

Babel Buster™ 232 LonWorks® Gateway
LonWorks® to Modbus/RTU
User Guide



CONTROL SOLUTIONS, INC.

2179 FOURTH STREET, SUITE 2-G • PO BOX 10789
WHITE BEAR LAKE, MN 55110-0789

www.csimn.com

Control Solutions, Inc.
Babel Buster 232 LonWorks Gateway
LonWorks to Modbus/RTU
User Guide

Table of Contents

Please read at least the first 2 paragraphs of section 9 if you chose to skip reading the entire manual.

1. Introduction	1
2. Modbus Master, Slave, and Client/Server	2
3. Modbus Registers	3
4. Data Translation	4
5. LonMark Functional Profiles	5
6. Node Object	6
7. Open Loop Sensor Object.	9
8. Open Loop Actuator Object	13
9. Getting Started	17
10. Functional Block (Object) Status	18
11. Changing Network Variable Type.	21
12. Power Supply & Comm Port Wiring	23
13. Front Panel Operation & LED Indications	24
14. Connector Terminals.	25
15. Opening the Case	26
16. Jumper Settings	26
17. Mounting on DIN Rail	27
18. Trouble Shooting.	27
19. Trouble Shooting Tools.	29
20. A Concise Guide to RS-232 (Reference)	31
21. Specifications	33

Control Solutions, Inc.
Babel Buster 232 LonWorks Gateway
LonWorks to Modbus/RTU

User Guide
Rev. 1.0 • September 2005

IMPORTANT SAFETY CONSIDERATIONS:

Proper system design is required for reliable and safe operation of distributed control systems incorporating Babel Buster series modules and other such devices. 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 the Babel Buster series module or any other 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.

© 2005 Control Solutions, Inc.

LonWorks, LonMaker, LonMark, Neuron Chip, and Echelon are trademarks of Echelon Corporation registered in the United States and other countries. All other trademarks mentioned in this document are the property of their respective owners. Information in this document is subject to change without notice and does not represent a commitment on the part of Control Solutions, Inc. This document 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 manual or in the product(s) and/or the program(s) described in this manual at any time. This product could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes may be incorporated in new editions of the publication.



1. Introduction to Babel Buster

What is Babel Buster? Technically a gateway, it is essentially a “language translator” or more appropriately, a protocol translator. It translates Modbus protocol to LonWorks protocol and vice versa. More specifically, it maps Modbus registers to LonWorks network variables and vice versa.

Modbus registers and LonWorks network variables have this in common: They are each an individual element of data accessible via a network. A Modbus device exposes itself to the world via its registers. A LonWorks device exposes itself to the world via its network variables. The data format of Modbus registers and LonWorks network variables are not generally interchangeable. Furthermore, the physical interface is often different.

Babel Buster provides two physical interfaces, one for the Modbus connection, and one for the LonWorks connection. The LonWorks connection is free topology TP/FT-10 operating at 78kbps. There are two versions of Babel Buster, the 232 and the 10/100. Babel Buster 232 provides a Modbus connection having an RS-232 serial interface operating at 9600 to 38,400 baud, communicating via the Modbus RTU protocol. Babel Buster 10/100 provides a Modbus connection having an Ethernet connection, auto sensing 10BaseT or 100BaseT, and communicating via the Modbus/TCP protocol.

Babel Buster provides a logical connection between Modbus registers and LonWorks network variables. All of the most commonly recognized Modbus registers formats are supported. On the LonWorks side, all scalar LonMark standard network variable types (SNVTs) are supported. Structures are not supported, with the exception of SNVT_switch, SNVT_lev_disc, and SNVT_state which are supported.

2. Modbus Master, Slave, and Client/Server

Modbus protocol is defined as a master/slave protocol, meaning a device operating as a master will poll one or more devices operating as a slave. This means a slave device cannot volunteer information; it must wait to be asked for it. The master will write data to a slave device's registers, and read data from a slave device's registers. A register address or register reference is always in the context of the slave's registers.

Modbus/TCP makes the definition of master and slave less obvious because Ethernet allows peer to peer communication. The definition of client and server are better known entities in Ethernet based networking. In this context, the slave becomes the server and the master becomes the client. There can be more than one client obtaining data from a server. In Modbus terms, this means there can be multiple masters as well as multiple slaves. Rather than defining master and slave on a physical device by device basis, it now becomes the system designer's responsibility to create logical associations between master and slave functionality.

Babel Buster 232 can operate as a master or a slave, but only in one mode at a time. The entire gateway is assigned either master or slave functionality. Modbus register references when operating in slave mode mean a master can access these registers in the Babel Buster gateway. Modbus register references when operating in master mode mean the Babel Buster gateway will attempt to access these registers in the remote slave device.

Babel Buster 10/100 provides both a client and a server which function simultaneously. Each functional block is individually assigned master/slave status. As a server (slave), the Modbus register references define the addresses that a remote TCP client will use to access that data in the Babel Buster. As a client (master), the Modbus register references define the addresses that the Babel Buster gateway will attempt to access in a remote server.

Regardless of whether 232 or 10/100, a functional block operating as a slave will be a passive member of the Modbus network simply waiting for a master (or client) to read or write data. A functional block operating as a master will be an active member of the Modbus network, initiating data transfers as determined by its configuration. A functional block defined as a master, and associated with a LonWorks input network variable, will initiate a Modbus data transfer upon each network variable update, and optionally periodically. A functional block defined as master, and associated with a LonWorks output network variable, will initiate a Modbus data transfer periodically at a rate set in configuration properties, and will update the output network variable according to the send on delta, minimum and maximum send times, etc., as defined by LonMark functional profiles.

3. Modbus Registers

Commonly used Modbus register formats which are supported by Babel Buster include the following:

- Bit or Boolean
- Signed or unsigned 16-bit integer
- Signed or unsigned 32-bit integer
- Floating point (single precision IEEE-754 big endian)

Modbus registers defined as 32-bit integer or floating point are “double registers” meaning they occupy two 16-bit register addresses. If there is a mismatch in interpretation between the Babel Buster gateway and the Modbus device being interfaced, data transfer may return half of a floating point value as an integer (resulting in an invalid number), or may write a floating point to two consecutive integer registers (resulting in two invalid numbers). Other invalid combinations are possible. It is simply necessary to note that the Babel Buster must be configured to look for and provide the correct register format to avoid unexpected results.

Double registers are sometimes “swapped”. By default, the high order bytes are expected to be found in the first register and the low order bytes are expected to be found in the second register. If the reverse is true, double register swapping may be turned on by setting `nciGatewayOpts.bit6` to 1. (Note that “bit6” refers to the `SNVT_state` definition of bits in which `bit0` is the MSB and `bit15` is the LSB since the bits are defined as a structure.)

The types of registers commonly referenced in Modbus devices, and supported by Babel Buster, include the following:

- Coil (Discrete Output)
- Discrete Input
- Input Register
- Holding Register

Other register types, including memory reference, are defined in the protocol but not supported by Babel Buster.

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 currently supported by Babel Buster is 0 to 9999.

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. To simplify documentation, an old defacto standard widely recognized as Modbus is also used in Babel Buster. This defacto standard uses the first digit of a register reference to identify the register type. A less useful aspect of this defacto standard is that register references use a 1-based index while addresses use a 0-based index.

Register types and reference ranges recognized by Babel Buster 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. The reference 40001 will appear in documentation and is used to define the Modbus register in the location property of the functional block. The address 0000 will be transmitted in the message packet. Addresses are not directly used by the application or the user.

4. Data Translation

Modbus registers are 16-bit or 32-bit integers, or floating point. It is common for Modbus devices to use integer representation of scaled integers. For example, 68.5 degrees would be recorded as a value of 685 in a register typically documented with an “x10” notation meaning the value represents 10 times the actual value.

LonMark Standard Network Variable Types come with all sorts of scale factors. For SNVT_lev_percent, a single count represents 0.05%. For SNVT_temp, a single count represents 0.1 degree Celsius, and is offset by 274 degrees.

To see how data is tracked through the gateway, consider the conversion of SNVT_temp to a 16-bit integer Modbus register. Consider, as an example, a temperature of 21.4 degrees C. This is stored in the SNVT_temp in raw binary as 2954. This is stored in the gateway’s internal data buffer as a floating point 21.5. This will be transmitted to the Modbus register as 21 if the scale factor (SCPTgain) is set to 1,1. If the functional block’s SCPTgain is set to 10,1 (denoted as x10 in Modbus documentation), the number transmitted to a 16-bit integer Modbus register would be 214.

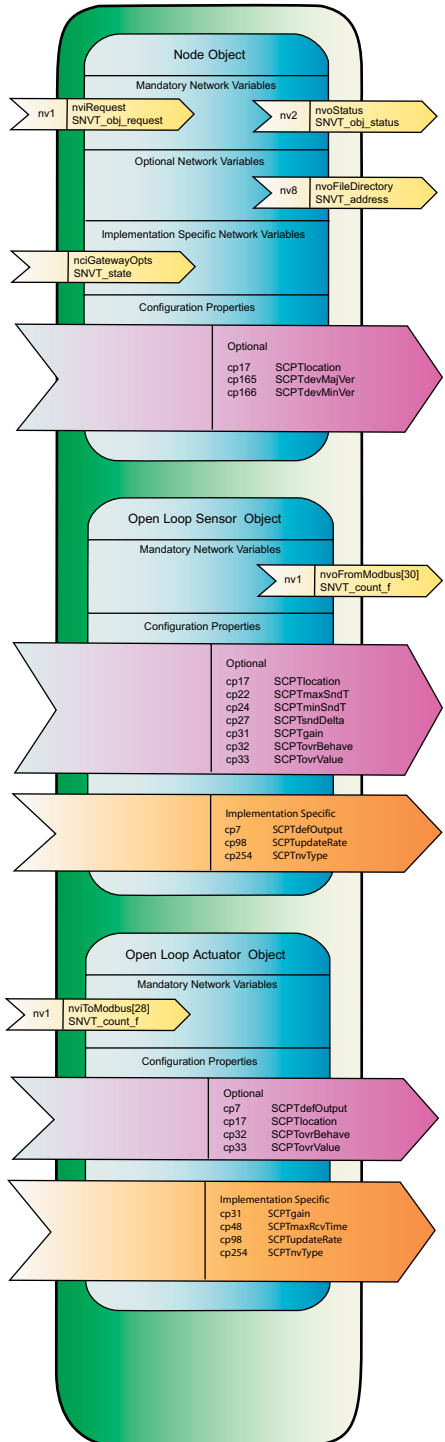
Data being sent to Modbus will be multiplied by the first factor in SCPTgain, and divided by the second factor, then formatted to the appropriate register type. This process is reversed when data is retrieved from Modbus. This scale factor is applied regardless of the format of the Modbus register. Therefore, the scaling will also be applied to a floating point register even though it is generally unnecessary to further scale a floating point value. The scale would generally be set at 1,1 (the default value) for floating point values. Scale factors other than powers of 10 are mathematically valid, but not commonly used in Modbus register definitions.

5. LonMark Functional Profiles

Each mapping relationship between a Modbus register and a LonMark network variable is defined as either a sensor or an actuator functional block. By definition, a LonMark sensor object derives data from a hardware device and presents that data to the LonWorks network via an output network variable. Conversely, a LonMark actuator object receives data from the LonWorks network via an input network variable and applies that data to a hardware device. In the case of Babel Buster, that “hardware device” is any device with a Modbus communications port.

Babel Buster sensor object output network variables are named `nvoFromModbus` because they provide data to the LonWorks network which is “From Modbus”. Babel Buster actuator object input network variables are named `nviToModbus` because they present data from the LonWorks network “To Modbus”.

Each “FromModbus” sensor and “ToModbus” actuator object have several configuration properties associated with them. A number of these properties define behavior which is standard to LonMark sensor and actuator objects. Additional properties are used to define Modbus behavior specific to the sensor and actuator objects.



The Babel Buster 232 functional block overview is shown here. There is a single node object, multiple open loop sensor objects, and multiple open loop actuator objects. Each of these functional blocks have configuration properties associated with them.

The following objects are implemented in the Babel Buster 232 LonWorks Gateway for Modbus/RTU:

0	Node Object
1-30	Open Loop Sensor
31-58	Open Loop Actuator

The node object is implemented per LonMark 3.3 standards. The sensor objects are used to move data from Modbus to LonWorks. The actuator objects are used to move data from LonWorks to Modbus. There is a one to one correspondence between network variables and Modbus registers. Each of these relationships is defined as a functional block per LonMark 3.3 standards.

6. Node Object

Object Request `nviRequest`

This network variable is used to request operations to be performed or modes to be specified for functional blocks within the device. The following requests are processed by all sensor and actuator objects:

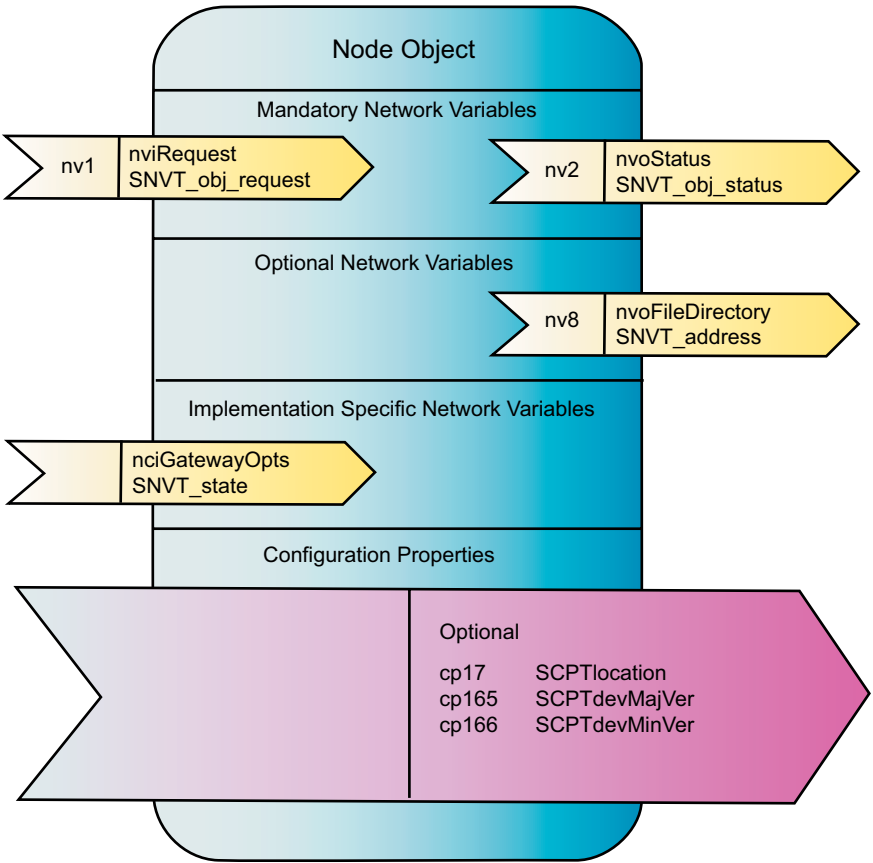
- RQ_NORMAL
- RQ_DISABLED
- RQ_UPDATE_STATUS
- RQ_REPORT_MASK
- RQ_OVERRIDE
- RQ_ENABLE
- RQ_RMV_OVERRIDE

Functional blocks must be enabled before they will produce data. The “enabled” status will be retained through power cycles. Override status will also be retained through power cycles.

Object Status `nvoStatus`

This output network variable reports the status for any functional block within the device. The following bit fields, as defined for `SNVT_obj_status`, are supported by Babel Buster:

- `invalid_id`
- `invalid_request`



disabled
 open_circuit
 feedback_failure
 unable_to_measure
 comm_failure
 in_override
 report_mask

The “invalid_request” bit will be set if an attempt is made to change a network variable to an invalid or unsupported SNVT type. The “unable_to_measure” bit will be set if an invalid Modbus register is referenced in the location string of the respective sensor or actuator object. The “comm_failure” and “open_circuit” indicate a communications failure in the attempt to connect with the Modbus device. All other bit fields are used as defined by LonMark standards.

File Directory `nvoFileDirectory`

SCPT values are accessed via the direct memory read/write file transfer method. The file directory is implemented as specified by LonMark standards, and is used primarily by network management tools.

Gateway Options `nciGatewayOpts`

This network variable, implemented in non-volatile memory, configures gateway options. The bits select options as follows for the RS-232 port:

- bit0 - 9600 baud N,8,1
- bit1 - 19,200 baud N,8,1
- bit2 - 38,400 baud N,8,1
- bit3 -
- bit4 -
- bit5 - Gateway is master if set (master=1, slave=0)
- bit6 - Double register swapping enabled if set
- bit7 -

The remaining bits are used differently by master and slave. If the gateway is operating as a master, the following applies:

- bit8 - 5 mSec slave response timeout
- bit9 - 10 mSec
- bit10 - 25 mSec
- bit11 - 50 mSec
- bit12 - 100 mSec (default)
- bit13 - 250 mSec
- bit14 - 500 mSec
- bit15 - 1000 mSec slave response timeout

If the gateway is operating as a slave, the following applies:

- bit8 - MSB of slave ID (ID defaults to 8)
- bit 9 - slave ID
- bit 10 - slave ID
- bit 11 - slave ID
- bit 12 - slave ID
- bit 13 - slave ID
- bit 14 - slave ID
- bit 15 - LSB of slave ID

Note that the bit names in the structure take on the opposite order of binary data

values, therefore the slave ID appears backwards as documented. If the gateway is the only Modbus device on the RS-232 port, the slave ID is unimportant except that it cannot be zero. Otherwise pick any 8-bit number not already used, and map it to the bit field (e.g., slave ID of 8 has 0,0,0,0,1,0,0,0 as last 8 bits of the options bit mask).

The gateway options are implemented as a SNVT_state. Therefore bits are accessed as members of a structure, starting with bit0 appearing first in the sequence of 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 as it appears in the LonMaker browser.

Node Location SCPTlocation

This configuration property may be used by the system integrator to store physical location information as desired.

Device Major Version SCPTdevMajVer

This configuration property is read-only, and provides software version information for this device as defined by LonMark standards.

Device Minor Version SCPTdevMinVer

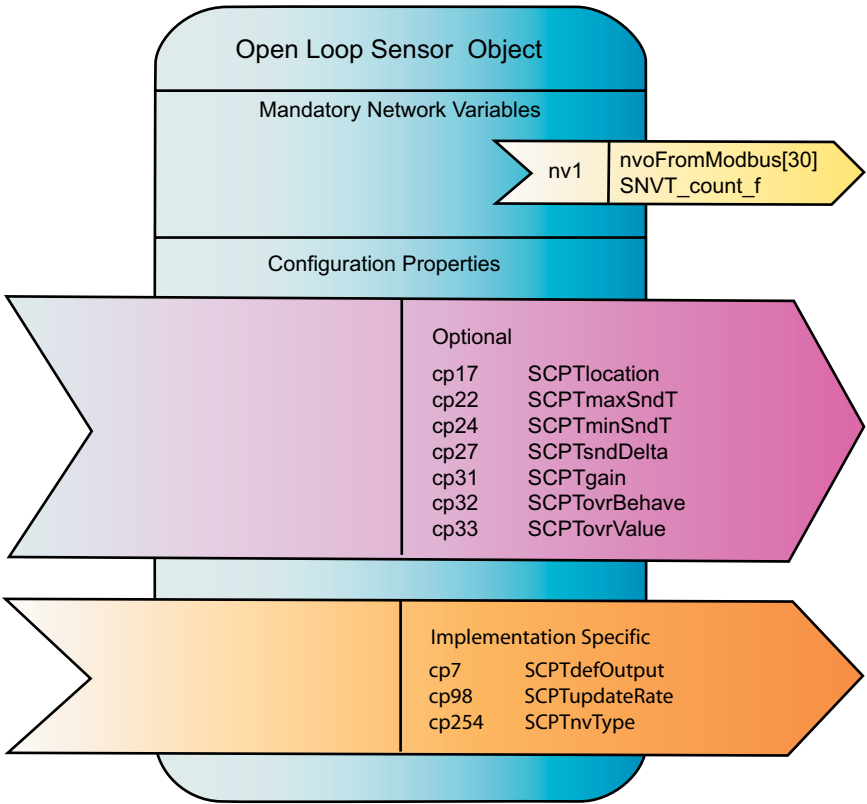
This configuration property is read-only, and provides software version information for this device as defined by LonMark standards.

7. Open Loop Sensor Object

Data From Modbus nvoFromModbus [30]

This network variable presents data obtained from a Modbus device to the LonWorks network. The default variable type is floating point SNVT_count_f, but may be changed by a network management tool.

To change the SNVT type from the LonMaker browser, right click on the variable to be changed, and select “Change Type...”. A dialog box will appear allowing the selection of a new type. It is not necessary to select a “same size” SNVT type. Any scalar type may be selected. In addition, SNVT_state, SNVT_lev_disc, and SNVT_switch are supported. The LonMaker browser will automatically change the corresponding SCPTnvType configuration property. It is not necessary to explicitly change this property.



Babel Buster software will recognize the SNVT type change when the functional block is enabled. It should be disabled while changing the type (and the Lon-Maker browser will do this automatically). Data conversions should be tested by the system integrator once configured. Overflow and underflow conditions are not necessarily bounded depending upon the type of conversion being made.

Default Network Variable Level SCPTdefOutput

This SCPTdeflftOut configuration property sets the default assumed by the output network variable when the functional block is overridden to the default value, or when the gateway has not received the expected response from the Modbus device. If an error has occurred which is indicated in the object status, the functional block will assume the default level. If maximum send time is non-zero, and maximum timeout is reached without any update from a Modbus master when operating in slave mode, a feedback_failure error bit will be set and the functional block will assume the default level. Maximum send time also causes a network variable update as defined by the LonMark functional profile.

Scale Factor SCPTgain

This configuration property sets the SCPTgain scale factor for scaling data obtained from the Modbus device. Manufacturers of Modbus devices often define register contents as X10, X100, etc. To translate these raw values to scaled values, SCPTgain is applied. If the Modbus register is not scaled, this property should be set to { 1, 1 } shown as “1/1” in the LonMaker browser. When scaling the data, the raw value obtained from the Modbus device is divided by the first number in the SCPTgain set, and multiplied by the second number.

To illustrate its use, suppose the Modbus manufacturer specifies that data in a register is presented as X10 meaning the integer value is tenths of degrees. A value of 72.5 degrees would be returned from the Modbus device as 725. To convert this properly, the SCPTgain should be { 10, 1 } or “10/1” so that 725 is divided by 10 before being converted to a SNVT.

Object Location SCPTlocation

This configuration property (SCPTlocation) is used to define the location of the Modbus register from which data will be obtained. The 31-character location string includes both a textual label that is arbitrary, and the identification of the Modbus register at that location.

The location string is expected to be in the format:

UTNNNNNxLabel

where:

U	is a unit number 1..247
T	is a register type (see below)
NNNNN	is the register reference
x	is any non-numeric character
Label	is any arbitrary text label

The “xLabel” can be omitted, and may be EOL. A completely empty string will indicate an unused object that is not disabled, but will not be active.

Register types T recognized by the Babel Buster gateway are as follows, and the letter shown should be substituted for T in the above format:

S - signed 16-bit	U - unsigned 16-bit
D - signed 32-bit	E - unsigned 32-bit
F - floating point	B - bit/boolean

Register references are recognized in the following ranges for the indicated Modbus types:

0x	Coil	00001-09999 (master only)
1x	Discrete Input	10001-19999 (master only)
3x	Input Register	30001-39999 (master only)
4x	Holding Register	40001-49999

Note: Format for nviDataTo SCPTlocation is identical except that only 0x and 4x references are recognized.

If the location string contains an invalid, unrecognized, or out of range register reference when the functional block is enabled via nviRequest, the nvoStatus (Node Object) will reflect “unable_to_measure”.

The rate at which the Modbus register is polled for data is specified by SCPTupdateRate (see definition that follows).

Maximum Send Time SCPTmaxSndT

This “Send Heartbeat” configuration property sets the maximum period of time that can expire before the functional block will automatically update the network variable nvoFromModbus. This is independent of the Modbus update rate.

When operating as a slave (server), this property also specifies the maximum amount of time that may expire without an update from the master (client) before the object will assume the default value and flag an error in the object status.

Minimum Send Time SCPTminSndT

This “output throttling” configuration property sets the minimum period of time that can expire before the functional block will automatically update the network variable nvoFromModbus. This is independent of the Modbus update rate.

Network Variable Type SCPTnvType

This configuration property specifies the SNVT type implemented by nvoFromModbus. It may be any scalar NV type. It is not necessary to match the 4-byte size of the default SNVT_count_f. The LonMaker browser will automatically set all fields of this SCPT when the NV type is changed by right clicking on the NV to be changed, and selecting “Change Type...” from the pop-up menu. Refer to the section titled Changing Network Variable Type.

Override Behavior `SCPTovrBehave`

This configuration property specifies the override behavior for the functional block. The object is put into override via `nviRequest` issued to the Node Object with this object specified as the object ID. The behavior is defined by `SCPTovrBehave` per LonMark standard. If changed while already in override, `RQ_OVERRIDE` needs to be re-issued.

Override Value `SCPTovrValue`

This configuration property specifies the override value that will be assumed by the functional block if override behavior is specified as `OV_SPECIFIED`. This property will not be used otherwise. This property inherits the NV type of `nvoFromModbus`.

Send On Delta `SCPTsndDelta`

This configuration property specifies the minimum change required to force transmission of the output value (update of `nvoFromModbus`). The property type is inherited from the type assigned to `nvoFromModbus`.

Update Rate `SCPTupdateRate`

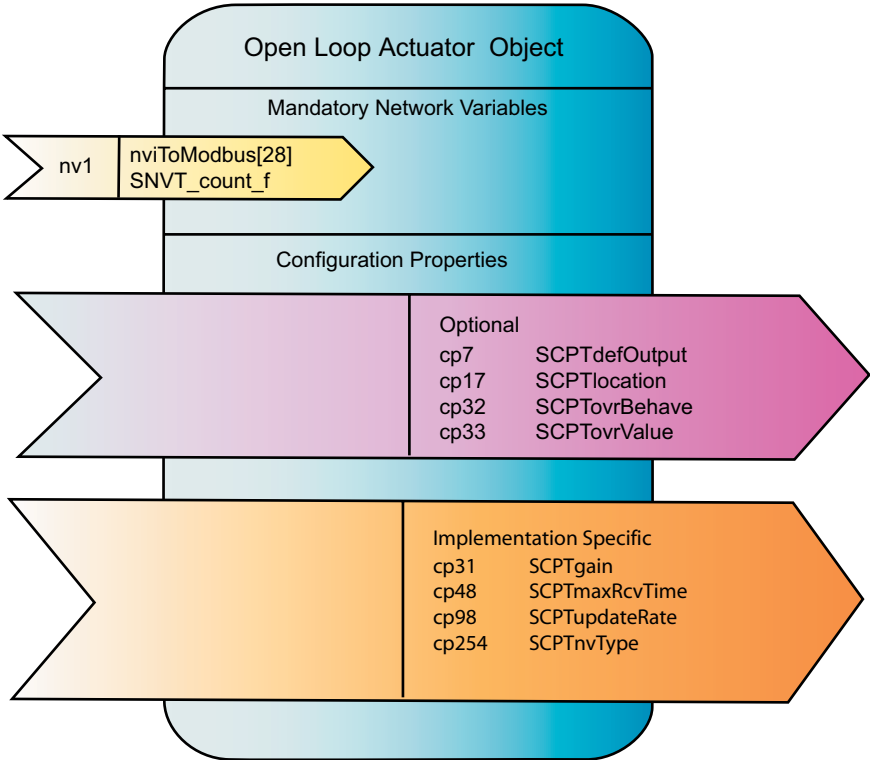
This configuration property specifies the polling interval, in tenths of seconds, defining the rate at which the Modbus device is polled to retrieve data for the respective functional block. This property defines how often the Modbus register is polled, not how often the respective LonWorks network variable is updated. The Modbus register data is qualified by `SCPTsndDelta`, `SCPTminSendT`, and `SCPTmaxSendT` before the NV is updated.

Traffic on the LonWorks network is throttled by `SCPTminSendT` and further minimized with `SCPTsndDelta`. Traffic on the Modbus network is set only by `SCPTupdateRate`. Since Modbus is a master/slave polling scheme, Modbus network traffic can become excessive with a low `SCPTupdateRate` value. The `SCPTupdateRate` property has no direct effect on LonWorks network traffic.

8. Open Loop Actuator Object

Data To Modbus `nviToModbus` [28]

This network variable presents data provided by the LonWorks network to a Mod-



bus device. The default variable type is floating point `SNVT_count_f`, but may be changed by a network management tool.

To change the SNVT type from the LonMaker browser, right click on the variable to be changed, and select “Change Type...”. A dialog box will appear allowing the selection of a new type. It is not necessary to select a “same size” SNVT type. Any scalar type may be selected. In addition, `SNVT_state`, `SNVT_lev_disc`, and `SNVT_switch` are supported. The LonMaker browser will automatically change the corresponding `SCPTnvType` configuration property. It is not necessary to explicitly change this property.

Babel Buster software will recognize the SNVT type change when the functional block is enabled. It should be disabled while changing the type (and the LonMaker browser will do this automatically). Data conversions should be tested by the system integrator once configured. Overflow and underflow conditions are not necessarily bounded depending upon the type of conversion being made.

Default Output Level `SCPTdefOutput`

This configuration property sets the default Modbus register value that is assumed when the object is disabled, or when overridden to the default value. This default value is also assumed if the maximum receive time is configured to nonzero, and the maximum receive time has been exceeded. This property inherits the same type as the input network variable.

Scale Factor SCPTgain

This configuration property sets the SCPTgain scale factor for scaling data prior to sending it to the Modbus device. Manufacturers of Modbus devices often define register contents as X10, X100, etc. To translate these raw values from scaled values, SCPTgain is applied. If the Modbus register is not scaled, this property should be set to { 1, 1 } shown as “1/1” in the LonMaker browser. When preparing data for Modbus, the scaled data is multiplied by the first number, and divided by the second.

To illustrate, suppose the value 72.5 is to be sent to a Modbus register defined by its manufacturer as X10, meaning the data in integer form should be sent as 725. The scale factor SCPTgain should then be { 10, 1 } or “10/1”.

Object Location SCPTlocation

This configuration property (SCPTlocation) is used to define the location of the Modbus register to which data will be sent. The 31-character location string includes both a textual label that is arbitrary, and the identification of the Modbus register at that location.

The location string is expected to be in the format:

UTNNNNNxLabel

where:

U	is a unit number 1..247
T	is a register type (see below)
NNNNN	is the register reference
x	is any non-numeric character
Label	is any arbitrary text label

The “xLabel” can be omitted, and may be EOL. A completely empty string will indicate an unused object that is not disabled, but will not be active.

Register types T recognized by the Babel Buster gateway are as follows, and the letter shown should be substituted for T in the above format:

S - signed 16-bit	U - unsigned 16-bit
D - signed 32-bit	E - unsigned 32-bit

F - floating point

B - bit/boolean

Register references are recognized in the following ranges for the indicated Modbus types:

0x	Coil	00001-09999 (master only)
4x	Holding Register	40001-49999

Note: Format for nvoDataFrom SCPTlocation is identical except that 1x and 3x references are also recognized.

If the location string contains an invalid, unrecognized, or out of range register reference when the functional block is enabled via nviRequest, the nvoStatus (Node Object) will reflect “unable_to_measure”.

The Modbus register specified for this functional block is updated at the same rate at which the network variable is updated, and optionally at the resend rate set by SCPTupdateRate.

Maximum Receive Time SCPTmaxRcvTime

This “receive heartbeat” SCPTmaxRcvTime configuration property sets the maximum time that can elapse after an update to the network input variable before the Modbus register will assume the default value. A value of zero (0) disables the receive failure detect mechanism.

Network Variable Type SCPTnvType

This configuration property specifies the SNVT type implemented by nviToModbus. It may be any scalar NV type. It is not necessary to match the 4-byte size of the default SNVT_count_f. The LonMaker browser will automatically set all fields of this SCPT when the NV type is changed by right clicking on the NV to be changed, and selecting “Change Type...” from the pop-up menu. Refer to the section titled Changing Network Variable Type.

Override Behavior SCPTovrBehave

This configuration property specifies the override behavior for the functional block. The object is put into override via nviRequest issued to the Node Object with this object specified as the object ID. The behavior is defined by SCPTovrBehave per LonMark standard.

Override Value SCPTovrValue

This configuration property specifies the override value that will be assumed by the functional block if override behavior is specified as OV_SPECIFIED. This property will not be used otherwise. This property inherits the NV type of nviToModbus.

Resend Rate SCPTupdateRate

This configuration property, if set to a nonzero time in tenths of seconds, causes the functional block to update the Modbus register in the remote server (slave) periodically in addition to upon input network variable updates. This is only applicable when the object (functional block) is configured to operate as a master (client).

9. Getting Started

Verify your wiring before applying power. **DO NOT connect AC common to ground.** Doing so will result in damage and failure of the device. Refer to section 13 for wiring examples.

When things don't work, check the object status by writing RQ_UPDATE_STATUS to nviRequest and observing nvoStatus for the object in question, or entire node (object #0). The object status bit summary at the end of section 11 can provide many clues about failures.

Each mapping of a Modbus register to a LonMark Standard Network Variable (SNVT) is defined as a LonMark functional profile. Each functional block, or object, will be disabled by default in the initial state of a Babel Buster gateway. The following steps need to be taken at a minimum to establish communications (in addition to obvious things like connecting network and power):

(1) Define the Modbus register mapping by entering the register definition in the SCPTlocation property of the associated network variable. Do this for each relationship desired using LonMaker's browser, or any other network management tool that provides access to the configuration property file using the direct read/write file access method.



FromModbus[9]		SCPTlocation	N	1U40001
---------------	--	--------------	---	---------

(2) Select master or slave mode in nciGatewayOpts. Enter the slave unit ID if configured to be a slave. Set baud rate for either master or slave. Set timeout if

configured as a master.

(3) Enable the object(s). This is done by issuing an enable request RQ_ENABLE to the functional block via the nviRequest found in the node object. The object status may be observed in nvoStatus also found in the node object. If the Modbus register reference is not recognized as valid, the object will remain disabled, and the “unable_to_measure” bit will also be set in the object status.

FromModbus[9]		SCPTlocation	N	1U40001
---------------	--	--------------	---	---------

10. Functional Block (Object) Status

Each of Babel Buster’s sensor and actuator objects maintains its own set of status bits. These are accessed via the node object’s nviRequest and nvoStatus as defined by LonMark standards. Object numbers (Object ID) used to access individual object status are as follows:

0	Node Object	
1-30	Open Loop Sensor Objects	nvoFromModbus
31-58	Open Loop Actuator Objects	nviToModbus

To find the Object ID associated with any particular network variable, you can use the LonMaker browser to look it up. Right click on the desired variable, and select “Properties...” from the popup menu. Select the “LonMark” tab on the property sheet that appears next. The number shown near the bottom of the page indicated as “Object Number” is the number to make a note of for use in making object requests.

LonMaker Browser - Untitled

File Edit Browse Help

31.RQ_ENABLE

Subsystem	Device	Functional Block	Network Variable	Config Prop	Mon	Value
Subsystem 1	ModbusTCP	FromModbus[9]		SCPTgain	N	1/1
Subsystem 1	ModbusTCP	FromModbus[9]		SCPTlocation	N	1U40001
Subsystem 1	ModbusTCP	FromModbus[9]		SCPTmasterSlave	N	1
Subsystem 1	ModbusTCP	FromModbus[9]		SCPTupdateRate	N	5.0
Subsystem 1	ModbusTCP	FromModbus[9]	nvoFromModbus_10			
Subsystem 1	ModbusTCP	FromModbus[9]	nvoFromModbus_10		S	
Subsystem 1	ModbusTCP	FromModbus[9]	nvoFromModbus_10		S	
Subsystem 1	ModbusTCP	FromModbus[9]	nvoFromModbus_10		S	
Subsystem 1	ModbusTCP	FromModbus[9]	nvoFromModbus_10		S	
Subsystem 1	ModbusTCP	NodeObject			S	
Subsystem 1	ModbusTCP	NodeObject			S	
Subsystem 1	ModbusTCP	NodeObject			S	
Subsystem 1	ModbusTCP	NodeObject			S	
Subsystem 1	ModbusTCP	NodeObject	nciGatewayOpts			
Subsystem 1	ModbusTCP	NodeObject	nviRequest			
Subsystem 1	ModbusTCP	NodeObject	nvoFileDirectory			
Subsystem 1	ModbusTCP	NodeObject	nvoStatus			
Subsystem 1	ModbusTCP	ToModbus[0]			S	
Subsystem 1	ModbusTCP	ToModbus[0]		SCPTgain	N	1/1
Subsystem 1	ModbusTCP	ToModbus[0]		SCPTlocation	N	1U40001
Subsystem 1	ModbusTCP	ToModbus[0]		SCPTmasterSlave	N	1
Subsystem 1	ModbusTCP	ToModbus[0]		SCPTmaxRcvTime	N	0.0
Subsystem 1	ModbusTCP	ToModbus[0]		SCPTovrBehave	N	OV_RETAIN
Subsystem 1	ModbusTCP	ToModbus[0]		SCPTupdateRate	N	5.0
Subsystem 1	ModbusTCP	ToModbus[0]	nviToModbus_1		N	5441
Subsystem 1	ModbusTCP	ToModbus[0]	nviToModbus_1	SCPTnvType	N	PD 0:0:0:0:0:0:0:0, Scope 0, Index 51, NVT_CAT_
Subsystem 1	ModbusTCP	ToModbus[0]	nviToModbus_1	SCPTovrValue	N	0

Ready

266

Network Variable Properties

Description | NV Attributes | Connection Attributes | LonMark | Monitor Options

Function Profile Name:
Value output

Function Profile Programmatic Name:
nvoValue

Function Profile Description:
Transmits the value from the sensor after conversion to correct units

Member Index: 0

Member Number: 1

Object Number: 10

Manufacturer Assigned

OK Cancel Apply Help

Once you have determined which object number to issue a request to, enter the object number followed by the request in the edit box at the top of the browser window, and click on the button to the left (red arrow pointing down). The screen shot shown here illustrates issuing the enable request RQ_ENABLE to object #31 which is nviToModbus[0], or actuator object #1.

Right click on nvoStatus, and select “Monitor” from the popup menu to enable monitoring of status so that it is updated following each request that is made. The “Mon” column will indicate “Y” when monitoring is enabled, “N” if not.

Requests that are honored by Babel Buster include the following:

```
RQ_NORMAL
RQ_DISABLED
RQ_UPDATE_STATUS
RQ_REPORT_MASK
RQ_OVERRIDE
RQ_ENABLE
RQ_RMV_OVERRIDE
```

The “Normal” request is the same as issuing both “Enable” and “Remove Override” (RQ_RMV_OVERRIDE) requests at the same time. All objects default to disabled until configured and then enabled by RQ_ENABLE or RQ_NORMAL.

The status is indicated as:

```
31 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

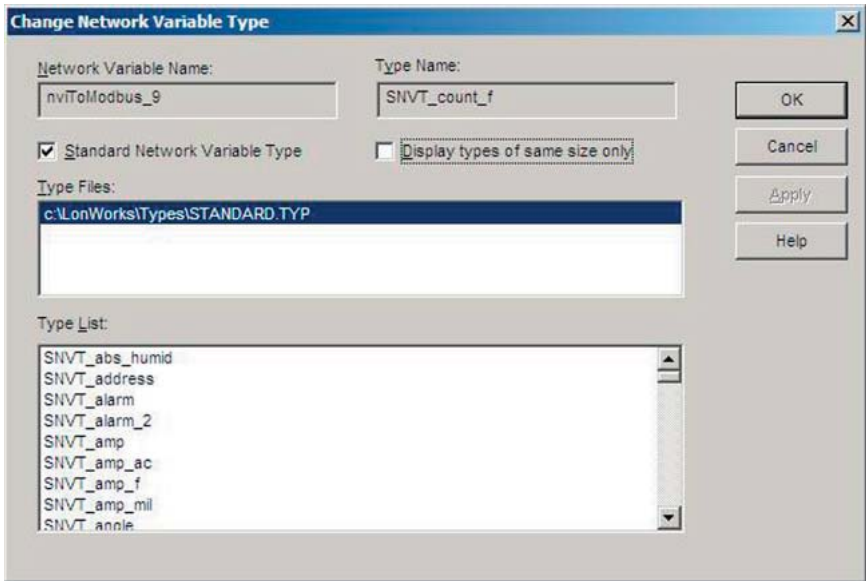
Bits which are utilized by Babel Buster are indicated as follows:

```
31 a,b,c,d,0,0,e,0,0,f,g,0,0,0,0,h,i,0,0,0,0
```

where ‘0’ will always be returned as zero, and bit positions labeled will be set to 1 (one) when the following conditions are true:

- a – invalid_id: Object ID was invalid, e.g., object #70 does not exist
- b – invalid_request: Unsupported SNVT type change requested
- c – disabled: Object is disabled
- d – open_circuit: Timeout, no response received from slave (5 tries)
- e – feedback_failure: Timeout, no poll from master
- f – unable_to_measure: Modbus register reference invalid
- g – comm_failure: Modbus exception code
- h – in_override: Object is in override state
- i – report_mask: Set to indicate this value is a report mask

The open_circuit fault is only flagged after 5 tries and 5 consecutive timeouts waiting for a response from the slave. Intermittent timeouts will be disregarded.



The check box labeled “Standard Network Variable Type” must be checked. The check box labeled “Display types of same size only” does not need to be checked. Select the desired network variable type, and click Apply or OK.

Once the network variable type has been changed, check the object’s status. If the selected type is not supported by Babel Buster, the `invalid_request` bit will be set, and the object will be disabled.

SNVT types that are supported by Babel Buster include all scalar types such as signed or unsigned integer (8-bit, 16-bit, 32-bit), or floating point. In addition, `SNVT_switch`, `SNVT_lev_disc`, and `SNVT_state` are provided with special case translation. Enumerations and structures other than state and switch are not supported.

`SNVT_switch` is translated as follows: When converting to `SNVT_switch`, the value is bounded to 0..100%, and state set to 1 if value is nonzero. When converting from `SNVT_switch`, the intermediate value is set to zero if the state is zero, otherwise the intermediate value is set to the `SNVT_switch` value field.

`SNVT_lev_disc` is translated as follows: When converting from `SNVT_lev_disc`, the value is set to the following: `ST_OFF=0`, `ST_LOW=25`, `ST_MED=50`, `ST_HIGH=75`, `ST_ON=100`. When converting to `SNVT_lev_disc`, the value is bounded at 100, then divided by 25 and truncated to an integer in the range of 0 to 4 (`ST_OFF` to `ST_ON`).

SNVT_state is translated as follows: When converting to SNVT_state, the value is converted to unsigned 16-bit binary, and placed in SNVT_state. Note that since SNVT_state is defined as a structure, but is converted as a 16-bit binary number, the structure's bit0 is the most significant bit (MSB) of the binary number and the structure's bit15 is the least significant bit (LSB). The process is simply reversed when converting from SNVT_state.

Scalar types are converted according to the scale factors set in theSCPTnvType configuration property. These factors will be set automatically if the change is made via the LonMaker browser. The scale factors are assumed to be 1,1,0 for floating point types, but vary for other integer types that are more application specific. In conjunction with applying the scale factors to non-floating point types, the type category is used to make data size conversions on scalar types.

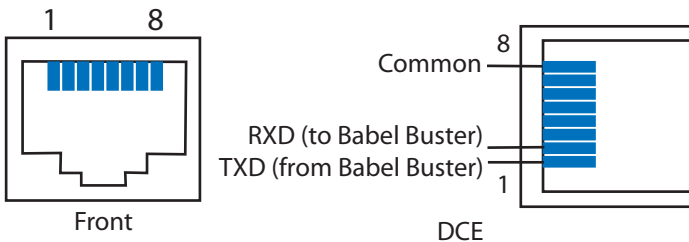
12. Power Supply & Comm Port Wiring

Babel Buster includes a TP/FT-10 (FTT-10) free topology 78k baud transceiver. Network wiring should be per LonWorks guidelines for the TP/FT-10 channel.

Babel Buster will operate from 12-24VAC or 10-30VDC. Power supply wiring differs depending on whether AC or DC power is provided. The AC is rectified by a bridge type rectifier. Therefore **AC power inputs must not be grounded**. The following diagrams illustrate power wiring options.



The RS-232 port is connected via an RJ-45 connector that follows Modbus specification pin-out as illustrated below. Standard Ethernet patch cords can be used for interconnect cables.



The following table indicates the connector pin designations specified by Modbus-IDA, the official Open Modbus organization.

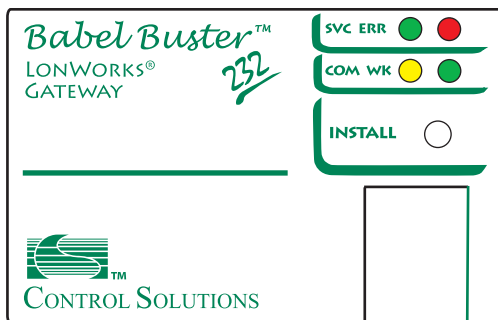
DCE <u>Underlined</u> pins can be output			Circuit			DTE <u>Underlined</u> pins can be output		
Pin on RJ45	Pin on D9-shell	Level of requirement	Name	Description	RS232 Source	Level of requirement	Pin on RJ45	Pin on D9-shell
<u>1</u>	<u>2</u>	required	TXD	Transmitted Data	DTE	required	<u>2</u>	<u>3</u>
2	3	required	RXD	Received Data	DCE	required	1	2
3	7	optional	CTS	Clear to Send	DCE	optional	6	8
<u>6</u>	<u>8</u>	optional	RTS	Request to Send	DTE	optional	<u>3</u>	<u>7</u>
8	5	required	Common	Signal Common	--	required	8	5

13. Front Panel Operation & LED Indications

The Babel Buster 232 front panel is shown below. The “install” button is the standard Neuron Chip service button used to commission the node. SVC is the green LED on the top left, and behaves as specified for the standard Neuron service pin LED.

The green LED on the lower right, labeled WK, is the “wink” LED as commonly referred to in LonWorks terminology. It will normally be on, blinking off briefly once every 5 seconds. This is the “I’m alive” indication. Upon receiving a wink command from the LonWorks network, the wink LED will blink rapidly for a couple of seconds, then return to its normal behavior.

The two remaining LEDs function as required by Modbus protocol specification for serial line implementation. The red LED on the upper right, marked ERR, is the error LED which will flash briefly once for each error received. The yellow LED on the lower left, marked COM, is the communications LED which will flash once briefly for each packet transaction involving this node. (Other traffic on the network will not affect the COM LED.)



Power up behavior of the SVC LED is as follows: Immediately upon applying power, the SVC LED will flash once very briefly. If it flashes on, then off, then remains on, it has become set the applicationless state. If it flashes continuously, it has become unconfigured. Both of these can result from a flawed attempt at network installation.

Power up behavior of the remaining LEDs is as follows: Several seconds after applying power, the red and yellow LEDs will flash once, and the green WK LED will flash several times exhibiting the wink command behavior, then go to its normal mode behavior indicating “I’m alive”.

14. Connector Terminals

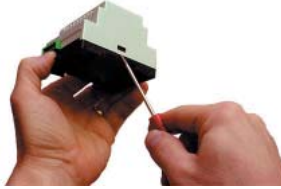
The screw terminals for wiring Babel Buster are labeled on the enclosure. An illustration of the labels and notation of their signal grouping is shown below.

The terminal blocks are screw clamps accepting 22-14AWG wire. The terminal block may be unplugged from the base unit.



15. Opening the Case

Open the case by gently sliding a screw driver under the edge of the cover at the locking tab. Lift up over the tabs on both sides. Lift both sides evenly and lift cover straight away from base.

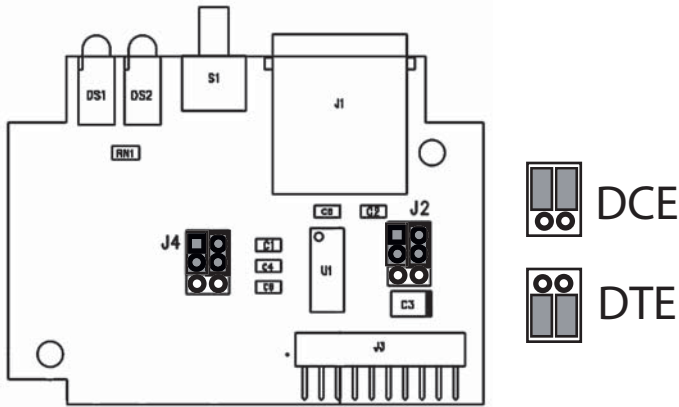


16. Jumper Settings

Babel Buster 232 is shipped configured as DCE. The DCE/DTE selection jumpers are located on the small daughter card in the area shown below. There are two sets of two 3-position jumpers.



Jumper positions are illustrated below. DCE or DTE are selected by moving the jumpers as a set. All jumpers must be set in the same position (DCE or DTE).



J2: Transmit/Receive Data
 J4: Hardware Handshake
 (handshake not supported)

17. Mounting on DIN Rail

Babel Buster is designed to be mounted on DIN rail. To attach to the rail, hook Babel Buster on the top edge, and push onto the bottom edge. The enclosure should snap into place.

To remove, insert a small screw driver in the release tab at the bottom center of the enclosure. Gently pry downward no more than 1/8" (3 mm). Pull the bottom of Babel Buster away from the rail and lift upward.



18. Trouble Shooting

No data transfer appears to occur.

(a) Is the object enabled? See section on Functional Block Status.

(b) Is the Modbus register defined in the SCPTlocation for the object in question? Was the register reference accepted by the object? (If “unable to measure” is set in the object status, it was not accepted.) Refer to section on Modbus Registers and discussion of SCPTlocation.

(c) Is slave ID correct (applies to RTU as slave)? Is the slave timeout set too short (applies to RTU as master)?

(d) Is the master/slave state set correctly? For Babel Buster 232, this is set one time for the entire device in nciGatewayOpts in the node object.

(e) What do LEDs indicate? See section on LED Indications.

(f) Does object status indicate other errors? See section on Functional Block Status.

Data appears, data changes, but does not appear correct.

(a) Did you change the network variable type? If so, did the object status indicate “invalid request”? If this bit is set in the node object nvoStatus for the object ID in question, the type change was not accepted. See section on Changing Network Variable Types.

(b) Is the Modbus register indicated in SCPTlocation for the object correct? Refer to section on Modbus Registers and discussion of SCPTlocation.

(c) Have you defined a double Modbus register when the Modbus device is expecting a single register or vice versa? Are double registers swapped? Refer to section on Modbus Registers.

Data exchange takes place correctly, but response is very slow.

(a) This is most likely for a sensor nvoFromModbus. The minimum send time is designed to throttle network traffic, and defaults to 15 seconds. This means the nvo (network variable output) will update at a maximum of once every 15 seconds. Lower this number if needed.

(b) The actuator nviToModbus variables will transfer data immediately upon network variable update. Verify that network variable updates are in fact occurring.

I’m confused about Master/Slave, Client/Server, To/From.

Master is synonymous with client. Slave is synonymous with server. A “From” object refers to a mapping whereby LonWorks is receiving data from Modbus. A “To” object refers to a mapping whereby LonWorks is sending data to Modbus.

Babel Buster may be either master or slave, client or server, and in fact can be client and server simultaneously. Deciding who is which is a matter of establishing who will initiate the request for Modbus data.

If “master” is selected, the Babel Buster client will read registers (0x, 1x, 3x, 4x) from the remote server using the open loop sensor object “FromModbus” and put the data received in the object’s output network variable.

If “master” is selected, the Babel Buster client will take data from the input network variable of the open loop actuator object “ToModbus” and write to registers (0x, 4x) of the remote server.

If “slave” is selected, the Babel Buster server will passively wait for the remote client to write registers (4x only) allocated to the open loop sensor object “From-Modbus” and put the received data in the object’s output network variable when received.

If “slave” is selected, the Babel Buster server will passively wait for the remote client to read registers (4x only) where it will find data from the input network variable of the open loop actuator object “ToModbus” which is merely buffered waiting to be read by the remote Modbus client.

How do I know the Modbus protocol is compatible between Babel Buster and my Modbus/RTU device?

The Modbus protocol used in Babel Buster 232 is written according to the protocol standards found on www.modbus.org and was tested using third party protocol tools. The Babel Buster master uses function codes 1, 2, 3, 4, 15, and 16. The Babel Buster slave responds to function codes 3 and 16. The master will use only codes 3 and 16 if only 4x registers are configured. Function codes 3 and 16 are the minimum set of functions that will be supported by any Modbus standard device.

19. Trouble Shooting Tools

There are times when simple observations as outlined in the trouble shooting section are not enough to fully diagnose problems. Diagnostics accessible from the devices themselves are most often not detailed enough to locate the source of problems. To complicate matters, one needs to know if the master failed to send a request or the slave failed to send the reply before much progress can be made.

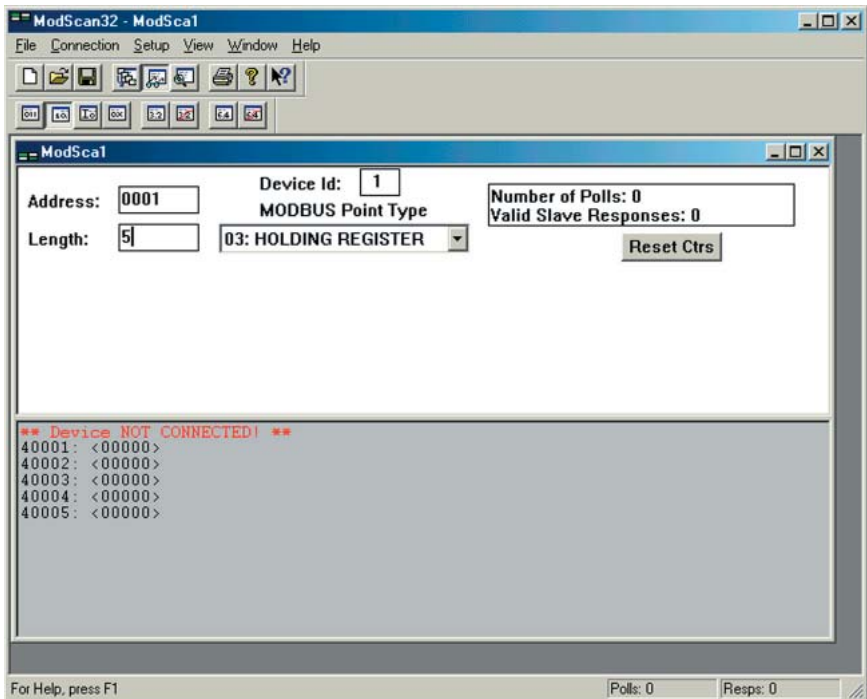
To assist in analyzing exactly what is really going on, we at Control Solutions use an inexpensive third party tool to help out, namely a Modbus scanner and simulator. We also have access to a LonWorks protocol analyzer, but don’t really need to

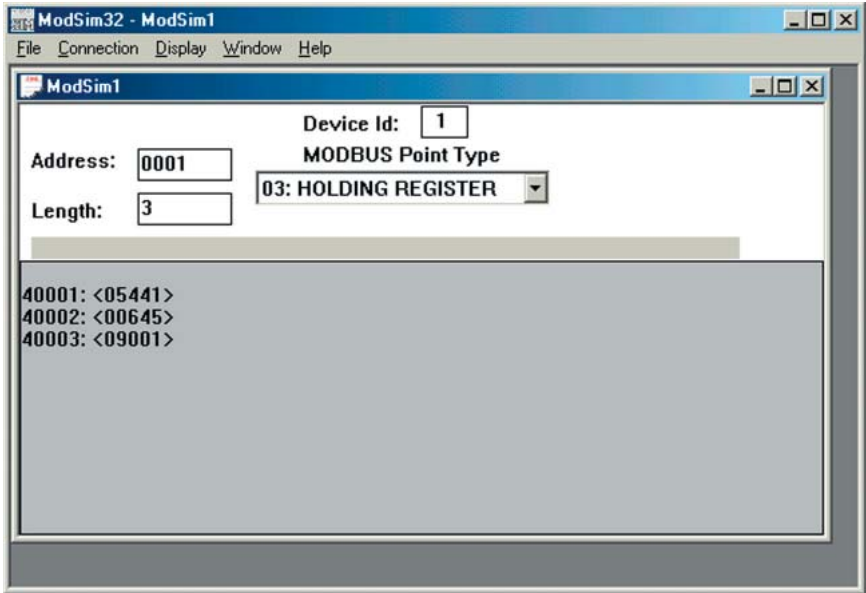
use it all that often since the only network traffic on the LonWorks side is strictly network variable updates. We rely on the LonMaker browser to provide a higher level view of network variable updates.

Communication with Modbus devices, both Ethernet and serial (RTU RS-232), can be independently tested using scanner and simulator software such as ModScan32 and ModSim32 from WinTECH Software at www.win-tech.com. This is commercial software requiring the purchase of a license, but is reasonably priced and well worth the investment for any system integrator doing much Modbus work. Other equivalent tools are available, and are sometimes included in front end software packages used by system integrators.

A Modbus slave (server) may be tested using ModScan32. This package periodically polls the Modbus device, displays data from the replies, and indicates errors if any.

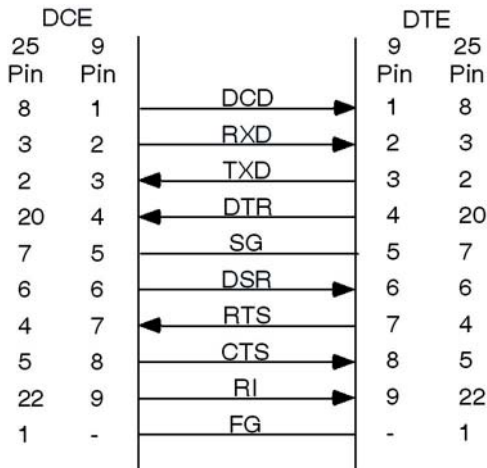
A Modbus master (client) may be tested using ModSim32. This package simulates a slave (server). It is polled by the master, and whatever data you enter on the PC will be returned to the master.





20. A Concise Guide to RS-232 (EIA-232)

The signal definitions and pin numbers on “D-sub” connectors are shown below.



DCE stands for Data Communications Equipment. DTE stands for Data Terminal Equipment. These definitions go back to the days when the teletype machine was DTE, and a modem was DCE.

The MALE connector is located on or associated with DTE.
The FEMALE connector is located on or associated with DCE.

IBM PC 9-pin COM1 or COM2 connectors follow the DTE pinout. Older IBM PC 25-pin COM1 or COM2 connectors follow the DCE pinout.

MARK = OFF = Binary 1 data = -12V
SPACE = ON = Binary 0 data = +12V

Interpretation of Signals:

RTS is turned on by DTE indicating that DTE wishes to transmit to DCE. DCE responds by turning on CTS at which time DTE may transmit data to DCE via TXD. When DTE turns off RTS off, DCE responds by turning CTS off.

DCE may also turn off CTS to cause DTE to suspend transmission via TXD while DCE processes data already received.

DSR is turned on by DCE to indicate that DCE is connected, and in the case of a modem that dialing has been completed and a connection made.

RI is the ring indicator that turns on when an incoming call is detected (applies to modems on phone lines).

DCD is the carrier detect which turns on when a good signal is being received from the modem at the other end of the line.

DTR is turned on by DTE indicating DTE is ready to transmit or receive data. DTR is turned off by DTE to indicate that DCE should stop sending data to DTE via RXD.

Summary:

CTS enables data on TXD from DTE to DCE.
DTR enables data on RXD from DCE to DTE.

When used between computers rather than with modems, DSR is usually just turned on to indicate that a cable is connected, and therefore the equipment generating DSR simply connects it to +12V.

RI and DCD are commonly not used between computers. Most often, only CTS and DTR are used for handshaking between computers if there is any handshaking at all.

21. Specifications

- LonWorks TP/FT-10 (FTT-10A) to Modbus RS-232 Gateway
- Translate 58 LonWorks network variables to Modbus registers
- LonMark Ver. 3.3 Sensor & Actuator Functional Profiles
- Supports Modbus “coils”, input registers, and holding registers
- Single or double Modbus registers, signed, unsigned, IEEE 754
- All scalar LonMark SNVT types supported
- SNVT_switch & SNVT_state also supported
- SNVT types dynamically assignable via LonMaker
- Modbus register mapping dynamically mappable via LonMaker
- Modbus registers may be scaled (x10, x100, x0.1, x0.01, etc.)
- Min/max send time, send on delta configurable per point
- Modbus (master) polling interval configurable per point
- Object override supported on point by point basis
- Bidirectional communication between LonWorks and Modbus
- LonMark certified
- 3150L Neuron® Chip, 48K Flash memory, 10K RAM
- FTT-10A transceiver
- Powered by 10-30VDC or 12-24VAC 50/60 Hz
- Power consumption: 0.1A @ 24VDC
- DIN rail mounting, 100mm H x 70mm W x 60mm D
- Modbus standard RJ-45 connector, RS-232 pin-out
- Operating temperature: -40°C to +85°C; Humidity 5% to 90%

Control Solutions' Babel Buster 232 has been tested to CFR 47: 2003, §15.107 and §15.109, Class B. Babel Buster 232 has been tested to comply with FCC standards for home and office use. This device complies with Part 15 of the FCC rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

Control Solutions' Babel Buster 232 has been tested to and complies with EN 55022: 1998, Class B; EN 61000-3-2: 2000; EN 61000-3-3: 1995 for emissions; and to EN 55024: 1998 for immunity.

WARRANTY

The Babel Buster series modules are warranted against defects in materials and workmanship for a period of 1 (one) year from date of shipment from factory. 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.

*Quality Lon Works®
Product Development
Since 1995.*



CONTROL SOLUTIONS, INC.
2179 FOURTH STREET, SUITE 2-G • PO BOX 10789
WHITE BEAR LAKE, MN 55110-0789

www.csimn.com