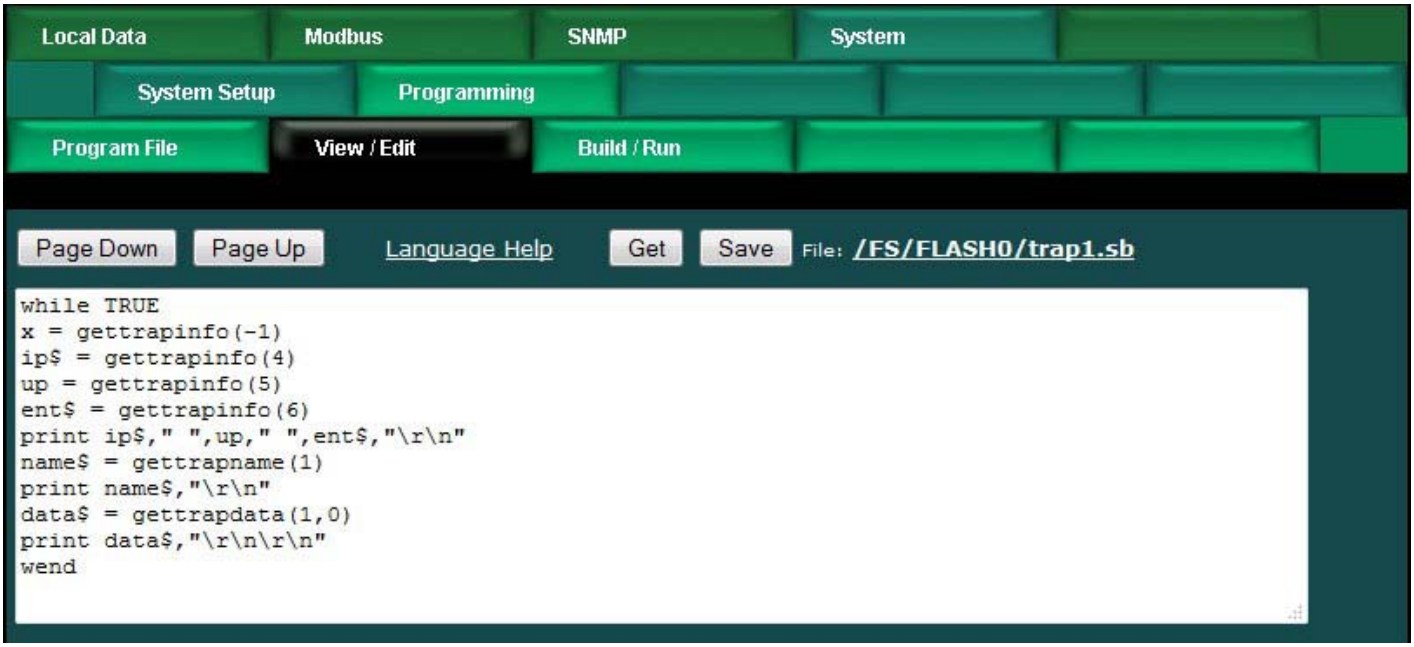
Start by selecting a Directory. For most applications, you would select the Flash directory since anything stored in the Ram directory will be lost when power is lost.

Enter a new file name ending in ".sb" for your program. The program should use only alphanumeric characters and be limited to 20 characters. As soon as you click the New button, the file will be created and automatically selected. If you are returning to edit a previously created program. select that file from the File list, and click Select.

There is a local, very simple text editor available via the web View/Edit page. To edit an existing file, start by clicking Get. After entering a new program, or editing an existing program, click Save.

It is recommended that you use an external text editor for large programs, and simply upload it on the Program File page.

The Language Help link provides a summary of Script Basic. For a complete reference, use the external compiled help file available for Script Basic.



13.2 Testing the Program

The virtual terminal can be used while testing programs. Any "print" statement will send its output to the Output window, and any "line input" statement will take input from the Input window. The print and line input statements will have no effect when running in the background (i.e. running as Auto run from startup).

The Virtual Terminal page does not auto refresh, therefore any output produced by a print statement will not appear until you click the Refresh button. Some initial output may appear immediately since the program began running faster than the initial page refresh that occurs after clicking Start, but you will need to click Refresh to see additional output.

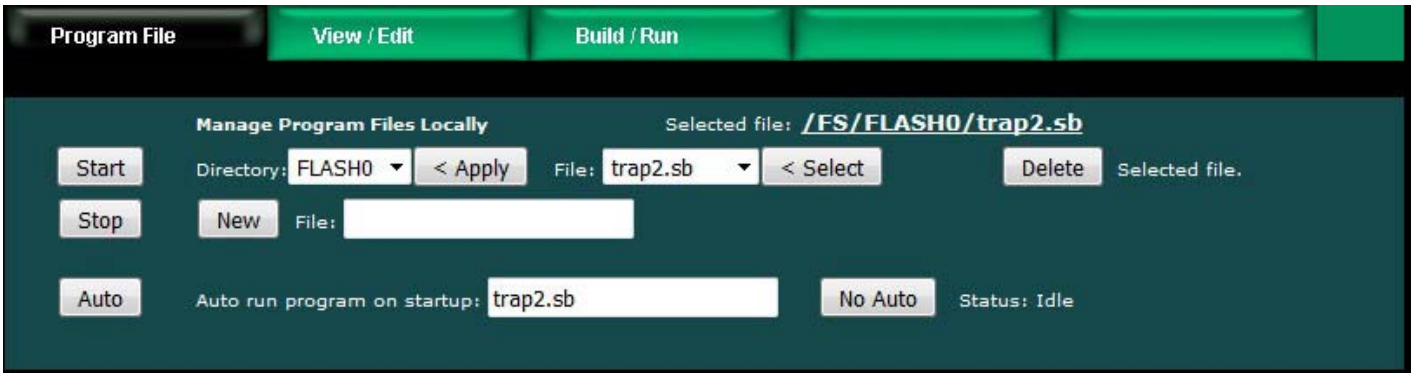
WARNING: If you are testing a program, you should select No Auto (see below), then restart the Babel Buster Pro. You will have two programs running at the same time if the auto run program is running and you are also running a program here. The results can be unpredictable if you are running two copies of the same or similar program at the same time.

An abbreviated screen shot of the Virtual Terminal is illustrated below. In this example, output from the above trap test program is illustrated.

13.3 Setting the Program to Auto-Run on Startup

Your program will not be very useful if it does not start up when the Babel Buster Pro boots up. You cause that to happen by selecting your file from the list, and then clicking the Auto button. Setting the auto run program will take several seconds as Flash memory is being reprogrammed at this point.

From this point on, your program named in the "Auto run program on startup" window will be automatically started upon bootup. Of course, if the program has errors, it might not keep on running. Be sure to test your program ahead of time, and also consider use of the "On Error" feature of Basic. What exactly you might do upon error is entirely up to you, but one potentially useful option is to set an error number of your own making in a specific data object that will be reported to the web portal using "Report on Delta".



To eliminate the auto run program, simply click the No Auto button. Clicking the No Auto button will only make certain no program is started upon bootup from this point forward. You will need to restart the Babel Buster Pro to halt any program that was already running from the last bootup.

13.4 Special Functions for SNMP Access and Diagnostics

The Babel Buster Pro includes several functions that are unique to its implementation of Script Basic. The language in general is summarized on the page found at the Language Help link on the View/Edit page. A much more detailed collection of help may be found in the external compiled help file that may be downloaded at csimn.com. The compiled help includes reference guides for all of the string manipulation functions. The compiled help does not include the special functions documented here.

Trap Information Retrieval:

```
n = gettrapinfo (x)
```

This function is used first of all to see if any trap has been received. Then, if a trap has been received, it is used to retrieve several pieces of information about that trap. Each time gettrapinfo(0) or gettrapinfo(-1) returns a non-zero value indicating a trap was received, the previously retained trap is discarded. Until gettrapinfo(0) or gettrapinfo(-1) is called again, the same received trap will be retained for all subsequent trap related function calls.

For gettrapinfo(x), the various pieces of information are requested using selector 'x' using one of:
-1 = same as 0, but suspend until there is a trap to return
0 = return trap version if new trap (0=no new trap, 1=v1, 2=v2)
- each gettrapinfo(0) releases previous trap, queues up new trap if any for subsequent calls using x=1..6
1 = return trap generic number if v1, or 0
2 = return trap specific number if v1, or 0
3 = return number of varbinds in trap
4 = return IP address of sender as character string
5 = return uptime from trap as integer

6 = return enterprise OID from trap as character string

Trap Name Retrieval:

n = gettrapname (x)

Returns var name (OID) as string for varbind 'x', where x is 1..max as returned by gettrapinfo(3).

Trap Data Retrieval:

n = gettrapdata (x,y)

Returns var value for varbind 'x', where x is 1..max as returned by gettrapinfo(3). Format of data is specified by format 'y' where format is:
0 = want character string
1 = want integer value (convert as necessary)
2 = want floating point (convert as necessary) (also indicates RFC 6340 expected if octet string)

The following additional functions are unique to the Pro gateway, but are not strictly SNMP related.

Error Information Retrieval:

n = geterr (x,y)

Returns n=0 for no error, otherwise returns error code or status for item specified by 'x', item number 'y' (1..N). Error codes returned are those displayed on the respective web pages as error codes or device status.

Item 'x' may be:
1 = SNMP client device
2 = SNMP trap sender device
3 = SNMP trap receiver device
4 = Modbus TCP device
5 = Modbus RTU device
6 = SNMP client read map
7 = SNMP client write map
8 = SNMP table walk rule
9 = SNMP trap send rule
10 = Modbus TCP read rule
11 = Modbus TCP write rule
12 = Modbus RTU read rule
13 = Modbus RTU write rule

Most errors are displayed on the web pages as a numeric code with those codes explained in the Quick Help section of the respective page. Modbus errors, however,

are displayed as a description. The error numbers returned here are decoded as follows:

If non-zero, the code will be "abb" where "a" is the error type, and "bb" is the Modbus protocol exception error number if "a" indicates exception. Mathematically, you can divide the error by 100 to get the error type.

Error type "a" is as follows:
1 = Modbus TCP transaction ID error
2 = Slave returned an exception code
3 = Function code returned by slave did not match request
4 = Received too few bytes in reply
5 = No response, request timed out
6 = CRC error (applies to RTU only)

IP Address Retrieval:

```
ip$ = getipaddr (x,y)
```

Returns IP address for item type 'x', item number 'y' (1..N), formatted as character string.

Item 'x' may be:
0 = Our own IP address ('y' disregarded)
1 = SNMP client device
2 = SNMP trap sender device
3 = SNMP trap receiver device
4 = Modbus TCP device (BACnet IP device)

Comm Port Timeout:

```
timeout (n)
```

Sets timeout in seconds that the "line input #n" request for input from the communiation port will wait before returning an empty string if nothing was received on the communication port. Otherwise, the line input will wait indefinitely for a line that ends in a line end character.

The line end character is otherwise known as line feed or "\n". The carriage return will be ignored, unless it is specified to be the line end instead.

To open the communication port (provided Modbus RTU is disabled) as file #2 for example, you would use:

```
open "COM:9600" for comm as 2
```

To open the comm port using the carriage return as line end instead of the (Linux) line end, you would use:

```
open "COM:9600,CR" for comm as 2
```

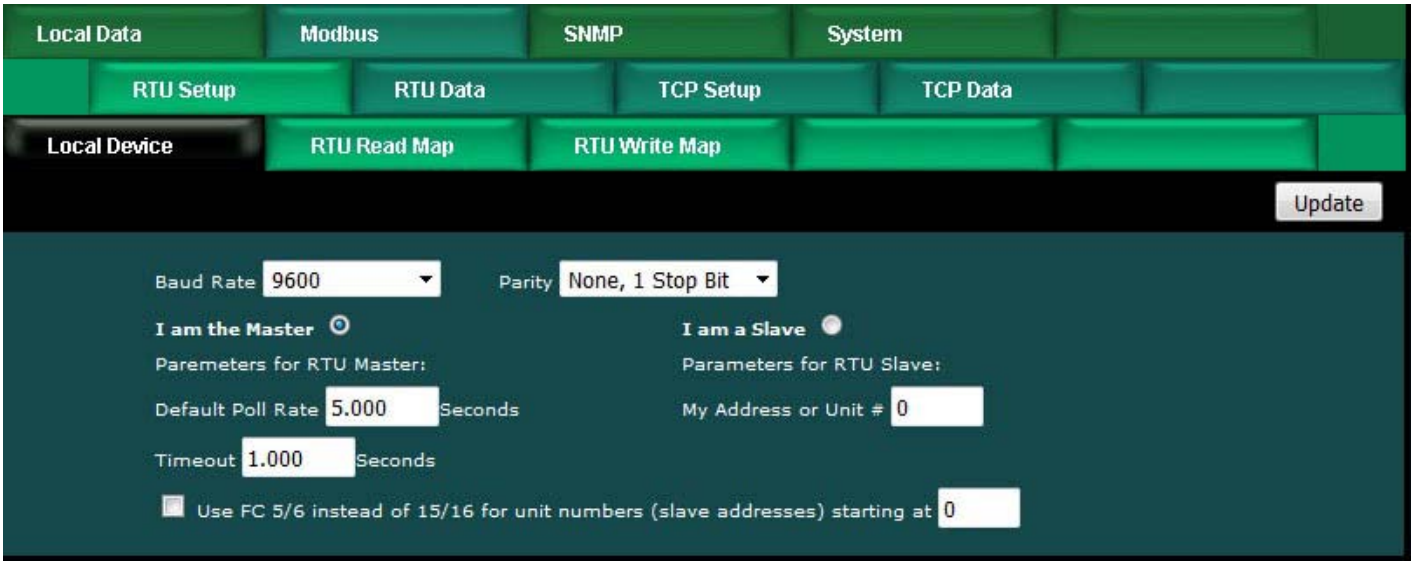
Many devices return both carriage return and line feed at the ends of lines. The sole line end character recognized as the end of line for the comm port will end the line while the other will be discarded and not returned in the string that gets placed in a variable for Basic.

Any of the baud rates valid for Modbus RTU may be specified in place of the 9600 baud used in the above examples.

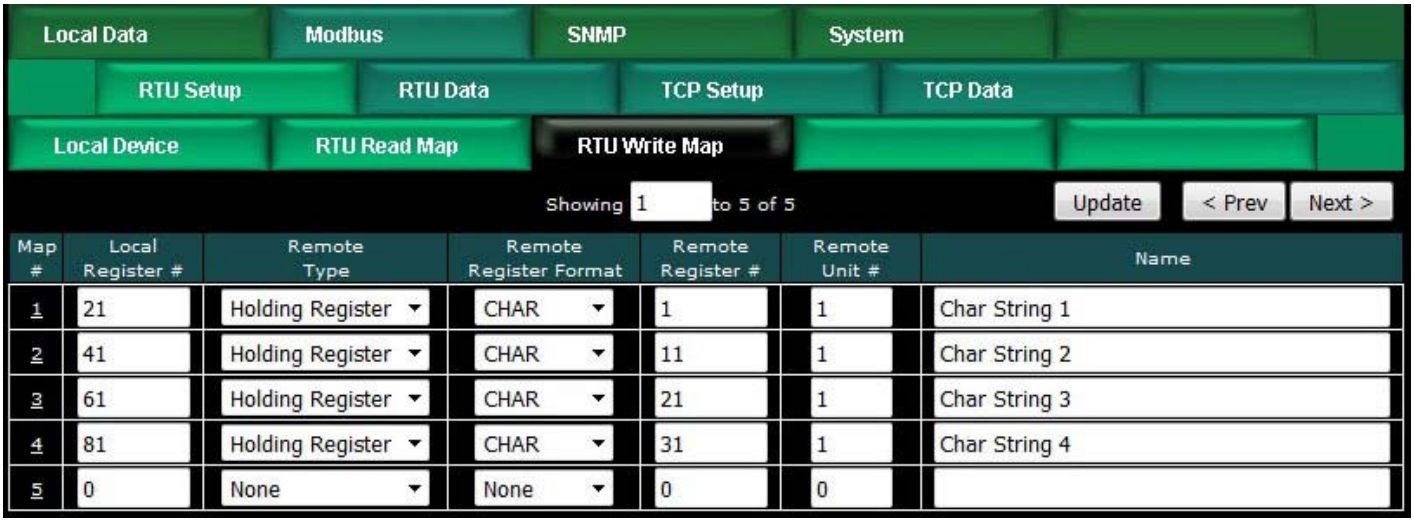
13.5 Example: Writing Alphanumeric Display

The Babel Buster Pro provides support for character strings as a series of Modbus registers that are treated as a single piece of data just like 32-bit integers are a pair of registers treated as one. The most powerful aspect of this feature is the ease of creating and manipulating character strings in Script Basic. Consider the following simple program.

We are going to send this 4-line message to an alphanumeric display that has a Modbus RTU interface. To start out, we need to configure the Modus RTU port for the baud rate that matches the display. The Babel Buster Pro will be the master since the display is an RTU slave.



Next, we create a set of 4 Modbus RTU Write Maps, one per line since the 4 lines are treated as 4 Modbus registers here.

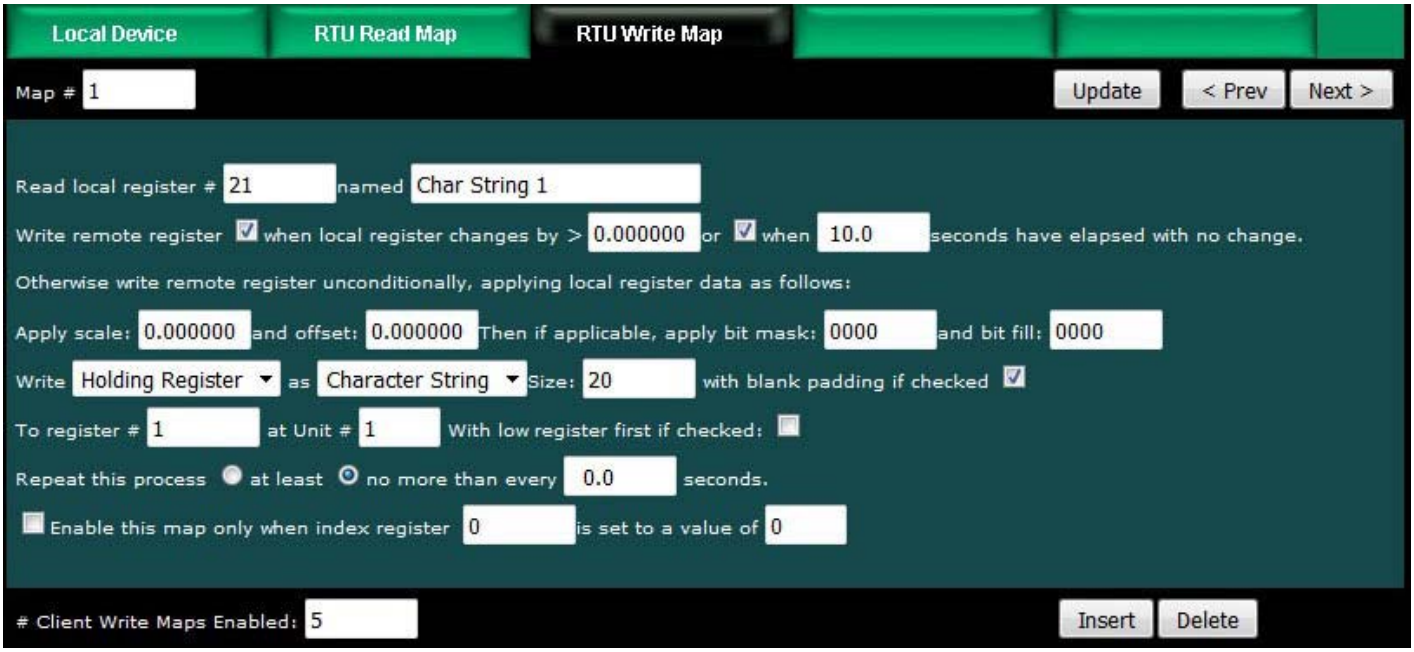


Map #	Local Register #	Remote Type	Remote Register Format	Remote Register #	Remote Unit #	Name
1	21	Holding Register	CHAR	1	1	Char String 1
2	41	Holding Register	CHAR	11	1	Char String 2
3	61	Holding Register	CHAR	21	1	Char String 3
4	81	Holding Register	CHAR	31	1	Char String 4
5	0	None	None	0	0	

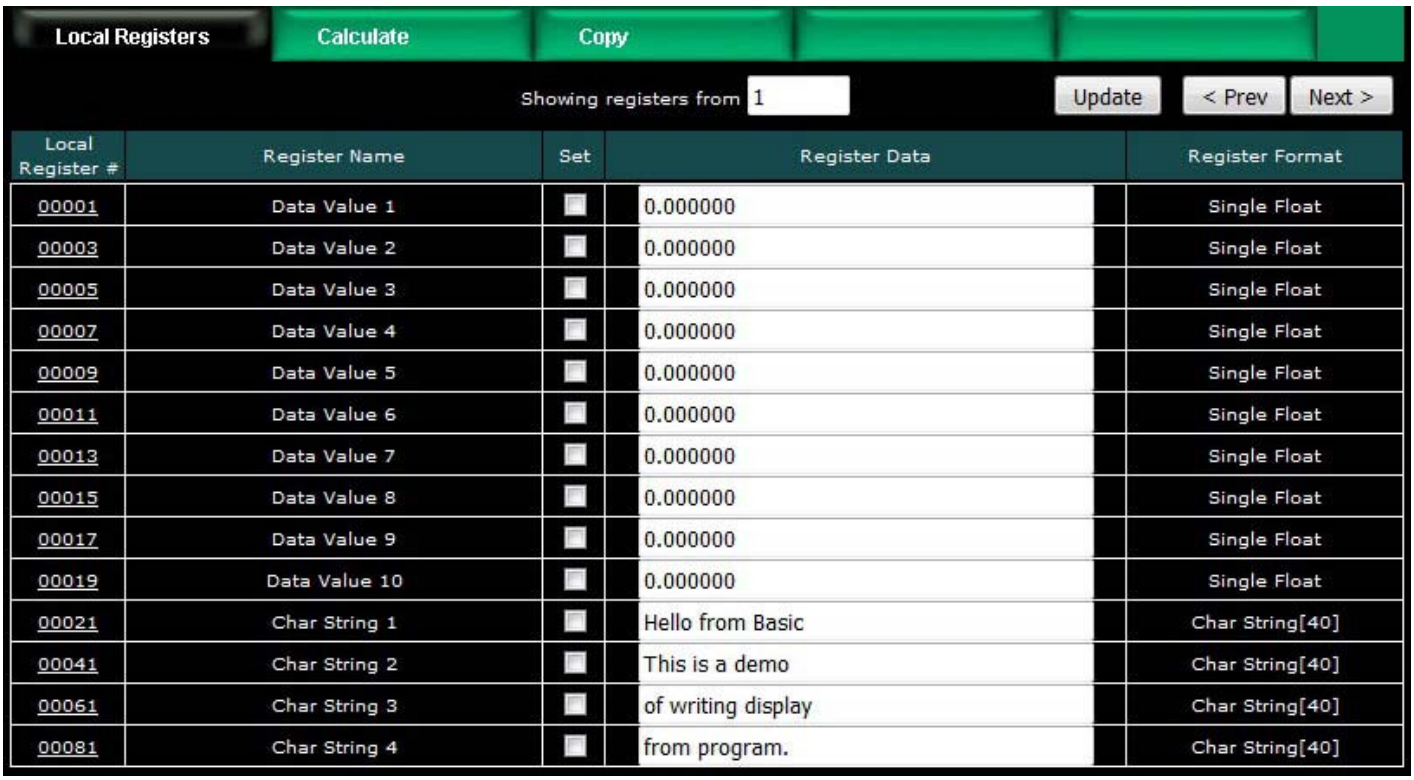
When creating character string maps, it is necessary to use the expanded form to enter the map. Click on the Map number in the first column to access the expanded form for each register. Once you have the expanded form showing for the first register, you can simply click Next to move to the next one.

The one bit of configuration important to character strings that must be entered here is the number of characters. Note that this is number of characters, not number of Modbus registers. The Babel Buster Pro automatically stores two ASCII characters per 16-bit holding register. Therefore, our 20-character strings will occupy 10 registers in Modbus address space.

Note that "when local register changes by" is functional for character registers. The "greater than" value should be left set at zero so that any change results in updating the display. Any attempt at a numeric comparison with a string for purposes of update will not succeed.



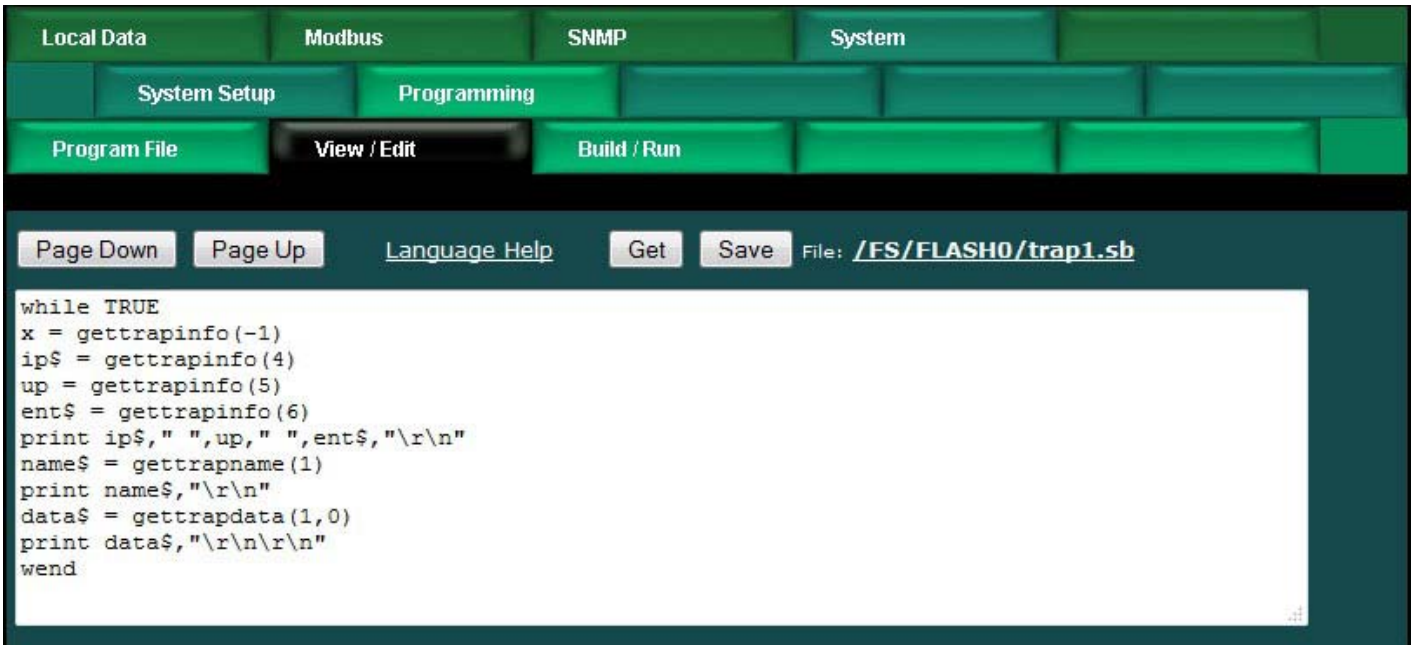
If we click Start on the Programming Build/Run page, the character registers will be set as illustrated below.



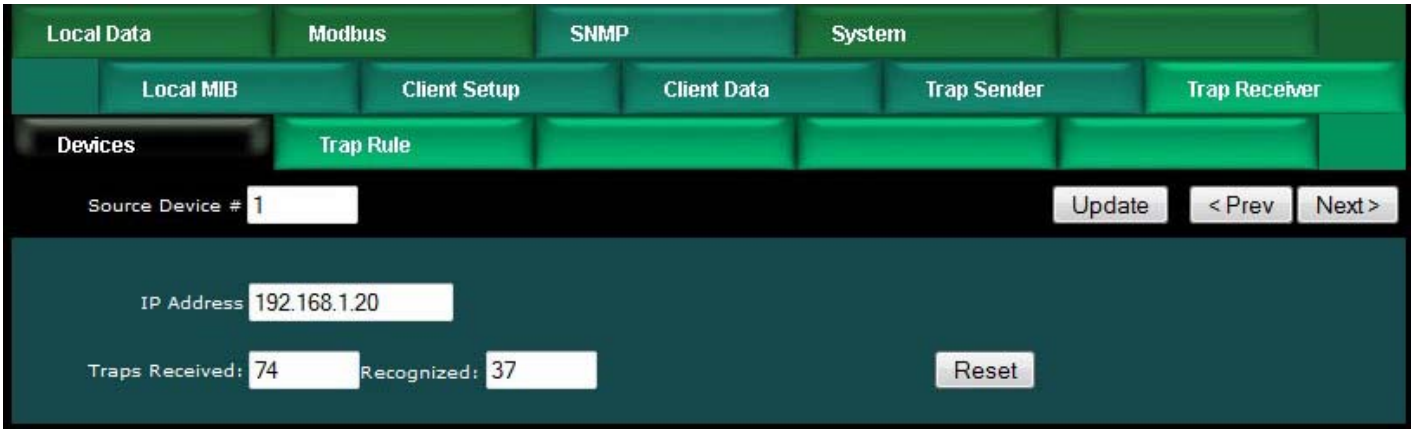
Assuming the Modbus mapping has been set up as illustrated above, the alphanumeric display will now display the message generated by Script Basic. This particular inexpensive display is the model SC2004MBS-B with RS485 interface available at siliconcraft.net. If you set the Modbus RTU port parameters to 9600 baud, and set write maps to send data to slave address (unit) 1, no configuration of the display is needed out of the box.

13.6 Example: Trap Receiver Test

The following example program utilizes some of the special SNMP functions found in this version of Script Basic. Refer to the special functions descriptions above. This test program will receive a trap, and display results in the virtual terminal screen of the Babel Buster Pro. This is not useful for a final in-service program since output from the print functions are only viewable via the web pages. But this is useful to see if you are receiving traps in Basic.

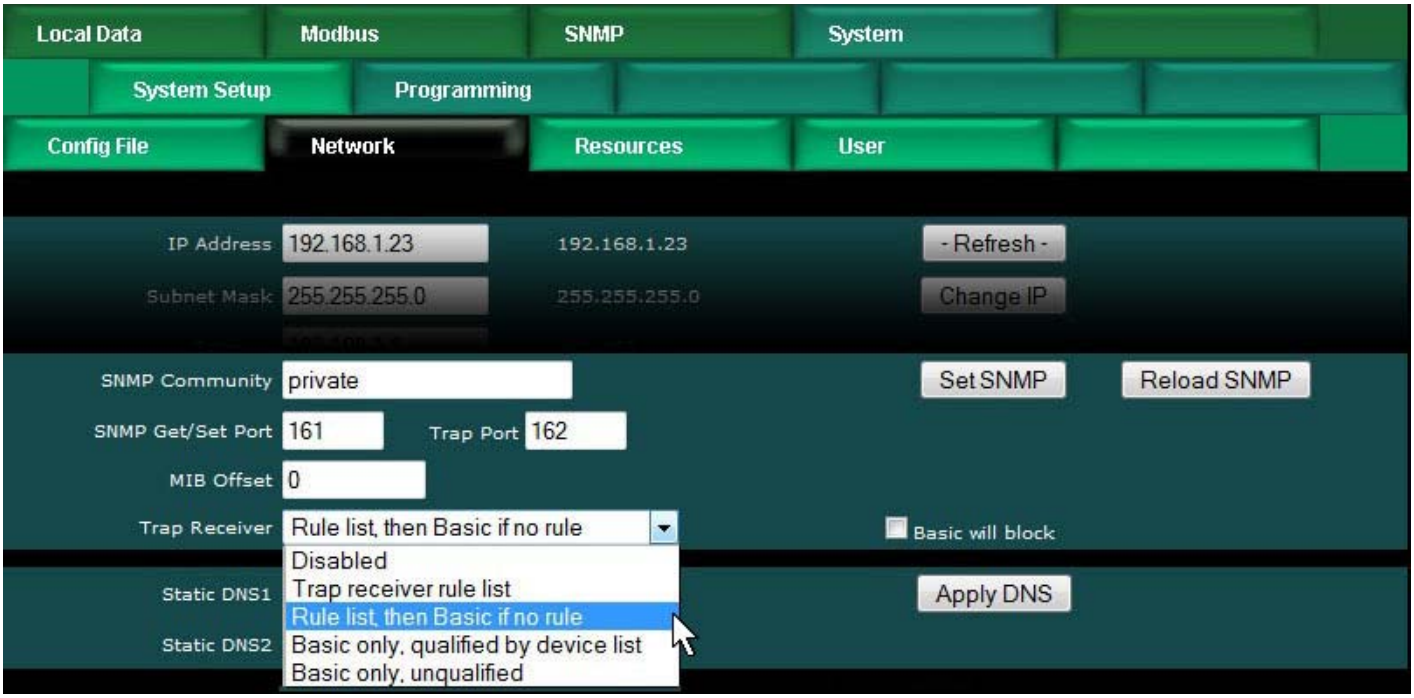


To set up for receiving traps, you need to enter the IP address of the device that will be sending traps on the Devices page of the Trap Receiver, assuming the trap receiver is enabled as illustrated shortly.



We are going to allow ourselves the option of adding trap rules later. For now, we want Basic to receive all traps from the above IP address, and therefore our Trap Rule list should be empty as illustrated here.

Enable trap receiving by selecting any of the Basic options on the Network setup page. If you select either of the "Basic only" options, then any rules entered later in the Trap Rule list will be ignored. Selecting "Rule list, then Basic if no rule" means that Basic will receive the trap if there is no rule matching the received trap.



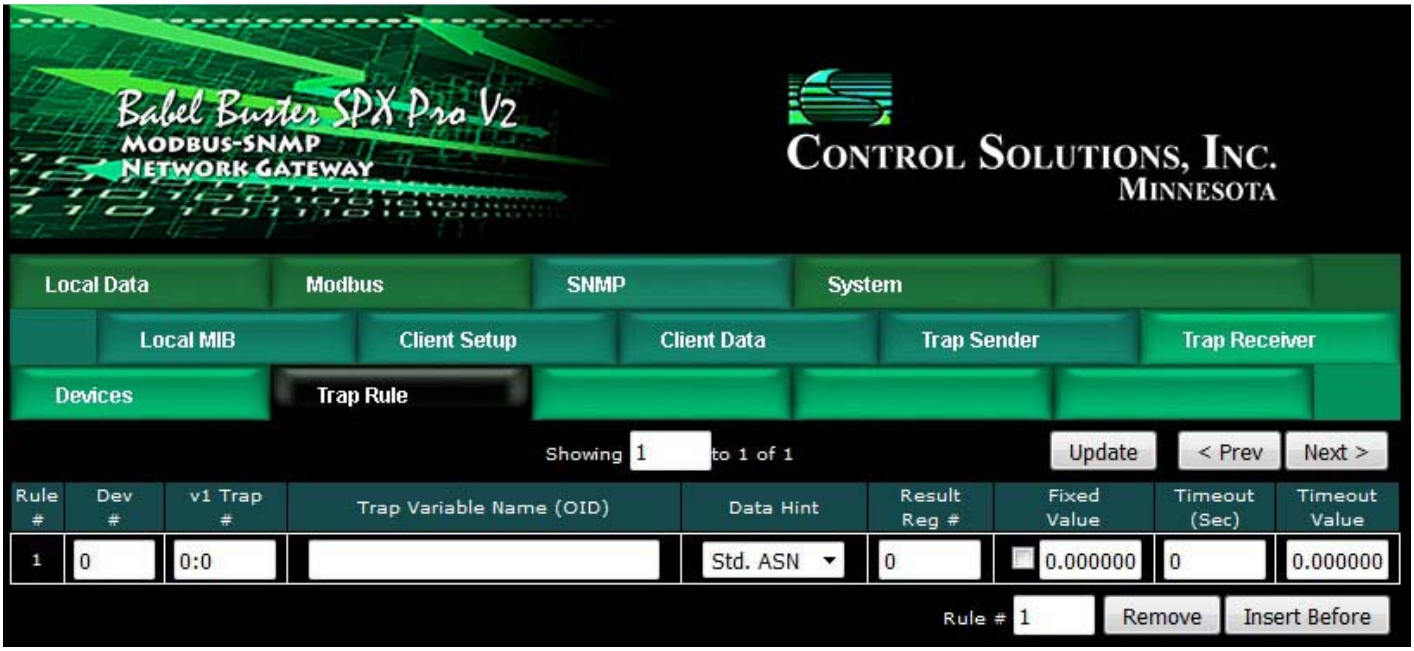
Click the Start button on the Build/Run page. Now cause your trap sending device to send some traps. In our example, we are receiving traps from an APC UPS. This UPS has both RFC 1628 enabled, and APC's own MIB. The traps from RFC 1628 are those with OIDs that start with 1.3.6.1.2.1.33 and the APC MIB OIDs start with 1.3.6.1.4.1.318.

The traps received above happened when power to the UPS was removed. The traps below were received when power was restored.

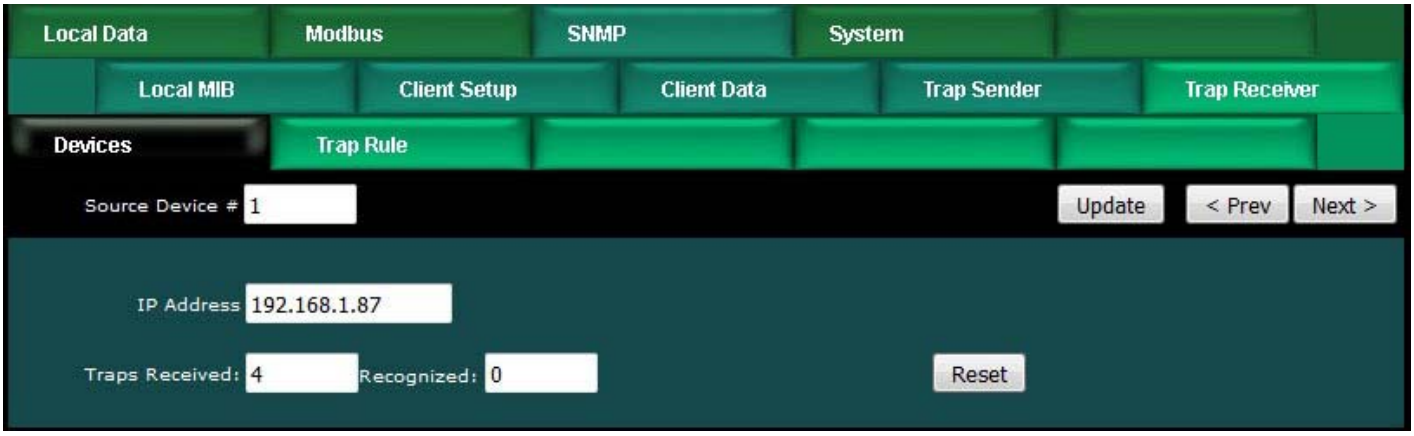
13.7 Example: Trap Info to Alphanumeric Display

The previous example illustrated receiving traps from an SNMPv1 device and displaying results in the virtual terminal for test purposes. In this example, we will be receiving traps from an SNMPv2 device and displaying some results on an alphanumeric display, which is potentially useful for more than simply testing.

As with the previous example of receiving traps in Script Basic, we will leave our Trap Rule list empty.

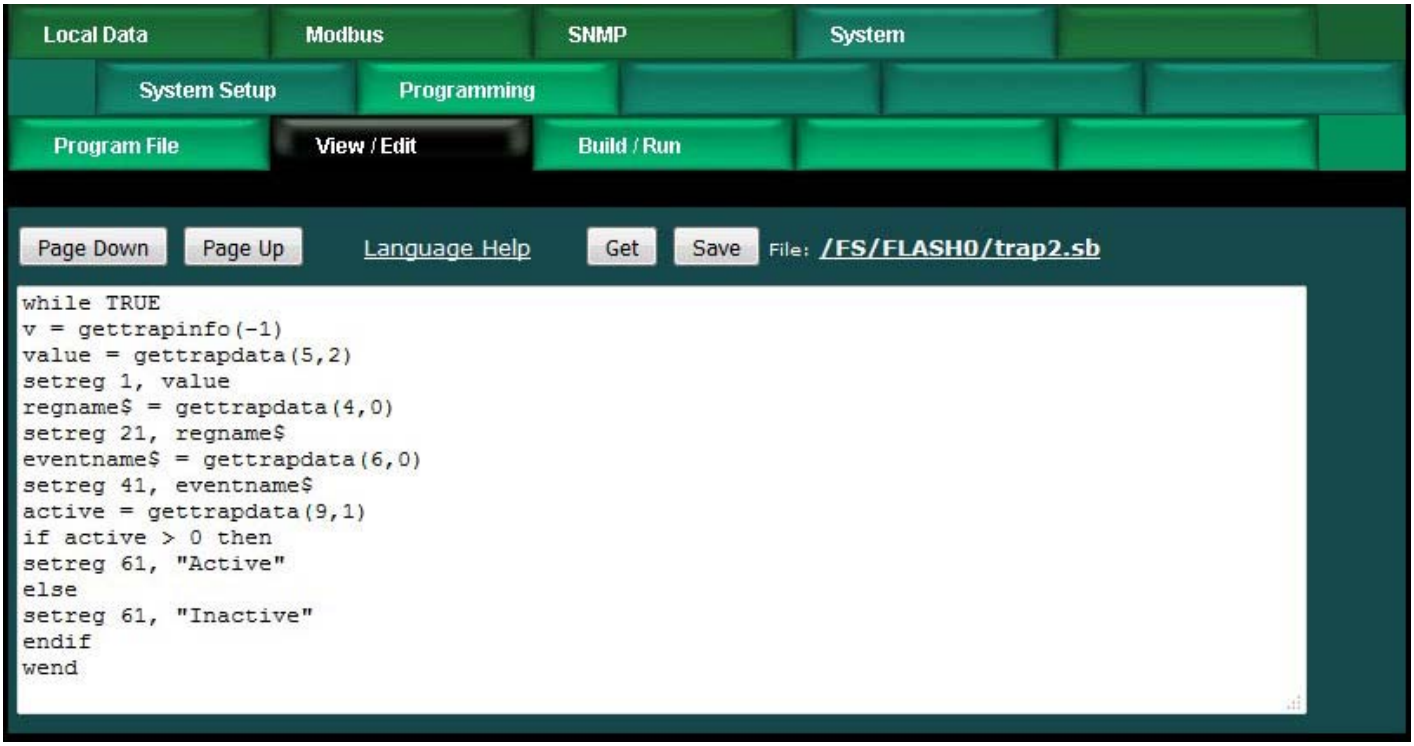


We will enter the source of traps in the Devices list of the Trap Receiver.



The Trap Receiver is again enabled to send traps to Basic if there is no matching rule.

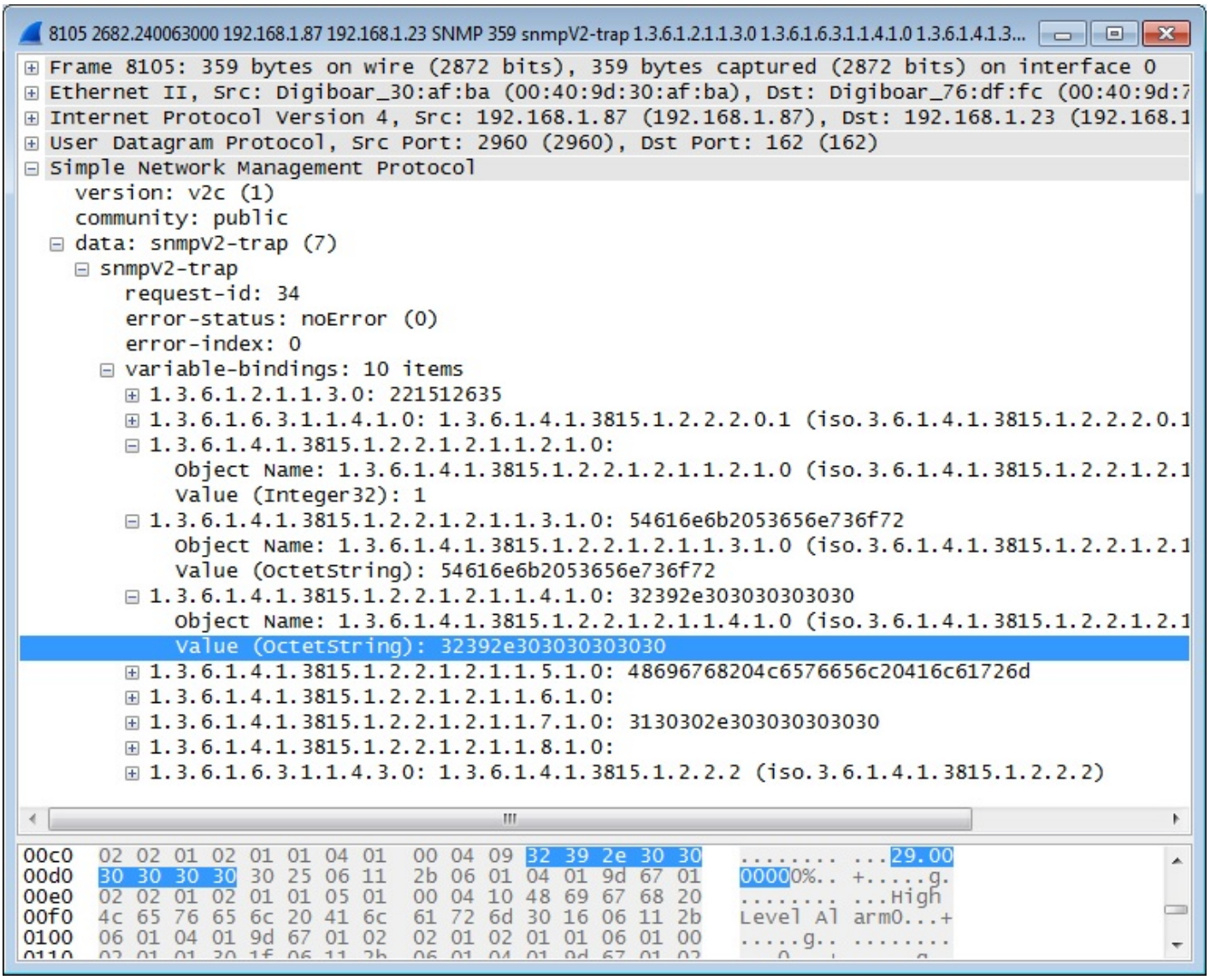
Our program in this case does not print anything to the virtual terminal. Instead, we send the character strings to character registers in the Babel Buster Pro. We are sending some strings exactly as received to the character registers. The one OID that tells us whether an alarm is active or not is being checked, and the string sent to the character register will be "Active" or "Inactive" rather than just a numeric 1 or 0.



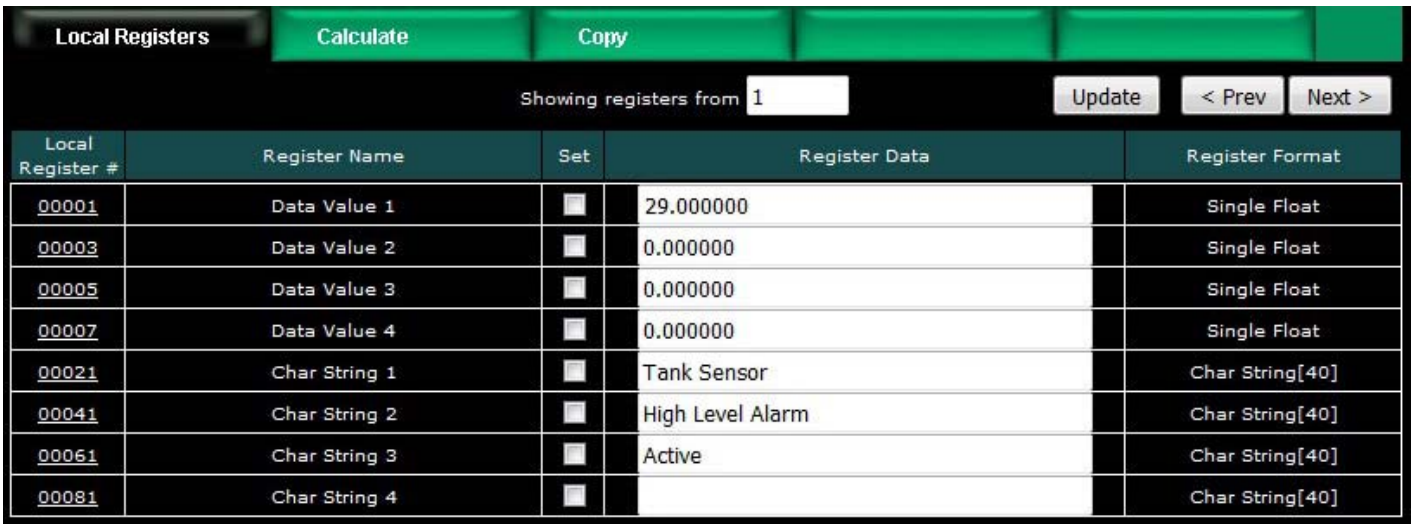
Whether writing a program or creating trap rules to receive a trap, you will need to know what you expect to find in the trap. This will be available in the MIB associated with the SNMP device you are receiving the traps from. You need to obtain this from the manufacturer of the SNMP device. In our example, the SNMP device is a Control Solutions BB2-6010, and the trap portion of its MIB is illustrated here.

```
eventStateChange      NOTIFICATION-TYPE
      OBJECTS {
            eventRegNum,
            eventRegName,
            eventRegData,
            eventName,
            eventTestType,
            eventTestVal,
            eventState
      }
      STATUS            current
      DESCRIPTION
            "This trap is sent when an event changes state
            if notifications are enabled for the object that changed."
      ::= { modbusTraps 1 }
```

Interpreting the MIB can be a project in itself. The MIB definitions are effectively a tree structure, but the tree is documented numerically in the MIB which means you have to follow the entire MIB through to see what the entire OID of any given entry is. It is sometimes helpful to use Wireshark to capture some traps and confirm the OIDs in the received traps. The screen shot below illustrates a trap received from the BB2-6010.



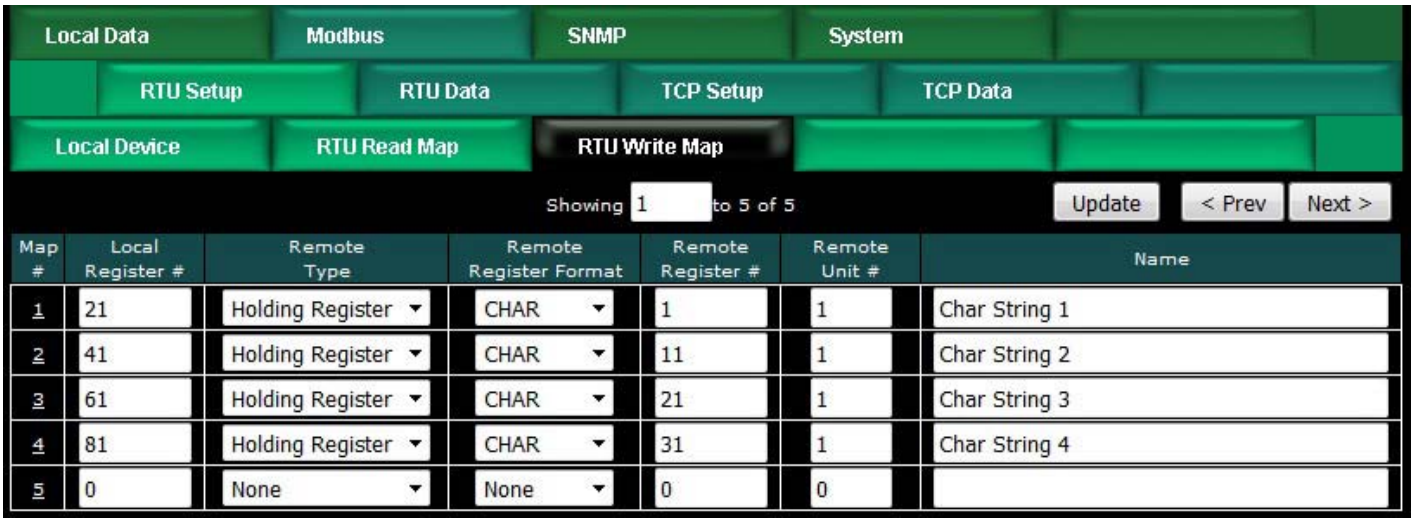
The above trap running the program illustrated in this example results in the character registers being set as illustrated here.



Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	☐	29.000000	Single Float
00003	Data Value 2	☐	0.000000	Single Float
00005	Data Value 3	☐	0.000000	Single Float
00007	Data Value 4	☐	0.000000	Single Float
00021	Char String 1	☐	Tank Sensor	Char String[40]
00041	Char String 2	☐	High Level Alarm	Char String[40]
00061	Char String 3	☐	Active	Char String[40]
00081	Char String 4	☐		Char String[40]

Refer to the prior example on sending character strings from Basic to the alphanumeric display. We have again set up a set of Modbus RTU Write Maps to send

character strings to the display.

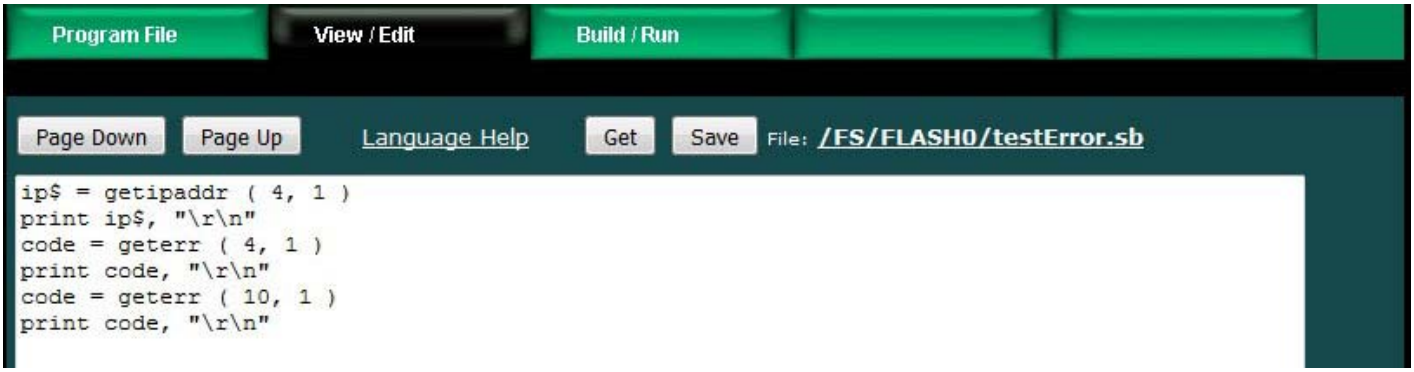


Map #	Local Register #	Remote Type	Remote Register Format	Remote Register #	Remote Unit #	Name
1	21	Holding Register	CHAR	1	1	Char String 1
2	41	Holding Register	CHAR	11	1	Char String 2
3	61	Holding Register	CHAR	21	1	Char String 3
4	81	Holding Register	CHAR	31	1	Char String 4
5	0	None	None	0	0	

The trap now results in an alarm status being indicated on the alphanumeric display.

13.8 Example: Device and Rule Diagnostics

The special functions of greatest interest are probably the SNMP functions, but here is a short example illustrating the diagnostic functions also available. In this example, we are checking on our Babel Buster Pro's attempts to query a Modbus TCP device. The device is #1 in the Modbus TCP device list, and the rule we are checking is Modbus TCP Read Map #1.



Normal results will be to display the IP address and two error codes that will be zero if there are no problems.

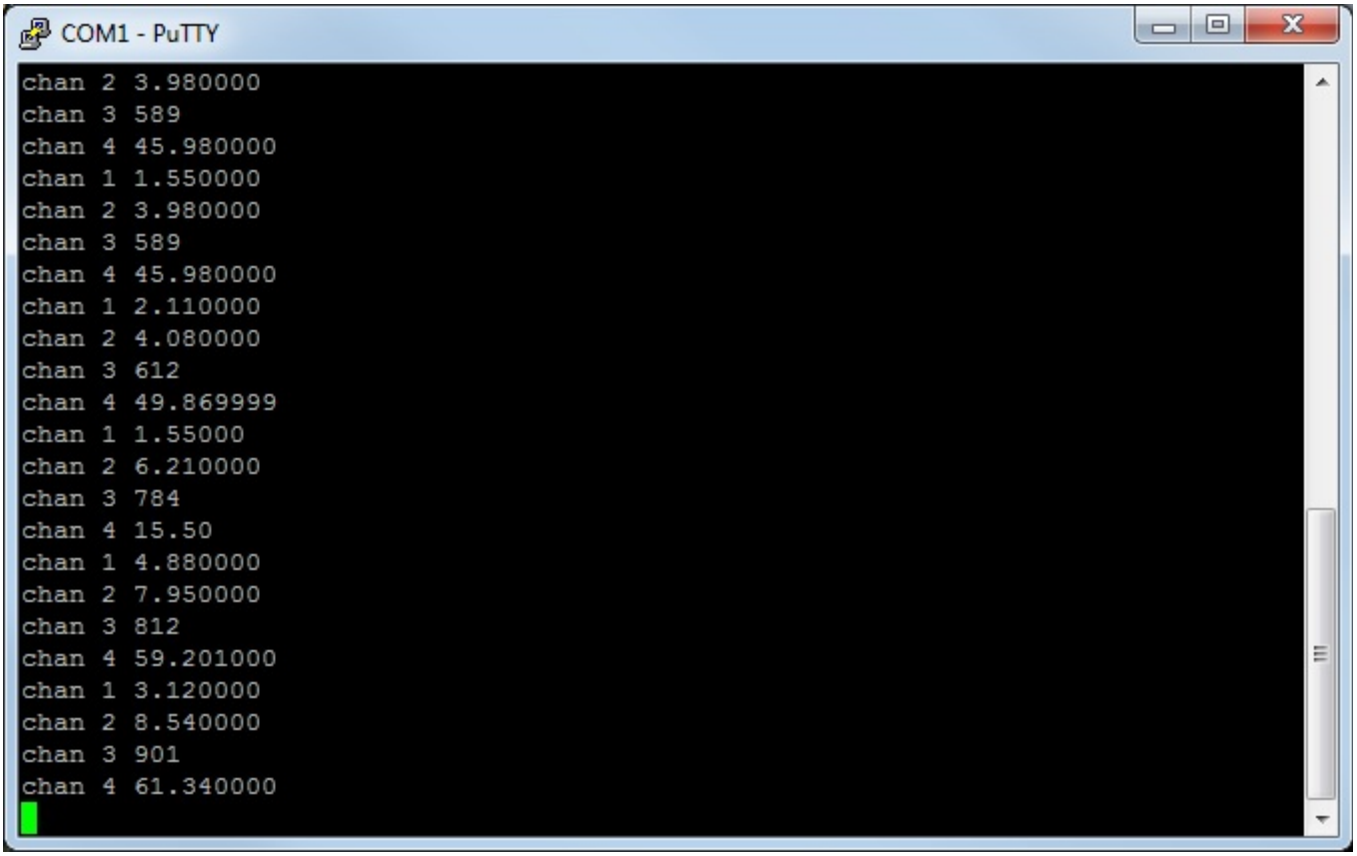
Error 202 from the read map says we are receiving an exception code from the Modbus TCP slave. In this case, we deliberately tried to read a holding register when the slave only had input registers available, for illustration purposes.

This example was created using ModSim as the Modbus TCP slave. By terminating the ModSim program, we no longer have any Modbus TCP device accessible. This results in a timeout indicated by the read map (500) . The device status is indicating 108, which means the IP address is reachable, but Modbus TCP cannot be found there - and this is the expected error in this deliberately flawed example.

13.9 Example: SNMP Enabling a Serial Device

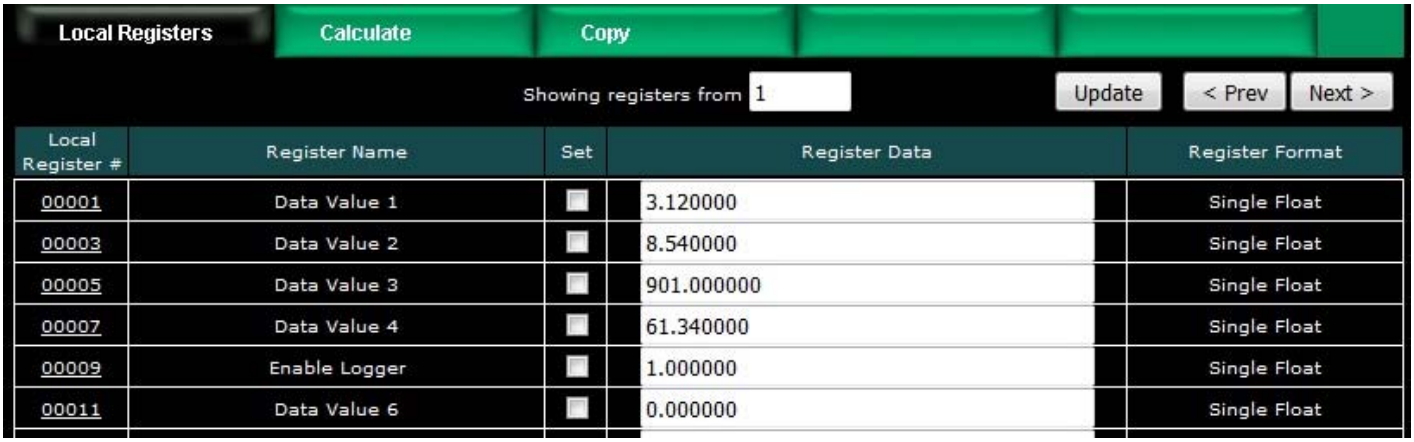
A very useful feature available with the Babel Buster Pro is the ability to disable Modbus RTU and use the serial port for a proprietary serial protcol instead. When Modbus RTU is disabled, the baud rate of the port is set by Basic. The proprietary protocol may be one where you need to send commands to the device in order to receive responses, or it may be a simple serial data logger that periodically sends character strings that you wish to capture.

Our first example will capture data from a data logger type device that periodically sends strings of data. In this case, it is simply a channel number and that channel's data value. This device is a 4-channel device, and its output is illustrated here by simply connecting it to a PC (via RS485 to RS232 adapter) and running PuTTY to see its output.



The program to capture data from this device and store the results in local registers is illustrated below. The key line to notice here is the "split" where the data line is effectively parsed. The "toss$" is going to end up holding the string "chan" which we don't care about. The numbers representing channel number and that channel's data value will end up in the variables "chan" and "data". Then, based on channel number, we calcualte a register number. This is needed because we are going to store the data in floating point registers which occupy 2 Modbus registers worth of address space each.

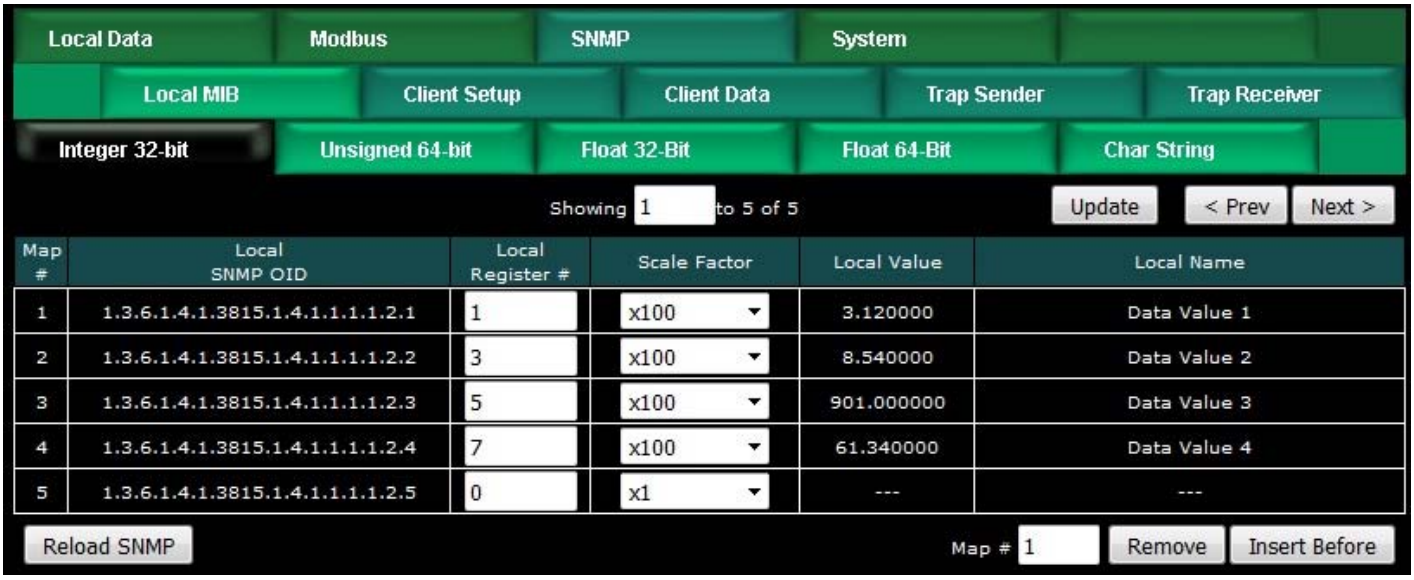
The local register values are going to continue changing as new data is received. The screen shot below illustrates the most recently received data in this example.



We are going to make these data values available via our own internal MIB so that other SNMP devices can do an SNMP Get to retrieve these numbers. To do this, we configure the local MIB to reference the registers we are using. Note that while our local registers are floating point, we are going to provide the results as scaled integers. This is the most universally compatible and interoperable means of sharing SNMP data.

This example provides the data as variables in our local MIB. You could also set

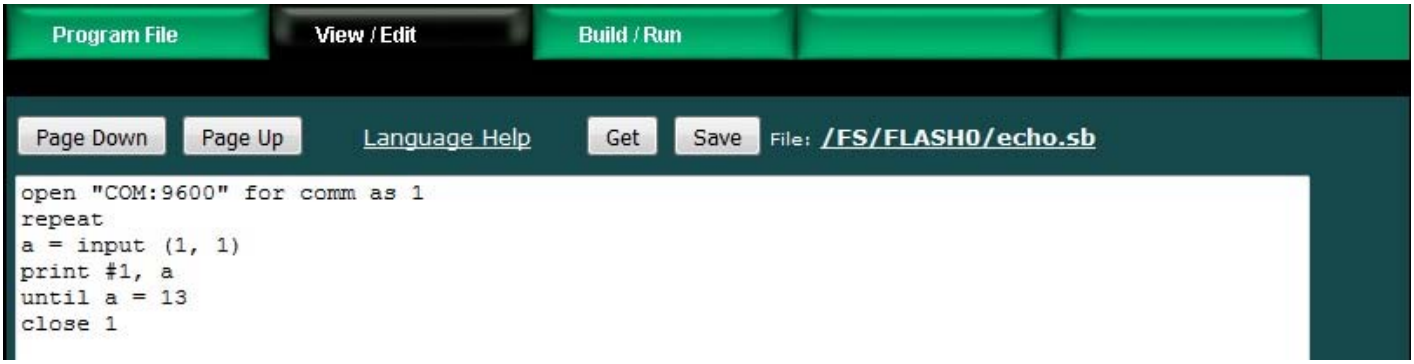
the Babel Buster Pro to send traps based on these numbers exceeding certain thresholds. This would be done as discussed in the Trap Sender portion of this user guide.



The above example is always just receiving data that is sent automatically. If you needed to query a device in order to receive data, you can do that from Script Basic. The following example shows querying one particular type of device. The string printed in the QueryZone function is unique to this device - you need to format whatever is unique to your device.

Note the "timeout" function. This causes the serial port driver to return an empty string if nothing is received after waiting half a second in this example. This allows your Basic program to continue on with other things if this particular device did not respond (e.g. got disconnected).

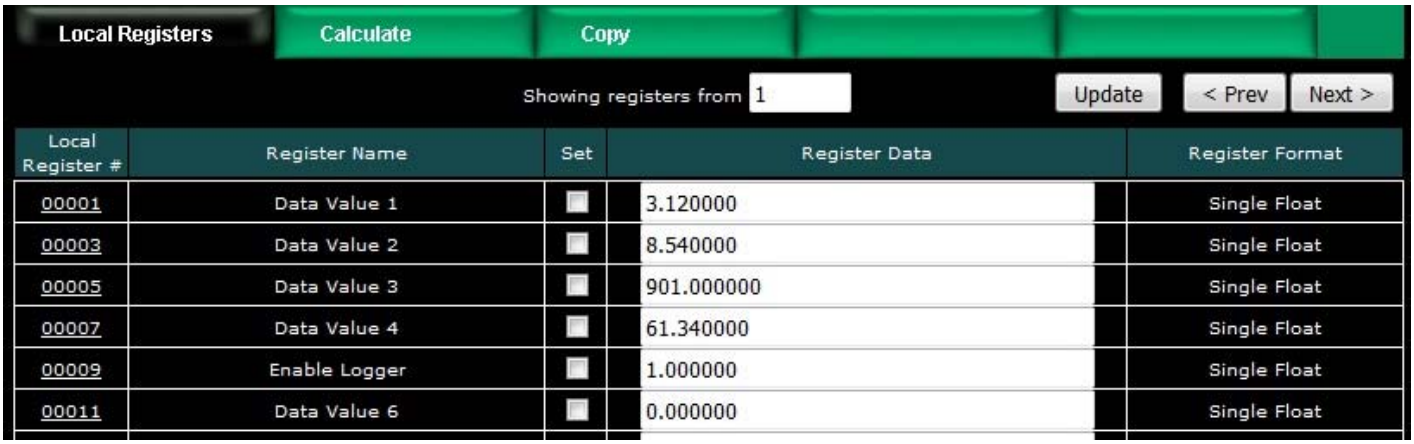
The "line input" in Basic expects to receive a full line terminated by a line end character. If you want to capture one character at a time and not be concerned with whole lines or line end characters, the following exmple illustrates capturing one character at a time and outputing one character at a time to the virtual terminal screen, until that one character is a carriage return - then the program closes the port and termiantes in this simple example.



13.10 Example: Modbus Enabling a Serial Device

You cannot connect both Modbus RTU and a proprietary serial device at the same time - the serial port can only be one or the other. However, if you wanted to use the serial port for the proprietary serial device, you can still make its data available via Modbus TCP.

To Modbus TCP enable the proprietary serial device, follow the previous example (section 13.9) to the point of getting data in the local registers.



Once you have data appearing in the local registers, you can read them via Modbus TCP (or use the Modbus TCP client to write them to another Modbus TCP device). You do not need to configure anything in the local MIB in order to use Modbus TCP.

14. System Configuration and Resources

14.1 Configuration Files

The Config File page is probably one of the most important pages to know about. This is where you tell the gateway to save all of the changes you have made. The various "Update" buttons on the many pages in the web user interface only copy your configuration from your PC's browser to temporary memory in the gateway. To retain those changes indefinitely (i.e. through restart or power cycle), you need to tell the gateway to save those changes in a configuration file.

The configuration files are stored in non-volatile (Flash) memory. The process of reprogramming the Flash takes a little time. It would be cumbersome to rewrite that file every time you made a minor change. Therefore, in the interest of being more responsive, and in the interest of extending the life of the Flash, configuration is only saved to Flash when you direct it to do so.

This is also where you tell the Babel Buster Pro what configuration to automatically load upon startup. To set the startup configuration, select the file from the list, and click the Boot button. The name of the startup file, along with a few other important things like the gateway's own IP address, are stored in a different area of Flash that is not part of the file system. There will be a delay while this area of Flash is reprogrammed after clicking the Boot button.

There is a means of restoring the Babel Buster Pro to "manufacturer's default settings". First of all, make sure that the Boot configuration file is set to "BootConfig.xml". Then, after selecting this file as the boot file, delete it. Now restart the gateway. Upon restart, and upon finding that the boot configuration name is BootConfig.xml, and it does not exist, the gateway will automatically create one with default parameters. The automatic creation of a default file will not occur with any other file name.

To create a new configuration file, enter a name in the new file window (making sure it has a .xml suffix), check the "create new" box, and then click Save.

It is possible to manually edit the XML file outside of the gateway. However, doing so is very prone to errors. If there are errors in the XML file, it will not load successfully on startup. If the configuration does not load on startup, none of the scanners will begin scanning. Because they are all blocked by configuration failure, entering new configuration via the web pages will not result in functionality being restored. You must successfully load a configuration file before the gateway will become functional. To check for errors, select the file here, and click the Load button. Error messages that would have been discarded by the automatic loading at startup will now be displayed on an error page if there are any.

To save a copy of the configuration to your PC, select the file and click the View button. Your browser will now display the XML file. DO NOT do a text copy/paste to try to create an XML file - doing so will result in an invalid file format that cannot be loaded again. You must use the browser's "save as" or "save page" function. The

browser should default to wanting to save a file with a .xml suffix. If correctly saved on your PC, you should be able to double click on the saved file and it will result in opening the file automatically in your browser. It was saved correctly if the browser does not give any error messages when displaying the XML (which should now look exactly as it did when you first clicked the View button).

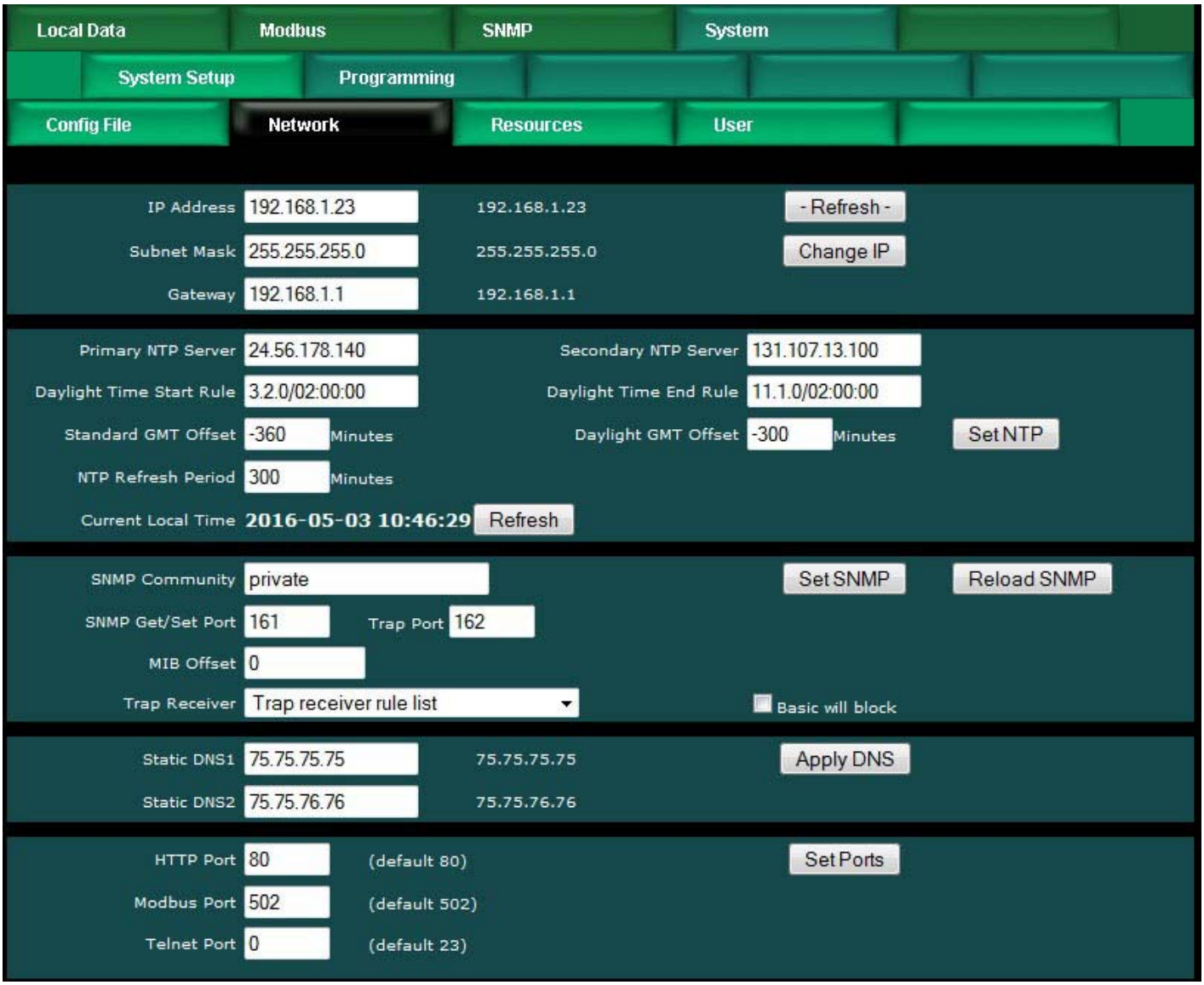
To upload a configuration file from your PC to the gateway, use the Browse button to find the file on your PC, open the file in the PC's file dialog box, and then click Upload. Saving the configuration file to your PC, and then uploading on a different device, is a quick and easy way to configure two Babel Buster Pros the same way.

The other thing found on the Config File page is the means to restart the Babel Buster Pro without a power cycle. Check the restart confirmation box, and click Restart. After a couple of minutes, you will be able to resume browsing the web user interface.

Note: Your browser may cache files. If you view a file, make configuration changes, save the file, then view the file again, you may see the old file cached by the browser. To see the updated file, go to "Internet Options" in your browser's "Tools" menu, and delete temporary Internet files (or delete cache files). Also, if you upload a file, make changes on your PC, and re-upload the same file, the browser may send the old file. Again, you will need to find the button inside your browser options that lets you delete the cached files from your PC.

14.2 Network Configuration

The Network Configuration page is where you set the Babel Buster Pro's IP address as well as a few other important things.



To change the Babel Buster Pro's IP address, enter the IP address, subnet mask, and gateway IP, then click Change IP. The new IP address will not take effect until the device is restarted. The configure IP address is shown in the input window. The actual IP address currently in use is displayed next to the input window.

If you want the Babel Buster Pro to request a dynamic IP address from a DHCP server upon startup, enter 255.255.255.255 for IP address. This is the "flag" that the Babel Buster should use DHCP. However, if no DHCP server is present, the Babel Buster will not boot up.

The Babel Buster Pro maintains time and date via SNTP services. If you are writing a Script Basic program that wants to know time or date, you will need to provide NTP server addresses, set the daylight savings rules, and provide GMT offset. Additional directions for creating the daylight rules are provided in the Quick Help section of the Network configuration web page in the device.

The SNMP section is used to enter the SNMP community that will be used by the Babel Buster Pro as its own. Community strings needed to access other devices are

provided in the SNMP client configuration. The ports are normally 161 for SNMP Get/Set and 162 for SNMP traps.

The MIB offset lets you effectively move the entire MIB. This is required if more than one Babel Buster Pro are going to be used on the same network but their configuration is different. Most SNMP managers do not know how to deal with MIBs that have the same set of OIDs but which mean different things in different devices using the "same" MIB. If the offset is changed, you will need to click Reload SNMP to cause it to take effect.

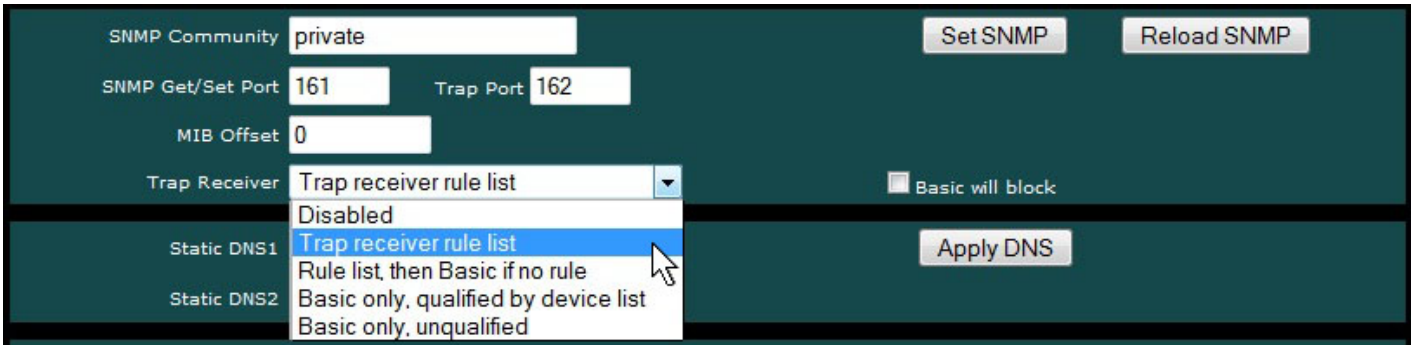
To save changes made on this page, click Set SNMP. To reload the Babel Buster Pro's MIB tables because they have changed (on other configuration pages) click the Reload SNMP button. The Reload SNMP button is also replicated on those pages where you would make changes that will require reloading.

DNS servers will need to be provided if you reference any devices in your configuration by domain name rather than static IP address. Enter the static IP addresses of the DNS servers and click Apply DNS.

The HTTP port for browsing the user interface can be moved away from the default HTTP port 80. Select a different port, click Set Ports, and then restart the gateway to make that new port take effect. Don't forget to append the port number to the gateway's IP address when attempting to browse the web user interface if it has been moved from port 80.

The Modbus port will be set to zero, meaning Modbus TCP is disabled, when the device is new. Enter the standard port 502 and click Set Ports to set the Modbus port. Set the port to some other port if you know that Modbus TCP operates on a non-standard port on your network. The device needs to be restarted after changing the Modbus TCP port.

Telnet is disabled by default (port set to zero). Enable it by setting the port that Telnet should use, then restart the device to activate Telnet. The only meaningful thing you can do via Telnet if needed is to restore the MAC address in the unlikely event it has become corrupted.



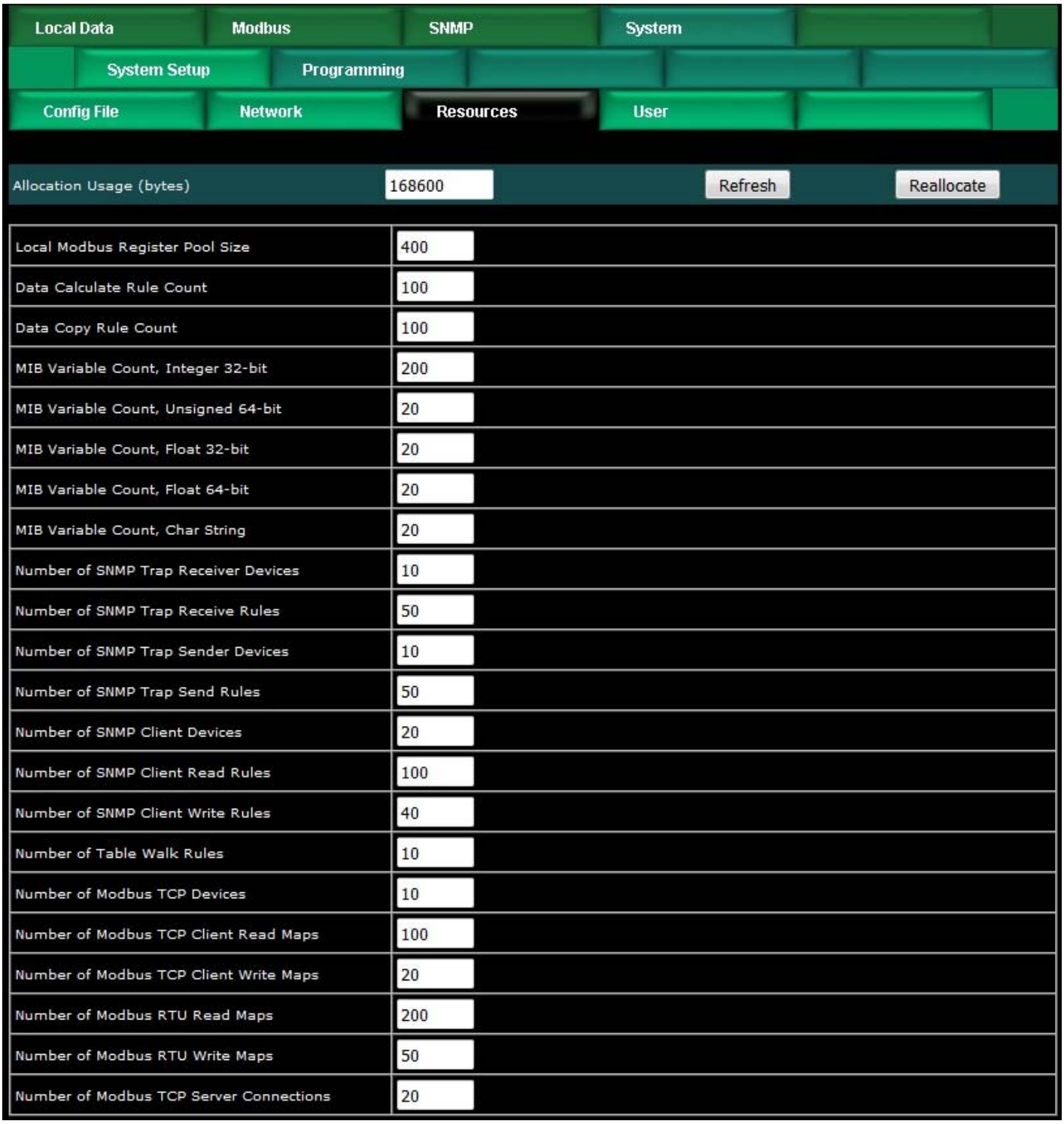
The one other SNMP setting that was not mentioned above was enabling the trap receiver. Do this in conjunction with configuring the trap receiver (see section 12). The options for enabling the trap receiver are illustrated above. The description of what

these mean is found in section 12.

14.3 Resource Allocation

Historically, Control Solutions gateways had a fixed set of Modbus registers and other resources to work with. Invariably, there was always somebody that wanted less of this and more of that. Therefore, while there are still maximums imposed, you can now shift resources around as needed. The default values as shipped are shown below.

IMPORTANT: To change allocation values, enter the new values, then click Reallocate. If you click Refresh instead, the original numbers will be restored. To complete the reallocation process, after entering new numbers and clicking Reallocate, you must go to the Config File page, Save your configuration, and then click Load to reload the file just saved. This reloading of the configuration will reconfigure the device using the new allocation numbers.



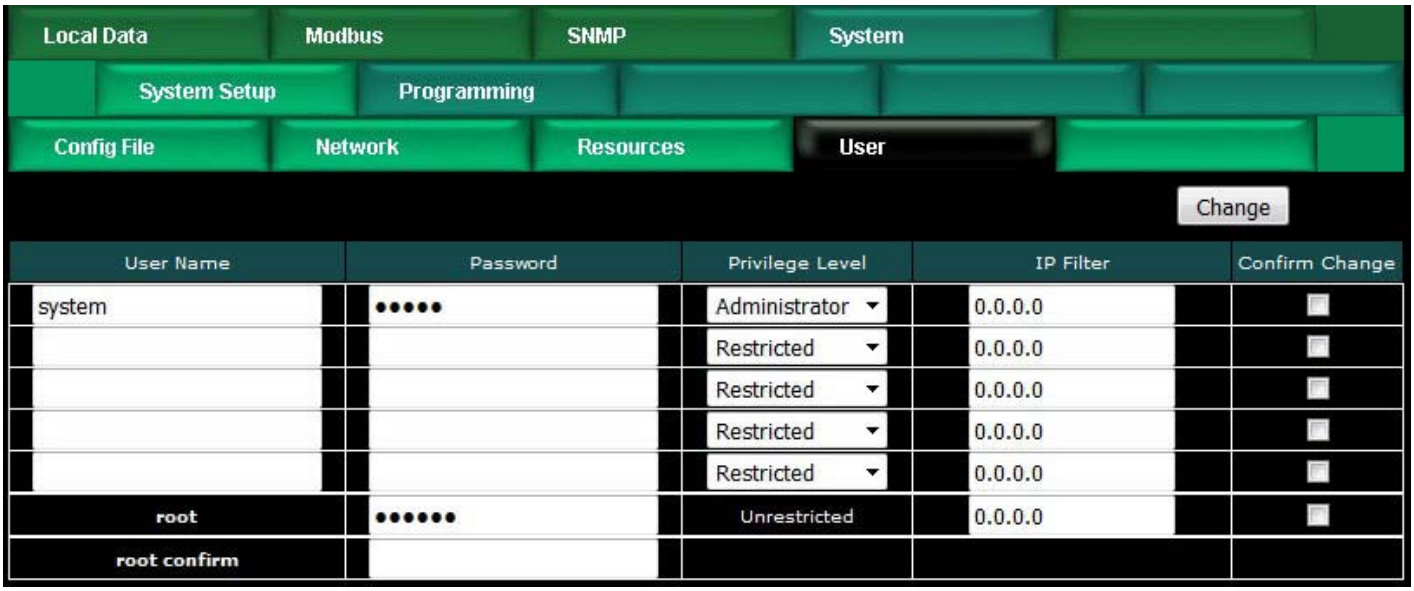
There is no magic formula for determining the "right" size configuration. All resources are allocated from "free memory". This free memory is also used by Script Basic for its variables. It is also used by the SNMP engine for temporary variable buffering while an external SNMP client/manager is sending Get/Set requests to the Babel Buster Pro. With the numbers of registers and various rules provided by default in older Control Solutions gateways, the remaining memory allowed successful MIB walks of up to approximately 400 MIB variables. If you are starting to run short on memory, a failed MIB walk will usually be the first place you see this.

14.4 User Login Passwords

There are two default logins provided initially. They are username "root" with password "buster", and username "system" with password "admin". When logged in as anybody other than "root", you can only change your own password. But if you are logged in as the root user, you can change all passwords and add user names.

The privilege level Administrator lets that user see and change anything. The privilege level Maintenance allows the user to log in and see (and change) values in the local registers via the Local Registers page, but cannot access any other pages. The Restricted level has no meaning in the Babel Buster Pro (other than block access to everything) since it does not operate as a user defined web server.

You also have the option of IP filtering. If set, then the user can only access Babel Buster Pro's web pages from that IP address. Leave set to 0.0.0.0 to disable filtering.



15. Trouble Shooting

15.1 Modbus RTU Trouble Shooting

You will find message and error counters listed on the Error Counts page under RTU Data. If the Babel Buster Pro is configured as Modbus master, then the Error Counts page will list counts by slave address. If the Babel Buster is configured as Modbus slave, then errors show up on the first line (Unit # 1) regardless of what address the Babel Buster is configured to be.

The Errors: Read Maps and Errors: Write Maps pages will tell you exactly which maps are getting errors when the Babel Buster Pro is configured as Modbus Master.

The most frequent problem is "no response" or timeout. This means the master and slave are not connecting for any of several possible reasions: (a) There is a wiring problem; (b) Port parameters are not configured the same (baud rate, etc); (c) Master's timeout setting is too short.

When it comes to wiring, remember that RS-458 is NOT truly a 2-wire interface as it is commonly referred to. Refer to the RS-485 FAQ under Support at csimn.com if you have questions or concerns about wiring.

If you are getting CRC errors, that is almost always a wiring problem, but can be a port problem such as mismatched parity setting. A CRC error will not be caused by incorrect configuration of a Read Map or Write Map.

If you are getting exception errors, that is somewhat good news - it means that at least you are successfully communicating. An exception error most often means the master is asking the slave for a register that the slave does not have. If the Babel Buster Pro is configured as Modbus master, this means the Read Map or Write Map is not configured correctly.

15.2 Modbus TCP Trouble Shooting

You will find message and error counters listed on the Error Counts page under TCP Data for Modbus client activity. Counts will be listed by device number for those devices found on the TCP Setup Devices page.

The Errors: Read Maps and Errors: Write Maps pages will tell you exactly which maps

are getting errors when the Babel Buster Pro is operating as Modbus TCP client (master).

The most frequent problem is "no response" or timeout. The most common cause of this problem for Modbus TCP is a network configuration problem, such as incorrect IP address or IP address that cannot be reached as configured. The problem sometimes lies outside the Babel Buster and may require consulting with the IT personnel responsible for the network if on a large network.

If you are getting exception errors, that is somewhat good news - it means that at least you are successfully communicating. An exception error most often means the master is asking the slave for a register that the slave does not have. If the Babel Buster Pro is configured as Modbus master, this means the Read Map or Write Map is not configured correctly.

15.3 SNMP Trouble Shooting

Assuming you have IP addresses configured correctly and the SNMP ports are open through any routers and firewalls between devices, the most common cause of not communicating is a mismatching community string. This is sort of like a password. Without the correct community string, most devices will simply ignore the request, making it look as if there is no connection when in fact there is nothing wrong with the connection.

Another common oversight is that when adding local registers to the local MIB, you need to click the Reload SNMP button at the bottom of the Local MIB pages to cause SNMP to reload its internal tables with the new configuration you just entered. The Local MIB pages are effectively a list of instructions for loading the SNMP MIB, but the MIB is not automatically rebuilt every time you add another line to the Local MIB pages. To reload the MIB according to the list of instructions you provided on the Local MIB web pages, you need to click Reload SNMP. (Note: The Reload SNMP button is found on multiple pages, but they all perform the same function and any of the Reload SNMP buttons will reload all branches of the MIB.)

If you are working with the SNMP Client, working to Get or Set MIB variables in other SNMP devices, the list of possible errors is found in section 9.4 of this user guide.

Two of the most useful tools in trouble shooting SNMP are Wireshark and a MIB browser. Refer to Appendix C for more information about Wireshark, including examples of its use. Examples of testing the local MIB are illustrated in section 8.3.

15.4 Modbus Reference Information

Modbus Register Types

The types of registers referenced in Modbus devices include the following:
• Coil (Discrete Output)

- Discrete Input
- Input Register
- Holding Register

Whether a particular device includes all of these register types is up to the manufacturer. It is very common to find all I/O mapped to holding registers only. Coils are 1-bit registers, are used to control discrete outputs, and may be read or written. Discrete Inputs are 1-bit registers used as inputs, and may only be read. Input registers are 16-bit registers used for input, and may only be read. Holding registers are the most universal 16-bit register, may be read or written, and may be used for a variety of things including inputs, outputs, configuration data, or any requirement for "holding" data.

Modbus Function Codes

Modbus protocol defines several function codes for accessing Modbus registers. There are four different data blocks defined by Modbus, and the addresses or register numbers in each of those overlap. Therefore, a complete definition of where to find a piece of data requires both the address (or register number) and function code (or register type).

The function codes most commonly recognized by Modbus devices are indicated in the table below. This is only a subset of the codes available - several of the codes have special applications that most often do not apply.

Function Code	Register Type
1	Read Coil
2	Read Discrete Input
3	Read Holding Registers
4	Read Input Registers
5	Write Single Coil
6	Write Single Holding Register
15	Write Multiple Coils
16	Write Multiple Holding Registers

Modbus Exception (error) Codes

When a Modbus slave recognizes a packet, but determines that there is an error in the request, it will return an exception code reply instead of a data reply. The exception reply consists of the slave address or unit number, a copy of the function code with the high bit set, and an exception code. If the function code was 3, for example, the function code in the exception reply will be 0x83. The exception codes will be one of the following:

1	Illegal Function	The function code received in the query is not recognized

		by the slave or is not allowed by the slave.
2	Illegal Data Address	The data address (register number) received in the query is not an allowed address for the slave, i.e., the register does not exist. If multiple registers were requested, at least one was not permitted.
3	Illegal Data Value	The value contained in the query's data field is not acceptable to the slave.
4	Slave Device Failure	An unrecoverable error occurred (for BB2-3060 means corrupt packet was received).
6	Slave Device Busy	The slave is engaged in processing a long-duration command. The master should try again later.
10 (hex 0A)	Gateway Path Unavailable	Gateway could not establish communication with target device. In the case of BB2-3060, will most often mean there is no IP address configured.
11 (hex 0B)	Gateway Target Device Failed to Respond	Specialized use in conjunction with gateways, indicates no response was received from the target device. In the case of BB2-3060, means there was a TCP link failure, unable to connect to TCP target device.
17 (hex 11)	Gateway Target Device Failed to Respond	No response from slave, request timed out.

Modicon convention notation for Modbus registers

Modbus was originally developed by Gould-Modicon, which is presently Schneider Electric. The notation originally used by Modicon is still often used today, even though considered obsolete by present Modbus standards. The advantage in using the Modicon notation is that two pieces of information are included in a single number: (a) The register type; (b) The register number. A register number offset defines the type.

The types of registers referenced in Modbus devices, and supported by Babel Buster gateways, include the following:
- Coil (Discrete Output)
- Discrete Input
- Input Register
- Holding Register

Valid address ranges as originally defined for Modbus were 0 to 9999 for each of the above register types. Valid ranges allowed in the current specification are 0 to 65,535. The address range applies to each type of register, and one needs to look at the function code in the Modbus message packet to determine what register type is being referenced. The Modicon convention uses the first digit of a register reference to identify the register type.

Register types and reference ranges recognized by Babel Buster gateways are as

follows:

0x = Coil = 00001-09999
1x = Discrete Input = 10001-19999
3x = Input Register = 30001-39999
4x = Holding Register = 40001-49999

Translating references to addresses, reference 40001 selects the holding register at address 0000, most often referred to as holding register number 1. The reference 40001 will appear in documentation using Modicon notation, but Babel Buster gateways require specifying "holding register" and entering that register number as just "1".

On occasion, it was necessary to access more than 10,000 of a register type using Modicon notation. Based on the original convention, there is another defacto standard that looks very similar. Additional register types and reference ranges recognized by Babel Buster gateways are as follows:

0x = Coil = 000001-065535
1x = Discrete Input = 100001-165535
3x = Input Register = 300001-365535
4x = Holding Register = 400001-465535

If registers are 16-bits, how does one read Floating Point or 32-bit data?

Modbus protocol defines a holding register as 16 bits wide; however, there is a widely used defacto standard for reading and writing data wider than 16 bits. The most common are IEEE 754 floating point, and 32-bit integer. The convention may also be extended to double precision floating point and 64-bit integer data.

The wide data simply consists of two consecutive "registers" treated as a single wide register. Floating point in 32-bit IEEE 754 standard, and 32-bit integer data, are widely used. Although the convention of register pairs is widely recognized, agreement on whether the high order or low order register should come first is not standardized. For this reason, many devices, including all Control Solutions gateways, support register "swapping". This means you simply check the "swapped" option (aka "High reg first" in some devices) if the other device treats wide data in the opposite order relative to Control Solutions default order.

Control Solutions Modbus products all default to placing the high order register first, or in the lower numbered register. This is known as "big endian", and is consistent with Modbus protocol which is by definition big endian.

What does notation like 40001:7 mean?

This is a commonly used notation for referencing individual bits in a register. This particular example, 40001:7, references (Modicon) register 40001, bit 7. Bits are generally numbered starting at bit 0, which is the least significant or right most bit in

the field of 16 bits found in a Modbus register.

How do I read individual bits in a register?

Documentation tends to be slightly different for every Modbus device. But if your device packs multiple bits into a single holding register, the documentation will note up to 16 different items found at the same register number or address. The bits may be identified with “Bn” or “Dn” or just “bit n”. Most of the time, the least significant bit will be called bit 0 and the most significant will be bit 15. It is possible you could find reference to bit 1 through bit 16, in which case just subtract one from the number to reference the table below.

You cannot read just one bit from a holding register. There is no way to do that - Modbus protocol simply does not provide that function. You must read all 16 bits, and then test the individual bit you are interested in for true or false (1 or 0). Babel Buster gateways provide an automatic way of doing that by including a "mask" in each register map or rule. Each time the register is read, the mask will be logically AND-ed with the data from the register, and the result will be right justified to yield a 1 or 0 based on whether the selected bit was 1 or 0. Babel Buster gateways provide optimization when successive read maps or rules are selecting different bits from the same register. The Modbus register will be read from the slave once, and the 16-bit data will be shared with successive maps or rules, with each map or rule selecting its bit of interest.

The bit mask shown in the expanded form of the Babel Buster RTU read map is a 4 digit hexadecimal (16 bit) value used to mask out one or more bits in a register. The selected bits will be right justified, so a single bit regardless of where positioned in the source register will be stored locally as 0 or 1. The hex bit mask values would be as follows:

B0/D0/bit 0 mask = 0001
B1/D1/bit 1 mask = 0002
B2/D2/bit 2 mask = 0004
B3/D3/bit 3 mask = 0008
B4/D4/bit 4 mask = 0010
B5/D5/bit 5 mask = 0020
B6/D6/bit 6 mask = 0040
B7/D7/bit 7 mask = 0080
B8/D8/bit 8 mask = 0100
B9/D9/bit 9 mask = 0200
B10/D10/bit 10 mask = 0400
B11/D11/bit 11 mask = 0800
B12/D12/bit 12 mask = 1000
B13/D13/bit 13 mask = 2000
B14/D14/bit 14 mask = 4000
B15/D15/bit 15 mask = 8000

Some Modbus devices also back two 8-bit values into a single 16-bit register. The two values will typically be documented as “high byte” and “low byte” or simply have “H” and “L” indicated. If you run into this scenario, the masking for bytes is as follows:

High byte mask = FF00
Low byte mask = 00FF

When the mask value in a Babel Buster gateway is more than just one bit, the mask is still logicalle AND-ed with the data from the Modbus slave, and the entire resulting value is right justified to produce an integer value of less than the original bit width of the original register.

There have been a few instances of documenting packed bits in a 32-bit register. Although Modbus protocol is strictly 16-bit registers, some implementations force you to read pairs of registers. If your device documents 32 packed bits, then you would insert 0000 in front of each mask above, and the remainder of the list would be as follows:

B16/D16/bit 16 mask = 00010000
B17/D17/bit 17 mask = 00020000
B18/D18/bit 18 mask = 00040000
B19/D19/bit 19 mask = 00080000
B20/D20/bit 20 mask = 00100000
B21/D21/bit 21 mask = 00200000
B22/D22/bit 22 mask = 00400000
B23/D23/bit 23 mask = 00800000
B24/D24/bit 24 mask = 01000000
B25/D25/bit 25 mask = 02000000
B26/D26/bit 26 mask = 04000000
B27/D27/bit 27 mask = 08000000
B28/D28/bit 28 mask = 10000000
B29/D29/bit 29 mask = 20000000
B30/D30/bit 30 mask = 40000000
B31/D31/bit 31 mask = 80000000

Deciphering Modbus Documentation

Documentation for Modbus is not well standardized. Actually there is a standard, but not well followed when it comes to documentation. You will have to do one or more of the following to decipher which register a manufacturer is really referring to:

a) Look for the register description, such as holding register, coil, etc. If the documentation says #1, and tells you they are holding registers, then you have holding register #1. You also have user friendly documentation.

b) Look at the numbers themselves. If you see the first register on the list having a number 40001, that really tells you register #1, and it is a holding register. This form of notation is often referred to as the old Modicon convention.

c) Look for a definition of function codes to be used. If you see a register #1, along with notation telling you to use function codes 3 and 16, that also tells you it is holding register #1.

IMPORTANT: Register 1 is address 0. Read on…

d) Do the numbers in your documentation refer to the register number or address? Register #1 is address zero. If it is not clear whether your documentation refers to register or address, and you are not getting the expected result, try plus or minus one for register number. All Control Solutions products refer to register numbers in configuration software or web pages. However, some manufacturers document their devices showing address, not register numbers. When you have addresses, you must add one when entering that register into configuration software from Control Solutions.

Can I put 2 gateways on the same Modbus network?

You can not have more than one Master on a Modbus RTU (RS-485) network. Therefore, if the gateway is to be configured as the Master, you can only have 1 gateway. You cannot use multiple gateways to read more points from the same Modbus slave device.

Multiple gateways configured as slaves can reside on the same Modbus RS-485 network.

If you are using RS-232 devices, you can have only two devices total, regardless of how they are configured. RS-232 is not multi-drop.

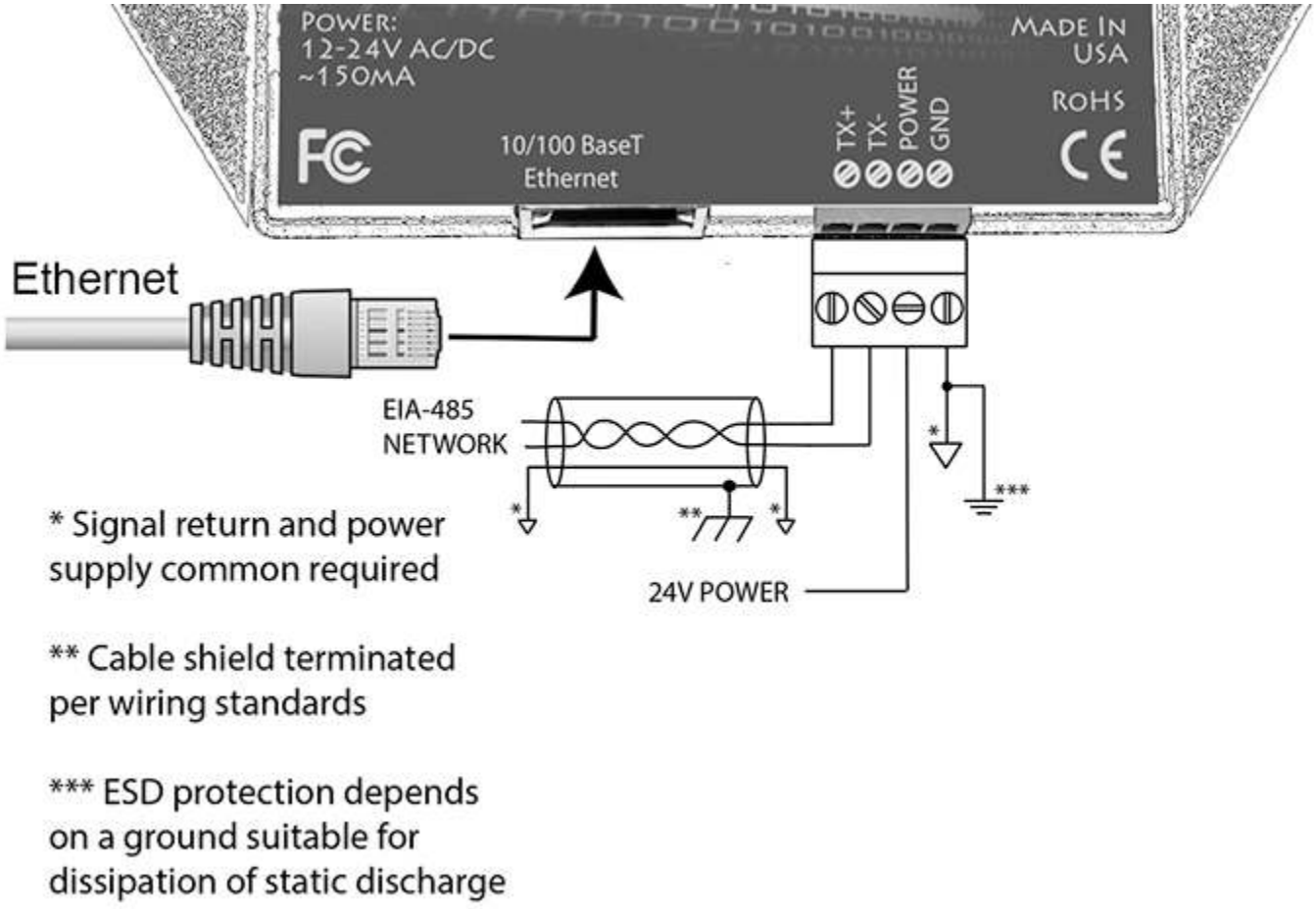
How many devices can I have on a Modbus RTU network?

Logically you can address over 250 devices; however, the RS-485 transceivers are not capable of physically driving that many devices. Modbus protocol states that the limit is 32 devices, and most RS-485 transceivers will agree with this. Only if all devices on the network have low load transceivers can you have more than 32 devices.

Appendix A Hardware Details

A.1 Wiring

Wiring for the Babel Buster SPX Pro V2 is illustrated below.



Wire the gateway as illustrated. Follow all conventional standards for wiring of EIA-485 networks when connecting the Modbus RTU EIA-485 (RS485) network. This includes use and termination of shield, termination of the network, and grounding.

IMPORTANT: Although EIA-485 (RS485) is thought of as a 2-wire network, you MUST include a third conductor connected to GND or common at each device so that all devices are operating at close to the same ground potential. Proper grounding of equipment should ensure proper operation without the third conductor; however, proper grounding often cannot be relied upon. If large common mode voltages are present, you may even need to insert optically isolated repeaters between EIA-485

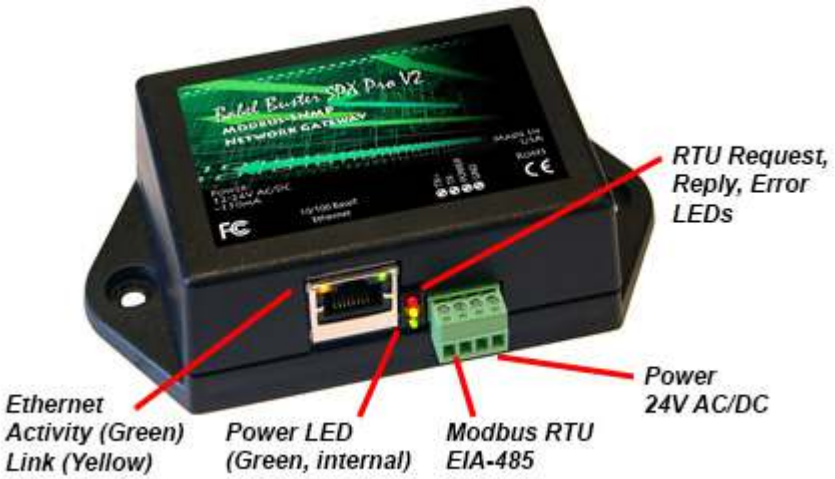
devices.

Use standard CAT5 cables for Ethernet connections. Use control wire as applicable for local electrical codes for connecting the 24V (AC or DC) power supply.

Note that in addition to connecting power supply common to a GND terminal, you must also connect a GND terminal to earth ground in order to ensure proper ESD protection.

A.2 Front Panel LED Indicators

Power-up LED behavior: Following a server boot-up delay, all LEDs on front panel will turn on for half a second. Then they will proceed to indicate as normally defined for the indicators.



Babel Buster SPX Pro V2 request/reply LEDs reflect RTU traffic while the Ethernet activity LED will indicate TCP traffic. To see TCP errors, one needs to look at the Errors page in the web UI.

Babel Buster SPX Pro V2 LEDs indicate as follows:

Yellow	Flashes each time a request is sent when operating as Modbus Master, or each time a request is received when operating as Modbus Slave.
Green	Operating as Modbus Master, flashes each time a good response is received. Operating as Modbus Slave, flashes each time a good response is sent.
Red	Operating as Modbus Master, flashes when an error code is received, the request times out, or there is a flaw in the response such as CRC error. Operating as Modbus Slave, flashes if an exception code is sent (meaning the received request resulted in an error).

Ethernet link LED is the yellow LED integrated into the CAT5 connector. Ethernet activity LED is the green LED integrated into the CAT5 connector.

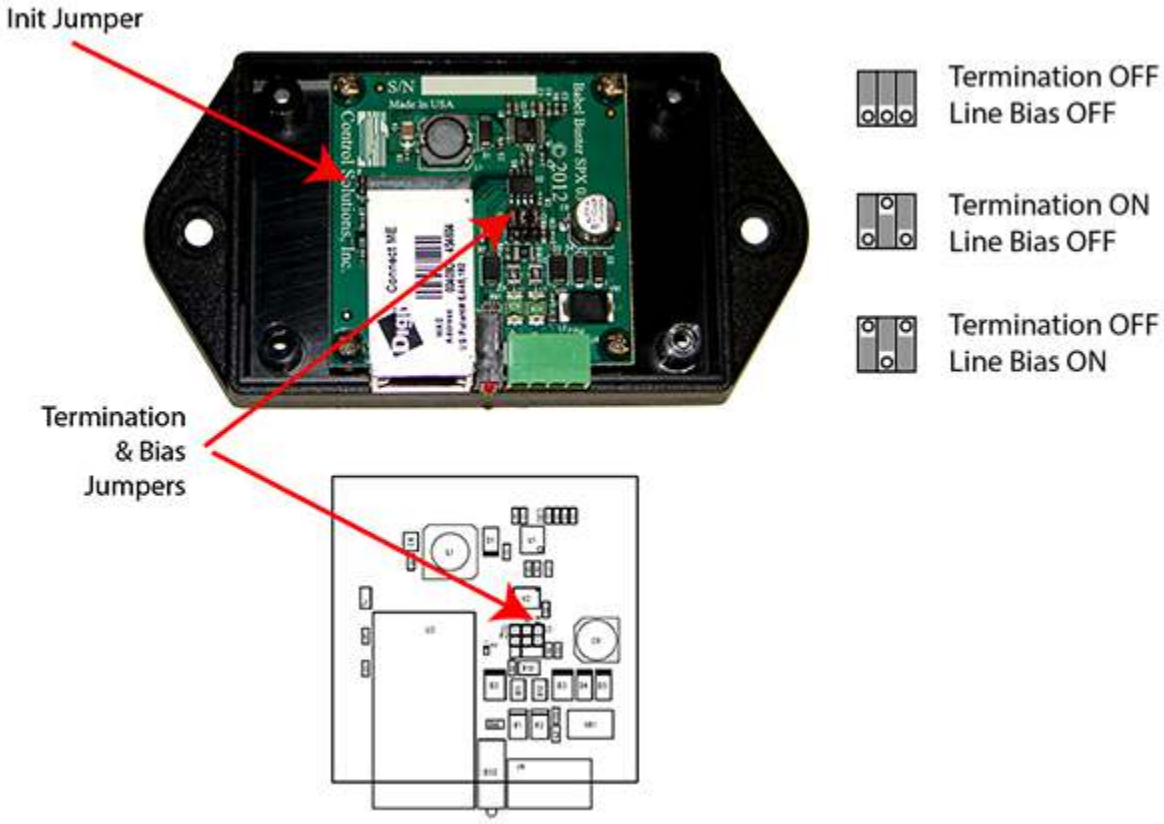
A.3 RS-485 Line Termination & Bias

Enable line termination only when this device is placed at the end of the network. Termination should only be enabled at two points on the network, and these two points must be specifically the end points.

Enable line bias when needed. Line bias should only be enabled at one point on the network, and does not have to be the end point. Line bias holds the line in a known neutral state when no devices are transmitting. Without bias, the transition from offline to online by a transmitter can look like a false start bit and cause loss of communication.

The line conditioning options are enabled when the respective shunt is moved to the position indicated by the diagram below.

Jumper locations for Babel Buster SPX Pro V2:



The "Init" jumper on the server module should only be used when advised by tech support. Installing this jumper prior to power-up causes the server to go into firmware update mode.

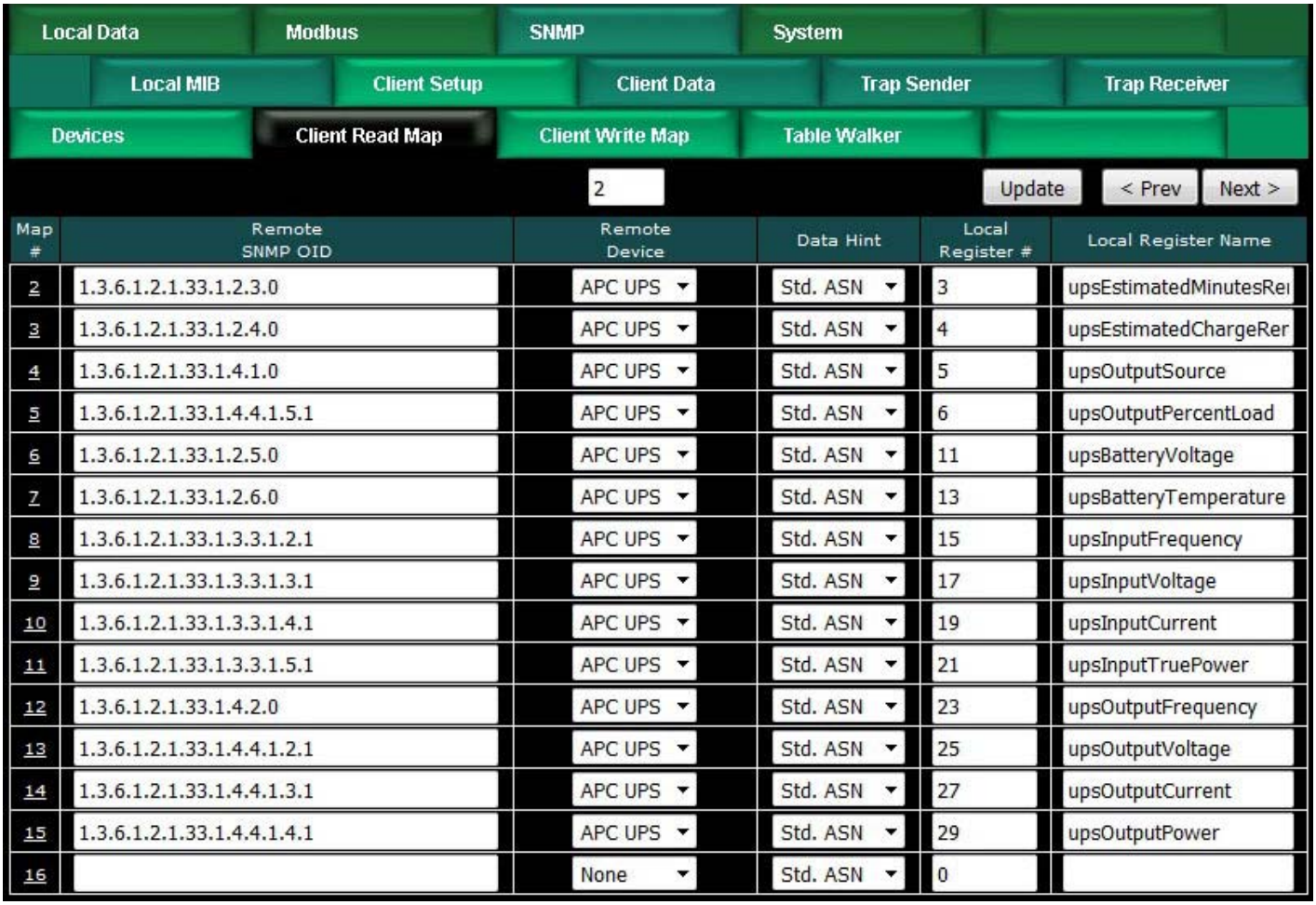
Appendix B Example Application: RFC 1628 UPS

B.1 MIB Query

One of the most useful features of the Babel Buster Pro is its ability to interface to a UPS that implements RFC 1628. The MIB in this type of UPS can be queried to get data like voltage and current, but interpreting its alarms is a bit more complex. There is no way to query some particular OID and see that alarm condition exists. RFC 1628 implements an alarm table that does not lend itself well to simple polling like a gateway would normally like to do.

A configuration file for the Babel Buster Pro named "rfc1628.xml" is available for download at csimn.com. This provides at least a good starting point for getting your UPS connected to Modbus, and may even be all you need. If your UPS is a more complex multi-line UPS, then you will need to use this configuration as an example to build out the full configuration, but if the UPS is a more common single line (single output) UPS, then this configuration will work as is.

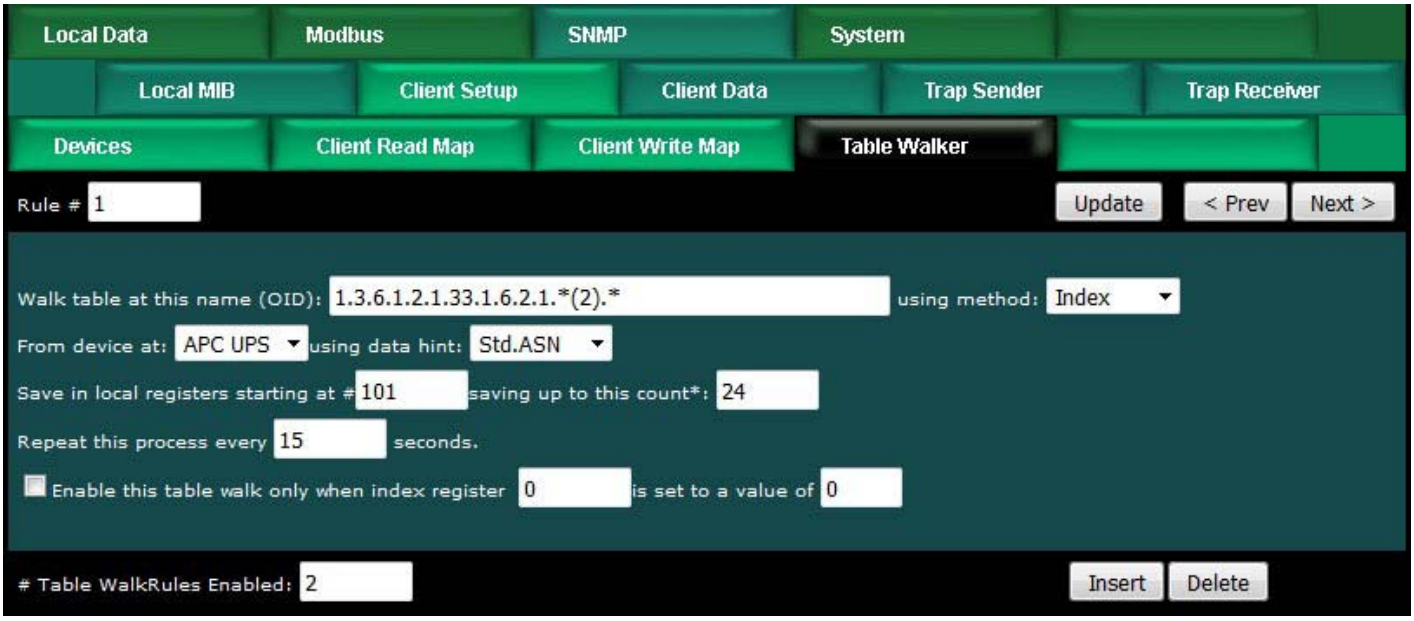
The values available in the MIB that can simply be polled are found in the SNMP Client Read Map list illustrated here.



Map #	Remote SNMP OID	Remote Device	Data Hint	Local Register #	Local Register Name
2	1.3.6.1.2.1.33.1.2.3.0	APC UPS	Std. ASN	3	upsEstimatedMinutesRe
3	1.3.6.1.2.1.33.1.2.4.0	APC UPS	Std. ASN	4	upsEstimatedChargeRer
4	1.3.6.1.2.1.33.1.4.1.0	APC UPS	Std. ASN	5	upsOutputSource
5	1.3.6.1.2.1.33.1.4.4.1.5.1	APC UPS	Std. ASN	6	upsOutputPercentLoad
6	1.3.6.1.2.1.33.1.2.5.0	APC UPS	Std. ASN	11	upsBatteryVoltage
7	1.3.6.1.2.1.33.1.2.6.0	APC UPS	Std. ASN	13	upsBatteryTemperature
8	1.3.6.1.2.1.33.1.3.3.1.2.1	APC UPS	Std. ASN	15	upsInputFrequency
9	1.3.6.1.2.1.33.1.3.3.1.3.1	APC UPS	Std. ASN	17	upsInputVoltage
10	1.3.6.1.2.1.33.1.3.3.1.4.1	APC UPS	Std. ASN	19	upsInputCurrent
11	1.3.6.1.2.1.33.1.3.3.1.5.1	APC UPS	Std. ASN	21	upsInputTruePower
12	1.3.6.1.2.1.33.1.4.2.0	APC UPS	Std. ASN	23	upsOutputFrequency
13	1.3.6.1.2.1.33.1.4.4.1.2.1	APC UPS	Std. ASN	25	upsOutputVoltage
14	1.3.6.1.2.1.33.1.4.4.1.3.1	APC UPS	Std. ASN	27	upsOutputCurrent
15	1.3.6.1.2.1.33.1.4.4.1.4.1	APC UPS	Std. ASN	29	upsOutputPower
16		None	Std. ASN	0	

B.2 Alarm Table Walker

Finding active alarms in an RFC 1628 UPS is a bit trickier than just polling a given MIB. You need to walk the alarm table. This is discussed in much greater detail earlier in this user guide. The particular table walk needed to find alarms in the UPS is illustrated here and included in the rfc1628.xml configuration.

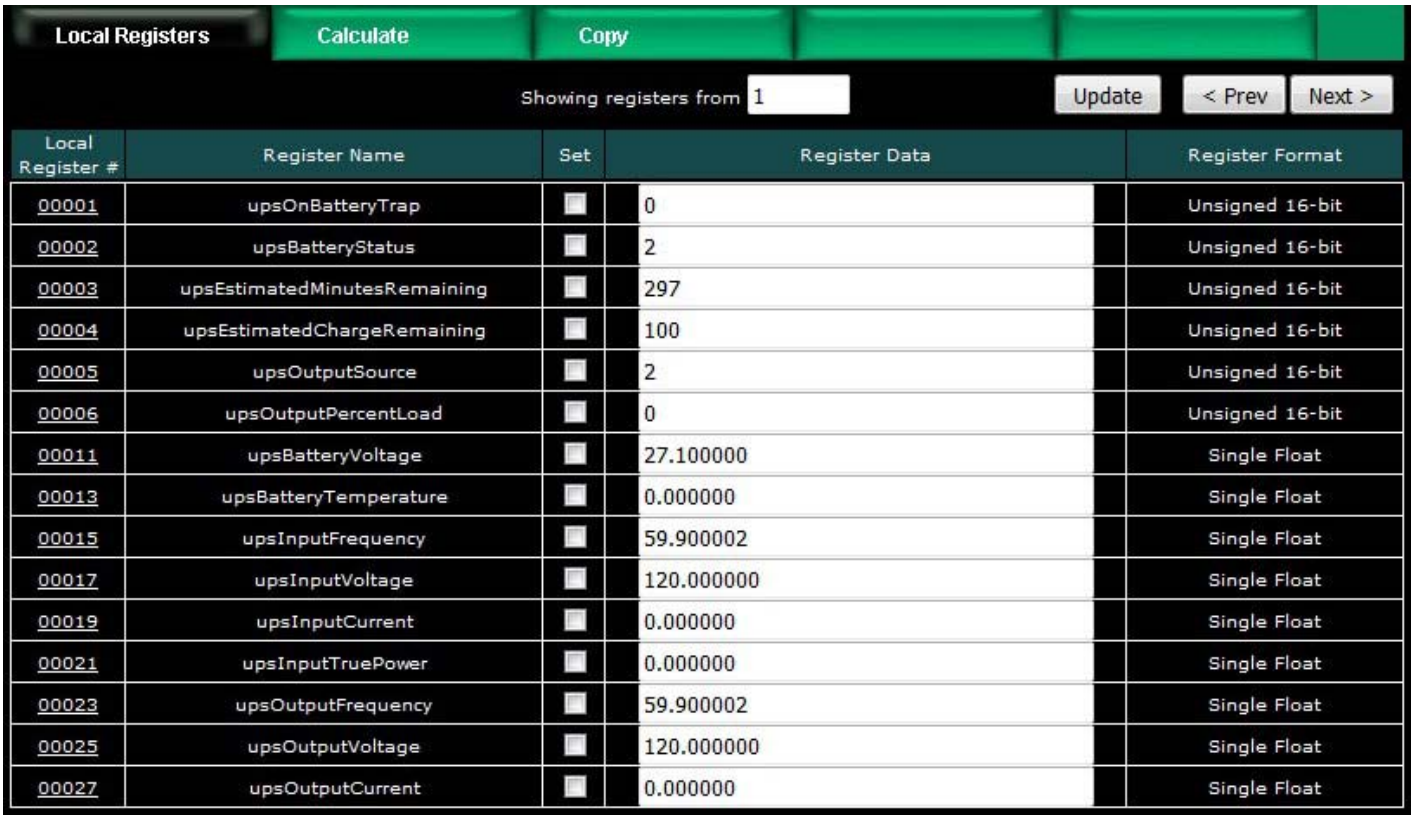


B.3 Trap Receiver

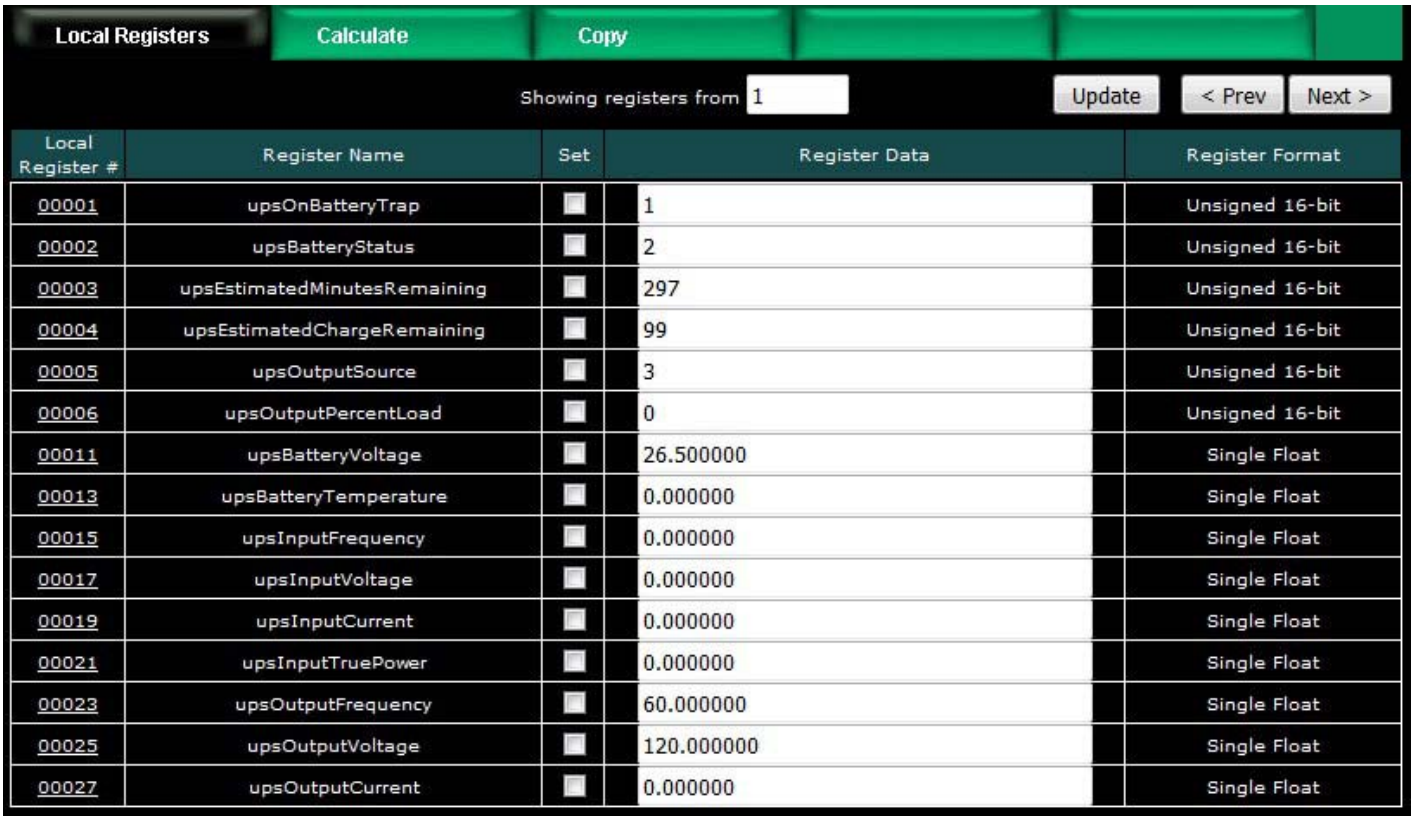
The rfc1628.xml configuration also includes a trap receiver. This trap receiver may be considered redundant with the alarm table walker, but is included as an alternate alarm indication method that may be used if desired.

B.4 Example Data

The polled data when the UPS has normal utility power available is illustrated here.



When utility power is removed and the UPS is running on battery, the data will appear more as follows, except in this test case the UPS does not have a load and therefore current values are zero. Note that register 1 is indiating that the "UPS on battery" trap has been received.

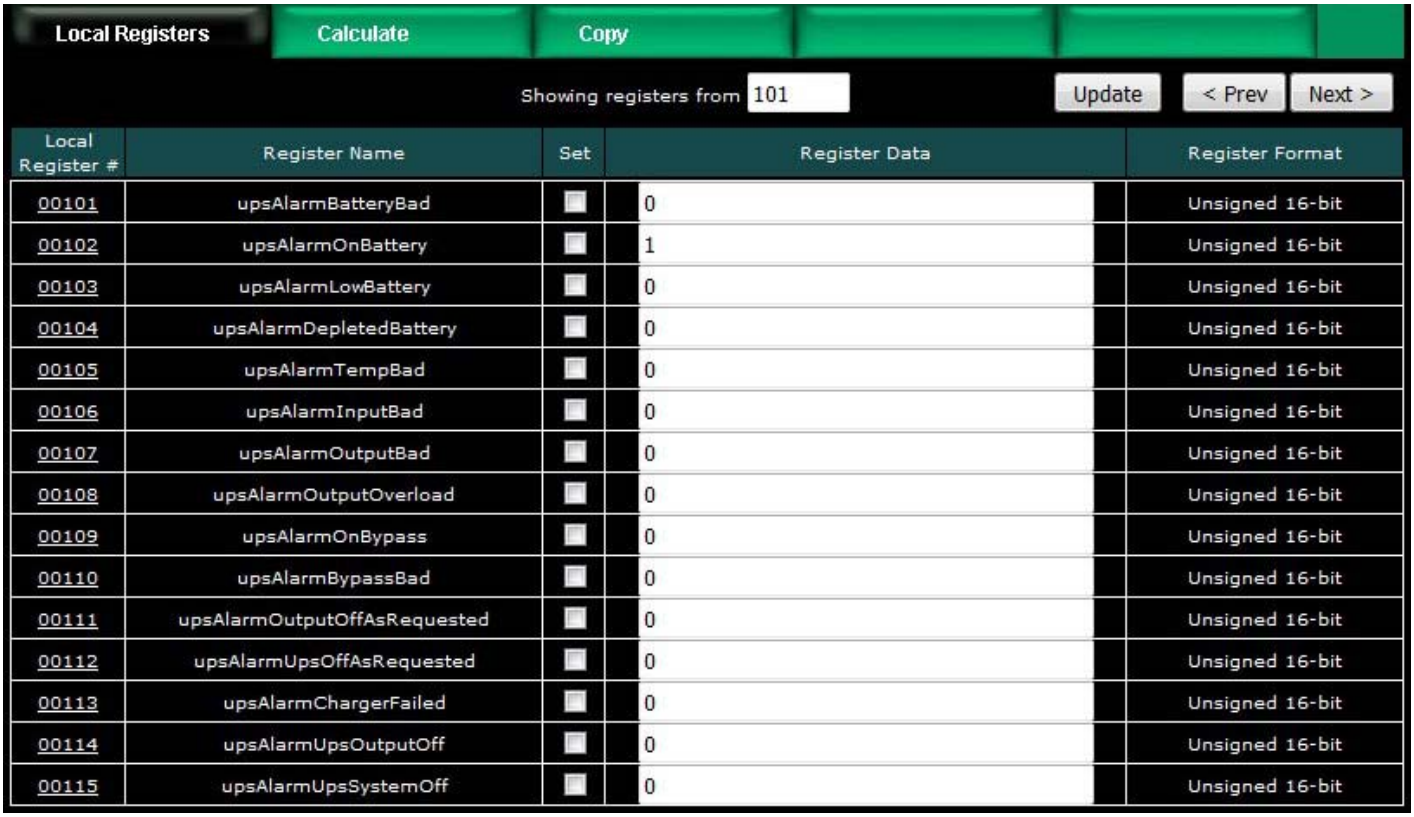


Local Registers | Calculate | Copy

Showing registers from 1 Update < Prev Next >

Local Register #	Register Name	Set	Register Data	Register Format
00001	upsOnBatteryTrap	☐	1	Unsigned 16-bit
00002	upsBatteryStatus	☐	2	Unsigned 16-bit
00003	upsEstimatedMinutesRemaining	☐	297	Unsigned 16-bit
00004	upsEstimatedChargeRemaining	☐	99	Unsigned 16-bit
00005	upsOutputSource	☐	3	Unsigned 16-bit
00006	upsOutputPercentLoad	☐	0	Unsigned 16-bit
00011	upsBatteryVoltage	☐	26.500000	Single Float
00013	upsBatteryTemperature	☐	0.000000	Single Float
00015	upsInputFrequency	☐	0.000000	Single Float
00017	upsInputVoltage	☐	0.000000	Single Float
00019	upsInputCurrent	☐	0.000000	Single Float
00021	upsInputTruePower	☐	0.000000	Single Float
00023	upsOutputFrequency	☐	60.000000	Single Float
00025	upsOutputVoltage	☐	120.000000	Single Float
00027	upsOutputCurrent	☐	0.000000	Single Float

The table walker in this case is setting a series of 24 registers starting at 101 to a value of 1 if the corresponding alarm is active. These may be read as holding registers if the Babel Buster Pro is configured to act as a Modbus slave, or may be read as coils or discrete inputs by the remote Modbus master. These registers will return to zero when the alarm condition clears because the rfc1628.xml configuration includes resetting the registers to a default value of zero after a timeout. If you change the table walk poll rate, be sure to change the default timeouts on these registers as well.



Local Registers | Calculate | Copy

Showing registers from 101 Update < Prev Next >

Local Register #	Register Name	Set	Register Data	Register Format
00101	upsAlarmBatteryBad	☐	0	Unsigned 16-bit
00102	upsAlarmOnBattery	☐	1	Unsigned 16-bit
00103	upsAlarmLowBattery	☐	0	Unsigned 16-bit
00104	upsAlarmDepletedBattery	☐	0	Unsigned 16-bit
00105	upsAlarmTempBad	☐	0	Unsigned 16-bit
00106	upsAlarmInputBad	☐	0	Unsigned 16-bit
00107	upsAlarmOutputBad	☐	0	Unsigned 16-bit
00108	upsAlarmOutputOverload	☐	0	Unsigned 16-bit
00109	upsAlarmOnBypass	☐	0	Unsigned 16-bit
00110	upsAlarmBypassBad	☐	0	Unsigned 16-bit
00111	upsAlarmOutputOffAsRequested	☐	0	Unsigned 16-bit
00112	upsAlarmUpsOffAsRequested	☐	0	Unsigned 16-bit
00113	upsAlarmChargerFailed	☐	0	Unsigned 16-bit
00114	upsAlarmUpsOutputOff	☐	0	Unsigned 16-bit
00115	upsAlarmUpsSystemOff	☐	0	Unsigned 16-bit

Appendix C Using Wireshark for Trouble Shooting

C.1 Hardware Requirements

There are no particular hardware requirements regarding the PC you run Wireshark on. Basically anything running any version of Windows can run Wireshark. There are also Linux and Mac OS X versions.

The "hardware requirement" that is of most concern is the means of connecting to the network. We typically just connect everything Ethernet to a switch and don't worry about it. However, switches are really unmanaged routers, and they filter traffic. Therefore, your PC will not see traffic passing back and forth between two other devices that are not the PC. In order to see that network traffic using Wireshark, you need to come up with the right kind of network connection.

If your PC itself is one end of the network conversation you wish to capture, for example when running Modscan, then Wireshark will capture all network traffic to and from the PC however connected. It is when your PC wants to simply "eavesdrop" that you run into problems with the network switch.

A while back, 10BaseT hubs were common. A 10BaseT hub is not as smart as a switch and does not filter traffic. If you have an old 10BaseT hub collecting dust somewhere, you now have a new use for it. It will let Wireshark see all traffic from the PC that goes between any other devices connected to that 10BaseT hub. Beware of devices calling themselves "hubs" but support 100BaseT connections. These are switches.

Since manufacturers of hubs decided nobody should have a use for them anymore, they are generally out of production. Finding a 10BaseT hub for sale is not easy (try eBay). But there are other alternatives.

One means of monitoring network traffic is to get a managed switch that supports "port mirroring". One such device we have tested is the TP-LINK model TL-SG105E. Setting it up requires utility software (provided with the switch) and takes a little effort to get configured. But once configured, it works well without any further monkeying around. And it is inexpensive.

The other means of monitoring traffic is with the use of a device made specifically for use with Wireshark. The "SharkTap" provides two connections for the network

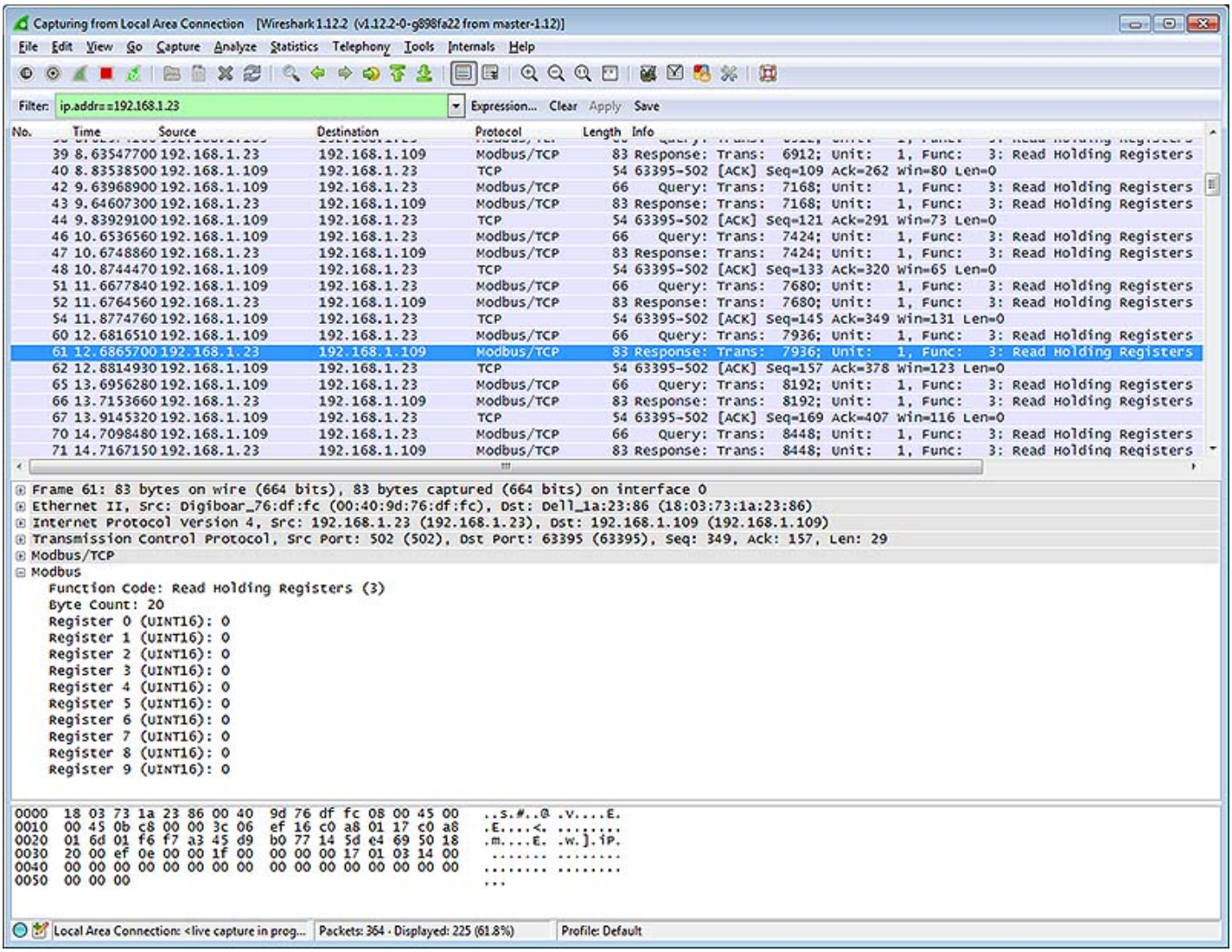
pass-through, and a third "tap" connection where you connect your PC running Wireshark. There is no configuration required. It is the simplest way to monitor network traffic, and it is a current production item available on Amazon (as of 2016).

C.2 Example of Using Wireshark

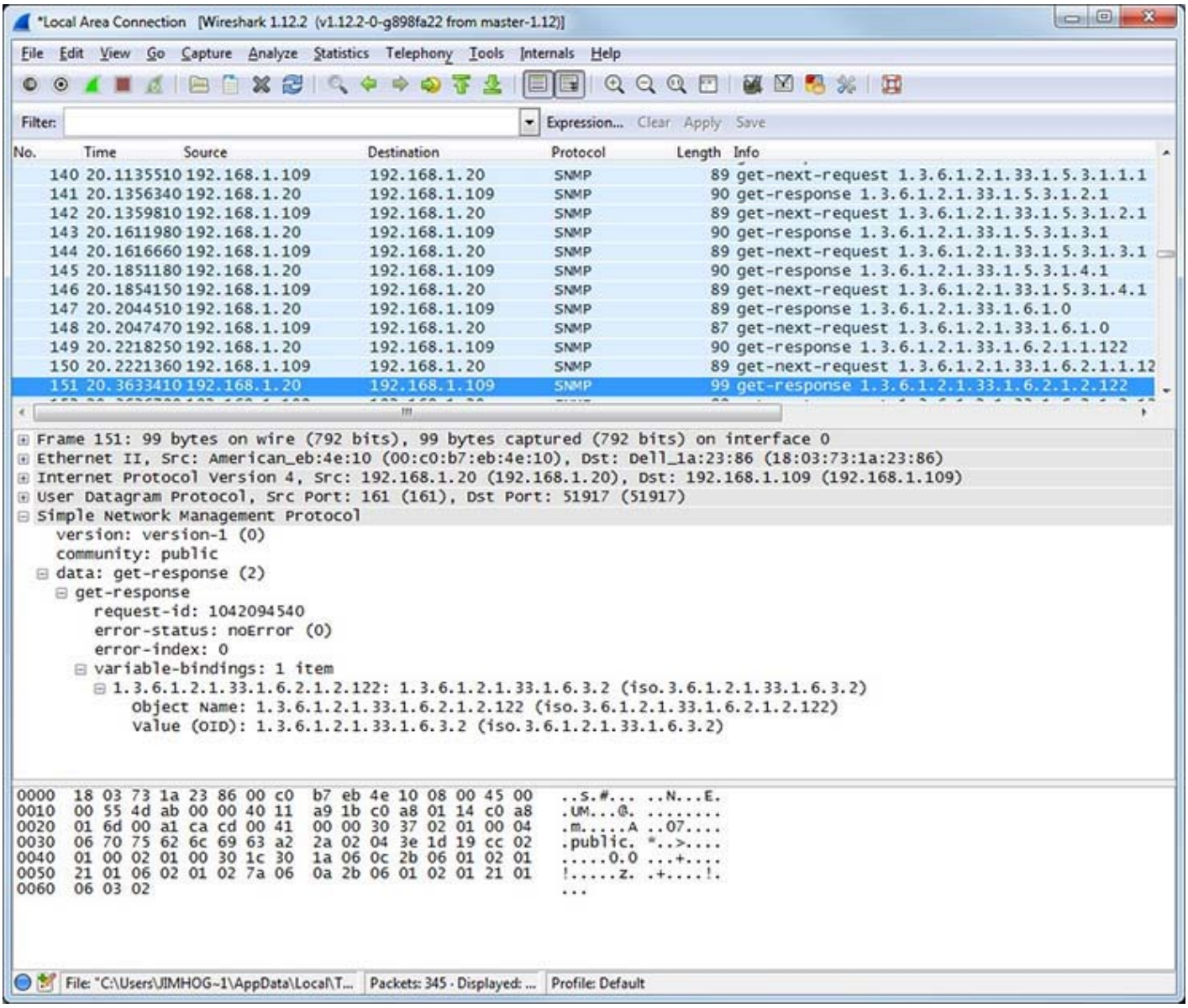
Using Wireshark is fairly easy. Get a copy at www.wireshark.org and install it. Once installed, running it is straight forward. Select your local network interface and click Start on the startup screen. The screen will look something like the example below once Wireshark starts collecting data. Click the red icon in the toolbar to stop capturing traffic. Control Solutions technical support will often ask for a copy of the Wireshark data when a network issue seems evident. You can save a copy of all of the network traffic captured under the File menu, and you will generally save it to a .pcap or .pcapng file. This saved Wireshark log can be posted as an attachment in support tickets.

The screen shot below shows Wireshark capturing Modbus TCP traffic between a client and our Babel Buster Pro. If you click on a packet, the details of that packet will be displayed in the lower part of the screen. You can expand the tree view to see further detail. In the case of Modbus, we will see function code, register count or data count, etc.

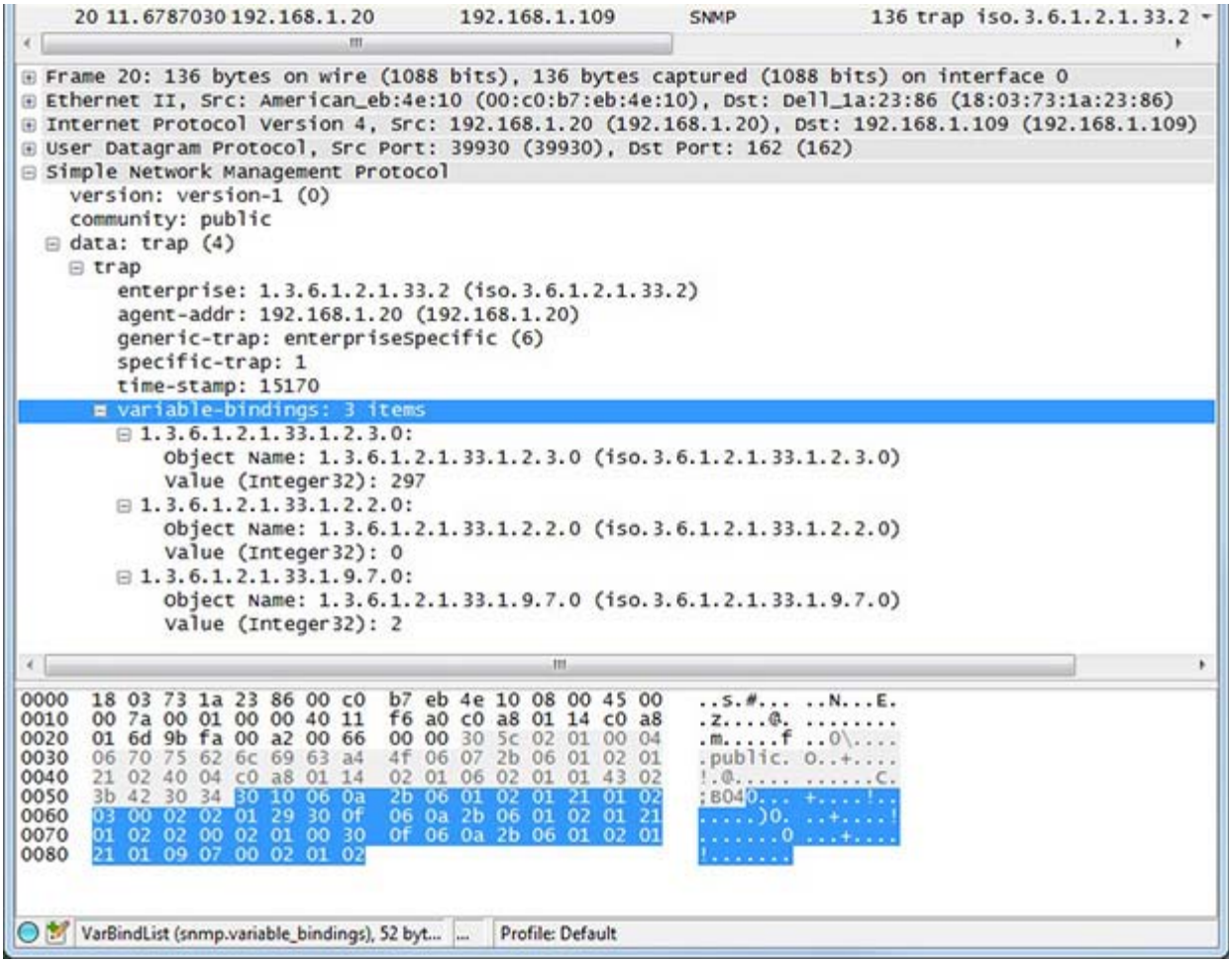
A lot of times you will see a lot of network traffic that is not of interest to you. You can filter network traffic to only display traffic to/from the device you are interested in. Do this by entering "ip.addr==192.168.1.23" in the Filter window as illustrated below. (Substitute your own device's IP address.)



Capturing a series of SNMP traffic will look like the example below. In this example, a table walk is being performed. A table talk is simply a series of Get-Next requests and responses. The table walk typically ends when the response comes back with an OID that is beyond the range of the walk criteria.



If you are working on getting a trap receive rule to work, you will be interested in looking at traps in Wireshark. A trap from an RFC 1628 UPS is illustrated below, with the trap message expanded in the tree view, as well as the varbinds expanded to show OID and value.


20 11.6787030 192.168.1.20 192.168.1.109 SNMP 136 trap iso.3.6.1.2.1.33.2
Frame 20: 136 bytes on wire (1088 bits), 136 bytes captured (1088 bits) on interface 0
Ethernet II, Src: American_eb:4e:10 (00:c0:b7:eb:4e:10), Dst: Dell_1a:23:86 (18:03:73:1a:23:86)
Internet Protocol Version 4, Src: 192.168.1.20 (192.168.1.20), Dst: 192.168.1.109 (192.168.1.109)
User Datagram Protocol, Src Port: 39930 (39930), Dst Port: 162 (162)
Simple Network Management Protocol
version: version-1 (0)
community: public
data: trap (4)
trap
enterprise: 1.3.6.1.2.1.33.2 (iso.3.6.1.2.1.33.2)
agent-addr: 192.168.1.20 (192.168.1.20)
generic-trap: enterpriseSpecific (6)
specific-trap: 1
time-stamp: 15170
variable-bindings: 3 items
1.3.6.1.2.1.33.1.2.3.0:
Object Name: 1.3.6.1.2.1.33.1.2.3.0 (iso.3.6.1.2.1.33.1.2.3.0)
Value (Integer32): 297
1.3.6.1.2.1.33.1.2.2.0:
Object Name: 1.3.6.1.2.1.33.1.2.2.0 (iso.3.6.1.2.1.33.1.2.2.0)
Value (Integer32): 0
1.3.6.1.2.1.33.1.9.7.0:
Object Name: 1.3.6.1.2.1.33.1.9.7.0 (iso.3.6.1.2.1.33.1.9.7.0)
Value (Integer32): 2
0000 18 03 73 1a 23 86 00 c0 b7 eb 4e 10 08 00 45 00 ..s.#... ..N...E.
0010 00 7a 00 01 00 00 40 11 f6 a0 c0 a8 01 14 c0 a8 .z....@. ....
0020 01 6d 9b fa 00 a2 00 66 00 00 30 5c 02 01 00 04 .m.....f ..0\....
0030 06 70 75 62 6c 69 63 a4 4f 06 07 2b 06 01 02 01 .public. O..+....
0040 21 02 40 04 c0 a8 01 14 02 01 06 02 01 01 43 02 !.@..... ......C.
0050 3b 42 30 34 30 10 06 0a 2b 06 01 02 01 21 01 02 ;B040... +....!..
0060 03 00 02 02 01 29 30 0f 06 0a 2b 06 01 02 01 21 .....)0. ..+....!
0070 01 02 02 00 02 01 00 30 0f 06 0a 2b 06 01 02 01 ....0 ...+....
0080 21 01 09 07 00 02 01 02 !.......
VarBindList (snmp.variable_bindings), 52 byt...
Profile: Default