User Guide

Babel Buster 3

Model BB3-6101-V2C
Model MX-61-V2C

Modbus-SNMP Gateway
with SNMPv2 and Client
Rev. 1.2 – June 2021

User Guide Contents

Babel Buster® 3
MODEL BB3-6101
MODEL MX-61
NETWORK GATEWAY

CONTROL SOLUTIONS MINNESOTA

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 Sections 2 and 3, and other sections specific to your intended use. 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 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.

1.3 Warranty

This 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 at any time. This documentation could include technical inaccuracies, typographical errors, and the like. Changes are periodically made to the information herein; these changes may be made without notice.

Product Warranty: All Control Solutions products are warranted against defects in materials and workmanship for a period of time from date of shipment from factory as follows: Two years on non-mechanical parts, one year on mechanical parts (e.g. relays). Defective units will be repaired or replaced, at manufacturer's discretion, at no cost to user except when negligence or improper use has resulted in damage. The express warranty stated herein is in lieu of all other warranties, express or implied, including without limitation any warranties of merchantability or fitness for a particular purpose and all other warranties are hereby disclaimed and excluded by Control Solutions, Inc.

Configuration errors made by customer are not covered under warranty. Damage caused by incorrect electrical connection is not covered under warranty. Removing circuit boards from their enclosures will void the warranty - the complete product with all of its original circuit boards and components must be returned for warranty consideration.

2. Connecting Gateway for the First Time

2.1 Where to Start

The Babel Buster BB3-6101/MX-61 is used to share data between Modbus RTU or Modbus TCP devices and SNMP. If you are looking for a connection to some other protocol, then start by looking for a different model number.

Start by getting familiar with this User Guide and the sections that pertain to your application. Be sure to review the remainder of this section. Online videos are also available to demonstrate key operations in setting up the BB3-6101/MX-61.

You will need to obtain information such as a "register map" from the vendor of the Modbus device you wish to connect. The register map may go by any of several names in the manufacturer's documentation. You will need to know what registers to read in order to interface a Modbus device to the SNMP network.

If you get stuck, you can open a support ticket at https://ticket.csimn.com where response time is generally 24 hours or less, and often as little as 2 hours, and at no cost.

NOTE: Screen shots throughout this User Guide illustrate BB3-6101; however, the screens in the MX-61 are identical with the only exception being model number indicated at the top of the page.

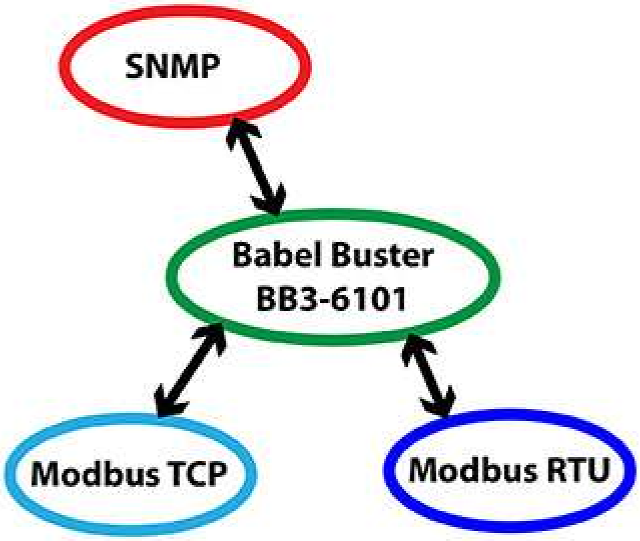
2.2 Overview of Model BB3-6101/MX-61

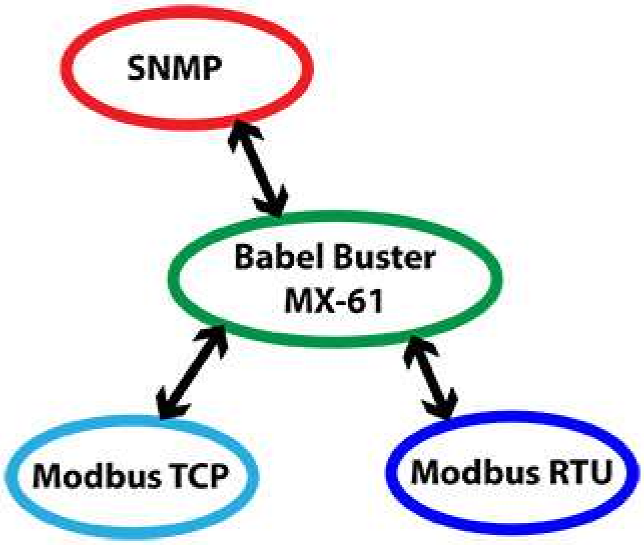
2.2.1 Application of the BB3-6101/MX-61

The Babel Buster BB3-6101/MX-61 is a gateway used to interface between SNMP and Modbus devices. The gateway may be used as an SNMP Agent, Modbus TCP client and server, and Modbus RTU master or slave.

The BB3-6101/MX-61 may be thought of as a data server with multiple network ports that have access to the internal database. The BB3-6101/MX-61 is not a router - SNMP cannot interact directly with Modbus devices or vice versa. Both SNMP and Modbus have access to the internal data objects which are updated according to rules you define.

The BB3-6101/MX-61 can be thought of as a collection of sensors and actuators. The means by which the sensors and actuators interface to hardware will vary by application. In the case of the BB3-6101/MX-61, that hardware consists of one or more Modbus devices which the BB3-6101/MX-61 is set up to automatically interact with, without any constraints on the SNMP side.





The most common application for the BB3-6101/MX-61 is interfacing a Modbus RTU device to the SNMP management system. The BB3-6101/MX-61 will automatically poll the Modbus RTU device, and store the content if its registers in local registers you assign. The SNMP manager may then use the standard SNMP GET to access the content of the Modbus registers.

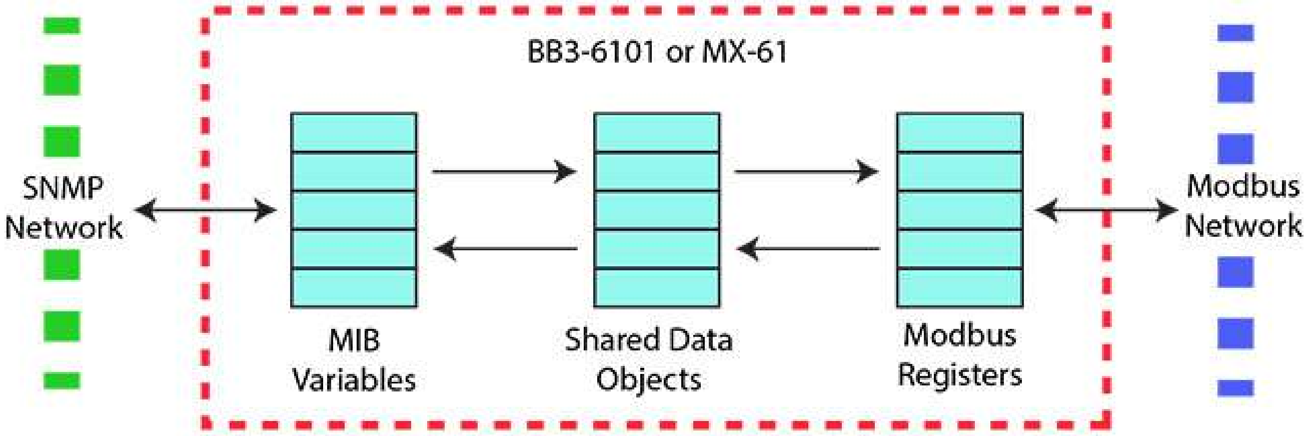
The BB3-6101/MX-61 can be also configured as a Modbus RTU slave. This is useful when a PLC wants to write data to the BB3-6101/MX-61, thereby making the PLC's data available as SNMP MIB variables.

All of the applications noted for Modbus RTU are also available to Modbus TCP with the BB3-6101/MX-61.

2.2.2 Object Server Model for a Gateway

Control Solutions gateways are not simple protocol translators. It is not possible to do an effective job of simply converting one protocol directly to another. Any attempt to do so would likely have negative effects on the networks on both sides of the gateway. An effective solution requires an intelligent device that can properly and efficiently act as a native device on each network. Control Solutions gateways function as two native devices, one on each network, with a shared data base in between them. They function as clients and/or servers on each network.

The central data element in every Control Solutions gateway is an “object”. Each object has rules for accessing that object which are specific to the protocol of the network. Each object has at least two sets of rules, one set for each of the two (or more) networks that may access the object. The object model is often optimized to cater to a specific protocol, and will most often favor the more complex protocol.



Control Solutions gateways will function as servers, providing a copy of the most recent data found in its data base when a client requests that data. In master/slave terms, the server is a slave while the client is a master. Some applications will treat the gateway as a server from both (all) networks connected. But most applications will want the gateway to be a server on one side, and a client on the other side. The most frequent application of the BB3-6101/MX-61 will have it functioning as a Modbus master (client).

Client functionality of a Control Solutions gateway is autonomous. In other words, when acting as a Modbus master (client), the gateway will continuously poll the Modbus slave device(s) on its own, and keep a copy of the most recent data obtained from (or sent to) the Modbus slave device(s). Most often, the gateway is configured to read slave devices periodically, and write to the slave devices when new data is received from a client.

2.3 What is New in Model BB3-6101/MX-61

The BB3-6101 is a significant enhancement over its predecessor, the BB2-6010. The MX-61 is the equivalent enhancement over its predecessor, the SPX. The hardware includes a faster processor and hardware encryption engine for efficient rendering of secure web pages and for support of encryption as needed for SNMPv3. The software includes numerous enhancements.

- SNMPv3 support
- IPv6 support
- Secure (HTTPS) web server
- Faster processor
- Higher point count, up to 1,000 MIB variables
- User defined register map for Modbus
- Greater flexibility in assigning local register data types
- Support for reading character strings from Modbus
- CSV import of register maps for client/master configuration
- Menu options to clear part or all of configuration

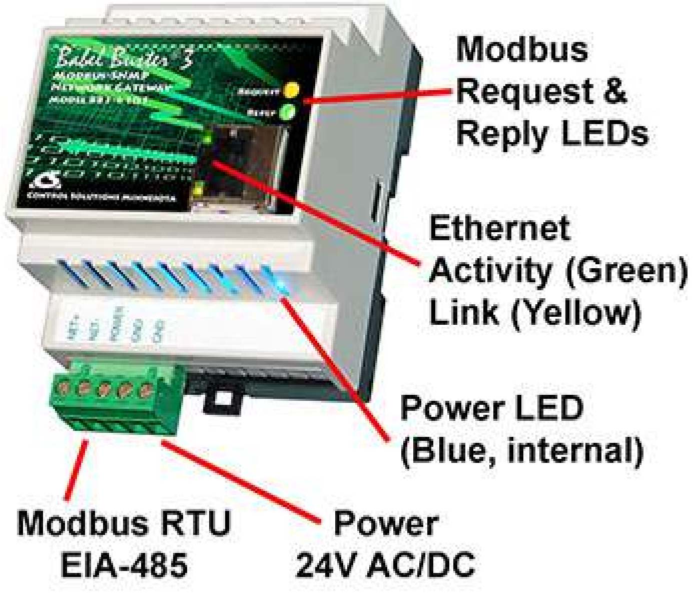
Control Solutions has benchmark tested a configuration in which the BB3-6101 MIB size was 1,500 variables and external SNMP manager tools could successfully walk the entire MIB in a very short time.

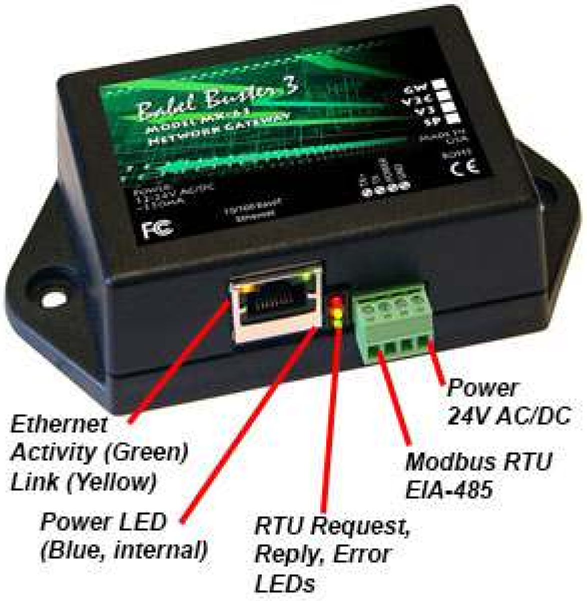
IMPORTANT: Configuration files from older gateways are not directly usable in the BB3-6101/MX-61, but the Babel Buster Configuration Builder program can be used to convert a BB2-6010/SPX configuration into CSV files that may be imported into the BB3-6101/MX-61.

2.4 Connectors and Indicators

Follow these steps to make the initial connection to the BB3-6101/MX-61.

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


Modbus Request & Reply LEDs
Ethernet Activity (Green) Link (Yellow)
Power LED (Blue, internal)
Modbus RTU EIA-485
Power 24V AC/DC


Power 24V AC/DC
Ethernet Activity (Green) Link (Yellow)
Modbus RTU EIA-485
Power LED (Blue, internal)
RTU Request, Reply, Error LEDs

(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 (or your PC supports auto-MDX, which many newer laptops do).

(c) Apply power.

A blue 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.

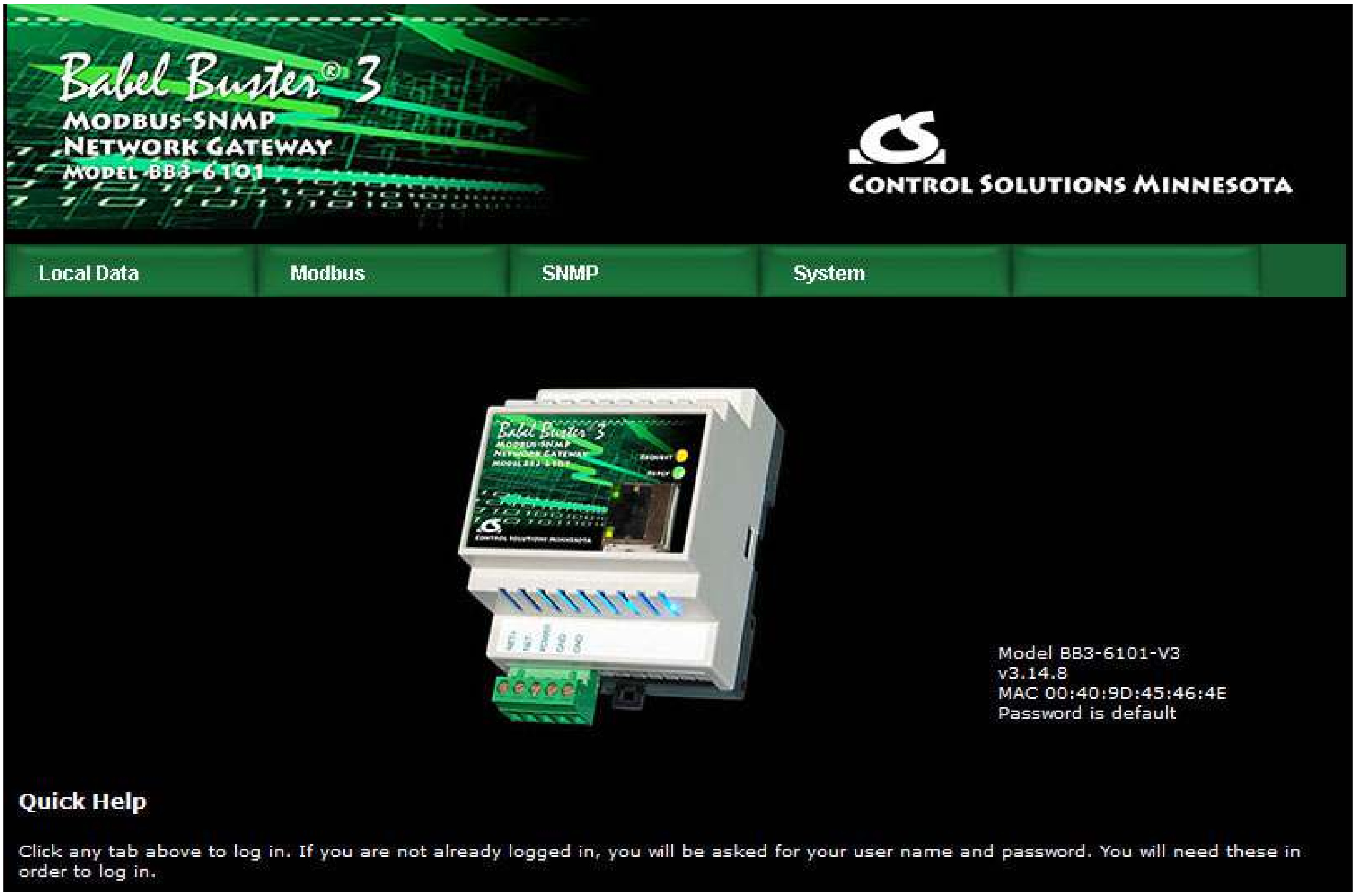
Ethernet link LED is the yellow LED integrated into the CAT5 connector. Ethernet activity LED is the green LED integrated into the CAT5 connector.

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

2.5 Opening the Web User Interface

The default IP address as shipped is 10.0.0.101. Open your browser, and enter "http://10.0.0.101/" in the address window. You should see a page with the "Babel Buster 3" header shown below. From this point, you will find help on each page in the web site contained within the product.

If your PC is not already on the 10.0.0.0 domain, and you are unable to connect, you may need to temporarily change your computer's IP address to a static IP address that starts with 10.0.0. and ends with anything but 101.



Model BB3-6101-V3
v3.14.8
MAC 00:40:9D:45:46:4E
Password is default

Quick Help

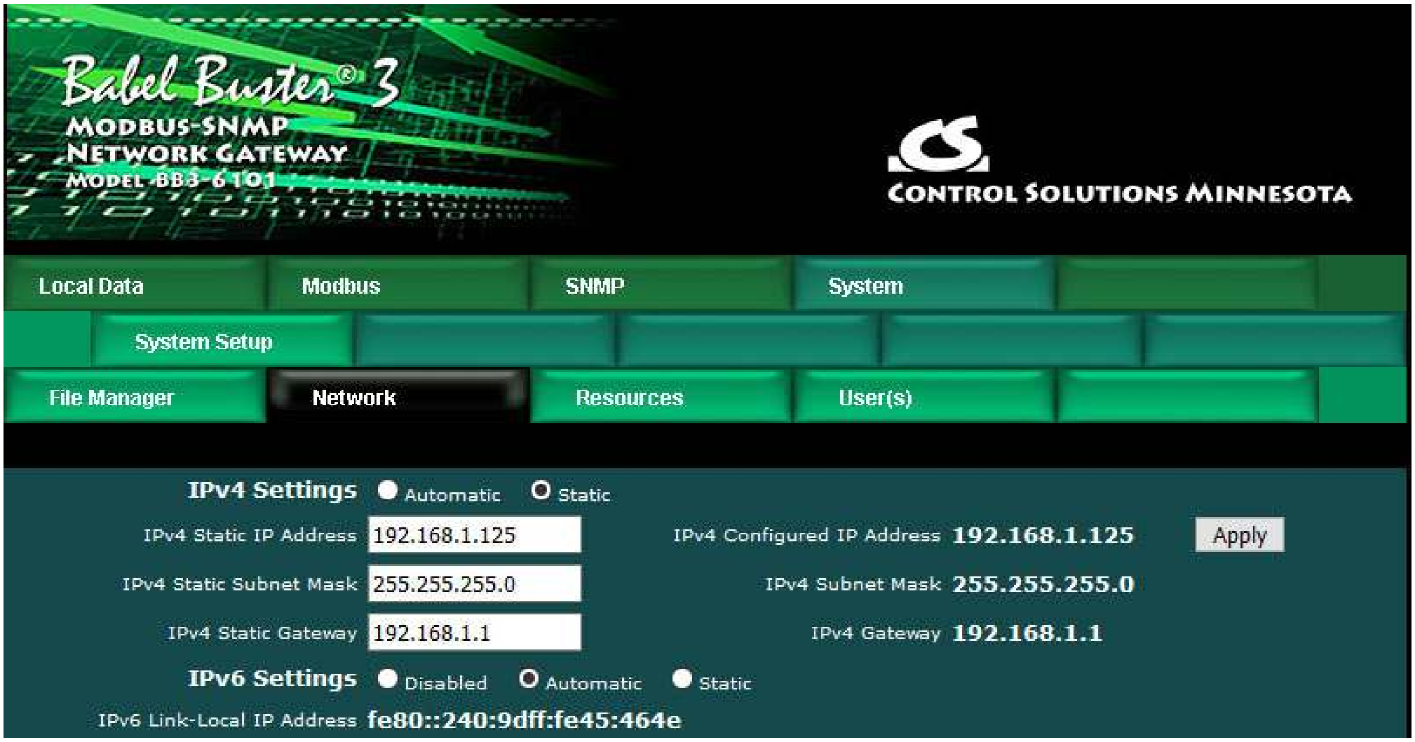
Click any tab above to log in. If you are not already logged in, you will be asked for your user name and password. You will need these in order to log in.

When you click on any of the page tabs such as System, you will be asked for a user name and password. The only login as shipped is user name “root” with a unique password generated specifically for your Babel Buster. Your password should be included on a document included with the gateway, or on a label attached to the gateway.

If the unique automatically generated password is currently in effect for user "root", it will be indicated by "Password is default" as shown in the above screen shot. If you have changed the root password to something of your own making, then this line is absent.

There is no way to get the BB3-6101/MX-61 to show you what the default root password is. If you have lost track of it, make a note of the MAC address, and open a support ticket at https://ticket.csimn.com to request the default root password (you will need to provide the MAC address in order to obtain the password).

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.



IPv4 Settings ○ Automatic ● Static

IPv4 Static IP Address 192.168.1.125 IPv4 Configured IP Address 192.168.1.125

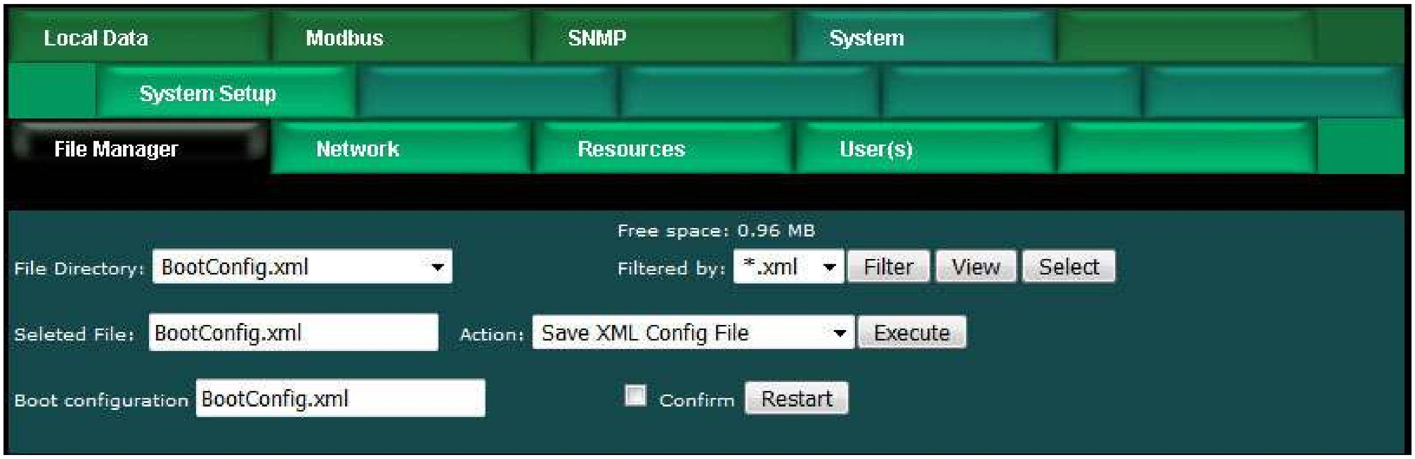
IPv4 Static Subnet Mask 255.255.255.0 IPv4 Subnet Mask 255.255.255.0

IPv4 Static Gateway 192.168.1.1 IPv4 Gateway 192.168.1.1

IPv6 Settings ○ Disabled ● Automatic ○ Static

IPv6 Link-Local IP Address fe80::240:9dff:fe45:464e

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 select your file, select the Save XML Config File action, and then click Execute on the File Manager page. Refer to Section 3 for more about the File Manager.



Free space: 0.96 MB

File Directory: BootConfig.xml

Filtered by: *.xml

Seleted File: BootConfig.xml

Action: Save XML Config File

Boot configuration BootConfig.xml

□ Confirm

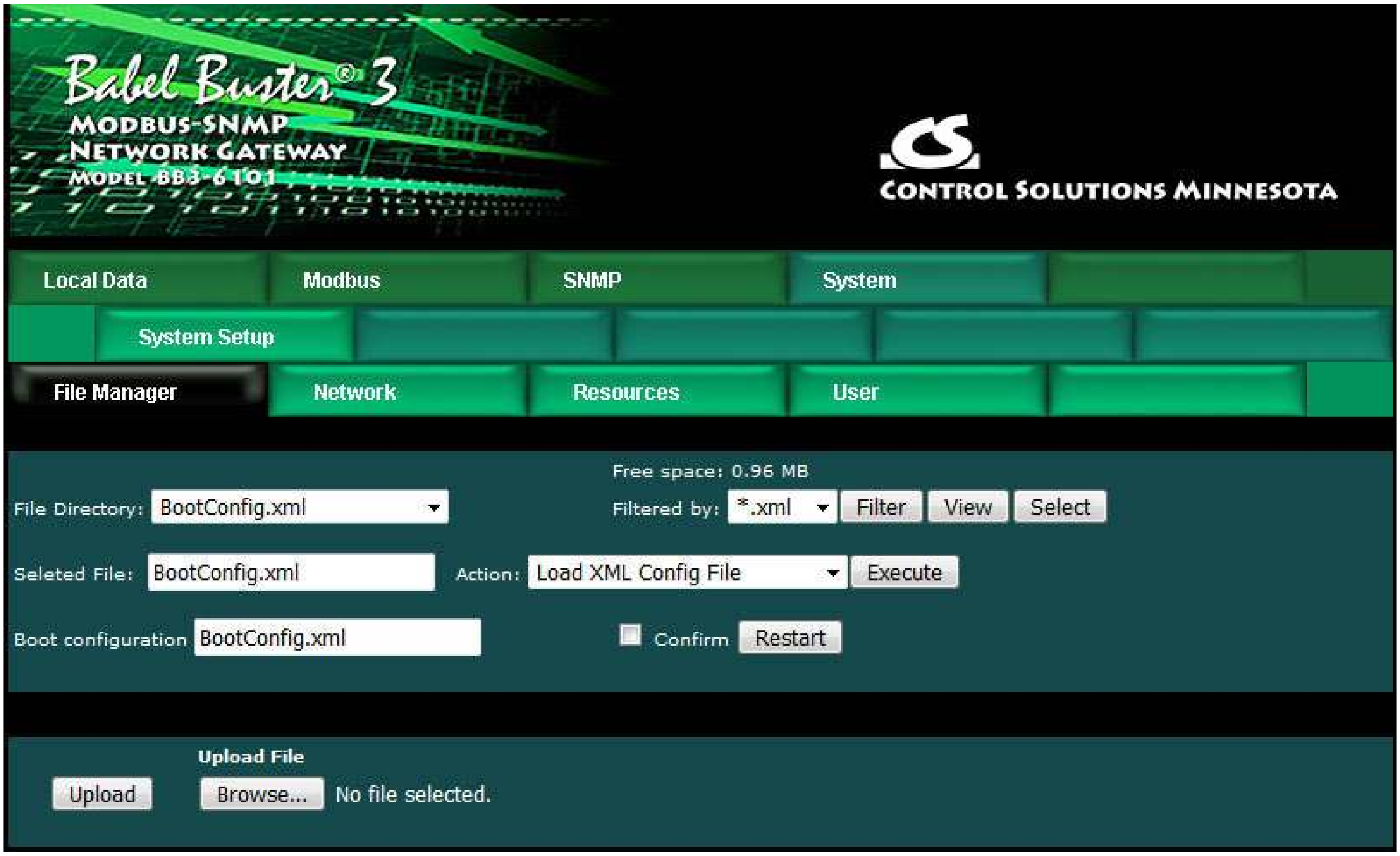
3. System Configuration and Resources

3.1 Using the File Manager

The File Manager page is probably one of the most important pages to know about. Among other things, 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.

The File Manager is used in several other ways in addition to managing your XML configuration files. You upload SSL certificates here. You import CSV files for Modbus client configuration here.



Free space: 0.96 MB

File Directory: BootConfig.xml

Filtered by: *.xml

Seleted File: BootConfig.xml

Action: Load XML Config File

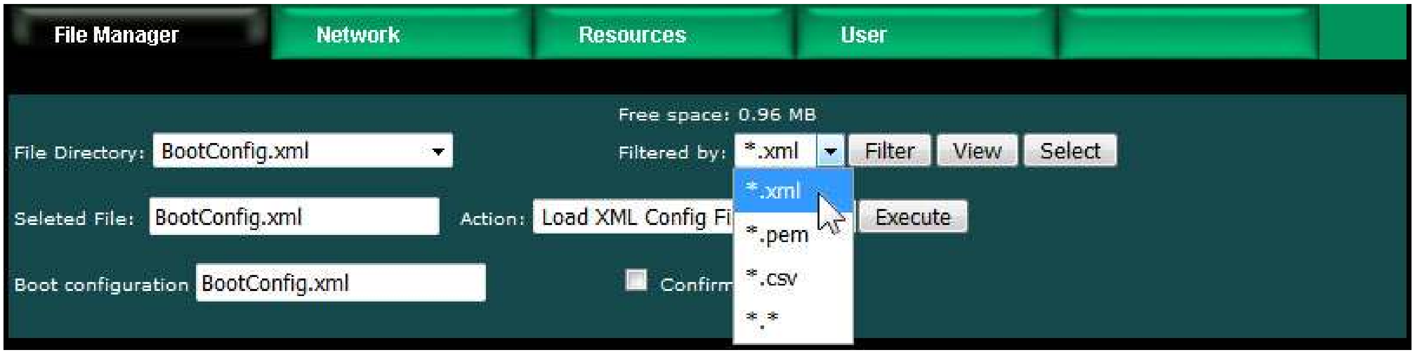
Boot configuration BootConfig.xml

□ Confirm

Upload File

No file selected.

The File Directory is a list of files that are currently stored in the Babel Buster's Flash file system. To filter files by type, select a type from the Filtered by list, and click Filter.



Free space: 0.96 MB

File Directory: BootConfig.xml

Filtered by: *.xml

Seleted File: BootConfig.xml

Action: Load XML Config Fi

Boot configuration BootConfig.xml

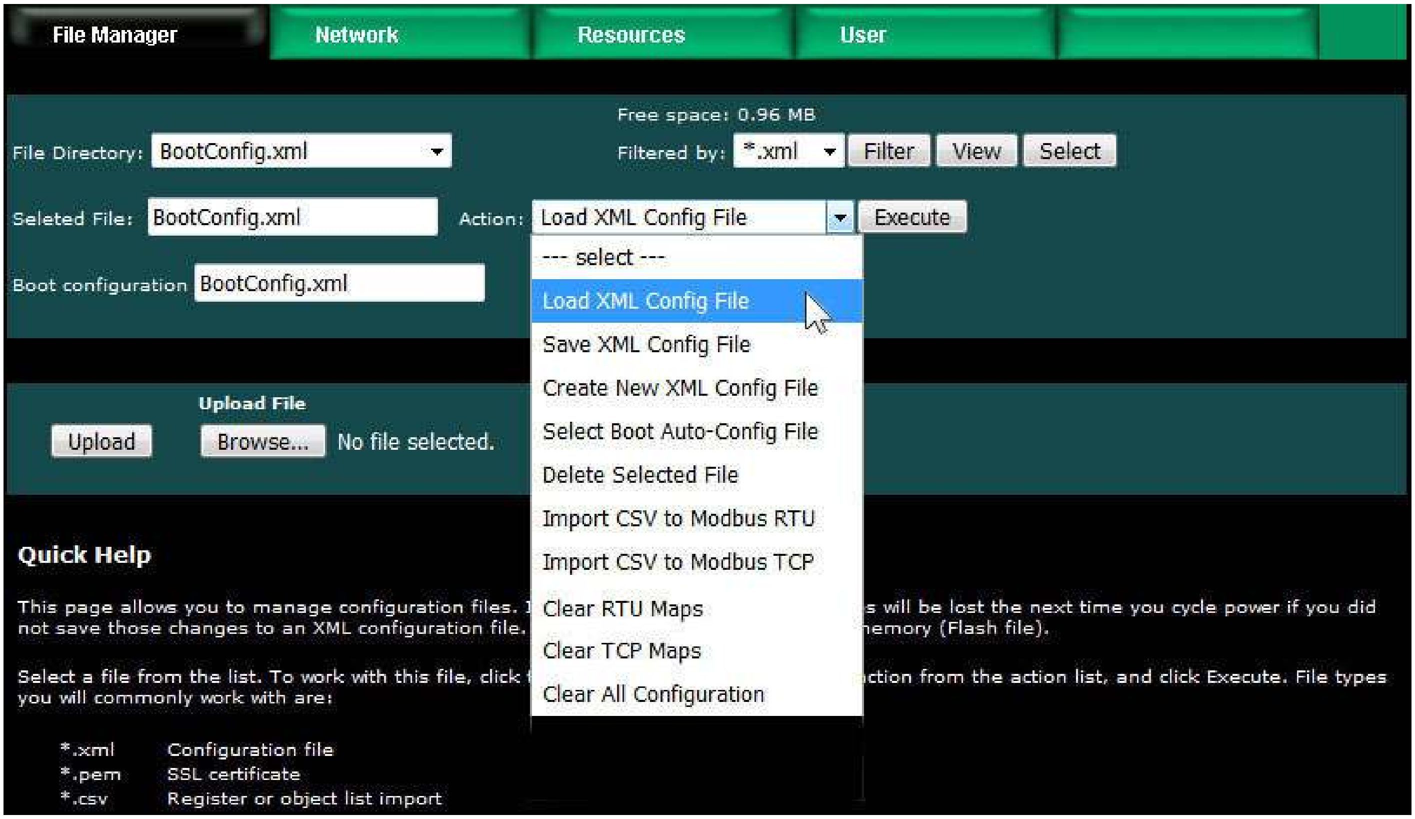
□ Confirm

File type filters are as follows:

*.xml	XML configuration files
*.pem	SSL certificates
*.csv	CSV spreadsheet for Modbus register import
.	Display all files

There are several file related actions you may take. To take action with a certain file, select that file from the File Directory list, and click Select. That file should now show up in the Selected File window.

Once a file has been selected, choose your action from the Action list, and click Execute.



Free space: 0.96 MB

File Directory: BootConfig.xml

Filtered by: *.xml

Seleted File: BootConfig.xml

Action: Load XML Config File

Boot configuration BootConfig.xml

No file selected.

Quick Help

This page allows you to manage configuration files. I s will be lost the next time you cycle power if you did not save those changes to an XML configuration file. nemory (Flash file).

Select a file from the list. To work with this file, click t ction from the action list, and click Execute. File types you will commonly work with are:

*.xml	Configuration file
*.pem	SSL certificate
*.csv	Register or object list import

You must use the Select button to populate the Selected File window prior to executing any action from the list. Choose a file from the drop down list that shows all available files, then click the Select button. You may then act on that file.

You do not need to use the Select button to simply View a file. Clicking View will cause your browser to display the file chosen from the drop down list. If you attempt to View a CSV file, your PC will likely ask if you want to download the file or open it with your spread sheet program (e.g. Excel).

Upload File: To upload a file from your PC to this 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.

NOTE: If you get a "File upload error: -1" message, click the browser's "back" button, then simply click the View button to view any file (does not matter which file), and then click browser's "back" button again to return to this page. This gets the browser and HTTP server back in sync, and this requirement generally happens only once following power-up.

Restart: To restart the gateway, check Confirm and click Restart. This is a hard reset that will accomplish the same thing as a power cycle without physically disconnecting and reconnecting power.

3.1.1 Load, Save, Create XML Configuration File

IMPORTANT: Configuration files from older gateways are not directly usable in the BB3-6101/MX-61, but the Babel Buster Configuration Builder program can be used to convert a BB2-6010/SPX configuration into CSV files that may be imported into the BB3-6101/MX-61.

Load XML Config File: The configuration file shown in the "Boot configuration" window will be loaded automatically at startup. If you have uploaded a new configuration file and wish to use it without restarting, select that file (choose from list, click Select), select this action, and click Execute.

HINT: If you are loading a file generated externally and you get "parameter out of range" errors pertaining to defining registers or "table full" errors while loading maps or rules, you might not have sufficient resources allocated. You may need to increase some counts on the Resources page.

Save XML Config File: Any time you have made configuration changes that you want to retain as permanent, you need to come here, select the file from the directory list, and execute this Save action.

Create New XML Config File: You have the option to create a totally new configuration file. This is often suitable if you started with an existing configuration, made changes, and want to save your changes without replacing the original configuration. To create a new file, rather than selecting a file from the directory list, simply type a new name into the Selected file window. The name cannot contain spaces or special characters, and be sure to use the correct file suffix. Enter the name and execute this action.

3.1.2 Select Startup Configuration

Select Boot Auto-Config File: This is where you tell the Babel Buster what configuration to automatically load upon startup. To set the Boot configuration, select the XML file from the list, and execute this action. 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.

When selecting a new Boot configuration file, it is a good idea to select the file, and execute Load XML Config File. If there are errors, they will be displayed. If there are errors in the file but you do not fix them, then the gateway will not fully start up the next time it restarts. The web user interface will be available, but it will not be talking to Modbus devices.

3.1.3 Delete a File

Remove a file from the Flash file system by selecting it from the list and executing the Delete Selected File action.

3.1.4 Import CSV File

Import CSV to Modbus RTU: You can configure Modbus RTU read and write maps in bulk by importing the maps as a CSV file that you created using a standard spreadsheet program. Refer to Appendix B for details about the CSV format. Note that maps will be added to the existing map list. If you want to replace existing maps with imported maps, execute Clear RTU Maps first.

Import CSV to Modbus TCP: You can configure Modbus TCP read and write maps in bulk by importing the maps as a CSV file that you created using a standard spreadsheet program. Refer to Appendix B for details about the CSV format. Note that maps will be added to the existing map list. If you want to replace existing maps with imported maps, execute Clear TCP Maps first.

HINT: If you get "table full" errors while importing CSV files, you might not have sufficient resources allocated. You may need to increase some counts on the Resources page.

3.1.5 Clear Configuration

Clear RTU Maps: Execute this action to clear (completely remove) all Modbus RTU read and write maps.

Clear TCP Maps: Execute this action to clear (completely remove) all Modbus TCP read and write maps. The Modbus TCP device table will be left intact.

Clear All Configuration: Execute this action to completely wipe out all configuration. This includes all Modbus maps and devices, all SNMP rules and devices, and all local registers. This will put you back to a "reset to factory" condition with the exception that your IP address is left unchanged. (See Appendix A, Section A.6, regarding forced hard configuration reset that includes IP address and root password.) If you want to make the now empty configuration permanent, select the file that is also selected as Boot configuration, and execute the Save XML Config File action.

The other means of completely wiping out all saved configuration is to simple delete the file named as the Boot configuration file, and then restart or power cycle the Babel Buster. Upon restart, a new empty configuration file will be created automatically.

3.2 Configuration Files and Restoring Default Settings

There is a means of restoring the Babel Buster 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.

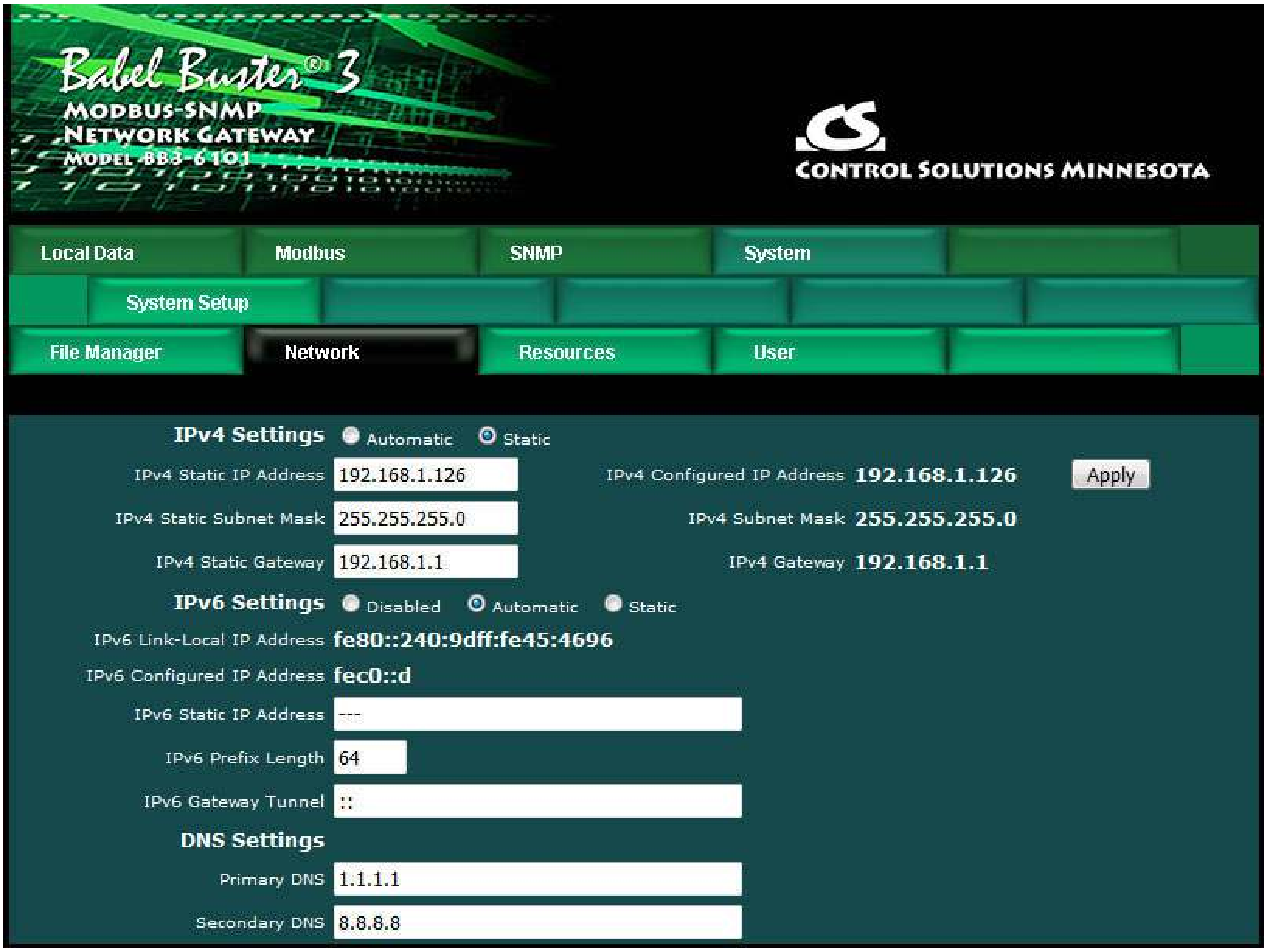
Manual Editing: It is possible to manually edit the XML file outside of the gateway. However, doing so is very prone to errors. If there are errors in the XML file, it will not load successfully on startup. If the configuration does not load on startup, none of the scanners will begin scanning. Because 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, select Load XML Config File, and click Execute. 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.

Backup Copy of XML Config File: 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). Saving the configuration file to your PC, and then uploading on a different device, is a quick and easy way to configure two Babel Busters the same way.

Note about caching: 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 "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. 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.

3.3 Network Configuration

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



IPv4 Settings ○ Automatic ● Static

IPv4 Static IP Address 192.168.1.126 — IPv4 Configured IP Address **192.168.1.126**

IPv4 Static Subnet Mask 255.255.255.0 — IPv4 Subnet Mask **255.255.255.0**

IPv4 Static Gateway 192.168.1.1 — IPv4 Gateway **192.168.1.1**

IPv6 Settings ○ Disabled ● Automatic ○ Static

IPv6 Link-Local IP Address **fe80::240:9dff:fe45:4696**

IPv6 Configured IP Address **fec0::d**

IPv6 Static IP Address ---

IPv6 Prefix Length 64

IPv6 Gateway Tunnel ::

DNS Settings

Primary DNS 1.1.1.1

Secondary DNS 8.8.8.8

3.3.1 IPv4, IPv6 Settings

To change the IP address(es) of this device, make the applicable entries and click Apply. The "automatic" selection means DHCP. Changes to the IPv4 IP address will take effect upon the next system restart.

If IPv6 is enabled, IPv6 will always have a Link-Local address, plus one configured address. The configured address will be either the static IP address, or an IPv6 address obtained from an IPv6 DHCP server. If no configured address appears, the DHCP server may have been unreachable.

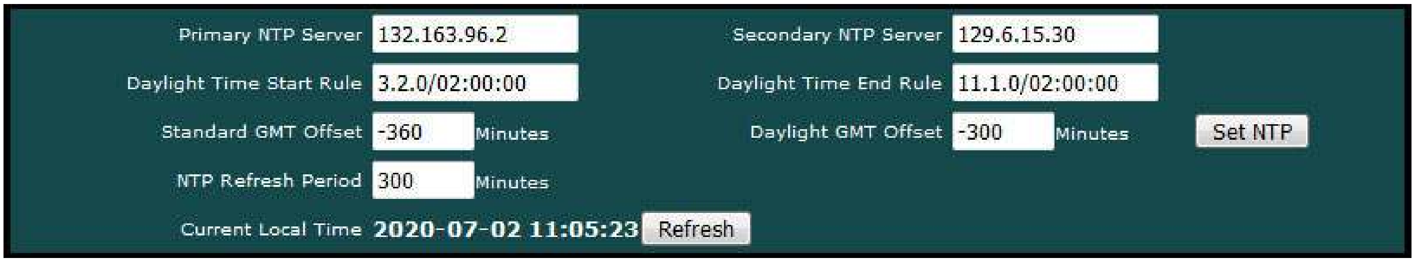
The IPv6 static IP address window is the configured static address. If "Static" is selected and a new IP address entered as the static address, this new address will not take effect until the next system restart.

The numbers shown to the right of the IPv4 input windows are the actual numbers currently in use. If static IP addresses have been entered but the gateway has not been restarted yet, these numbers will not be the same.

You may use domain names instead of static IP addresses in several instances. If domain names are used, you must supply the IP address of at least one DNS server here. The DNS server must be at a static IP address. These changes take effect immediately. Note: If you are using DHCP, the DNS addresses will be supplied by the DHCP server and should be set to 0.0.0.0 here.

3.3.2 NTP Time Server Settings

The Babel Buster maintains time and date via SNTP services.



Primary NTP Server	132.163.96.2	Secondary NTP Server	129.6.15.30
Daylight Time Start Rule	3.2.0/02:00:00	Daylight Time End Rule	11.1.0/02:00:00
Standard GMT Offset	-360 Minutes	Daylight GMT Offset	-300 Minutes
NTP Refresh Period	300 Minutes		
Current Local Time	2020-07-02 11:05:23		

NTP setup: Enter a primary and secondary IP address of NTP servers, such as those found at wwv.nist.gov (go to http://tf.nist.gov/tf-cgi/servers.cgi to find more). Enter daylight start/end rules, and offset from GMT for both standard and daylight time. Offset is a negative number in the western hemisphere. Enter an NTP update time in minutes. Do not set NTP to update too frequently or you risk being denied service by the NTP server. Click the Set NTP button after all settings have been made. The Flash update will take several seconds. The initial update of local time may take a minute or two.

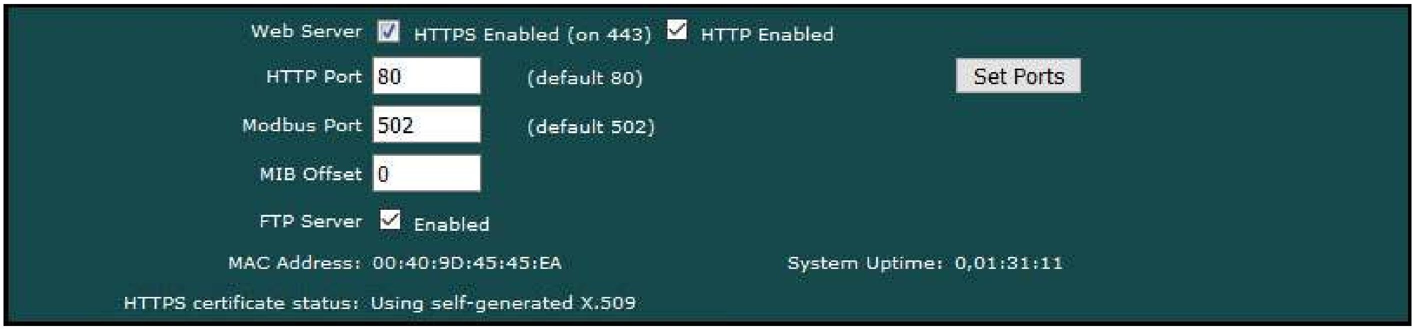
Daylight savings time start/end rules consist of "date/time" where the date (m.n.d) indicates the day when summer time starts or ends, and time (hour:min:sec) is the current local time when summer time starts/ends. The date portion of the rule is formatted as follows:

- m indicates the month (1 <= m <= 12)
- n indicates which week of the month (1 <= n <= 5). 5 = the last week in the month.
- d indicates what day of the week (0 <= d <= 6). 0 = Sunday

For example: Start "4.1.0/02:00:00", end "10.5.0/02:00:00" means summer time starts at 2am on the first Sunday in April and ends at 2am on last Sunday in October. That was the old US rule. The new US rule is start "3.2.0/02:00:00" and end "11.1.0/02:00:00", which is start at 2am on the second Sunday in March, end at 2am on the first Sunday in November.

Note about time maintained here: Modbus gateway functionality has no use for time and date. The only time you have a need for valid time and date is when using secure connections. If you are using a secure web connection and having trouble connecting, be sure NTP is set up here. Also, if you are using SNMPv3 and are getting errors related to "timeliness", check NTP here.

3.3.3 Port Settings



Web Server ☑ HTTPS Enabled (on 443) ☑ HTTP Enabled
HTTP Port 80 (default 80)
Modbus Port 502 (default 502)
MIB Offset 0
FTP Server ☑ Enabled
MAC Address: 00:40:9D:45:45:EA System Uptime: 0,01:31:11
HTTPS certificate status: Using self-generated X.509

Secure browsing can be enabled here, and non-secure can be disabled. You cannot disable both, and a forced configuration reset will restore HTTP (non-secure) web browsing. In order to use HTTPS, you must first upload the necessary SSL certificates (see Appendix E) or allow the certificates to be self-generated by explicitly deleting existing certificates.

IMPORTANT: It is highly recommended that in making the transition from HTTP to HTTPS, you enable both until you confirm HTTPS is functional. If there is a problem with the SSL certificates provided for HTTPS, then HTTPS will not run and you will find an error message on the "HTTPS certificate status" line. If you disable standard HTTP without first verifying that HTTPS is functional, you may end up locked out and will then need to do a forced hard reset (Appendix A.6).

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 away 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.

The MIB offset lets you effectively move the entire MIB. This is required if more than one Babel Buster 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 Reload SNMP to cause it to take effect.

FTP is enabled by default to allow firmware update uploads. It may be optionally disabled here. Just remember to enable it again before attempting a firmware update.

Any changes to this port numbers or enabling/disabling features requires restarting the Babel Buster before they will take effect.

3.4 Resource Allocation

Historically, Control Solutions gateways had a fixed set of resources to work with.

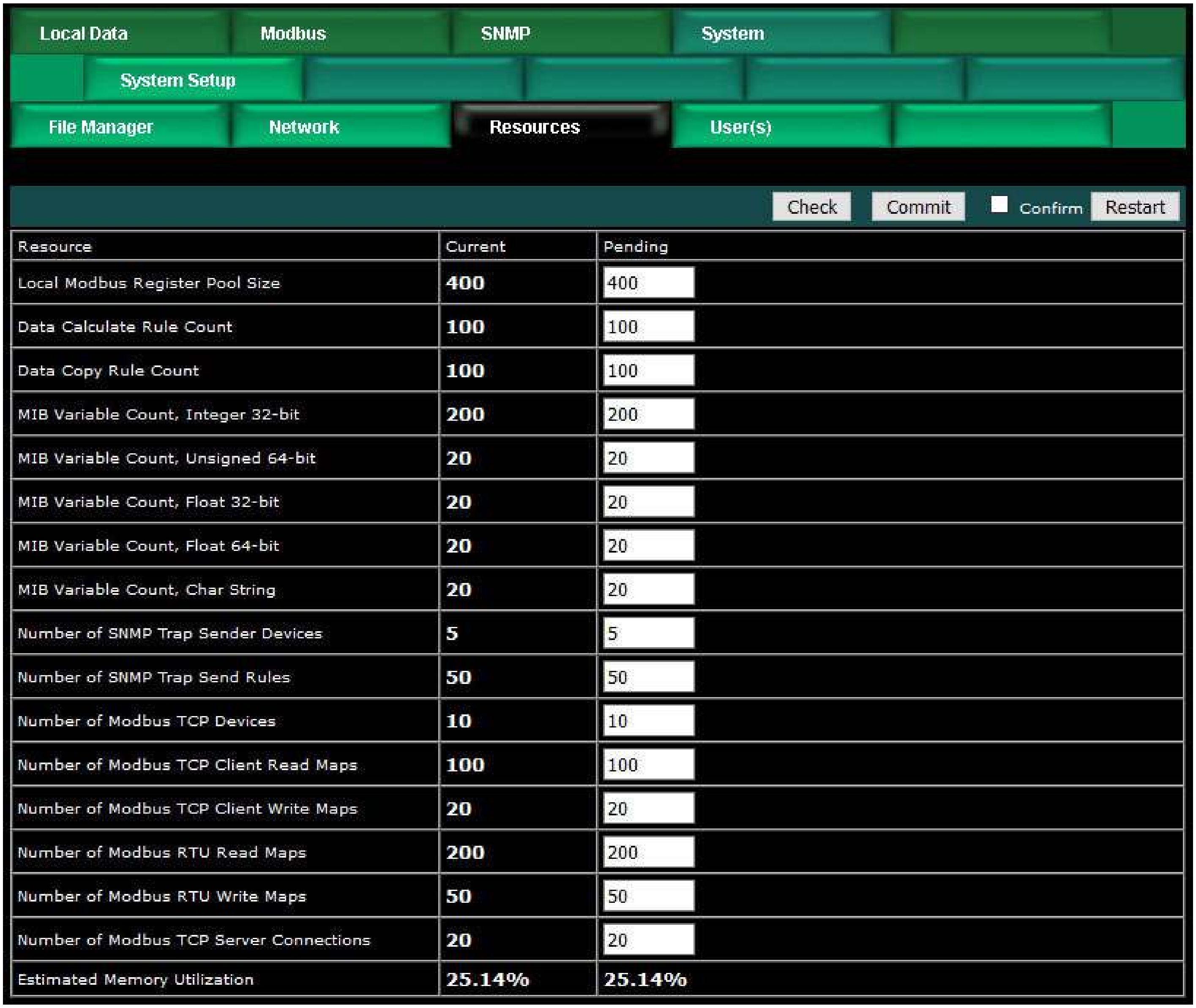
Invariably, there were always users that wanted less of this and more of that. Therefore, while there are still maximums imposed, you can now shift resources around as best suits your application. An example is shown below.

The values in the Pending column are those found in the most recently loaded XML configuration file. When saving or creating a new XML file, the numbers in the Current column will be written to the file. To change the allocations, change numbers in the Pending column. When you are ready to commit these changes, click the Commit button. To cause the changes to go into use, you must restart the device since memory allocation can occur only once at startup.

You can click the Check button prior to Commit to see if the values you have entered will be accepted. If adjustments need to be made, the values in the Pending column will be updated.

The first time you visit this page, you will see the initial default values. Should you change any of them, minimums and maximums currently defined in firmware will be imposed. If you see a value smaller than what you entered, it may be that you had exceeded the internal limit.

If you see that numbers toward the top of the list are large, and numbers near the bottom are all set to 1, it means the system has run out of free memory and you need to reallocate resources.



Resource	Current	Pending
Local Modbus Register Pool Size	400	400
Data Calculate Rule Count	100	100
Data Copy Rule Count	100	100
MIB Variable Count, Integer 32-bit	200	200
MIB Variable Count, Unsigned 64-bit	20	20
MIB Variable Count, Float 32-bit	20	20
MIB Variable Count, Float 64-bit	20	20
MIB Variable Count, Char String	20	20
Number of SNMP Trap Sender Devices	5	5
Number of SNMP Trap Send Rules	50	50
Number of Modbus TCP Devices	10	10
Number of Modbus TCP Client Read Maps	100	100
Number of Modbus TCP Client Write Maps	20	20
Number of Modbus RTU Read Maps	200	200
Number of Modbus RTU Write Maps	50	50
Number of Modbus TCP Server Connections	20	20
Estimated Memory Utilization	25.14%	25.14%

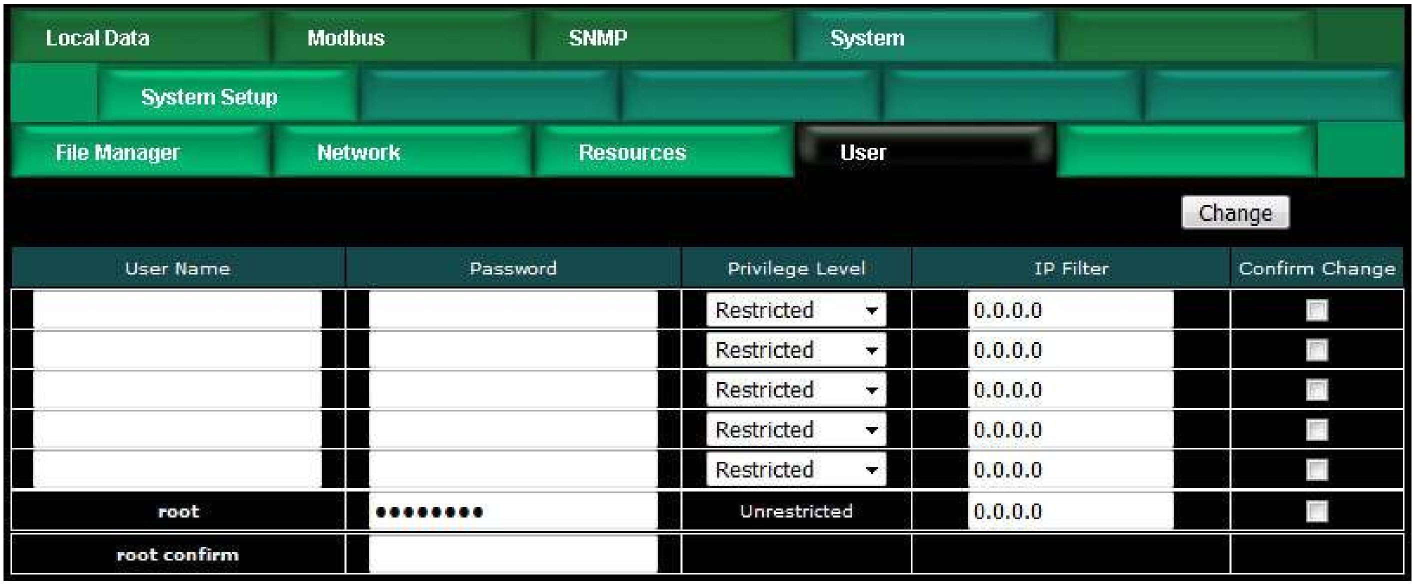
The estimated memory utilization shown at the bottom gives you an indication of how close you are to running out of memory. You will not be allowed to commit a resource allocation greater than 100%.

3.5 User Login Passwords

There is only one default login provided initially, namely the username "root" with a unique password generated specifically for your particular Babel Buster. This password is provided to you in either external documentation included with the gateway, or it may be found on a label attached to the gateway. Network security laws in some jurisdictions require that Internet connected (or connectable) devices be shipped with unique default passwords, and the BB3-6101/MX-61 complies with this requirement.

Additional user logins may be created. 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 objects via the Local Objects page, but cannot access any other pages. The Restricted level has no meaning in the BB3-6101/MX-61 (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's web pages from that IP address. Leave set to 0.0.0.0 to disable filtering.



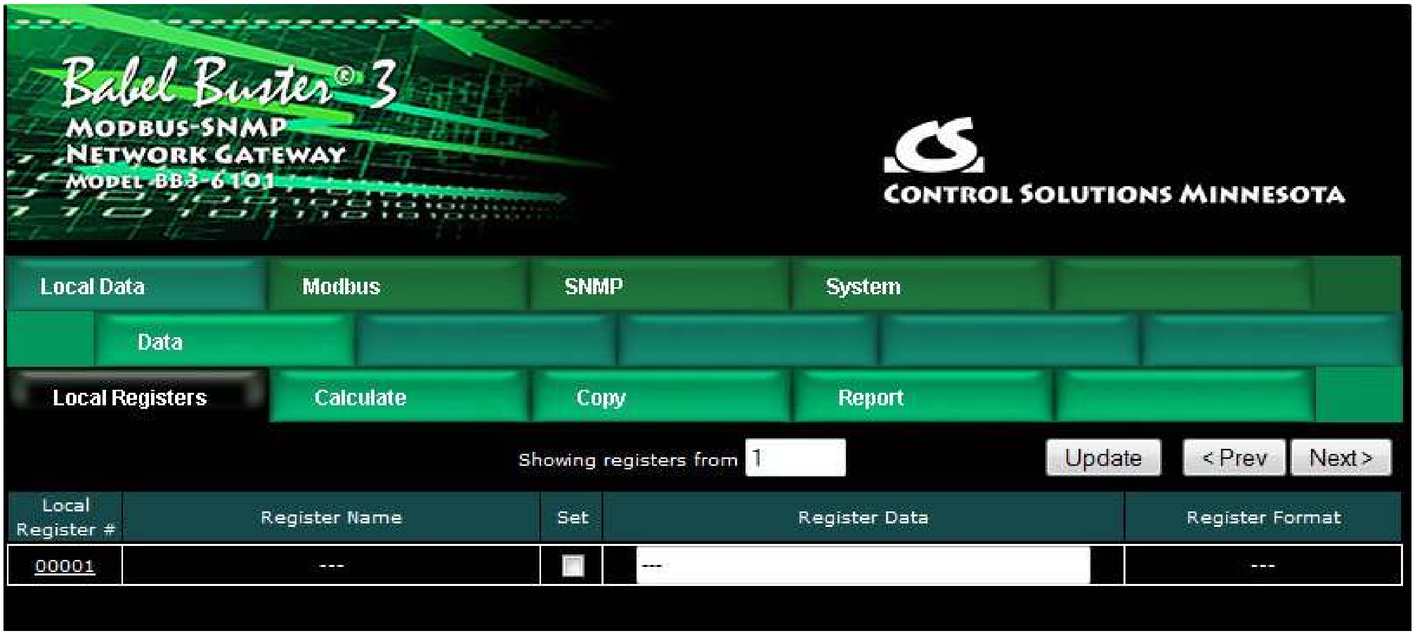
User Name	Password	Privilege Level	IP Filter	Confirm Change
		Restricted	0.0.0.0	□
		Restricted	0.0.0.0	□
		Restricted	0.0.0.0	□
		Restricted	0.0.0.0	□
		Restricted	0.0.0.0	□
root	●●●●●●●●●	Unrestricted	0.0.0.0	□
root confirm				

4. Configuring Local Registers

4.1 Creating Local Registers

Babel Buster gateways originally came with a predefined set of local registers that were accessible externally as Modbus registers. Newer models take a completely different approach to defining registers. When you first take the Babel Buster BB3-6101/MX-61 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.

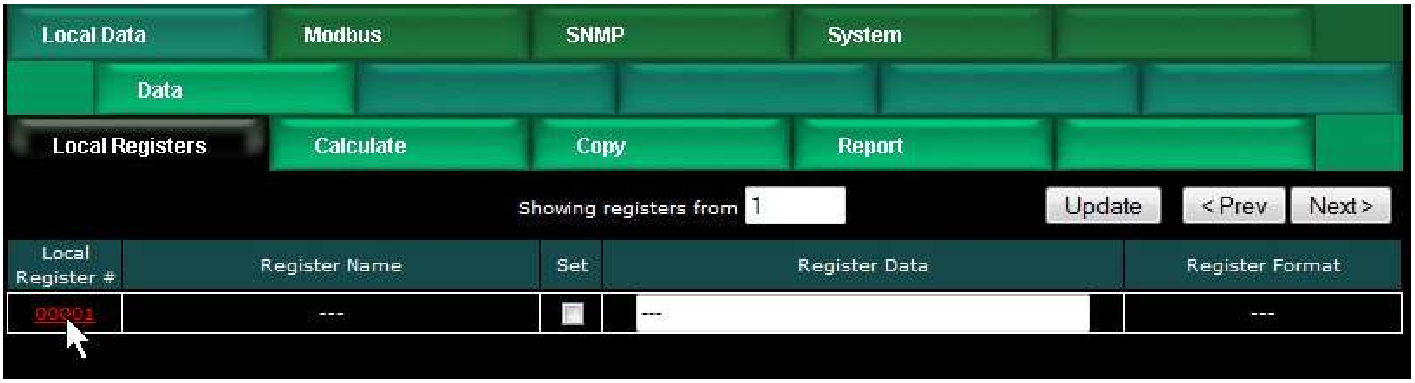
Your first visit to the Local Registers page will be the uneventful display illustrated below.



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	---	□	---	---

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.

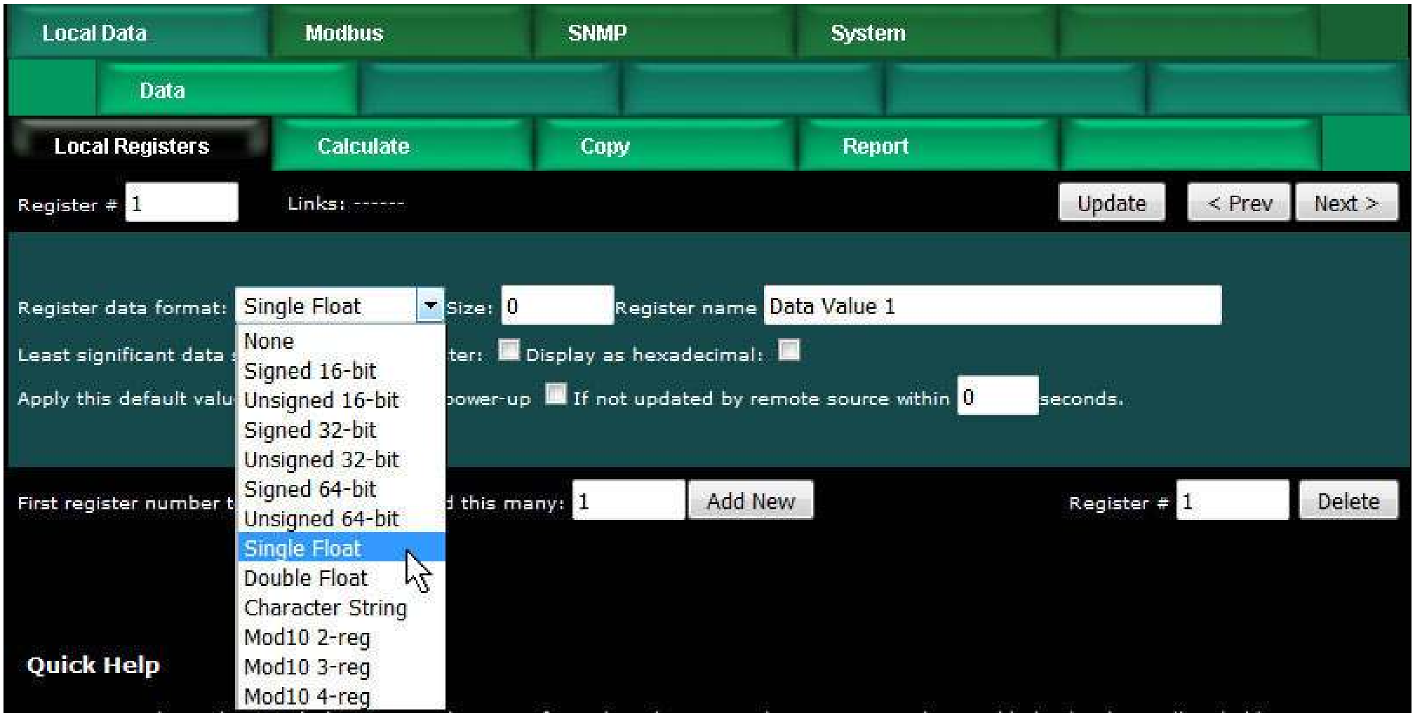


Local Register #	Register Name	Set	Register Data	Register Format
00001	---	□	---	---

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, there are restrictions on changing data format because this impacts how many Modbus registers are actually used. You can change unsigned 16-bit to signed 16-bit, for example, but cannot change 16-bit to 32-bit. 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 various rules and maps 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 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.



Register # 1

Links: ------

Register data format: Single Float

Size: 0

Register name Data Value 1

Display as hexadecimal: □

If not updated by remote source within 0 seconds.

this many: 1

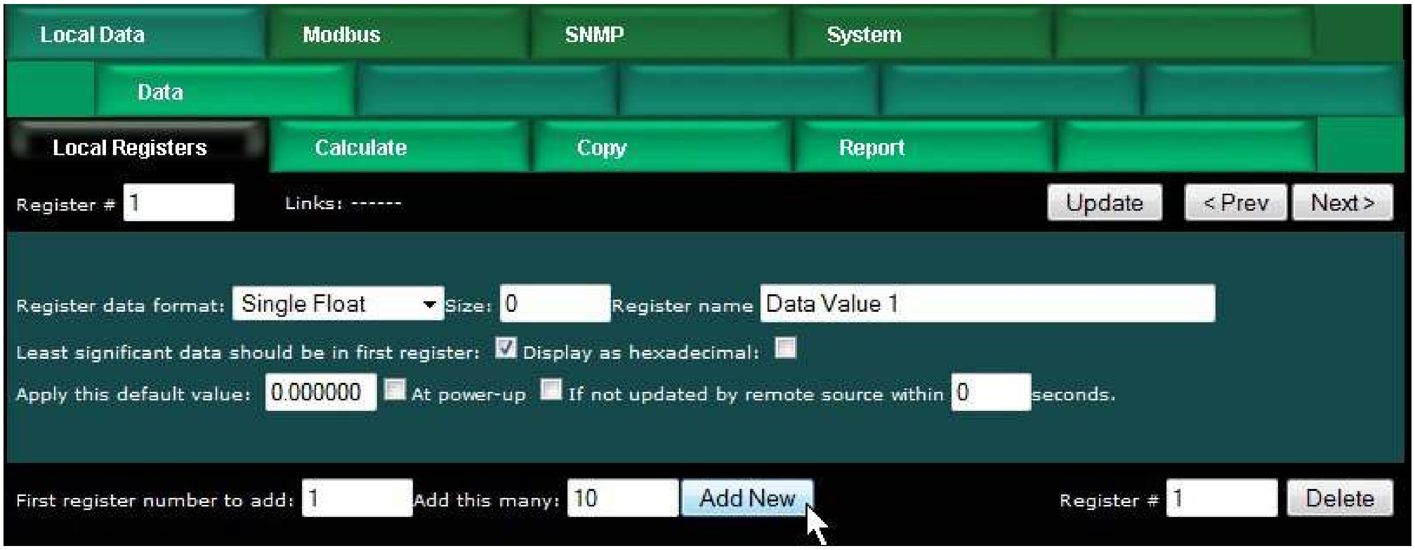
Register # 1

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 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.



Register # 1 Links: ------

Register data format: Single Float Size: 0 Register name Data Value 1

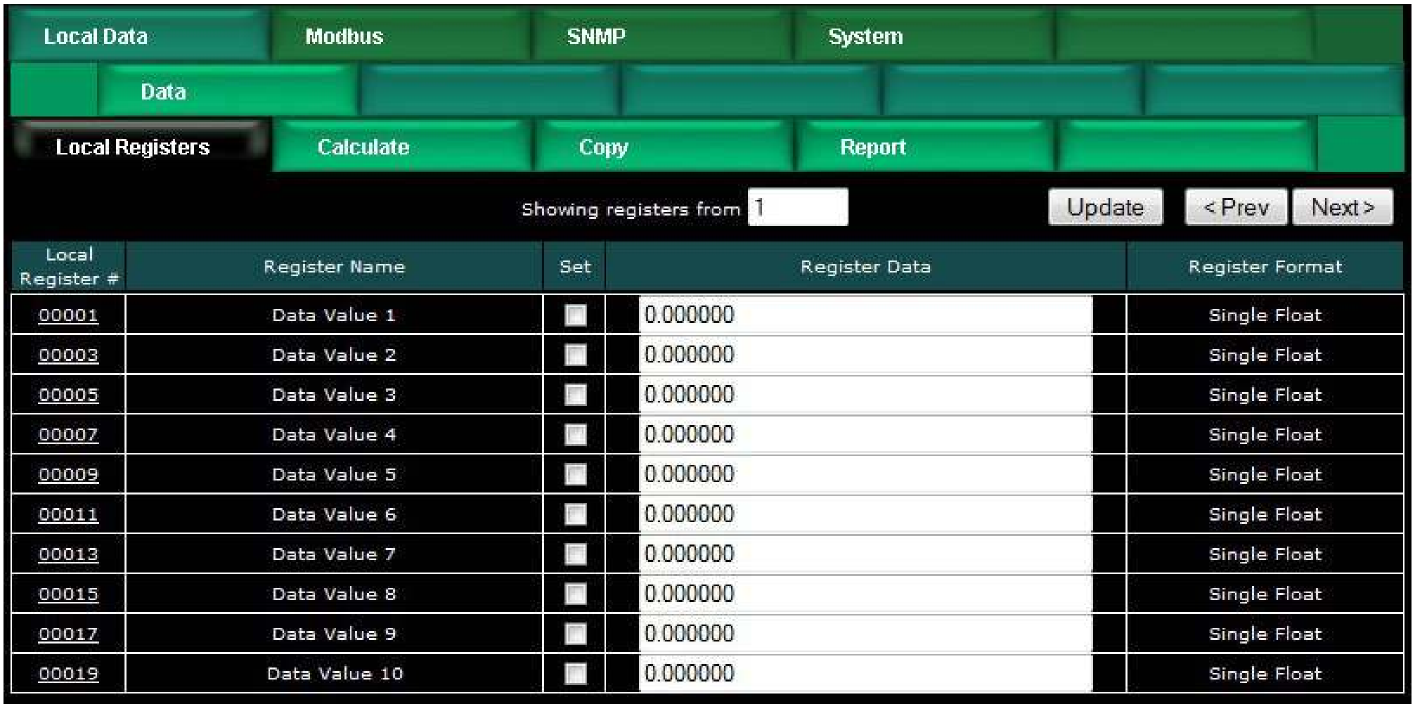
Least significant data should be in first register: ☑ Display as hexadecimal: □

Apply this default value: 0.000000 □ At power-up □ If not updated by remote source within 0 seconds.

First register number to add: 1 Add this many: 10 Add New Register # 1 Delete

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.

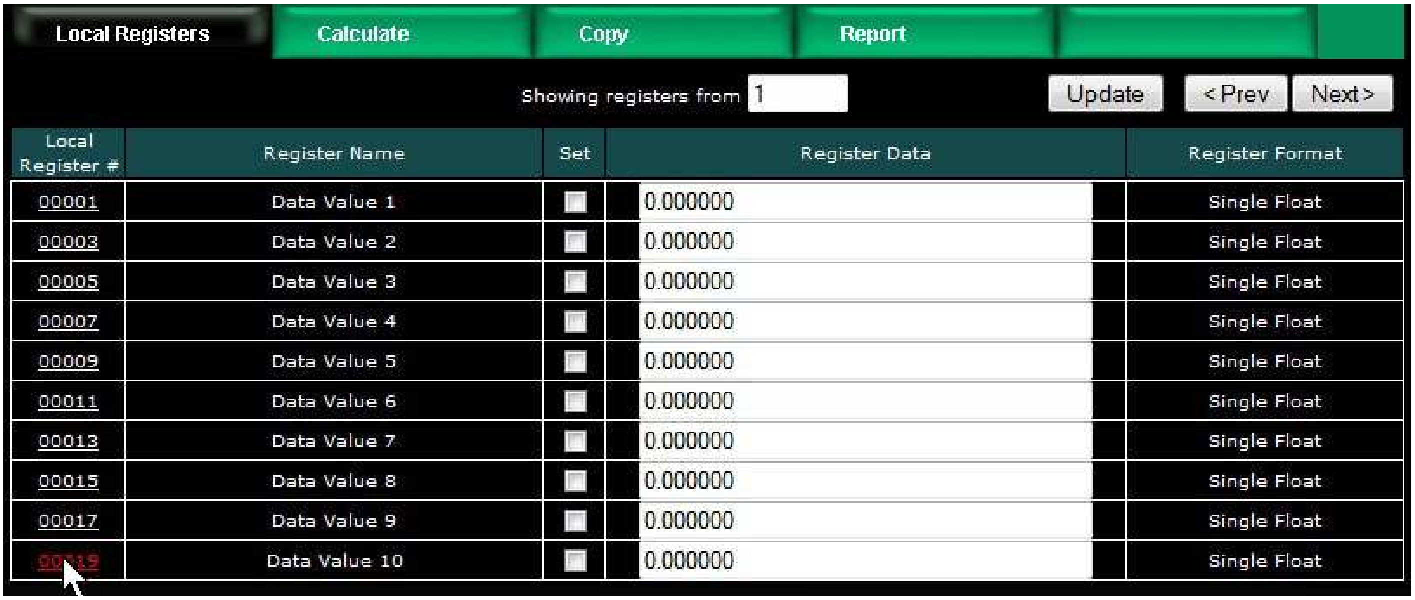
Note that the name given in the screen above was "Data Value 1" but our series of 10 new registers came up with 10 sequential names. If the last thing to appear in the name given when assigning new registers is a number, this value will be incremented by one in the name of each successive new register.



Showing registers from 1

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

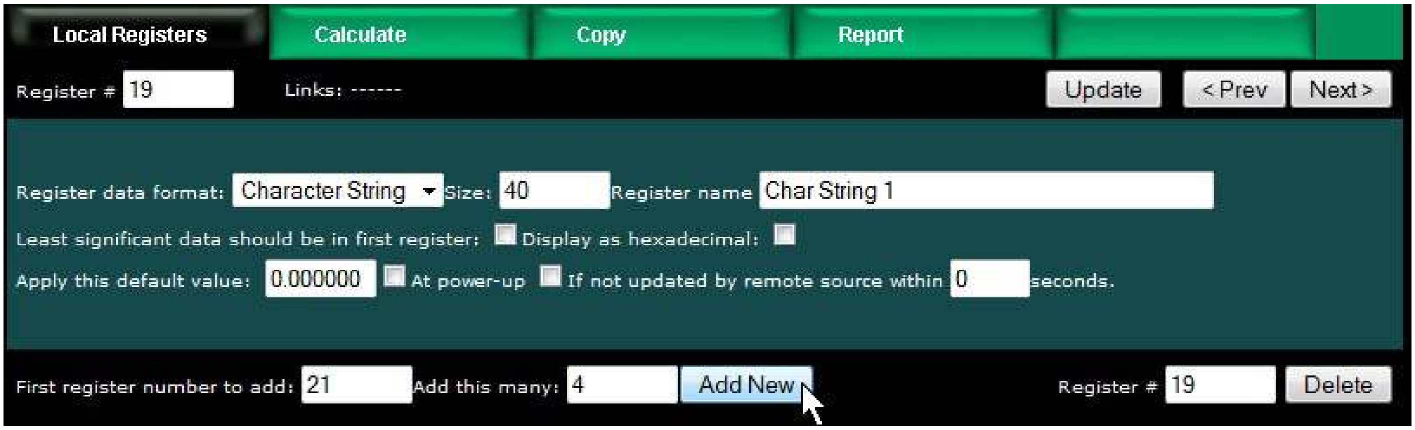
Now let's proceed to add some character strings. Click on the last register number assigned thus far to open the register detail page.



Showing registers from 1

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

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.



Register # 19 Links: ------

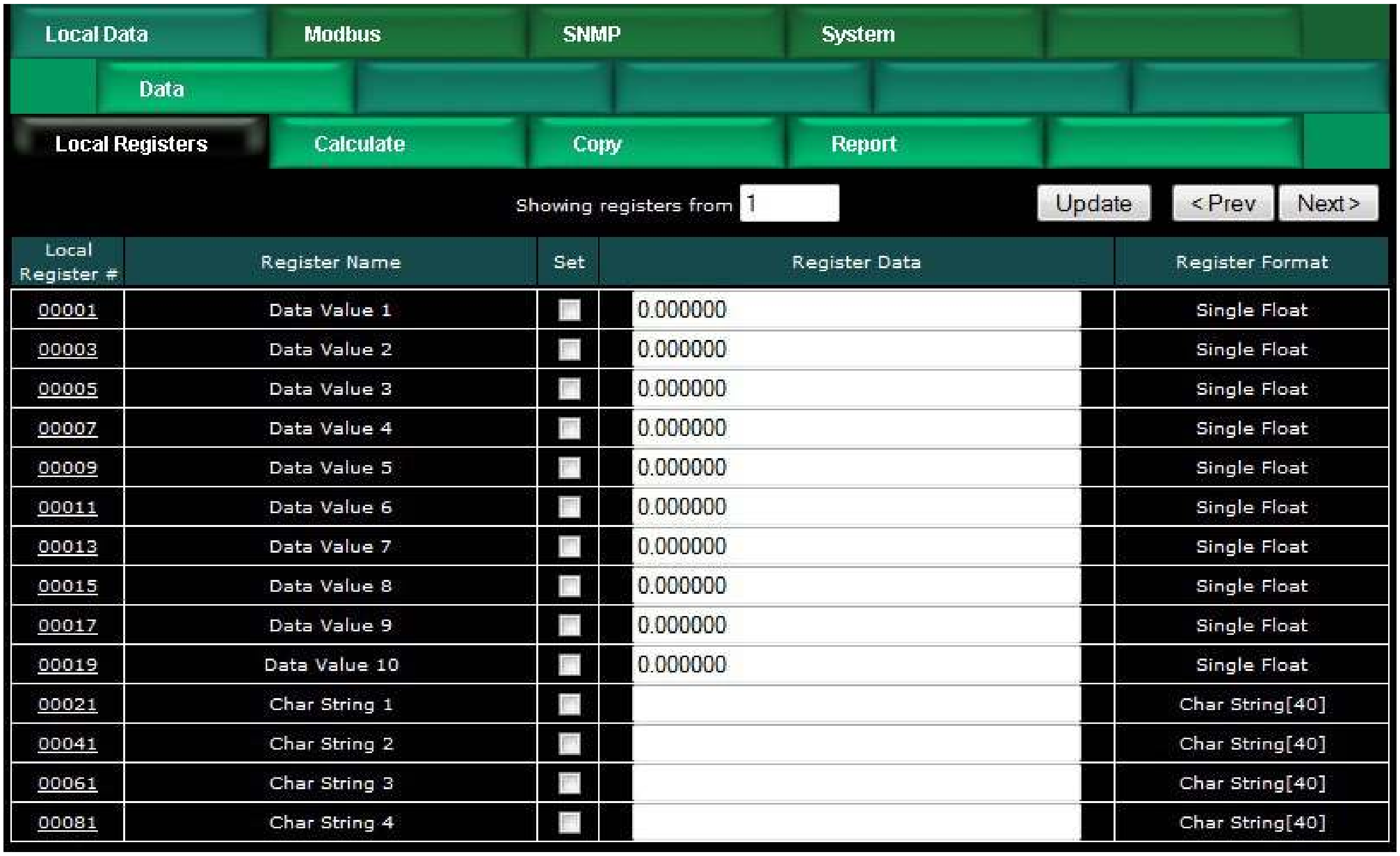
Register data format: Character String Size: 40 Register name Char String 1

Least significant data should be in first register: □ Display as hexadecimal: □

Apply this default value: 0.000000 □ At power-up □ If not updated by remote source within 0 seconds.

First register number to add: 21 Add this many: 4 Register # 19

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.

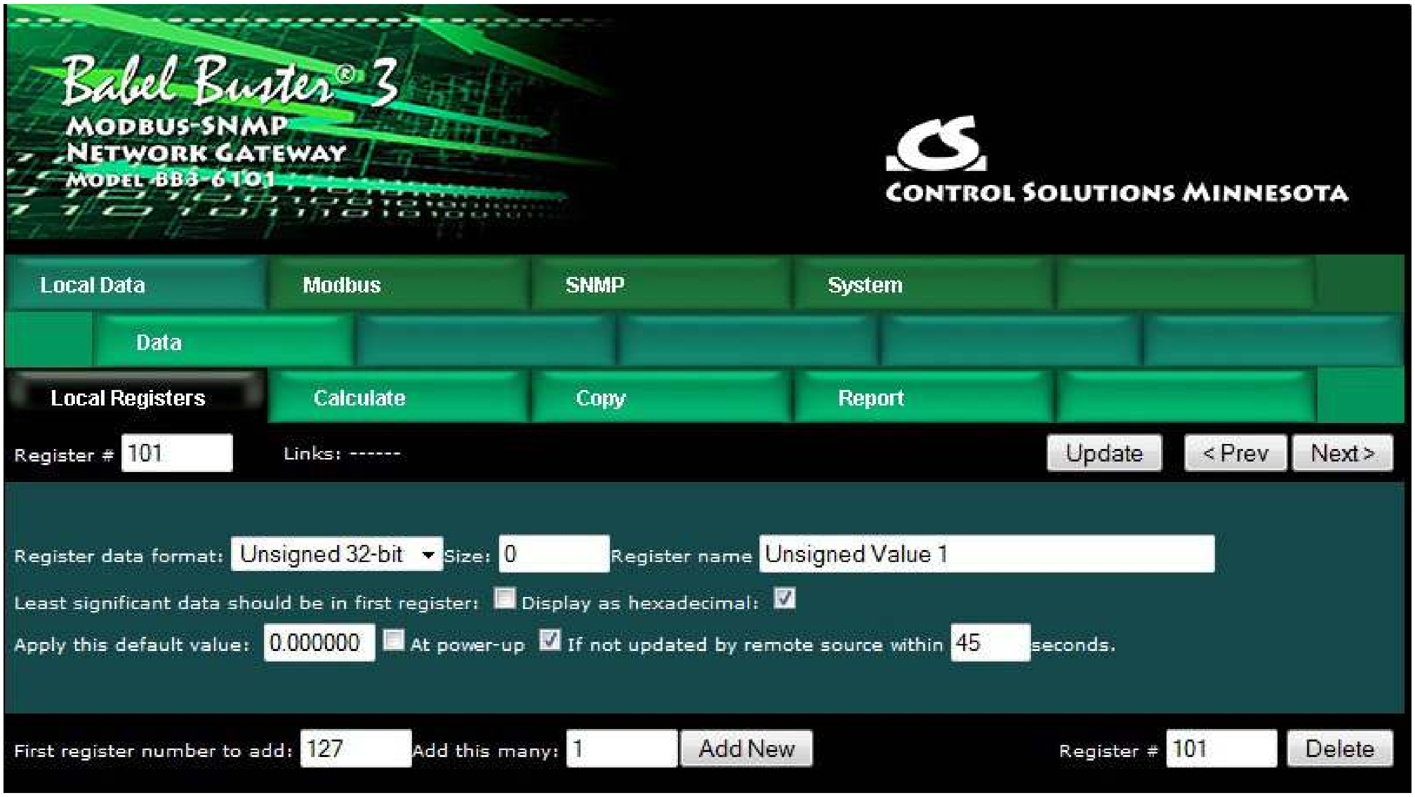


Showing registers from 1

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]

4.2 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.



Register # 101 Links: ------

Register data format: Unsigned 32-bit Size: 0 Register name Unsigned Value 1

Least significant data should be in first register: □ Display as hexadecimal: ☑

Apply this default value: 0.000000 □ At power-up ☑ If not updated by remote source within 45 seconds.

First register number to add: 127 Add this many: 1 Register # 101

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 Babel Buster gateway is acting as a slave or server).

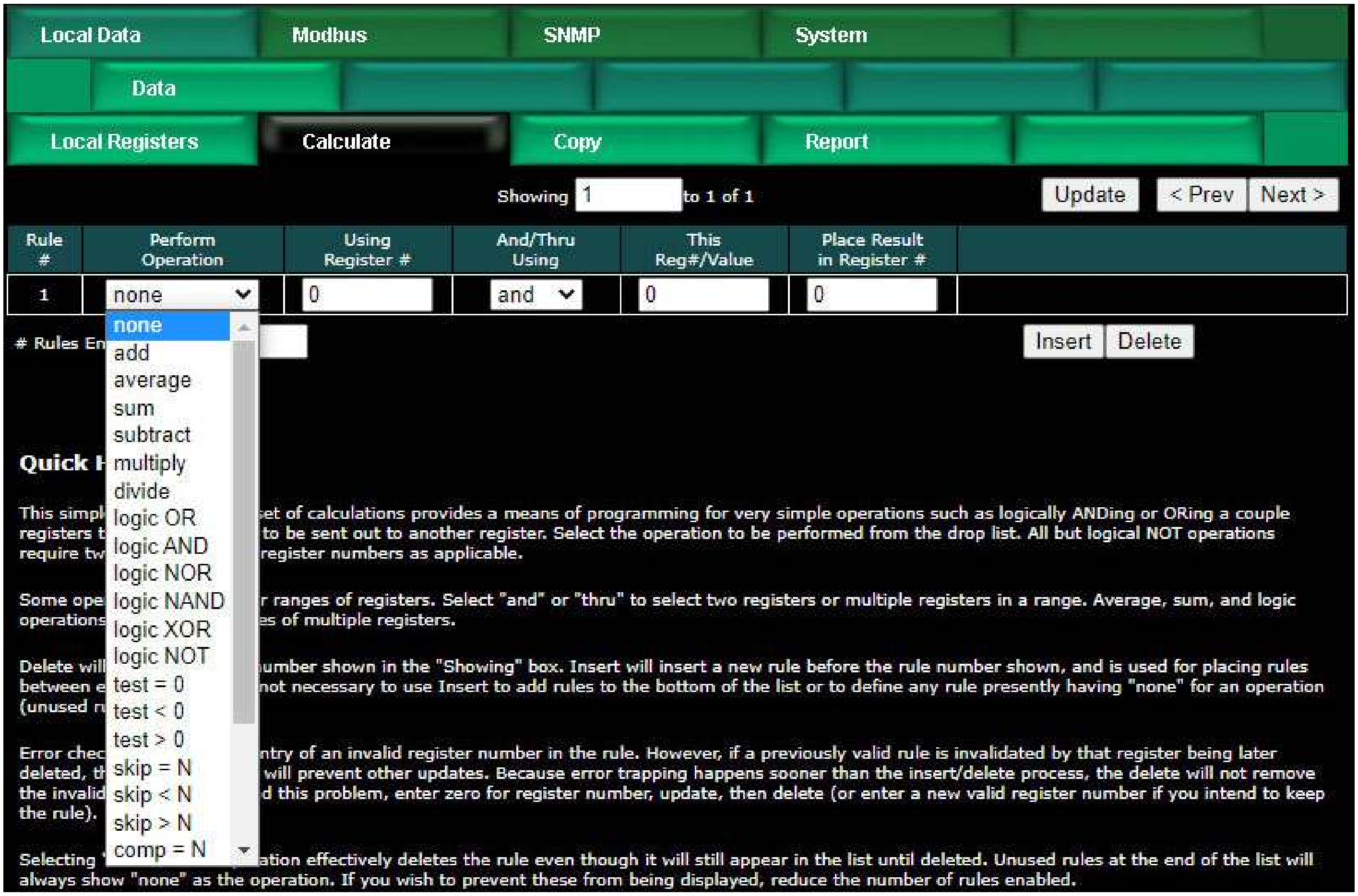
If this register is being used to hold a value provided by some other device acting as master, and you want a way of externally detecting when that device has failed to communicate, you can cause a "flag" value to appear in this register when a new value has not been received within the given timeout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

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.

4.3 Local Register Calculate Rules

The Babel Buster BB3-6101 or MX-61 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".



Showing 1 to 1 of 1

Rule #	Perform Operation	Using Register #	And/Thru Using	This Reg#/Value	Place Result in Register #
1	none	0	and	0	0

Rules En

Quick H

This simpl set of calculations provides a means of programming for very simple operations such as logically ANDing or ORing a couple
registers t to be sent out to another register. Select the operation to be performed from the drop list. All but logical NOT operations
require tw register numbers as applicable.

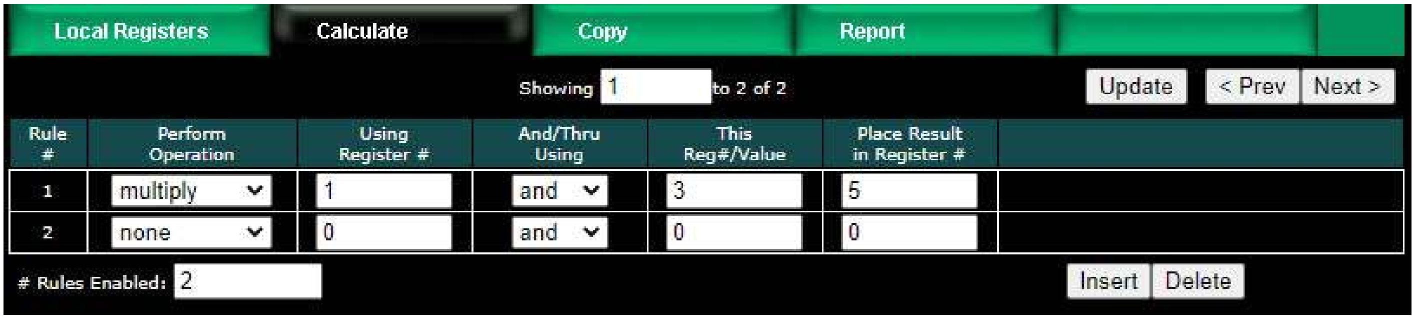
Some ope r ranges of registers. Select "and" or "thru" to select two registers or multiple registers in a range. Average, sum, and logic
operations es of multiple registers.

Delete will umber shown in the "Showing" box. Insert will insert a new rule before the rule number shown, and is used for placing rules
between e not necessary to use Insert to add rules to the bottom of the list or to define any rule presently having "none" for an operation
(unused r

Error chec ntry of an invalid register number in the rule. However, if a previously valid rule is invalidated by that register being later
deleted, th will prevent other updates. Because error trapping happens sooner than the insert/delete process, the delete will not remove
the invalid d this problem, enter zero for register number, update, then delete (or enter a new valid register number if you intend to keep
the rule).

Selecting ' ation 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 operation. If you wish to prevent these from being displayed, reduce the number of rules enabled.

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

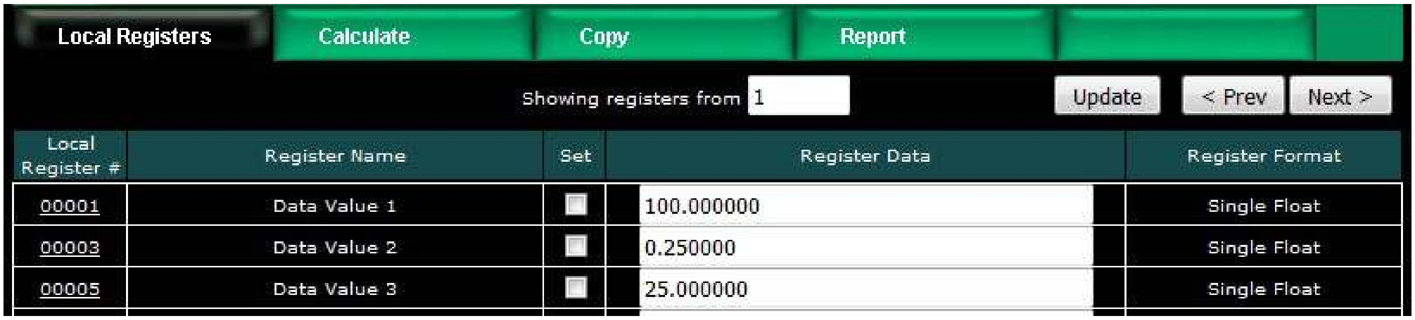


Showing 1 to 2 of 2

Rule #	Perform Operation	Using Register #	And/Thru Using	This Reg#/Value	Place Result in Register #
1	multiply	1	and	3	5
2	none	0	and	0	0

Rules Enabled: 2

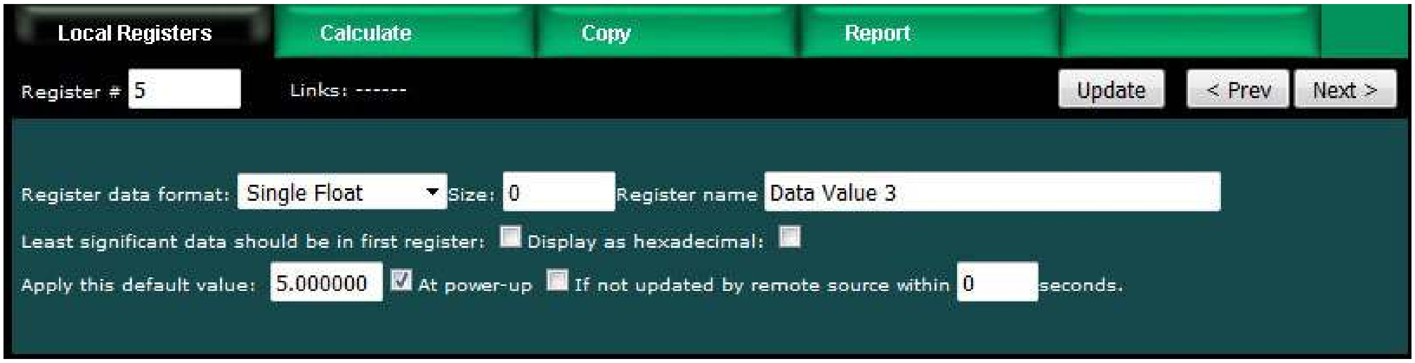
If registers 1 and 3 contain the values shown below, then the result shown in register 5 would be expected.



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	100.000000	Single Float
00003	Data Value 2	□	0.250000	Single Float
00005	Data Value 3	□	25.000000	Single Float

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.



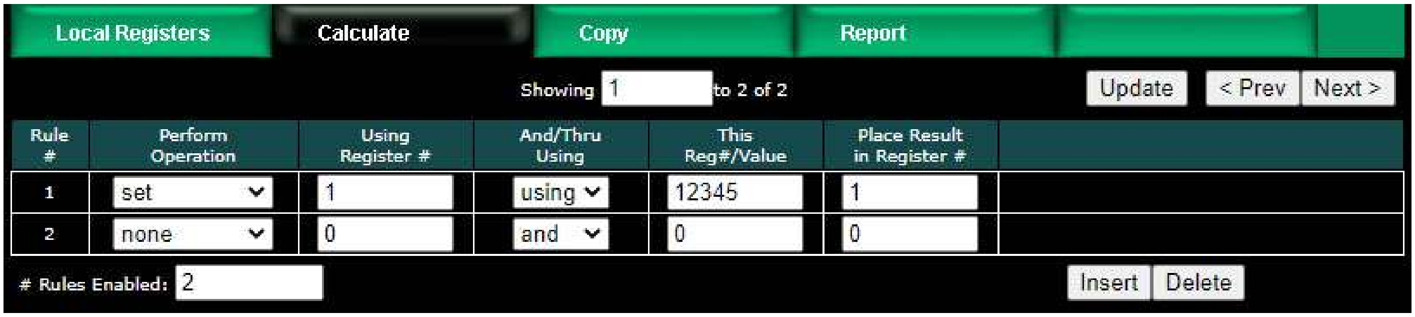
Register # 5 Links: ------

Register data format: Single Float Size: 0 Register name Data Value 3

Least significant data should be in first register: □ Display as hexadecimal: □

Apply this default value: 5.000000 ☑ At power-up □ If not updated by remote source within 0 seconds.

The other option is to use the set operation, but the value is limited to unsigned integer with the set operation. The example illustrated below will set register 1 to a value of 12345.



Showing 1 to 2 of 2

Rule #	Perform Operation	Using Register #	And/Thru Using	This Reg#/Value	Place Result in Register #
1	set	1	using	12345	1
2	none	0	and	0	0

Rules Enabled: 2

Operations available on two or more registers using 'and' or 'thru':

add	Add two registers
average	Average two or more registers
sum	Sum two or more registers
subtract	Subtract second register from first
multiply	Multiply two registers
divide	Divide first register by second
logic OR	Logically OR two or more registers
logic AND	Logically AND two or more registers
logic NOR	Logically NOR two or more registers
logic NAND	Logically NAND two or more registers
logic XOR	Logically Exclusive-OR two registers

Operations available on one register:

logic NOT	Generate bit-wise negation of register
test = 0	Set result to 'true' if register is zero
test < 0	Set result to 'true' if register is less than zero
test > 0	Set result to 'true' if register is greater than zero

Operations available on one register 'using' a given value:

set	Set register to given value (unsigned 32-bit integer)
skip = N	Skip next operation if register is equal to given value
skip < N	Skip next operation if register is less than given value
skip > N	Skip next operation if register is greater than given value
comp = N	Compare, set result 'true' if register is equal to given value
comp < N	Compare, set result 'true' if register is less than given value
comp > N	Compare, set result 'true' if register is greater than given value
pack	Perform Pack operation (see text)
fill	Perform Fill operation (see text)
unpack	Perform Unpack operation (see text)

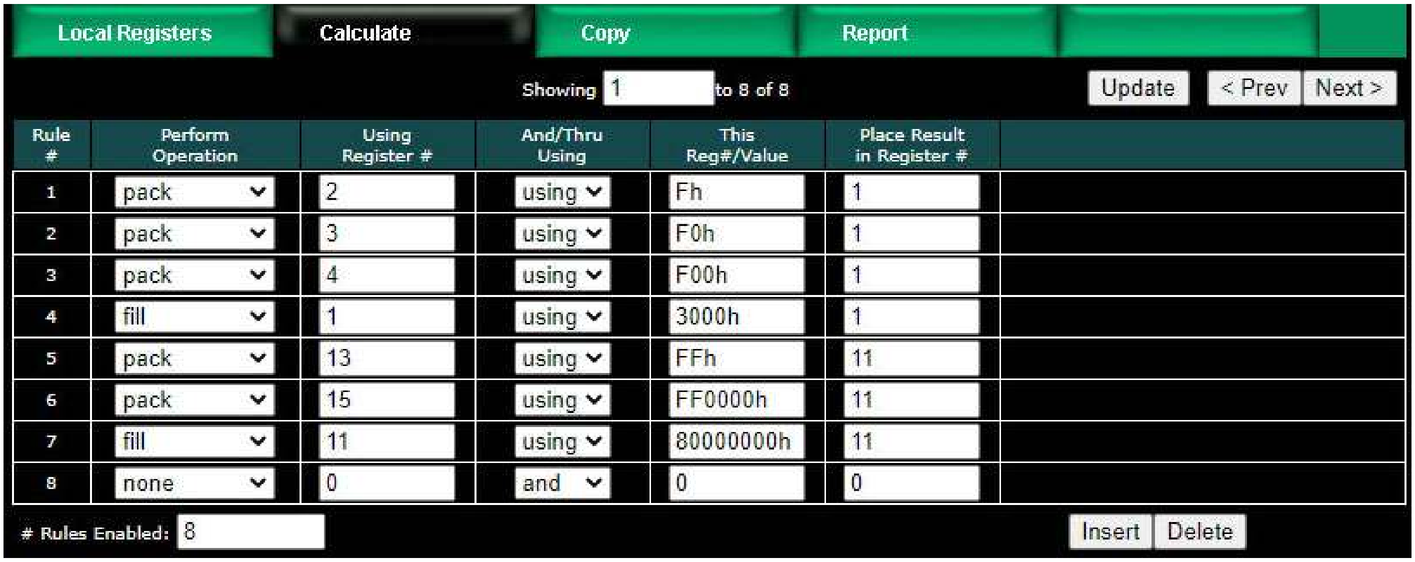
Operations "using" a given value will have an unsigned integer value in the "This Reg#/Value" column rather than a register number. These values will be displayed as integer for most operations, but will be displayed in hexadecimal for pack, fill, and unpack operations since these operate primarily on bit mask values.

The result of a test or compare will be zero if false, or one(s) if true. The true value will be the maximum unsigned 16-bit or 32-bit integer value if the result register is integer. If displayed as unsigned hexadecimal, it will be FFFF or FFFFFFFF. Displayed as signed 16-bit integer, "true" will be 32767. If the result register is floating point, then "true" will just be 1.0. The purpose of using FFFF for unsigned integer true is so that the result is useful as a bitmask.

Pack and fill are used for packing multiple local registers into a single register for purposes of emulating existing equipment when the Babel Buster is functioning as a server (slave). When pack and fill are used, "using" should be selected, and the second entry is a hexadecimal mask or fill value.

The pack mask is both a bit mask and position indicator. To calculate the contribution of a given calculate rule, the mask is right shifted until the least significant bit is nonzero, then this shifted mask is logically AND-ed with the local register content. The resulting masked value is then left shifted back to the original mask position. This final shifted result is then logically OR-ed into the result register (after first clearing the bits in the affected position of the result register).

Fill is simple - it simply logically OR's the bit mask into the result register.

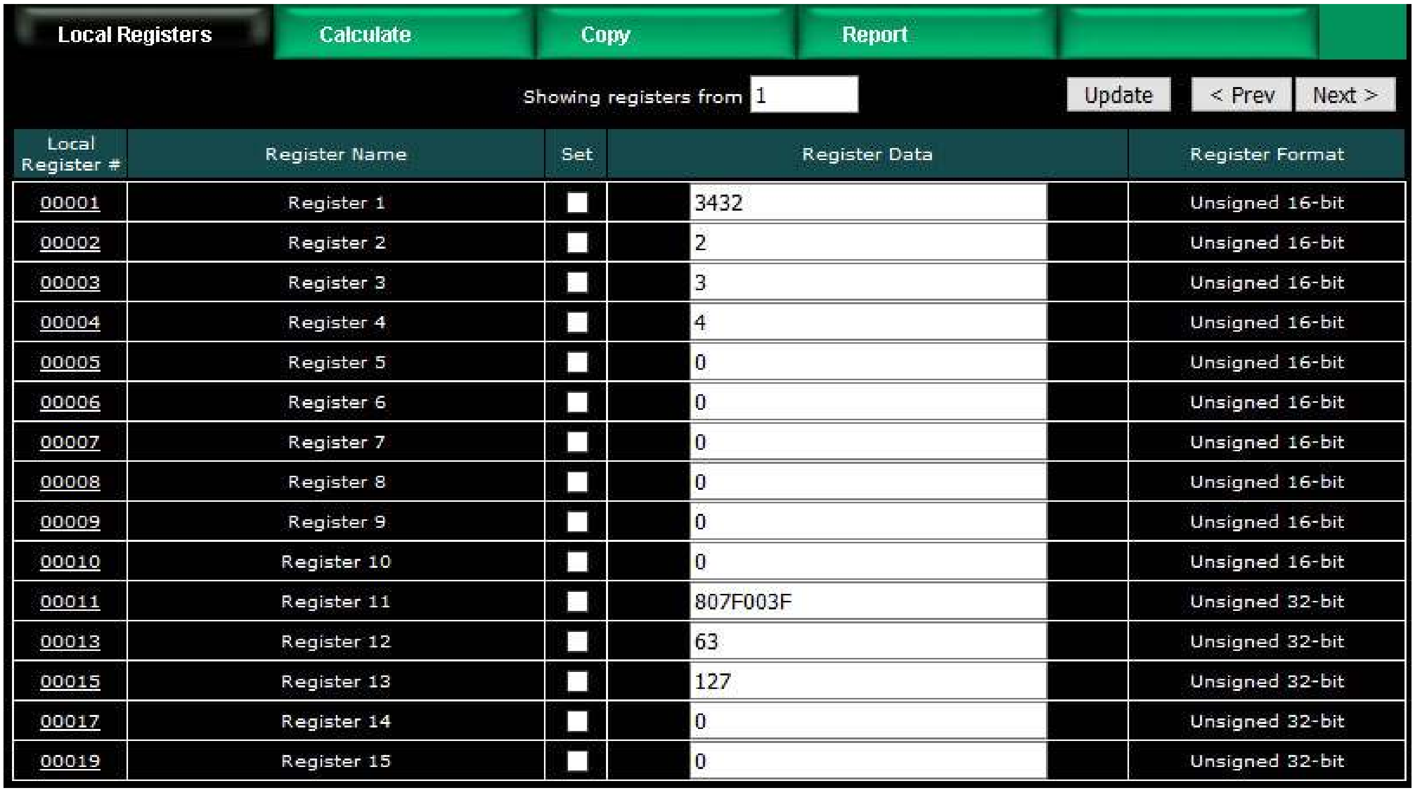


Showing 1 to 8 of 8

Rule #	Perform Operation	Using Register #	And/Thru Using	This Reg#/Value	Place Result in Register #
1	pack	2	using	Fh	1
2	pack	3	using	F0h	1
3	pack	4	using	F00h	1
4	fill	1	using	3000h	1
5	pack	13	using	FFh	11
6	pack	15	using	FF0000h	11
7	fill	11	using	80000000h	11
8	none	0	and	0	0

Rules Enabled: 8

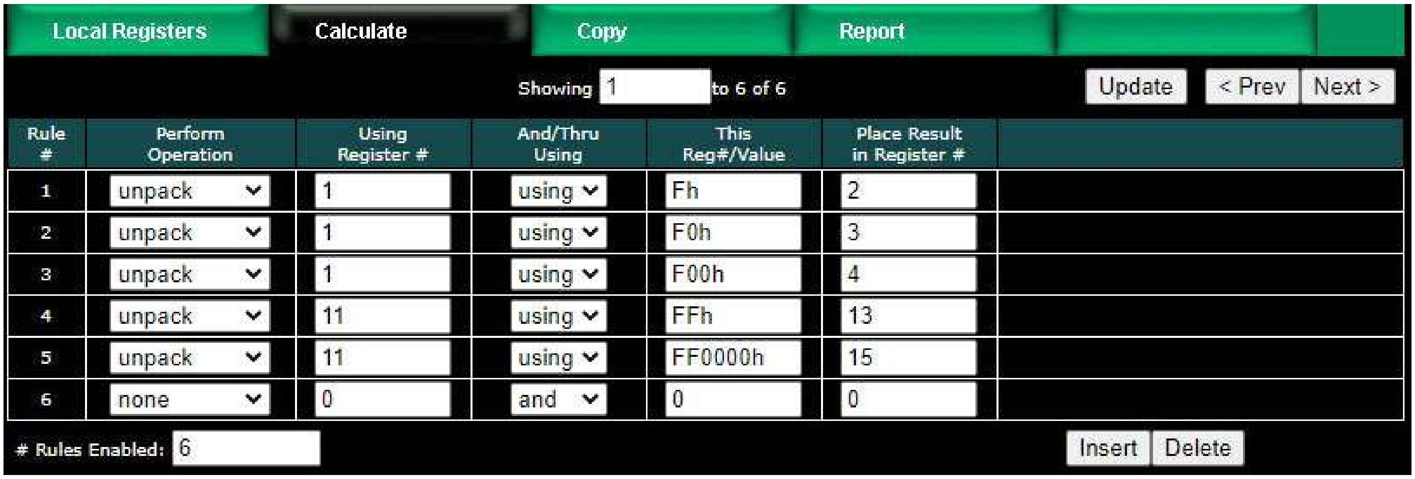
The example below shows the content of result registers 1 and 11 using the above calculate rules and the local register values shown. The contents of registers 2, 3, and 4 are packed into register 1. The contents of registers 13 and 15 are packed into 11 (all 32-bit register pairs).



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Register 1	□	3432	Unsigned 16-bit
00002	Register 2	□	2	Unsigned 16-bit
00003	Register 3	□	3	Unsigned 16-bit
00004	Register 4	□	4	Unsigned 16-bit
00005	Register 5	□	0	Unsigned 16-bit
00006	Register 6	□	0	Unsigned 16-bit
00007	Register 7	□	0	Unsigned 16-bit
00008	Register 8	□	0	Unsigned 16-bit
00009	Register 9	□	0	Unsigned 16-bit
00010	Register 10	□	0	Unsigned 16-bit
00011	Register 11	□	807F003F	Unsigned 32-bit
00013	Register 12	□	63	Unsigned 32-bit
00015	Register 13	□	127	Unsigned 32-bit
00017	Register 14	□	0	Unsigned 32-bit
00019	Register 15	□	0	Unsigned 32-bit

This process can be reversed using the "unpack" operation. The following calculate rules exactly reverse the above packing operations (discarding fill in this case).

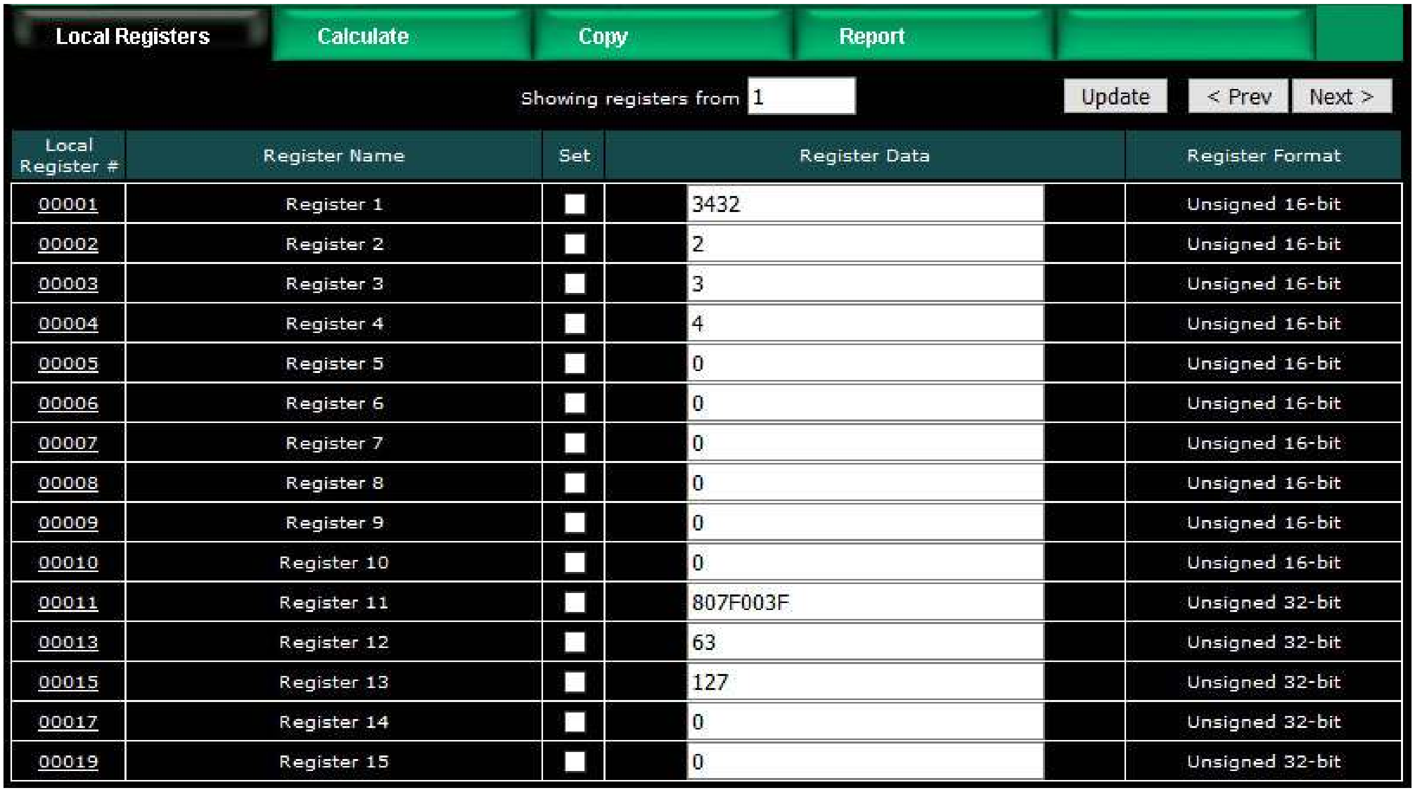


Showing 1 to 6 of 6

Rule #	Perform Operation	Using Register #	And/Thru Using	This Reg#/Value	Place Result in Register #
1	unpack	1	using	Fh	2
2	unpack	1	using	F0h	3
3	unpack	1	using	F00h	4
4	unpack	11	using	FFh	13
5	unpack	11	using	FF0000h	15
6	none	0	and	0	0

Rules Enabled: 6

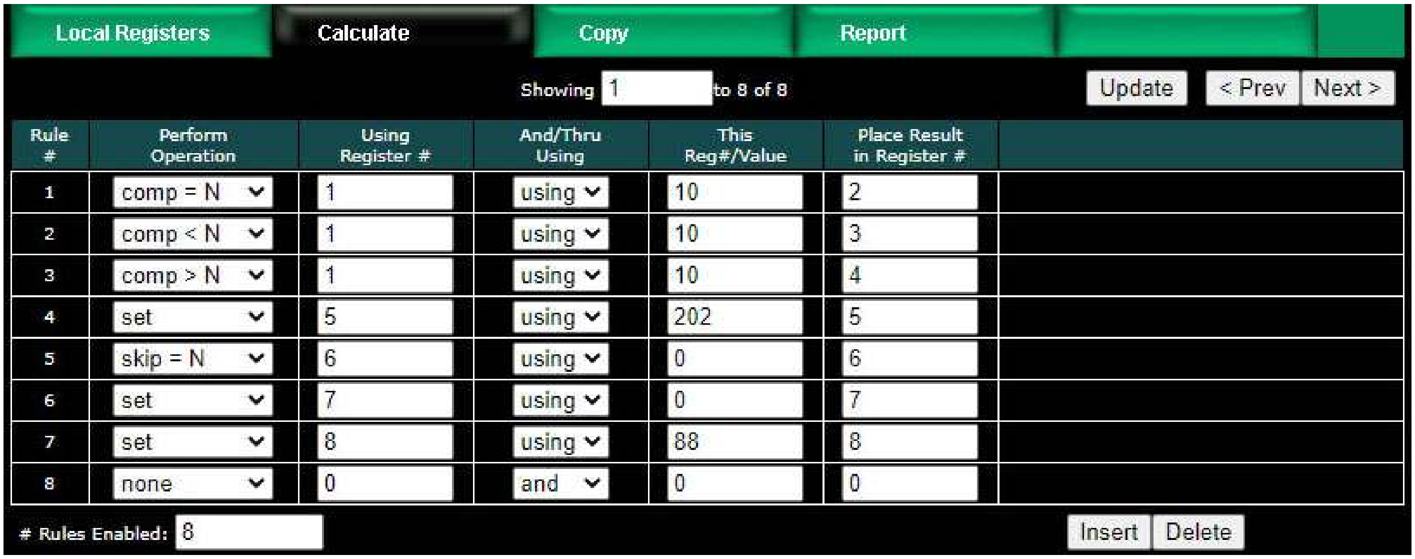
Register 1 is unpacked into registers 2, 3 and 4. Register 11 is unpacked into registers 13 and 15 (all 32-bit register pairs).



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Register 1	□	3432	Unsigned 16-bit
00002	Register 2	□	2	Unsigned 16-bit
00003	Register 3	□	3	Unsigned 16-bit
00004	Register 4	□	4	Unsigned 16-bit
00005	Register 5	□	0	Unsigned 16-bit
00006	Register 6	□	0	Unsigned 16-bit
00007	Register 7	□	0	Unsigned 16-bit
00008	Register 8	□	0	Unsigned 16-bit
00009	Register 9	□	0	Unsigned 16-bit
00010	Register 10	□	0	Unsigned 16-bit
00011	Register 11	□	807F003F	Unsigned 32-bit
00013	Register 12	□	63	Unsigned 32-bit
00015	Register 13	□	127	Unsigned 32-bit
00017	Register 14	□	0	Unsigned 32-bit
00019	Register 15	□	0	Unsigned 32-bit

The next two screen shots illustrate compare, set, and skip operations. Rule 5 says that rule 6 will not be executed if register 6 contains a zero. If register 6 is not equal to zero, then rule 6 will be executed. (The numbers rule 6 and register 6 are not related in any other way, this is just conincidence in the example.)

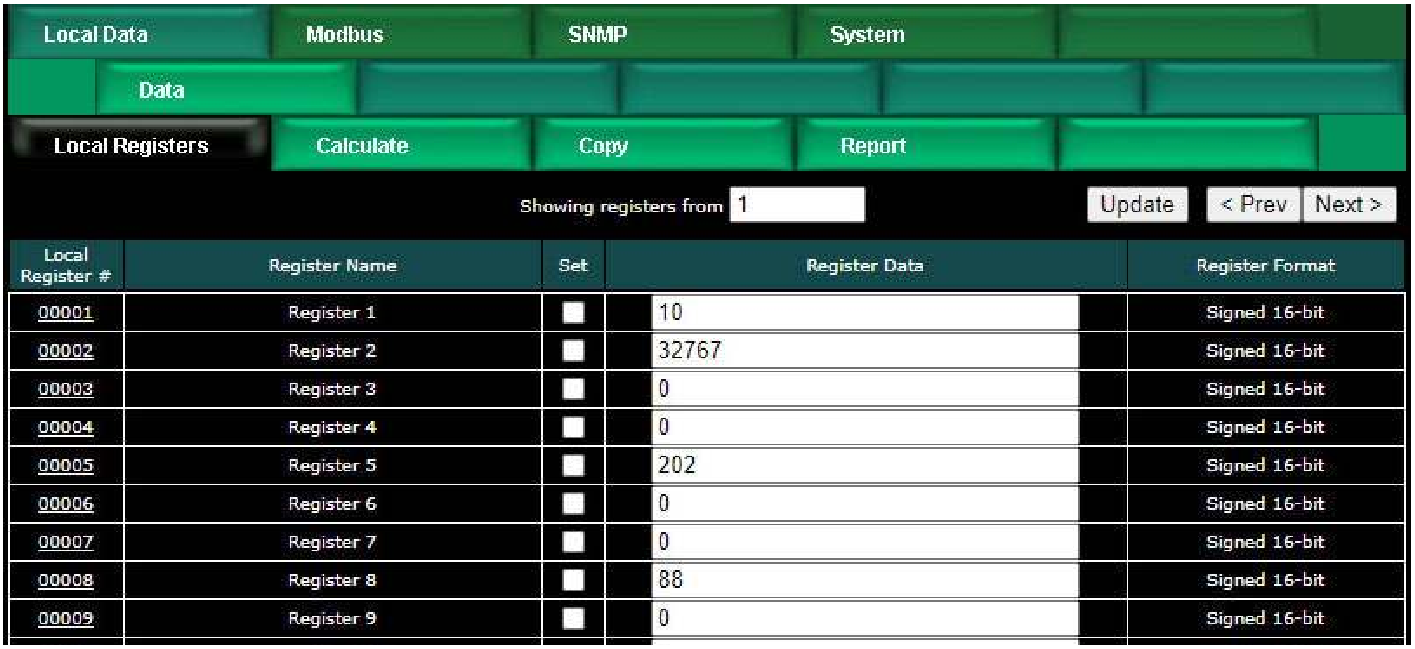


Showing 1 to 8 of 8

Rule #	Perform Operation	Using Register #	And/Thru Using	This Reg#/Value	Place Result in Register #
1	comp = N	1	using	10	2
2	comp < N	1	using	10	3
3	comp > N	1	using	10	4
4	set	5	using	202	5
5	skip = N	6	using	0	6
6	set	7	using	0	7
7	set	8	using	88	8
8	none	0	and	0	0

Rules Enabled: 8

Register values for examples using the above operations are illustrated below.

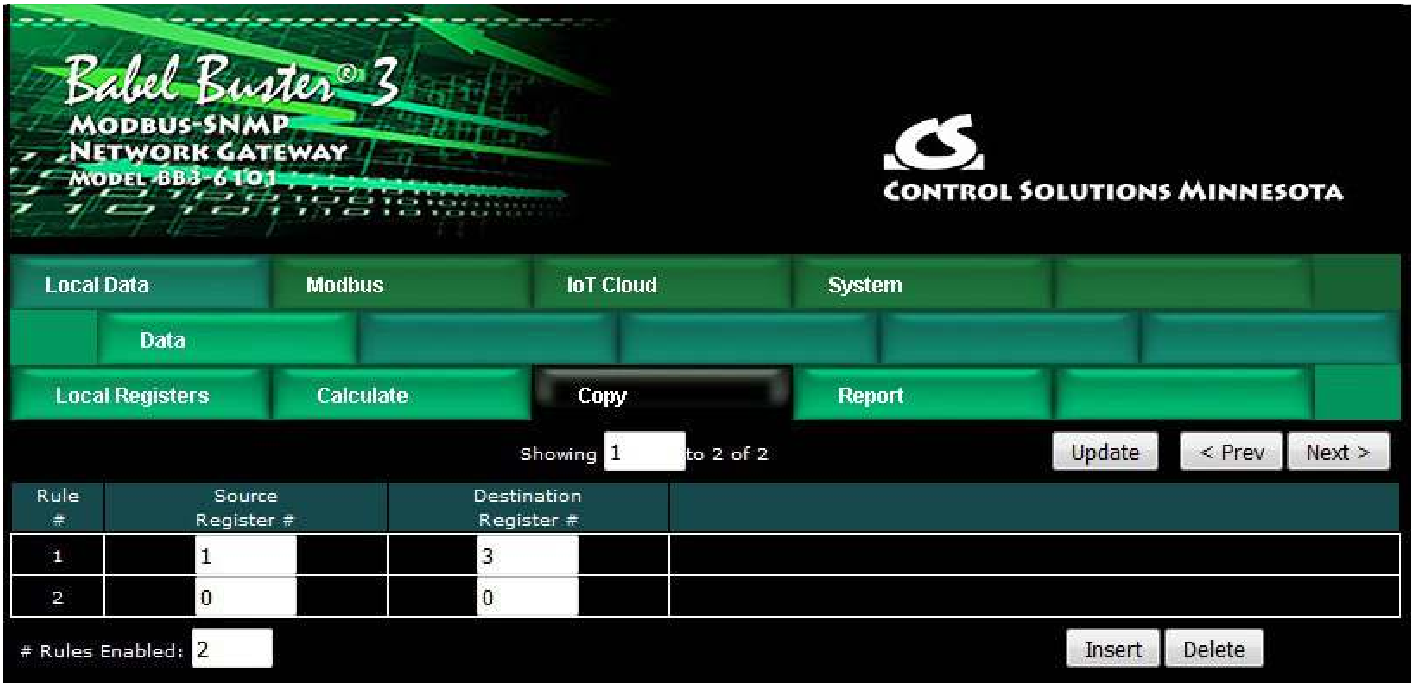


Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Register 1	□	10	Signed 16-bit
00002	Register 2	□	32767	Signed 16-bit
00003	Register 3	□	0	Signed 16-bit
00004	Register 4	□	0	Signed 16-bit
00005	Register 5	□	202	Signed 16-bit
00006	Register 6	□	0	Signed 16-bit
00007	Register 7	□	0	Signed 16-bit
00008	Register 8	□	88	Signed 16-bit
00009	Register 9	□	0	Signed 16-bit

4.4 Local Register Copy Rules

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

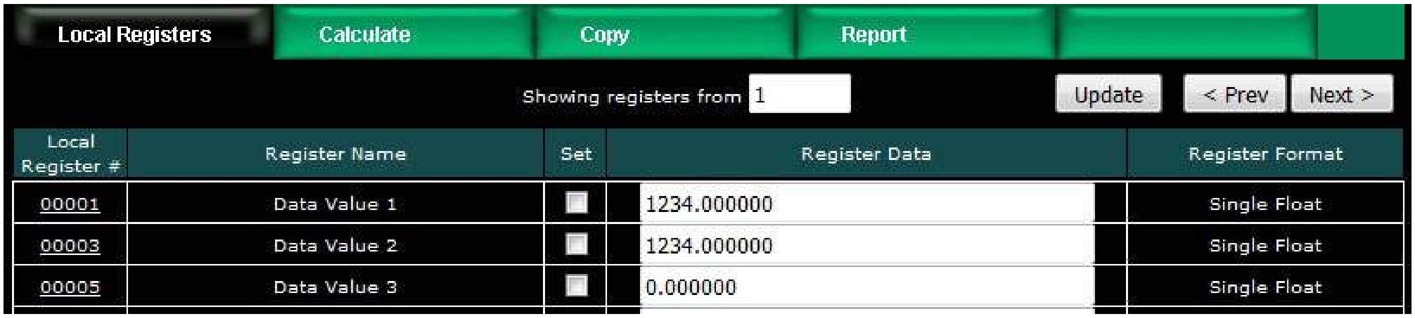


Showing 1 to 2 of 2

Rule #	Source Register #	Destination Register #
1	1	3
2	0	0

Rules Enabled: 2

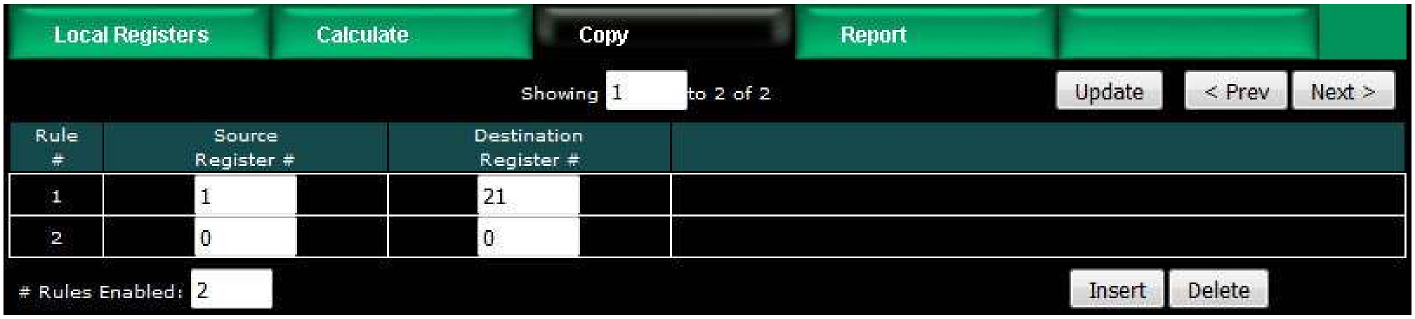
The above rule would cause the following data copy to occur:



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	1234.000000	Single Float
00003	Data Value 2	□	1234.000000	Single Float
00005	Data Value 3	□	0.000000	Single Float

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

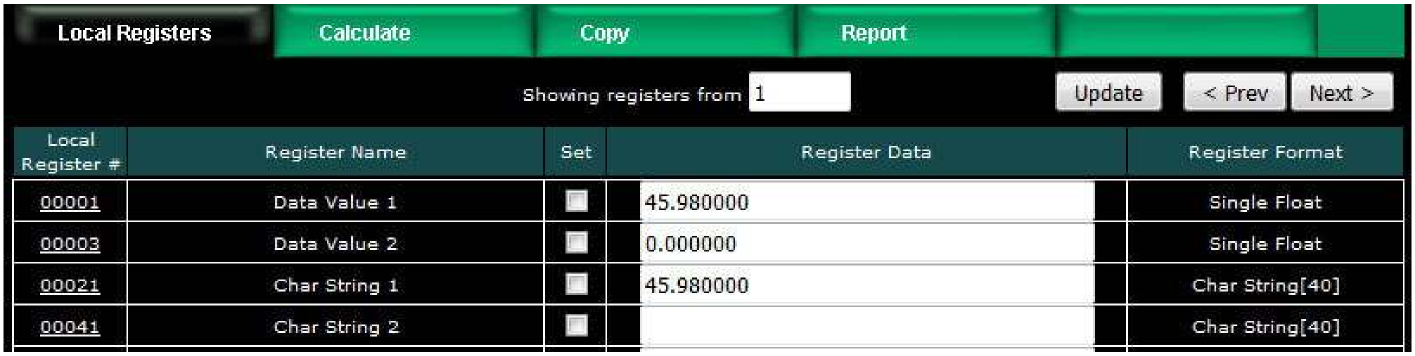


Showing 1 to 2 of 2

Rule #	Source Register #	Destination Register #
1	1	21
2	0	0

Rules Enabled: 2

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.



Showing registers from 1

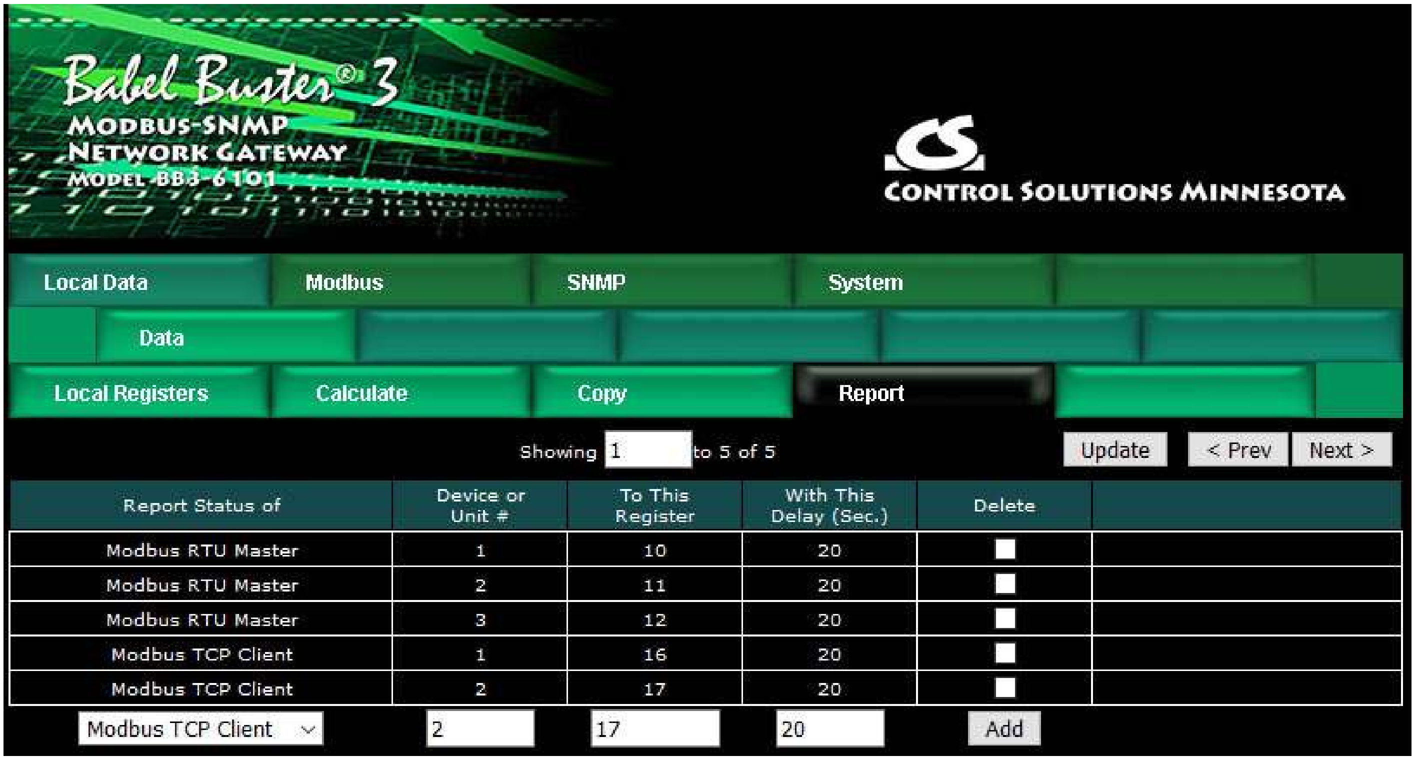
Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	45.980000	Single Float
00003	Data Value 2	□	0.000000	Single Float
00021	Char String 1	□	45.980000	Char String[40]
00041	Char String 2	□		Char String[40]

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.

4.5 Device Status Reporting

The Babel Buster BB3-6101/MX-61 read maps include the ability to set a default value upon 'n' read fails, meaning that if the Babel Buster gets an error 'n' times attempting to read that point, it will automatically set the corresponding local register to the given default value to indicate the problem. This indication applies on a point by point basis, but of course any one point can be used as an indication that the entire device may be offline.

The BB3-6101/MX-61 also includes the ability to report device errors to an assigned status register rather than rely on default values. This reporting is configured on the Report page.



Showing 1 to 5 of 5

Report Status of	Device or Unit #	To This Register	With This Delay (Sec.)	Delete
Modbus RTU Master	1	10	20	□
Modbus RTU Master	2	11	20	□
Modbus RTU Master	3	12	20	□
Modbus TCP Client	1	16	20	□
Modbus TCP Client	2	17	20	□
Modbus TCP Client	2	17	20	Add

This optional list allows reporting device errors as register values to make it easier to monitor communication failures. The length of the list is variable. To add to the list, select the type of device to report on, select the device instance or unit number to report on, and select a register in which to put the status indication. Enter a delay if desired, and then click Add.

The delay is optional. If zero, there is no delay. If some number of seconds is entered, then the error condition will not be reported until this time period expires. If the error clears before the time is up, then the error is never reported. This is useful for spurious errors that would result in nuisance indications.

To remove a report from the list, check the box in the Delete column and then click Update. Click Prev or Next to scroll through the list.

Error codes placed into the reporting register will be as follows:
0 = No error
1 = Timeout, no response from remote device
2 = Error message received from remote device (e.g. Modbus exception)
3 = Line fault (e.g. CRC error, socket connection error, etc)

Once a Timeout error indication has been set (following delay if applicable), it will automatically return to zero upon the next successful communication with that device.

Once either the error message or line fault indication has been set, following delay if applicable, communication must continue free of this same error condition for at least the same delay period before the indication will be reset to zero. If an error message (e.g. Modbus exception) is reported for one data point, but multiple others are error free, then the one error would be hidden without this delay before reset. Ideally, this delay period should be at least as great as the poll period for the slowest point mapped.

Babel Buster® 3
MODEL BB3-6101
MODEL MX-61
NETWORK GATEWAY

CONTROL SOLUTIONS MINNESOTA

5. Configuring Gateway as a Modbus RTU Master

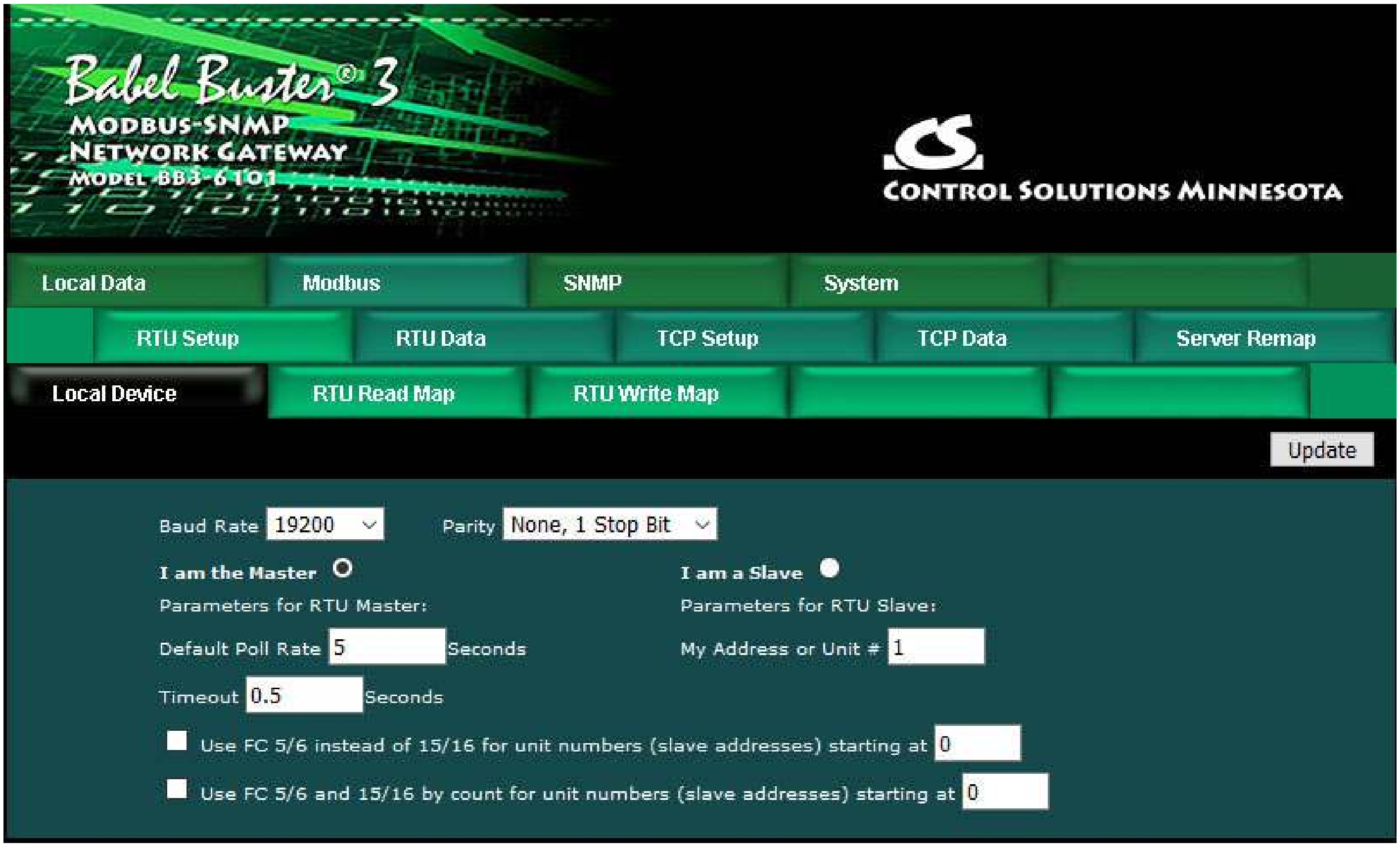
The Babel Buster BB3-6101/MX-61 can be a Modbus RTU master or slave. As a master you can read Modbus data from, or write Modbus data to, other Modbus slaves. The gateway will periodically poll the other Modbus devices according to register maps you set up. To read from a remote Modbus device, configure 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 SNMP.

The following section details the process of setting up read and write maps via the web interface. Once familiar with map content, you have the option of importing bulk configurations as a CSV file. The import function is found on the File Manager page, and details about the content and format of the CSV file are found in Appendix B.

5.1 Modbus RTU Device Configuration

Modbus device configuration for RTU really consists of just port configuration.



Baud Rate 19200 Parity None, 1 Stop Bit

I am the Master ○ I am a Slave ●

Parameters for RTU Master:

Default Poll Rate 5 Seconds

Timeout 0.5 Seconds

Parameters for RTU Slave:

My Address or Unit # 1

□ Use FC 5/6 instead of 15/16 for unit numbers (slave addresses) starting at 0

□ Use FC 5/6 and 15/16 by count for unit numbers (slave addresses) starting at 0

Select baud rate and parity from the drop down list. Click either Master or Slave buttons to select type of operation (in this section, we are discussing Master). 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.

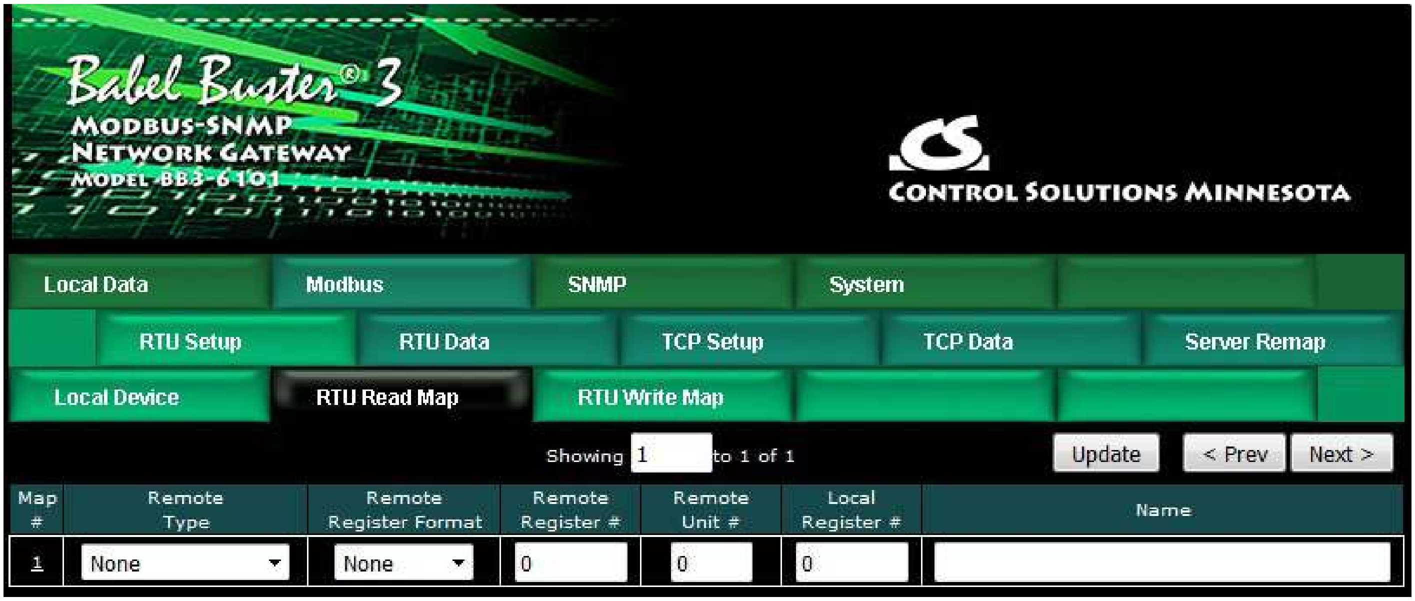
If your slave/server device only supports function codes 5 and 6 for writing coils and holding registers, check the Use FC 5/6 box. The default function codes are 15 and 16, which are most widely used. If you check the box, you should also enter a "starting at" unit # or slave address. This allows supporting 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.)

If your slave/server device is especially picky, it may require function codes 5 and 6 for single writes, but yet expect function codes 15 and 16 for multiple register writes. Select "Use FC 5/6 by count..." if this is the case, and provide a starting unit number.

The starting unit number will most often be simply "1" unless you have a mix of device types that all follow different rules for their function codes. Note that the selection of FC 5/6 only pertains to write maps. Reading registers will always use the same function codes for reading.

5.2 Modbus RTU Master Read Maps

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

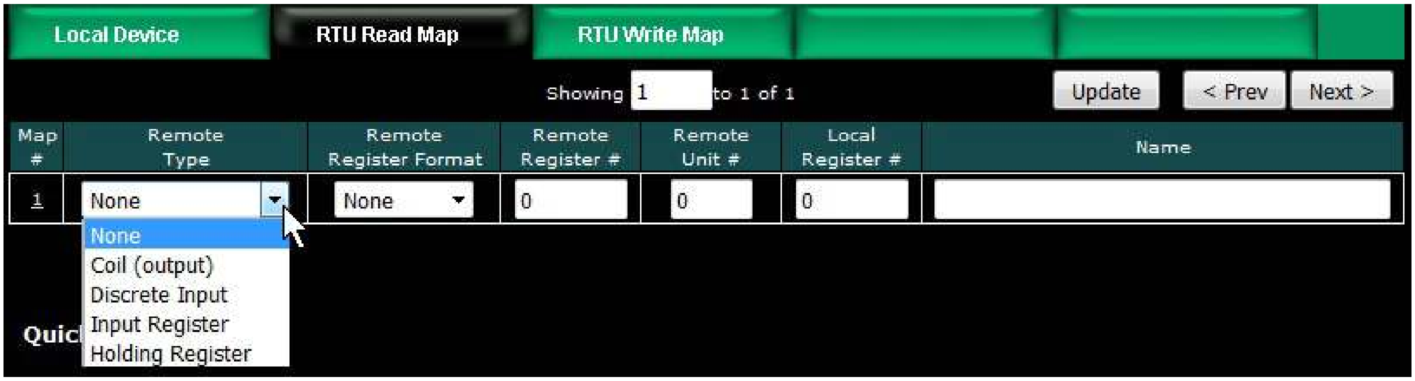


Showing 1 to 1 of 1

Map #	Remote Type	Remote Register Format	Remote Register #	Remote Unit #	Local Register #	Name
1	None	None	0	0	0	

Map 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.

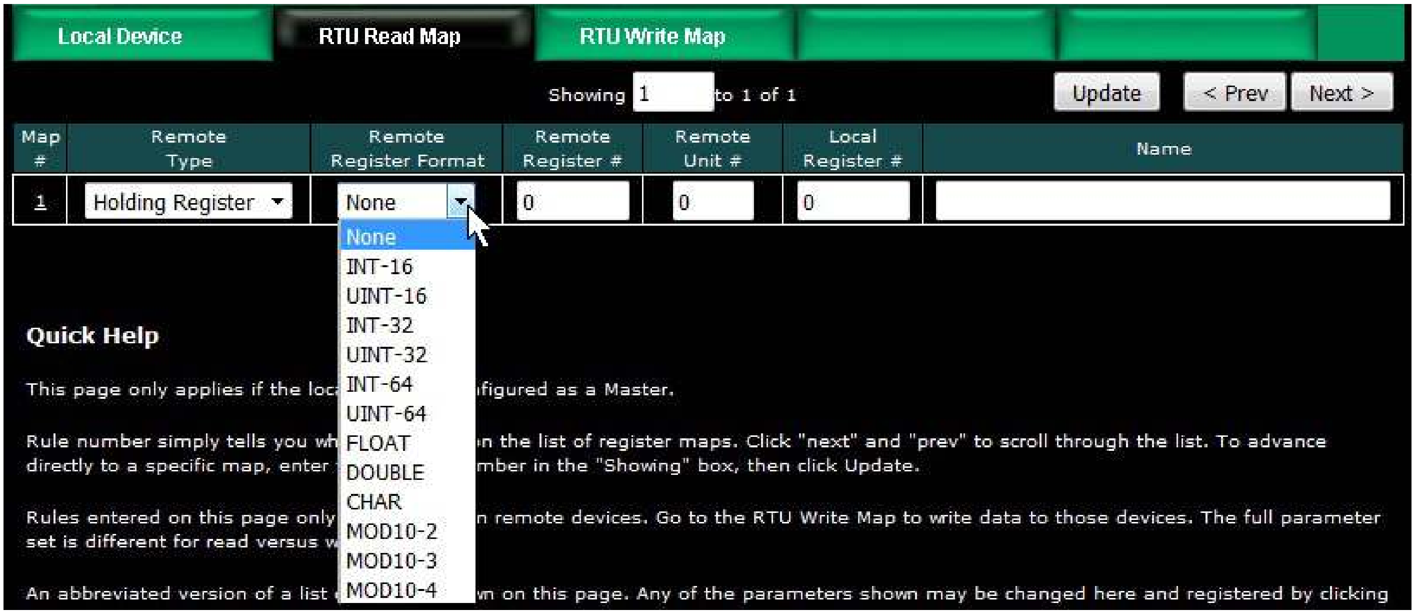
Maps entered on this page only read data from remote devices. Go to the RTU Write Map to write data to those devices. The full parameter set is different for read versus write.



Showing 1 to 1 of 1

Map #	Remote Type	Remote Register Format	Remote Register #	Remote Unit #	Local Register #	Name
1	None	None	0	0	0	

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



Showing 1 to 1 of 1

Map #	Remote Type	Remote Register Format	Remote Register #	Remote Unit #	Local Register #	Name
1	Holding Register	None	0	0	0	

Quick Help

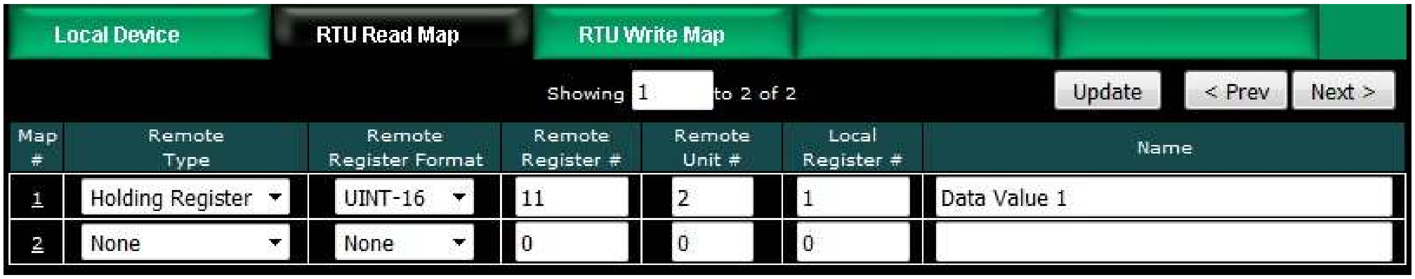
This page only applies if the loc figured as a Master.

Rule number simply tells you wh n the list of register maps. Click "next" and "prev" to scroll through the list. To advance directly to a specific map, enter mber in the "Showing" box, then click Update.

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

An abbreviated version of a list wn on this page. Any of the parameters shown may be changed here and registered by clicking

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.)

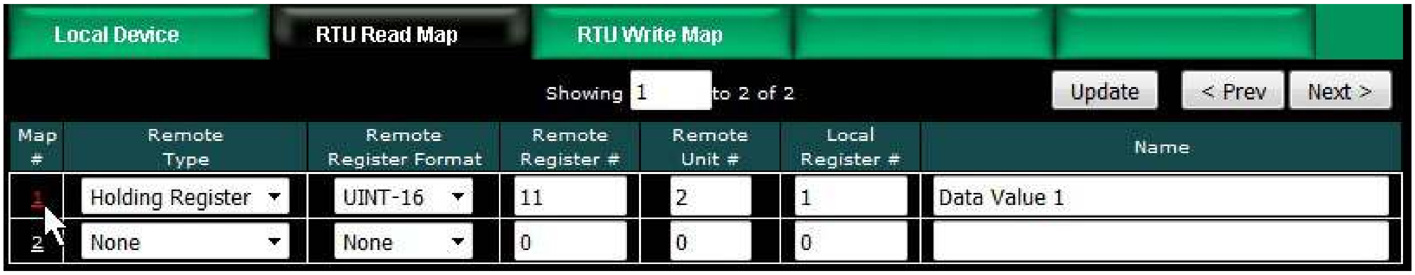


Showing 1 to 2 of 2

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	None	None	0	0	0	

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.

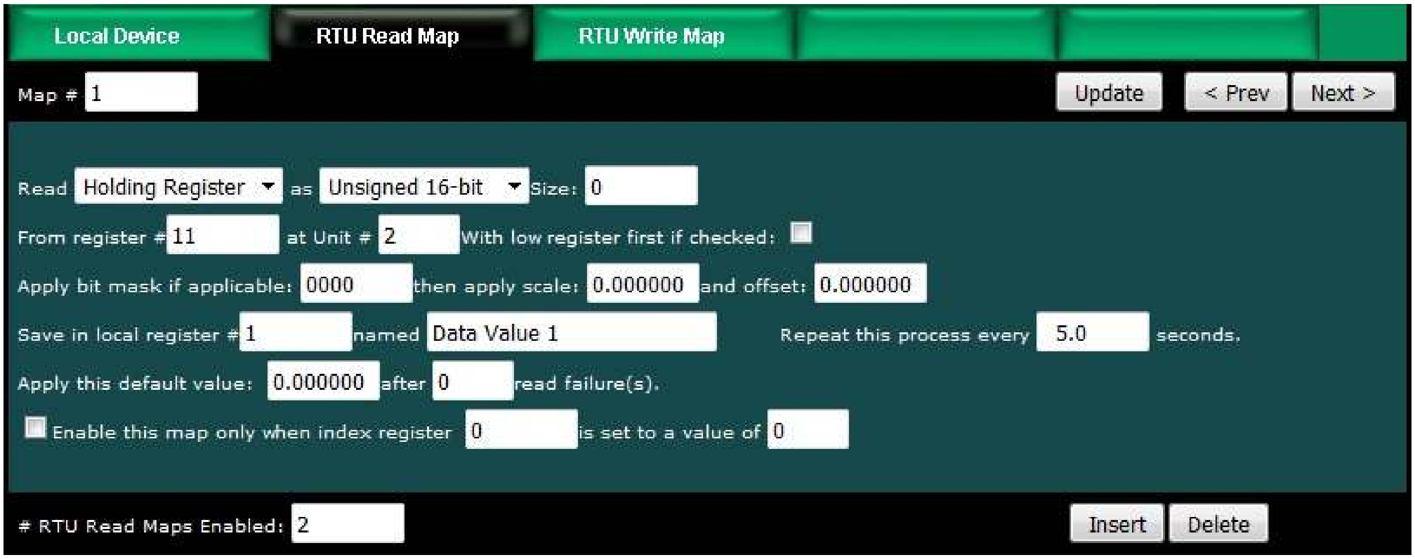
The Local Register is where data read from the remote Modbus RTU slave will be stored locally in the Babel Buster BB3-6101/MX-61. If the local register data format does not match what you are reading from the Modbus slave, the data will be converted automatically when it is read.



Showing 1 to 2 of 2

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	None	None	0	0	0	

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



Map # 1

Read Holding Register as Unsigned 16-bit Size: 0

From register # 11 at Unit # 2 With low register first if checked: □

Apply bit mask if applicable: 0000 then apply scale: 0.000000 and offset: 0.000000

Save in local register # 1 named Data Value 1 Repeat this process every 5.0 seconds.

Apply this default value: 0.000000 after 0 read failure(s).

□ Enable this map only when index register 0 is set to a value of 0

RTU Read Maps Enabled: 2

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.

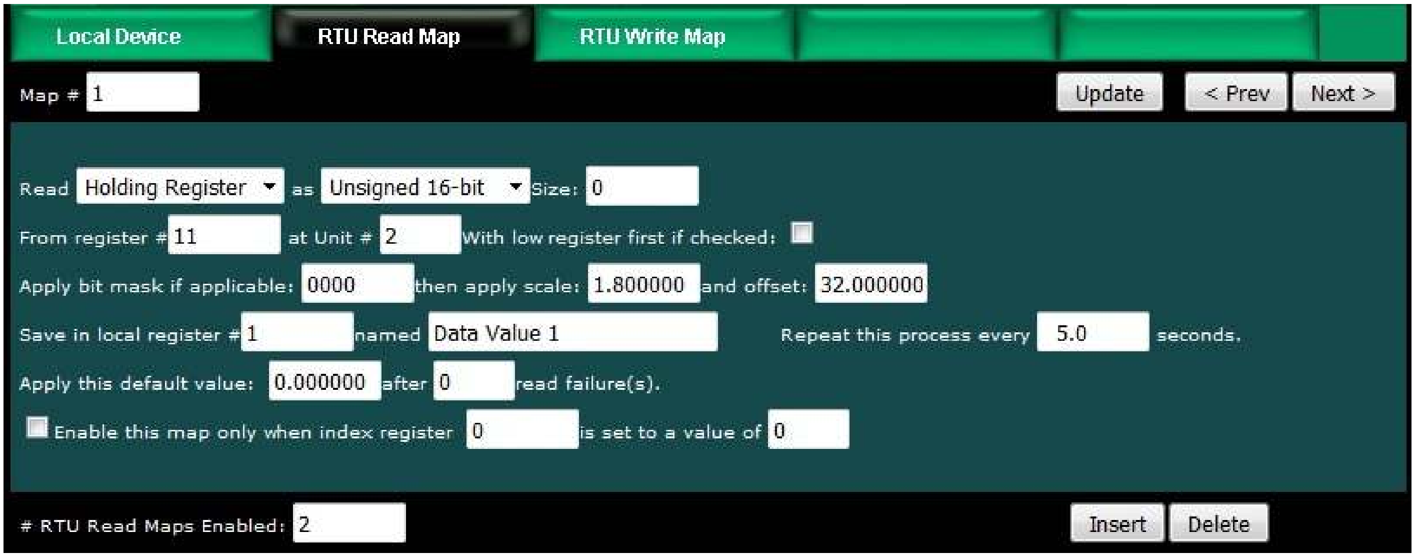
You have the option of making this Read Map conditional. If an index register number is provided and the Enable box is checked, then this read map will only be executed when the index register (local register) contains the value given. This allows multiple read maps to supply data to the same local register 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.

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

Selecting "none" for remote type effectively deletes the map even though it will still appear in the list until deleted. Unused maps 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 maps enabled.

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



Map # 1

Read Holding Register as Unsigned 16-bit Size: 0

From register # 11 at Unit # 2 With low register first if checked: □

Apply bit mask if applicable: 0000 then apply scale: 1.800000 and offset: 32.000000

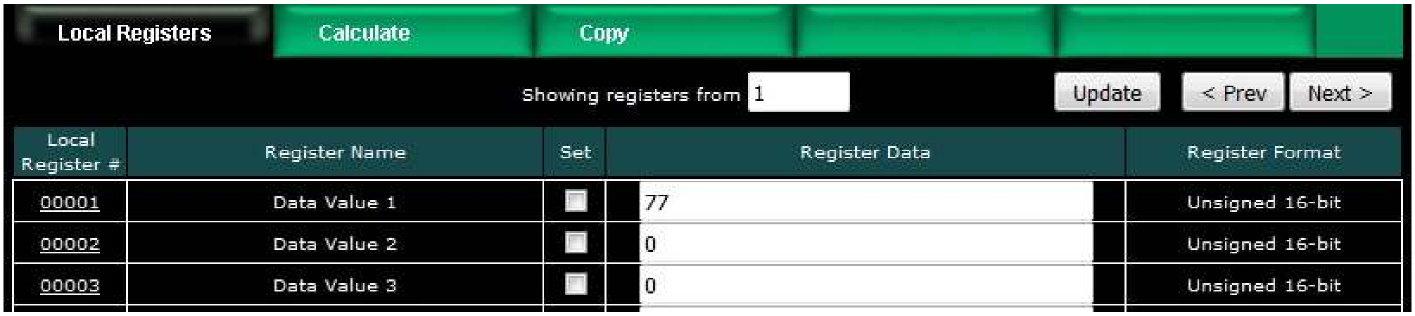
Save in local register # 1 named Data Value 1 Repeat this process every 5.0 seconds.

Apply this default value: 0.000000 after 0 read failure(s).

□ Enable this map only when index register 0 is set to a value of 0

RTU Read Maps Enabled: 2

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 multiplied 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).

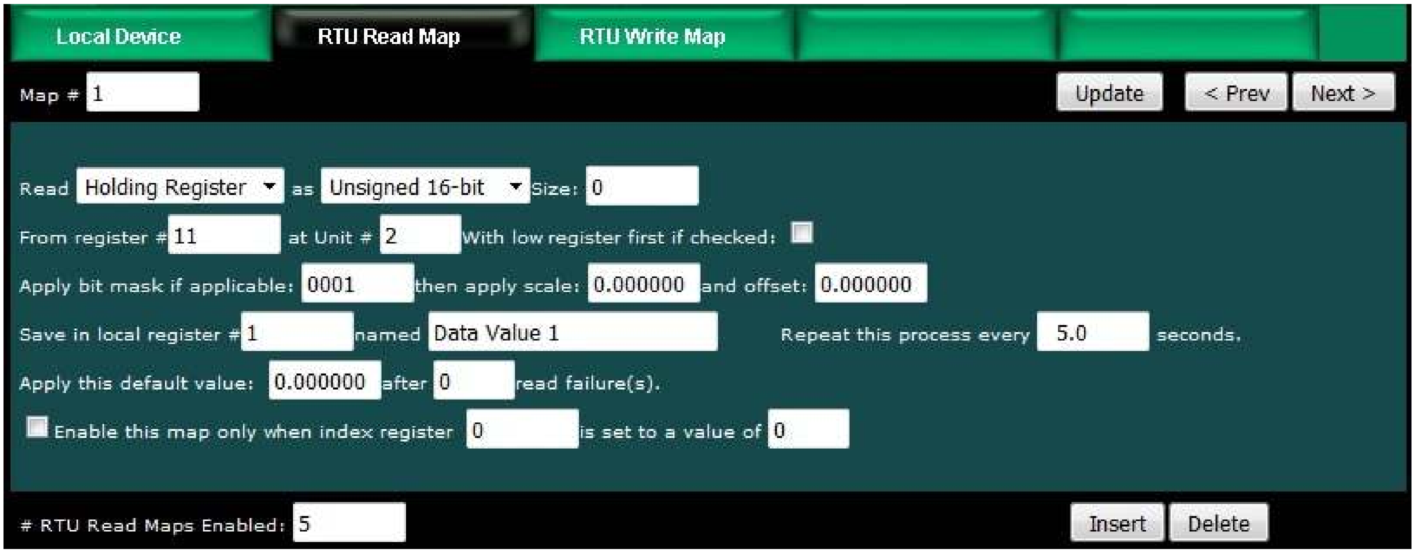


Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	77	Unsigned 16-bit
00002	Data Value 2	□	0	Unsigned 16-bit
00003	Data Value 3	□	0	Unsigned 16-bit

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, it is sometimes 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.



Map # 1

Read Holding Register as Unsigned 16-bit Size: 0

From register # 11 at Unit # 2 With low register first if checked: □

Apply bit mask if applicable: 0001 then apply scale: 0.000000 and offset: 0.000000

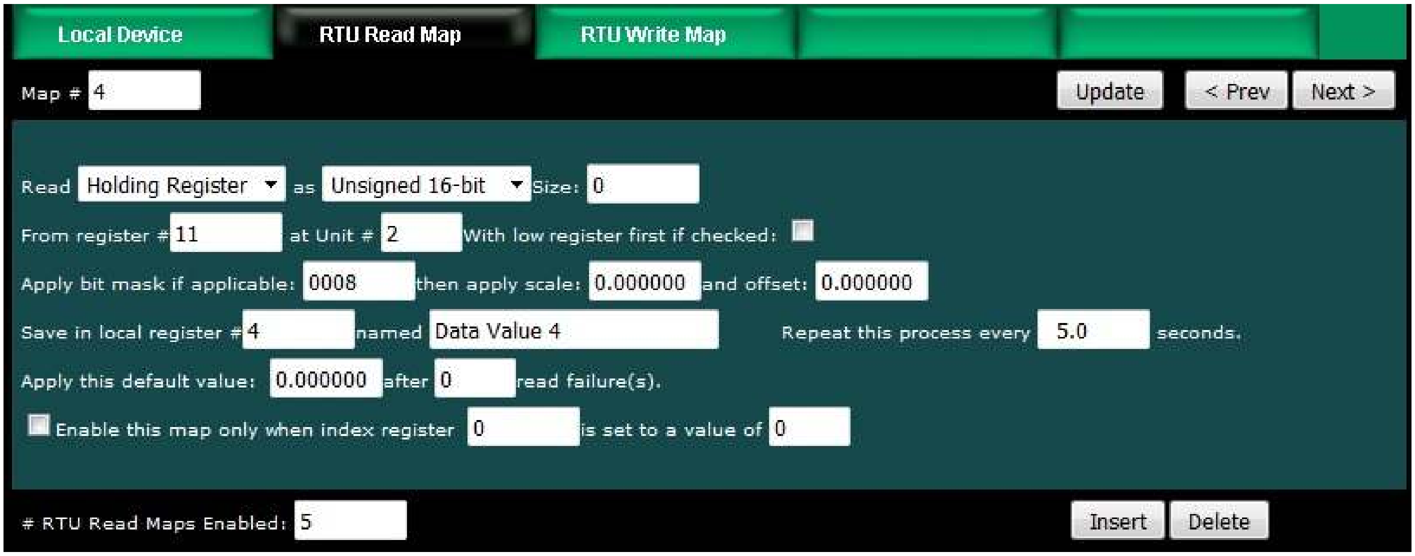
Save in local register # 1 named Data Value 1 Repeat this process every 5.0 seconds.

Apply this default value: 0.000000 after 0 read failure(s).

□ Enable this map only when index register 0 is set to a value of 0

RTU Read Maps Enabled: 5

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.



Map # 4

Read Holding Register as Unsigned 16-bit Size: 0

From register # 11 at Unit # 2 With low register first if checked: □

Apply bit mask if applicable: 0008 then apply scale: 0.000000 and offset: 0.000000

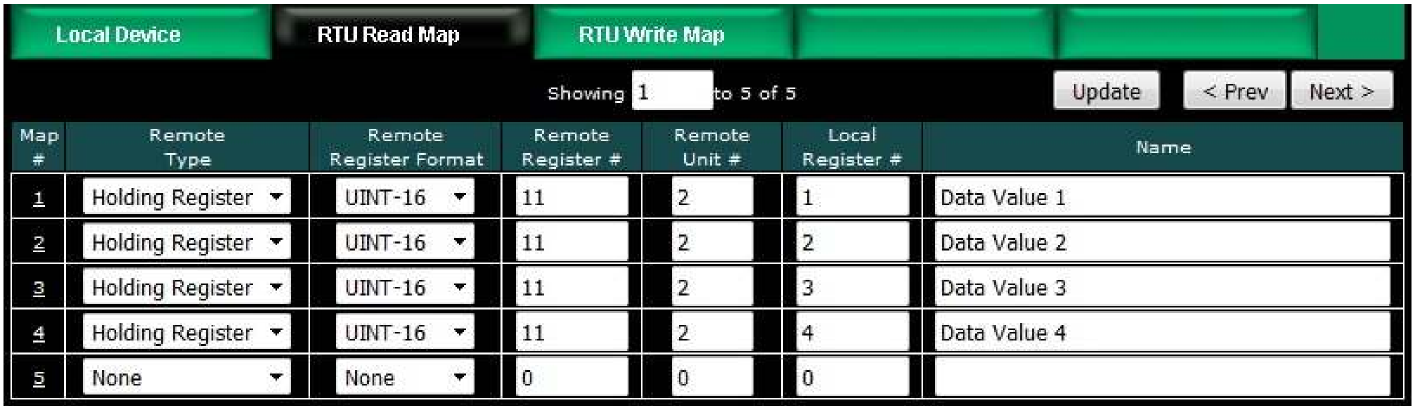
Save in local register # 4 named Data Value 4 Repeat this process every 5.0 seconds.

Apply this default value: 0.000000 after 0 read failure(s).

□ Enable this map only when index register 0 is set to a value of 0

RTU Read Maps Enabled: 5

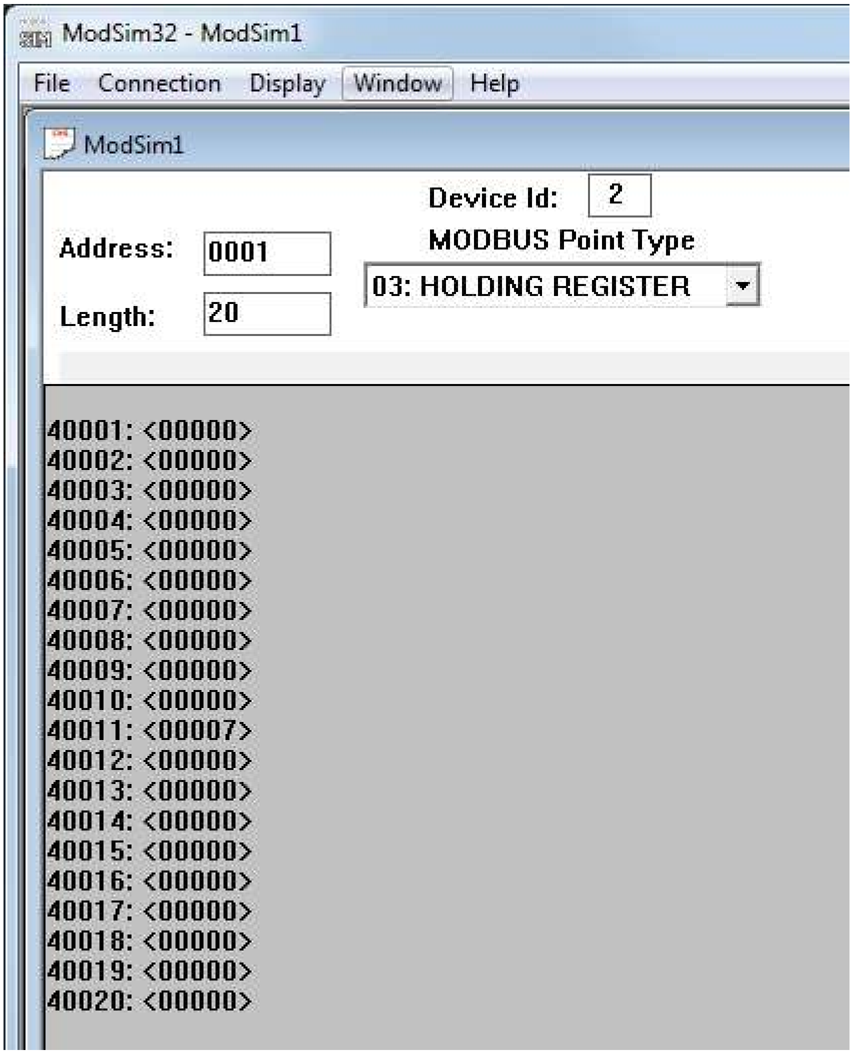
If the read maps referencing the same remote register are created in sequential contiguous order, the Babel Buster will optimize the RTU activity by reading the remote register once and then sharing the data with all of the read maps in the group. In the example illustrated here, four consecutive read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



Showing 1 to 5 of 5

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.



Device Id: 2

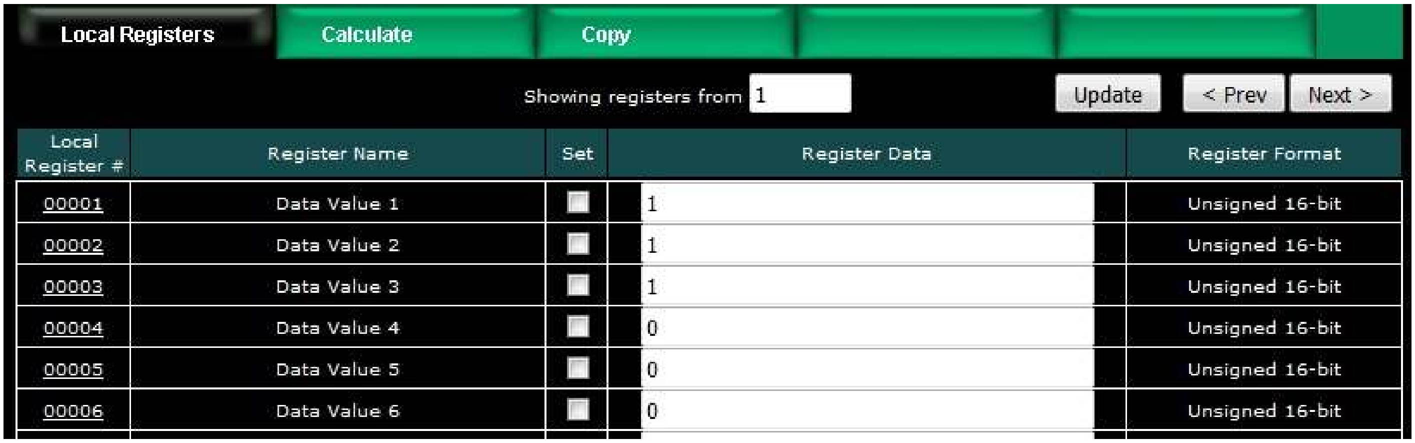
Address: 0001

Length: 20

MODBUS Point Type: 03: HOLDING REGISTER

```
40001: <00000>
40002: <00000>
40003: <00000>
40004: <00000>
40005: <00000>
40006: <00000>
40007: <00000>
40008: <00000>
40009: <00000>
40010: <00000>
40011: <00007>
40012: <00000>
40013: <00000>
40014: <00000>
40015: <00000>
40016: <00000>
40017: <00000>
40018: <00000>
40019: <00000>
40020: <00000>
```

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

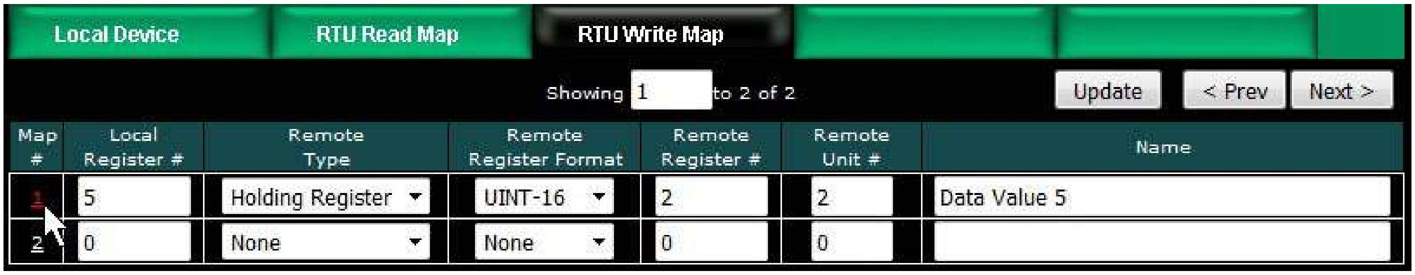


Showing registers from 1

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

5.3 Modbus RTU Master Write Maps

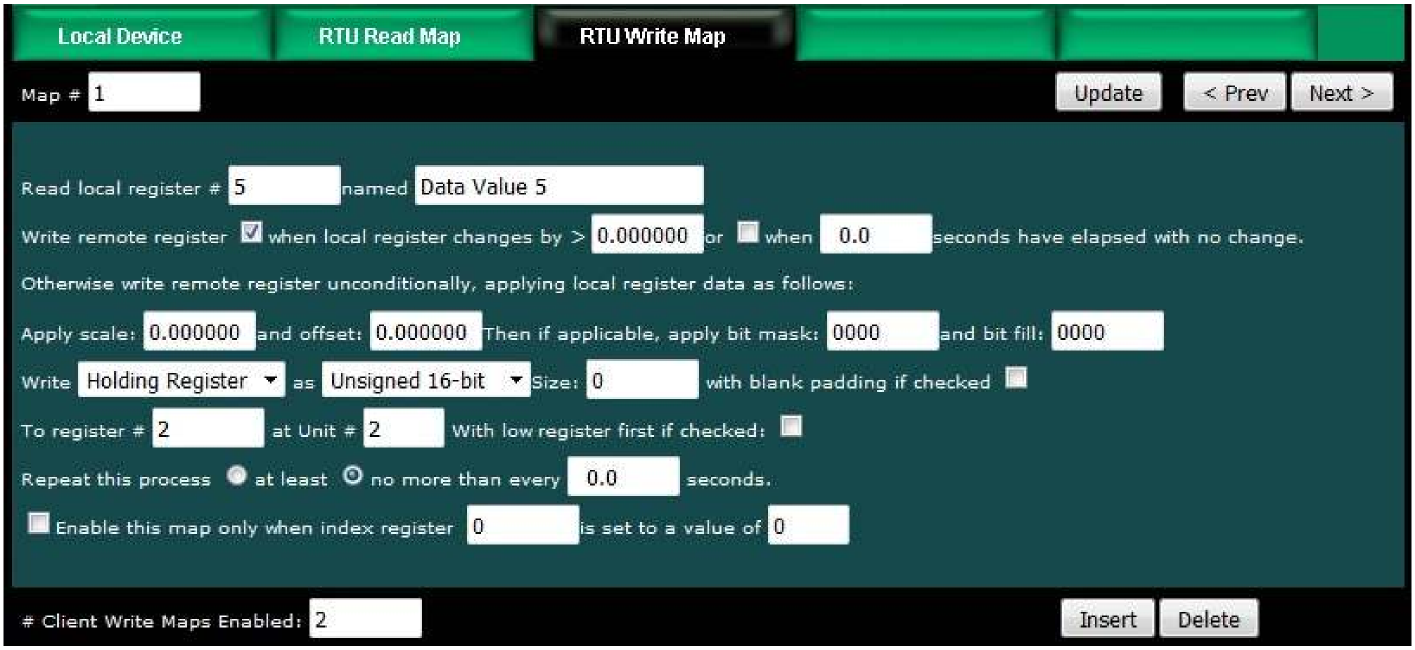
Getting the gateway to write registers to another Modbus device requires setting up a "Write Map" as shown here. Much of the Write Map is configured the same as a Read Map.



Showing 1 to 2 of 2

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

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 configuration parameters.



Map # 1

Read local register # 5 named Data Value 5

Write remote register ☑ when local register changes by > 0.000000 or □ when 0.0 seconds have elapsed with no change.

Otherwise write remote register unconditionally, applying local register data as follows:

Apply scale: 0.000000 and offset: 0.000000 Then if applicable, apply bit mask: 0000 and bit fill: 0000

Write Holding Register as Unsigned 16-bit Size: 0 with blank padding if checked □

To register # 2 at Unit # 2 With low register first if checked: □

Repeat this process ○ at least ● no more than every 0.0 seconds.

□ Enable this map only when index register 0 is set to a value of 0

Client Write Maps Enabled: 2

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 maps 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 maps are sequential, the results of all qualifying maps 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 minimum 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.

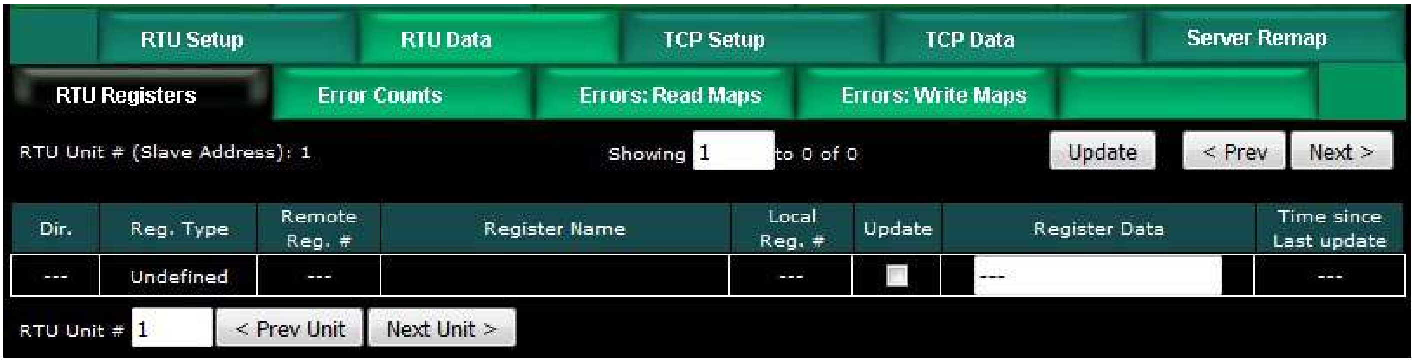
You have the option of making this Write Map conditional. If an index register number is provided and the Enable box is checked, then this write map will only be executed when the index register (local register) contains the value given. This allows multiple write maps to supply data to the same remote 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 map number shown in the "Map #" box. Insert will insert a new map before the map number shown, and is used for placing maps between existing maps. It is not necessary to use Insert to add maps to the bottom of the list or to define any map presently having zero for a source register or "none" for remote type.

Selecting "none" for remote type effectively deletes the map even though it will still appear in the list until deleted. Unused maps 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 maps enabled.

5.4 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.



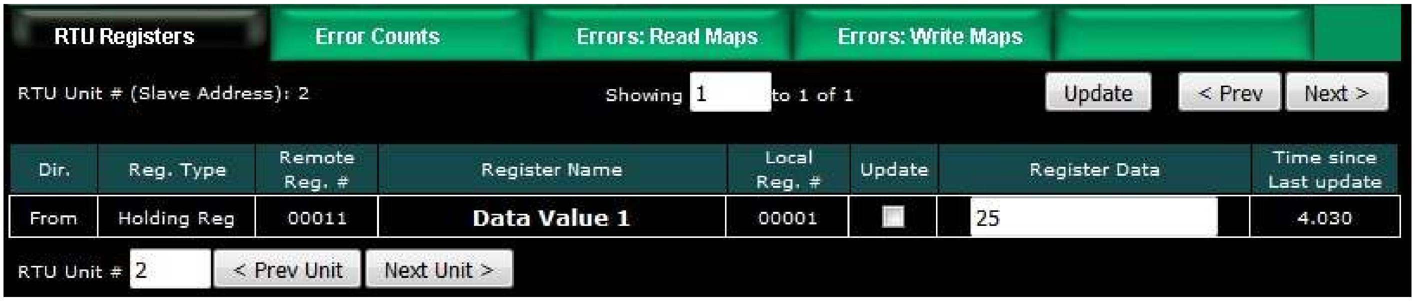
RTU Unit # (Slave Address): 1

Showing 1 to 0 of 0

Dir.	Reg. Type	Remote Reg. #	Register Name	Local Reg. #	Update	Register Data	Time since Last update
---	Undefined	---		---	□	---	---

RTU Unit # 1

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.



RTU Unit # (Slave Address): 2

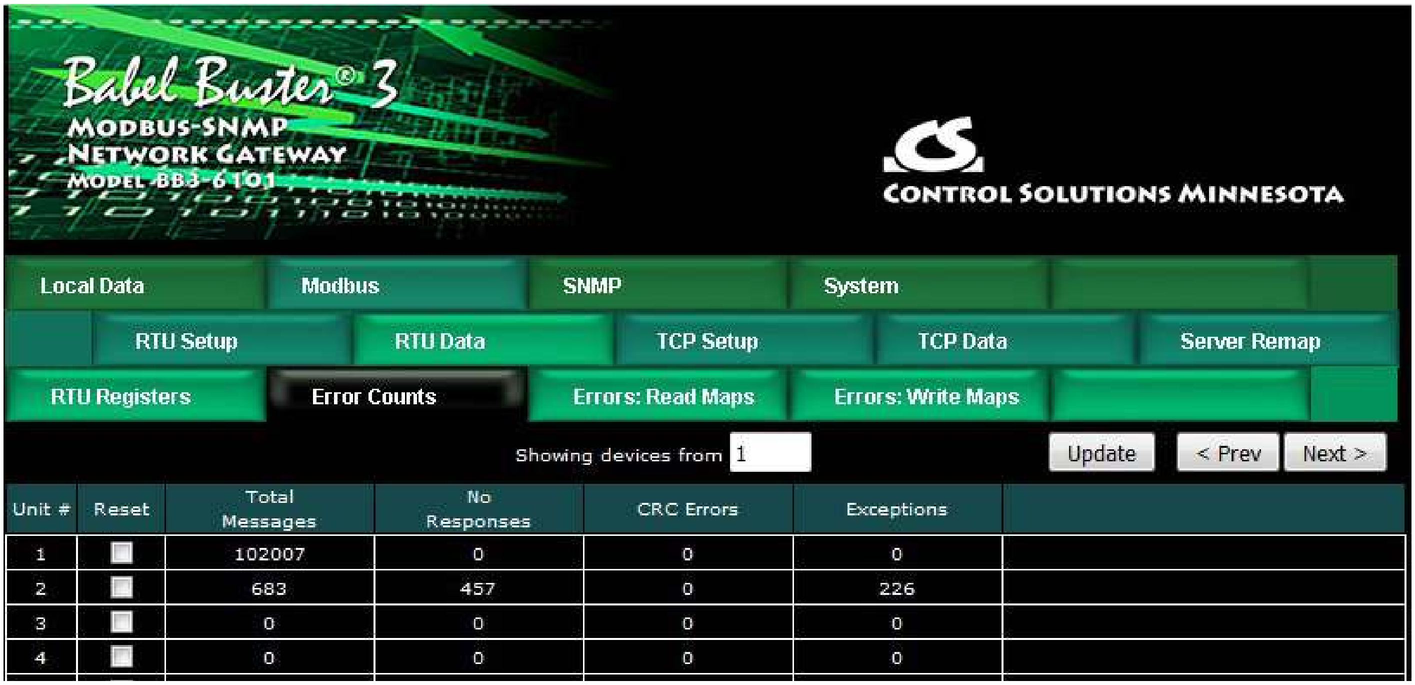
Showing 1 to 1 of 1

Dir.	Reg. Type	Remote Reg. #	Register Name	Local Reg. #	Update	Register Data	Time since Last update
From	Holding Reg	00011	Data Value 1	00001	□	25	4.030

RTU Unit # 2

5.5 Modbus RTU Errors

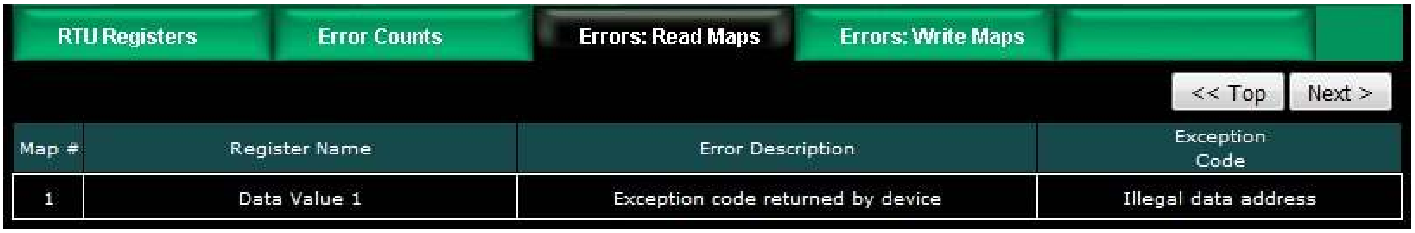
The Error Counts page shows a tabulation, by RTU slave, of all errors observed. In the example below, we can see that the RTU device at slave address 1 is running flawlessly while slave #2 has had some issues.



Showing devices from 1

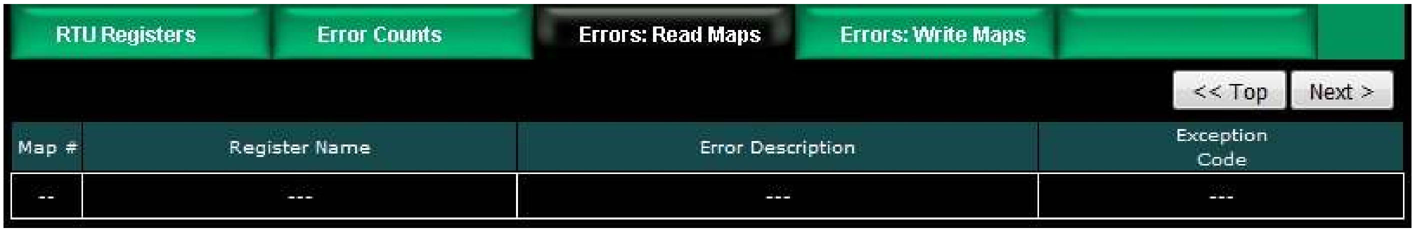
Unit #	Reset	Total Messages	No Responses	CRC Errors	Exceptions	
1	□	102007	0	0	0	
2	□	683	457	0	226	
3	□	0	0	0	0	
4	□	0	0	0	0	

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.



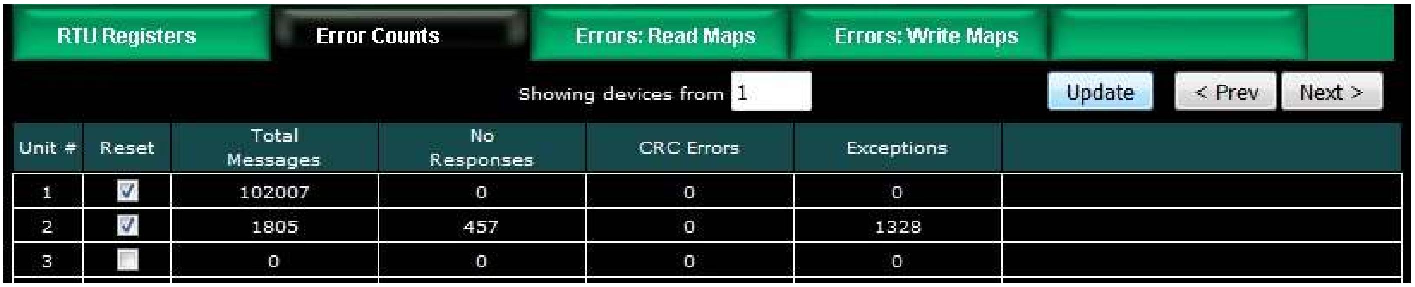
Map #	Register Name	Error Description	Exception Code
1	Data Value 1	Exception code returned by device	Illegal data address

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



Map #	Register Name	Error Description	Exception Code
--	---	---	---

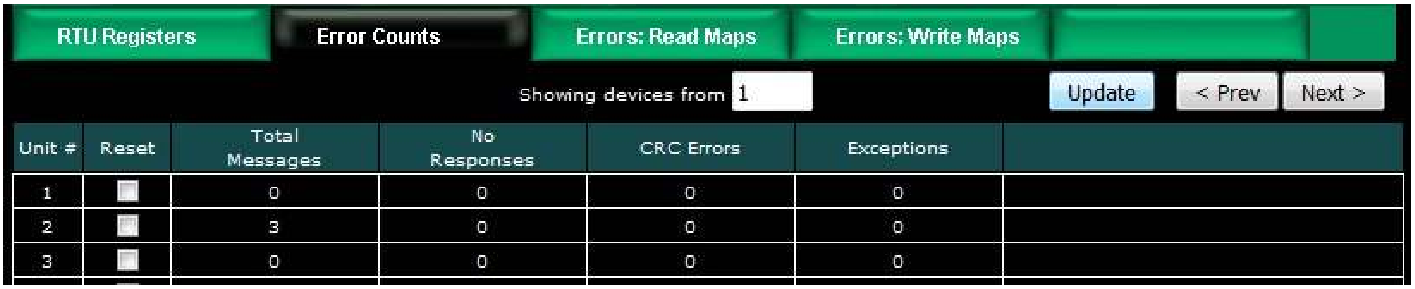
Once you have resolved problems, you can reset the error counts by checking the box in the Reset column and then clicking Update.



Showing devices from 1

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.



Showing devices from 1

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

Babel Buster® 3
MODEL BB3-6101
MODEL MX-61
NETWORK GATEWAY

CONTROL SOLUTIONS MINNESOTA

6. Configuring Gateway as a Modbus TCP Client

The Babel Buster BB3-6101/MX-61 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.

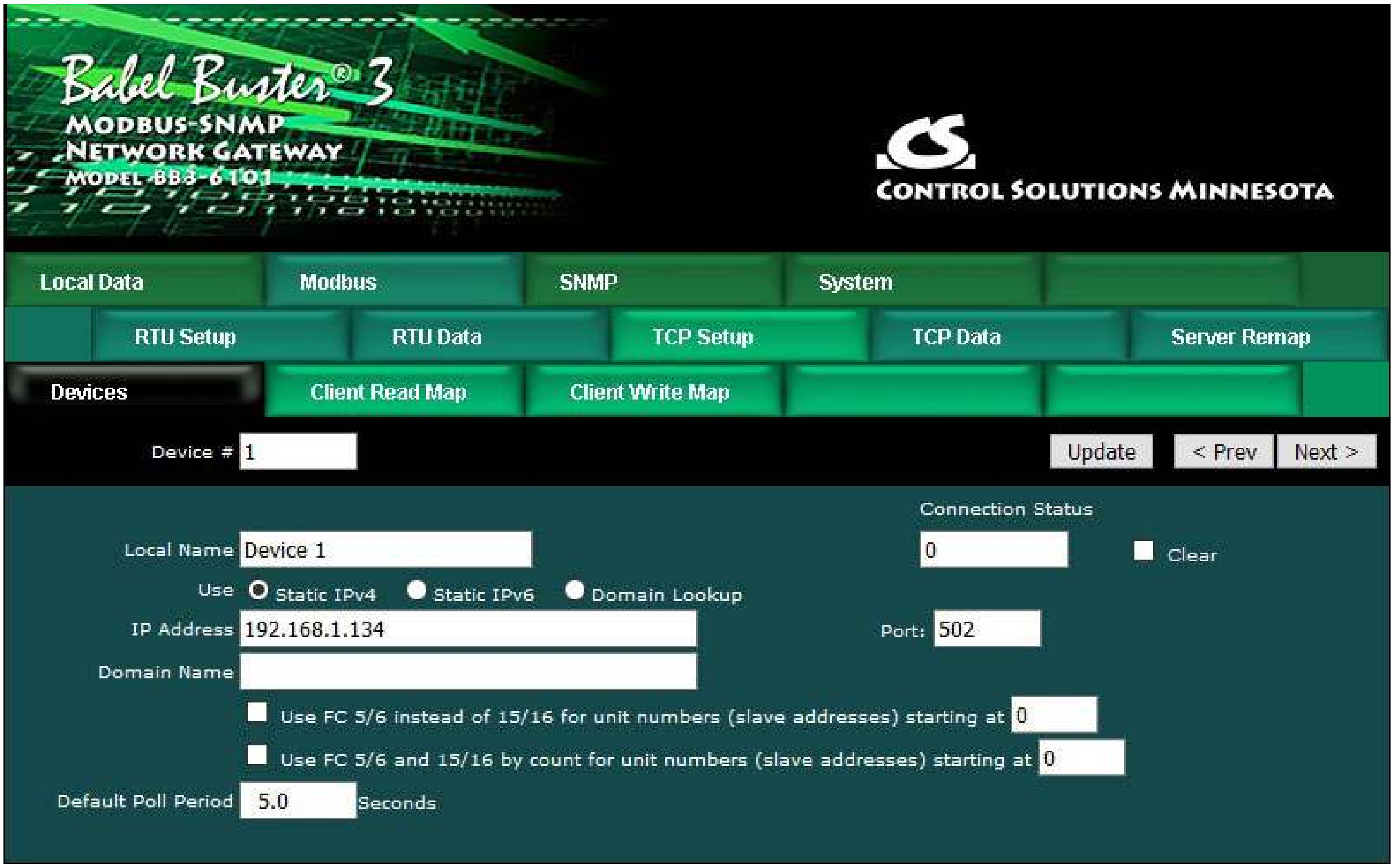
Data read from a remote device is stored in a local register when received. Data written to a remote device is taken from a local register when sent. The local registers are the same collection of registers that are accessible to other Modbus TCP clients as a collection of holding registers. These local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

The following section details the process of setting up read and write maps via the web interface. Once familiar with map content, you have the option of importing bulk configurations as a CSV file. The import function is found on the File Manager page, and details about the content and format of the CSV file are found in Appendix B.

6.1 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.



Device # 1

Connection Status 0 □ Clear

Local Name Device 1

Use ◉ Static IPv4 ○ Static IPv6 ○ Domain Lookup

IP Address 192.168.1.134 Port: 502

Domain Name

□ Use FC 5/6 instead of 15/16 for unit numbers (slave addresses) starting at 0

□ Use FC 5/6 and 15/16 by count for unit numbers (slave addresses) starting at 0

Default Poll Period 5.0 Seconds

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.) Unit numbers are entered on each individual read or write map.

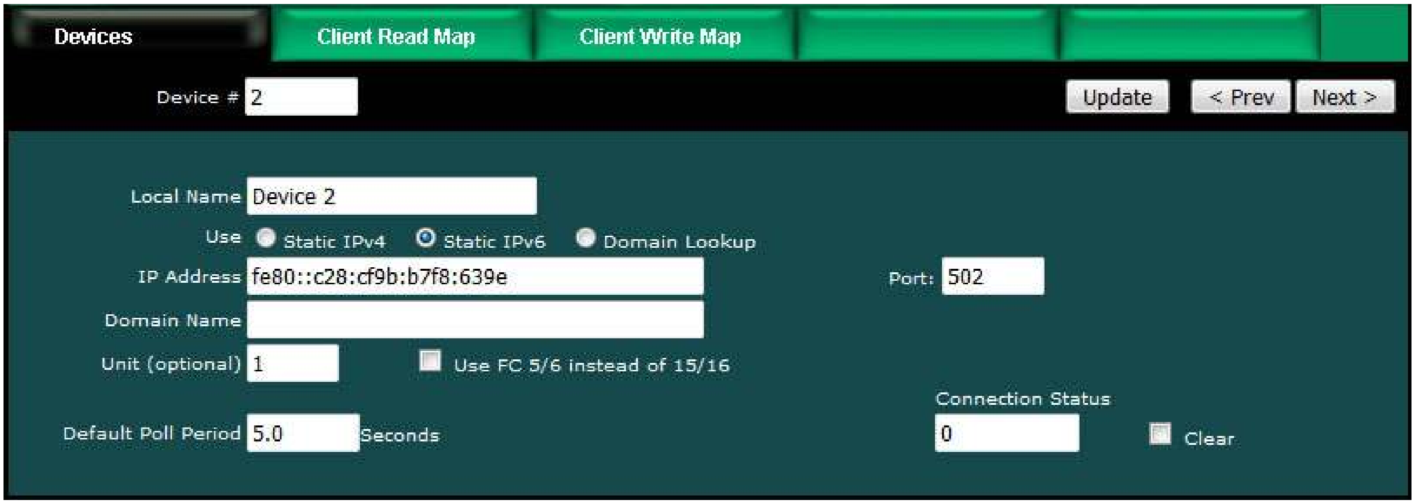
If your slave/server device only supports function codes 5 and 6 for writing coils and holding registers, check the Use FC 5/6 box. The default function codes are 15 and 16, which are most widely used. If you check the box, you should also enter a "starting at" unit # or slave address. This allows supporting both types of devices at the same time provided you assign slave addresses in two non-overlapping groups.

If your slave/server device is especially picky, it may require function codes 5 and 6 for single writes, but yet expect function codes 15 and 16 for multiple register writes. Select "Use FC 5/6 by count..." if this is the case, and provide a starting unit number.

The starting unit number will most often be simply "1" unless you have a mix of device types that all follow different rules for their function codes. Note that the selection of FC 5/6 only pertains to write maps. Reading registers will always use the same function codes for reading.

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.

The static IP address for the Modbus TCP device can be either IPv4 or IPv6. Along with selecting the desired IP version, be sure to enter the applicable IP address format. In addition, if IPv6 is used, be sure IPv6 is enabled on the Network configuration page.



Device # 2

Local Name Device 2

Use ○ Static IPv4 ● Static IPv6 ○ Domain Lookup

IP Address fe80::c28:cf9b:b7f8:639e

Port: 502

Domain Name

Unit (optional) 1

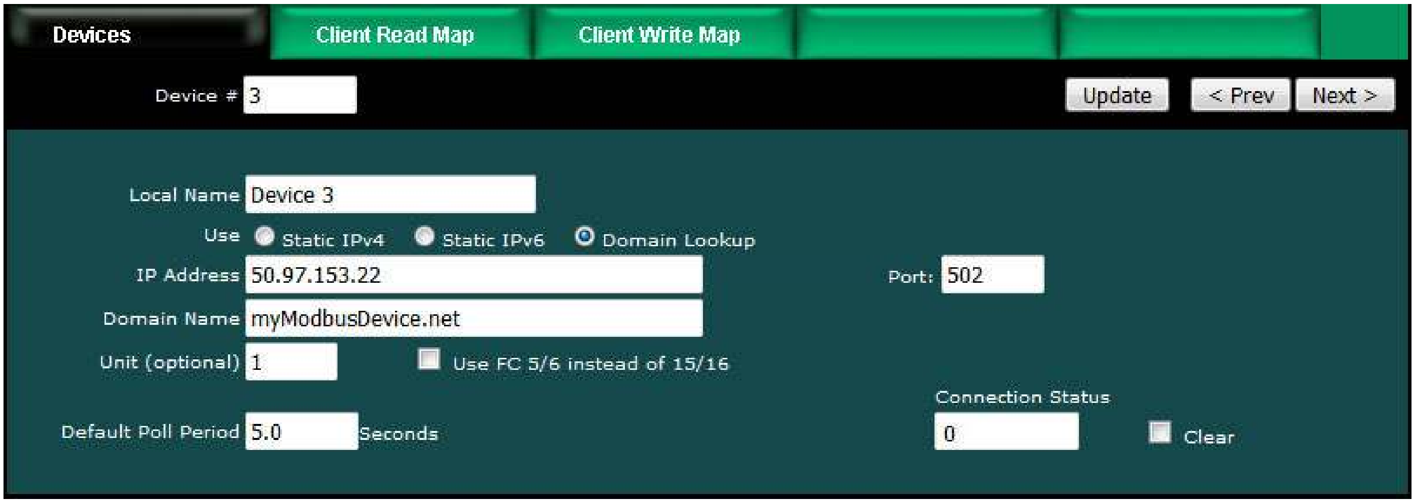
□ Use FC 5/6 instead of 15/16

Connection Status 0

□ Clear

Default Poll Period 5.0 Seconds

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. If found, the IP address provided by DNS will be displayed here (but not saved in the XML configuration file).



Device # 3

Local Name Device 3

Use ○ Static IPv4 ○ Static IPv6 ● Domain Lookup

IP Address 50.97.153.22

Port: 502

Domain Name myModbusDevice.net

Unit (optional) 1

□ Use FC 5/6 instead of 15/16

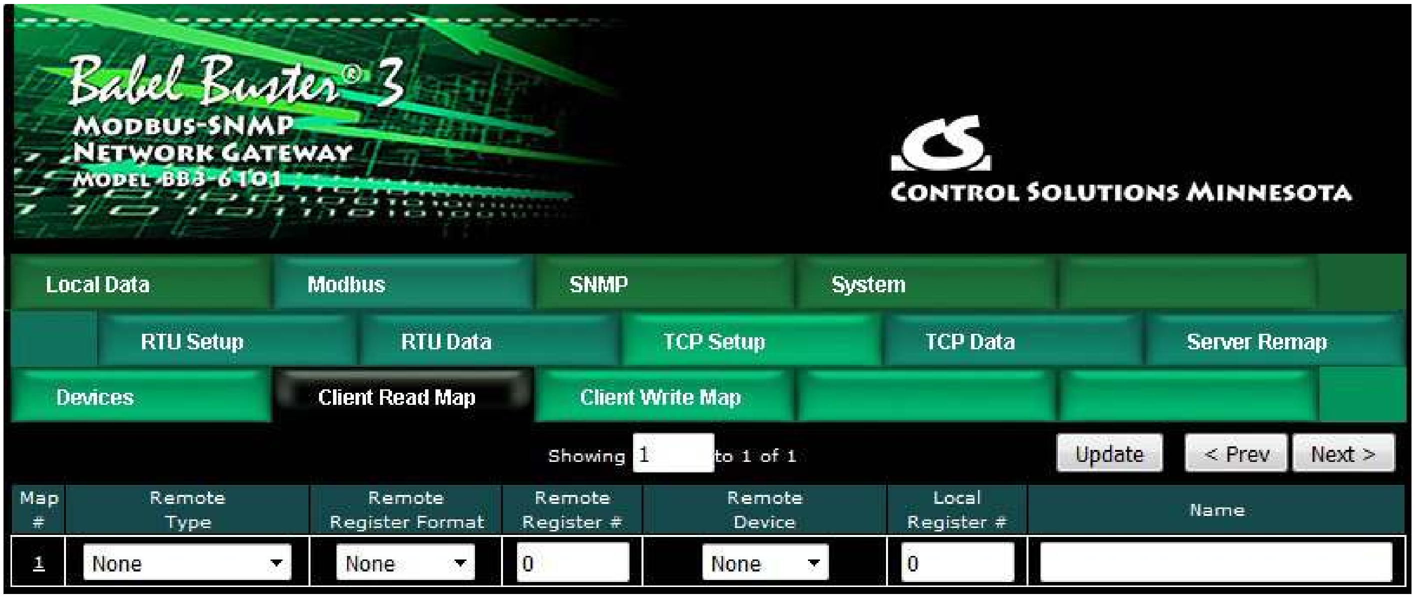
Connection Status 0

□ Clear

Default Poll Period 5.0 Seconds

6.2 Modbus TCP Client Read Maps

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

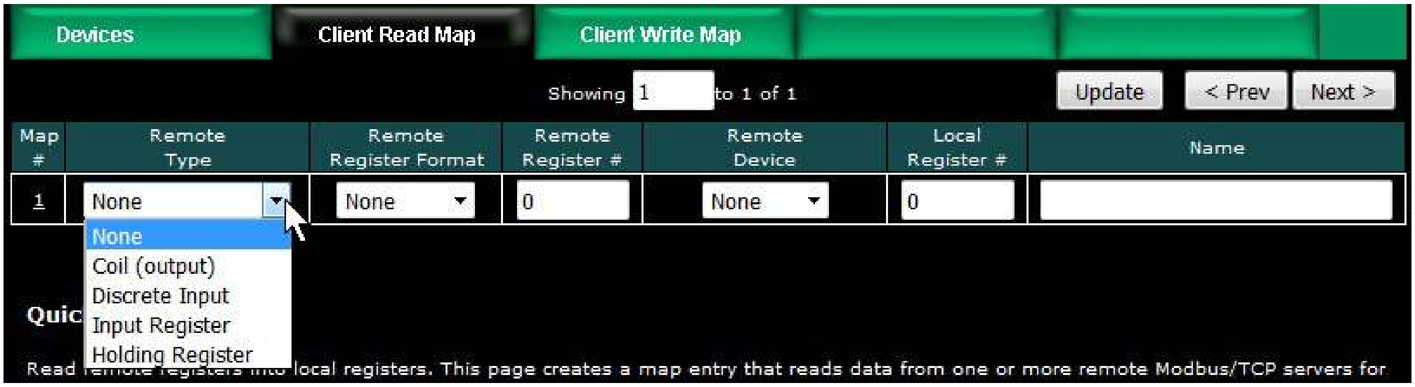


Showing 1 to 1 of 1

Map #	Remote Type	Remote Register Format	Remote Register #	Remote Device	Local Register #	Name
1	None	None	0	None	0	

Map 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.

Maps 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.



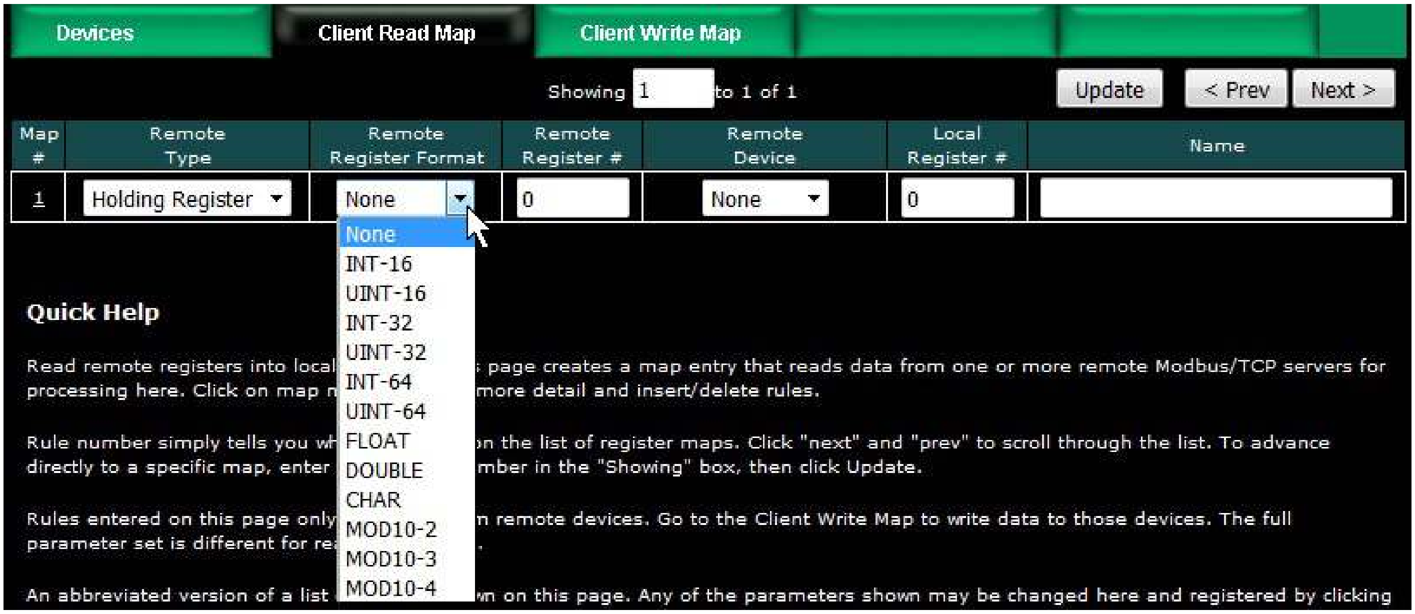
Showing 1 to 1 of 1

Map #	Remote Type	Remote Register Format	Remote Register #	Remote Device	Local Register #	Name
1	None	None	0	None	0	

Quic

Read remote registers into local registers. This page creates a map entry that reads data from one or more remote Modbus/TCP servers for

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



Map #	Remote Type	Remote Register Format	Remote Register #	Remote Device	Local Register #	Name
1	Holding Register	None	0	None	0	

Quick Help

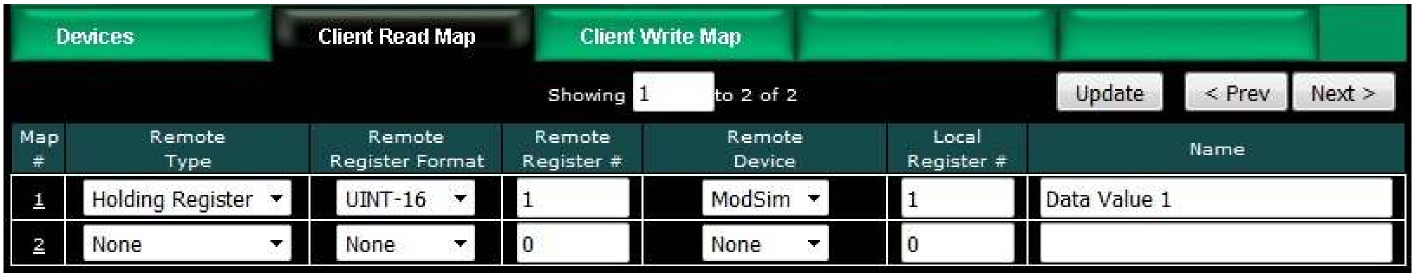
Read remote registers into local … page creates a map entry that reads data from one or more remote Modbus/TCP servers for processing here. Click on map n… more detail and insert/delete rules.

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

Rules entered on this page only … n remote devices. Go to the Client Write Map to write data to those devices. The full parameter set is different for re… .

An abbreviated version of a list … wn on this page. Any of the parameters shown may be changed here and registered by clicking

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.)

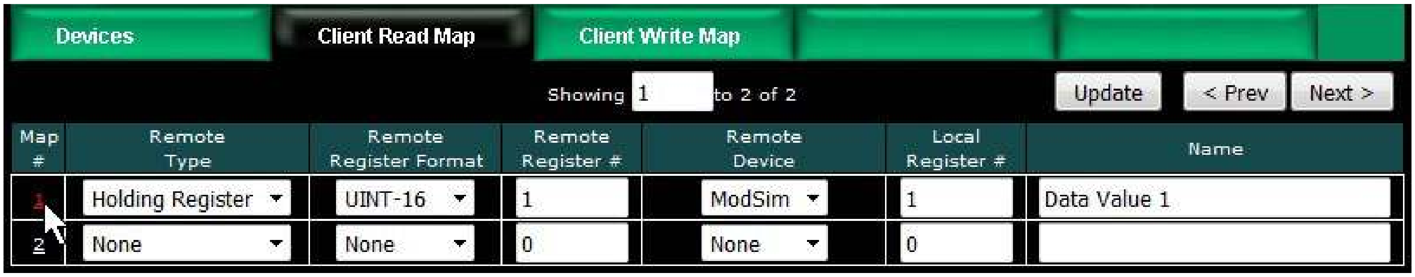


Map #	Remote Type	Remote Register Format	Remote Register #	Remote Device	Local Register #	Name
1	Holding Register	UINT-16	1	ModSim	1	Data Value 1
2	None	None	0	None	0	

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.

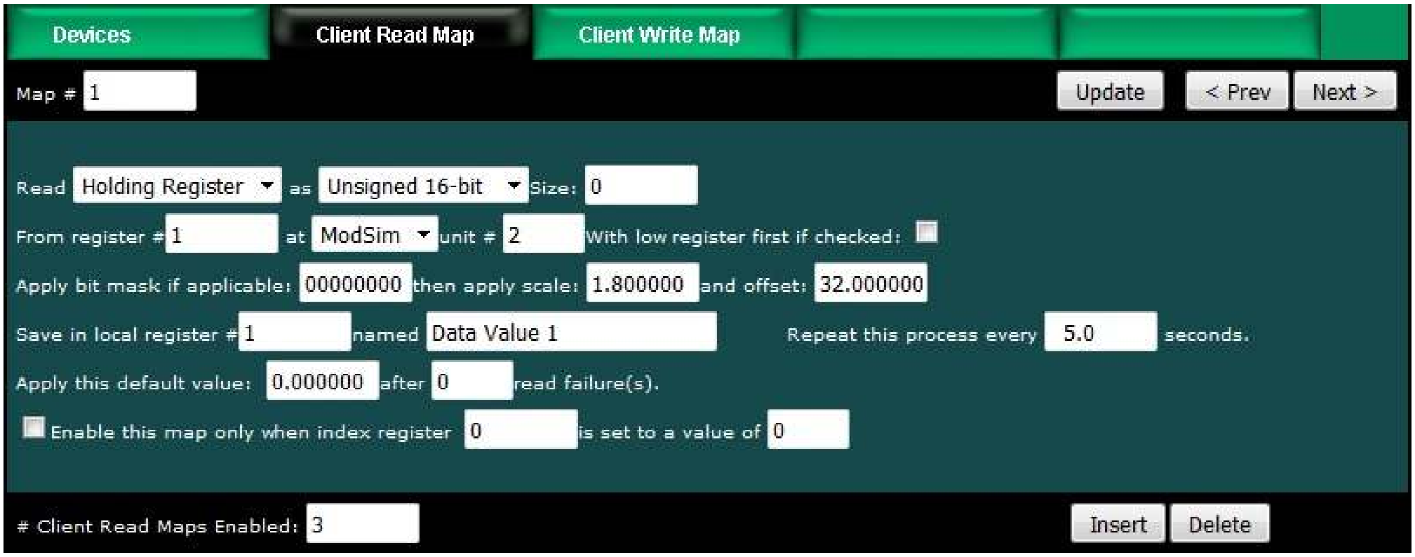
The Local Register is where data read from the remote Modbus TCP server will be stored locally in the Babel Buster BB3-6101/MX-61. If the local register data format does not match what you are reading from the Modbus device, the data will be converted automatically when it is read.



Showing 1 to 2 of 2

Map #	Remote Type	Remote Register Format	Remote Register #	Remote Device	Local Register #	Name
1	Holding Register	UINT-16	1	ModSim	1	Data Value 1
2	None	None	0	None	0	

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



Map # 1

Read Holding Register as Unsigned 16-bit Size: 0

From register # 1 at ModSim unit # 2 With low register first if checked: □

Apply bit mask if applicable: 00000000 then apply scale: 1.800000 and offset: 32.000000

Save in local register # 1 named Data Value 1 Repeat this process every 5.0 seconds.

Apply this default value: 0.000000 after 0 read failure(s).

□ Enable this map only when index register 0 is set to a value of 0

Client Read Maps Enabled: 3

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.

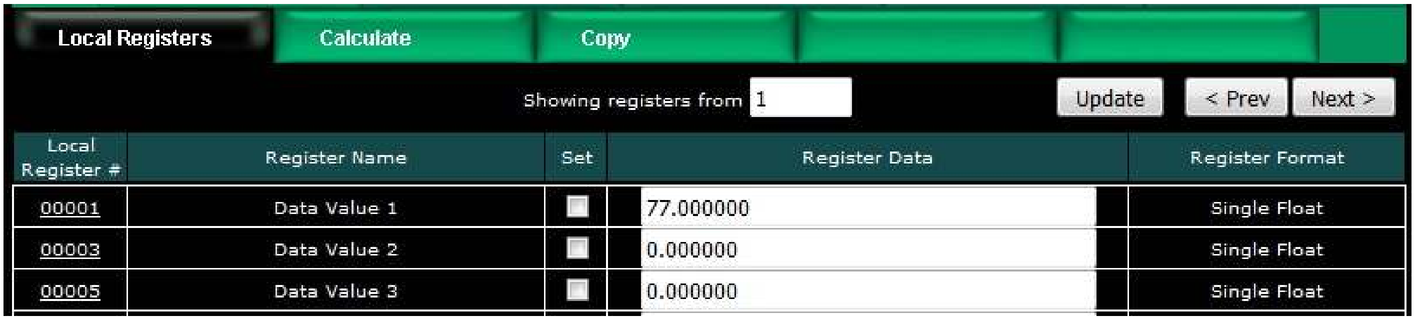
You have the option of making this Read Map conditional. If an index register number is provided and the Enable box is checked, then this read map will only be executed when the index register (local register) contains the value given. This allows multiple read maps to supply data to the same local register 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.

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

Selecting "none" for remote type effectively deletes the map even though it will still appear in the list until deleted. Unused maps 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 maps enabled.

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



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	77.000000	Single Float
00003	Data Value 2	□	0.000000	Single Float
00005	Data Value 3	□	0.000000	Single Float

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 multiplied 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).

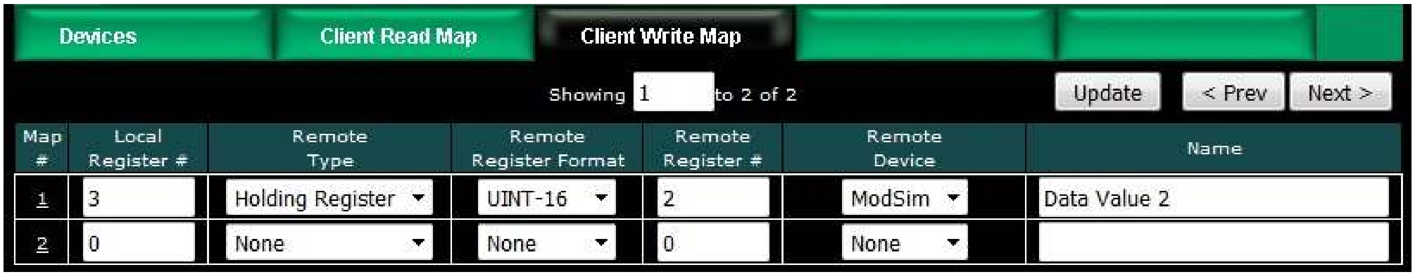
It is common for Modbus devices to pack a number of status bits into a single holding register. In order to do meaningful things based on a single bit, it is sometimes necessary to split that register into multiple local registers. Babel Buster supports this requirement by providing an optional bit mask.

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 5.2 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 map is selecting bit 0, the second selecting bit 1, and so on.

6.3 Modbus TCP Client Write Maps

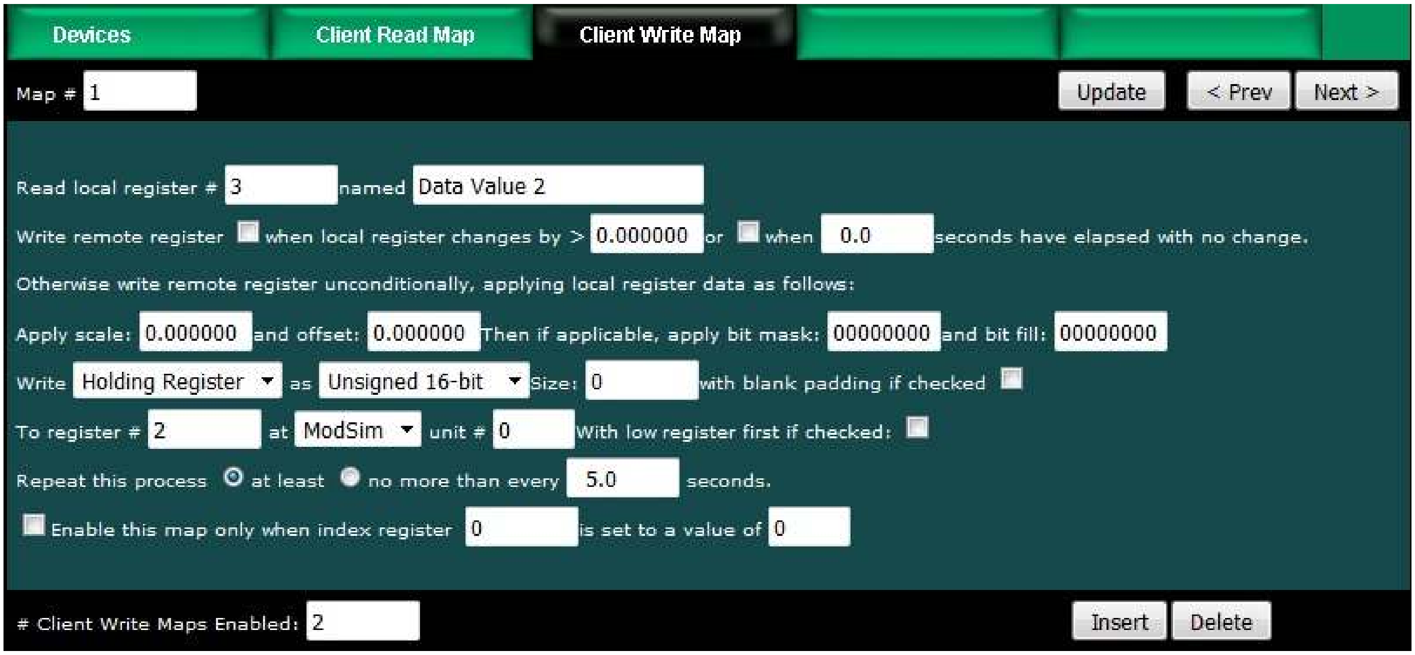
Getting the gateway to write registers to another Modbus device requires setting up a "Write Map" as shown here. Much of the Write Map is configured the same as a Read Map.



Showing 1 to 2 of 2

Map #	Local Register #	Remote Type	Remote Register Format	Remote Register #	Remote Device	Name
1	3	Holding Register	UINT-16	2	ModSim	Data Value 2
2	0	None	None	0	None	

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.



Map # 1

Read local register # 3 named Data Value 2

Write remote register □ when local register changes by > 0.000000 or □ when 0.0 seconds have elapsed with no change.

Otherwise write remote register unconditionally, applying local register data as follows:

Apply scale: 0.000000 and offset: 0.000000 Then if applicable, apply bit mask: 00000000 and bit fill: 00000000

Write Holding Register ▾ as Unsigned 16-bit ▾ Size: 0 with blank padding if checked □

To register # 2 at ModSim ▾ unit # 0 With low register first if checked: □

Repeat this process ◉ at least ○ no more than every 5.0 seconds.

□ Enable this map only when index register 0 is set to a value of 0

Client Write Maps Enabled: 2

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 maps 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 maps are sequential, the results of all qualifying maps 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.

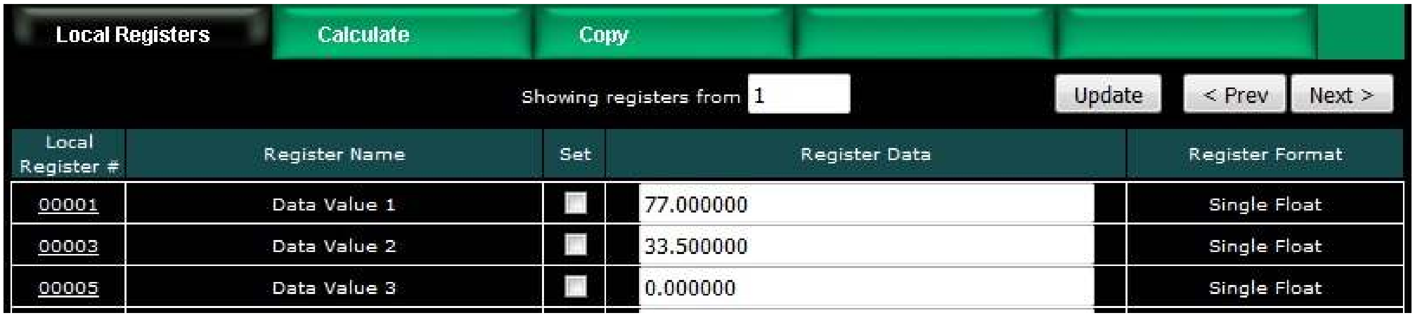
Modbus protocol treats all holding registers as strictly 16-bit registers. To accommodate 32-bit or longer data, Modbus devices use multiple consecutive "registers" to hold the data. There is no standardization of whether the least significant part of the data comes first or last. Therefore, Babel Buster 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 minimum 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 map number shown in the "Map #" box. Insert will insert a new map before the map number shown, and is used for placing maps between existing maps. It is not necessary to use Insert to add maps to the bottom of the list or to define any map presently having zero for a source register or "none" for remote type.

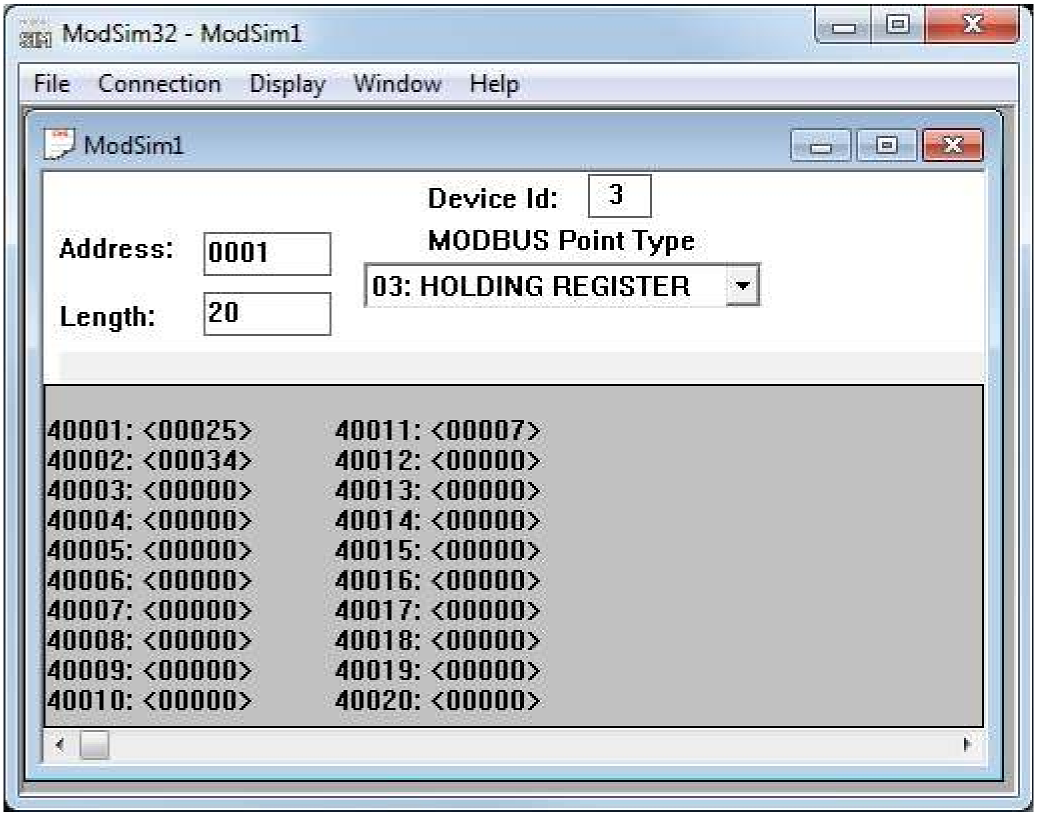
Selecting "none" for remote type effectively deletes the map even though it will still appear in the list until deleted. Unused maps at the end of the list will always show none as the type. If you wish to prevent these from being displayed, reduce the number of maps enabled.



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	77.000000	Single Float
00003	Data Value 2	□	33.500000	Single Float
00005	Data Value 3	□	0.000000	Single Float

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.



Device Id: 3

Address: 0001

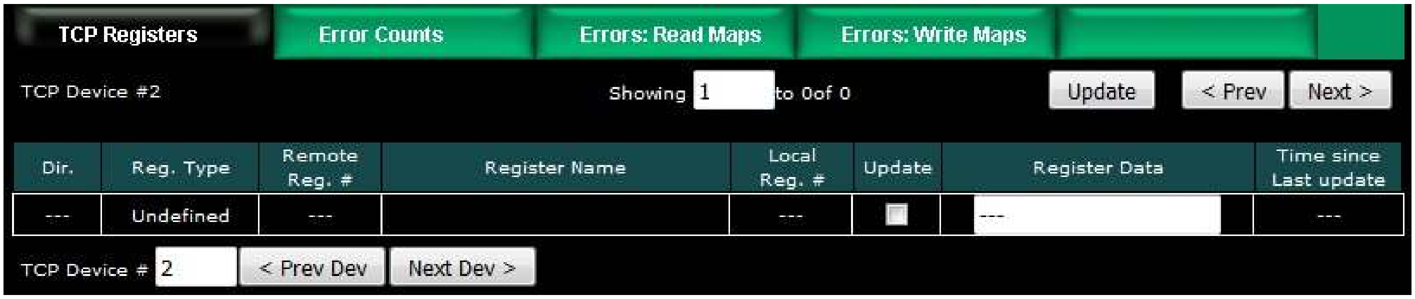
MODBUS Point Type: 03: HOLDING REGISTER

Length: 20

```
40001: <00025>     40011: <00007>
40002: <00034>     40012: <00000>
40003: <00000>     40013: <00000>
40004: <00000>     40014: <00000>
40005: <00000>     40015: <00000>
40006: <00000>     40016: <00000>
40007: <00000>     40017: <00000>
40008: <00000>     40018: <00000>
40009: <00000>     40019: <00000>
40010: <00000>     40020: <00000>
```

6.4 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.



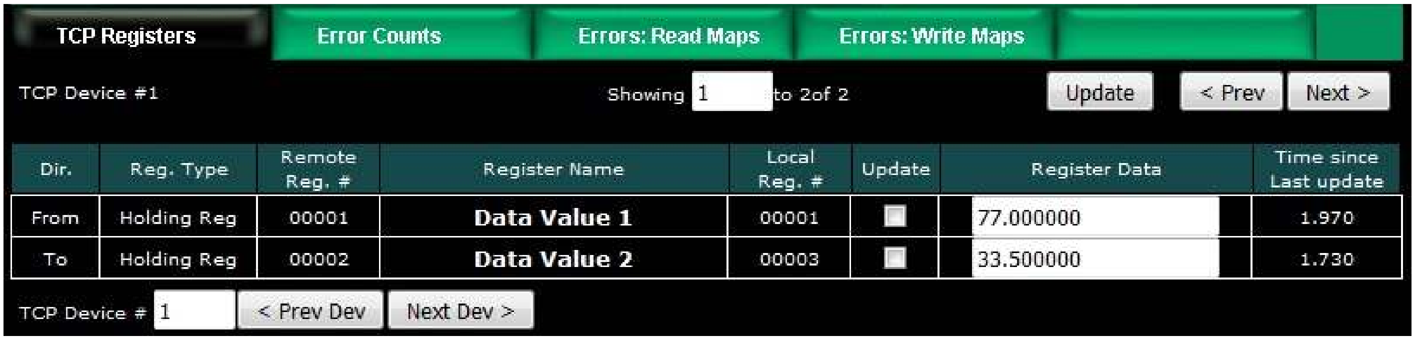
TCP Device #2

Showing 1 to 0of 0

Dir.	Reg. Type	Remote Reg. #	Register Name	Local Reg. #	Update	Register Data	Time since Last update
---	Undefined	---		---	□	---	---

TCP Device # 2

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.



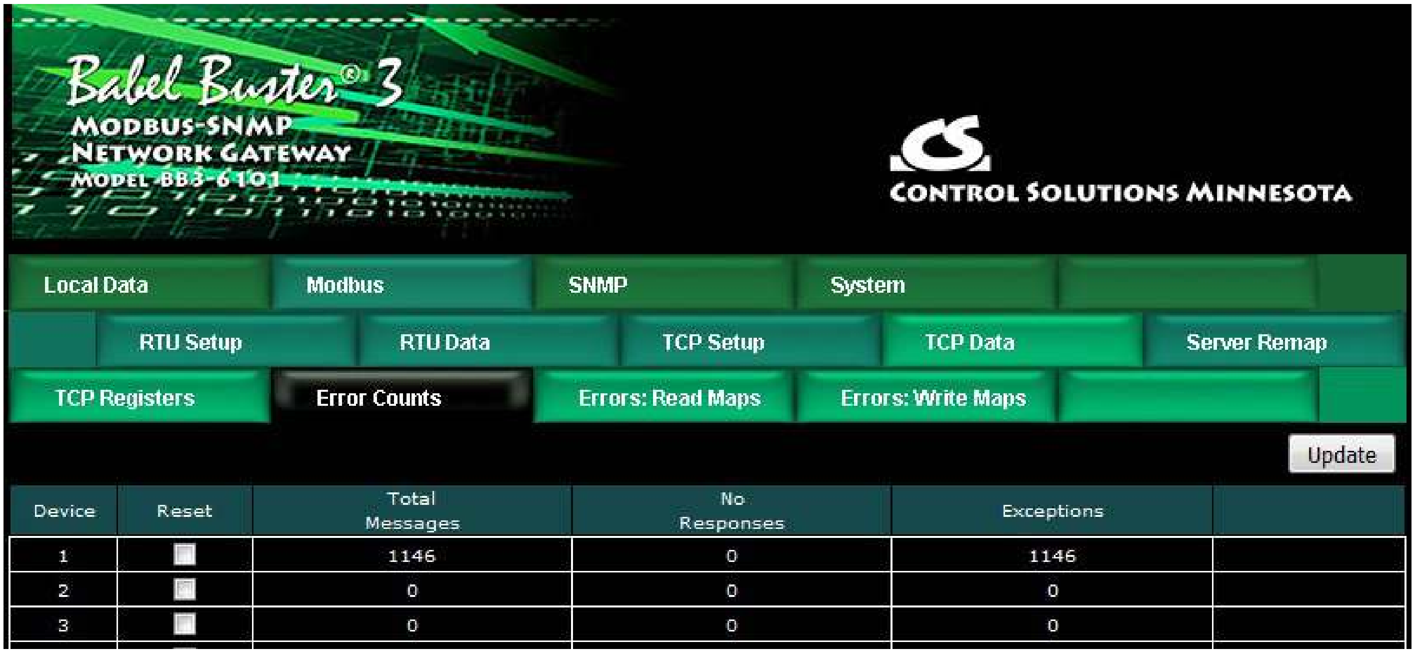
TCP Device #1 Showing 1 to 2of 2

Dir.	Reg. Type	Remote Reg. #	Register Name	Local Reg. #	Update	Register Data	Time since Last update
From	Holding Reg	00001	**Data Value 1**	00001	□	77.000000	1.970
To	Holding Reg	00002	**Data Value 2**	00003	□	33.500000	1.730

TCP Device # 1

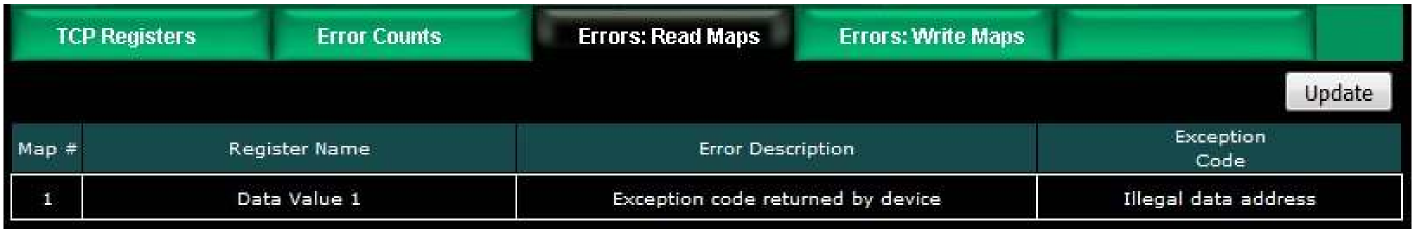
6.5 Modbus TCP Errors

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 exception errors as messages sent.



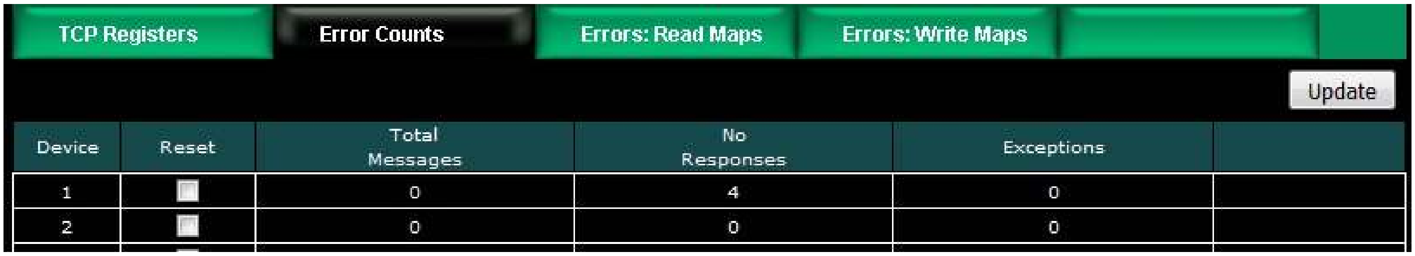
Device	Reset	Total Messages	No Responses	Exceptions	
1	□	1146	0	1146	
2	□	0	0	0	
3	□	0	0	0	

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.



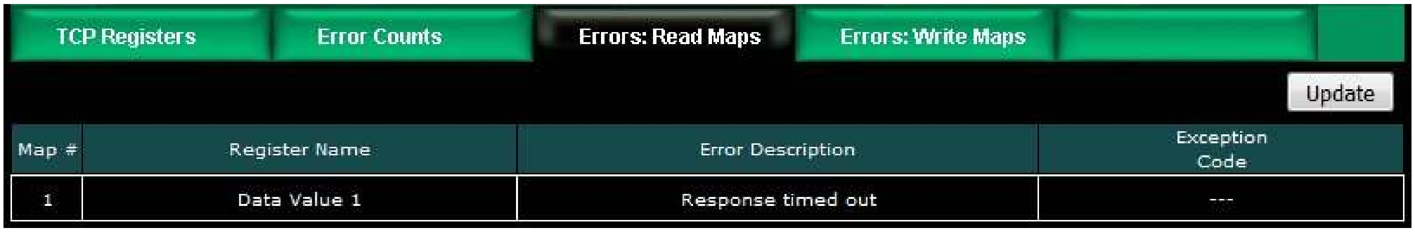
Map #	Register Name	Error Description	Exception Code
1	Data Value 1	Exception code returned by device	Illegal data address

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.



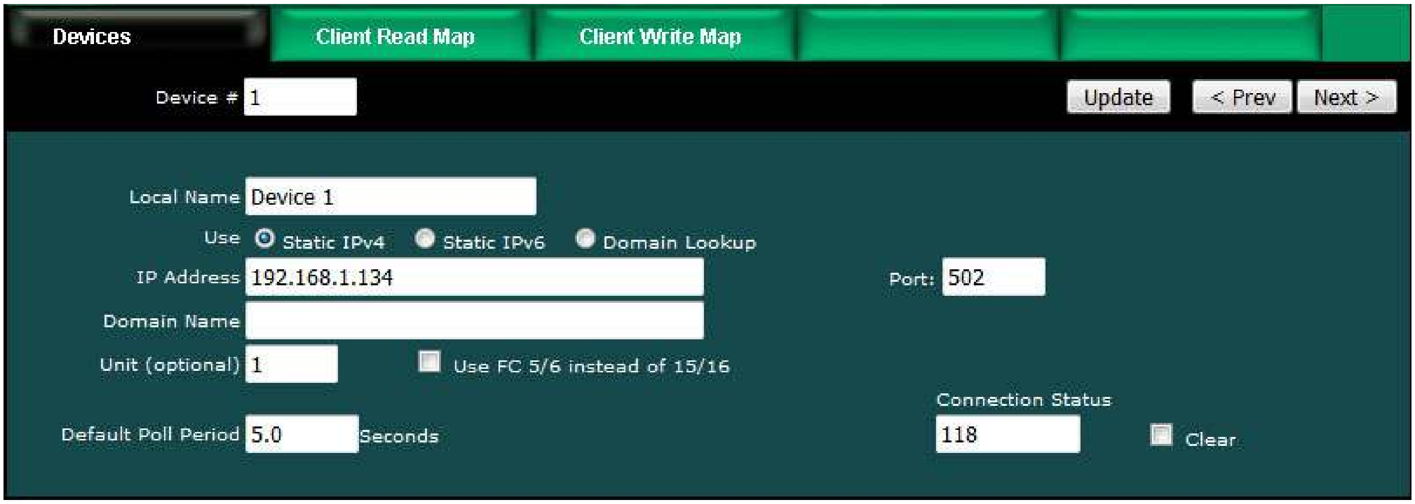
Device	Reset	Total Messages	No Responses	Exceptions
1	□	0	4	0
2	□	0	0	0

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.



Map #	Register Name	Error Description	Exception Code
1	Data Value 1	Response timed out	---

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



Device # 1

Local Name Device 1

Use ● Static IPv4 ○ Static IPv6 ○ Domain Lookup

IP Address 192.168.1.134 Port: 502

Domain Name

Unit (optional) 1 □ Use FC 5/6 instead of 15/16

Default Poll Period 5.0 Seconds

Connection Status 118 □ Clear

Connection status codes you may see include:

5 = Connection attempt timed out, unable to establish connection (usually means remote device not connected or not reachable)
104 = Connection reset by peer
111 = Connection refused
113 = Connection aborted
114 = Network is unreachable
115 = Network interface not configured
116 = Connection timed out
118 = Host is unreachable
125 = Address not available
205 = DNS error

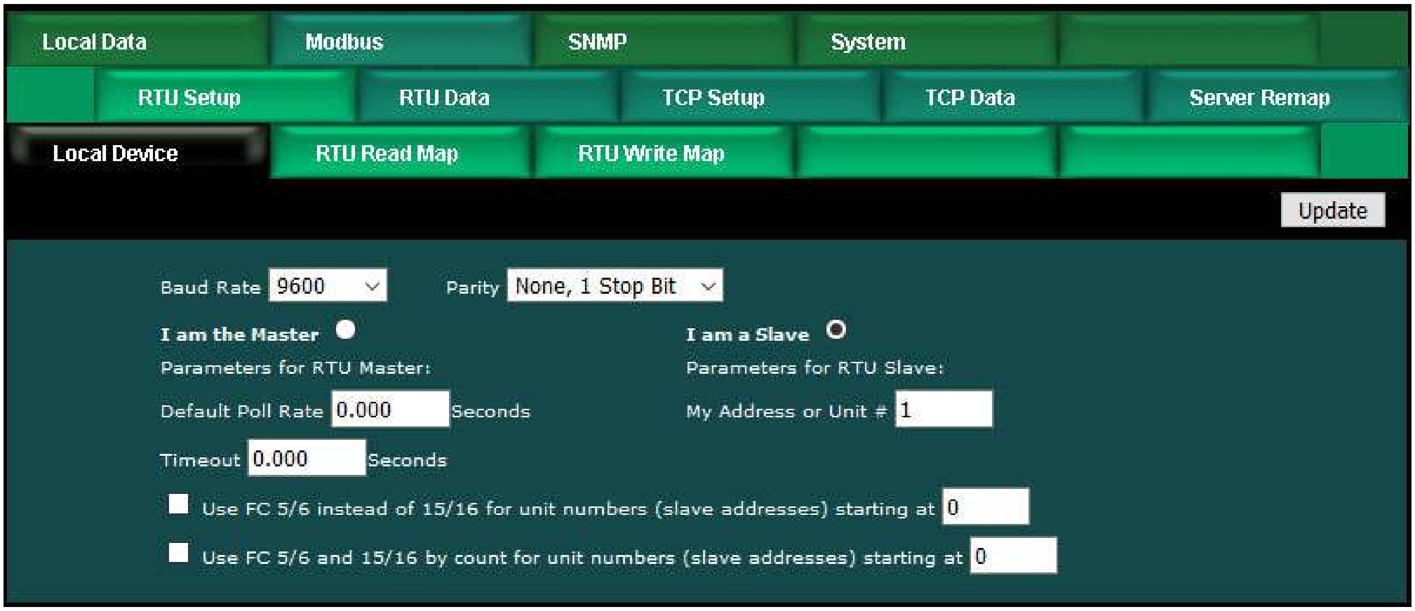
Babel Buster® 3
MODEL BB3-6101
MODEL MX-61
NETWORK GATEWAY

CONTROL SOLUTIONS MINNESOTA

7. Configuring Gateway as a Modbus RTU Slave

7.1 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.



Baud Rate 9600 Parity None, 1 Stop Bit

I am the Master ○ I am a Slave ●

Parameters for RTU Master: Parameters for RTU Slave:

Default Poll Rate 0.000 Seconds My Address or Unit # 1

Timeout 0.000 Seconds

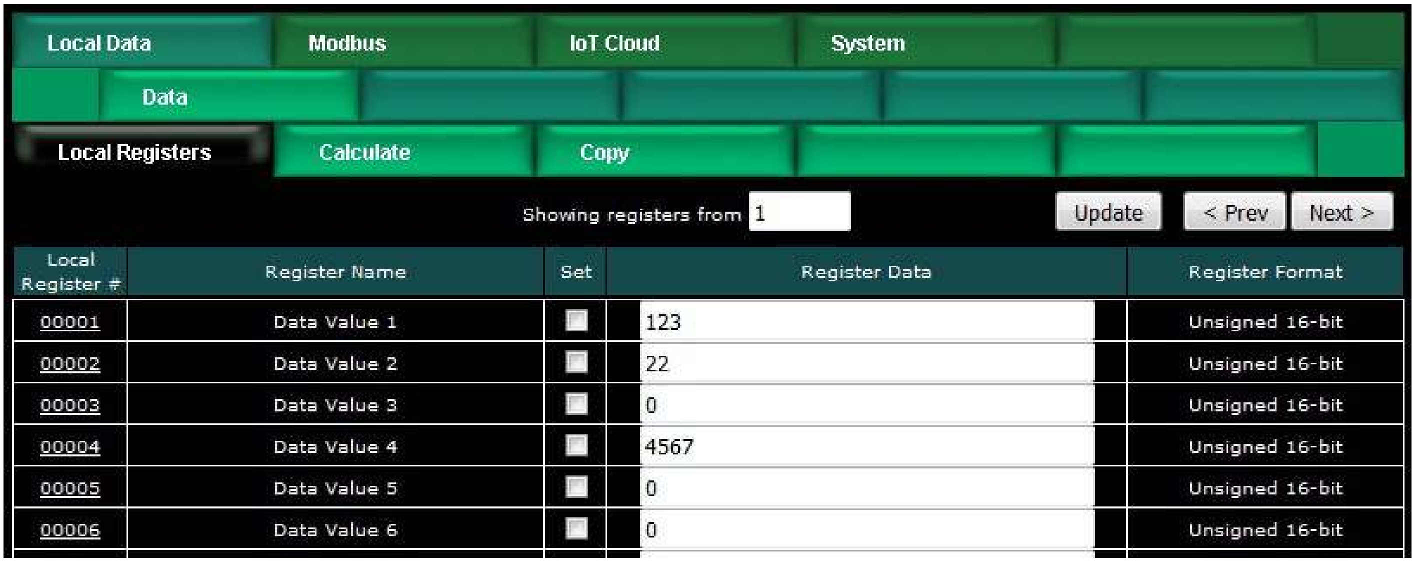
□ Use FC 5/6 instead of 15/16 for unit numbers (slave addresses) starting at 0

□ Use FC 5/6 and 15/16 by count for unit numbers (slave addresses) starting at 0

7.2 Modbus RTU Slave Register Map

The local registers in the Babel Buster BB3-6101/MX-61 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.

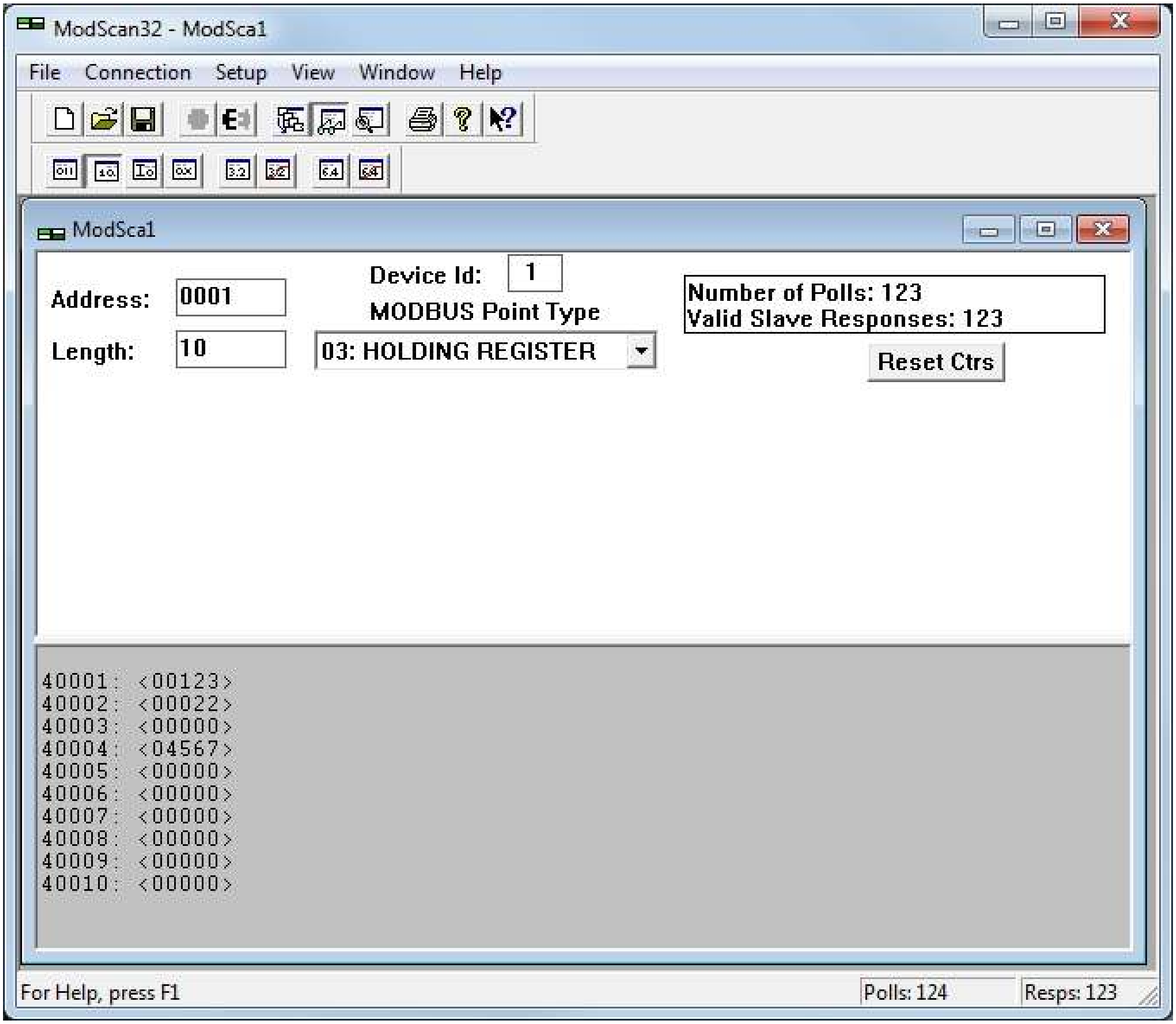
The register numbers that the remote Modbus RTU master should read or write are simply those shown in the first column on the Local Registers page.



Showing registers from 1

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.

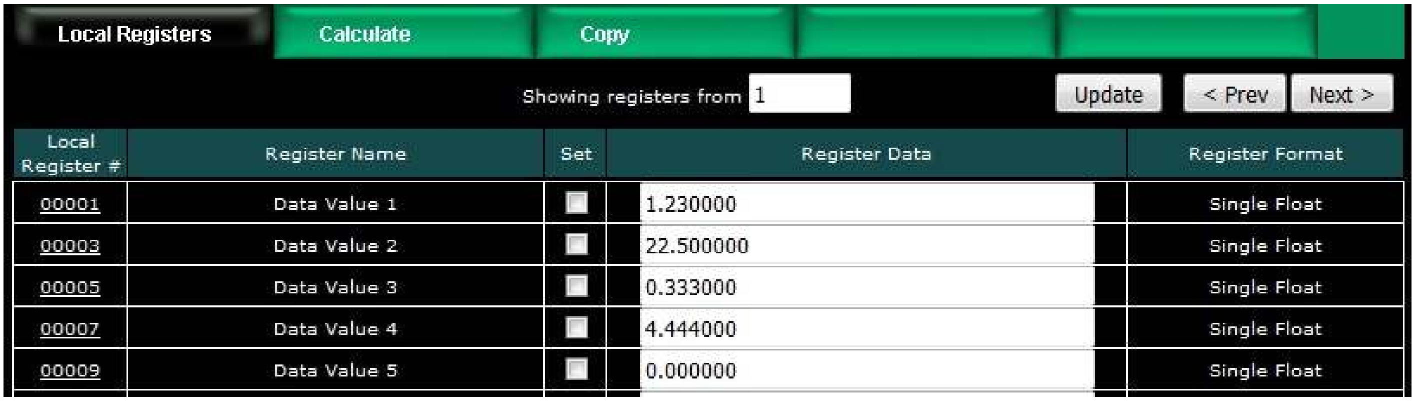


Address: 0001
Length: 10
Device Id: 1
MODBUS Point Type
03: HOLDING REGISTER

Number of Polls: 123
Valid Slave Responses: 123

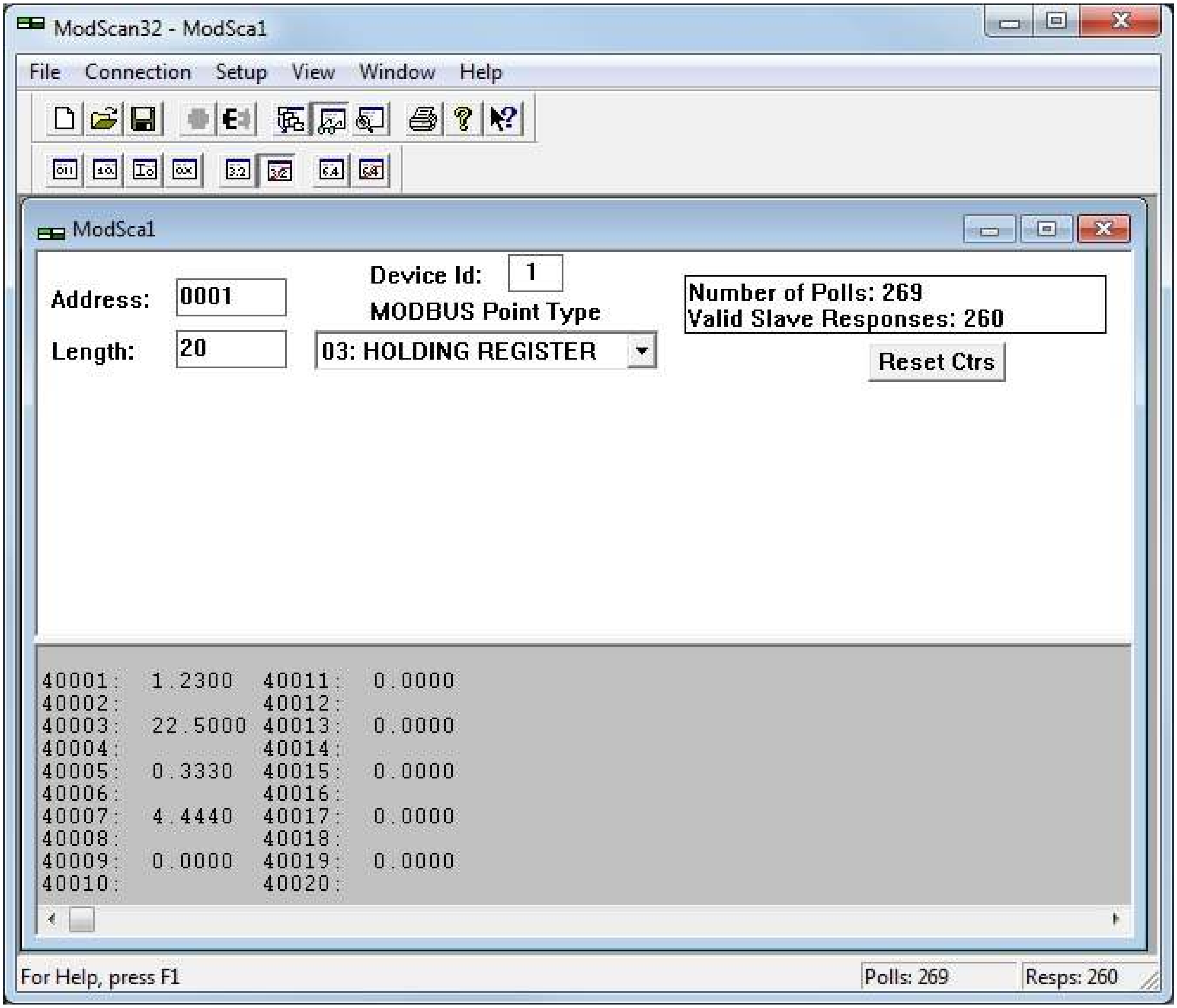
```
40001: <00123>
40002: <00022>
40003: <00000>
40004: <04567>
40005: <00000>
40006: <00000>
40007: <00000>
40008: <00000>
40009: <00000>
40010: <00000>
```

For Help, press F1 Polls: 124 Resps: 123



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 register #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).



Address: 0001
Length: 20
Device Id: 1
MODBUS Point Type
03: HOLDING REGISTER

Number of Polls: 269
Valid Slave Responses: 260

```
40001:  1.2300  40011:  0.0000
40002:          40012:
40003:  22.5000 40013:  0.0000
40004:          40014:
40005:  0.3330  40015:  0.0000
40006:          40016:
40007:  4.4440  40017:  0.0000
40008:          40018:
40009:  0.0000  40019:  0.0000
40010:          40020:
```

7.3 Modbus RTU Slave Diagnostic

The Babel Buster 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".

7.4 Modbus RTU Slave Register Remapping

The Server Remapping described in section 8.4 also applies to Modbus RTU. If Modbus RTU is configured as a slave and remapping is configured, then the register map is replaced with the map indicated on the Server Remap page.

Babel Buster® 3
MODEL BB3-6101
MODEL MX-61
NETWORK GATEWAY

CONTROL SOLUTIONS MINNESOTA

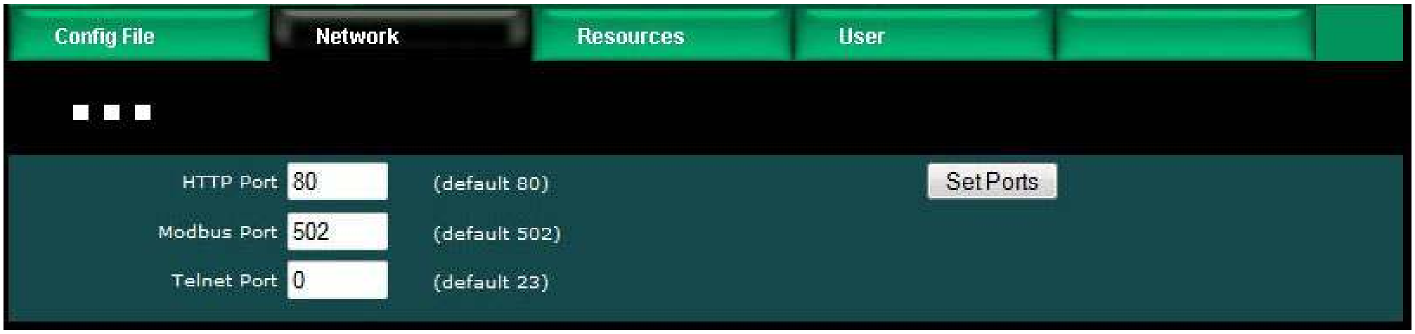
8. Configuring Gateway as a Modbus TCP Server

8.1 Modbus TCP Device Configuration

There is really little to do to configure the Babel Buster BB3-6101/MX-61 to be a Modbus TCP server. The gateway needs an IP address and you have already set that via the Network page. 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.

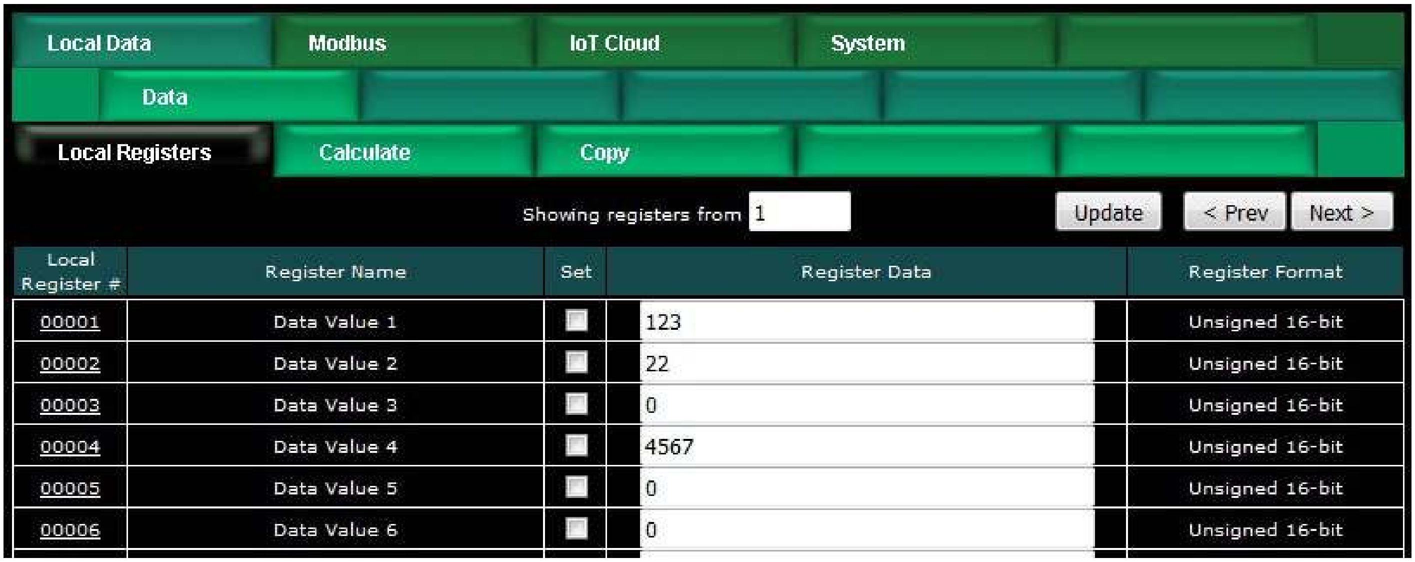


HTTP Port 80 (default 80)
Modbus Port 502 (default 502)
Telnet Port 0 (default 23)

8.2 Modbus TCP Server Register Map

The local registers in the Babel Buster BB3-6101/MX-61 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.

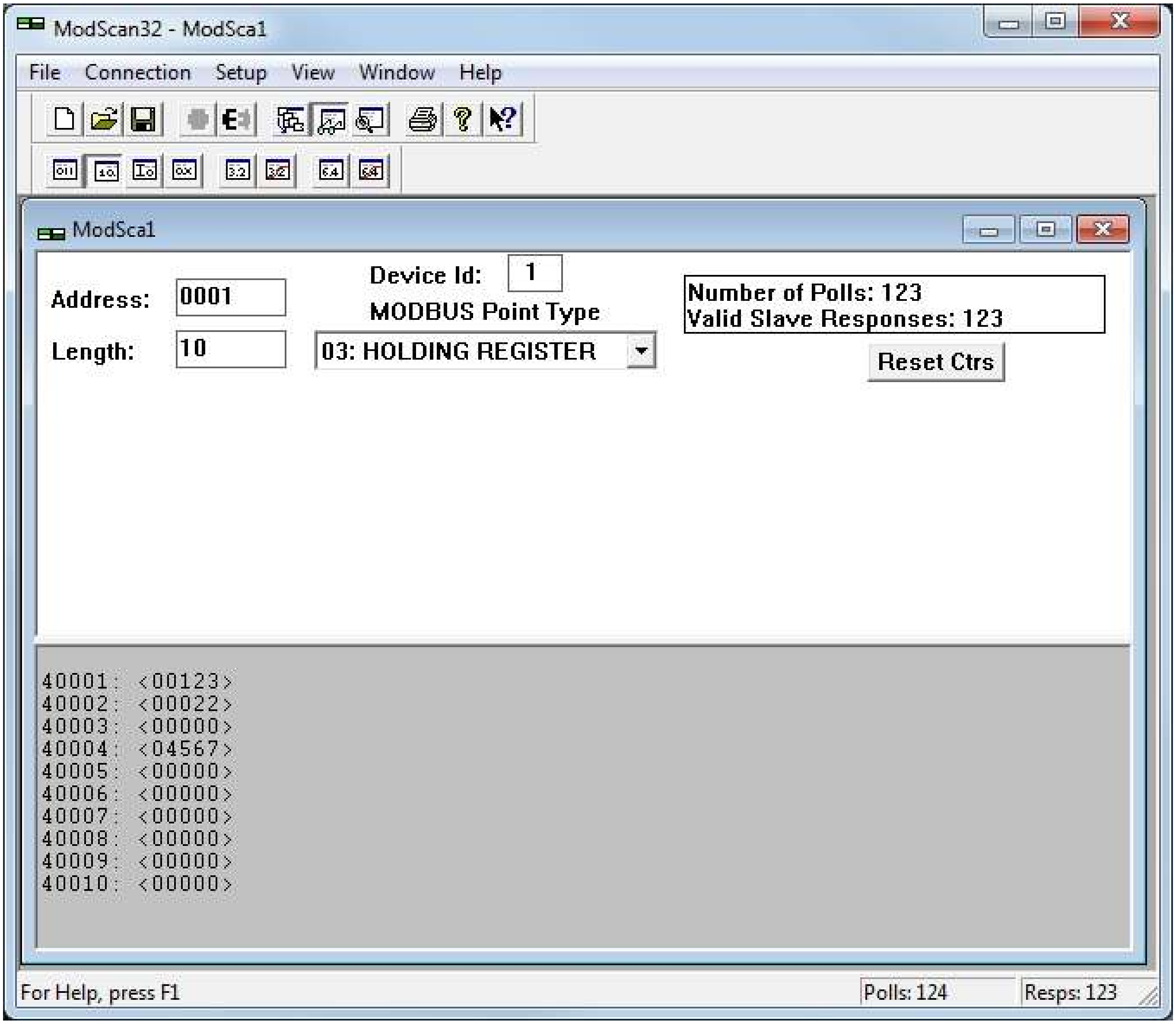
The register numbers that the remote Modbus TCP client should read or write are simply those shown in the first column on the Local Registers page.



Showing registers from 1

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.

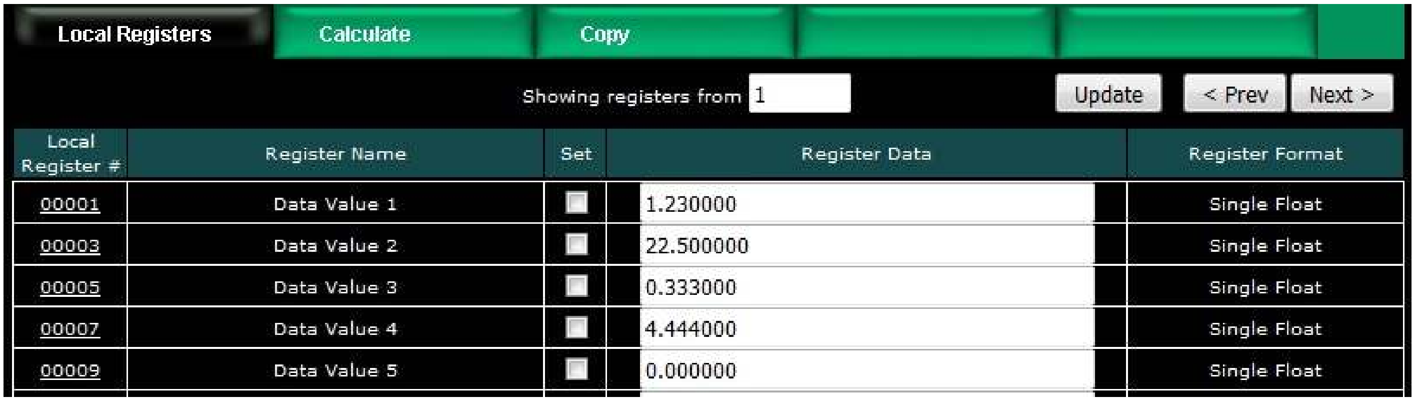


Address: 0001
Device Id: 1
MODBUS Point Type
Length: 10
03: HOLDING REGISTER

Number of Polls: 123
Valid Slave Responses: 123

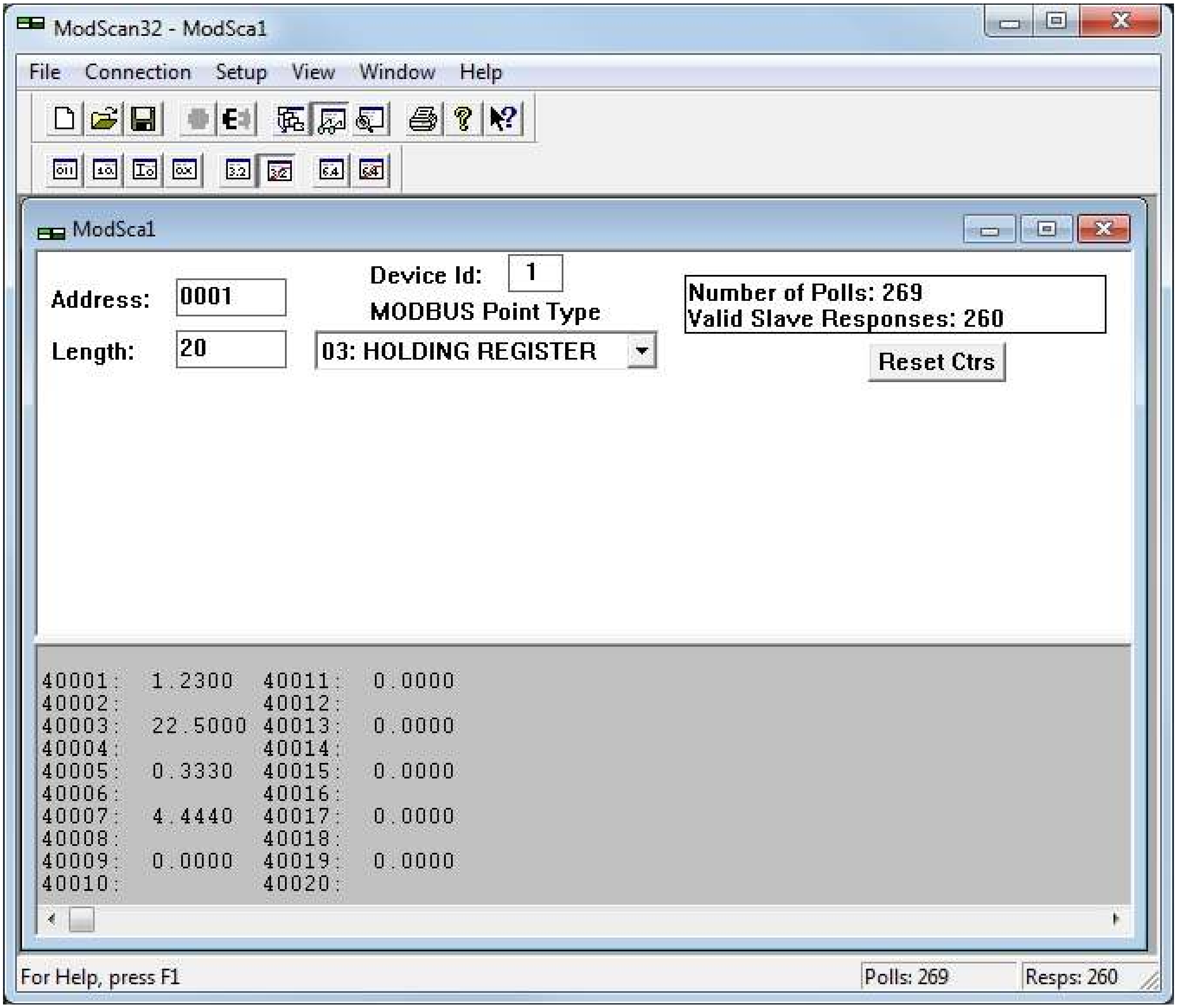
```
40001: <00123>
40002: <00022>
40003: <00000>
40004: <04567>
40005: <00000>
40006: <00000>
40007: <00000>
40008: <00000>
40009: <00000>
40010: <00000>
```

For Help, press F1 Polls: 124 Resps: 123



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 register #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).



Address: 0001
Length: 20
Device Id: 1
MODBUS Point Type: 03: HOLDING REGISTER

```
Number of Polls: 269
Valid Slave Responses: 260
```

```
40001:  1.2300  40011:  0.0000
40002:          40012:
40003:  22.5000 40013:  0.0000
40004:          40014:
40005:  0.3330  40015:  0.0000
40006:          40016:
40007:  4.4440  40017:  0.0000
40008:          40018:
40009:  0.0000  40019:  0.0000
40010:          40020:
```

Polls: 269 Resps: 260

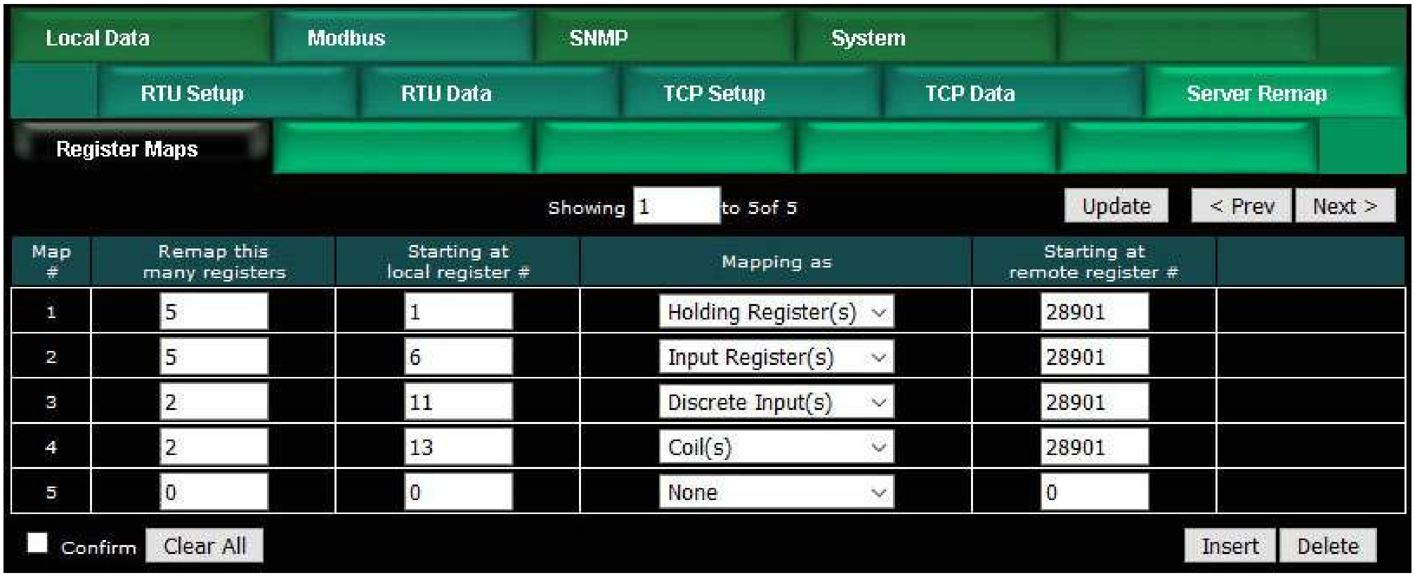
8.3 Modbus TCP Server Diagnostic

The Babel Buster brand new out of the box will have no registers configured. 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".

8.4 Modbus TCP Server Register Remapping

Remote Modbus clients (masters) will normally read the register numbers indicated on the Local Data page. In some cases, you may want to reassign those registers to be at arbitrary numbers all over the map in order to emulate some piece of existing equipment. When you have this requirement, this page is where you create your arbitrary register map.

Server remapping is automatically in effect any time maps are displayed on this page. If nothing is displayed here, then refer to the Local Data page to see what Modbus register numbers your client should read when this gateway is acting as a server. If anything is displayed here, then only these registers will be recognized by the server and any attempt to read registers not listed here will result in an exception being returned.



Showing 1 to 5of 5

Map #	Remap this many registers	Starting at local register #	Mapping as	Starting at remote register #
1	5	1	Holding Register(s)	28901
2	5	6	Input Register(s)	28901
3	2	11	Discrete Input(s)	28901
4	2	13	Coil(s)	28901
5	0	0	None	0

☐ Confirm

To remap registers, start by entering a count of registers to remap. With a count greater than one, you can remap many registers in a row with just one line on the above table. Provide the starting local register number - this is the number displayed on the Local Data page. Select what type of Modbus register you want this to pretend to be. And finally, provide the starting register number that you want the remote Modbus client to see these registers as. Click Update to accept the line and make room for another.

The example above illustrates the following: Local registers 1 through 5 will be read as holding registers 28901 through 28905. Local registers 6 through 10 will be read as input registers 28901 through 28905. Local registers 11 and 12 will be read as discrete inputs 28901 and 28902. Local registers 13 and 14 will be read as coils 28901 and 28902. The holding registers and coils may also be written at these register numbers.

When this table is complete, be sure to go to the File Manager page and save your updated configuration.

Use Prev and Next to scroll through the table if it is larger than a few lines. Delete will remove the map number shown in the "Showing" box. Insert will insert a new map before the map number shown, and is used for placing maps between existing maps. It is not necessary to use Insert to add maps to the bottom of the list or to define any map presently having "none" for a register type (unused rule).

To completely erase this entire table, check the "Confirm" box and click Clear All.

Modbus holding registers default to being 16-bit integer. If your remote client expects to see floating point, then create the local register as floating point on the Local Data page. The remote client will have access to whatever data types are used to create the local registers on the Local Data page.

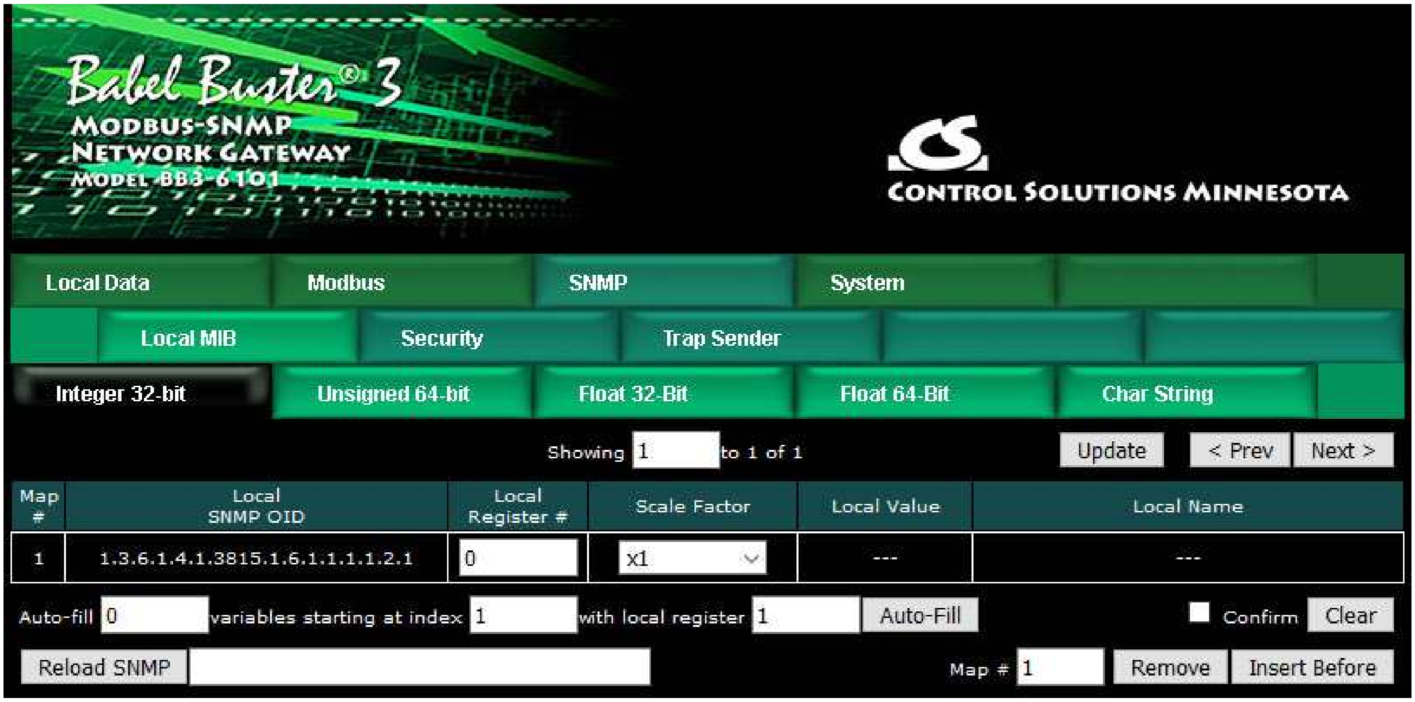
When holding register (or input register) data is more than 16 bits, the register order matters, but is not standardized by Modbus protocol. The order in which multiple registers will be provided to the remote client is part of the definition of the local register created on the Local Data page. Click on the register number in the first column on that page to see how that register is defined.

Modbus sometimes "packs" multiple data items into a single holding register. You may "pack" registers such that a single register viewed remotely is constructed from the contents of multiple local registers. To accomplish this, refer to the pack and fill operations found on the Calculate page under Local Data.

9. Configuring Gateway as an SNMP Server

9.1 Creating Local SNMP MIB

The Babel Buster BB3-6101/MX-61 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 BB3 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.



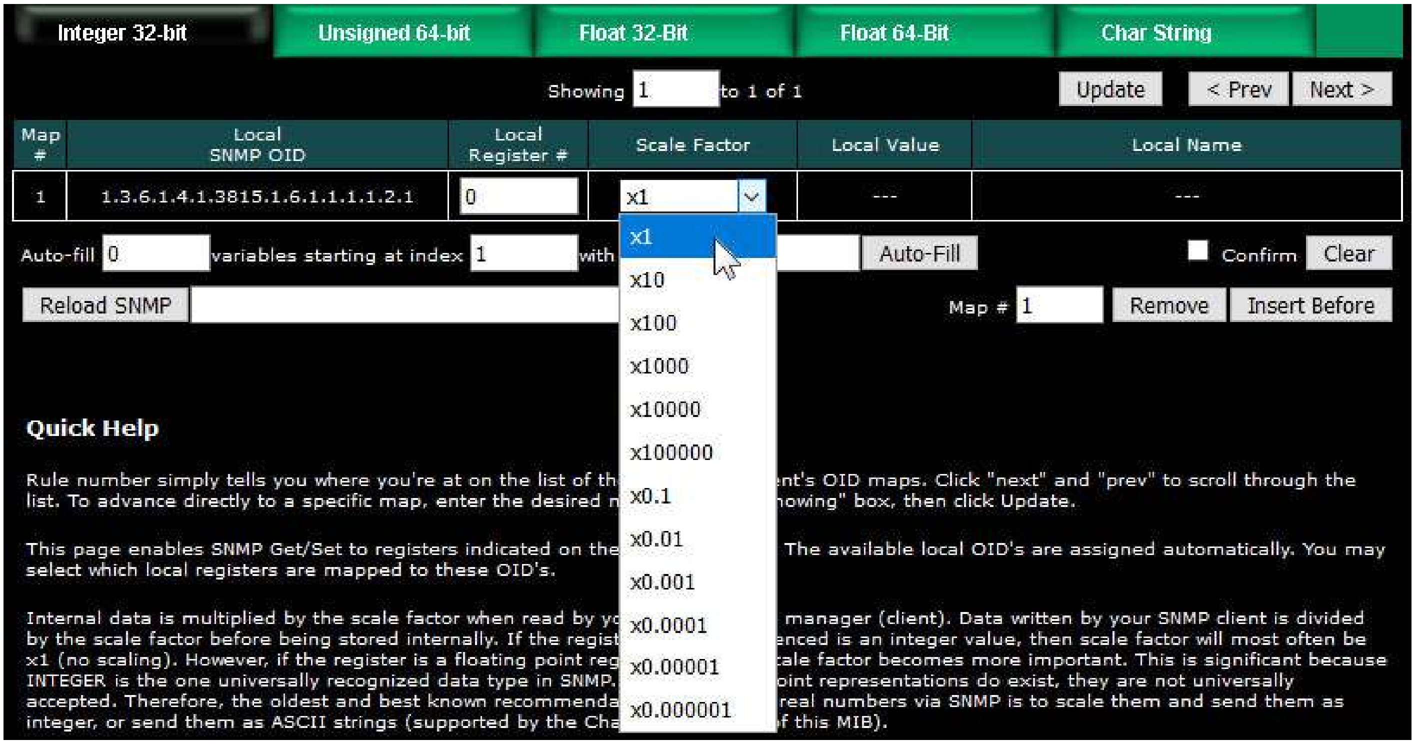
Showing 1 to 1 of 1

Map #	Local SNMP OID	Local Register #	Scale Factor	Local Value	Local Name
1	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1	0	x1	---	---

Auto-fill 0 variables starting at index 1 with local register 1 ☐ Confirm

Map # 1

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.



Showing 1 to 1 of 1

Map #	Local SNMP OID	Local Register #	Scale Factor	Local Value	Local Name
1	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1	0	x1	---	---

Auto-fill 0 variables starting at index 1 with □ Confirm

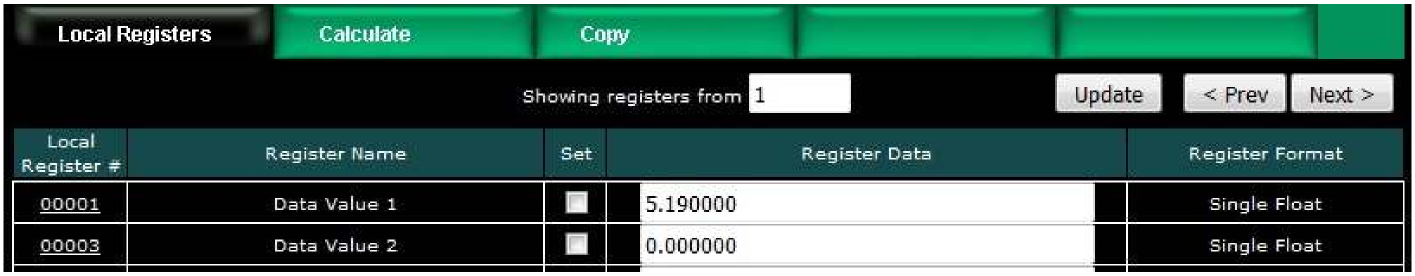
Map # 1

Quick Help

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

This page enables SNMP Get/Set to registers indicated on the The available local OID's are assigned automatically. You may select which local registers are mapped to these OID's.

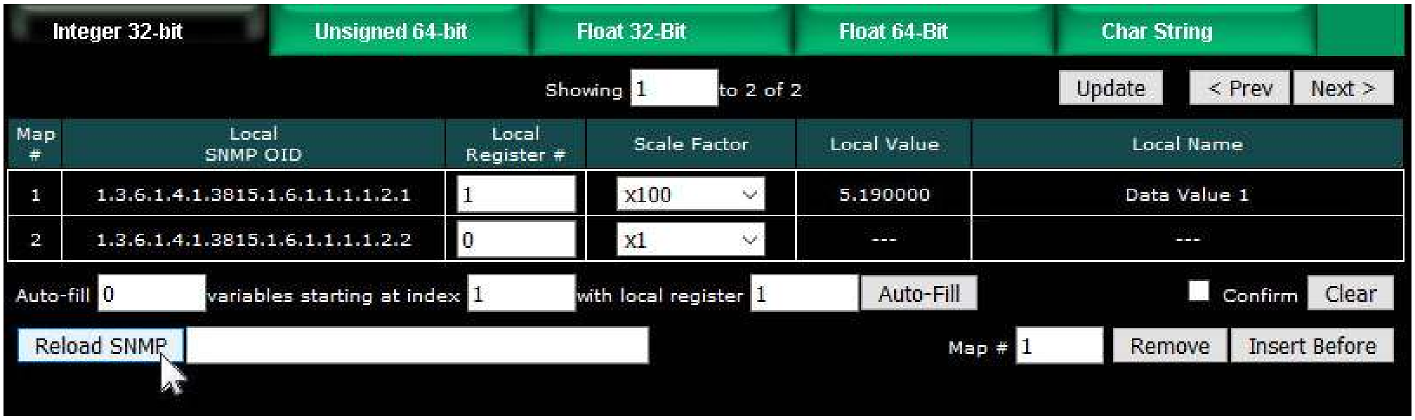
Internal data is multiplied by the scale factor when read by yo manager (client). Data written by your SNMP client is divided by the scale factor before being stored internally. If the regist enced is an integer value, then scale factor will most often be x1 (no scaling). However, if the register is a floating point reg ale factor becomes more important. This is significant because INTEGER is the one universally recognized data type in SNMP. oint representations do exist, they are not universally accepted. Therefore, the oldest and best known recommenda real numbers via SNMP is to scale them and send them as integer, or send them as ASCII strings (supported by the Cha f this MIB).



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	5.190000	Single Float
00003	Data Value 2	□	0.000000	Single Float

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.



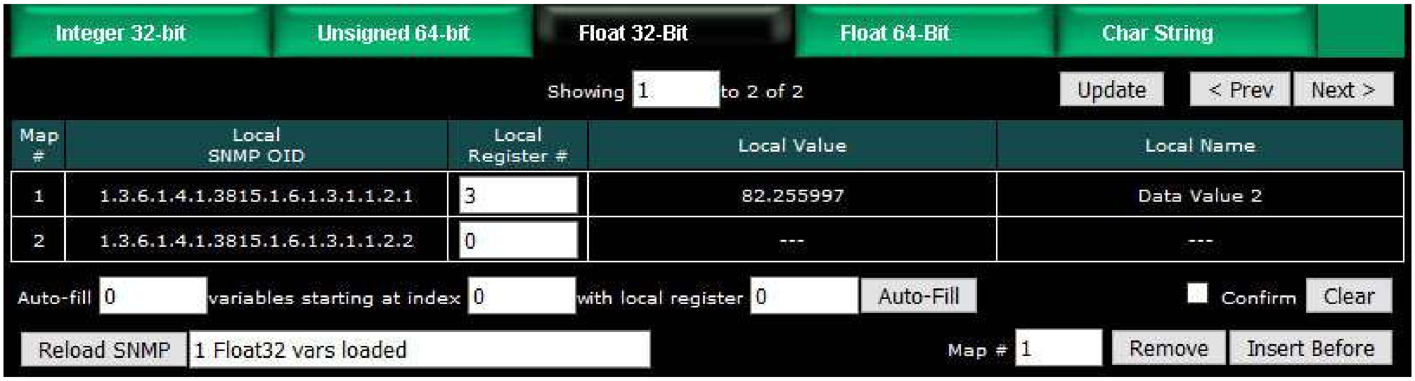
Showing 1 to 2 of 2

Map #	Local SNMP OID	Local Register #	Scale Factor	Local Value	Local Name
1	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1	1	x100	5.190000	Data Value 1
2	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.2	0	x1	---	---

Auto-fill 0 variables starting at index 1 with local register 1 □ Confirm

Map # 1

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.



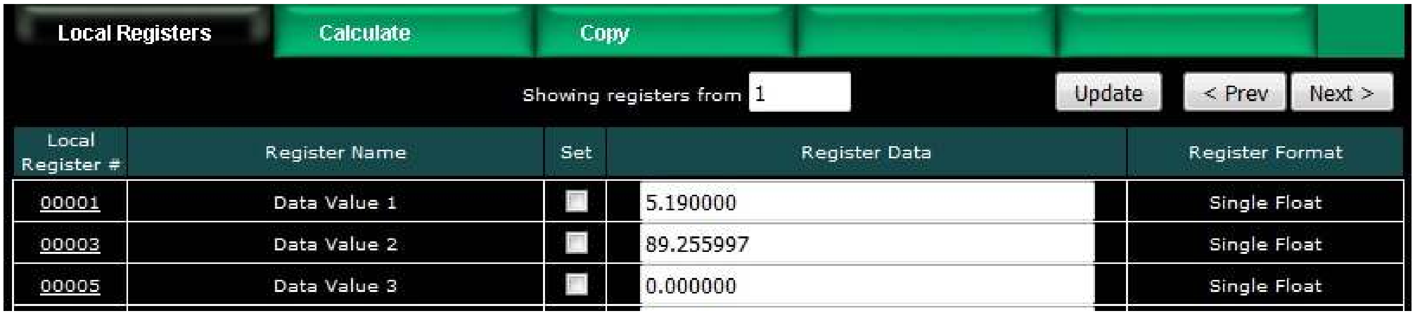
Showing 1 to 2 of 2

Map #	Local SNMP OID	Local Register #	Local Value	Local Name
1	1.3.6.1.4.1.3815.1.6.1.3.1.1.2.1	3	82.255997	Data Value 2
2	1.3.6.1.4.1.3815.1.6.1.3.1.1.2.2	0	---	---

Auto-fill 0 variables starting at index 0 with local register 0 Auto-Fill □ Confirm Clear

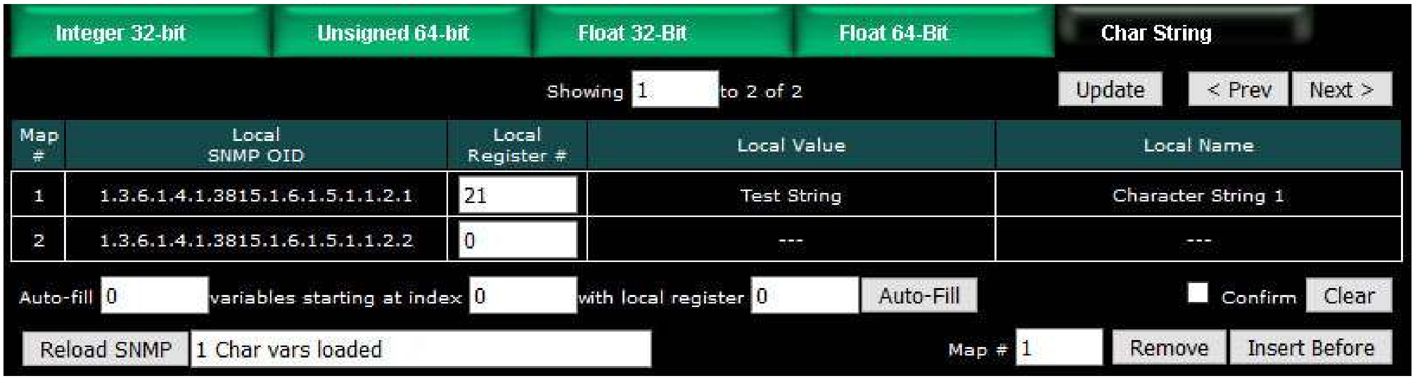
Reload SNMP 1 Float32 vars loaded Map # 1 Remove Insert Before

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



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	5.190000	Single Float
00003	Data Value 2	□	89.255997	Single Float
00005	Data Value 3	□	0.000000	Single Float



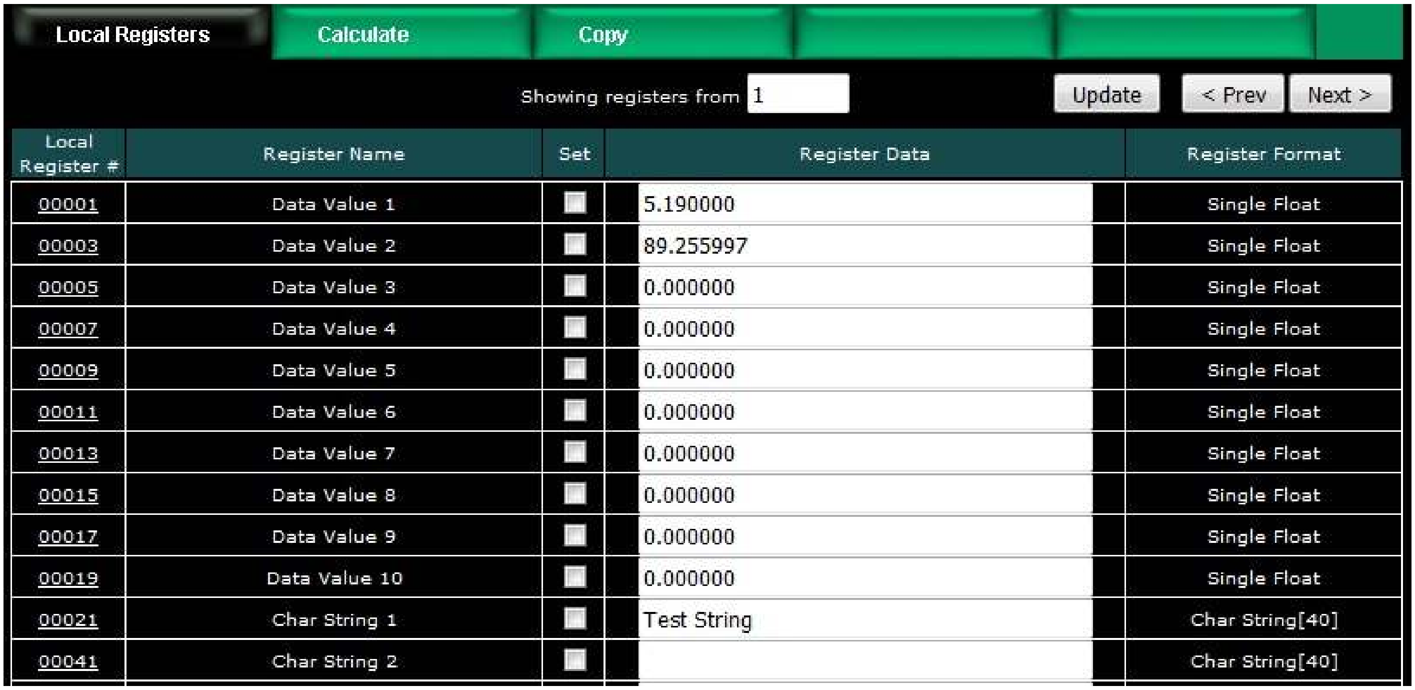
Showing 1 to 2 of 2

Map #	Local SNMP OID	Local Register #	Local Value	Local Name
1	1.3.6.1.4.1.3815.1.6.1.5.1.1.2.1	21	Test String	Character String 1
2	1.3.6.1.4.1.3815.1.6.1.5.1.1.2.2	0	---	---

Auto-fill 0 variables starting at index 0 with local register 0 Auto-Fill □ Confirm Clear

Reload SNMP 1 Char vars loaded Map # 1 Remove Insert Before

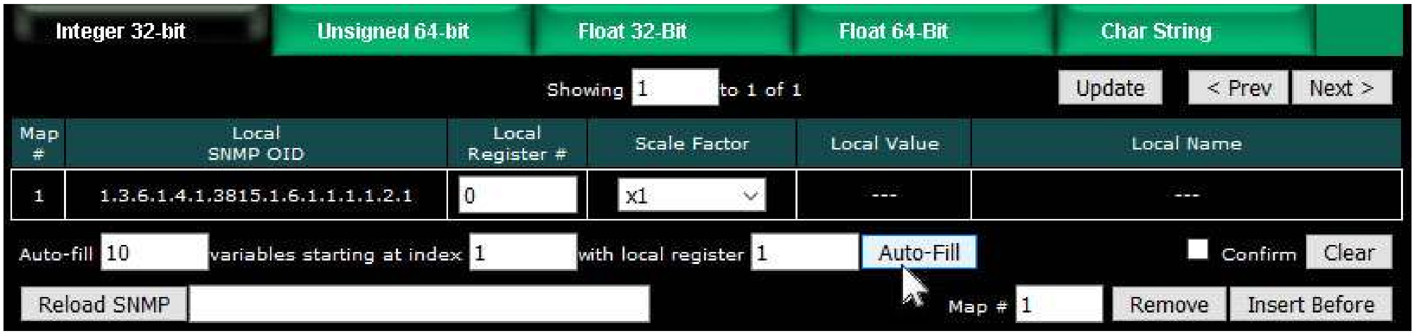
Local registers defined as character strings may be added to the Char String branch of the MIB. Registers in this branch will be provided as an Octet String.



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	5.190000	Single Float
00003	Data Value 2	□	89.255997	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	□	Test String	Char String[40]
00041	Char String 2	□		Char String[40]

The "Auto-fill" at the bottom of each MIB page can be used to very quickly create a long list of sequential MIB table entries. Enter the number of variables to fill, the starting MIB index, and the starting local register number. Then click the Auto-Fill button.



Showing 1 to 1 of 1

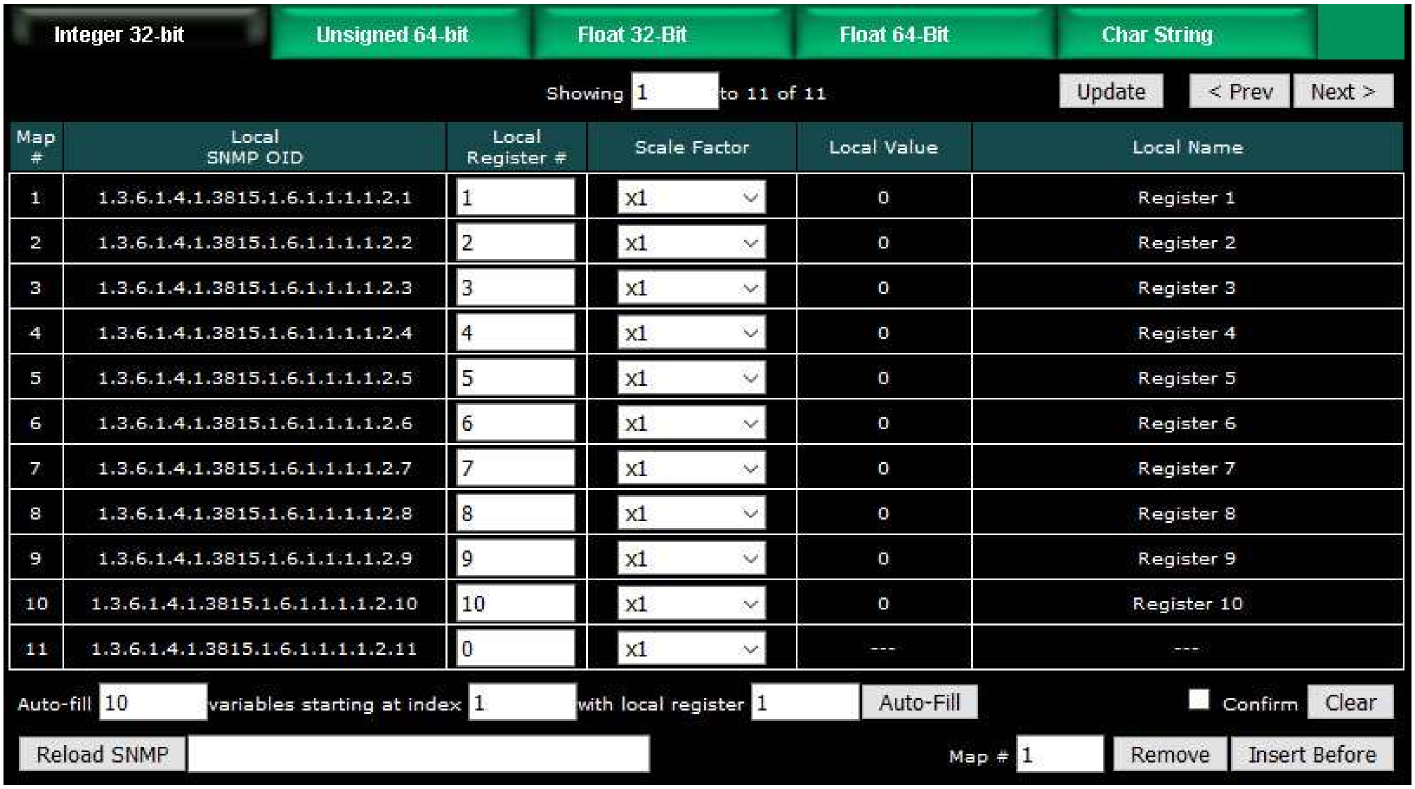
Map #	Local SNMP OID	Local Register #	Scale Factor	Local Value	Local Name
1	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1	0	x1	---	---

Auto-fill 10 variables starting at index 1 with local register 1

□ Confirm

Map # 1

The example above will create the list illustrated below, assuming the first ten local registers are all 16-bit integer registers.



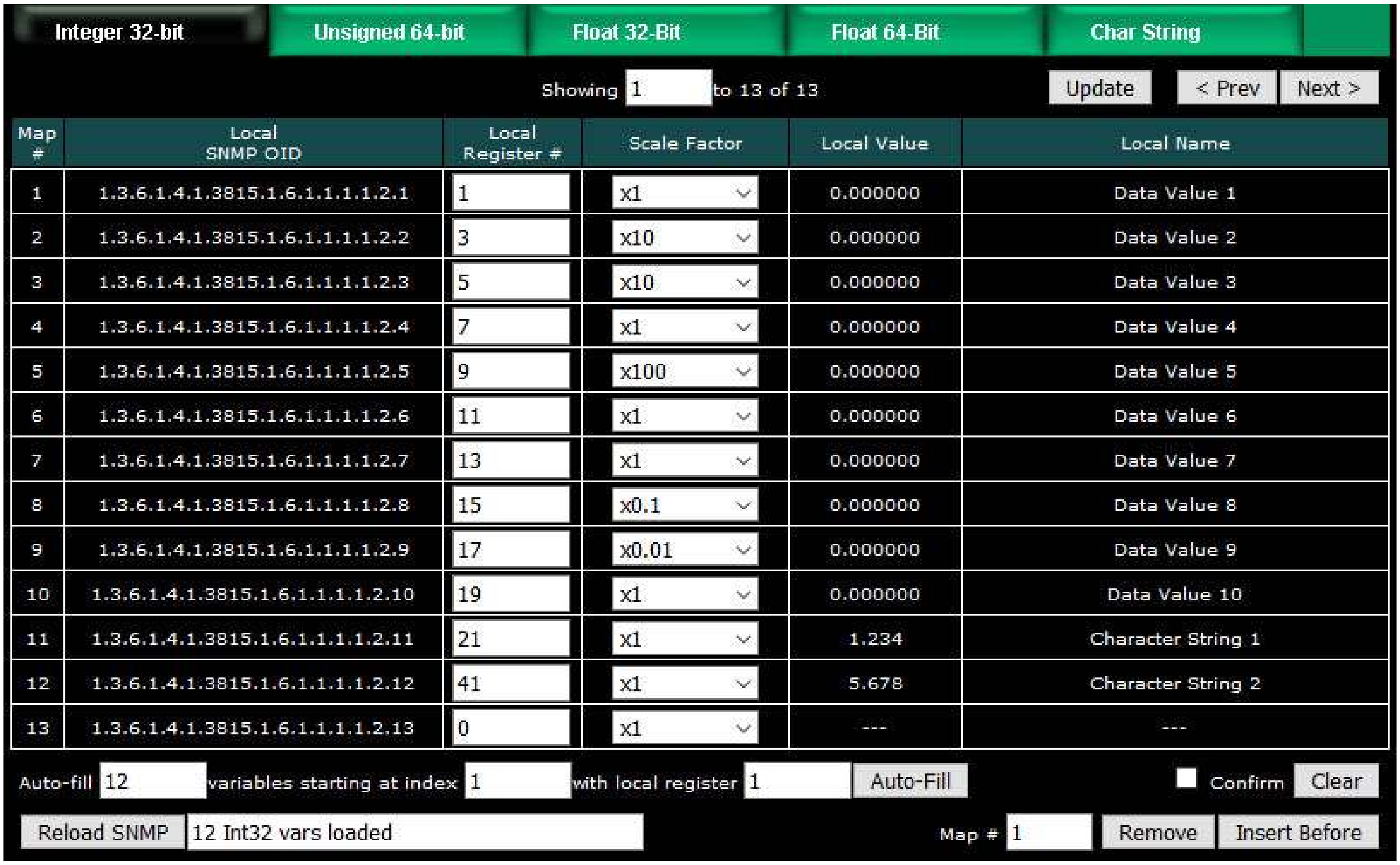
Showing 1 to 11 of 11

Map #	Local SNMP OID	Local Register #	Scale Factor	Local Value	Local Name
1	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1	1	x1	0	Register 1
2	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.2	2	x1	0	Register 2
3	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.3	3	x1	0	Register 3
4	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.4	4	x1	0	Register 4
5	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.5	5	x1	0	Register 5
6	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.6	6	x1	0	Register 6
7	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.7	7	x1	0	Register 7
8	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.8	8	x1	0	Register 8
9	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.9	9	x1	0	Register 9
10	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.10	10	x1	0	Register 10
11	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.11	0	x1	---	---

Auto-fill 10 variables starting at index 1 with local register 1 ☐ Confirm

Map # 1

The example illustrated below is a little more complex. The first ten local registers are floating point, which means they are treated as register pairs. Then the next couple of local "registers" are character strings each 40 characters in length, and each occupying 20 consecutive Modbus registers. Note that the "auto-fill" automatically accounts for data size, and choses the next local register appropriately. In this example, although the registers are a mix of floating point and character string, they are assigned to the Integer 32-bit branch of the MIB and will be read via SNMP as Integer values. The numbers in the character string registers are ASCII string representations of numbers, but will be converted to integer. For example, MIB index 12, with the character string "5.678", will be returned as an integer value 6 in an SNMP Get (converted and rounded).



Showing 1 to 13 of 13

Map #	Local SNMP OID	Local Register #	Scale Factor	Local Value	Local Name
1	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1	1	x1	0.000000	Data Value 1
2	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.2	3	x10	0.000000	Data Value 2
3	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.3	5	x10	0.000000	Data Value 3
4	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.4	7	x1	0.000000	Data Value 4
5	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.5	9	x100	0.000000	Data Value 5
6	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.6	11	x1	0.000000	Data Value 6
7	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.7	13	x1	0.000000	Data Value 7
8	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.8	15	x0.1	0.000000	Data Value 8
9	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.9	17	x0.01	0.000000	Data Value 9
10	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.10	19	x1	0.000000	Data Value 10
11	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.11	21	x1	1.234	Character String 1
12	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.12	41	x1	5.678	Character String 2
13	1.3.6.1.4.1.3815.1.6.1.1.1.1.2.13	0	x1	---	---

Auto-fill 12 variables starting at index 1 with local register 1 Auto-Fill ☐ Confirm Clear

Reload SNMP 12 Int32 vars loaded Map # 1 Remove Insert Before

The Auto-Fill simply sets up the MIB mapping table. Before the variables can be accessed by SNMP, you need to click Reload SNMP. The Remove and Insert Before buttons also just set up the mapping table, and you need to click Reload SNMP to make the new assignments accessible.

To clear the mapping table displayed, click the Confirm box and then click the Clear button. This clears the mapping table, but as with Auto-Fill, no changes (from the perspective of SNMP as viewed from the outside) take place until you click Reload SNMP. The Clear button is mainly useful if you change your mind about the auto-fill you just did.

Due to SNMP's memory allocation scheme, you can always increase the size of the MIB by simply clicking Reload SNMP; however, to reduce the size of the MIB, you need to save your configuration on the File Manager page, and then restart the gateway. If you reduce the number of entries in the mapping table, the original MIB variables at the end of the original list will simply return zero or empty strings until the gateway is restarted.

9.2 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 refer to RFC 6340 encoding.

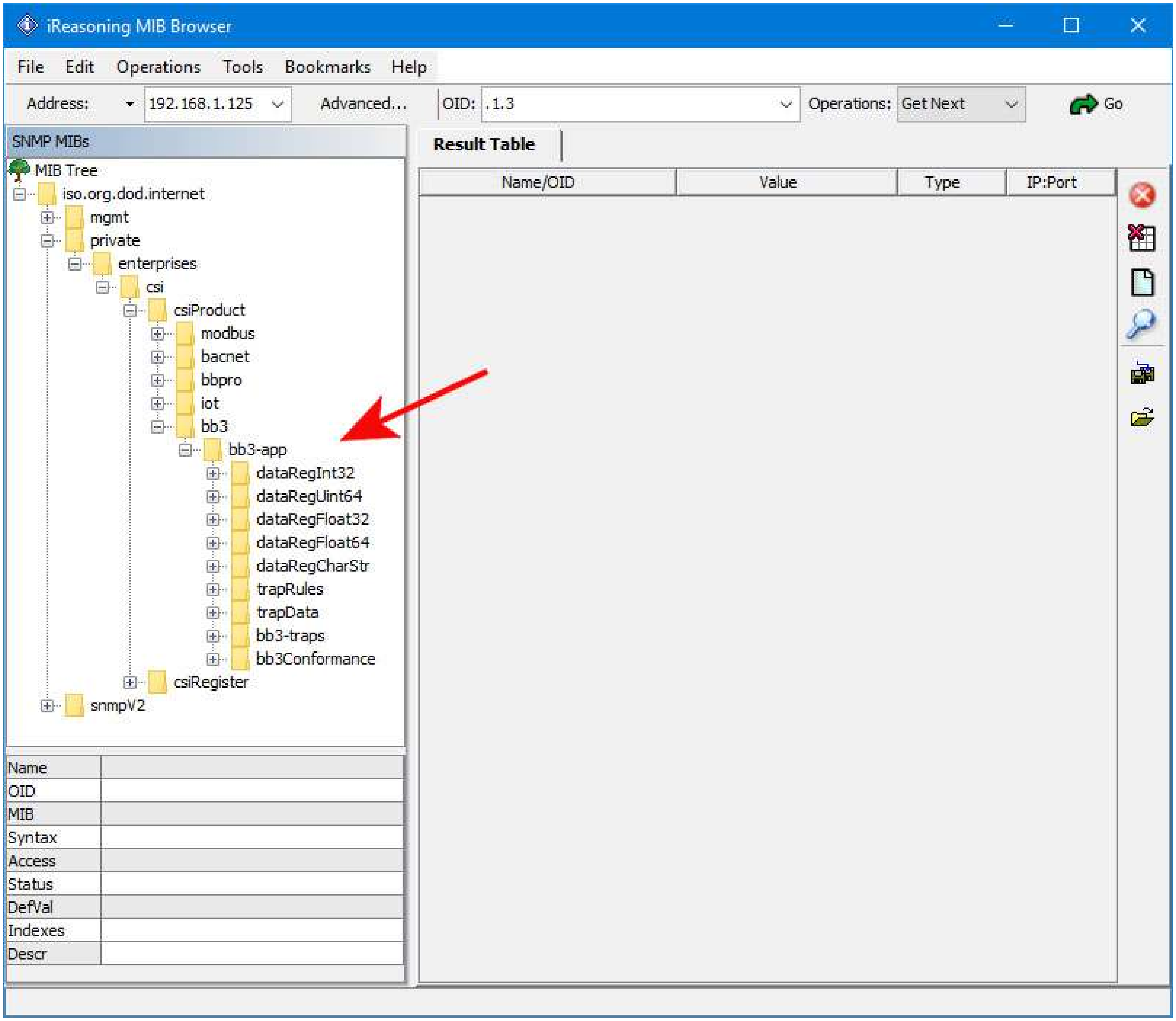
Specifically, the data types found in the Babel Buster MIB are encoded with ASN types as follows:

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)

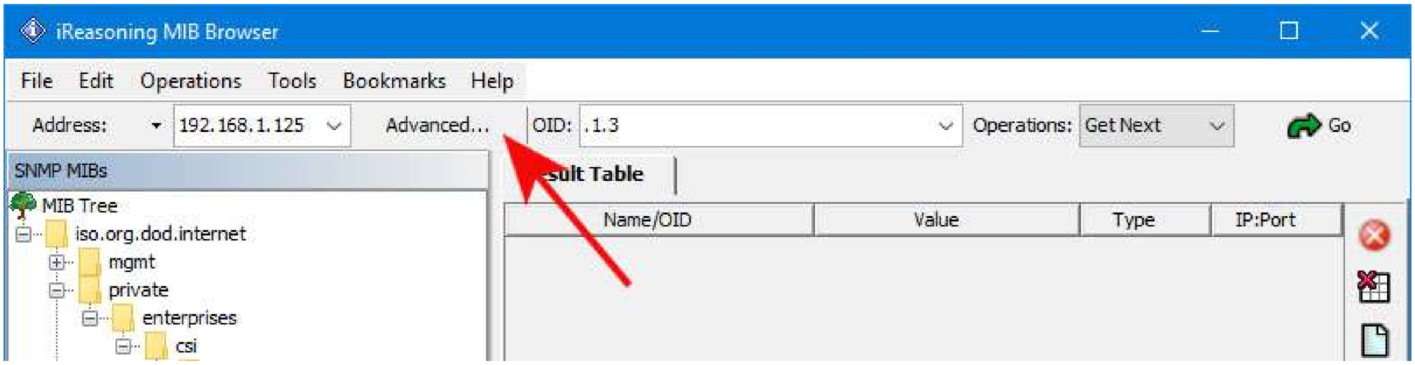
9.3 Testing the SNMP Agent

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.

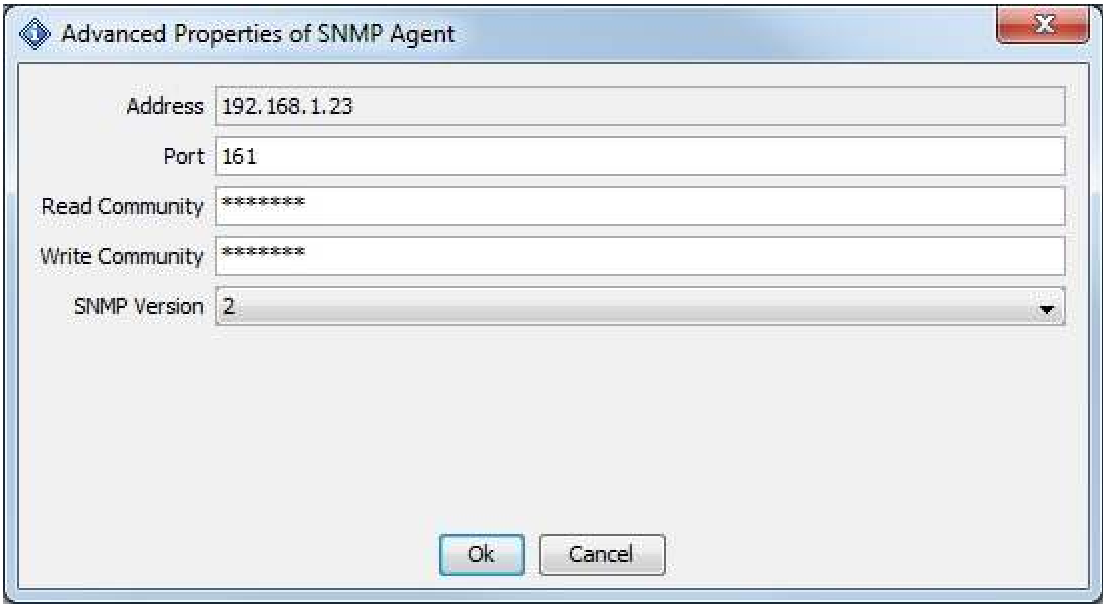
The MIB browser allows you to view the MIB variables in the Babel Buster. Before you can browse the MIB in a meaningful way, you need to load the MIB files that tell the browser what it needs to know about the BB3's MIB. There are one or more files that need to be loaded, and these are available under Documents and Links on the BB3-6101/MX-61 product page at csimn.com. 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 BB3-6101/MX-61 in the Address window. In addition. click on Advanced... to set access parameters.

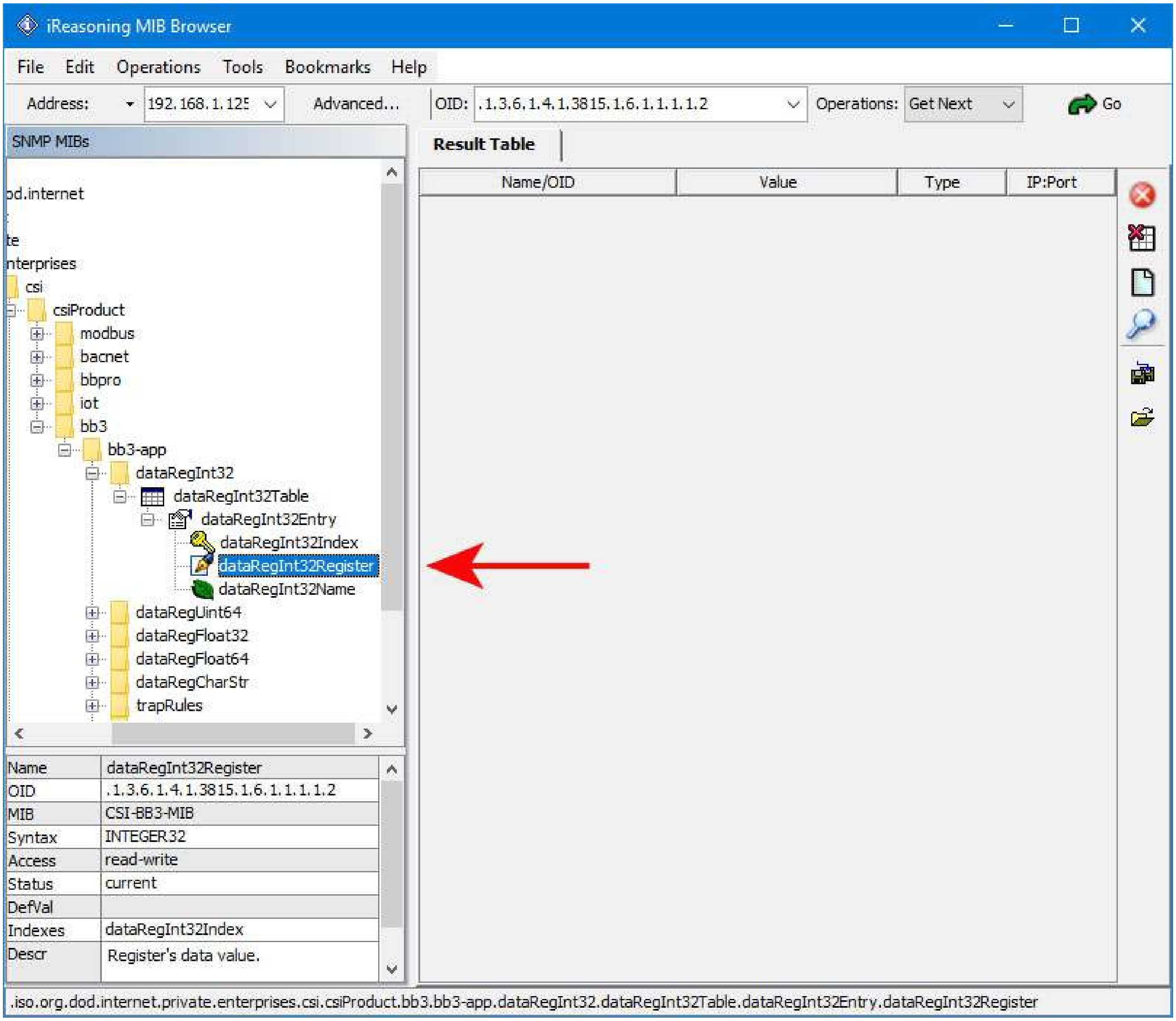


Click Advanced... to open the dialog that lets you enter the user name and community required for accessing the MIB in the Babel Buster.



Address 192.168.1.23
Port 161
Read Community ********
Write Community ********
SNMP Version 2

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



Address: 192.168.1.125

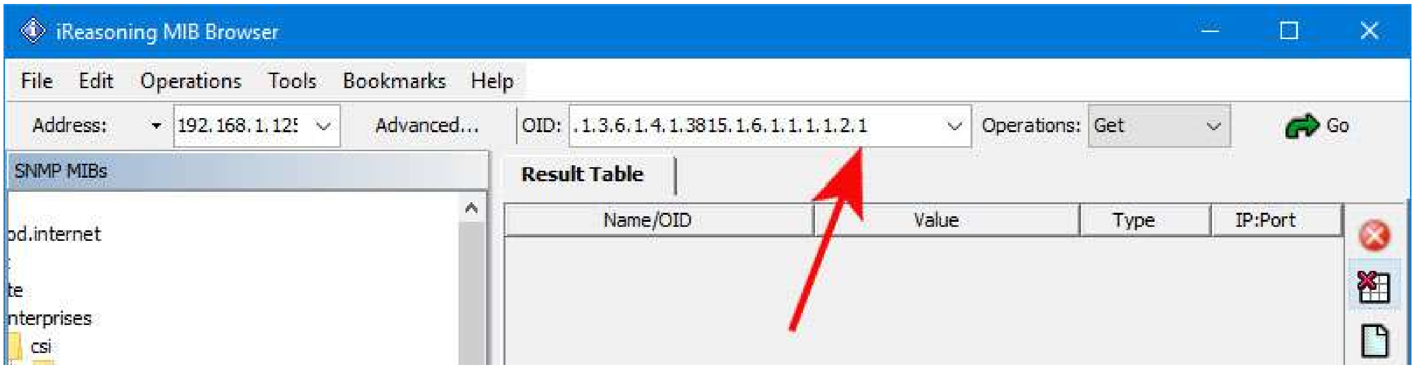
OID: .1.3.6.1.4.1.3815.1.6.1.1.1.1.2

Operations: Get Next

Name	dataRegInt32Register
OID	.1.3.6.1.4.1.3815.1.6.1.1.1.1.2
MIB	CSI-BB3-MIB
Syntax	INTEGER32
Access	read-write
Status	current
DefVal	
Indexes	dataRegInt32Index
Descr	Register's data value.

.iso.org.dod.internet.private.enterprises.csi.csiProduct.bb3.bb3-app.dataRegInt32.dataRegInt32Table.dataRegInt32Entry.dataRegInt32Register

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.

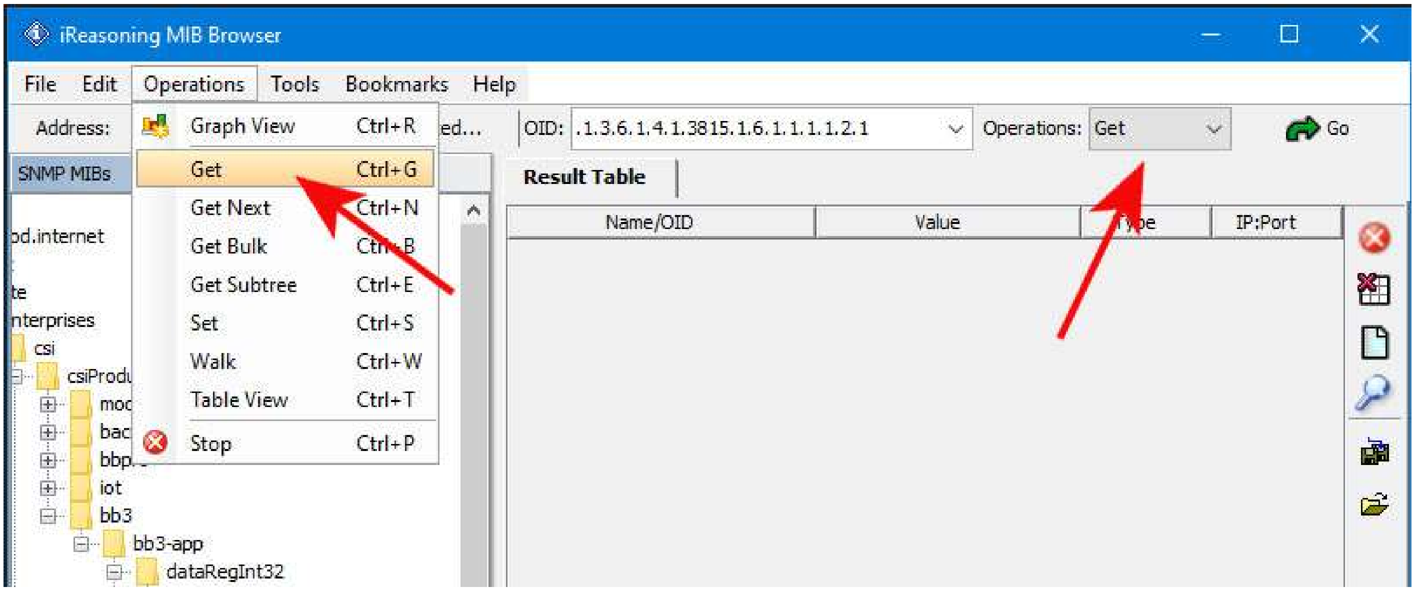


Address: 192.168.1.125

OID: .1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1

Operations: Get

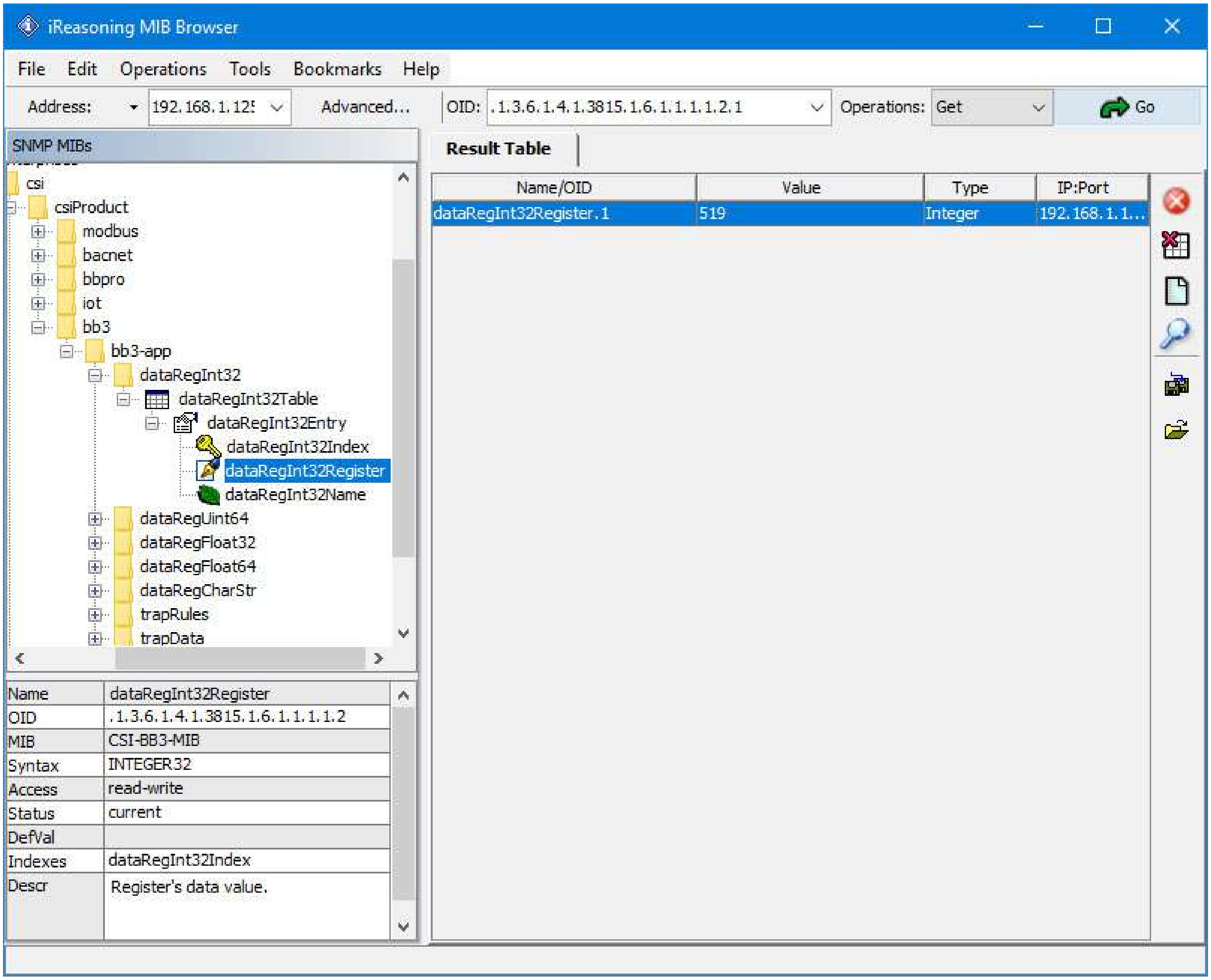
To cause the MIB browser to Get a value from the Babel Buster, select Get from the Operations menu. You can select Get from either menu, left or right, in the iReasoning browser. Once Get is selected on the right, you only need to click the Go button to repeat the Get.



OID: .1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1 Operations: Get

Name/OID	Value	Type	IP:Port

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

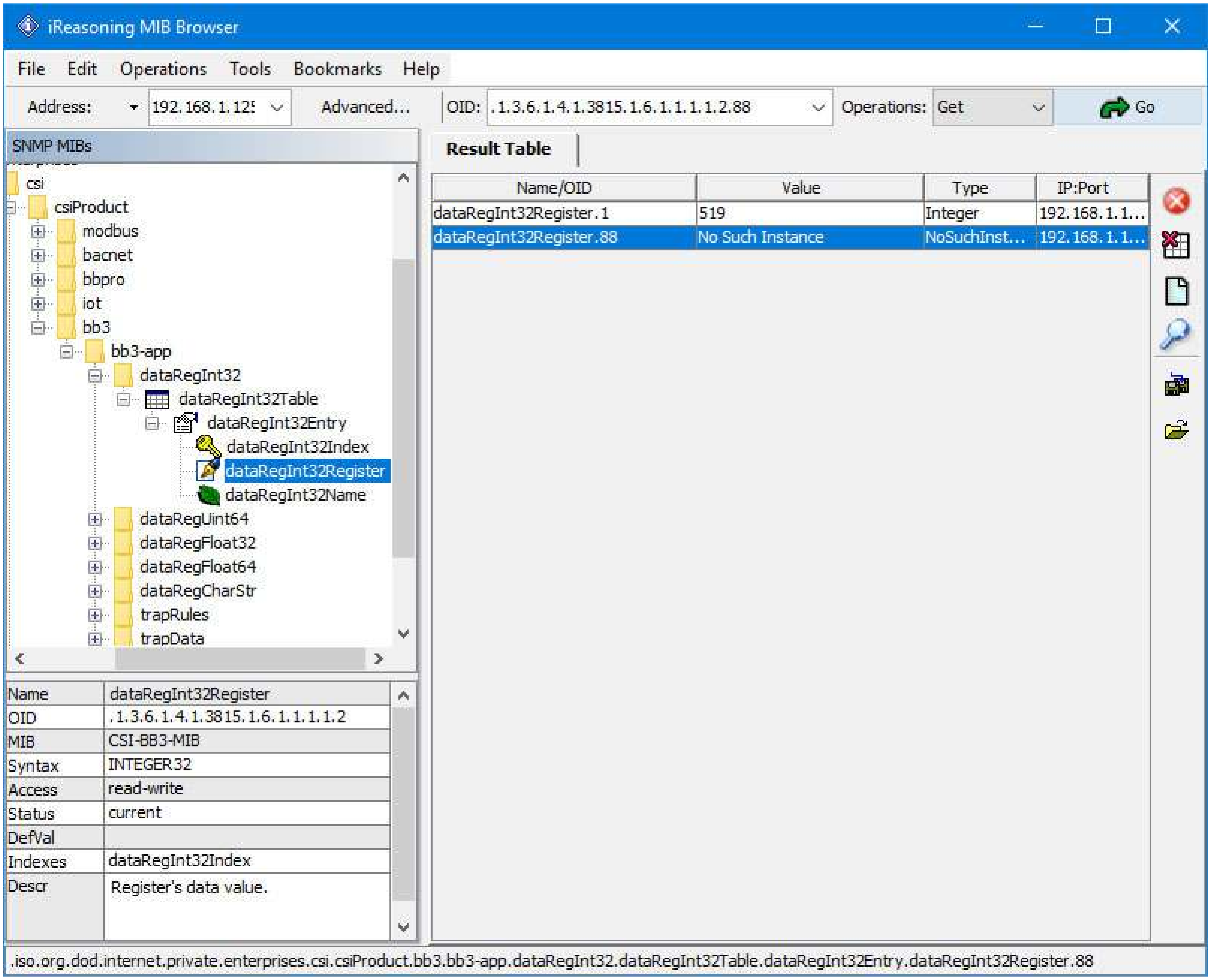


Address: 192.168.1.12! OID: .1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1 Operations: Get

Name/OID	Value	Type	IP:Port
dataRegInt32Register.1	519	Integer	192.168.1.1...

Name	dataRegInt32Register
OID	.1.3.6.1.4.1.3815.1.6.1.1.1.1.2
MIB	CSI-BB3-MIB
Syntax	INTEGER32
Access	read-write
Status	current
DefVal	
Indexes	dataRegInt32Index
Descr	Register's data value.

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.



Address: 192.168.1.12!

OID: .1.3.6.1.4.1.3815.1.6.1.1.1.1.2.88

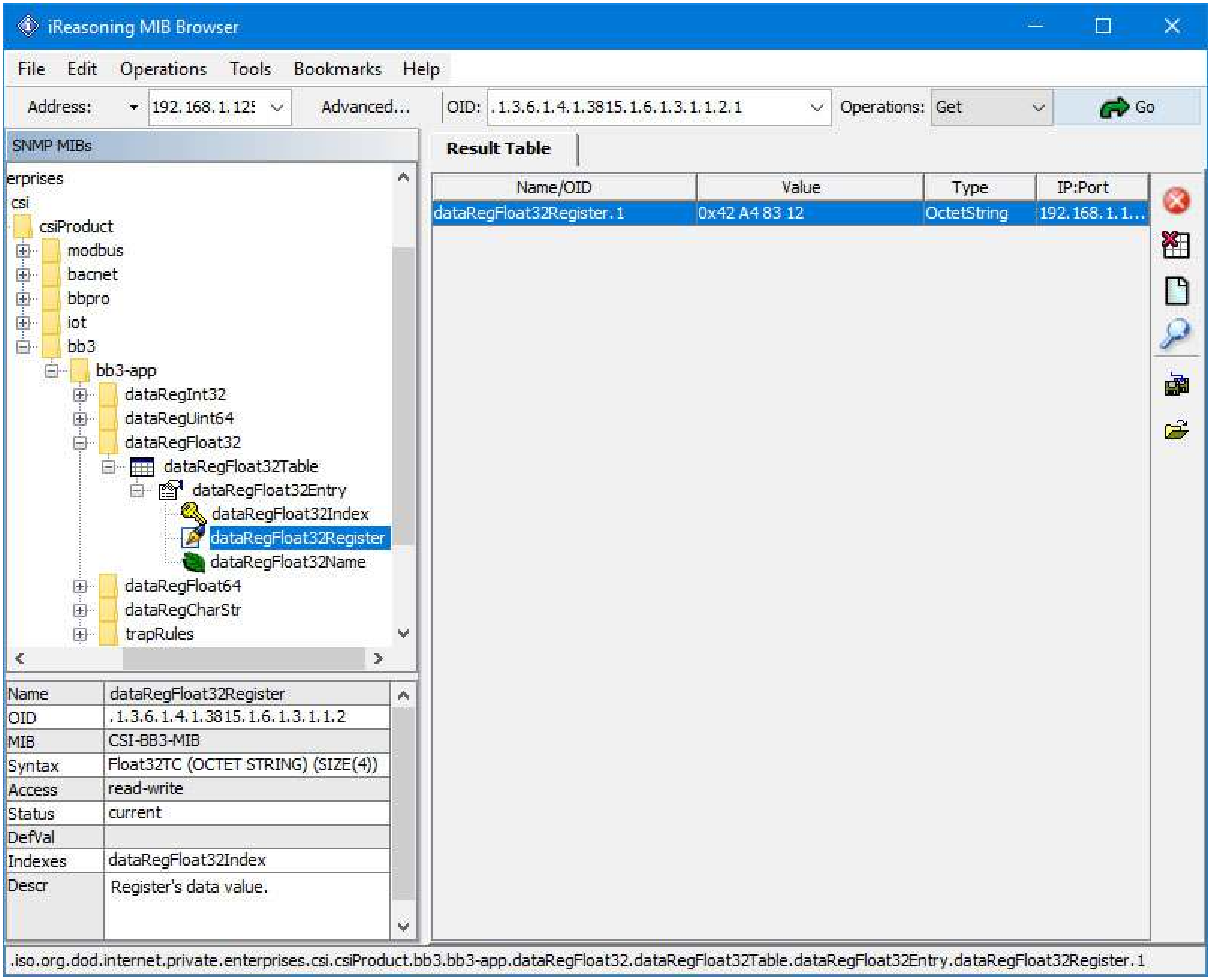
Operations: Get

Name/OID	Value	Type	IP:Port
dataRegInt32Register.1	519	Integer	192.168.1.1...
dataRegInt32Register.88	No Such Instance	NoSuchInst...	192.168.1.1...

Name	dataRegInt32Register
OID	.1.3.6.1.4.1.3815.1.6.1.1.1.1.2
MIB	CSI-BB3-MIB
Syntax	INTEGER32
Access	read-write
Status	current
DefVal	
Indexes	dataRegInt32Index
Descr	Register's data value.

.iso.org.dod.internet.private.enterprises.csi.csiProduct.bb3.bb3-app.dataRegInt32.dataRegInt32Table.dataRegInt32Entry.dataRegInt32Register.88

Getting our floating point value will appear as follows. The bytes "42 A4 83 12" translate to the floating point version of 82.255997. You can use Google to locate conversion tools to go from IEEE 754 hexadecimal representation (as shown in the browser) to decimal.



Address: 192.168.1.12!

OID: .1.3.6.1.4.1.3815.1.6.1.3.1.1.2.1

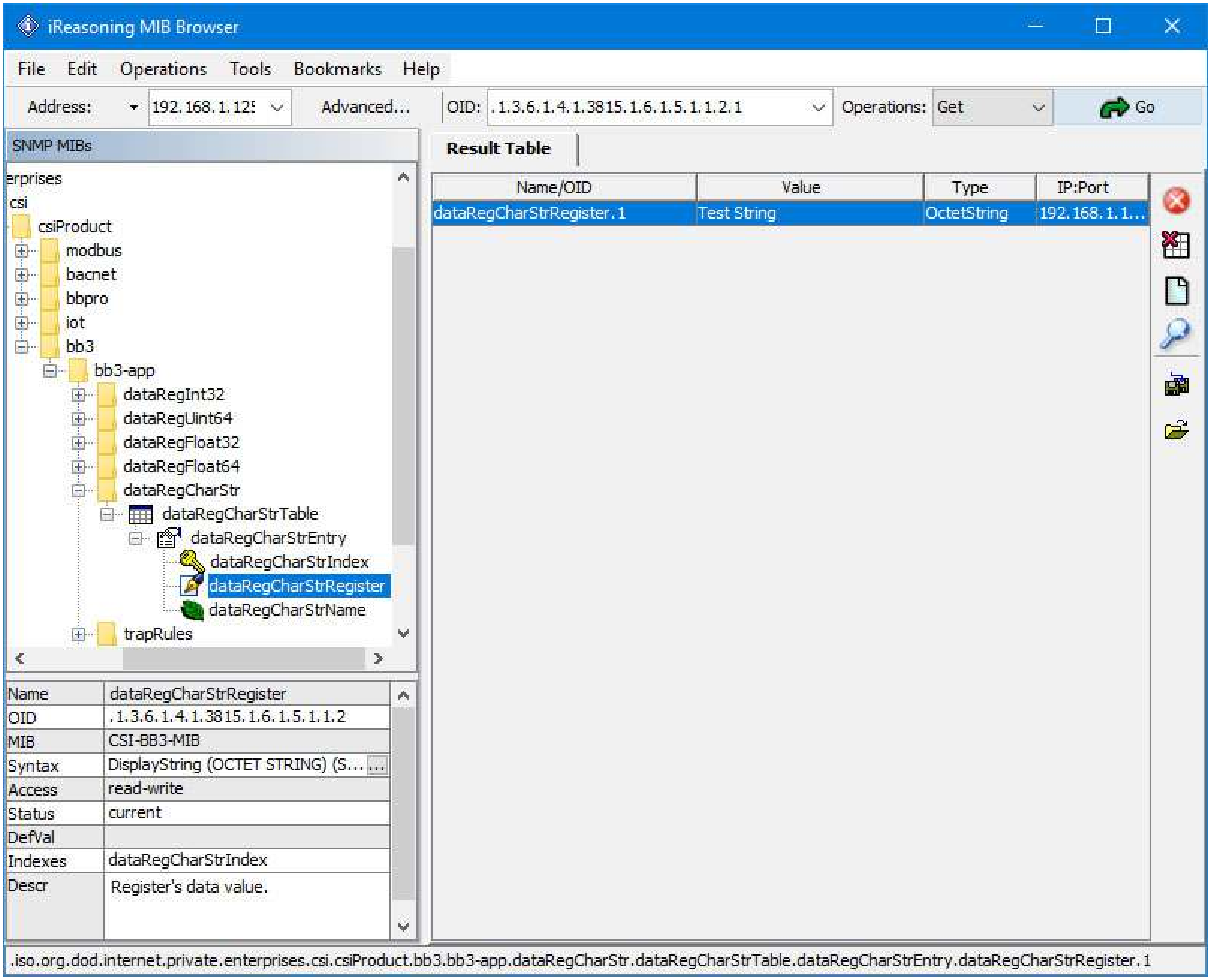
Operations: Get

Name/OID	Value	Type	IP:Port
dataRegFloat32Register.1	0x42 A4 83 12	OctetString	192.168.1.1...

Name	dataRegFloat32Register
OID	.1.3.6.1.4.1.3815.1.6.1.3.1.1.2
MIB	CSI-BB3-MIB
Syntax	Float32TC (OCTET STRING) (SIZE(4))
Access	read-write
Status	current
DefVal	
Indexes	dataRegFloat32Index
Descr	Register's data value.

.iso.org.dod.internet.private.enterprises.csi.csiProduct.bb3.bb3-app.dataRegFloat32.dataRegFloat32Table.dataRegFloat32Entry.dataRegFloat32Register.1

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.



Name/OID	Value	Type	IP:Port
dataRegCharStrRegister.1	Test String	OctetString	192.168.1.1...

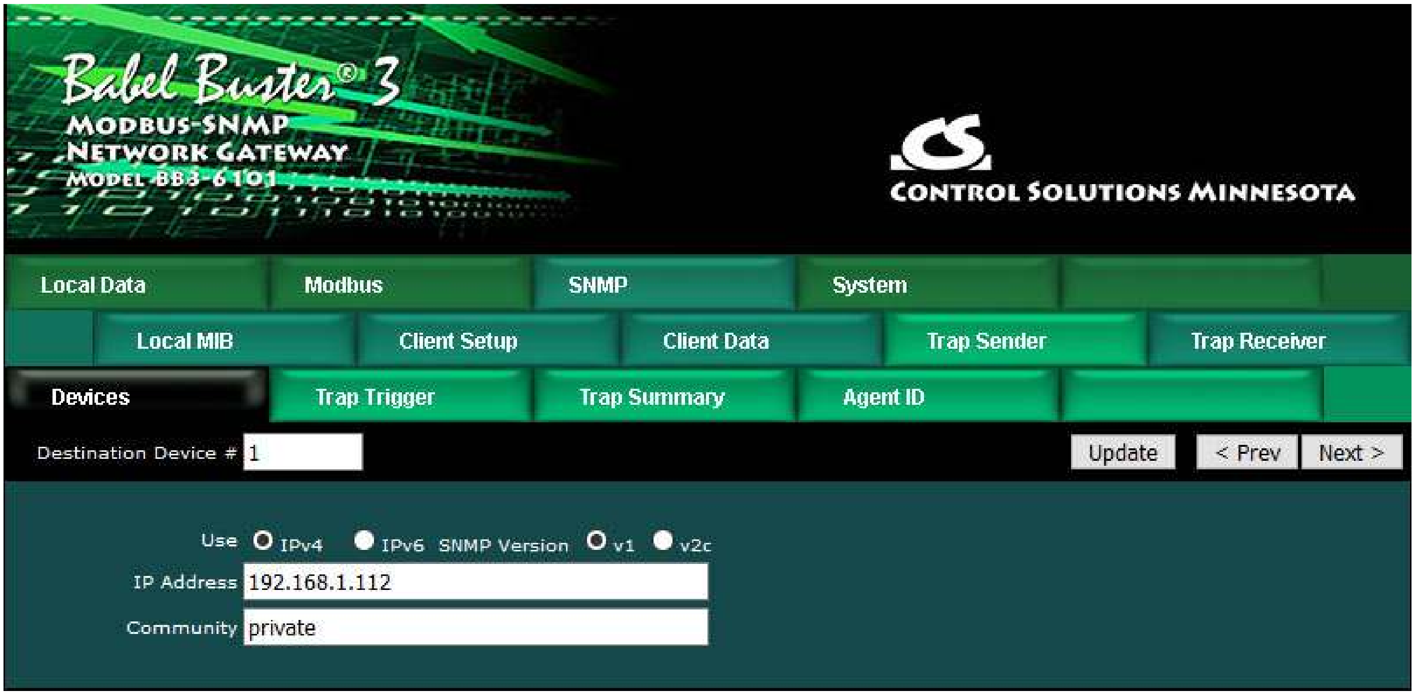
Name	dataRegCharStrRegister
OID	.1.3.6.1.4.1.3815.1.6.1.5.1.1.2
MIB	CSI-BB3-MIB
Syntax	DisplayString (OCTET STRING) (S...
Access	read-write
Status	current
DefVal	
Indexes	dataRegCharStrIndex
Descr	Register's data value.

.iso.org.dod.internet.private.enterprises.csi.csiProduct.bb3.bb3-app.dataRegCharStr.dataRegCharStrTable.dataRegCharStrEntry.dataRegCharStrRegister.1

10. Configuring SNMP Trap Sender

10.1 SNMP Trap Destinations

The first step in sending traps is to tell Babel Buster where to send them. This is done on the Devices page of the Trap Sender. Traps or Notifications generated by this device will be sent to port 162 at the IP addresses provided here.



Destination Device # 1

Use ○ IPv4 ● IPv6 SNMP Version ○ v1 ● v2c

IP Address 192.168.1.112

Community private

Traps may be sent as v1 or v2c. The same application varbinds will be included in either trap type, in addition to header and varbinds required for that trap type. (Note: We use the term "trap" generically here. The v2c "trap" is properly known as a Notification.)

To remove a destination, simply leave the IP address field blank. To add destinations, or change or remove existing entries, make the necessary changes here, then go to the File Manager page and save your configuration to the Boot configuration XML file. Then restart this device. The trap destinations are loaded one time prior to startup of the SNMP engine. To reload the trap destination table, you must restart the device.

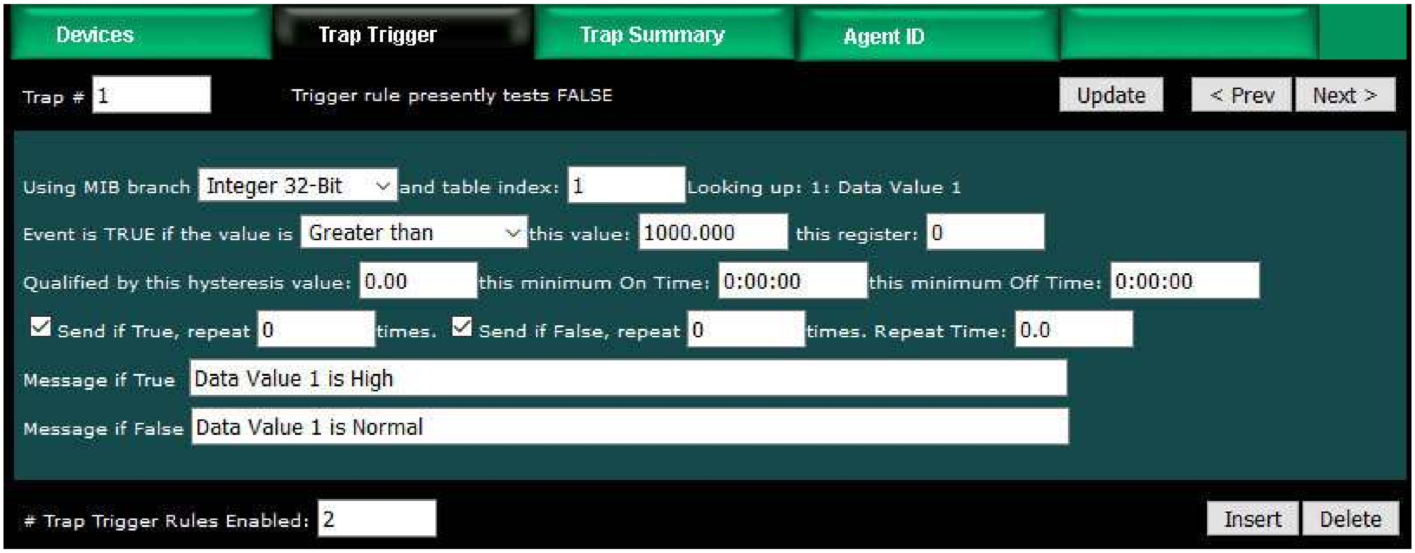
The community string entered here will be included in traps sent to this destination.

10.2 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.

The register found in the MIB branch, at the table index given, is displayed for reference as "Looking up...". To change this register, go to that MIB branch and table index, and change the MIB definition, or select a different branch and/or table index.



Trap # 1

Trigger rule presently tests FALSE

Using MIB branch Integer 32-Bit and table index: 1 Looking up: 1: Data Value 1

Event is TRUE if the value is Greater than this value: 1000.000 this register: 0

Qualified by this hysteresis value: 0.00 this minimum On Time: 0:00:00 this minimum Off Time: 0:00:00

☑ Send if True, repeat 0 times. ☑ Send if False, repeat 0 times. Repeat Time: 0.0

Message if True Data Value 1 is High

Message if False Data Value 1 is Normal

Trap Trigger Rules Enabled: 2

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. If register is zero, then the fixed value will be used. If a non-zero register number is given, then the fixed value is disregarded. The trap trigger will be "true" when the MIB variable meets the criteria given.

Quaifications are optional, and enabled only when values are nonzero. How hysteresis is applied depends on the comparison. For a test that becomes true if greater than, the test will not return to false until the local register is less than the test value by a margin of at least this hysteresis value. If a test becomes true if less than, it will not return to false until the local register is greater than the test value by a margin of at least this hysteresis value.

Special test types: "Deviates from" will test against the value given, and use Hysteresis as the margin of deviation. This is effectively a "greater than or less than" test for deviation from a setpoint. "Changes by" will become true each time the given variable changes by the value given, and Hysteresis has no effect on this test. If "Changes by" references a value of zero, then this becomes a special test whereby the event is true any time something in the system updates the variable. A "changes by zero" should not be used when the variable is continuously read from a slave device since this will result in continuous traps.

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. Times are given in HH:MM:SS format (hours, minutes, seconds).

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".

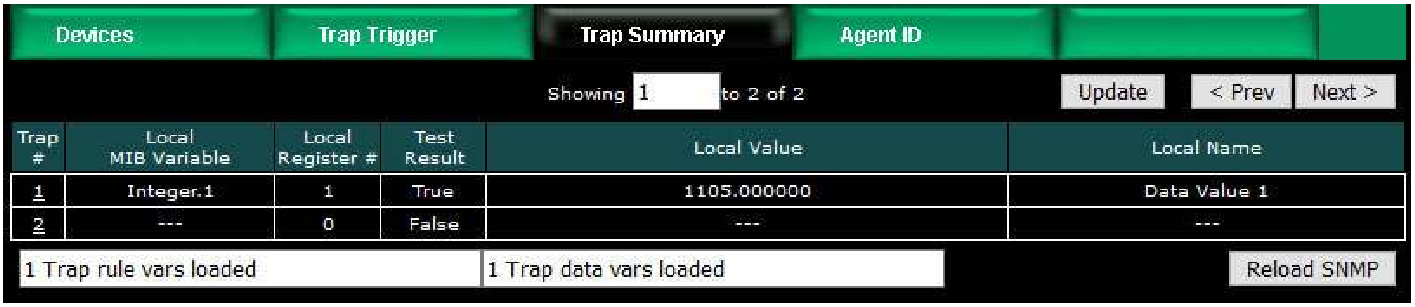
Enter a repeat count if the trap should be repeated. If repeat count is zero, the trap will be sent one time. If repeat count is 1, the trap will be sent 2 times, and so on. The interval between traps will be the Repeat Time in seconds. Enter -1 for trap repeat count if the trap should simply repeat indefinitely at the Repeat Time interval. A repeat count of -1 for "Send if True" is acceptable. A repeat count of -1 for "Send if False" will be treated as no repeat since indefinite repeating of non-true events is ill-advised.

One of the varbinds in the trap message is an arbitrary ASCII character string, sent as an ASN Octet string. The "True" message will be sent when the trap event is true, and the "False" message will be sent when the trap event is false.

To delete the rule shown, click Delete. To insert a new rule before the rule displayed, click Insert. To add a rule to the end of the list, click Next when at the end of the list, enter new rule, and click Update. 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.

10.3 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 "true" status of the above trigger.



Showing 1 to 2 of 2

Trap #	Local MIB Variable	Local Register #	Test Result	Local Value	Local Name
1	Integer.1	1	True	1105.000000	Data Value 1
2	---	0	False	---	---

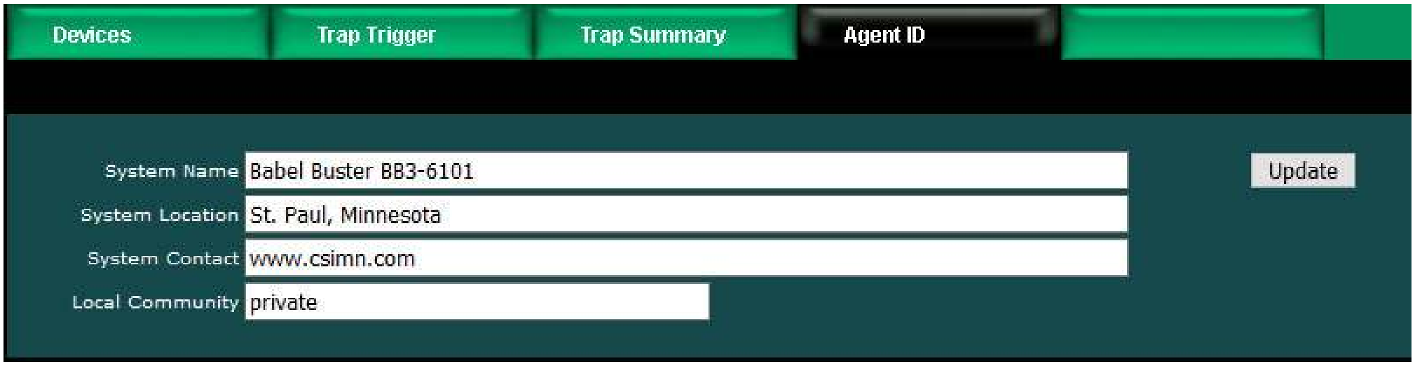
1 Trap rule vars loaded

1 Trap data vars loaded

10.4 SNMP Identity

The name, location, and contact listed above may be retrieved by the remote SNMP client.

The local community is the name that will be sent in Traps if no alternate community string is provided for the trap destination device. This name must also be used by the remote SNMP client to Get or Set this device. The name "public" is no longer accepted (unless you explicitly enter it here - not recommended).



System Name: Babel Buster BB3-6101

System Location: St. Paul, Minnesota

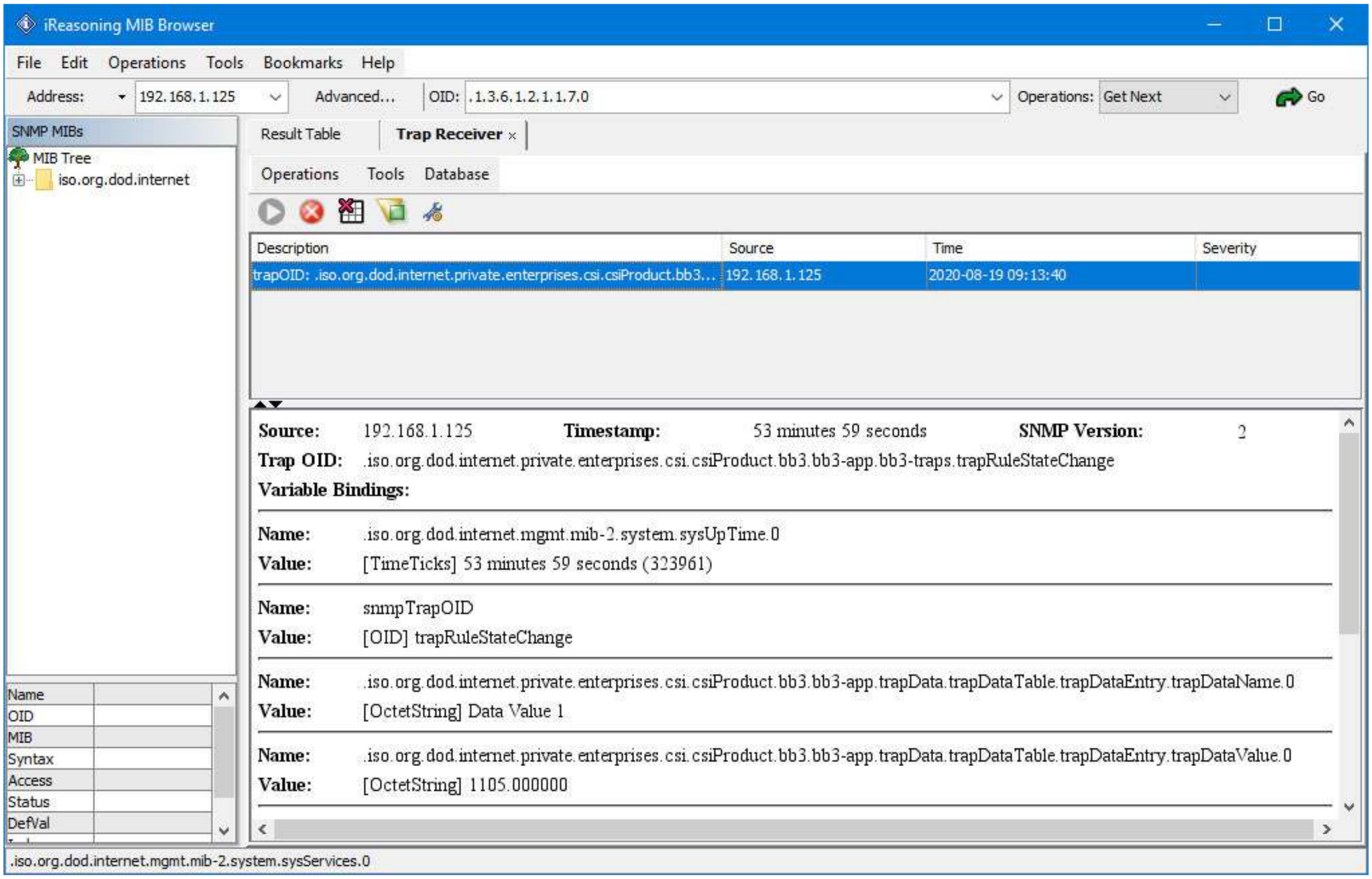
System Contact: www.csimn.com

Local Community: private

10.5 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 the above trap from our Babel Buster.



Address: 192.168.1.125

OID: .1.3.6.1.2.1.1.7.0

Operations: Get Next

Description	Source	Time	Severity
trapOID: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bb3...	192.168.1.125	2020-08-19 09:13:40	

Source: 192.168.1.125 **Timestamp:** 53 minutes 59 seconds **SNMP Version:** 2

Trap OID: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bb3.bb3-app.bb3-traps.trapRuleStateChange

Variable Bindings:

Name: .iso.org.dod.internet.mgmt.mib-2.system.sysUpTime.0
Value: [TimeTicks] 53 minutes 59 seconds (323961)

Name: snmpTrapOID
Value: [OID] trapRuleStateChange

Name: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bb3.bb3-app.trapData.trapDataTable.trapDataEntry.trapDataName.0
Value: [OctetString] Data Value 1

Name: .iso.org.dod.internet.private.enterprises.csi.csiProduct.bb3.bb3-app.trapData.trapDataTable.trapDataEntry.trapDataValue.0
Value: [OctetString] 1105.000000

.iso.org.dod.internet.mgmt.mib-2.system.sysServices.0

Babel Buster® 3
MODEL BB3-6101
MODEL MX-61
NETWORK GATEWAY

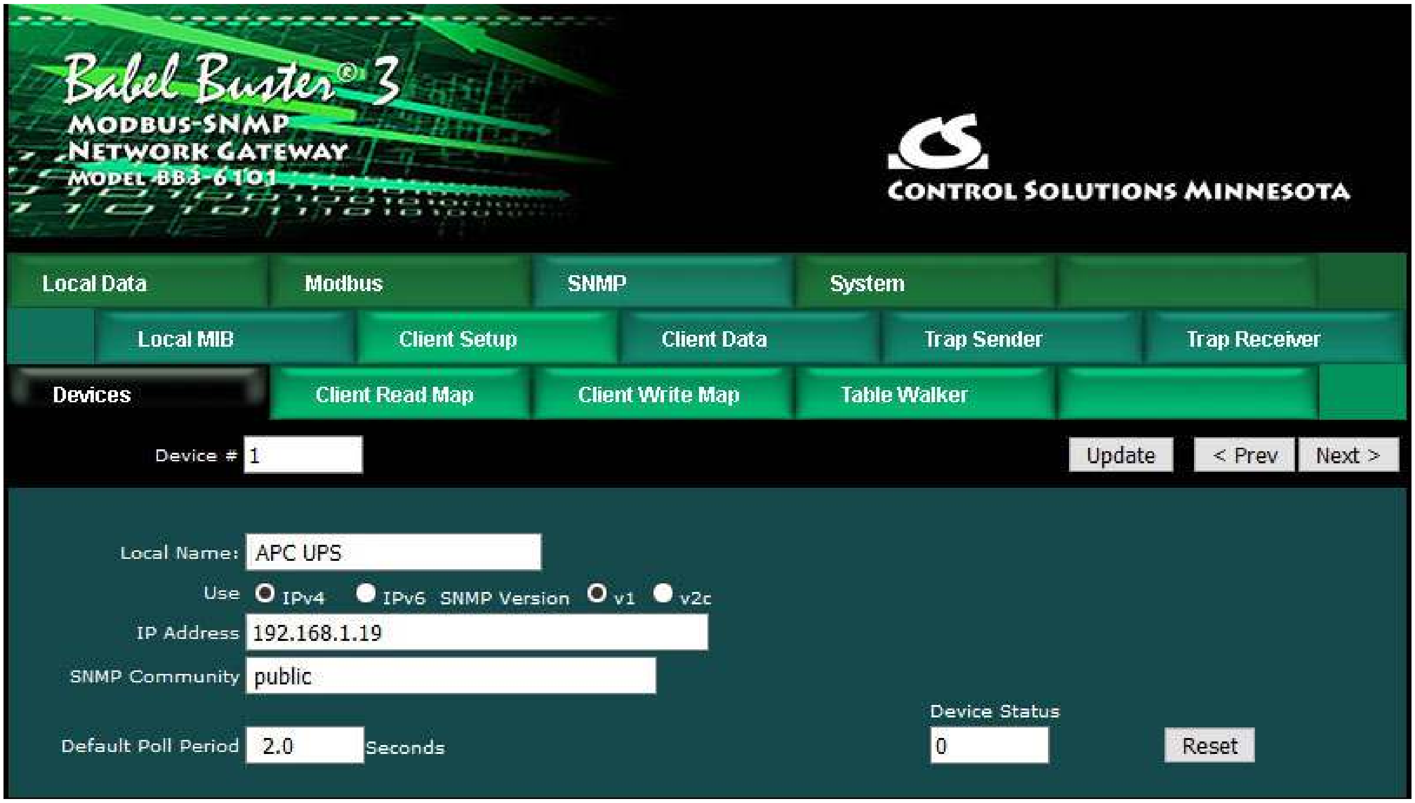
CONTROL SOLUTIONS MINNESOTA

11. 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's own MIB.

11.1 SNMP Device Configuration

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



Device # 1

Local Name: APC UPS

Use ◉ IPv4 ○ IPv6 SNMP Version ◉ v1 ○ v2c

IP Address 192.168.1.19

SNMP Community public

Default Poll Period 2.0 Seconds

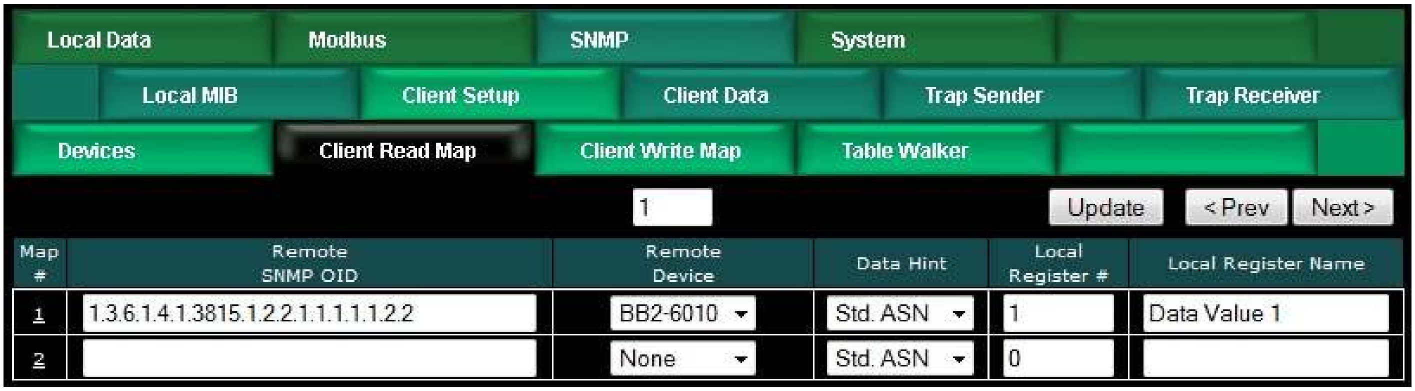
Device Status 0

Enter the IP address of the SNMP agent that the Babel Buster will send Get or Set requests to. You can enter either IPv4 or IPv6 addresses. 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 will periodically Get or Set the variables identified in the Read and Write Maps.

11.2 SNMP Client Read Maps (Get)

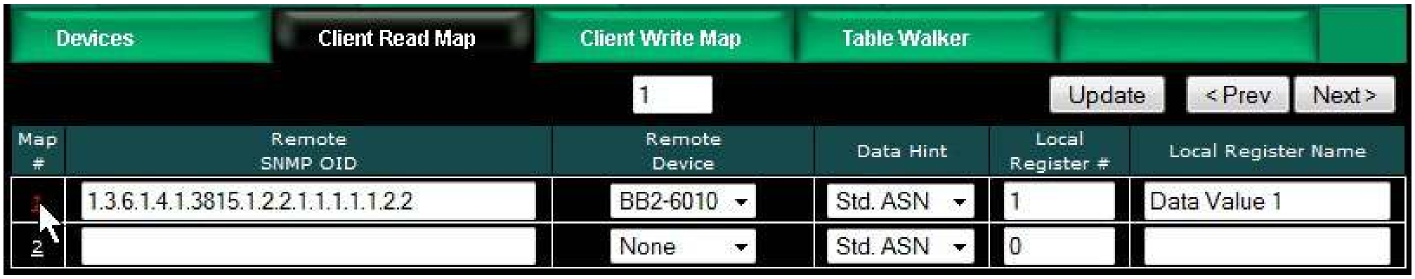
Reading or Getting data from another SNMP device requires setting up a Read Map.



Map #	Remote SNMP OID	Remote Device	Data Hint	Local Register #	Local Register Name
1	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.2	BB2-6010	Std. ASN	1	Data Value 1
2		None	Std. ASN	0	

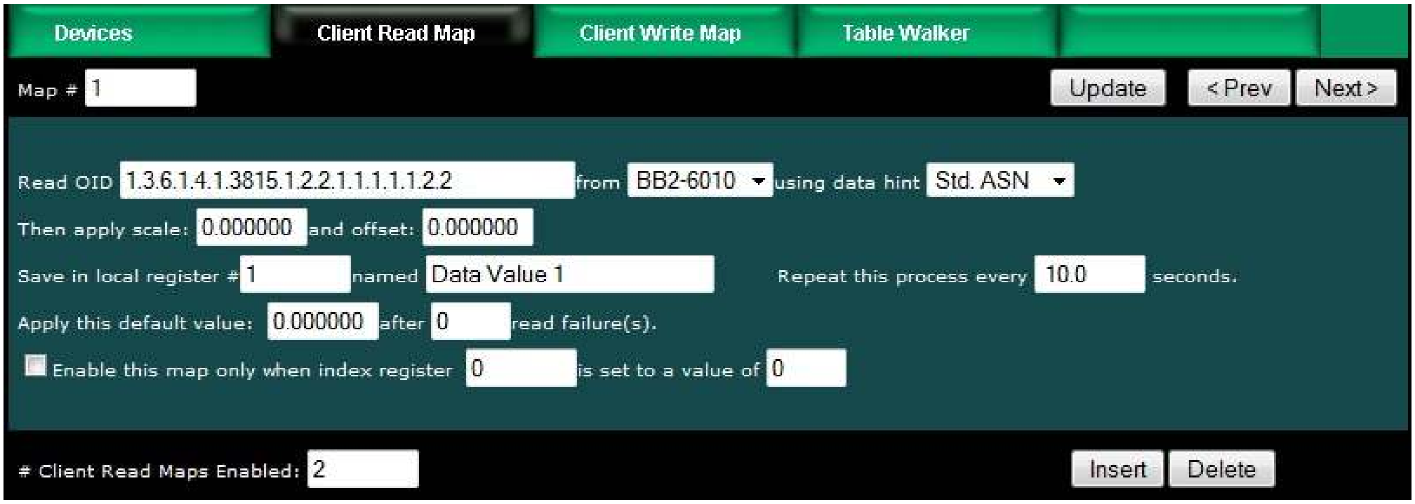
To configure a Read Map, enter the OID of the variable you wish to Get, select the SNMP device from the device list, and provide a local register number where the received data should be placed. If the remote device will be providing floating point as RFC 6340, select that in the Data Hint list, otherwise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



Map #	Remote SNMP OID	Remote Device	Data Hint	Local Register #	Local Register Name
1	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.2	BB2-6010	Std. ASN	1	Data Value 1
2		None	Std. ASN	0	

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.



Map # 1

Read OID 1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.2 from BB2-6010 using data hint Std. ASN

Then apply scale: 0.000000 and offset: 0.000000

Save in local register # 1 named Data Value 1 Repeat this process every 10.0 seconds.

Apply this default value: 0.000000 after 0 read failure(s).

□ Enable this map only when index register 0 is set to a value of 0

Client Read Maps Enabled: 2

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.

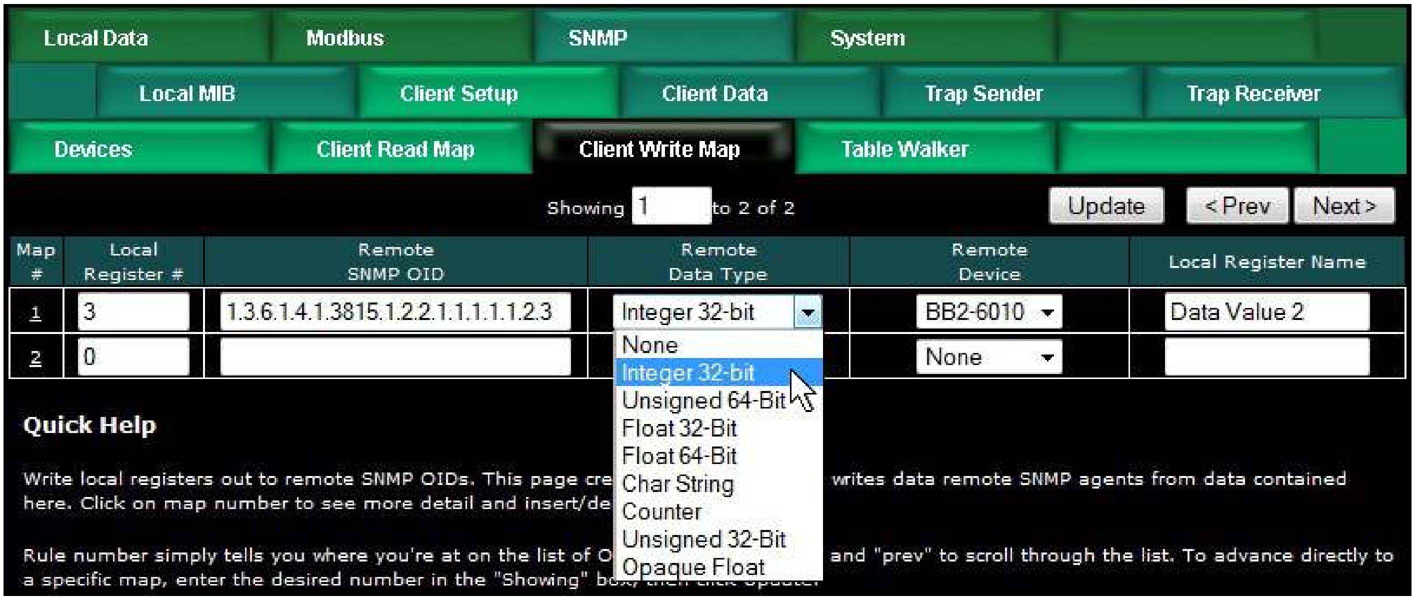
Delete will remove the rule number shown in the "Map #" box. Insert will insert a new rule before the rule number shown, and is used for placing rules between existing rules. It is not necessary to use Insert to add rules to the bottom of the list or to define any rule presently having zero for a source 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.

11.3 SNMP Client Write Maps (Set)

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



Showing 1 to 2 of 2

Map #	Local Register #	Remote SNMP OID	Remote Data Type	Remote Device	Local Register Name
1	3	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.3	Integer 32-bit	BB2-6010	Data Value 2
2	0			None	

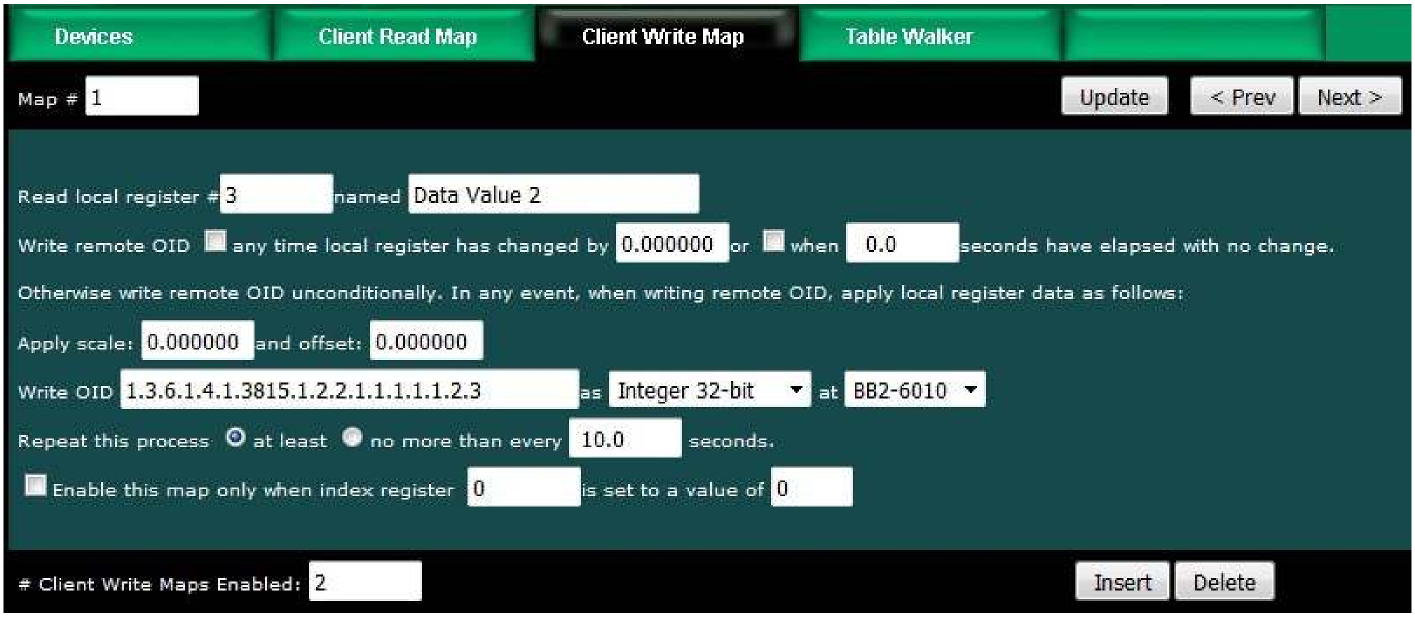
Quick Help

Write local registers out to remote SNMP OIDs. This page cre writes data remote SNMP agents from data contained here. Click on map number to see more detail and insert/de

Rule number simply tells you where you're at on the list of O and "prev" to scroll through the list. To advance directly to a specific map, enter the desired number in the "Showing" bo

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.



Map # 1

Read local register # 3 named Data Value 2

Write remote OID □ any time local register has changed by 0.000000 or □ when 0.0 seconds have elapsed with no change.

Otherwise write remote OID unconditionally. In any event, when writing remote OID, apply local register data as follows:

Apply scale: 0.000000 and offset: 0.000000

Write OID 1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.3 as Integer 32-bit at BB2-6010

Repeat this process ● at least ○ no more than every 10.0 seconds.

□ Enable this map only when index register 0 is set to a value of 0

Client Write Maps Enabled: 2

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.

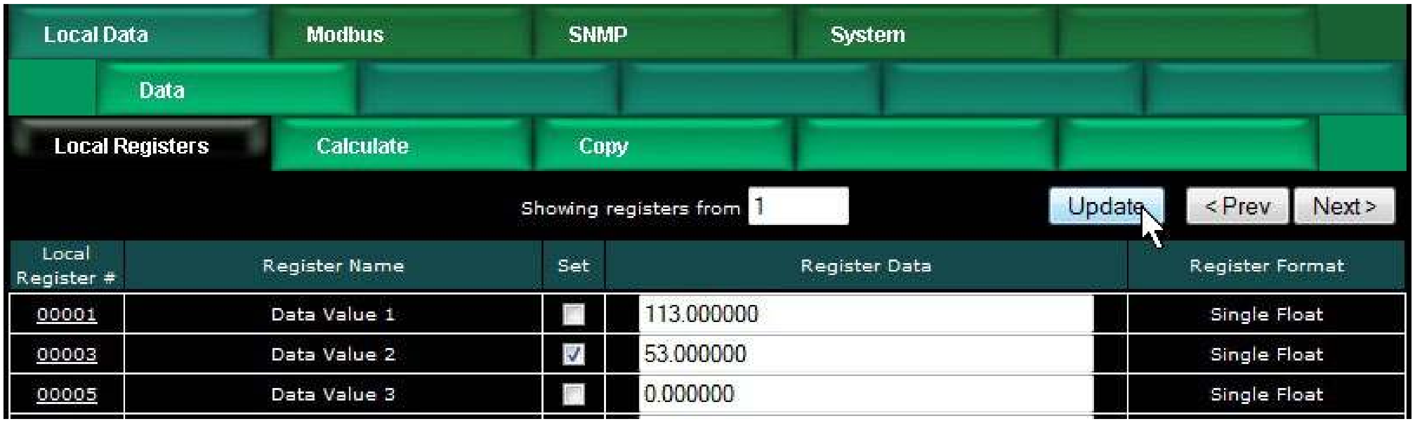
Enter the OID to write in that device. Select the data format that will be expected by the selected device at the OID given. Select a device from the device list. Only devices entered on the Devices tab will appear in this list.

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.



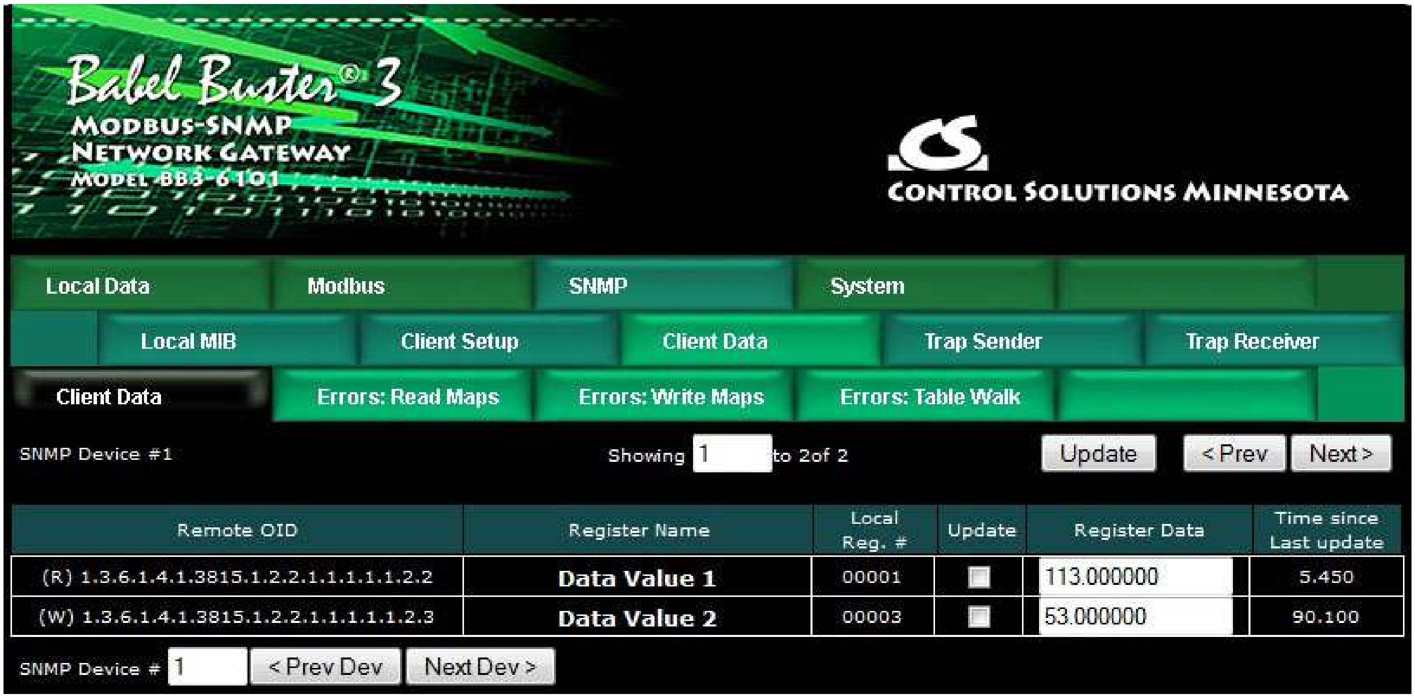
Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	☐	113.000000	Single Float
00003	Data Value 2	☑	53.000000	Single Float
00005	Data Value 3	☐	0.000000	Single Float

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.

11.4 SNMP Client Data Displayed by Agent

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



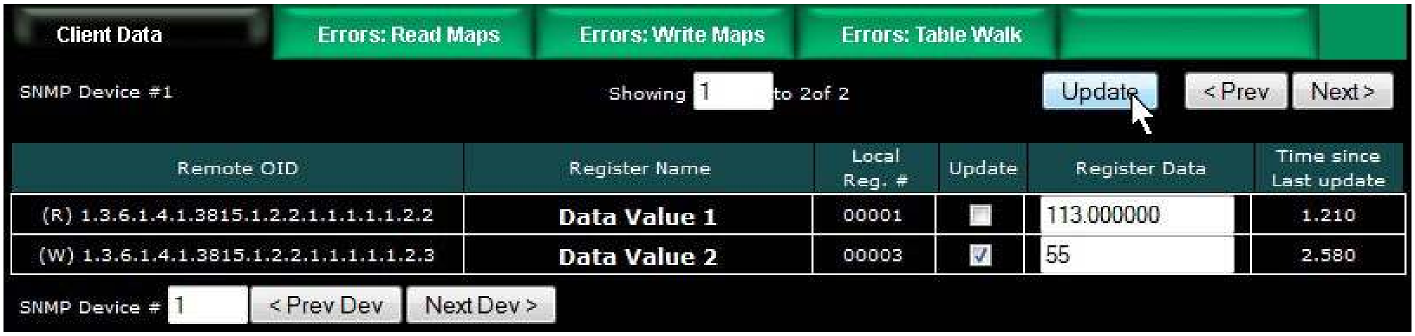
SNMP Device #1

Showing 1 to 2of 2

Remote OID	Register Name	Local Reg. #	Update	Register Data	Time since Last update
(R) 1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.2	Data Value 1	00001	☐	113.000000	5.450
(W) 1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.3	Data Value 2	00003	☐	53.000000	90.100

SNMP Device # 1

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.



SNMP Device #1

Showing 1 to 2of 2

Remote OID	Register Name	Local Reg. #	Update	Register Data	Time since Last update
(R) 1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.2	Data Value 1	00001	☐	113.000000	1.210
(W) 1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.3	Data Value 2	00003	☑	55	2.580

SNMP Device # 1

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.

11.5 Supported Data Formats

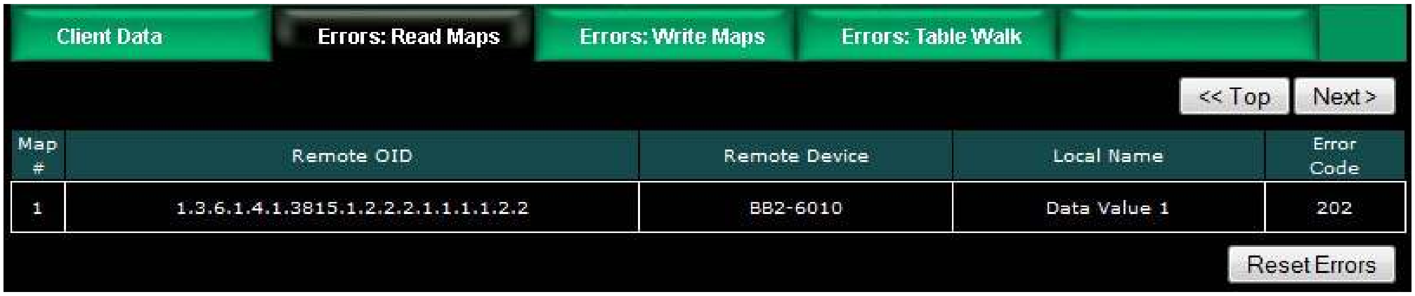
SNMP does not have a universally accepted representation for floating point. The one universally known data type is INTEGER. A commonly recomended means of transmitting floating point data is either as a scaled integer or as an ASCII character string. A well known but application specific implementation (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's own MIB refer to RFC 6340 encoding.

Data types recognized by Babel Buster 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:

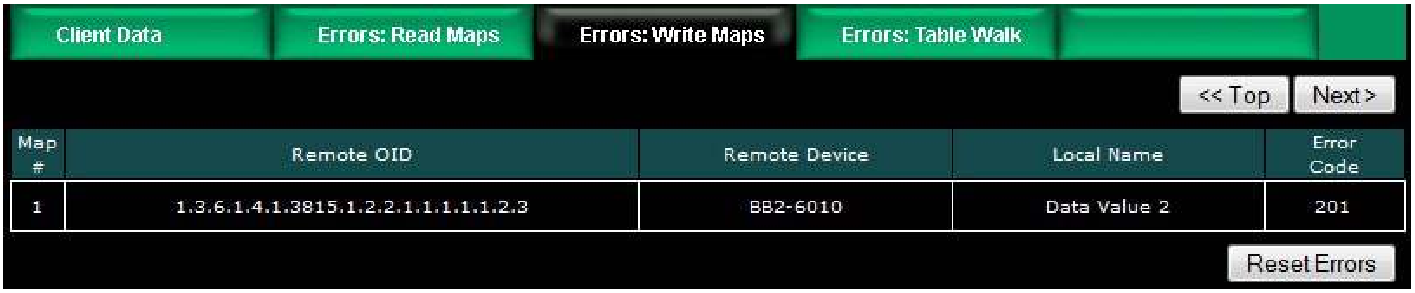
Integer 32-Bit	INTEGER	ASN_INTEGER
Unsigned 64-Bit	COUNTER64	(ASN_APPLICATION | 6)
Float 32-bit	OCTET STRING	ASN_OCTET_STR (length 4)
Float 64-bit	OCTET STRING	ASN_OCTET_STR (length 8)
Char String	OCTET STRING	ASN_OCTET_STR (length variable)
Counter	COUNTER	(ASN_APPLICATION | 1)
Unsigned 32-bit	UNSIGNED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

11.6 SNMP Errors

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



Map #	Remote OID	Remote Device	Local Name	Error Code
1	1.3.6.1.4.1.3815.1.2.2.2.1.1.1.1.2.2	BB2-6010	Data Value 1	202



Map #	Remote OID	Remote Device	Local Name	Error Code
1	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.3	BB2-6010	Data Value 2	201

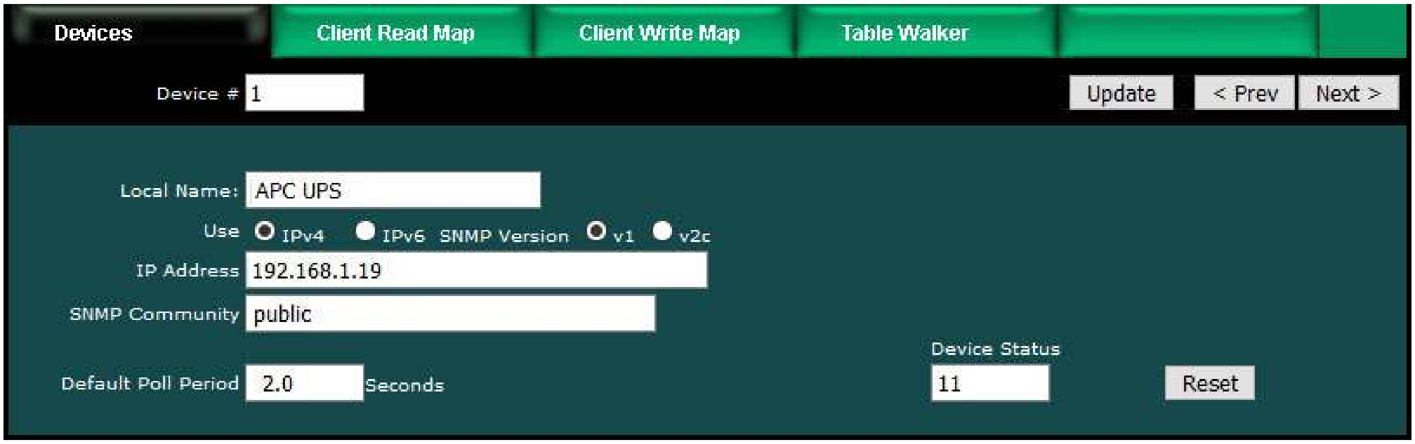
Error codes that may be returned by the SNMP client are as follows:

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 # 1

Local Name: APC UPS

Use ● IPv4 ○ IPv6 SNMP Version ● v1 ○ v2c

IP Address 192.168.1.19

SNMP Community public

Default Poll Period 2.0 Seconds

Device Status 11

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:

104 = Connection reset by peer
111 = Connection refused
113 = Connection aborted
114 = Network is unreachable

115 = Network interface not configured
116 = Connection timed out
118 = Host is unreachable
125 = Address not available

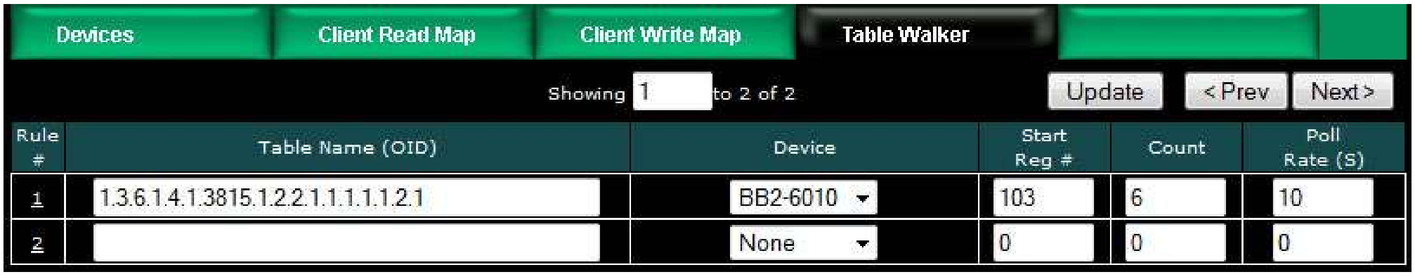
12. Configuring SNMP Table Walker

12.1 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).

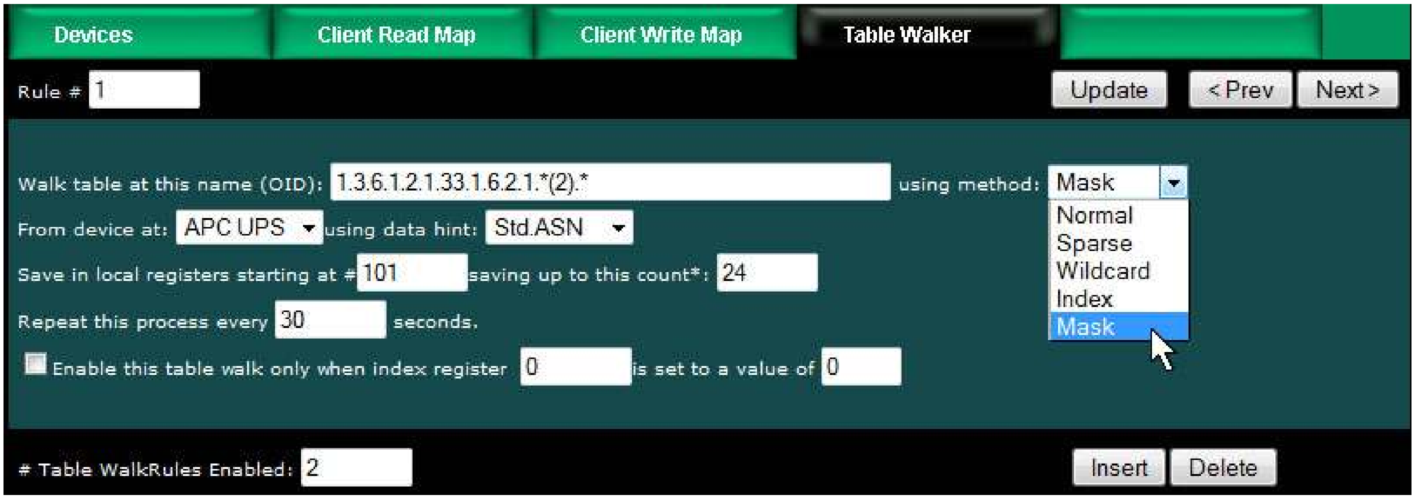
The Babel Buster table walker will scan the table, picking out one column per walk rule. For each row in the table, that column will be retrieved and its data placed into the local register provided. The first data value retrieved will be placed into the starting local register, and the walk will continue 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).



Rule #	Table Name (OID)	Device	Start Reg #	Count	Poll Rate (S)
1	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.1	BB2-6010	103	6	10
2		None	0	0	0

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.



Rule # 1

Walk table at this name (OID): 1.3.6.1.2.1.33.1.6.2.1.*(2).* using method: Mask

From device at: APC UPS using data hint: Std.ASN

Save in local registers starting at # 101 saving up to this count*: 24

Repeat this process every 30 seconds.

☐ Enable this table walk only when index register 0 is set to a value of 0

Table WalkRules Enabled: 2

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.

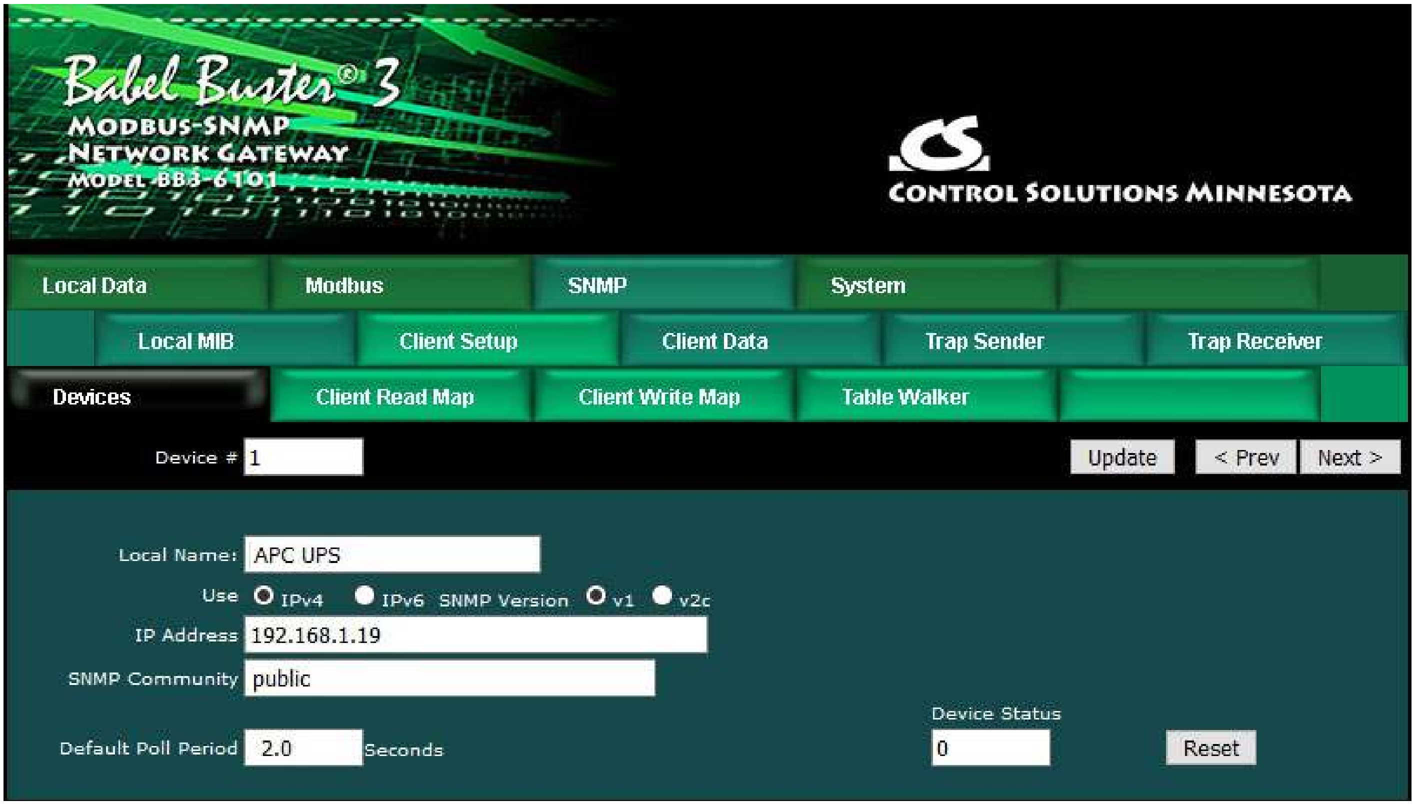
Method "**Index**" will walk the table, but expect to find that values are OIDs. In other words, the table name is an OID, but the contents of the variable at that name will be another OID. The result is that the OID index (last field of OID) from the value will be used as the offset to calculate which 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.)

12.2 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.



Device # 1

Local Name: APC UPS

Use ● IPv4 ○ IPv6 SNMP Version ● v1 ○ v2c

IP Address 192.168.1.19

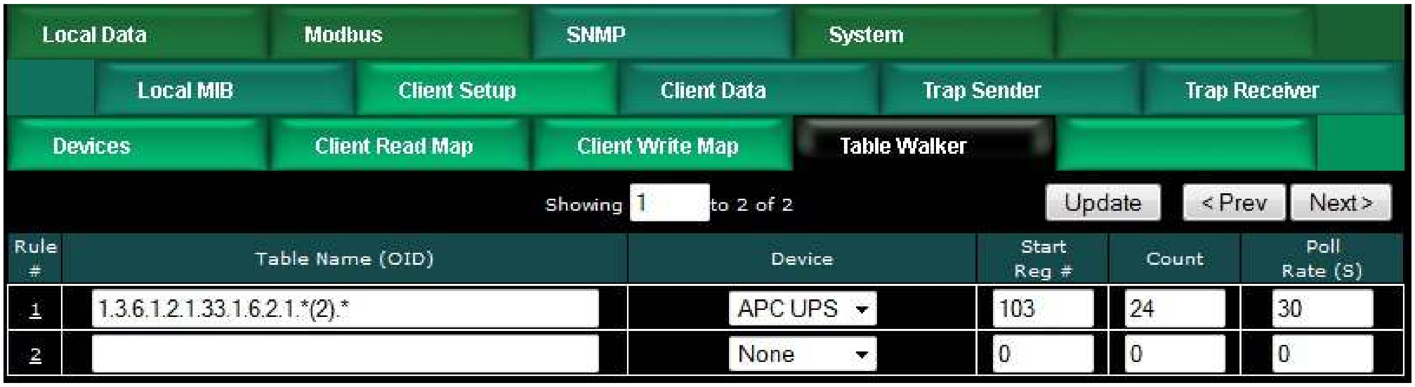
SNMP Community public

Default Poll Period 2.0 Seconds

Device Status 0

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.



Showing 1 to 2 of 2

Rule #	Table Name (OID)	Device	Start Reg #	Count	Poll Rate (S)
1	1.3.6.1.2.1.33.1.6.2.1.*(2).*	APC UPS	103	24	30
2		None	0	0	0

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.

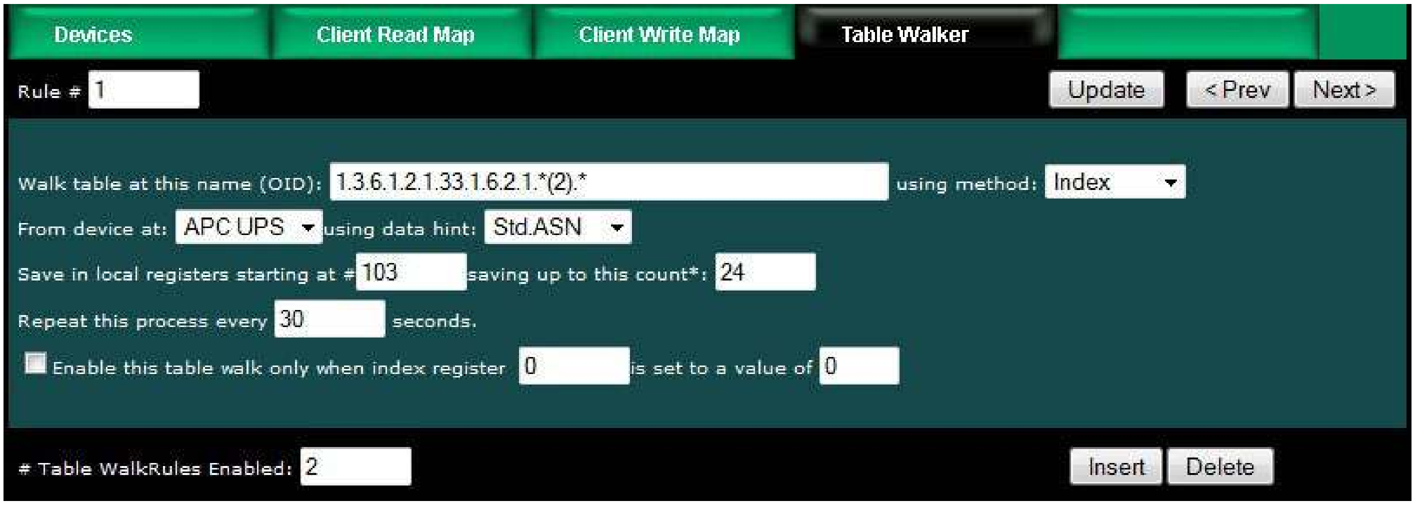
* Count has a different meaning when method is "Mask". In this case, the starting register is the only register affected, and count is the number of bit positions in that register that will be processed. It also implies that this many OIDs in the table will be read.

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:



Rule # 1

Walk table at this name (OID): 1.3.6.1.2.1.33.1.6.2.1.*(2).* using method: Index

From device at: APC UPS using data hint: Std.ASN

Save in local registers starting at # 103 saving up to this count*: 24

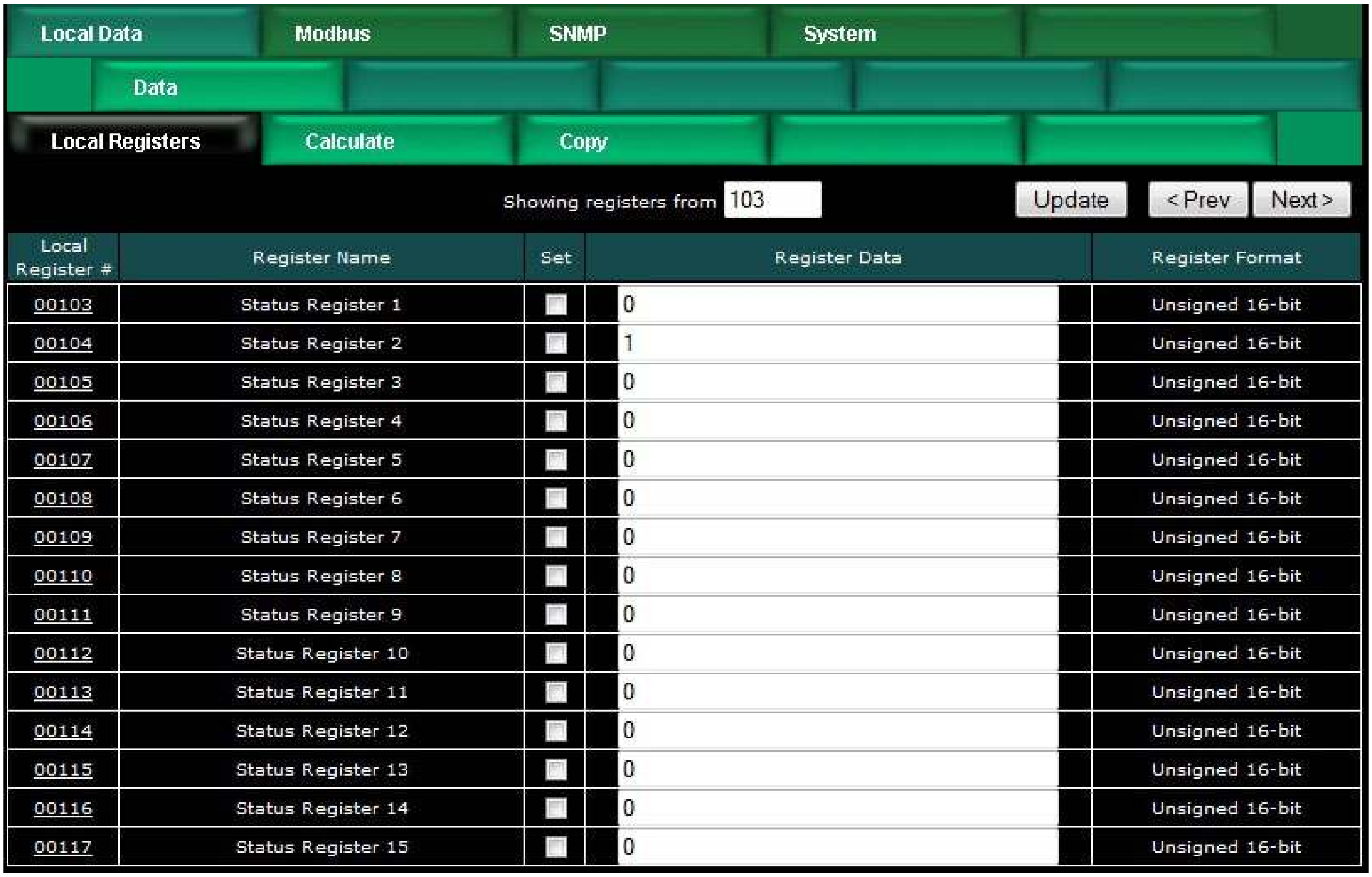
Repeat this process every 30 seconds.

□ Enable this table walk only when index register 0 is set to a value of 0

Table WalkRules Enabled: 2

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.

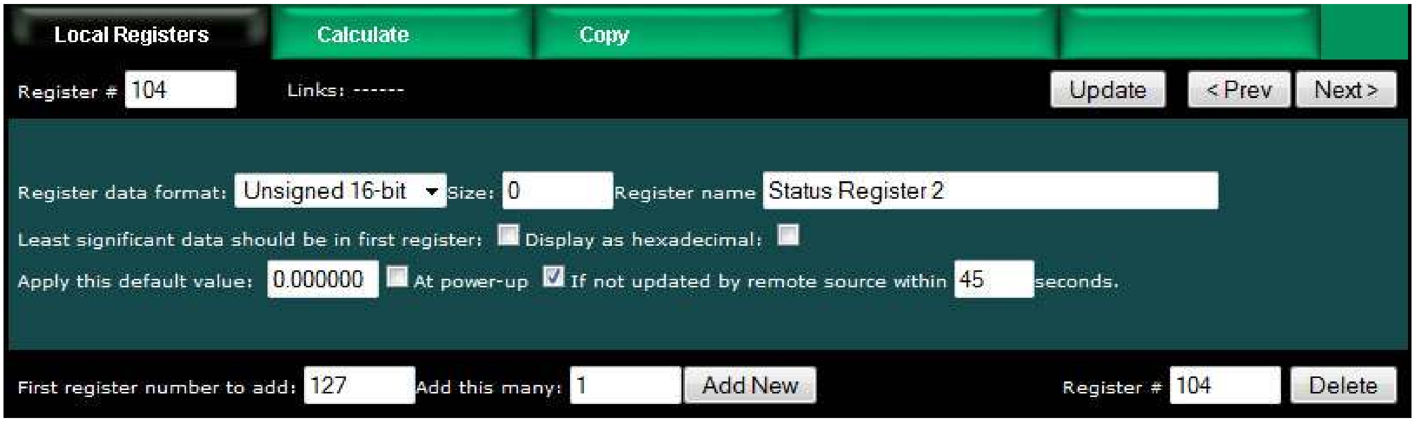
Note that while the minimum register size you can allocate in the Babel Buster is a 16-bit register, you can read the registers as Modbus coils if the gateway is operating as Modbus slave. In the example below, holding register 104 will return a value of 1, but coil 104 will also return a status of "on" 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.)



Showing registers from 103

Local Register #	Register Name	Set	Register Data	Register Format
00103	Status Register 1	□	0	Unsigned 16-bit
00104	Status Register 2	□	1	Unsigned 16-bit
00105	Status Register 3	□	0	Unsigned 16-bit
00106	Status Register 4	□	0	Unsigned 16-bit
00107	Status Register 5	□	0	Unsigned 16-bit
00108	Status Register 6	□	0	Unsigned 16-bit
00109	Status Register 7	□	0	Unsigned 16-bit
00110	Status Register 8	□	0	Unsigned 16-bit
00111	Status Register 9	□	0	Unsigned 16-bit
00112	Status Register 10	□	0	Unsigned 16-bit
00113	Status Register 11	□	0	Unsigned 16-bit
00114	Status Register 12	□	0	Unsigned 16-bit
00115	Status Register 13	□	0	Unsigned 16-bit
00116	Status Register 14	□	0	Unsigned 16-bit
00117	Status Register 15	□	0	Unsigned 16-bit

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.



Register # 104 Links: ------

Register data format: Unsigned 16-bit Size: 0 Register name Status Register 2

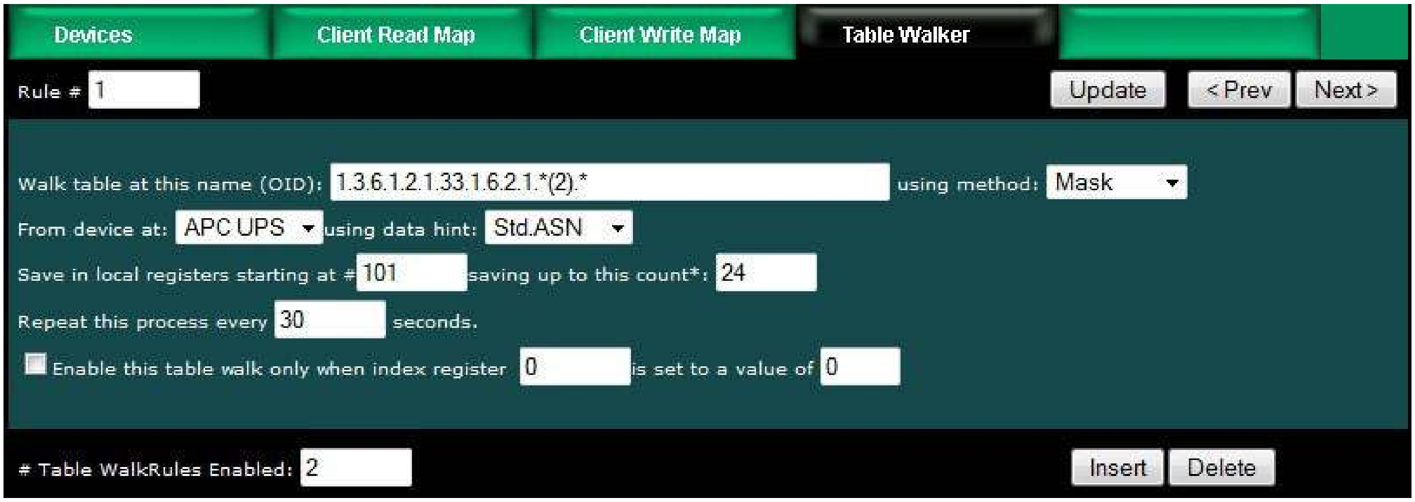
Least significant data should be in first register: □ Display as hexadecimal: □

Apply this default value: 0.000000 □ At power-up ☑ If not updated by remote source within 45 seconds.

First register number to add: 127 Add this many: 1 Register # 104

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.



Rule # 1

Walk table at this name (OID): 1.3.6.1.2.1.33.1.6.2.1.*(2).* using method: Mask

From device at: APC UPS using data hint: Std.ASN

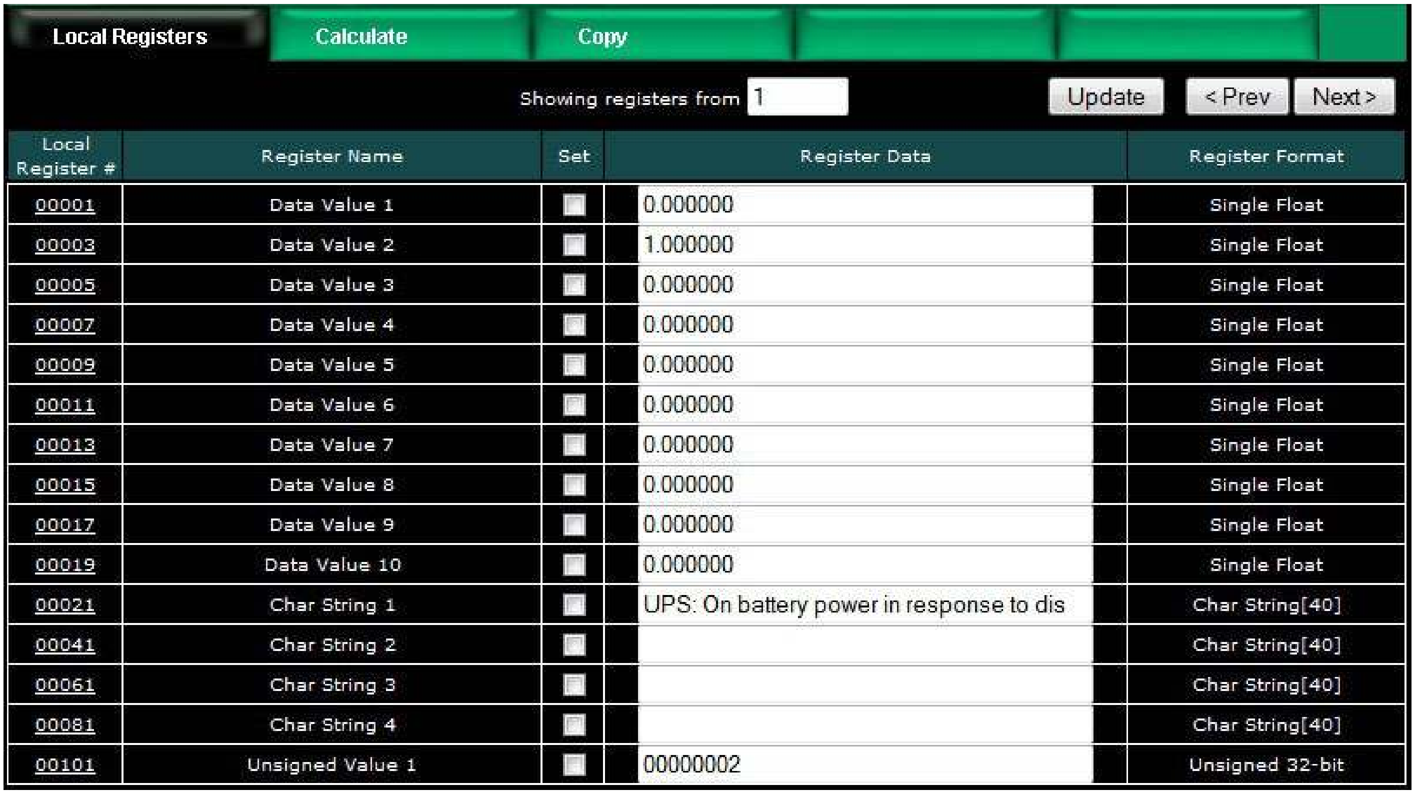
Save in local registers starting at # 101 saving up to this count*: 24

Repeat this process every 30 seconds.

□ Enable this table walk only when index register 0 is set to a value of 0

Table WalkRules Enabled: 2

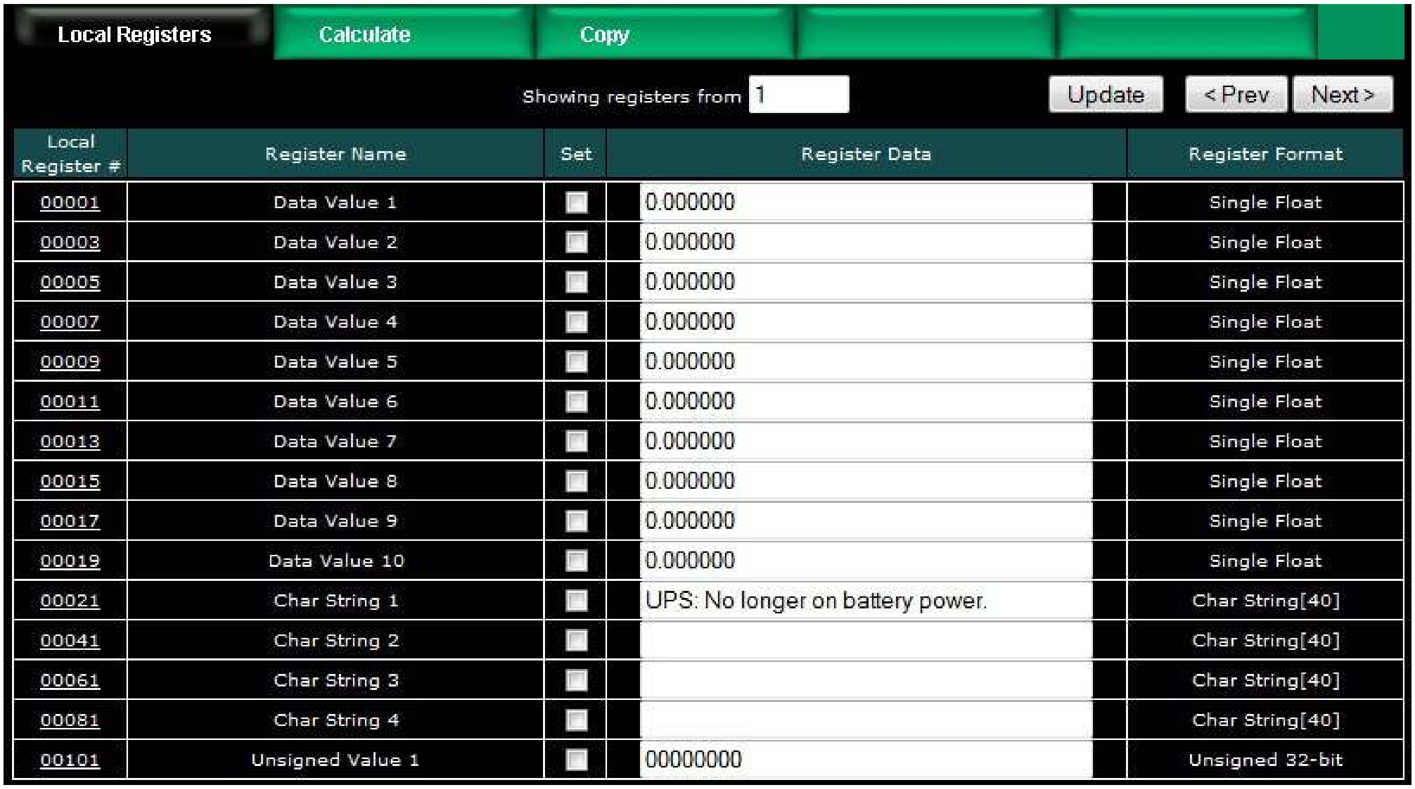
The resulting "UPS on battery" alarm, being alarm #2, sets bit 1 in the packed bit register 101 in our example. Alarm #1 sets the least significant bit, alarm #2 the next bit, and so on.



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	0.000000	Single Float
00003	Data Value 2	□	1.000000	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: On battery power in response to dis	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	□	00000002	Unsigned 32-bit

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.

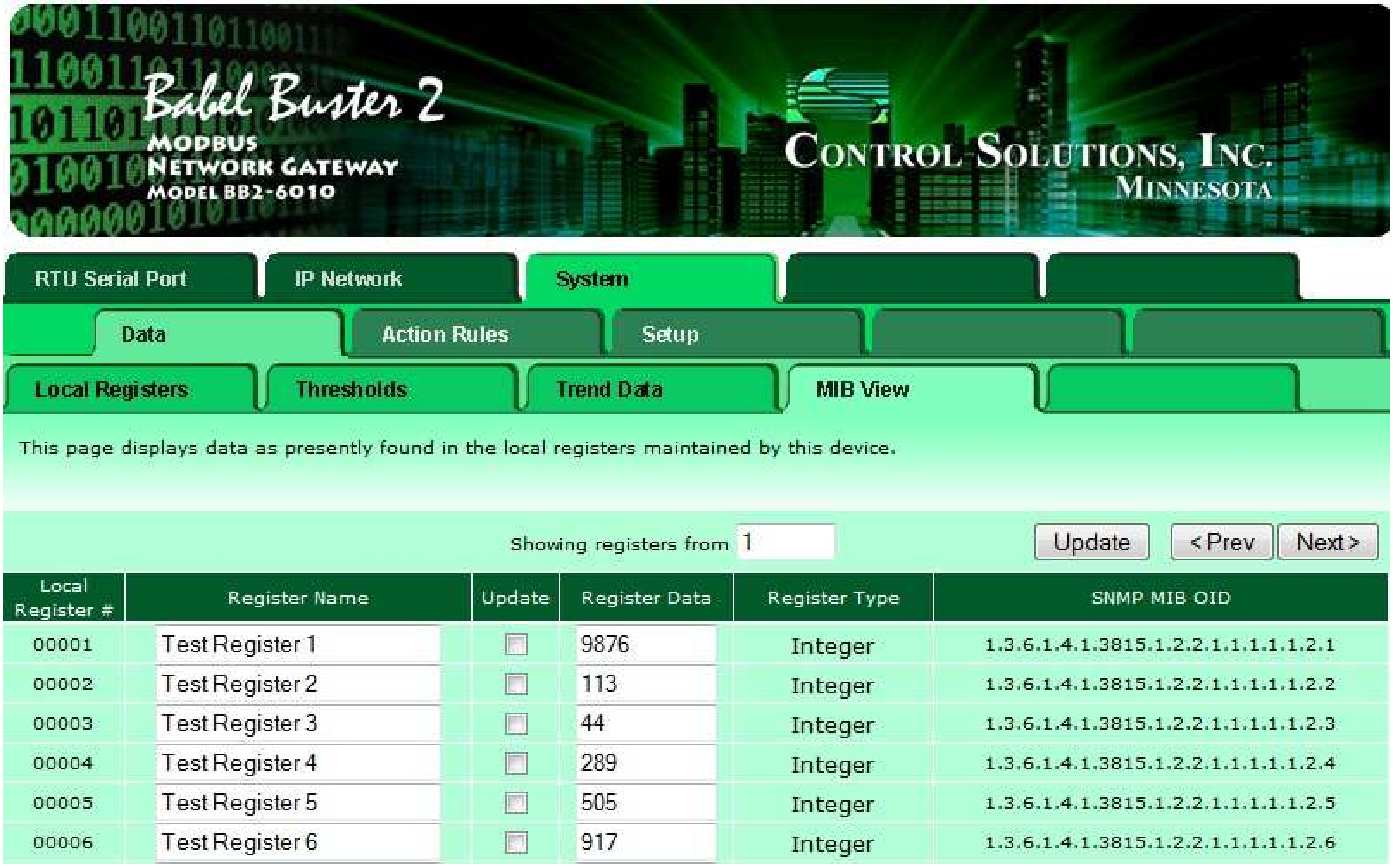


Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	0.000000	Single Float
00003	Data Value 2	□	0.000000	Single Float
00005	Data Value 3	□	0.000000	Single Float
00007	Data Value 4	□	0.000000	Single Float
00009	Data Value 5	□	0.000000	Single Float
00011	Data Value 6	□	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 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 BB3-6101/MX-61.

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.

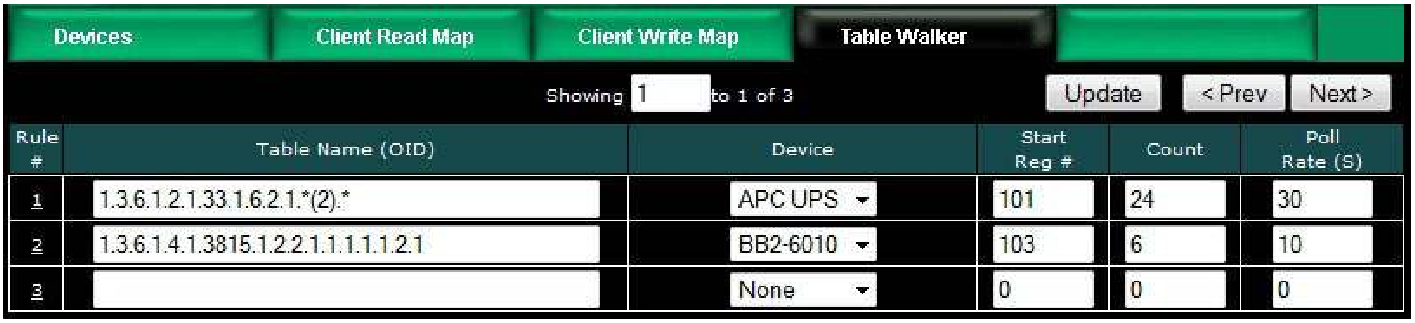


This page displays data as presently found in the local registers maintained by this device.

Showing registers from 1

Local Register #	Register Name	Update	Register Data	Register Type	SNMP MIB OID
00001	Test Register 1	□	9876	Integer	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.1
00002	Test Register 2	□	113	Integer	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.2
00003	Test Register 3	□	44	Integer	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.3
00004	Test Register 4	□	289	Integer	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.4
00005	Test Register 5	□	505	Integer	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.5
00006	Test Register 6	□	917	Integer	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.6

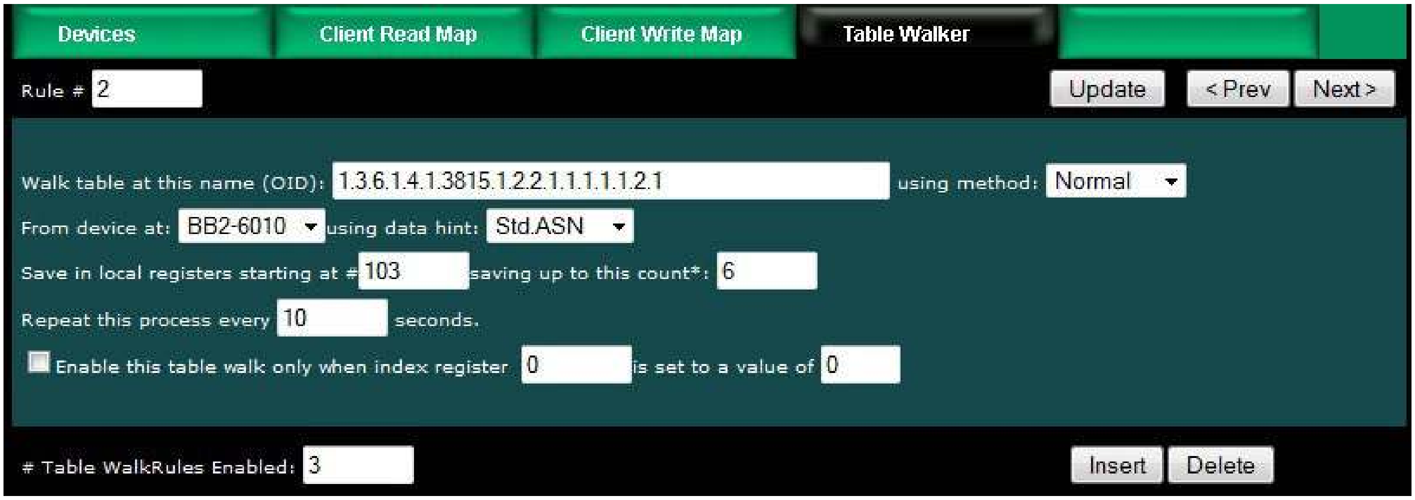
The screen shot of the walk rule list shows the previous example plus this example added as a second rule. Because this is a simple "Normal" table walk, everything needed can be entered on the tabular rule list. The first OID to be read is given as the table name. Select the device, starting local register 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.)



Showing 1 to 1 of 3

Rule #	Table Name (OID)	Device	Start Reg #	Count	Poll Rate (S)
1	1.3.6.1.2.1.33.1.6.2.1.*(2).*	APC UPS	101	24	30
2	1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.1	BB2-6010	103	6	10
3		None	0	0	0

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.



Rule # 2

Walk table at this name (OID): 1.3.6.1.4.1.3815.1.2.2.1.1.1.1.1.2.1 using method: Normal

From device at: BB2-6010 using data hint: Std.ASN

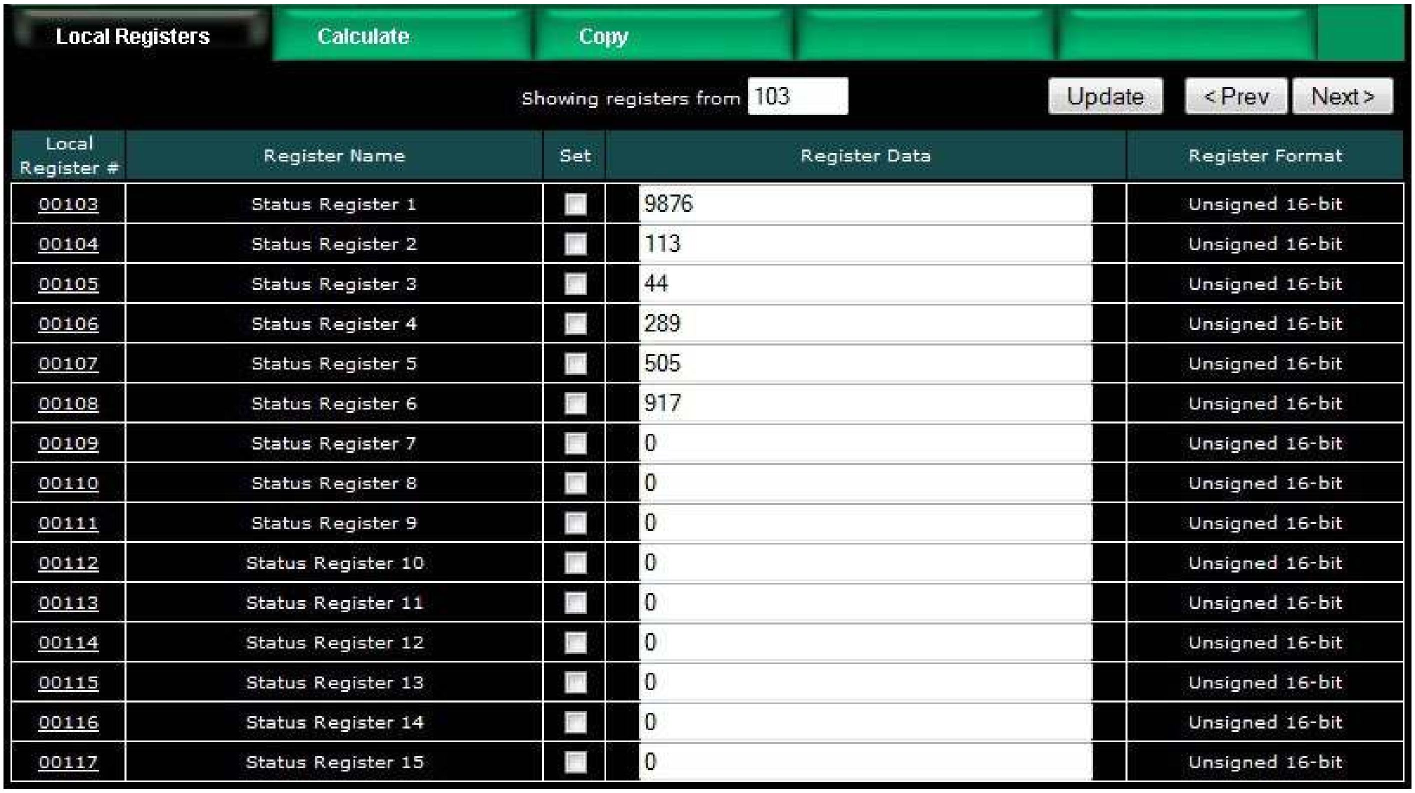
Save in local registers starting at # 103 saving up to this count*: 6

Repeat this process every 10 seconds.

□ Enable this table walk only when index register 0 is set to a value of 0

Table WalkRules Enabled: 3

Given the screen shot of the BB2-6010 MIB View, here is what we expect to see in the Babel Buster BB3-6101/MX-61 when the table walk starts running.

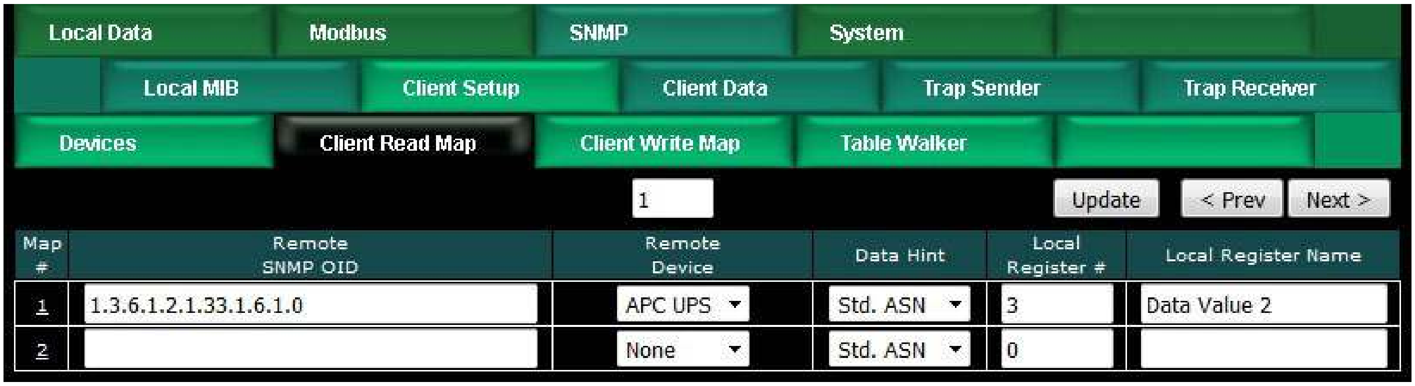


Showing registers from 103

Local Register #	Register Name	Set	Register Data	Register Format
00103	Status Register 1	□	9876	Unsigned 16-bit
00104	Status Register 2	□	113	Unsigned 16-bit
00105	Status Register 3	□	44	Unsigned 16-bit
00106	Status Register 4	□	289	Unsigned 16-bit
00107	Status Register 5	□	505	Unsigned 16-bit
00108	Status Register 6	□	917	Unsigned 16-bit
00109	Status Register 7	□	0	Unsigned 16-bit
00110	Status Register 8	□	0	Unsigned 16-bit
00111	Status Register 9	□	0	Unsigned 16-bit
00112	Status Register 10	□	0	Unsigned 16-bit
00113	Status Register 11	□	0	Unsigned 16-bit
00114	Status Register 12	□	0	Unsigned 16-bit
00115	Status Register 13	□	0	Unsigned 16-bit
00116	Status Register 14	□	0	Unsigned 16-bit
00117	Status Register 15	□	0	Unsigned 16-bit

Table Walk Example 4:

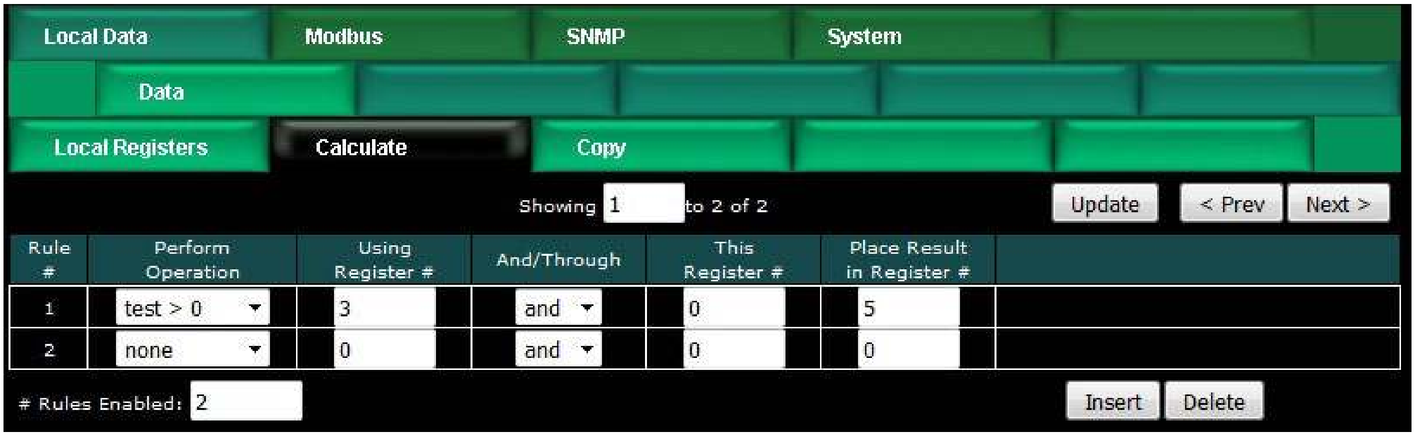
The following example elaborates on the RFC 1628 alarm table walk originally illustrated above. We are now going to walk the alarm table conditionally. Namely, we will only walk the table when we know there are alarms in the table. There is a variable in the RFC 1628 MIB that tells us how many alarm 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.



1

Map #	Remote SNMP OID	Remote Device	Data Hint	Local Register #	Local Register Name
1	1.3.6.1.2.1.33.1.6.1.0	APC UPS	Std. ASN	3	Data Value 2
2		None	Std. ASN	0	

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.

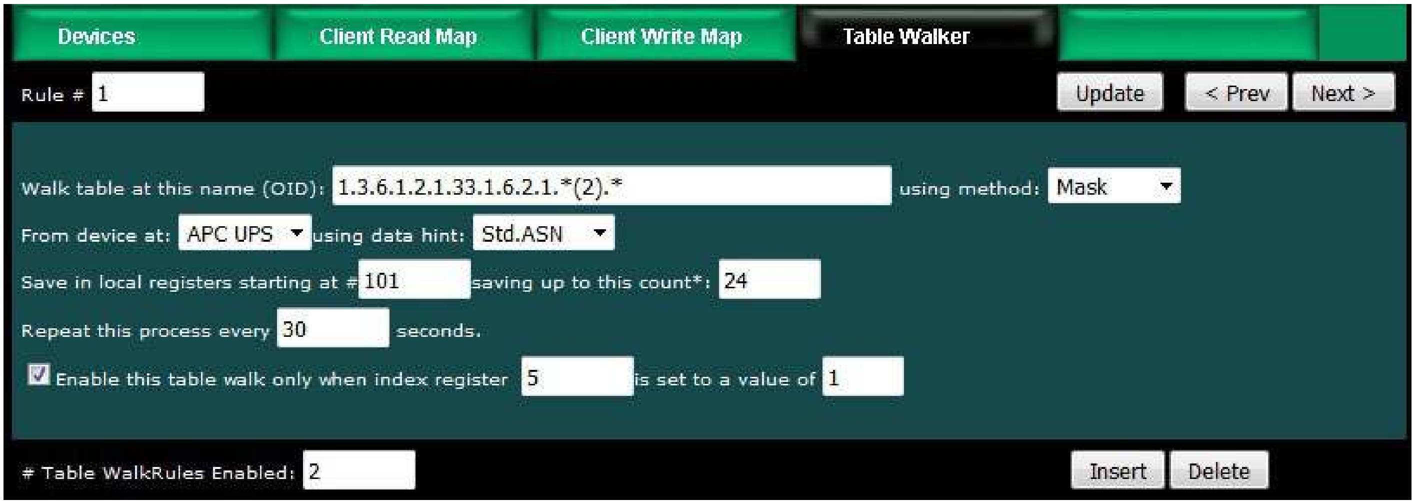


Showing 1 to 2 of 2

Rule #	Perform Operation	Using Register #	And/Through	This Register #	Place Result in Register #
1	test > 0	3	and	0	5
2	none	0	and	0	0

Rules Enabled: 2

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.



Rule # 1

Walk table at this name (OID): 1.3.6.1.2.1.33.1.6.2.1.*(2).* using method: Mask

From device at: APC UPS using data hint: Std.ASN

Save in local registers starting at # 101 saving up to this count*: 24

Repeat this process every 30 seconds.

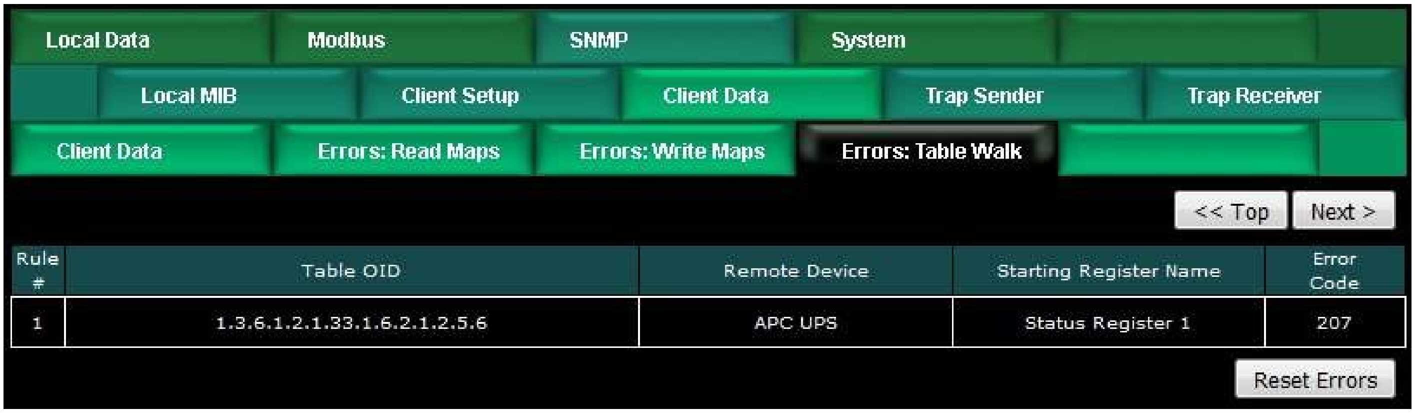
☑ Enable this table walk only when index register 5 is set to a value of 1

Table WalkRules Enabled: 2

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.

12.3 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.



Rule #	Table OID	Remote Device	Starting Register Name	Error Code
1	1.3.6.1.2.1.33.1.6.2.1.2.5.6	APC UPS	Status Register 1	207

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.

Babel Buster® 3
MODEL BB3-6101
MODEL MX-61
NETWORK GATEWAY

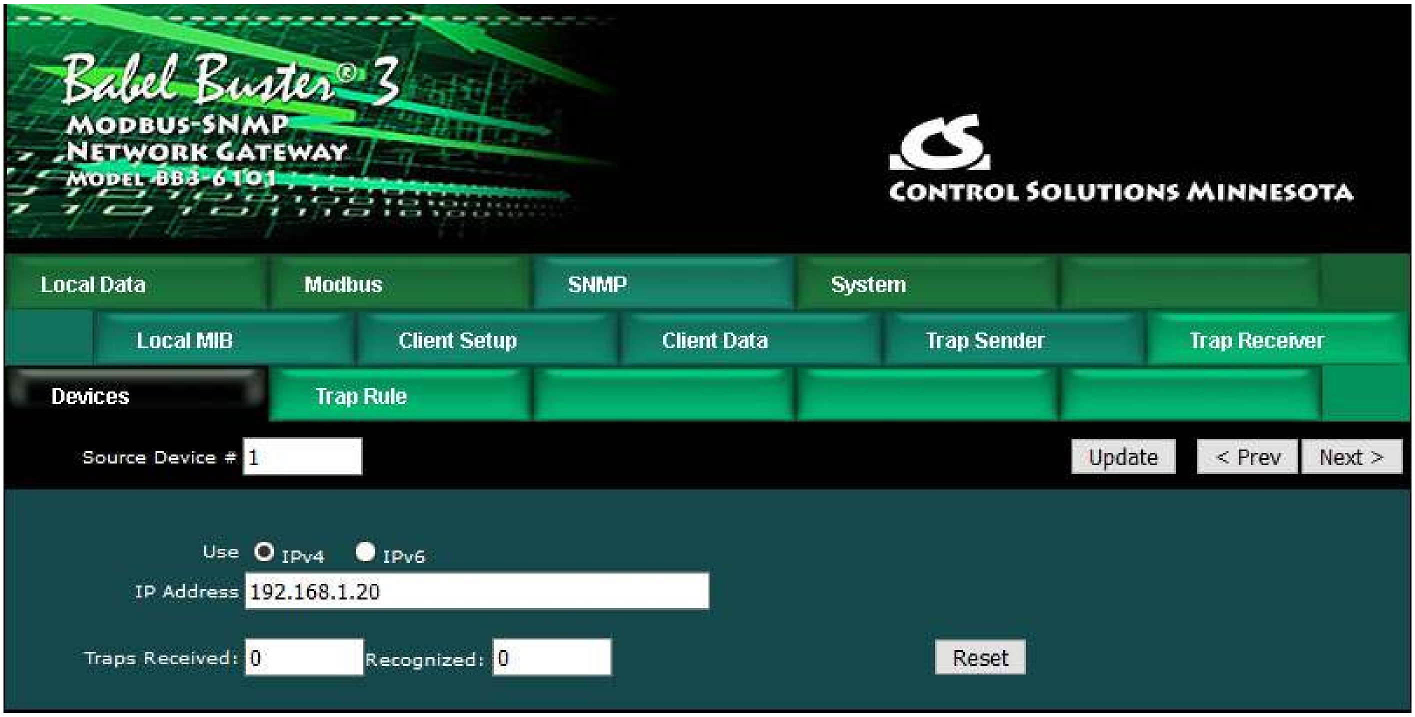
CONTROL SOLUTIONS MINNESOTA

13. Configuring SNMP Trap Receiver

13.1 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.



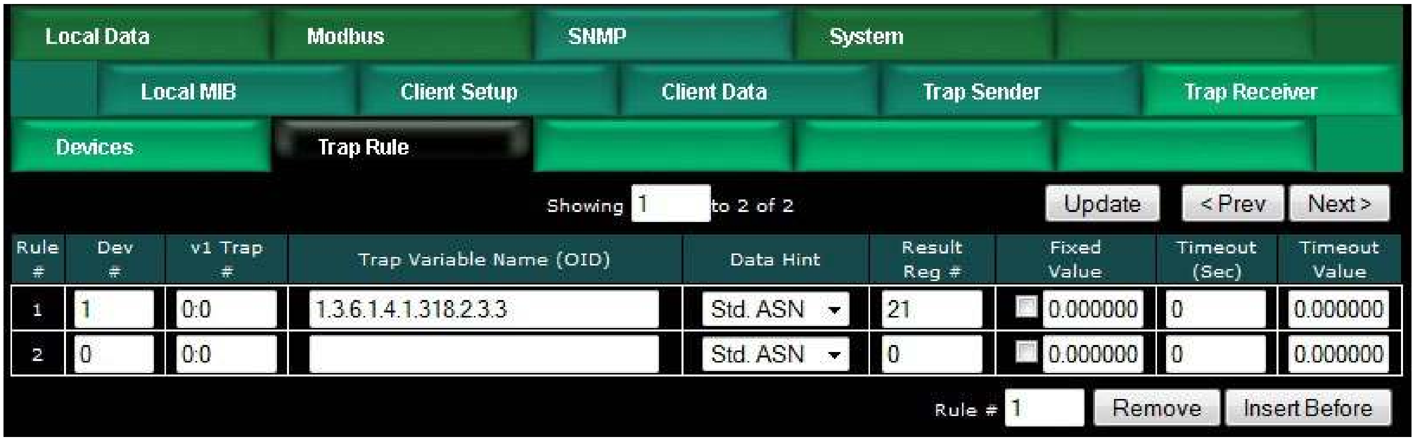
Source Device # 1

Use ● IPv4 ○ IPv6

IP Address 192.168.1.20

Traps Received: 0 Recognized: 0

13.2 Trap Receive Rules



Showing 1 to 2 of 2

Rule #	Dev #	v1 Trap #	Trap Variable Name (OID)	Data Hint	Result Reg #	Fixed Value	Timeout (Sec)	Timeout Value
1	1	0:0	1.3.6.1.4.1.318.2.3.3	Std. ASN	21	□ 0.000000	0	0.000000
2	0	0:0		Std. ASN	0	□ 0.000000	0	0.000000

Rule # 1

Rules for recognizing received traps are defined on this page. 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. 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.

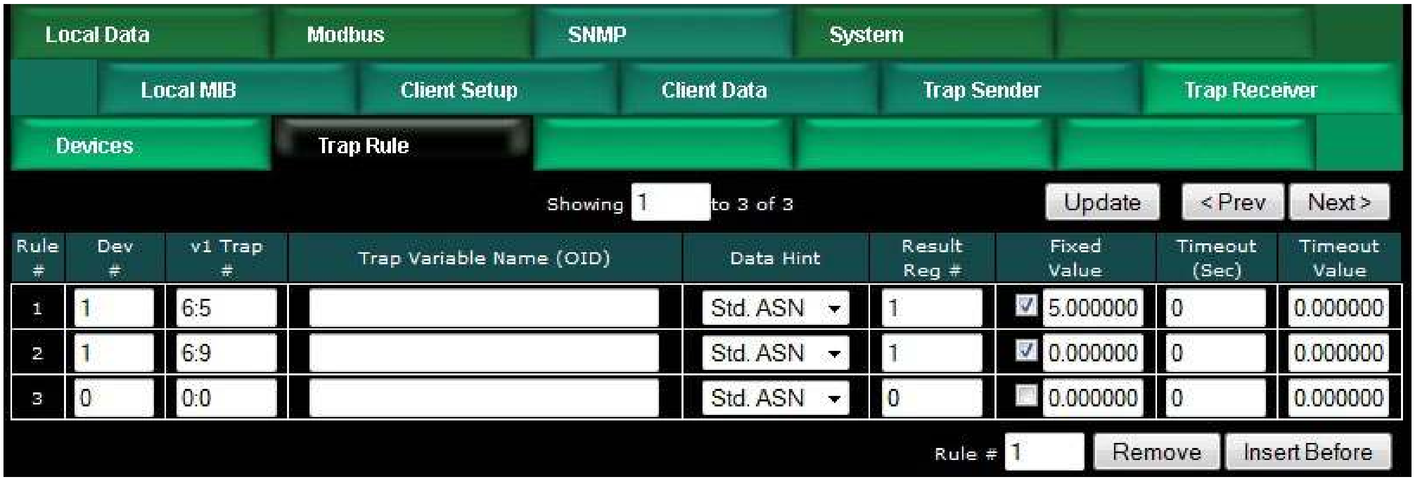
Timeout is optional, and will be in effect if a non-zero time in seconds is provided. If given, then the timeout value will be placed into the result register after this amount of time has elapsed since receiving the trap. If another of the same trap is received, the timer will be reset. Only after this 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.

13.3 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.

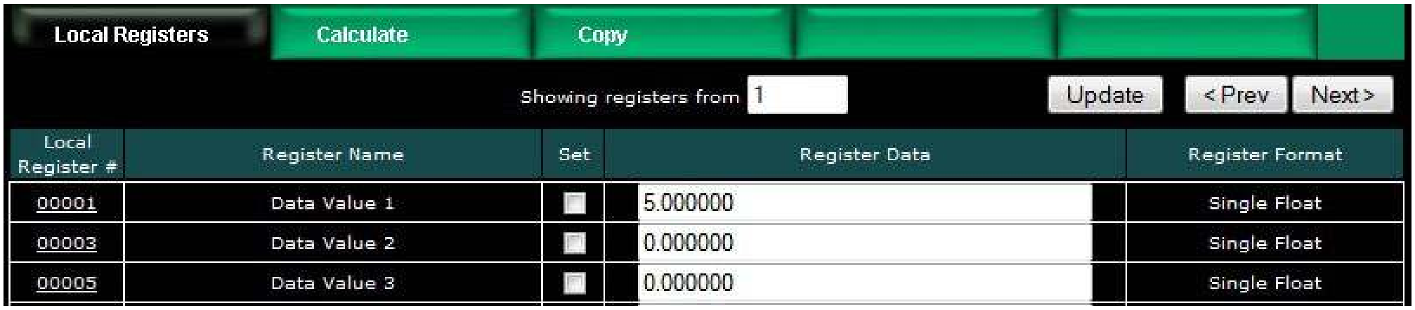


Showing 1 to 3 of 3

Rule #	Dev #	v1 Trap #	Trap Variable Name (OID)	Data Hint	Result Reg #	Fixed Value	Timeout (Sec)	Timeout Value
1	1	6:5		Std. ASN	1	☑ 5.000000	0	0.000000
2	1	6:9		Std. ASN	1	☑ 0.000000	0	0.000000
3	0	0:0		Std. ASN	0	□ 0.000000	0	0.000000

Rule # 1

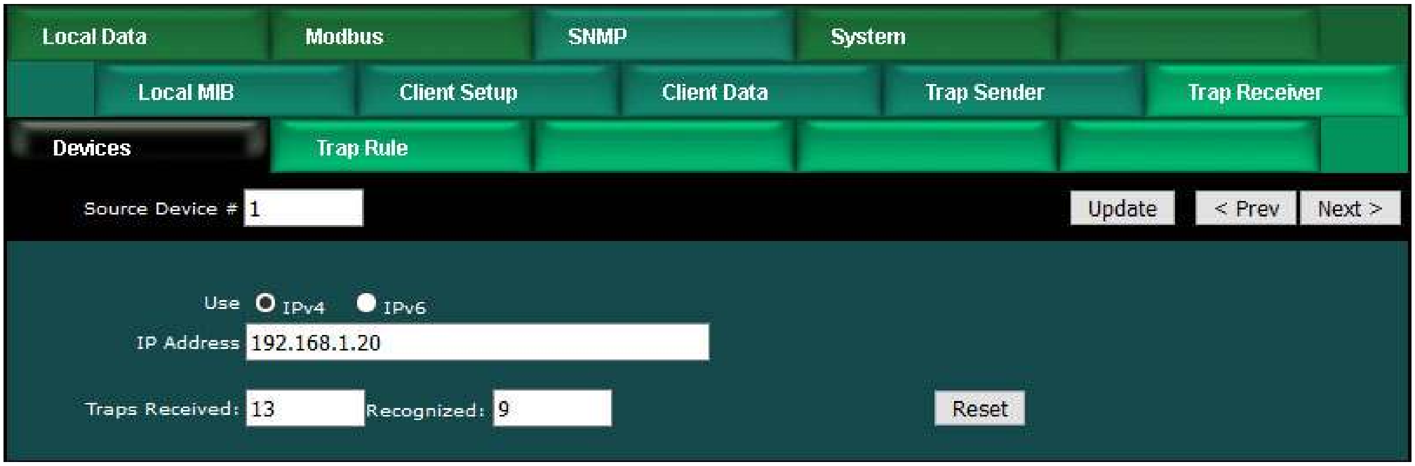
The resulting local register when utility power to the UPS is cut off in this example appears as follows:



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	□	5.000000	Single Float
00003	Data Value 2	□	0.000000	Single Float
00005	Data Value 3	□	0.000000	Single Float

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.



Source Device # 1

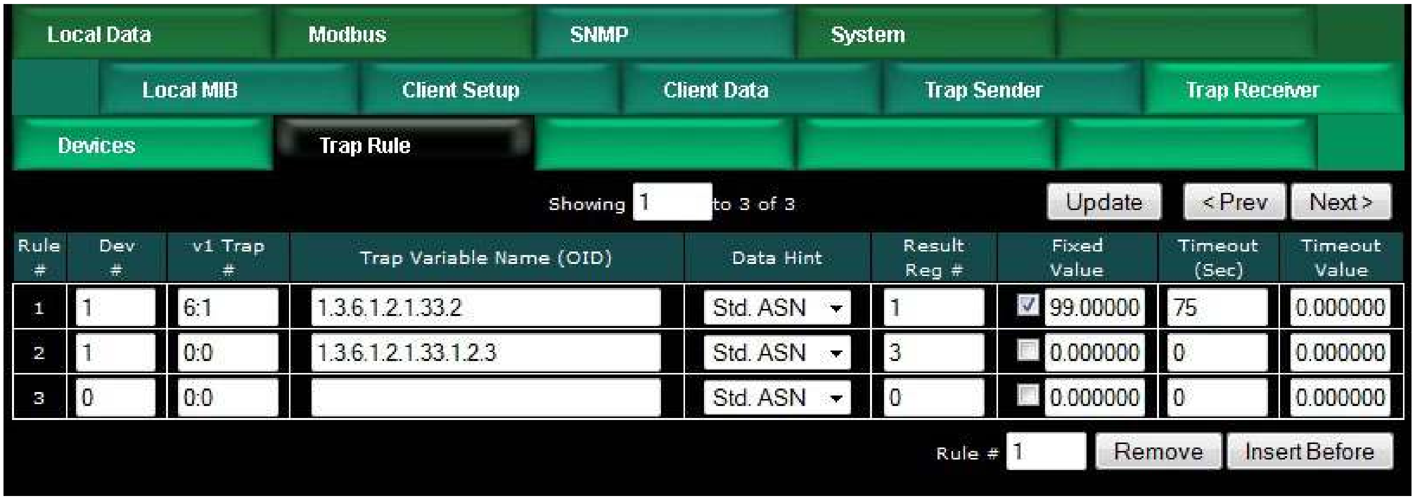
Use ○ IPv4 ● IPv6

IP Address 192.168.1.20

Traps Received: 13 Recognized: 9

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.

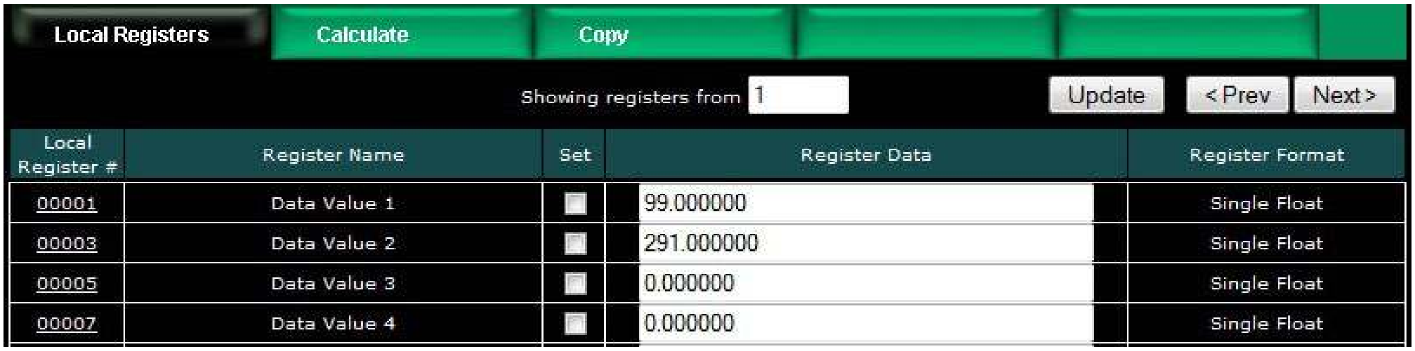


Showing 1 to 3 of 3

Rule #	Dev #	v1 Trap #	Trap Variable Name (OID)	Data Hint	Result Reg #	Fixed Value	Timeout (Sec)	Timeout Value
1	1	6:1	1.3.6.1.2.1.33.2	Std. ASN	1	☑ 99.00000	75	0.000000
2	1	0:0	1.3.6.1.2.1.33.1.2.3	Std. ASN	3	☐ 0.000000	0	0.000000
3	0	0:0		Std. ASN	0	☐ 0.000000	0	0.000000

Rule # 1

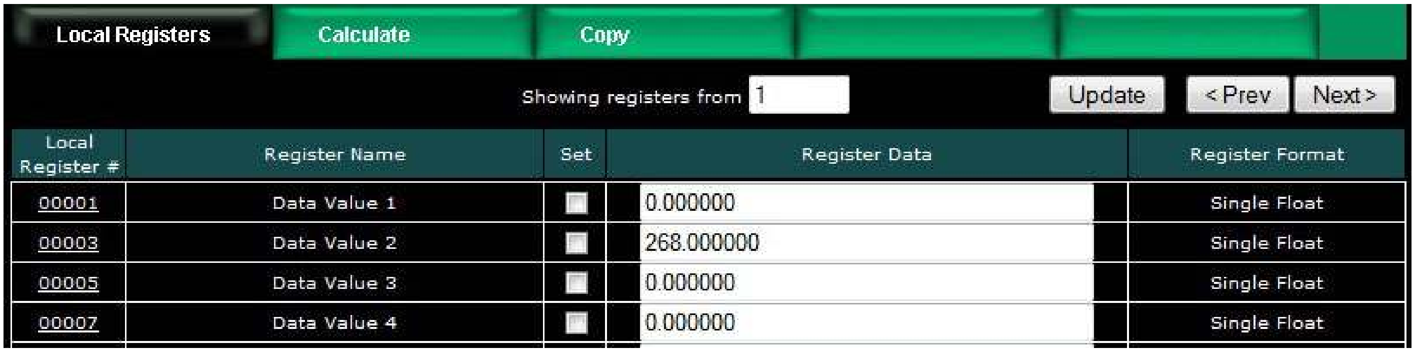
Immediately upon switching to battery power (utility power cut off), registers 1 and 2 are populated as illustrated below by the traps received from our test UPS. Data Value 1 is simply the fixed value flagging the fact that we are on battery power, and Data Value 2 is the number of minutes of remaining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



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	☐	99.000000	Single Float
00003	Data Value 2	☐	291.000000	Single Float
00005	Data Value 3	☐	0.000000	Single Float
00007	Data Value 4	☐	0.000000	Single Float

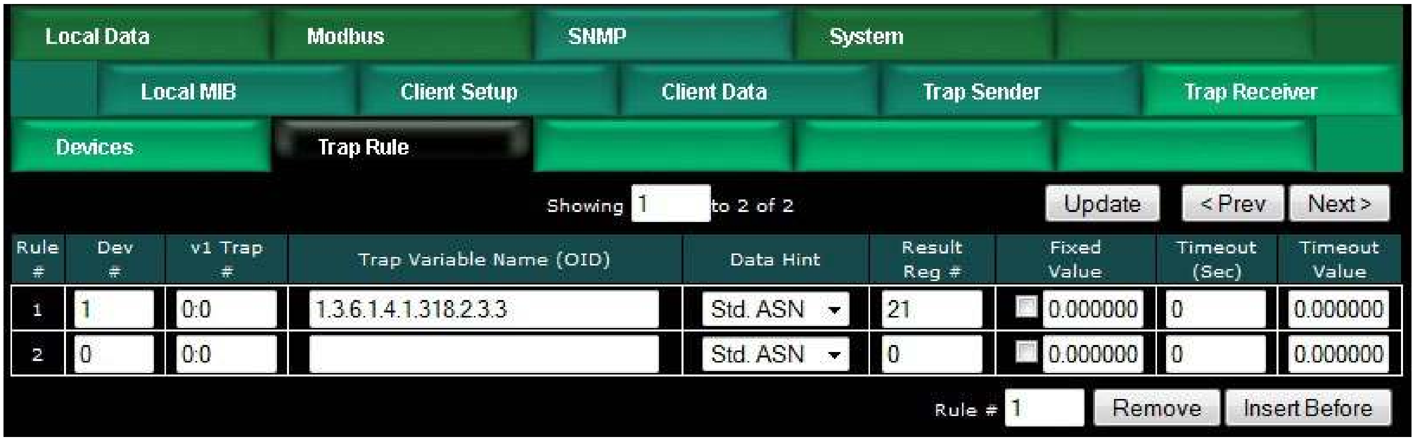
Upon restoring utility power, Data Value 1 returns to zero indicating we are no longer on battery power. Data Value 2 will continue to contain the last received "minutes remaining" value.



Showing registers from 1

Local Register #	Register Name	Set	Register Data	Register Format
00001	Data Value 1	☐	0.000000	Single Float
00003	Data Value 2	☐	268.000000	Single Float
00005	Data Value 3	☐	0.000000	Single Float
00007	Data Value 4	☐	0.000000	Single Float

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.

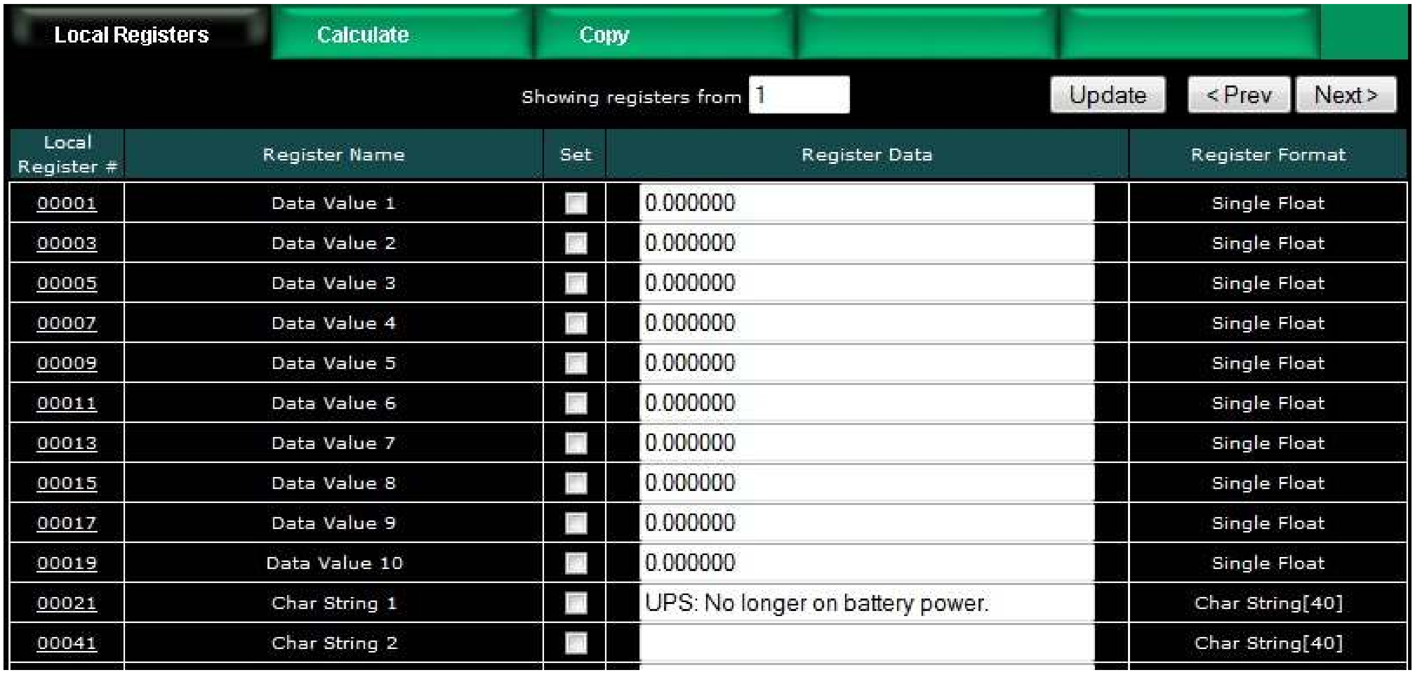


Showing 1 to 2 of 2

Rule #	Dev #	v1 Trap #	Trap Variable Name (OID)	Data Hint	Result Reg #	Fixed Value	Timeout (Sec)	Timeout Value
1	1	0:0	1.3.6.1.4.1.318.2.3.3	Std. ASN	21	□ 0.000000	0	0.000000
2	0	0:0		Std. ASN	0	□ 0.000000	0	0.000000

Rule # 1

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

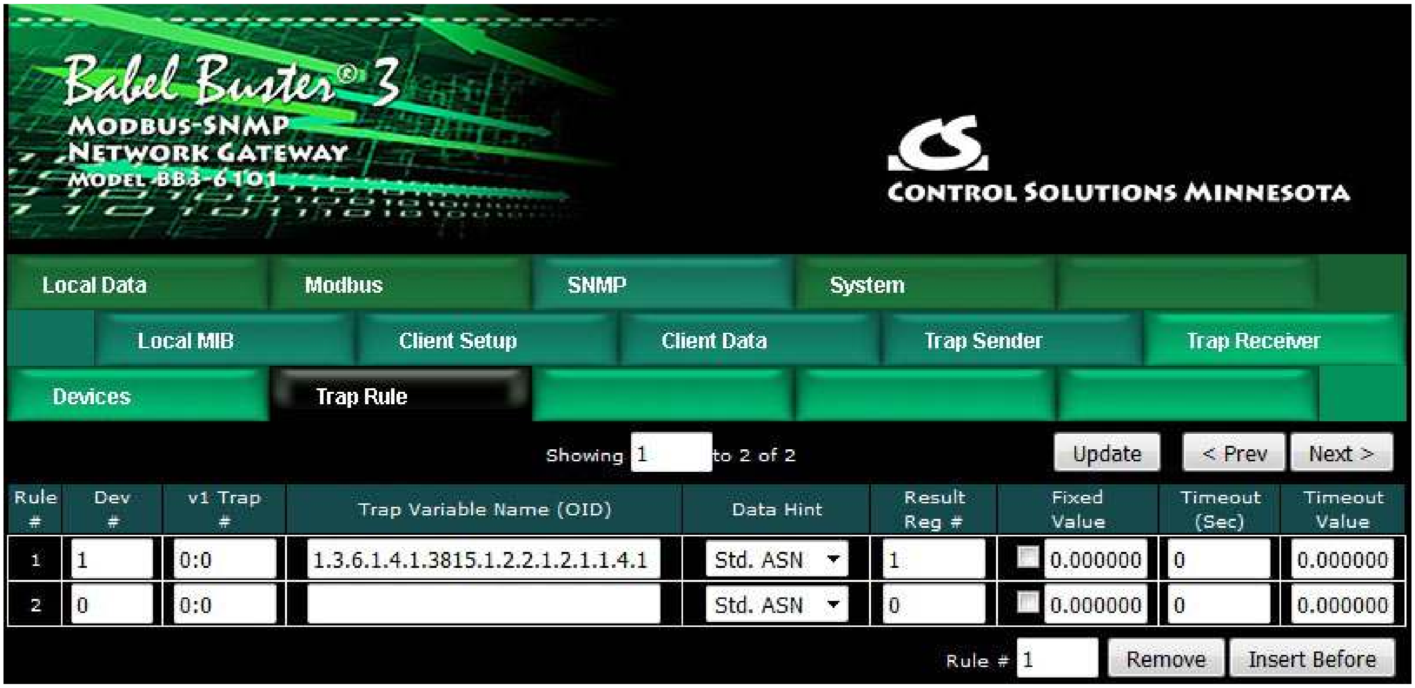


Showing registers from 1

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]

13.4 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.

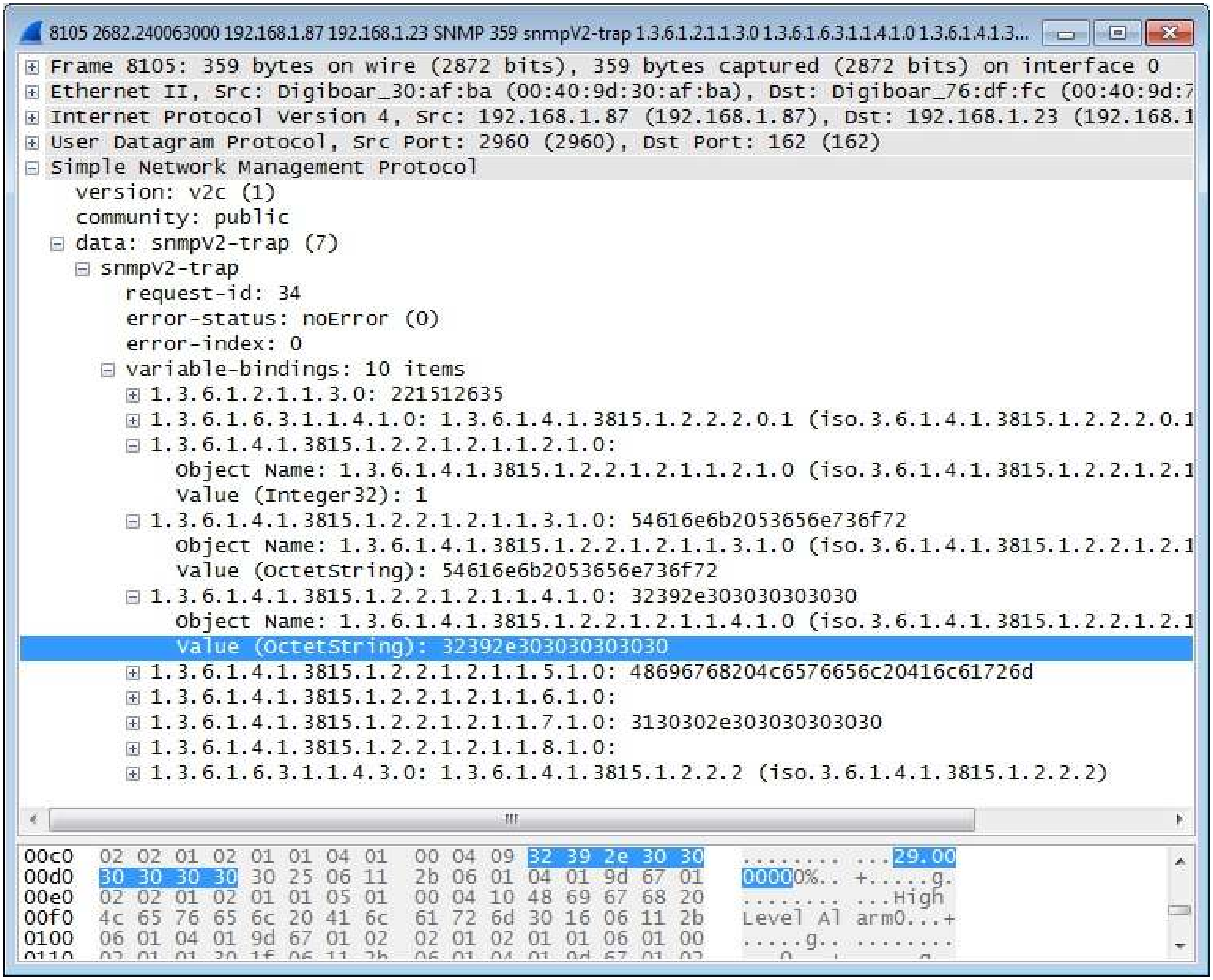


Showing 1 to 2 of 2

Rule #	Dev #	v1 Trap #	Trap Variable Name (OID)	Data Hint	Result Reg #	Fixed Value	Timeout (Sec)	Timeout Value
1	1	0:0	1.3.6.1.4.1.3815.1.2.2.1.2.1.1.4.1	Std. ASN	1	□ 0.000000	0	0.000000
2	0	0:0		Std. ASN	0	□ 0.000000	0	0.000000

Rule # 1

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.

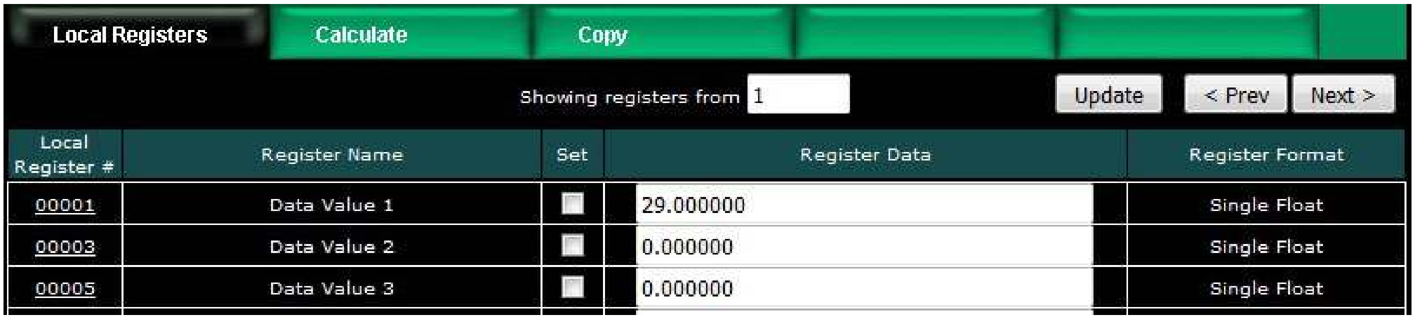


8105 2682.240063000 192.168.1.87 192.168.1.23 SNMP 359 snmpV2-trap 1.3.6.1.2.1.1.3.0 1.3.6.1.6.3.1.1.4.1.0 1.3.6.1.4.1.3...

```
⊞ Frame 8105: 359 bytes on wire (2872 bits), 359 bytes captured (2872 bits) on interface 0
⊞ Ethernet II, Src: Digiboar_30:af:ba (00:40:9d:30:af:ba), Dst: Digiboar_76:df:fc (00:40:9d:7
⊞ Internet Protocol Version 4, Src: 192.168.1.87 (192.168.1.87), Dst: 192.168.1.23 (192.168.1
⊞ User Datagram Protocol, Src Port: 2960 (2960), Dst Port: 162 (162)
⊟ Simple Network Management Protocol
    version: v2c (1)
    community: public
  ⊟ data: snmpV2-trap (7)
    ⊟ snmpV2-trap
        request-id: 34
        error-status: noError (0)
        error-index: 0
      ⊟ variable-bindings: 10 items
        ⊞ 1.3.6.1.2.1.1.3.0: 221512635
        ⊞ 1.3.6.1.6.3.1.1.4.1.0: 1.3.6.1.4.1.3815.1.2.2.2.0.1 (iso.3.6.1.4.1.3815.1.2.2.2.0.1
        ⊟ 1.3.6.1.4.1.3815.1.2.2.1.2.1.1.2.1.0:
            Object Name: 1.3.6.1.4.1.3815.1.2.2.1.2.1.1.2.1.0 (iso.3.6.1.4.1.3815.1.2.2.1.2.1
            Value (Integer32): 1
        ⊟ 1.3.6.1.4.1.3815.1.2.2.1.2.1.1.3.1.0: 54616e6b2053656e736f72
            Object Name: 1.3.6.1.4.1.3815.1.2.2.1.2.1.1.3.1.0 (iso.3.6.1.4.1.3815.1.2.2.1.2.1
            Value (OctetString): 54616e6b2053656e736f72
        ⊟ 1.3.6.1.4.1.3815.1.2.2.1.2.1.1.4.1.0: 32392e303030303030
            Object Name: 1.3.6.1.4.1.3815.1.2.2.1.2.1.1.4.1.0 (iso.3.6.1.4.1.3815.1.2.2.1.2.1
            Value (OctetString): 32392e303030303030
        ⊞ 1.3.6.1.4.1.3815.1.2.2.1.2.1.1.5.1.0: 48696768204c6576656c20416c61726d
        ⊞ 1.3.6.1.4.1.3815.1.2.2.1.2.1.1.6.1.0:
        ⊞ 1.3.6.1.4.1.3815.1.2.2.1.2.1.1.7.1.0: 3130302e303030303030
        ⊞ 1.3.6.1.4.1.3815.1.2.2.1.2.1.1.8.1.0:
        ⊞ 1.3.6.1.6.3.1.1.4.3.0: 1.3.6.1.4.1.3815.1.2.2.2 (iso.3.6.1.4.1.3815.1.2.2.2)
```

```
00c0  02 02 01 02 01 01 04 01  00 04 09 32 39 2e 30 30   ........ ...29.00
00d0  30 30 30 30 30 25 06 11  2b 06 01 04 01 9d 67 01   00000%.. +.....g.
00e0  02 02 01 02 01 01 05 01  00 04 10 48 69 67 68 20   ........ ...High
00f0  4c 65 76 65 6c 20 41 6c  61 72 6d 30 16 06 11 2b   Level Al arm0...+
0100  06 01 04 01 9d 67 01 02  02 01 02 01 01 06 01 00   .....g.. ........
0110  02 01 01 30 1f 06 11 2b  06 01 04 01 9d 67 01 02
```

Numeric converstion 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.



Showing registers from 1 Update < Prev Next >

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

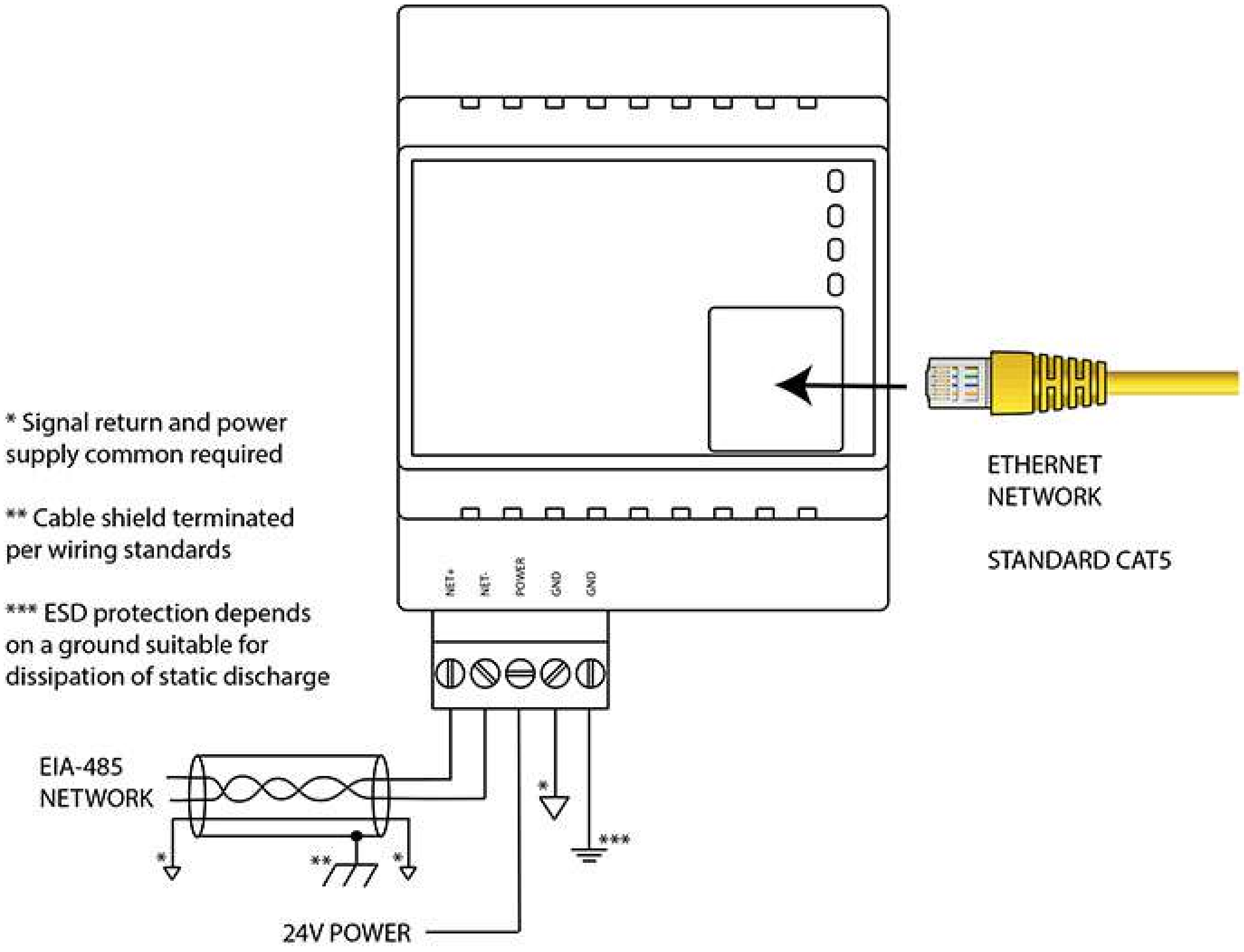
Babel Buster® 3
MODEL BB3-6101
MODEL MX-61
NETWORK GATEWAY

CONTROL SOLUTIONS MINNESOTA

Appendix A Hardware Details

A.1 Wiring

Wiring for the Babel Buster BB3-6101 is illustrated below.

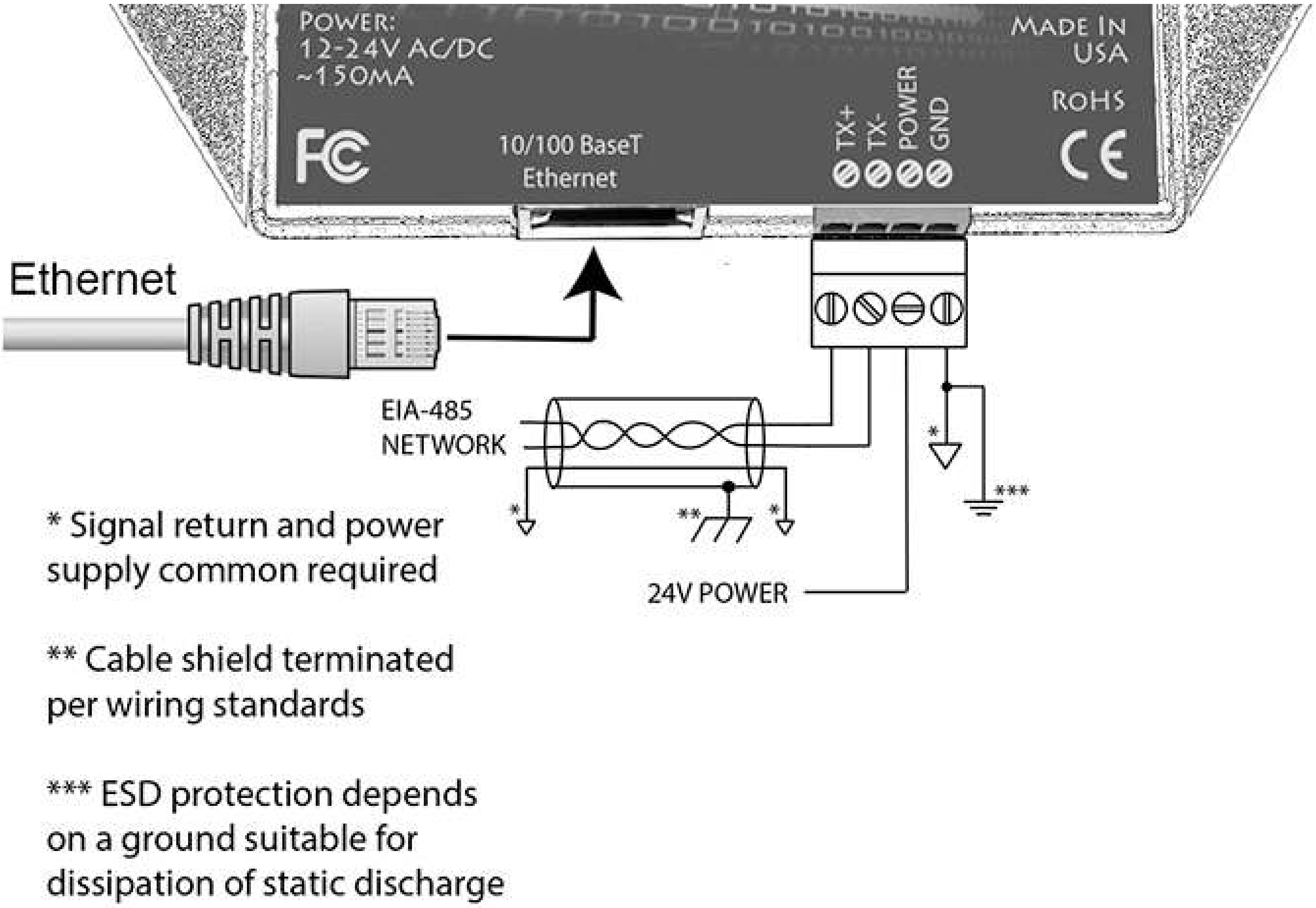


* Signal return and power supply common required

** Cable shield terminated per wiring standards

*** ESD protection depends on a ground suitable for dissipation of static discharge

Wiring for the Babel Buster MX-61 is illustrated below.



* Signal return and power supply common required

** Cable shield terminated per wiring standards

*** ESD protection depends on a ground suitable for dissipation of static discharge

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.

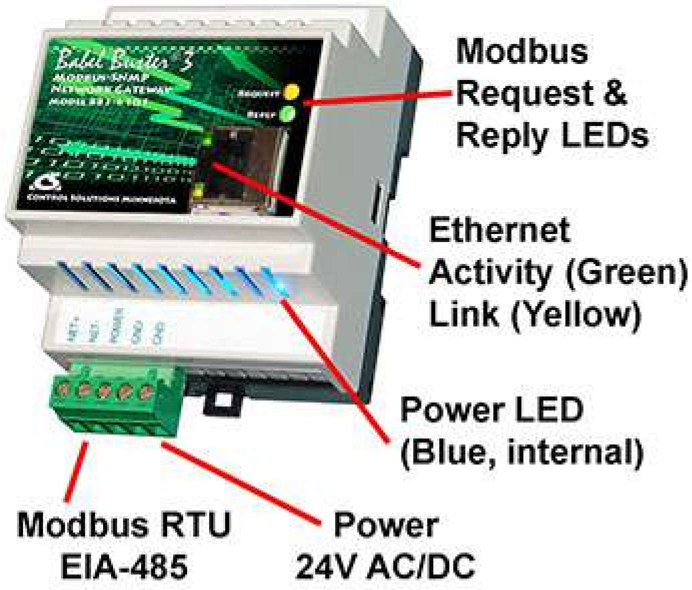
Note that in addition to connecting power supply common to a GND terminal, you must also connect a GND terminal to earth ground in order to ensure proper ESD protection.

BB3-6101-232: The standard BB3-6101 Modbus RTU port uses RS-485. The RS-232 version replaces the RS-485 transceiver with an RS-232 transceiver. The NET+/NET- terminals are replaced by TXD and RXD on the -232 version. TXD is data out from the BB3-6101-232, and RXD is data in to the gateway. Hardware handshake is not supported.

A.2 Front Panel LED Indicators

A.2.1 BB3-6101 LED Indicators

Power-up LED behavior: On power up, the Reply LED will remain on solid red for about 20 seconds, then the Request and Reply LEDs will do a "lamp test" where Request is yellow and Reply is Red simultaneously for about 1 second, and then both Request and Reply turn green simultaneously for about 1 second. The LEDs will then begin to operate according to their normal functionality.


Babel Buster 3
Modbus-SNMP Network Gateway
Model BB3-6101
Request
Reply
Control Solutions Minnesota
Modbus Request & Reply LEDs
Ethernet Activity (Green) Link (Yellow)
Power LED (Blue, internal)
Modbus RTU EIA-485
Power 24V AC/DC

Babel Buster BB3-6101 Request and Reply LEDs reflect Modbus RTU traffic, and the Ethernet activity LED will indicate network traffic in general. If Modbus RTU is not being used at all, then the Request and Reply LEDs will indicate TCP traffic. If Modbus RTU is in use, then the Request and Reply LEDs will indicate Modbus RTU traffic while the Ethernet LEDs will be the only indication of TCP traffic.

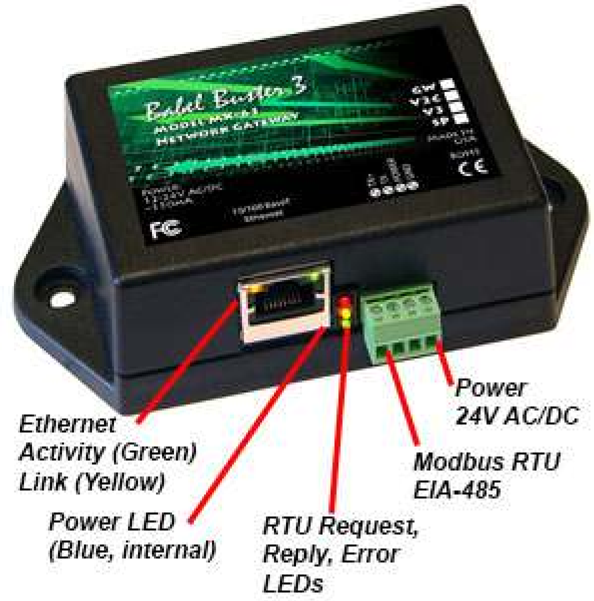
Babel Buster BB3-6101 LEDs indicate as follows (LEDs are bi-color):

REQUEST	Flashes yellow each time a request is sent when operating as Modbus Master, or each time a request is received when operating as Modbus Slave.
REPLY	Operating as Modbus Master, flashes green each time a good response is received, or red 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 green each time a good response is sent, or red if an exception code is sent (meaning the received request resulted in an error).
Ethernet Activity	Green LED is on solid during portions of the boot-up process, and then flashes briefly when Ethernet network traffic is

	detected.
Ethernet Link	Yellow LED indicates an Ethernet link is present. This indicator will light if a link is present regardless of processor or network activity. If not lit, check network wiring.
Status	Blue LED (internal) on any time power is present and internal power supply is functioning.

A.2.2 MX-61 LED Indicators

Power-up LED behavior: On power up, the Request, Reply and Error LEDs will remain off for about 20 seconds, then all three LEDs will do a "lamp test" where they all turn on simultaneously for about 1 second. The LEDs will then begin to operate according to their normal functionality.



Babel Buster MX-61 Request, Reply and Error LEDs reflect Modbus RTU traffic, and the Ethernet activity LED will indicate network traffic in general.

Babel Buster MX-61 LEDs indicate as follows (LEDs are each a single color):

Error (red)	Operating as Modbus Master, flashes red 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 red if an exception code is sent (meaning the received request resulted in an error).
Request (yellow)	Flashes yellow each time a request is sent when operating as Modbus Master, or each time a request is received when operating as Modbus Slave.

Reply (green)	Operating as Modbus Master, flashes green each time a good response is received. Operating as Modbus Slave, flashes green each time a good response is sent.
Ethernet Activity	Green LED is on solid during portions of the boot-up process, and then flashes briefly when Ethernet network traffic is detected.
Ethernet Link	Yellow LED indicates an Ethernet link is present. This indicator will light if a link is present regardless of processor or network activity. If not lit, check network wiring.
Status	Blue LED (internal) on any time power is present and internal power supply is functioning.

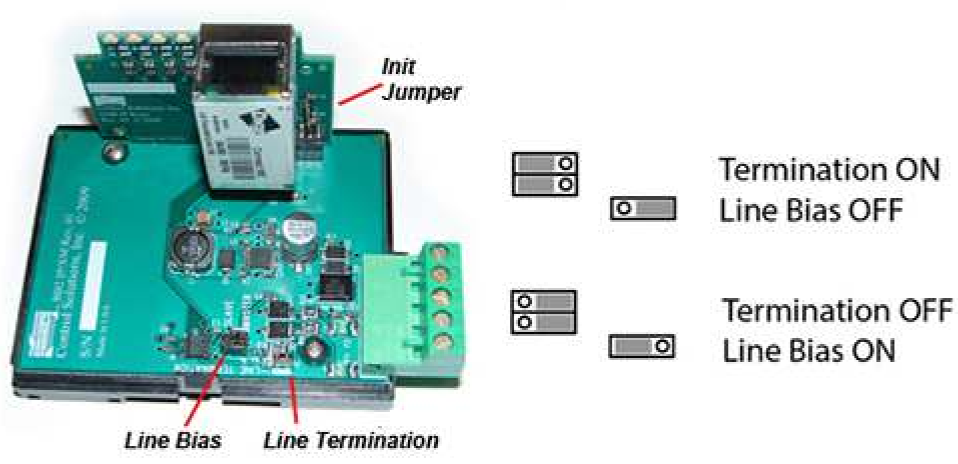
A.3 RS-485 Line Termination & Bias

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.

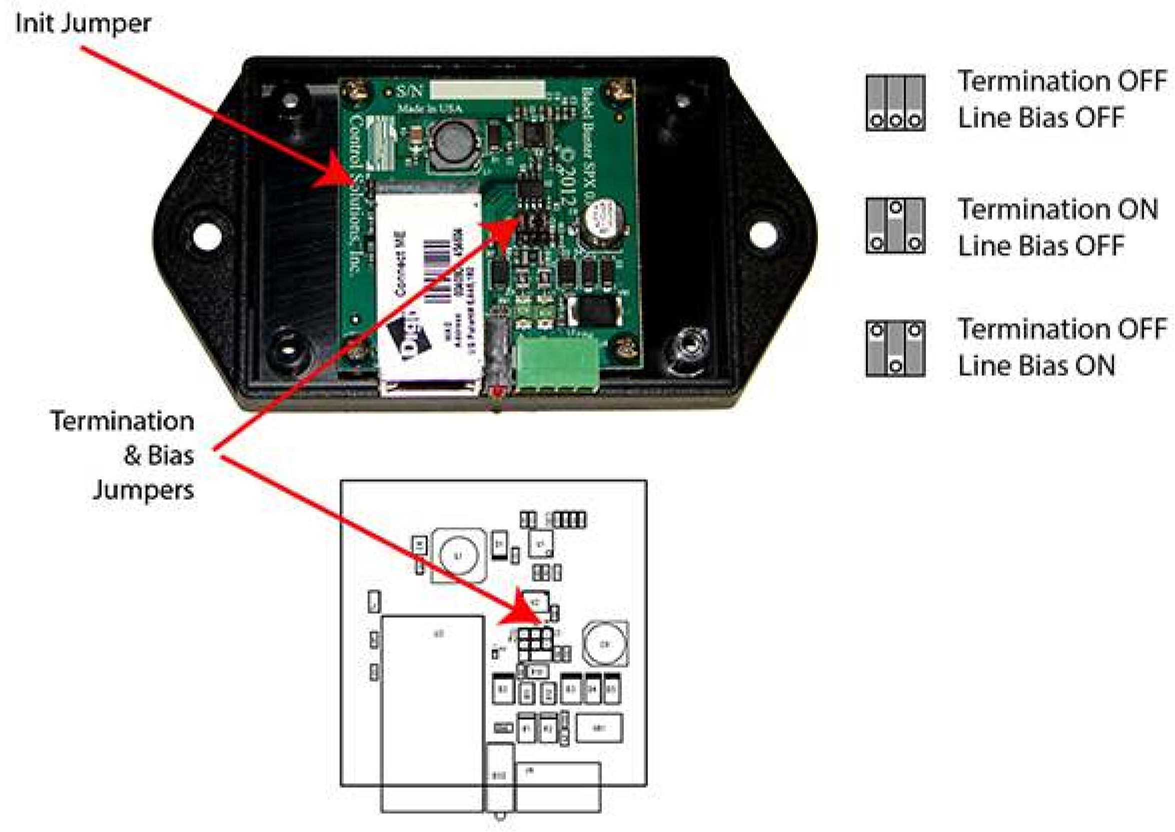
Enable line bias when needed. Line bias should only be enabled at one point on the network, and does not have to be the end point. Line bias holds the line in a known neutral state when no devices are transmitting. Without bias, the transition from offline to online by a transmitter can look like a false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

Jumper locations for Babel Buster BB3-6101:



Jumper locations for Babel Buster MX-61:


Init Jumper
Termination & Bias Jumpers
Termination OFF Line Bias OFF
Termination ON Line Bias OFF
Termination OFF Line Bias ON

A.4 Soft Configuration Reset

Soft reset should be used to remove all configuration information any time you do have the ability to connect to the gateway's web user interface. The "Clear Configuration" action is described in Section 3.1.5. Using the forced hard reset should only be used as a last resort if you are unable to connect to the gateway because the SSL certificates are invalid for a secure connection or you are unable to recover the lost IP address.

A.5 Discovering Lost IP Address

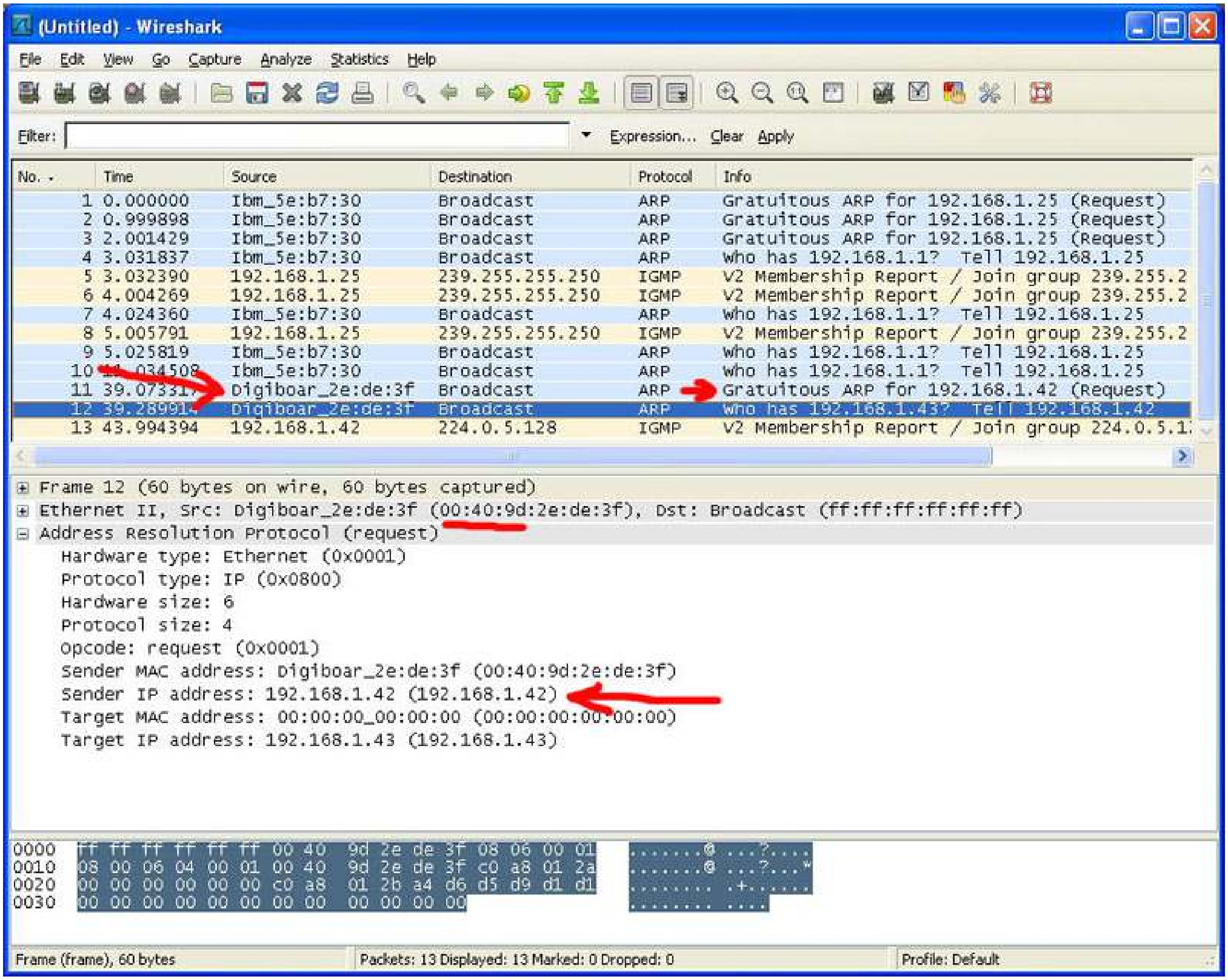
You can use Wireshark to discover a lost IP address if the gateway is still functional. Connect the gateway directly to your PC running Wireshark using a cross-over cable (or standard CAT5 cable if your PC supports auto-MDX). With Wireshark running, power up the gateway.

Upon power up, BB3-6101/MX-61 will ping its own IP address one or more times. This is part of its duplicate address resolution mechanism. If it finds another device with its own IP address, it will set its own IP address to a default pseudo-random address generally starting with 192.

Wait until you are certain BB3-6101/MX-61 has booted up, or wait 2-3 minutes to be sure if you don't recognize the bootup LED sequence. Now look for the ARP packets and note what IP address they came from. This is your device. (To make sure it is your device, connect only the BB3-6101/MX-61 to your PC while doing this exercise.)

Your device will have a MAC address that starts with 00:40:9D, also labeled with a source that starts with “Digiboar_”. This label comes from the fact that the server modules used on Control Solutions IP products are made by Digi International, previously known as “Digiboard”.

There will usually be one or more "pings" or ARP packets to the device's own IP address, and one last ping to its own address plus one. In the illustration here, the BB3-6101/MX-61 is located at 192.168.1.42.


(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. -   Time        Source              Destination        Protocol  Info
      1 0.000000    Ibm_5e:b7:30        Broadcast          ARP       Gratuitous ARP for 192.168.1.25 (Request)
      2 0.999898    Ibm_5e:b7:30        Broadcast          ARP       Gratuitous ARP for 192.168.1.25 (Request)
      3 2.001429    Ibm_5e:b7:30        Broadcast          ARP       Gratuitous ARP for 192.168.1.25 (Request)
      4 3.031837    Ibm_5e:b7:30        Broadcast          ARP       Who has 192.168.1.1?  Tell 192.168.1.25
      5 3.032390    192.168.1.25        239.255.255.250    IGMP      V2 Membership Report / Join group 239.255.2
      6 4.004269    192.168.1.25        239.255.255.250    IGMP      V2 Membership Report / Join group 239.255.2
      7 4.024360    Ibm_5e:b7:30        Broadcast          ARP       Who has 192.168.1.1?  Tell 192.168.1.25
      8 5.005791    192.168.1.25        239.255.255.250    IGMP      V2 Membership Report / Join group 239.255.2
      9 5.025819    Ibm_5e:b7:30        Broadcast          ARP       Who has 192.168.1.1?  Tell 192.168.1.25
     10 [illegible].034508   Ibm_5e:b7:30        Broadcast          ARP       Who has 192.168.1.1?  Tell 192.168.1.25
     11 39.073317   Digiboar_2e:de:3f   Broadcast          ARP       Gratuitous ARP for 192.168.1.42 (Request)
     12 39.289914   Digiboar_2e:de:3f   Broadcast          ARP       Who has 192.168.1.43?  Tell 192.168.1.42
     13 43.994394   192.168.1.42        224.0.5.128        IGMP      V2 Membership Report / Join group 224.0.5.1:
```

```
⊞ Frame 12 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: Digiboar_2e:de:3f (00:40:9d:2e:de:3f), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊟ Address Resolution Protocol (request)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    Sender MAC address: Digiboar_2e:de:3f (00:40:9d:2e:de:3f)
    Sender IP address: 192.168.1.42 (192.168.1.42)
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.1.43 (192.168.1.43)
```

```
0000  ff ff ff ff ff ff 00 40  9d 2e de 3f 08 06 00 01   .......@ ...?....
0010  08 00 06 04 00 01 00 40  9d 2e de 3f c0 a8 01 2a   .......@ ...?...*
0020  00 00 00 00 00 00 c0 a8  01 2b a4 d6 d5 d9 d1 d1   ........ .+......
0030  00 00 00 00 00 00 00 00  00 00 00 00              ........ ....
```

Frame (frame), 60 bytes | Packets: 13 Displayed: 13 Marked: 0 Dropped: 0 | Profile: Default

A.6 Forced Hard Configuration Reset

IMPORTANT: Before considering the forced hard reset, be sure you have considered soft configuration reset, or discovering lost IP address if applicable.

The "Init" jumper inside the BB3-6101/MX-61 serves two purposes, and what it does depends on whether you apply the jumper before or after the BB3-6101/MX-61 boots up.

Hard Configuration Reset:

Installing the jumper after bootup causes the BB3-6101/MX-61 to do a hard reset on its configuration memory. The IPv4 address will be reset to 10.0.0.101. The root password will be reset to the original default password. After clearing all configuration, the BB3-6101/MX-61 will automatically restart. Remove the jumper when you see the indication of restart after about 30 seconds, which is both LEDs coming on solid on the RJ45 Ethernet connector and remaining on for a couple of seconds. If you miss the start of reboot, both LEDs on the RJ45 will come on and stay on. It will now be attempting the firmware update, but you can abort that by simply powering down the BB3-6101/MX-61. If both LEDs on the RJ45 jack come on and remain on, remove the jumper and then power cycle the BB3-6101/MX-61.

Once you have regained access to the device, go to the File Manager page, execute the Clear All configuration action, then select the file named as "Boot configuration" and execute the Save XML Config File action to wipe out any configuration normally saved in the XML configuration file.

Note: The forced hard reset will restore HTTP web access and disable HTTPS web access. The forced hard reset will also restore FTP access to allow FTP firmware uploads if needed.

Note: The hard reset of configuration also means all of your resource allocations are reset to original factory defaults. If you want resource allocations that are different, you will need to repeate the allocation setup as described in Section 3.4.

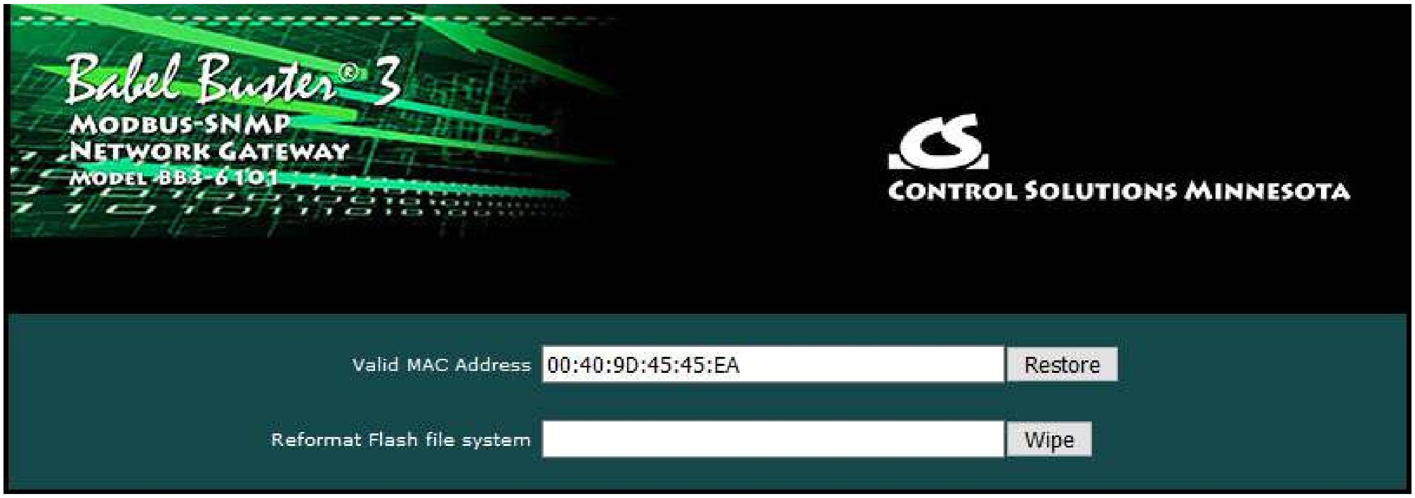
Firmware Update Recovery:

Installing this jumper prior to power-up causes the server to go into TFTP firmware update mode. Normally you would perform a firmware update by simply uploading a new image.bin file (provided by Control Solutions tech support) using the BB3-6101/MX-61's internal FTP server and a command line FTP session on your PC (Linux or Windows command line). Detailed instructions are included in the zip file that also contains the applicable image.bin file.

Should the FTP upload fail for some reason, then you need to resort to the TFTP upload method as the fallback method. Full details on how to go about this can be found under the topic "Restoring a corrupt application image" at https://info.csimn.com.

Additional maintenance page:

Go to http(s)://10.0.0.101/html/pgRestoreAddr.html to find the following page (substituting your IP address). It serves two purposes as noted below, which ideally you will never have a use for.



Valid MAC Address 00:40:9D:45:45:EA

Reformat Flash file system

File System Wipe:

On rare occasion, the Flash file system has been observed to get corrupted as a result of losing power while a write operation was in progress. This is most effectively confirmed by opening a command prompt FTP session (Windows 10 PowerShell) to try to view the files in the Flash file system. If FTP fails to show any files, in addition to other problems saving or loading files, it may be that the file system has gotton corrupted. If this happens, go to the page pictured above, and enter the Reformat key, then click Wipe, and then power cycle the device (or restart from the File Manager page). The reformat key is 55AAAA55. Simply type that into the window next to the Wipe button.

MAC Address Restore:

In the event the MAC address has been reset due to NVRAM checksum failure, this page will permit restoring the MAC address to its original address as printed on the component label internal to this device, or on the default password label found on the outside or on external documentation included with the device.

If the MAC address is deemed to be valid, the window will be labeled "Valid MAC Address" and you will not be allowed to change it. If the MAC address is deemed to be invalid, the window will be labeled "Restore MAC Address" and you should then enter the correct MAC address and click Restore. A restart is then needed.

A.7 Firmware Update Notes

The most up to date firmware is shipped with all new devices. This isn't like a new laptop where you spent the first half a day updating software on a computer you thought was brand new. If you believe you have discovered an issue that you believe a firmware update might fix, contact technical support first to confirm whether that is the case, and then to get a login to the firmware update support site.

The brute force approach to updating firmware using TFTP as noted in the section above is always available, but the more graceful approach is to use FTP to upload the new image.bin file. There is one minor problem: The upload wants to buffer the entire file in RAM while it procedes to reprogram the Flash memory. **If the memory**

utilization indicated on the Resources page in your device is above about 30%, the FTP upload will fail, and thus the firmware update will not take place.

You have two choices: (1) Use the TFTP approach, or (2) Temporarily reconfigure your gateway to use a minimum of resources to free up space to temporarily buffer the image.bin file upload.

More detailed instructions for the FTP upload are included in the zip file you will download to obtain the firmware update. Instructions for the TFTP upload are available in our knowledgebase at https://info.csimn.com.

Appendix B Modbus CSV Import Files

HINT: If you get "table full" errors while importing CSV files, you might not have sufficient resources allocated. You may need to increase some counts on the Resources page.

B.1 Modbus RTU Master Read/Write Maps

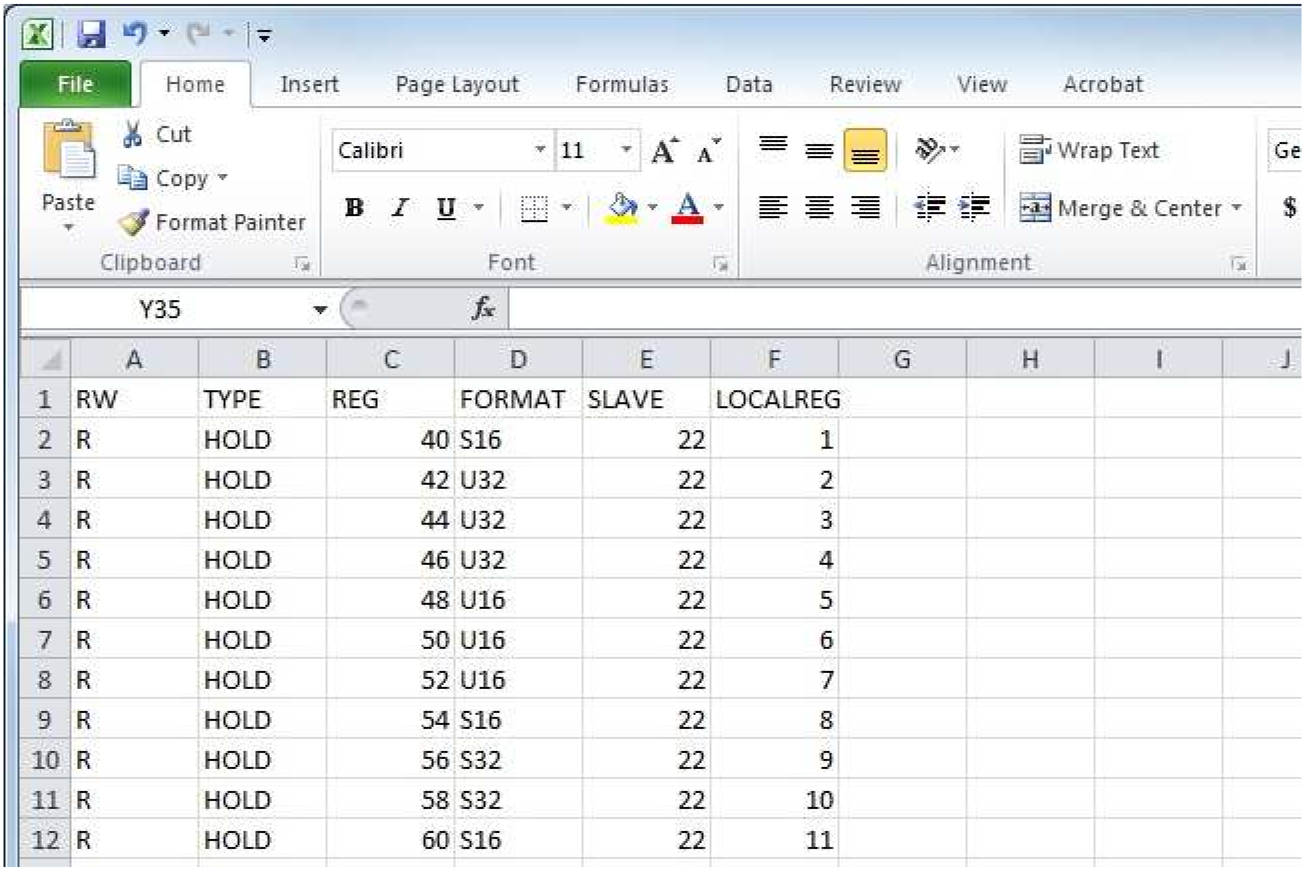
The CSV file for configuring Modbus TCP client read and write maps should contain a single header line with the labels indicated below, and content as applicable.

Header Line Label	Notes	Description of Use
RW	-	Enter 'R' to Read from a remote device, or 'W' to Write to a remote device.
TYPE	-	Use this column to specify remote registers by type (see Reference B)
REG	-	Use this column in conjunction with Type to specify remote register numbers of the selected type. Note that register numbers are 1-indexed, meaning raw address 0 should be entered as register #1.
FORMAT	-	Specify the format of the remote register to be read or written (see Reference C)
SLAVE	-	Provide the slave address, ID, or unit number, of the Modbus RTU slave to be polled.
SWAP	-	Any data item that occupies more than one Modbus register, e.g. 32-bit or Float, needs to have the register order defined since this is not standardized by Modbus protocol. The Babel Buster gateways default to the high order register first. If the remote Modbus slave has its registers ordered with low order first, then select 'T' (True) to "swap" the register order (or 'F' to keep the default order).
SCALE	-	Data is multiplied by this scale factor after read from a remote device or before being written to a remote device.
OFFSET	-	This offset is added to the data value after read from a remote device or before being written to a remote device.
POLL	-	Specify a periodic poll time in seconds (fractions of seconds are recognized).

LOCALREG	-	Specify a local Modbus register number where data read from a remote device will be placed, or where data written to a remote device will be taken from.
MASK	-	When READING: If a bit mask is entered (in hexadecimal), and the remote register type is signed or unsigned integer, the mask will be bit-wise logical AND-ed with the data, and the retained bits will be right justified in the result. When WRITING: If a bit mask is entered, and the remote register type is signed or unsigned, 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.
DEFAULT	-	When READING: The default value will be stored into the local object/register after the given number of read failures if the fail count (MAXFAIL) is non-zero. When WRITING: The default value will be stored into the local object/register if POR is set to True, or if the amount of time specified by TIMEOUT is exceeded without an update to the local object by a remote client.
MAXFAIL	1	If non-zero, sets the maximum number of times that a read attempt may fail before the default value will be placed in the local object/register. Setting the count to zero will disable the default, and the object/register will retain the most recent value obtained.
FILL	2	When WRITING: 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.
MAXQUIET	2	If using 'send on delta', to guarantee that the remote device will be written at least occasionally even if the data does not change, enter a maximum quiet time (in seconds).
MINQUIET	2	If using 'send on delta', and the delta increment is small, the result can be a large amount of network traffic. To limit network traffic, provide a MINQUIET time (in seconds) that must elapse between transmission of changed values.
DELTA	2	The local object/register data may be written to the remote device periodically, or when the local value changes, or both. To send upon change (send on delta), provide a DELTA value as the amount by which the local object/register must change before being written to the remote device. Leave blank if send on delta should not be used.

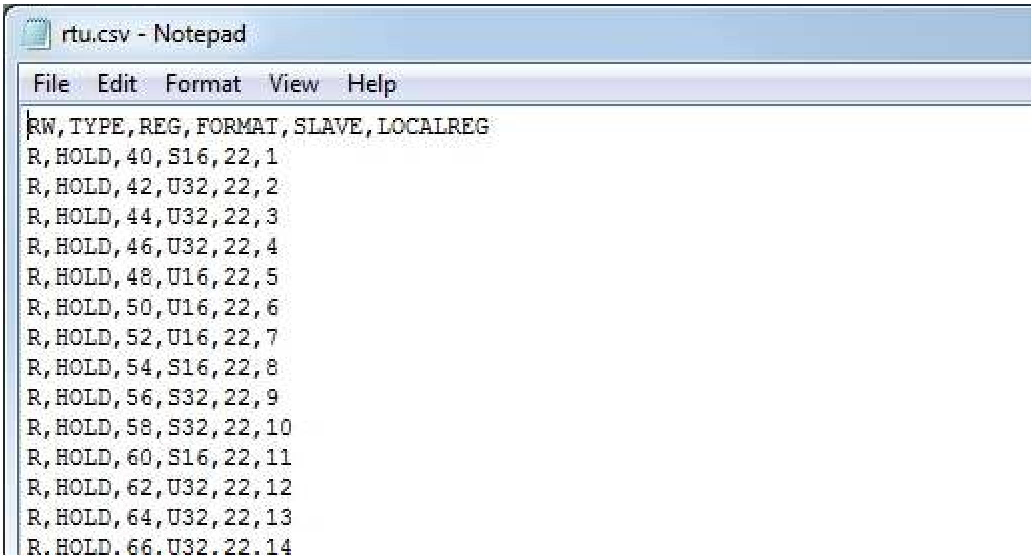
The minimum required header line for Modbus RTU must include RW, LOCALREG, TYPE, REG, FORMAT, and SLAVE. All other columns are optional.

This is an example of a minimum CSV file as it would appear in a spread sheet program:



	A	B	C	D	E	F
1	RW	TYPE	REG	FORMAT	SLAVE	LOCALREG
2	R	HOLD	40	S16	22	1
3	R	HOLD	42	U32	22	2
4	R	HOLD	44	U32	22	3
5	R	HOLD	46	U32	22	4
6	R	HOLD	48	U16	22	5
7	R	HOLD	50	U16	22	6
8	R	HOLD	52	U16	22	7
9	R	HOLD	54	S16	22	8
10	R	HOLD	56	S32	22	9
11	R	HOLD	58	S32	22	10
12	R	HOLD	60	S16	22	11

This is an example of how the minimum CSV file looks as just plain text:



```
RW,TYPE,REG,FORMAT,SLAVE,LOCALREG
R,HOLD,40,S16,22,1
R,HOLD,42,U32,22,2
R,HOLD,44,U32,22,3
R,HOLD,46,U32,22,4
R,HOLD,48,U16,22,5
R,HOLD,50,U16,22,6
R,HOLD,52,U16,22,7
R,HOLD,54,S16,22,8
R,HOLD,56,S32,22,9
R,HOLD,58,S32,22,10
R,HOLD,60,S16,22,11
R,HOLD,62,U32,22,12
R,HOLD,64,U32,22,13
R,HOLD,66,U32,22,14
```

B.2 Modbus TCP Client Read/Write Maps

The CSV file for configuring Modbus TCP client read and write maps should contain a single header line with the labels indicated below, and content as applicable.

The only difference between TCP and RTU formats is DEVNUM and UNIT in TCP, versus just SLAVE in RTU. Everything else is identical.

Header Line Label	Notes	Description of Use
RW	-	Enter 'R' to Read from a remote device, or 'W' to Write to a remote device.
TYPE	-	Use this column to specify remote registers by type (see Reference B)
REG	-	Use this column in conjunction with Type to specify remote register numbers of the selected type. Note that register numbers are 1-indexed, meaning raw address 0 should be entered as register #1.
FORMAT	-	Specify the format of the remote register to be read or written (see Reference C)
DEVNUM	-	Specify the device number where the remote register is to be found. This number is used to look up a device in the Modbus TCP Client Device table which contains the device's IP address, etc.
UNIT	-	Unit number is optional, and may be 1 to 247.
SCALE	-	Data is multiplied by this scale factor after read from a remote device or before being written to a remote device.
OFFSET	-	This offset is added to the data value after read from a remote device or before being written to a remote device.
POLL	-	Specify a periodic poll time in seconds (fractions of seconds are recognized).
LOCALREG	-	Specify a local Modbus register number where data read from a remote device will be placed, or where data written to a remote device will be taken from.
MASK	-	When READING: If a bit mask is entered (in hexadecimal), and the remote register type is signed or unsigned integer, the mask will be bit-wise logical AND-ed with the data, and the retained bits will be right justified in the result. When WRITING: If a bit mask is entered, and the remote register type is signed or unsigned, 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.
DEFAULT	-	When READING: The default value will be stored into the local object/register after the given number of read failures if the fail count (MAXFAIL) is non-zero. When WRITING: The default value will be stored into the local object/register if POR is set to True, or if the amount of time

		specified by TIMEOUT is exceeded without an update to the local object by a remote client.
MAXFAIL	1	If non-zero, sets the maximum number of times that a read attempt may fail before the default value will be placed in the local object/register. Setting the count to zero will disable the default, and the object/register will retain the most recent value obtained.
FILL	2	When WRITING: 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.
MAXQUIET	2	If using 'send on delta', to guarantee that the remote device will be written at least occasionally even if the data does not change, enter a maximum quiet time (in seconds).
MINQUIET	2	If using 'send on delta', and the delta increment is small, the result can be a large amount of network traffic. To limit network traffic, provide a MINQUIET time (in seconds) that must elapse between transmission of changed values.
DELTA	2	The local object/register data may be written to the remote device periodically, or when the local value changes, or both. To send upon change (send on delta), provide a DELTA value as the amount by which the local object/register must change before being written to the remote device. Leave blank if send on delta should not be used.

The minimum required header line for Modbus TCP must include RW, LOCALREG, TYPE, REG, FORMAT, and DEVNUM. All other columns are optional.

B.3 Register Types

The content of the TYPE column should contain one of the following CSV Labels:

CSV Label	Modbus Register Type	Function Code for Read	Function Code for Write
COIL	Coil (1 bit)	1	5 or 15
DISC	Discrete Input (1 bit)	2	n/a
INPUT	Input Register (16 bits)	4	n/a
HOLD	Holding Register (16 bits)	3	6 or 16

B.4 Register Formats

The content of the FORMAT column should contain one of the following CSV Labels:

CSV Label	Modbus Register Data Format	Occupies # Registers
S16	Signed 16-bit Integer	1
U16	Unsigned 16-bit Integer	1
S32	Signed 32-bit Integer	2

U32	Unsigned 32-bit Integer	2
FP	Floating Point, IEEE 754 32-bit	2
BIT	Bit (only used for Coil or Discrete Input registers)	-
MOD102	Mod-10, 2-register	2
MOD103	Mod-10, 3-register	3
MOD104	Mod-10, 4-register	4
S64	Signed 64-bit Integer	4

Appendix C Modbus Reference Information

C.1 Function Codes, Error Codes, and More

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.
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.
11 (hex 0B)	Gateway Target Device Failed to Respond	Specialized use in conjunction with gateways, indicates no response was received from the 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.

Babel Buster® 3
MODEL BB3-6101
MODEL MX-61
NETWORK GATEWAY

CONTROL SOLUTIONS MINNESOTA

Appendix D Trouble Shooting

D.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 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 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 is configured as Modbus master, this means the Read Map or Write Map is not configured correctly.

D.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 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 is configured as Modbus master, this means the Read Map or Write Map is not configured correctly.

D.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 for SNMPv2 or incorrect user credentials for SNMPv3. Without the correct credentials, 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.)

Two of the most useful tools in trouble shooting SNMP are Wireshark and a MIB browser. An example follows.

D.4 Wireshark 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 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 the MIB Browser, 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 that call themselves "hubs" but support 100BaseT connections. These are switches.

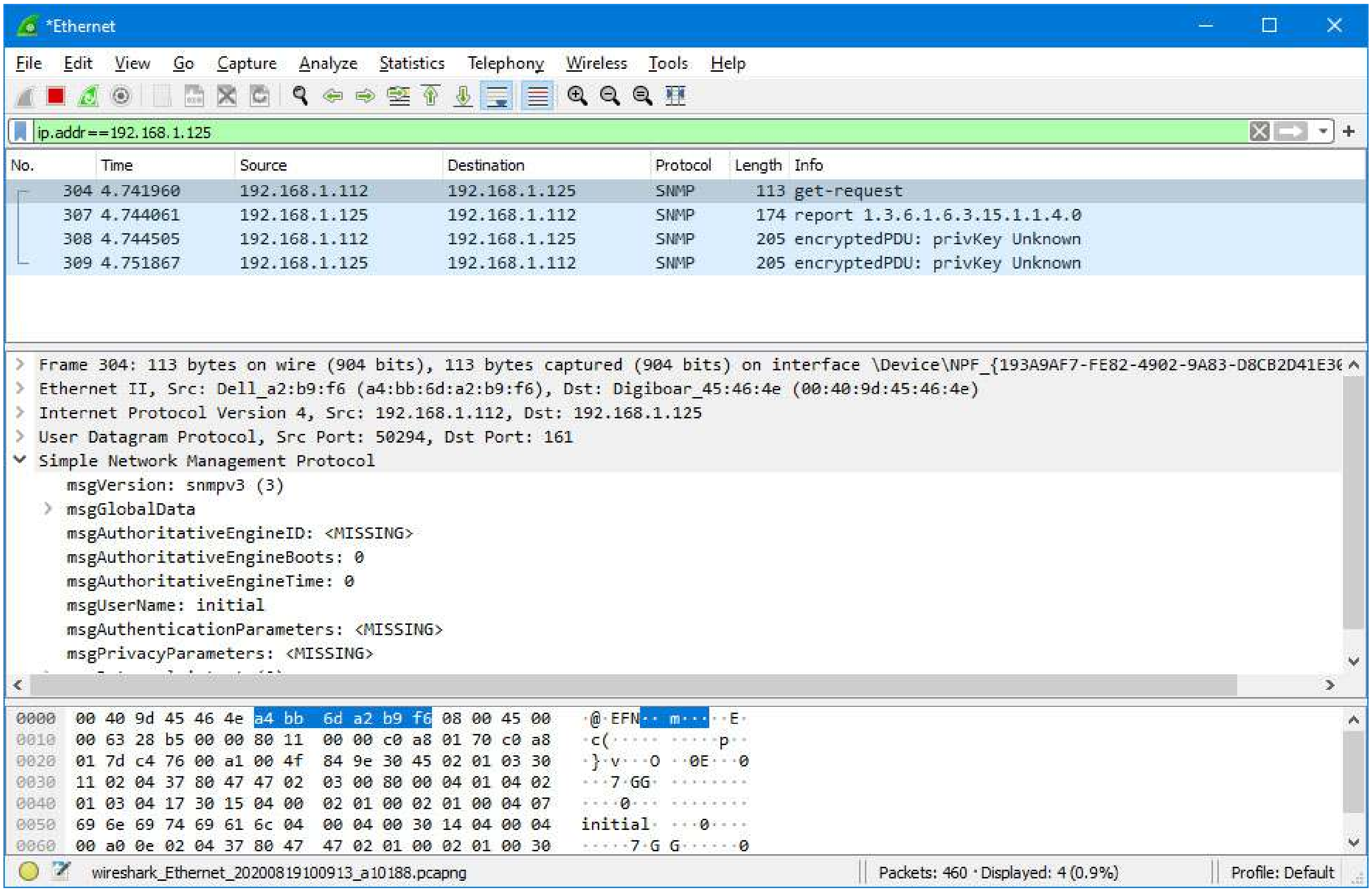
Since manufacturers of hubs decided nobody should have a use for them anymore, they are generally out of production. Finding a 10BaseT hub for sale is not easy (try eBay). But there are other alternatives.

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 2020).

D.5 Example of Using Wireshark

If you use Wireshark to capture a Get request in SNMPv3 with privacy configured, Wireshark won't be able to display anything meaningful until you provide Wireshark with some credentials.


*Ethernet
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr==192.168.1.125
```

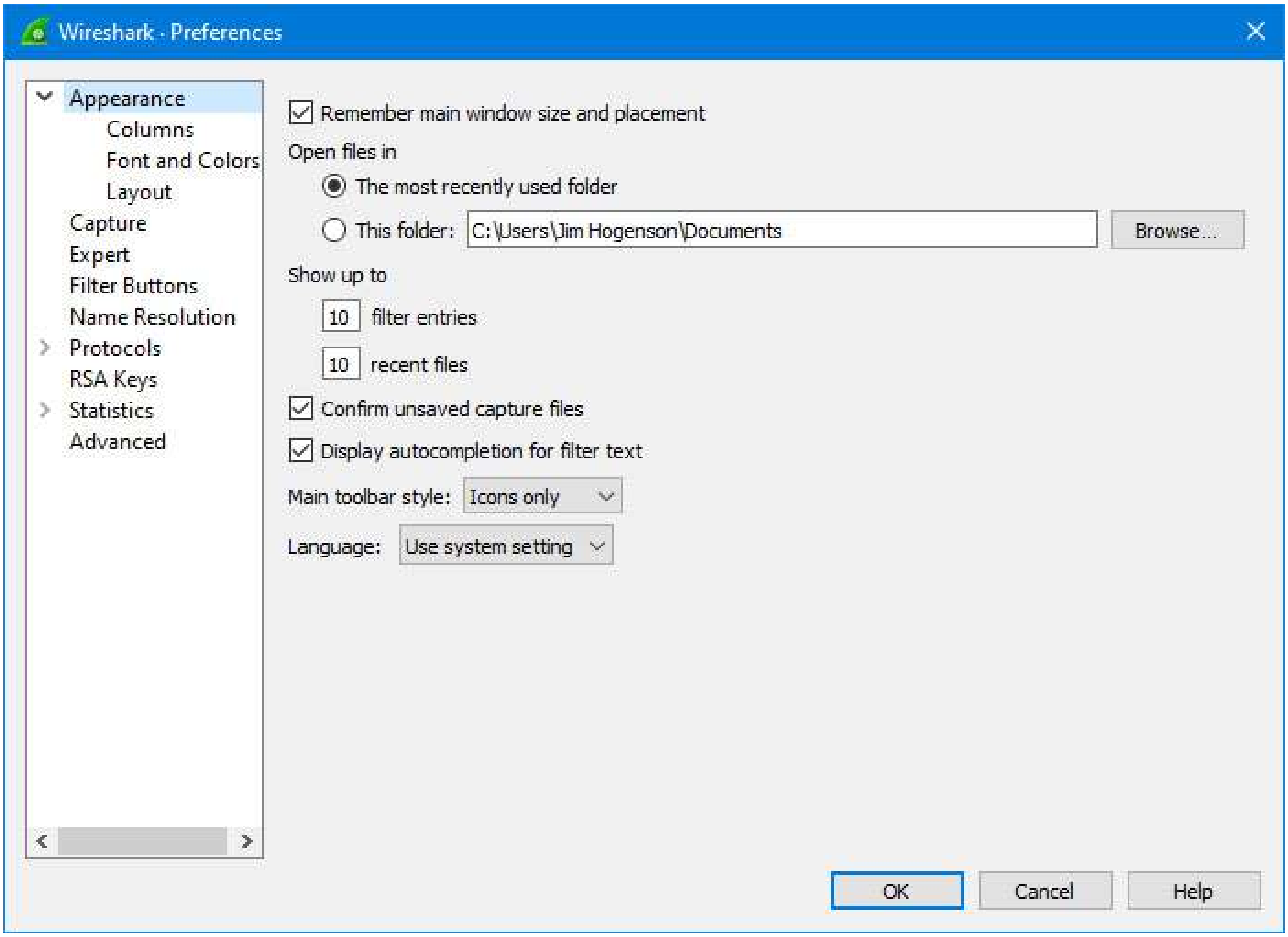
```
No.   Time      Source         Destination    Protocol  Length Info
  304 4.741960  192.168.1.112  192.168.1.125  SNMP      113 get-request
  307 4.744061  192.168.1.125  192.168.1.112  SNMP      174 report 1.3.6.1.6.3.15.1.1.4.0
  308 4.744505  192.168.1.112  192.168.1.125  SNMP      205 encryptedPDU: privKey Unknown
  309 4.751867  192.168.1.125  192.168.1.112  SNMP      205 encryptedPDU: privKey Unknown
```

```
> Frame 304: 113 bytes on wire (904 bits), 113 bytes captured (904 bits) on interface \Device\NPF_{193A9AF7-FE82-4902-9A83-D8CB2D41E36
> Ethernet II, Src: Dell_a2:b9:f6 (a4:bb:6d:a2:b9:f6), Dst: Digiboar_45:46:4e (00:40:9d:45:46:4e)
> Internet Protocol Version 4, Src: 192.168.1.112, Dst: 192.168.1.125
> User Datagram Protocol, Src Port: 50294, Dst Port: 161
v Simple Network Management Protocol
    msgVersion: snmpv3 (3)
  > msgGlobalData
    msgAuthoritativeEngineID: <MISSING>
    msgAuthoritativeEngineBoots: 0
    msgAuthoritativeEngineTime: 0
    msgUserName: initial
    msgAuthenticationParameters: <MISSING>
    msgPrivacyParameters: <MISSING>
```

```
0000  00 40 9d 45 46 4e a4 bb  6d a2 b9 f6 08 00 45 00   ·@·EFN·· m·····E·
0010  00 63 28 b5 00 00 80 11  00 00 c0 a8 01 70 c0 a8   ·c(····· ·····p··
0020  01 7d c4 76 00 a1 00 4f  84 9e 30 45 02 01 03 30   ·}·v···O ··0E···0
0030  11 02 04 37 80 47 47 02  03 00 80 00 04 01 04 02   ···7·GG· ········
0040  01 03 04 17 30 15 04 00  02 01 00 02 01 00 04 07   ····0··· ········
0050  69 6e 69 74 69 61 6c 04  00 04 00 30 14 04 00 04   initial· ···0····
0060  00 a0 0e 02 04 37 80 47  47 02 01 00 02 01 00 30   ·····7·G G······0
```

wireshark_Ethernet_20200819100913_a10188.pcapng
Packets: 460 · Displayed: 4 (0.9%)
Profile: Default

Under the Edit menu, select Preferences.



☑ Remember main window size and placement

Open files in

● The most recently used folder

○ This folder: C:\Users\Jim Hogenson\Documents

Show up to

10 filter entries

10 recent files

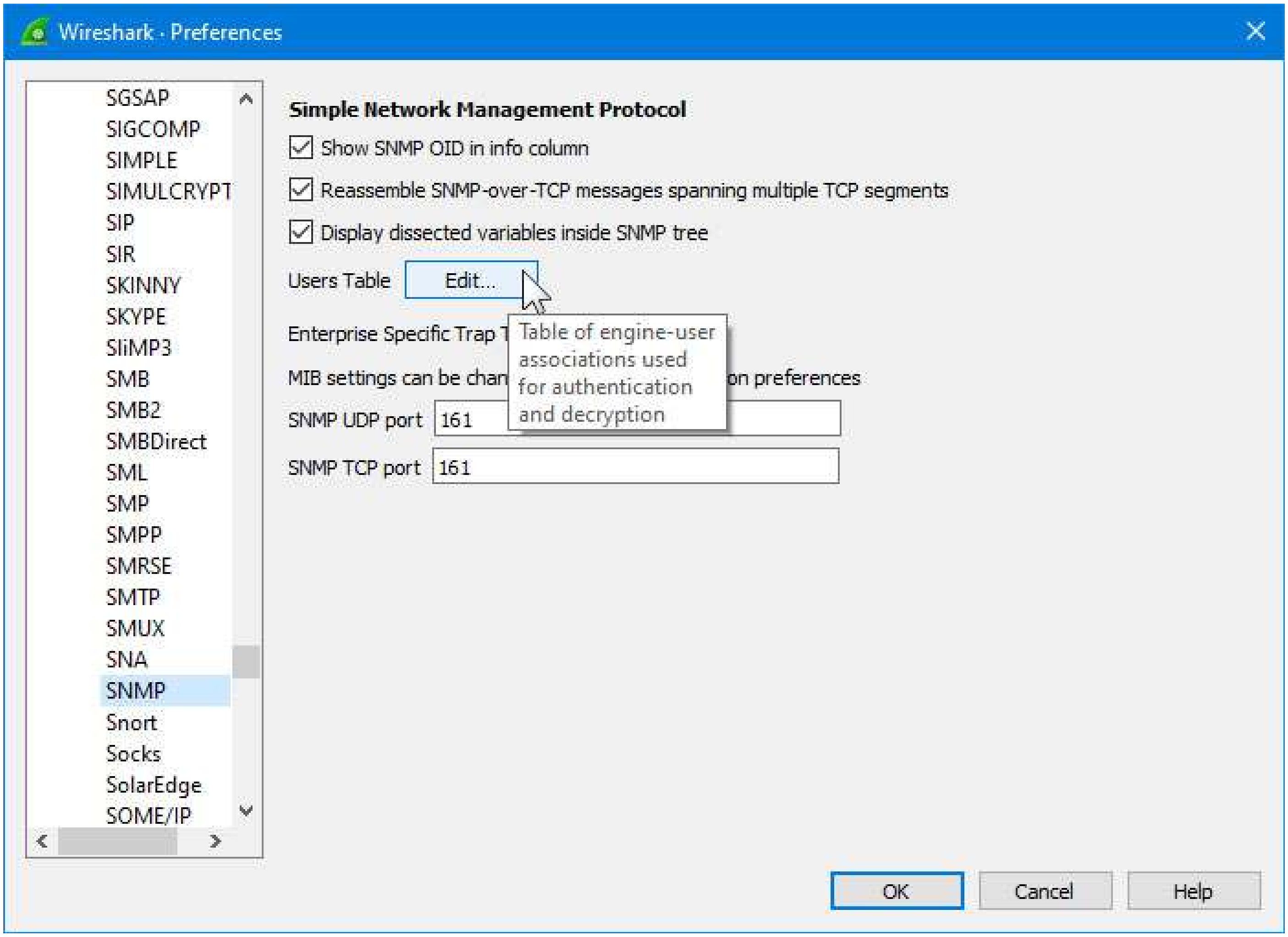
☑ Confirm unsaved capture files

☑ Display autocompletion for filter text

Main toolbar style: Icons only

Language: Use system setting

Expand the Protocols list and scroll down to SNMP. Select SNMP.



Simple Network Management Protocol

☑ Show SNMP OID in info column

☑ Reassemble SNMP-over-TCP messages spanning multiple TCP segments

☑ Display dissected variables inside SNMP tree

Users Table Edit...

Table of engine-user associations used for authentication and decryption

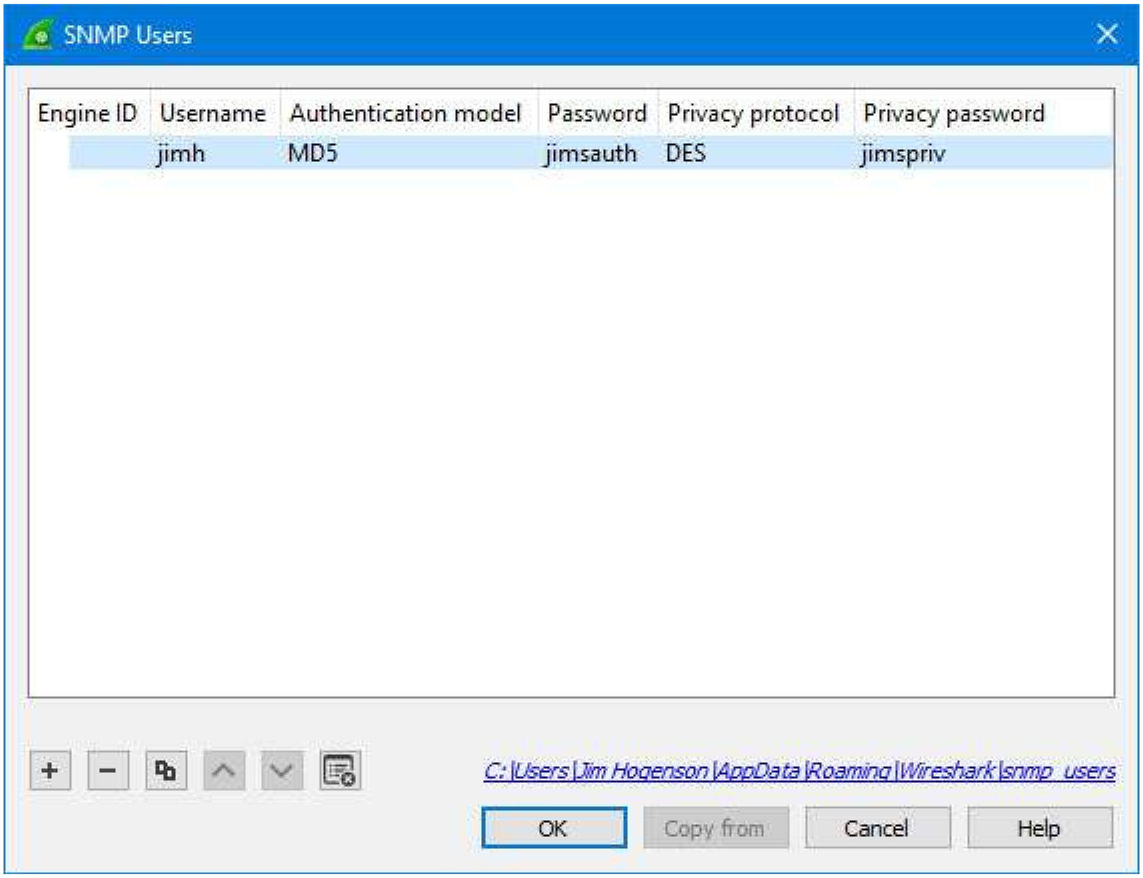
Enterprise Specific Trap T

MIB settings can be chan on preferences

SNMP UDP port 161

SNMP TCP port 161

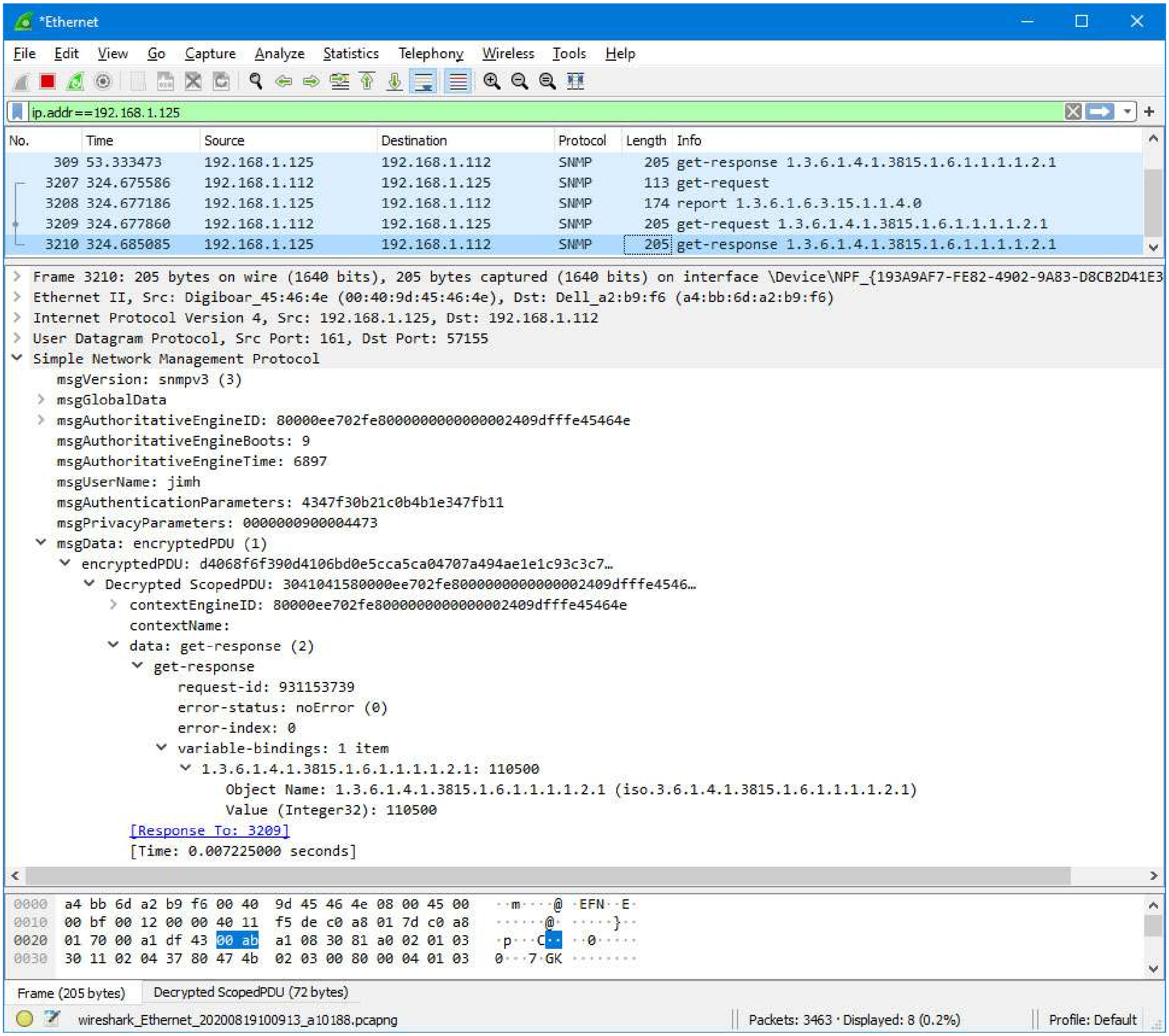
Next, click Edit (Users Table). Enter the user name and credentials being used in the Get request you wish to capture.



Engine ID	Username	Authentication model	Password	Privacy protocol	Privacy password
	jimh	MD5	jimsauth	DES	jimspriv

C:\Users\Jim Hogenson\AppData\Roaming\Wireshark\snmp_users

Once Wireshark has been provided with the user name and its associated credentials, Wireshark can show the full packet.


*Ethernet
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.addr==192.168.1.125

```
No.   Time        Source         Destination    Protocol Length Info
  309 53.333473   192.168.1.125  192.168.1.112  SNMP     205 get-response 1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1
 3207 324.675586  192.168.1.112  192.168.1.125  SNMP     113 get-request
 3208 324.677186  192.168.1.125  192.168.1.112  SNMP     174 report 1.3.6.1.6.3.15.1.1.4.0
 3209 324.677860  192.168.1.112  192.168.1.125  SNMP     205 get-request 1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1
 3210 324.685085  192.168.1.125  192.168.1.112  SNMP     205 get-response 1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1
```

```
> Frame 3210: 205 bytes on wire (1640 bits), 205 bytes captured (1640 bits) on interface \Device\NPF_{193A9AF7-FE82-4902-9A83-D8CB2D41E3
> Ethernet II, Src: Digiboar_45:46:4e (00:40:9d:45:46:4e), Dst: Dell_a2:b9:f6 (a4:bb:6d:a2:b9:f6)
> Internet Protocol Version 4, Src: 192.168.1.125, Dst: 192.168.1.112
> User Datagram Protocol, Src Port: 161, Dst Port: 57155
v Simple Network Management Protocol
    msgVersion: snmpv3 (3)
  > msgGlobalData
  > msgAuthoritativeEngineID: 80000ee702fe80000000000000002409dfffe45464e
    msgAuthoritativeEngineBoots: 9
    msgAuthoritativeEngineTime: 6897
    msgUserName: jimh
    msgAuthenticationParameters: 4347f30b21c0b4b1e347fb11
    msgPrivacyParameters: 0000000900004473
  v msgData: encryptedPDU (1)
    v encryptedPDU: d4068f6f390d4106bd0e5cca5ca04707a494ae1e1c93c3c7…
      v Decrypted ScopedPDU: 3041041580000ee702fe80000000000000002409dfffe4546…
        > contextEngineID: 80000ee702fe80000000000000002409dfffe45464e
          contextName:
        v data: get-response (2)
          v get-response
              request-id: 931153739
              error-status: noError (0)
              error-index: 0
            v variable-bindings: 1 item
              v 1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1: 110500
                  Object Name: 1.3.6.1.4.1.3815.1.6.1.1.1.1.2.1 (iso.3.6.1.4.1.3815.1.6.1.1.1.1.2.1)
                  Value (Integer32): 110500
          [Response To: 3209]
          [Time: 0.007225000 seconds]
```

```
0000  a4 bb 6d a2 b9 f6 00 40  9d 45 46 4e 08 00 45 00   ··m····@ ·EFN··E·
0010  00 bf 00 12 00 00 40 11  f5 de c0 a8 01 7d c0 a8   ······@· ·····}··
0020  01 70 00 a1 df 43 00 ab  a1 08 30 81 a0 02 01 03   ·p···C·· ··0·····
0030  30 11 02 04 37 80 47 4b  02 03 00 80 00 04 01 03   0···7·GK ········
```

Frame (205 bytes)   Decrypted ScopedPDU (72 bytes)

wireshark_Ethernet_20200819100913_a10188.pcapng   Packets: 3463 · Displayed: 8 (0.2%)   Profile: Default

Babel Buster® 3
MODEL BB3-6101
MODEL MX-61
NETWORK GATEWAY

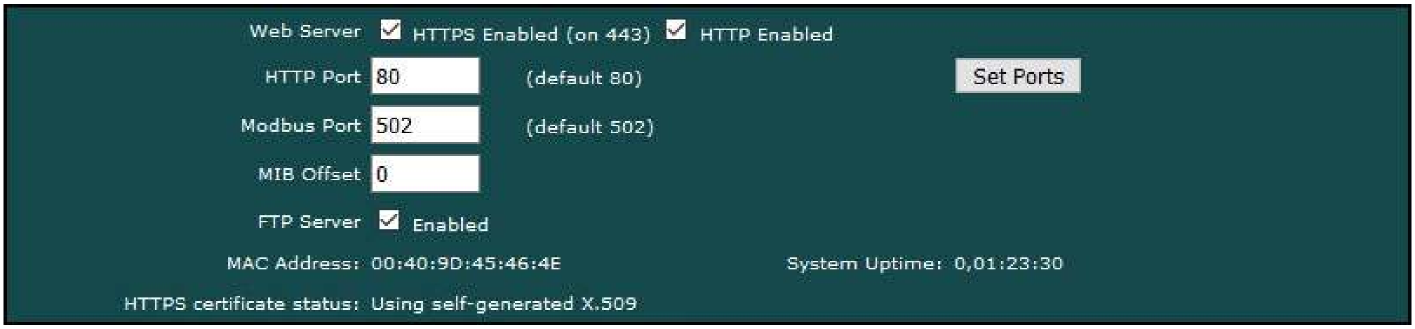
CONTROL SOLUTIONS MINNESOTA

Appendix E SSL Certificates for Secure Web (HTTPS)

The secure web server (HTTPS) requires SSL certificates in order to establish secure connections. The HTTPS certificates are only required if HTTPS is enabled on the Network configuration page in the Babel Buster BB3-6101/MX-61.

E.1 X.509 Auto-Certificate Generation

The Babel Buster BB3-6101/MX-61 Gateway will automatically generate X.509 certificates if no external certificates are found or could not be loaded correctly. These will be generated one time and saved in the Flash file system for subsequent reuse. When the self-generated X.509 certificates are in use, this will be indicated at the bottom of the Network configuration page.



Web Server ☑ HTTPS Enabled (on 443) ☑ HTTP Enabled

HTTP Port 80 (default 80)

Modbus Port 502 (default 502)

MIB Offset 0

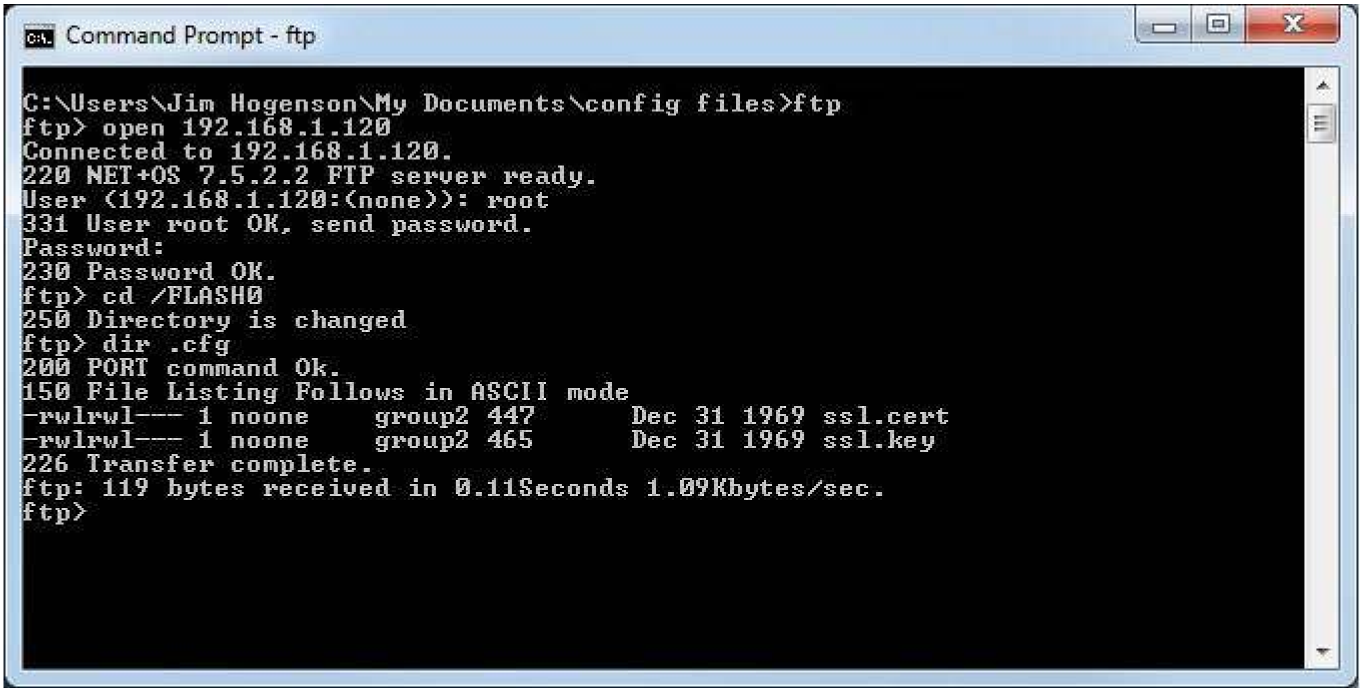
FTP Server ☑ Enabled

MAC Address: 00:40:9D:45:46:4E

System Uptime: 0,01:23:30

HTTPS certificate status: Using self-generated X.509

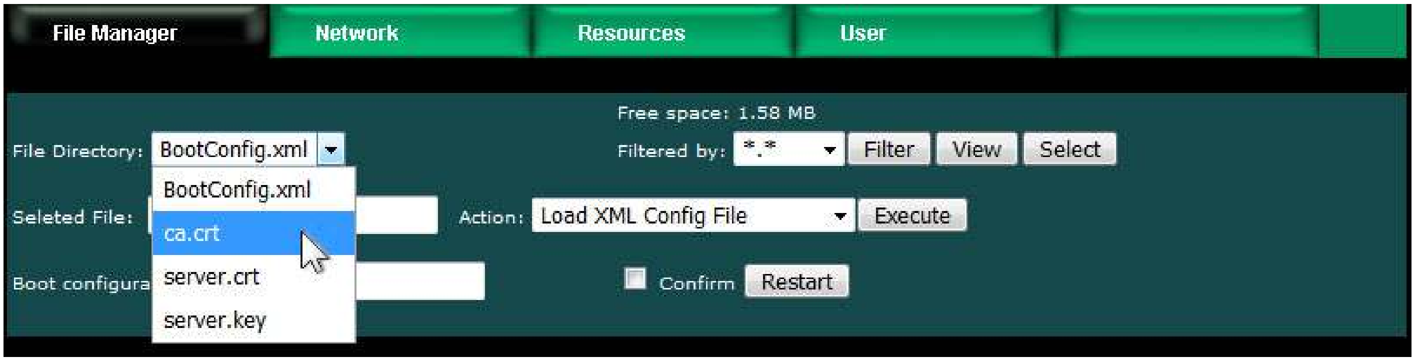
If there is a need to delete the self-generated certificates, you can do so by logging in via FTP. Change directory to /FLASH0, then to .cfg. The two certificate files that were self-generated are ssl.cert and ssl.key.



```
C:\Users\Jim Hogenson\My Documents\config files>ftp
ftp> open 192.168.1.120
Connected to 192.168.1.120.
220 NET+OS 7.5.2.2 FTP server ready.
User (192.168.1.120:(none)): root
331 User root OK, send password.
Password:
230 Password OK.
ftp> cd /FLASH0
250 Directory is changed
ftp> dir .cfg
200 PORT command Ok.
150 File Listing Follows in ASCII mode
-rwlrwl--- 1 noone    group2 447      Dec 31 1969 ssl.cert
-rwlrwl--- 1 noone    group2 465      Dec 31 1969 ssl.key
226 Transfer complete.
ftp: 119 bytes received in 0.11Seconds 1.09Kbytes/sec.
ftp>
```

E.2 External Certificates

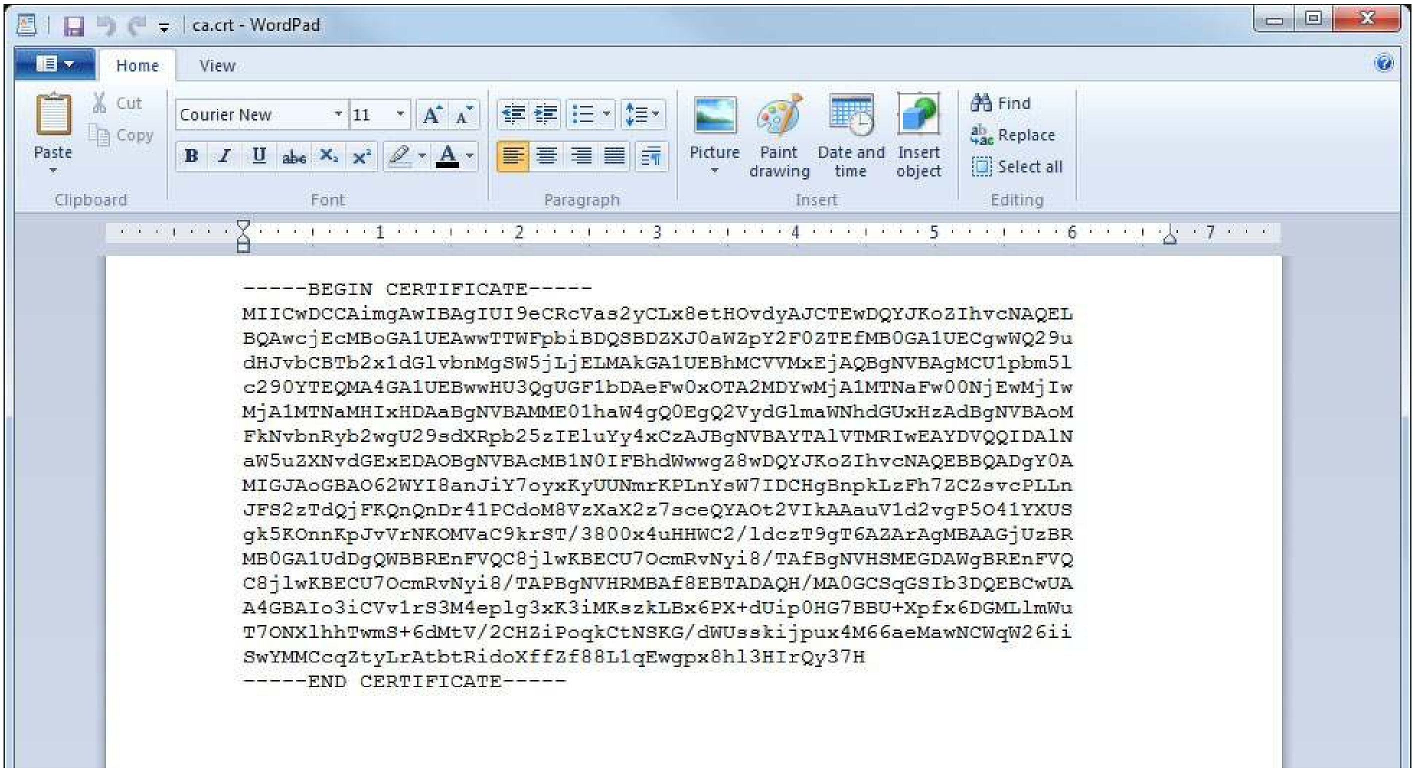
There are three certificates that you must generate and upload to use SSL certificates other than the self-generated X.509 certificates.



The required certificates are as follows, and must use exactly these names.

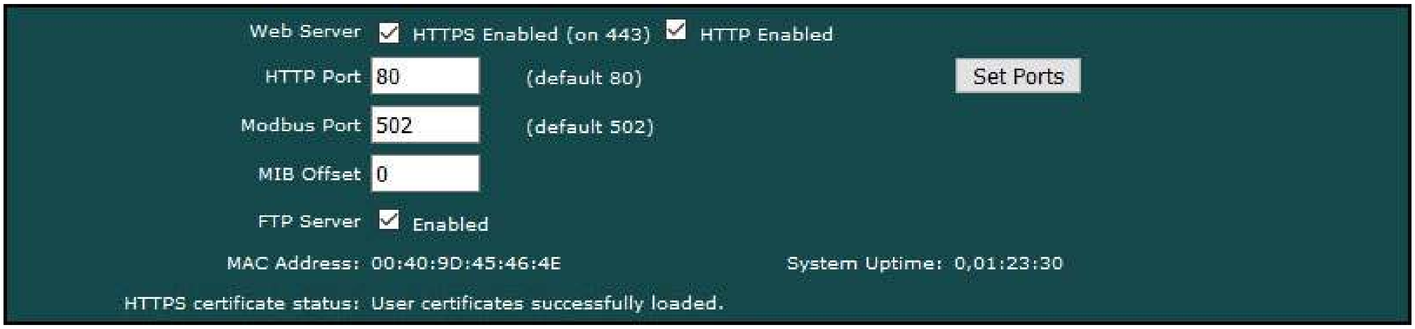
ca.crt	CA Root certificate in PEM format
server.crt	Server certificate in PEM format
server.key	Server private key in PEM format

The content of each certificate file will look something like the screen shot below. If you require external certificates for your secure web server, the requirement was likely imposed by your IT department. They should be able to provide the necessary certificates for you. For globally accessed use, the Root CA would come from somebody like GoDaddy or DigiCert (formerly Symantec).



```
-----BEGIN CERTIFICATE-----
MIICwDCCAimgAwIBAgIUI9eCRcVas2yCLx8etHOvdyAJCTEwDQYJKoZIhvcNAQEL
BQAwcjEcMBoGA1UEAwwTTWFpbiBDQSBDZXJ0aWZpY2F0ZTEfMB0GA1UECgwWQ29u
dHJvbCBTb2x1dGlvbnMgSW5jLjELMAkGA1UEBhMCVVMxEjAQBgNVBAgMCU1pbm5l
c290YTEQMA4GA1UEBwwHU3QgUGF1bDAeFw0xOTA2MDYwMjA1MTNaFw00NjEwMjIw
MjA1MTNaMHIxHDAaBgNVBAMME01haW4gQ0EgQ2VydGlmaWNhdGUxHzAdBgNVBAoM
FkNvbnRyb2wgU29sdXRpb25zIEluYy4xCzAJBgNVBAYTAlVTMRIwEAYDVQQIDAlN
aW5uZXNvdGExEDAOBgNVBAcMB1N0IFBhdWwwgZ8wDQYJKoZIhvcNAQEBBQADgY0A
MIGJAoGBAO62WYI8anJiY7oyxKyUUNmrKPLnYsW7IDCHgBnpkLzFh7ZCZsvcPLLn
JFS2zTdQjFKQnQnDr41PCdoM8VzXaX2z7sceQYAOt2VIkAAauV1d2vgP5O41YXUS
gk5KOnnKpJvVrNKOMVaC9krST/3800x4uHHWC2/ldczT9gT6AZArAgMBAAGjUzBR
MB0GA1UdDgQWBBREnFVQC8jlwKBECU7OcmRvNyi8/TAfBgNVHSMEGDAWgBREnFVQ
C8jlwKBECU7OcmRvNyi8/TAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBCwUA
A4GBAIo3iCVv1rS3M4eplg3xK3iMKszkLBx6PX+dUip0HG7BBU+Xpfx6DGMLlmWu
T7ONXlhhTwmS+6dMtV/2CHZiPoqkCtNSKG/dWUsskijpux4M66aeMawNCWqW26ii
SwYMMCcqZtyLrAtbtRidoXffZf88L1qEwgpx8hl3HIrQy37H
-----END CERTIFICATE-----
```

If external certificates were loaded successfully, that will be indicated at the bottom of the Network configuration page.



Web Server ☑ HTTPS Enabled (on 443) ☑ HTTP Enabled

HTTP Port 80 (default 80)

Modbus Port 502 (default 502)

MIB Offset 0

FTP Server ☑ Enabled

MAC Address: 00:40:9D:45:46:4E System Uptime: 0,01:23:30

HTTPS certificate status: User certificates successfully loaded.

E.3 Certificate Generation Script (Linux)

The art and science of generating SSL certificates is beyond the scope of this document. An example SSL certificate generation script is provided here as a reference.

The following script, run on a Linux system with OpenSSL installed, will generate the three required SSL certificate files. It will generate a number of intermediate files as well - you don't need to upload them. Replace references to Control Solutions in this script with your own company name.

```
#!/bin/bash
echo hello
# This will create some self signed certs, using one master CA.
#
# these can be the webserver DNS name, or an IP address, however you
access
```

```
# the resource, this needs to match.
if [ -z "$1"] || [ -z "$2"]; then
echo 'Usage: gen.sh <server-name> <client-name>'
echo ' <server-name> and <client-name> can be IP addresses'
echo ' or DNS names.'
exit 1
fi
SNAME=$1
CNAME=$2
#
# Bits for strength, 1024, 2048, 4096, etc.. (suggest 2k or 4k for web
servers)
BITS=1024
#
# HASH - Options are sha256, sha512, sha1, md5
HASH="sha256"
SN=`date +%Y%m%d%H%M%S`
################
# below is the entry for the CRL
# Do not use http://www.csimn.com/crl.pem for production keys and
certificates
# cat <<EOF >> extensions.cnf
# [ extensions_section ]
# crlDistributionPoints = URI:http://www.csimn.com/crl.pem
#
# basicConstraints = CA:FALSE
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment
# subjectAltName = DNS:${SNAME},IP:${SNAME}
# EOF
#################################################################
#################################################################
# first, lets generate some private keys...
openssl genrsa -out server.key ${BITS}
openssl genrsa -out client.key ${BITS}
# ok, and now the MAIN CA
openssl req -x509 -${HASH} -nodes -days 10000 -newkey rsa:${BITS} -keyout
ca.key -out ca.crt -subj "/CN=Main CA Certificate/O=Control Solutions
Inc./C=US/ST=Minnesota/L=St Paul"
######
#
# Create a CSR for both server and client
# Replace these values with one appropriate for your organization
openssl req -out server.csr -key server.key -new -subj "/CN=${SNAME}
/O=Control Solutions Inc./C=US/ST=Minnesota/L=St Paul"
openssl req -out client.csr -key client.key -new -subj "/CN=${CNAME}
/O=Control Solutions Inc./C=US/ST=Minnesota/L=St Paul"
#
#
######
# Sign the keys with the CA
openssl x509 -req -days 3650 -in server.csr -CA ca.crt -CAkey ca.key
-set_serial ${SN}01 -out server.crt -${HASH}
openssl x509 -req -days 3650 -in client.csr -CA ca.crt -CAkey ca.key
```

```
-set_serial ${SN}02 -out client.crt -${HASH}
# Create a windows file to import the client keys if needed in this
format
openssl pkcs12 -export -clcerts -in client.crt -inkey client.key -out
client.p12
# Create the client keys as a complete pem file if needed in this format
openssl pkcs12 -in client.p12 -out client-full.pem -clcerts
# mv -f server.key svrkey.pem
# mv -f server.crt svrcert.pem
# mv -f client.key clntkey.pem
# mv -f client.crt clntcert.pem
# cp -f ca.crt cacert.pem
####
# cleanup
# rm -f client.csr server.csr
#DLS 20160420
echo '***********************************************************'
echo '* WARNING: Do not use this script to generate production *'
echo '* keys and certificates. This script is for *'
echo '* demonstration purposes only. *'
echo '***********************************************************'
```

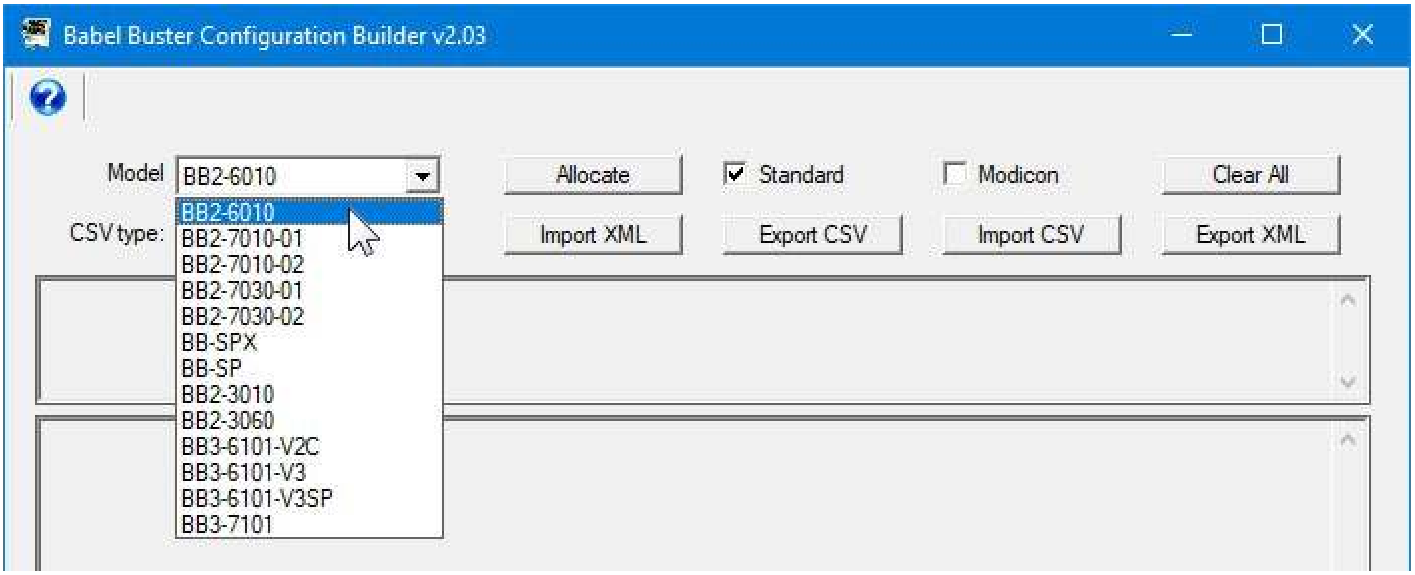
Appendix F Converting Older XML Files

F.1 BB2-6010/SPX to BB3-6101/MX-61 Conversion

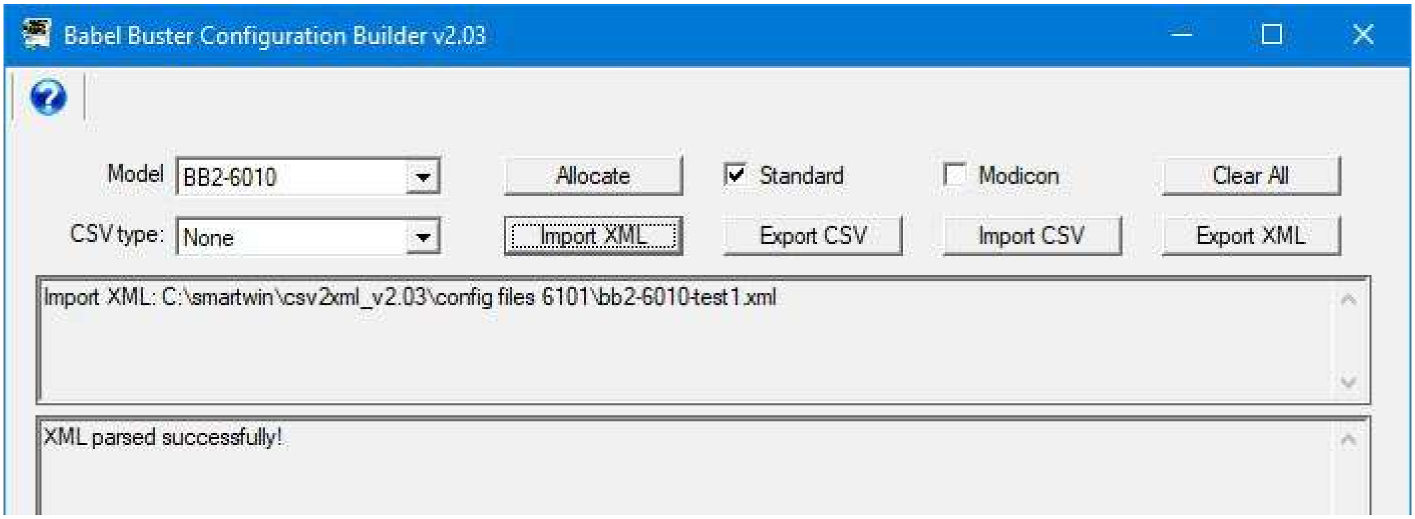
You cannot load a BB2-6010 configuration XML file into the BB3-6101, but you can easily convert it into a BB3-6101 XML file using the Babel Buster Configuration Builder tool available under the Support/Tools link at csimn.com.

In the examples below, SPX may be substituted for BB2-6010 and MX-61 may be substituted for BB3-6101. NOTE: An XML file exported for BB3-6101-V2C will load correctly in an MX-61-V2C.

Converting the file is simple. Start by selecting the BB2-6010 model.



Click the Import XML button. The Windows file dialog window will open. Select your XML file from wherever you saved it on your PC. Upon opening the XML file, it will read the file into the configuration builder tool.



Model BB2-6010

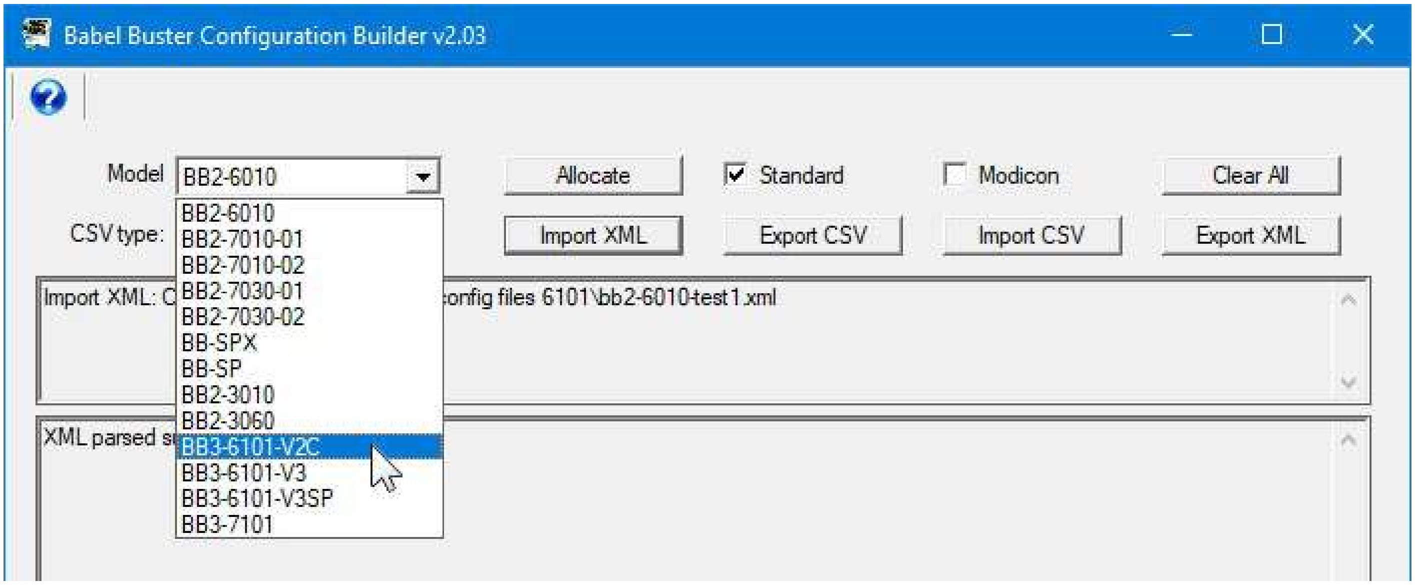
CSV type: None

☑ Standard □ Modicon

Import XML: C:\smartwin\csv2xml_v2.03\config files 6101\bb2-6010-test1.xml

XML parsed successfully!

Now go back to the model list and switch to BB3-6101.



Model BB2-6010

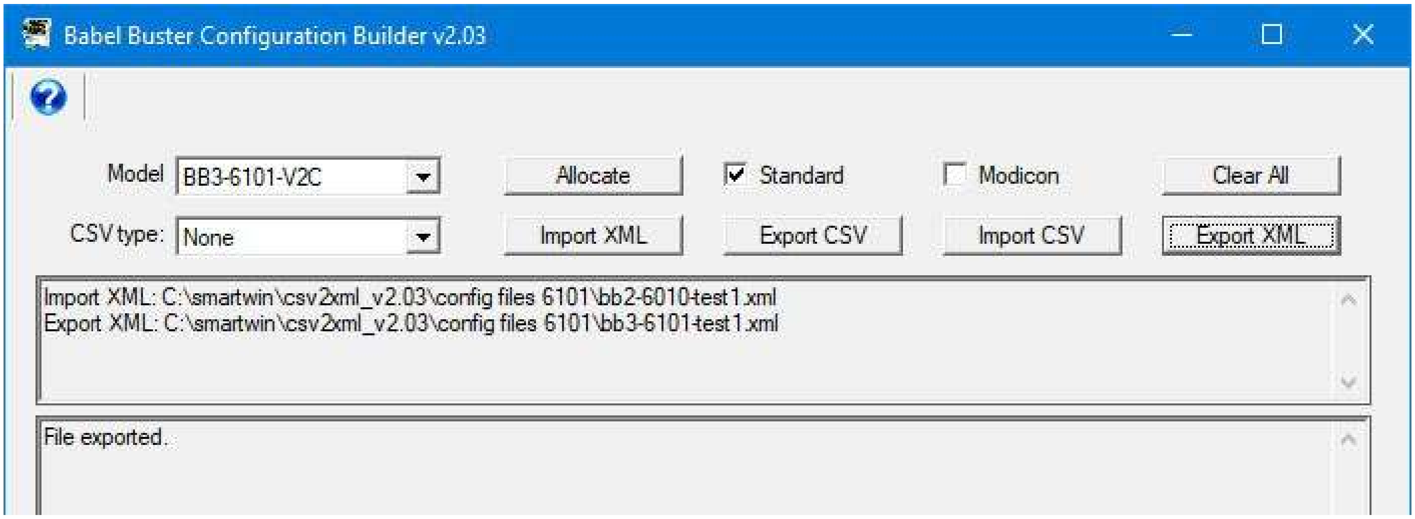
CSV type:

☑ Standard □ Modicon

Import XML: C ... onfig files 6101\bb2-6010-test1.xml

XML parsed s

Click the Export XML button. The Windows file dialog window will open again, this time wanting you to provide a name for the new XML file you are about to create. Choose a different name than the one you imported so that you don't lose your original file. Once the export is complete, you may now upload this new file to your BB3-6101 gateway.



Import XML: C:\smartwin\csv2xml_v2.03\config files 6101\bb2-6010-test1.xml
Export XML: C:\smartwin\csv2xml_v2.03\config files 6101\bb3-6101-test1.xml

File exported.

Note that there are two things not converted by the BB2-6010 to BB3-6101 conversion process: Resource allocation and SNMP trap rules.

Resource allocation is not generated in the Babel Buster 3 XML file because the older gateway did not have this feature. The resource section will be excluded from the XML file. This means that whatever resource allocation you previously had in effect in the BB3-6101 will remain as is. If your attempt to load the new XML file complains about invalid register number, check to see that it is not trying to reference a higher numbered local register than what you have allocated. Adjust your allocation if needed.

The other thing that cannot be converted is SNMP trap (notification) rules. This is because the trap structure in the Babel Buster 3 series gateway is too different to be able to make much use of the Babel Buster 2 trap rules. Rather than create a list of rules that cannot possibly be correct, the conversion process requires you to explicitly manipulate them. Note, however, that you can export the Babel Buster 2 trap rules as a CSV file, edit in bulk using a spread sheet program, then import the CSV back into the configuration builder as Babel Buster 3 trap rules. Then proceed to generate a new XML file.

F.2 CSV File Export

The BB3-6101 will not export CSV files directly. To create CSV files from an existing BB3-6101 configuration, download a copy of the file to your PC, and then use the Babel Buster Configuration Builder to export CSV files as needed. To save a copy of the XML configuration file to your PC, select the file and click the View button on the File Manager page in the BB3-6101. 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).

The Babel Buster Configuration Builder will export more types of CSV files than the BB3-6101 itself will import. The most frequently used CSV files - namely read/write maps for Modbus RTU and Modbus TCP - can be imported directly into the BB3-6101. Importing CSV lists of read/write maps assumes you have already created local registers.

Importing device lists directly into the BB3-6101 is not supported. You generally only need to enter things like IP address once, and any one IP address can only be used one time. The effort to enter the IP addresses into the web UI of the BB3-6101 is no greater than the effort to enter them into a spread sheet and then save and import. In fact, when it comes to setting up device lists, the CSV import would usually end up being more work. If you do have a device list in the form of a CSV file, import that into the configuration builder tool, export as XML (along with any other CSV files you import as part of the configuration), and then upload the resulting XML file to the BB3-6101.