Communication

for AGCx, AGDx, AGMx Controllers

User Manual

www.agito-akribis.com

Member of Akribis Systems group

agito

Revision History

Version	Description	Date
3.1	Removal of CCC section	4 Sep 2023
3.0	Revision of manual content and format	11 July 2023
2.0	Preliminary revision of manual content and format	3 May 2021
1.0	Initial release	9 June 2021

Contact Information

Manufacturer	Agito Akribis Systems Ltd., Member of Akribis Systems Group
Address	6 Yad-Harutsim St., P.O.Box 7172, Kfar-Saba 4464103
Telephone	agito.info@akribis-sys.com
Website	www.agito-akribis.com

Copyright Notice

©2023 Agito Akribis Systems Ltd.

All rights reserved. This work may not be edited in any form or by any means without written permission of Agito Akribis Systems Ltd.

Products Rights

AGDx, AGCx, AGMx, AGAx, AGIOx, and AGLx are products designed by Agito Akribis Systems Ltd. in Israel. Sales of the products are licensed to Akribis Systems Pte Ltd. under intercompany license agreement.

Agito Akribis Systems Ltd. has full rights to distribute above products worldwide.

Disclaimer

This product documentation was accurate and reliable at the time of its release.

Agito Akribis Systems Ltd. reserves the right to change the specifications of the product described in this manual without notice at any time.

Trademarks

Agito PCSuite is a trademark of Agito Akribis Systems Ltd..

Warranty

This product is warranted to be free of defects in material and workmanship and conforms to the specifications listed in this manual, for a period of 12 months from the shipment date from factory.

Table of Contents

1 Introduction 5
- 1.1 About this Manual 5
- 1.2 General Overview 5
- 1.3 Keywords 6
- 1.4 Command Messages 7
- 1.5 Reply Messages 8

2 Communication Channels 9

3 Hardware Connections 11
- 3.1 Connection Interfaces 11
- 3.2 RS232 Wiring 13
  - 3.2.1 RJ45 Serial RS232 Wiring 13
  - 3.2.2 Micro USB Serial RS232 Wiring 13
- 3.3 RS485 Wiring 13
- 3.4 CAN Bus Wiring 14
- 3.5 Ethernet Wiring 14

4 Communication Settings 15
- 4.1 RS232 Communication Settings 15
- 4.2 RS485 Communication Settings 15
- 4.3 CAN Bus Communication Settings 16
  - CAN Addressing Scheme 17
- 4.4 Ethernet Communication Settings 18

5 Message Structure 19
- 5.1 ASCII Encoding 19
  - 5.1.1 ASCII Syntax (Base) 19
    - ASCII Command (Base) 19
    - ASCII Reply (Base) 20
  - 5.1.2 RS232 ASCII Syntax 21
    - RS232 ASCII Command Messages 21
    - RS232 ASCII Reply Messages 21
  - 5.1.3 RS485 ASCII Syntax 21
    - RS485 ASCII Command Messages 21
    - RS485 ASCII Reply Messages 22
  - 5.1.4 Ethernet ASCII Syntax 22
    - Ethernet ASCII Message Types 22
    - Ethernet ASCII Command Messages (Standard) 22
    - Ethernet ASCII Bulk Command Messages (Intermittent) 23
    - Ethernet ASCII Bulk Reply Messages (Intermittent) 23
    - Ethernet ASCII Bulk Command Messages (Lengthy) 23
    - Ethernet ASCII Bulk Reply Messages (Lengthy) 23
- 5.2 Binary Encoding 24
  - 5.2.1 Binary Syntax (Base) 24
    - Binary Command (Base) 24
    - Binary Reply (Base) 25
  - 5.2.2 CAN Bus Binary Syntax 26
    - CAN Bus Command Messages 26
    - CAN Bus Reply Messages 26
  - 5.2.3 Ethernet Binary Syntax 27
    - Ethernet Binary Message Types 27
    - Ethernet Binary Command Messages (Type 0 – Standard) 27
    - Ethernet Binary Reply Messages (Type 0 – Standard) 27
    - Ethernet Binary Command Messages (Type 2 – Bulk) 28

Ethernet Binary Reply Messages (Type 2 – Bulk) 28
6 Examples 29
6.1 RS232 Communication Example 29
6.1.1 Connection and configuration of RS232 communication channel 29
6.1.2 Sending ASCII commands over RS232 communication channel 30
6.2 RS485 Communication Example 30
6.2.1 Connection and configuration of RS485 communication channel 30
6.2.2 Sending ASCII commands over RS485 communication channel 31
6.3 CAN Communication Example 32
6.3.1 Connection and configuration of CAN Bus communication channel 32
6.3.2 Sending binary commands over CAN Bus communication channel 33
6.4 Ethernet Communication Example 34
6.4.1 Connection and configuration of Ethernet communication channel 34
6.4.2 Sending ASCII commands over Ethernet communication channel 36
6.4.3 Sending binary commands over Ethernet communication channel 37

1 Introduction

1.1 About this Manual

The information in this manual applies to all Agito servo controllers, both standalone controllers and Central-i master controllers. The Central-i fieldbus protocol (between master and slaves) is handled internally and is transparent to the user, who must communicate only with the master.

Chapter 1 provides a general introduction about communication with an Agito controller. In particular, it describes fundamental components such as keywords, command messages, and replies.

Chapter 2 summarizes the various communication channels, and the pros and cons of each.

Chapter 3 describes the two data encoding formats, ASCII and binary. It also explains the syntax for each of the communication channels.

Chapter 4 details the hardware setup for each of the communication channels, communication parameters, and provides examples of how to establish such a communication line.

Chapter 5 presesnts the data structure of the messages and how they are encoded for the various communication channels.

Chapter 6 provides examples showing how you may implement such communication channels.

1.2 General Overview

Messages are encoded in either ASCII or binary data formats and sent via RS232, RS485, CAN Bus, and Ethernet TCP/IP communication channels. Regardless of encoding, protocol and hardware layer, the information contained in the messages remain the same.

Agito servo controllers adopt the server-client architecture in which the servo controller is the server. To interact with the controller, the user sends a command message to the controller and waits for a reply message.

- A **command message** is an instruction to either execute a function, assign a parameter value, or query a parameter value.
- A **reply message** is the response to the command message.

By sending command messages and receiving reply to messages, the user can operate the servo controller to perform the required task.

Application logic can be programmed at the software level utilizing low-level (AAComm) or high-level (AAMotion) APIs to send commands from the host. The APIs take care of establishing the communication channel and message formatting.

Alternatively, logic can also be embedded in the controller via the user program scripting feature. For details, see Agito *User Program Language Manual*.

In general, Agito controllers do not initiate messaging. An exception is when an Agito servo controller implements the Controller-to-Controller CAN bus feature; in such cases, the controller assumes the role of a client and sends messages to other controllers on the same CAN bus.

For details, see Agito *Programming Manual – Controller to Controller Communication* (not yet available).

1.3 Keywords

Keywords are essentially the servo controller's vocabulary. Each command message includes a unique keyword that defines the subject.

Communication keywords are explained in detail in the Agito *Keyword Reference Manual*. It is strongly recommended that you review the manual and familiarize yourself with the keywords used in Agito servo controllers.

Keywords are presented here briefly to give you a basic understanding of their use. Keywords have a number of attributes. The following tables uses the keyword **Vel** to explain their meaning and use of keyword attributes.

Table 1. Vel – Keyword Attributes

Attribute	Value	Description
Mnemonic (ASCII)	Vel	A string of 1 to 13 characters (1-13 bytes) that represents the keyword. The mnemonic representation provides readability.
CAN Code (binary)	5	A value between 0 to 1023 (10-bits) that represents the keyword. The CAN Code representation.
Type	Parameter	Indicates whether the keyword is a function or a parameter. **function:** executes an action (such as ABegin) **parameter:** reads or writes values to a memory. Assigns or queries a parameter when called. It can be read and/or written to, and it can be an array or a single element.
Access	Read only	Indicates whether a parameter is Read only or Read/Write. Although the general definition of parameters allows assignment and query of the parameter, some parameters are read-only (support query messages only). A read only parameter, for example, is the keyword **DInPort**, which serves to read (query) the digital inputs of the servo controller (as there is no way to assign a value to the digital inputs). Functions always appear as Read only.
Allowed during motion	Yes	Defines whether a value can be assigned to a parameter keyword, or if a function keyword can be executed, while the related axis is in motion.
Allowed during motor on	Yes	Defines whether a value can be assigned to a parameter keyword, or if a function keyword can be executed, while the related axis is in motor on (servo on) state.
Array with index range of	1:3	Applicable only for array parameters. If a parameter keyword is an array, the minimum and maximum allowed indices for the array are listed.
Save to flash	No	Applicable only for parameter keywords. Indicates whether the parameter value is saved to flash upon Save Parameters to Flash command message. These parameters are loaded from the flash upon power on or reset, or upon Load Parameters from Flash command message.

Attribute	Value	Description
Axis related	Yes	Indicates whether the keyword is relevant to a specific axis.
Min value	-2,147,483,648	
Max value	2,147,483,647	
Default value	0	Some parameters are assigned a default value upon power on or reset. In such cases, this attribute indicates the power on/reset value of the parameter. Other parameters are loaded from the flash memory upon power on or reset (restoring their values as last saved to the flash). While loading, the value of each parameter is verified to ensure that it is within the allowed range. If the value is not downloaded to the controller, the parameter will be assigned this default value. For a parameter with user unit scaling, the scaling will not be applied to the default value when it is assigned during power on or reset.
User units	in user units	Certain keywords can be scaled internally by a user-defined factor. This allows you to specify the value in other units. For example, you might want to specify velocity as mm/s instead of counts/s. If the encoder resolution is 1 count/µm, then you can scale the velocity by 1/1000 to show it in mm/s. This attribute simply indicates if the keyword is scaled.

1.4 Command Messages

Command messages are sent to the servo controller to do one of the following:

- query a parameter value or list of values
- assign a parameter value
- execute a function

In general, each command message is comprised of 2–4 pieces of information,

- **Axis reference** indicates to the controller which axis the command applies to.
  For example, the ASCII command **BSpeed** queries for the speed parameter of Axis **B**.
- **Keyword** specifies the parameter/function of interest. For example, the ASCII command **ASpeed** queries for the **speed** parameter of Axis A.
- **Index** is applicable only for array keywords. For example, the ASCII command **AVel[2]** queries for the second element of the velocity parameter.
- **Value** is only applicable during value assignment of write-enabled parameters. For example, the ASCII command **ASpeed=100** assigns a value of **100** to the speed parameter of Axis A.

For certain communication channels, auxiliary information might be embedded into the command message as well. For TCP/IP, the prefix A, L, or I before the command indicates ASCII data formatting. For RS485, an appended digit in front of the message specifies the controller which the message is addressed to.

Certain keywords are axis-related while others are not. For keywords that are not axis-related, you must add a placeholder, although it will not have any significance. For complete details, refer to Agito *Keywords Manual*.

1.5 Reply Messages

For every command message received by the controller, the controller responds with a reply message.

The response to a parameter assignment or function execution is either OK or NOT OK.

The response to a parameter query is the queried value or a list of queried values.

The following table provides an example of an exchange of messages to check the current position, and instruct the controller to move the motor to another position.

Example of Command and Reply Messages

Command/Reply	Type	Meaning
APos	Parameter value query	**Pos** is a parameter keyword that contains the current position of the system. The command is a query to retrieve the current position of Axis A.
1000>	Queried value	The response 1000> means the position of Axis A is at 1000 counts.
AAbsTrgt=10000	Parameter value assignment	**AbsTrgt** is a parameter keyword that contains the target position of the motion. The command is an assignment that sets the target position of the next motion.
OK>	OK	The response OK> means that AAbsTrgt parameter was successfully set to 10000.
ABegin	Function execution	**Begin** is a function keyword that begins the motion. The command is an execution command to begin the motion for Axis A.
ERR 39>	NOT OK	The response ERR 39> indicates that there was an error in executing the command. Error code 39 means **Can't start motion if motor is off**. For details, refer to the error list in Agito *Keyword Reference Manual*.

2 Communication Channels

In general, Agito servo controllers support the following communication channels:

- **Serial RS232** (ASCII encoding) is the most basic communication protocol and can be easily implemented. However, it lacks efficiency and speed as compared to other protocols.

  On Agito products, RS232 is supported over an RJ45 interface as well as a Micro USB port with built-in USB-to-RS232 hardware.

- **Serial RS485** (ASCII encoding) is an extension of RS232. Both RS485 and RS232 use the same RJ45 interface ports. The two protocols also have some of the same communication parameters (such as baud rate and parity).

  The difference between serial protocols is that RS232 is full duplex while RS485 is half-duplex. RS485 also utilizes an addressing scheme since it supports a 1-M connection.

  On Agito products, RS485 is supported over an RJ45 interface.

RS485 support not available on newer products

RS485 is only supported for older products and is not available on newer products such as AGM800 and AGD155 EtherCAT.

- **CAN bus** (binary encoding). The CAN bus is faster than serial communication and supports controller-to-controller communication.

  On Agito products, the CAN Bus is supported over an RJ45 interface.

  Agito servo controllers support controller-to-controller messaging over CAN.
  See Agito *Programming Manual – Controller to Controller Communication* (not yet available).

- **Ethernet TCP/IP** (binary or ASCII encoding) is the recommended communication channel for connecting the controller and the PC. It provides higher communication speeds, which facilitates swift data downloading and uploading. Certain features, such as FPGA download, are also supported only over Ethernet connections.

  On Agito products, Ethernet TCP/IP is supported over an RJ45 interface.

The following table summarizes the main differences between the various communication protocols.

Communication Channel Comparison

Property		RS232	RS485	CAN Bus	Ethernet
Hardware	Hardware layer	RJ45 USB	RJ45	RJ45	RJ45
Protocol	Base protocol	8-N-1	8-N-1	CAN 2.0A	TCP/IP
Encoding	ASCII	✓	✓	–	✓
	Binary	–	–	✓	✓
Usage	Operational	✓	✓	✓	✓
	Maintenance	✓	–	✓	✓
	Development	✓	–	✓	✓
Address	Broadcast address	NA	NA	0x400	NA
	Default address			0x400	172.1.1.101
	Configurable by parameter			✓	✓
	Configurable by DIP switch			✓	✓
Baud rate	Default baud rate (bits/s)	115,200	115,200	1,000,000	NA
	Max baud rate (bits/s)	115,200	115,200	1,000,000	
	Configurable by parameter	✓	✓	✓	
	Configurable by DIP switch	–	–	✓	
Supported functions	Standard features	✓	✓	✓	✓
	Agito PCSuite	✓	–	✓	✓
	API libraries (AAComm, C#)	✓	–	✓	✓
	API libraries (AAMotion, C#)	–	–	–	✓
	User program download	✓	–	✓	✓
	Firmware download	✓	–	✓	✓
	FPGA download	–	–	–	✓

Note: RS485 is not supported on newer Agito products.

3 Hardware Connections

3.1 Connection Interfaces

All Agito products have the same communication IN/OUT RJ45 ports, which are used for RS232, RS485, and CAN/Ethernet communication

The RJ45 connector pinout is the same in all products, as shown in the following table.

CAN, RS232, RS485 – RJ45 Connector Pinout

Pin #	Name	Description
1	GND	Digital ground
2	RS232_RX	RS232 input (product receive)
3	RS232_TX	RS232 output (product transmit)
4	RS485_B	RS485 bus, inverted
5	RS485_A	RS485 bus, not inverted
6	NA	Reserved for future use
7	CAN_L	CAN bus, Low
8	CAN_H	CAN bus, High

Connector type	RJ45 LAN 10/100Base-T connector
Mating connector part number	Any CAT5e compatible shielded connector
Cable	CAT5e or higher, standard Ethernet straight cable
Wiring	26 AWG, insulation rated for 100V

CAN bus and RS485 terminators

CAN bus lines have an optional 120Ω terminator that is connected/disconnected by DIP switch #1. Setting dip switch #1 to the ON position connects a 120Ω terminator between CAN_H and CAN_L. The terminator is required only in the last unit in the CAN bus chain.

RS485 lines have an integrated 120Ω terminator.

Ethernet – RJ45 Connector Pinout

Pin #	Name	Description
1	TX+_D1	Transmit data +
2	TX-_D1	Transmit data -
3	RX+_D2	Receive data +
4	BI+_D3	Bi-directional +
5	BI-_D3	Bi-directional +
6	RX-_D2	Receive data -
7	BI+_D4	Bi-directional +
8	BI-_D4	Bi-directional -

Connector type	RJ45 LAN 10/100Base-T connector
Mating connector part number	Any CAT5e compatible shielded connector
Cable	CAT5e or higher, standard Ethernet straight cable
Wiring	26 AWG, insulation rated for 100 V

RS232 – Micro USB

Pin #	Name	Description
1	Vcc	5V
2	D-	Data-
3	D+	Data+
4	ID	USB OTG ID
5	GND	GND

Connector type	Micro USB 2.0 B
Cable	Any Micro USB 2.0 B-type cable
Wiring	20–28 AWG, insulation rated for 100V

USB to RS232 bridge

The Micro USB connection is implemented using an internal converter/adapter from USB to RS232 (UART). Typically, the Windows OS contains a built-in driver for the convertor/adapter. If necessary, you can access drivers at: http://www.ftdichip.com/Drivers/D2XX.htm

3.2 RS232 Wiring

RS232 is supported over two physical layers – the Micro USB and the RJ45 port.

3.2.1 RJ45 Serial RS232 Wiring

To interface a PC via the RJ45 port for RS232 communication, an FTDI (USB-UART 232) adapter is required.

The following figure below shows an example using an FTDI adapter and an interface cable to establish a hardware connection via the RS232 RJ45 port.

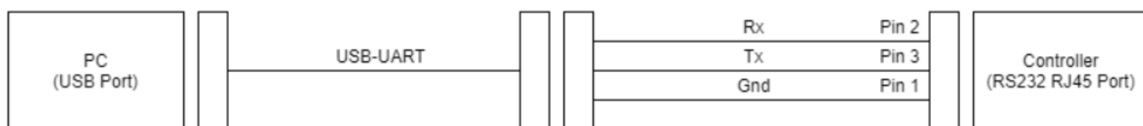

Figure 1.

3.2.2 Micro USB Serial RS232 Wiring

Agito controllers have built-in USB-UART chips to convert the signal from USB to UART. Thereafter, an internal bridge relays the signals to the common UART used for RS232 RJ45. As such, performance is identical over both layers.

The following figure shows an example using a standard USB-USB Micro cable to establish a hardware connection via the RS232 Micro USB port.

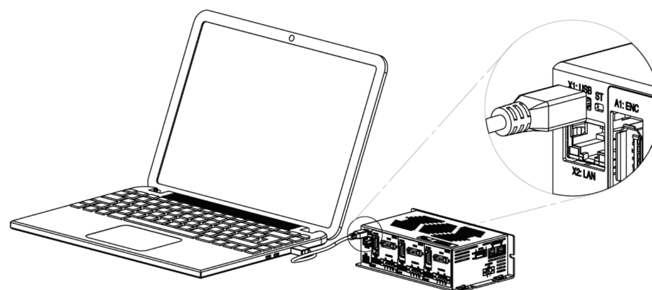

Figure 2.

3.3 RS485 Wiring

To interface a PC via the RJ45 port for RS485 communication, an FTDI (USB-UART 485) adapter is required.

The following figure shows an example using an FTDI adapter and an interface cable to establish a hardware connection via the RS485 RJ45 port. Standard shielded Ethernet cables can be used for daisy chaining controller to controller.

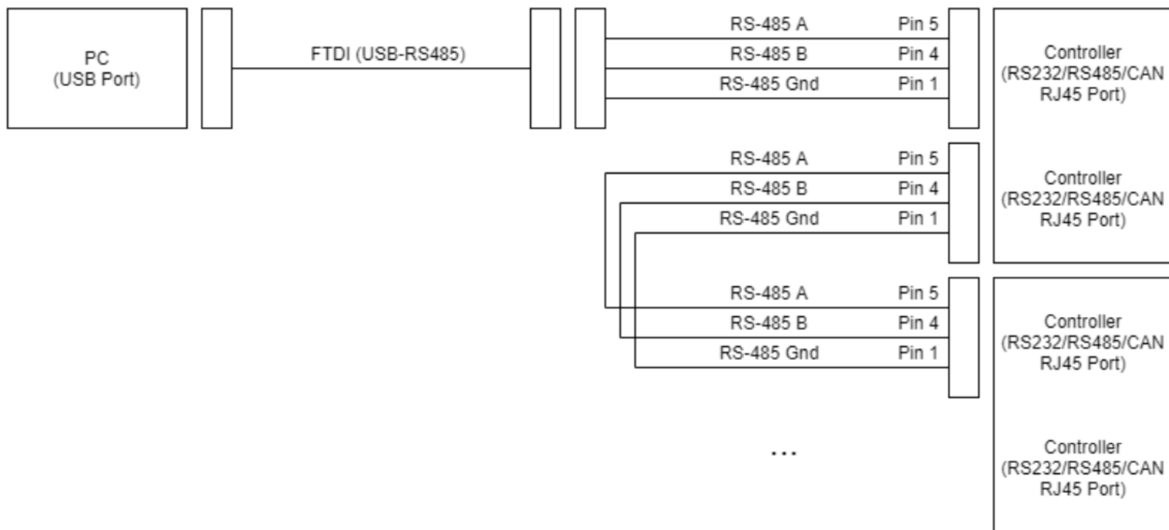

Figure 3.

3.4 CAN Bus Wiring

To interface a PC via the RJ45 port for CAN Bus communication, a CAN bus adapter is required.

The following figure shows an example using a CAN adapter and an interface cable to establish a hardware connection via the CAN RJ45 port. Standard shielded Ethernet cables can be used for daisy chaining controller to controller.

Agito has configurable built-in termination, so there is no need to add an external terminating resistor. On the last controller, toggle DIP switch [1] to ON state to terminate the CAN bus.

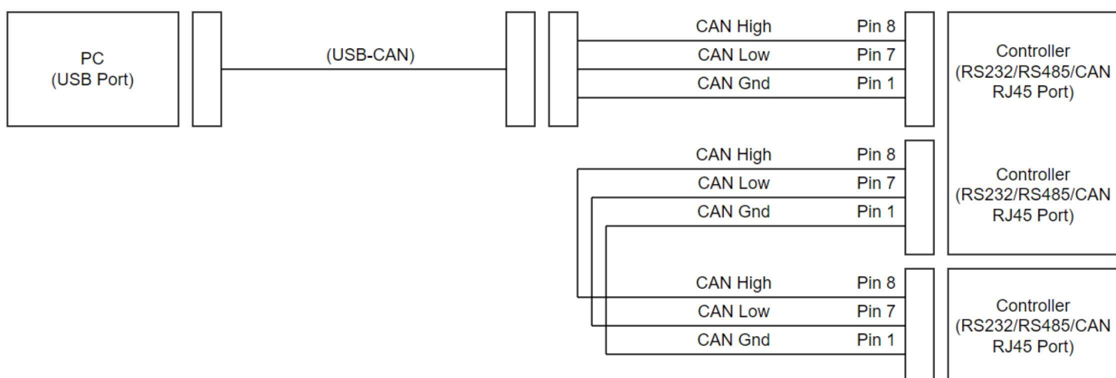

Figure 4.

3.5 Ethernet Wiring

To interface a PC via the RJ45 port for Ethernet communication, a standard shielded Ethernet cable is required.

4 Communication Settings

4.1 RS232 Communication Settings

Agito servo controller RS232 channels are fixed to utilize the 8-N-1 (8 data bits, no parity, 1 stop bit) serial port encoding. Only the baud rate parameter can be changed.

The baud rate can be changed through the keyword **RSBaud[]**.

RSBaud[ ] is an array with two elements:

- RSBaud[1] can be used to set the baud rate for the RS232 over the Micro USB port
- RSBaud[2] is for the RS232 over the RJ45 port.

The following table presents the configurable parameters for RS232 communication:

RS232 Communication Parameters

Parameter	Type	Value	Functions	Default Value
RSBaud[1] (Micro USB) RSBaud[2] (RJ45)	Software	1	Sets baud rate to 9,600 bits/s	4
		2	Sets baud rate to 19,200 bits/s	
		3	Sets baud rate to 38,400 bits/s	
		4	Sets baud rate to 115,200 bits/s	

4.2 RS485 Communication Settings

The following table presents the configurable parameters for RS232 communication:

RS485 Communication Parameters

Parameter	Type	Value	Functions	Default Value
RSBaud[2] (RJ45)	Software	1	Sets baud rate to 9,600 bits/s	4
		2	Sets baud rate to 19,200 bits/s	
		3	Sets baud rate to 38,400 bits/s	
		4	Sets baud rate to 115,200 bits/s	
ChainAddress	Software	-1	RS485 is disabled	-1
		0–7	RS485 is enabled and the address is equivalent to the value	

The software parameter **ChainAddress** can be used to set the RS485 address of the controller.
The allowable values for ChainAddress are in the range -1 to 7.
A value of -1 indicates the RS485 feature is disabled.
If the value is the range 0 to 7, the value is used as the address.

Note – Changing the address

After changing the address, you must save to flash and reset the controller for the change to take effect.

Note – Using RS232 and RS485 at the same time

Since RS485 and RS232 share the same UART, they cannot be used at the same time. However, even when RS485 is enabled, the controller still responds over RS232. This allows you to switch back to RS232 at any time by connecting the RS232 cable. It is important that you do not use both channels at the same time as an error will occur.

4.3 CAN Bus Communication Settings

For CAN communication, three software parameters are configured: **base address**, **baud rate**, and **list delay**.

In the software you can set the **base address** using the keyword **CANAddr**.

Using the DIP switch, you can add an offset to the address or force the base address to 64.

CAN Initial Address is the address that is used as the unit's preliminary address.

The following table presents the configurable parameters for CAN communication:

CAN Communication Parameters

Parameter	Type	Value	Functions	Default Value	Remarks
CANBaud (RJ45)	Software	1	Sets baud rate to 31,250 bits/s	6	
		2	Sets baud rate to 62,500 bits/s		
		3	Sets baud rate to 125,000 bits/s		
		4	Sets baud rate to 250,000 bits/s		
		5	Sets baud rate to 500,000 bits/s		
		6	Sets baud rate to 1,000,000 bits/s		
CANAddr	Software	1-2032	CAN Base Address	64	Use only in increments of 16, such as 64, 80, 96. (Each controller uses 15 trailing addresses for internal purposes.)
CANDelay	Software	0-1000	Sets the number of samples to delay the messages by	6	Does not affect normal communication rate.
DIP switch [1]	Hardware	0	No termination	0	Set to 1 for the last unit in the CAN bus network.
		1	CAN bus termination		
DIP switch [3]	Hardware	0	CAN Initial Address = CANAddr + CAN Offset	0	See the section CAN Addressing Scheme.

Parameter	Type	Value	Functions	Default Value	Remarks
		1	CAN Initial Address = 64 + CAN Offset		
DIP switch [4–6]	Hardware	000–111	CAN Offset is equal to the decimal value of the binary switches multiplied by 16	000	For example, if DIP switch 4 and 5 are on, the binary is [110], and the offset is 6 *16.

CAN Initial Addresses should be set in increments of 16 as each controller requires 16 addresses for internal purposes. For example, use address 64 for one controller and 80 for another. It is recommended that you always set the CAN Base Address (and therefore the CAN Initial Address) to a multiple of 16.

Agito servo controllers implement CAN 2.0A with an 11-bit identifier. This means there is a permutation of 2,048 possible address. Agito servo controllers restrict the maximum value for the CAN Base Address to 2,032, such that the 15 trailing addresses (to 2,047) are available for internal use. However, if the CAN Base Address is set to a high value and the DIP switches are used to add an offset, it is possible the CAN Initial Address might exceed 2,047. In such a case, an overflow will revert to a lower address. For example, if CANAddr is set to 2,032 and DIP switches [4-6] are toggled on, the expected CAN Initial Address will be 2,032 + 7*16 = 2,144. Since the maximum value of an 11-bit identifier is 2,047, the CAN Initial Address will actually be 2,144 - 2,048 = 96.

CAN Addressing Scheme

As mentioned, each servo controller uses a bank of 16 CAN addresses, which is organized into 8 pairs of Receive and Transmit addresses, as follows:

Bank Initial Address	Main address for receiving incoming messages.
Bank Initial Address + 1	Replies to incoming messages at Bank Initial Address are sent to this address.

These first two addresses are the main Receive/Transit CAN addresses of the servo controller. In normal operation, the client should send messages on Bank Initial Address and wait for replies on Bank Initial Address + 1.

The additional 7 pairs of addresses are alternative Receive/Transmit CAN addresses of the servo controller, which act just as the first pair of alternative addresses. This allows for other network architectures (such as, multiple client network, peer-to-peer network) to be implemented.

Bank Initial Address + 2	Alternate address for receiving incoming messages.
Bank Initial Address + 3	Replies to incoming messages at Bank Initial Address + 2 are sent to this address.

Bank Initial Address + 14	One more alternate address for receiving incoming messages.
Bank Initial Address + 15	Replies to incoming messages at Bank Initial Address + 6 are sent to this address.

This addressing scheme enables flexibility, as it supports multiple clients and servers, and allows the server to reply to each client independently.

4.4 Ethernet Communication Settings

For Ethernet communication parameters, two software parameters must be configured – **EthernetIP** and **EthernetPort**.

When using multiple controllers in the same network, the **Ethernet MAC** addresses of the controllers must not overlap. Out-of-factory, all controllers will be configured with different Ethernet MAC addresses and you should not need to touch this variable.

With the DIP switch, you may add an offset to the address or force the IP address to the default.

The following table presents the configurable parameters for Ethernet communication.

Ethernet Communication Settings

Parameter	Type	Value	Functions	Default Value	Remarks
EthernetIP[1]	Software	0–255	First octet of the controller's IP address.	172	
EthernetIP[2]			Second octet of the controller's IP address.	1	
EthernetIP[3]			Third octet of the controller's IP address.	1	
EthernetIP[4]			Fourth octet of the controller's IP address.	101	
EthernetPort	Software	0–65,535	Sets the controller's Ethernet port number.	50,000	
EthernetMAC[1-6]	Software	0–255	Assigns the MAC address for Ethernet communication. Each controller on the same network must have a unique MAC address.	-	
DIP Switch [3]	Hardware	0	IP address = EthernetIP[1]. EthernetIP[2]. EthernetIP[3]. EthernetIP[4]+IPOffset	0	
		1	Force IP address to 172.1.1.101		

5 Message Structure

Agito supports two types of data encoding for its messages – ASCII encoding and binary encoding.

When using Agito controllers, you will typically encounter the ASCII syntax. ASCII encoding is supported over the serial and Ethernet communication. It is also used to represent information in a readable manner in Agito PCSuite (such as Terminal, IDE+, and Data Recorder) and APIs.

It is recommended that you become familiar with the ASCII syntax to enable more effective use of the controller.

The binary format is more effective than the ASCII format as the data is represented is a manner that keeps the data size compact and consistent. A disadvantage of the binary format is that it is not directly readable. However, APIs and software solutions can perform low level interpretation of the binary format for you.

The use of binary format over TCP/IP is recommended for best communication speeds.

5.1 ASCII Encoding

ASCII encoding is supported over RS232, RS485, and Ethernet TCP/IP. The ASCII syntax over these three communication channels varies slightly to accommodate channel-specific functionality.

The ASCII syntax for these three communication channels extends from a base ASCII syntax.

5.1.1 ASCII Syntax (Base)

ASCII Command (Base)

To construct a **base ASCII command**, append the keyword mnemonic after the axis reference. This forms a command to query a parameter value or execute a function.
For array keywords, append the index within square brackets after the keyword mnemonic.
For parameter value assignment, append the symbol = followed by the value.

	<base ASCII command>						
	<axis reference>	<keyword mnemonic>	[	<index>	]	=	<value>
Example	A	Pos					
	A	Begin					
	A	Vel	[	2	]		
	A	Speed				=	11888
	A	GenData	[	50	]	=	888

Case format

The axis reference must be in upper case. The case of keyword mnemonic is irrelevant. For better readability in the examples, UpperCamelCase format is used.

ASCII Reply (Base)

A base ASCII reply is terminated with the symbol >.

There are four types of base ASCII replies:

- **OK**. The reply **OK>** indicates that the command message was received and successfully processed by the servo controller.

	<base ASCII reply>	
	OK	**>**
Example	OK	>

- **NOT OK**. The reply **ERR X>** indicates that the command message was not processed by the servo controller due to some error. In this case, the error code is also included in the reply message.

  Refer to Agito *Keyword Reference Manual* **ErrLog** keyword for the full list of errors.

	<base ASCII reply>		
	ERR	**<error code>**	**>**
Example	ERR	39	>

- **Queried Value.** Indicates that the command message to query a parameter value was received and successfully processed by the servo controller. The reply message includes the queried value.

	<base ASCII reply>	
	<value>	**>**
Example	11888	>

- **List of Queried Values**. Special keywords such as RecUpload or AllStat return a list of parameters instead of a single parameter. The list is delimited by commas and semicolons (to separate sub-lists). (The functionality of RecUpload and AllStat are not explained in this document.)

	<base ASCII reply>											
	<value>	**,**	**<value>**	**,**	**…**	**;**	**<value>**	**,**	**<value>**	**,**	**…**	**>**
Example	87	,	23	,	34	;	11	,	48	,	64	>

5.1.2 RS232 ASCII Syntax

RS232 ASCII syntax is used for sending messages over RS232/USB communication channels.

RS232 ASCII Command Messages

The RS232 ASCII command syntax is simply the base ASCII command with a carriage return appended at the end.

	<RS232 ASCII command syntax>	
	<base ASCII command>	<carriage return>
Example	APos	\r
	AVel	\r
	AAbsTrgt=888	\r
	ABegin	\r

RS232 ASCII Reply Messages

The RS232 ASCII reply syntax is simply the base ASCII reply with a carriage return appended at the end.

	<RS232 ASCII reply syntax>	
	<base ASCII reply>	<carriage return>
Example	200>	\r
	0>	\r
	OK>	\r
	ERR 39>	\r

5.1.3 RS485 ASCII Syntax

RS485 ASCII Command Messages

The syntax for RS485 ASCII is an extension of the RS232. Since RS485 enables daisy chaining of multiple slave controllers, an additional parameter is used to specify the device for which the message is intended. This is represented by an additional digit (0–7) in front of the message.

	<RS485 ASCII command syntax>		
	<address>	<base ASCII command>	<carriage return>
Example	0	APos	\r
	0	AVel	\r
	0	AAbsTrgt=888	\r
	0	ABegin	\r

RS485 ASCII Reply Messages

The syntax for RS485 ASCII is identical to that of the RS232. Addressing is not required as there is only one master.

	<RS485 ASCII reply syntax>	
	<base ASCII reply>	<carriage return>
Example	200>	\r
	0>	\r
	OK>	\r
	ERR 39>	\r

5.1.4 Ethernet ASCII Syntax

The syntax for Ethernet ASCII is an extension of the base ASCII command and reply. Ethernet allows for both ASCII and binary syntax. An additional character – A, I or L – is added as a prefix to specify that ASCII format is being used. For example, AAPos, IAPos, LAPos.

Ethernet ASCII Message Types

There are three message types – A, I and L. The following table summarizes the differences between the three types.

Ethernet ASCII Types

	A – Standard	I – Intermittent	L – Lengthy
Reply in ASCII/binary	Reply in binary	Reply in ASCII	Reply in ASCII
Bulk messaging (concatenation of messages)	Single message only	Bulk messaging	Bulk messaging
Continue upon error	NA	Stops processing the remaining commands if there is an error in one of the commands	Continues processing remaining commands even if there is an error in a previous command

Ethernet ASCII Command Messages (Standard)

Prefixing an **A** indicates that the command is of standard mode. Although the command is in ASCII, the reply from the controller will be in binary. This mode only supports single commands. Refer to the chapter on Ethernet binary reply for the reply.

	<Ethernet ASCII command syntax>		
	<type>	<base ASCII command>	<null char>
Example	A	APos	\0
	A	AVel	\0
	A	AAbsTrgt=888	\0
	A	ABegin	\0

Ethernet ASCII Bulk Command Messages (Intermittent)

Prefixing an **I** indicates bulk messaging – intermittent mode. In intermittent mode, an error in processing any of the messages will cause termination of the remaining commands in the message. This is useful in scenarios when the subsequent command is dependent on the previous.

For bulk messaging, a semi-colon **;** is used to delimit the individual commands. The last command should also be terminated with ;. The null character can also be used instead of the semi-colon.

	<Ethernet ASCII command syntax>						
	<type>	<base ASCII command>	;	<base ASCII command>	;	…	
Example	I	AMotorOn=1	;	AAbsTrgt=888	;	ABegin	;
	I	AMotorOn=1	\0	AAbsTrgt=888	\0	ABegin	\0

Ethernet ASCII Bulk Reply Messages (Intermittent)

Following the example above, the bulk command **IAMotorOn=1;AAbsTrgt=888;ABegin;** might be issued. If the second command fails, the subsequent commands will not be executed. The reply to such a message might be **OK>ERR 8>**. In this case, **ABegin** is not executed and there will be no reply to the command.

	<Ethernet ASCII command syntax>	
	<base ASCII reply>	…
Example	OK>	ERR 8>

Ethernet ASCII Bulk Command Messages (Lengthy)

Prefixing an **L** indicates bulk messaging – lengthy mode. In lengthy mode, the controller will continue processing the subsequent commands even if an error occurs for one. This mode is suitable if the commands are independent of one another.

For bulk messaging, a semi-colon **;** is used to delimit the individual commands. The last command should also be terminated with **;**. The null character can also be used instead of the semi-colon.

	<Ethernet ASCII command syntax>						
	<type>	<base ASCII command>	;	<base ASCII command>	;	…	
Example	I	AMotorOn=1	;	AAbsTrgt=888	;	ABegin	;
	I	AMotorOn=1	\0	AAbsTrgt=888	\0	ABegin	\0

Ethernet ASCII Bulk Reply Messages (Lengthy)

Following the example above, if the bulk command **LAMotorOn=1;AAbsTrgt=1000;ABegin;** is issued, the reply to such a message might be **OK>ERR 8>OK>**. In such a case, the motor will move to an unintended target position (old target), which could be dangerous.

	<Ethernet ASCII command syntax>		
	<base ASCII reply>	<base ASCII reply>	…
Example	OK>	ERR 8>	OK>

5.2 Binary Encoding

Unlike the ASCII syntax, the binary syntax keeps the length of the messages consistent and compact. This is accomplished by representing each piece of information using a fixed data size.

- **Axis References** are unsigned and represented using 5 bits, with value range of 0 to 31. Axis A is represented by 0, Axis B by 1, Axis C by 2, and so on.

Axis Reference

Axis Ref	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	…
Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…

- **Keywords** are unsigned and represented by their **CAN Code** in 10 bits, with value range of 0 to 1,023. Every Agito keyword is identified by a unique CAN Code. See Agito *Keyword Reference Manual* for the full list.
- **Index values** (for array keywords) are unsigned and represented by 16 bits, with value range of 0 to 65,535.
- **Parameter values** are signed and represented by 32 bits, with value range of 2,147,483,648 to -2,147,483,647.

5.2.1 Binary Syntax (Base)

All **binary commands** can fit into 8 bytes or less, and messages of the same type are always of a defined size.

Similarly, all **binary replies** can fit into 4 bytes or less, except for List of Queried Values reply messages.

Binary Command (Base)

For communication, the Big-Endian convention is adopted.

<axis ref + CAN> - The Axis Reference (5 bits) and CAN Code (10 bits) are used together and arranged into a word as follows.

Byte	0								1							
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Content	-	Axis Ref					CAN Code									

<index> - The Index (16 bits) is represented with a word as follows.

Byte	0								1							
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Content	Index															

<value> - The Parameter value (32 bits) is represented with a double word as follows.

Byte	0								1								2								3							
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Content	**Parameter Value**																															

	<Base Binary Command>							
Type	<byte0>	<byte1>	<byte2>	<byte3>	<byte4>	<byte5>	<byte6>	<byte7>
Execute Function	<axis ref + CAN>							
Example: ABegin	0x00	0x83						
Calculation	**A:0, Begin:131**, 0<<10 + 131 = 0x0083							
Query Parameter	<axis ref + CAN>							
Example: BSpeed	0x04	0x8A						
Calculation	**B:1, Speed:138**, 1<<10 + 138 = 0x048A							
Query Parameter of array element	<axis ref + CAN>		<index>					
Example: AVel[2]	0x00	0x05	0x00	0x02				
Calculation	**A:0, Vel:5, Array:2**, 0<<10 + 5 = 0x0005 **Index:2**, 2 = 0x0002							
Assign Parameter	<axis ref + CAN>		<value>					
Example: ASpeed=888	0x00	0x8A	0x00	0x00	0x03	0x78		
Calculation	**A:0, Speed:138**, 0<<10 + 138 = 0x008A **Value:888**, 888 = 0x0000 0378							
Assign Parameter of array element	<axis ref + CAN>		<index>		<value>			
Example: AGenData[10]=-200	0x00	0xED	0x00	0x0A	0xFF	0xFF	0xFF	0x38
Calculation	**A:0, GenData:237**, 0<<10 + 237 = 0x00ED **Index:10**, 10 = 0x000A **Value:-200**, -200 = 0xFFFF FF38							

Binary Reply (Base)

OK reply messages are an acknowledgement that the command was successfully executed. The base content of an **OK** reply is zero bytes in length. The constructed message is not actually empty; prefix or suffix bytes will be appended depending on the communication channel.

NOT OK reply messages indicate that the command failed due to certain reasons. The base content of a NOT OK reply is a two-byte signed integer that represents the error code.

QUERIED VALUE reply messages are responses to query value commands. The base content of a QUERIED VALUE reply is a four-byte signed integer which represent the queried value.

For special keywords such as AllStat and RecUpload, the controller will reply with a **LIST OF QUERIED VALUES**. This will not be covered in this document.

	<Base Binary Reply>							
Type	<byte0>	<byte1>	<byte2>	<byte3>	<byte4>	<byte5>	<byte6>	<byte7>
OK								
Example: OK								
NOT OK	<error code>							
Example: ERR 39	0x00	0x27						
QUERIED VALUE	<value>							
Example: 100000	0x00	0x01	0x86	0xA0				

5.2.2 CAN Bus Binary Syntax

CAN Bus Command Messages

The CAN bus command messages utilize the base binary commands.

Refer to the section Binary Command (Base).

	<CAN bus binary command syntax>
	<base binary command>
Example: ABegin	0x0083z
Example: BSpeed	0x048A
Example: AVel[2]	0x0005 0002
Example: ASpeed=888	0x008A 0000 0378
Example: AGenData[10]=-200	0x00ED 000A FFFF FF38

CAN Bus Reply Messages

For details on how to construct the message, refer to the section Binary Reply (Base).

	<CAN bus binary reply syntax>	
	<base binary reply>	<terminator>
Example: OK>	-	0x3E
Example: ERR 39>	0x0027	0x3E
Example: 100000>	0x0001 86A0	0x3E

5.2.3 Ethernet Binary Syntax

Ethernet Binary Message Types

There are three types of Ethernet binary messages. Type 0 is the standard type for single messages. Type 1 is for CNC messaging and is not covered in this document. Type 2 is for bulk messaging

	0 – Standard	1 – CNC	2 – Bulk
Bulk Messaging (concatenation of messages)	Single message only	Bulk messaging	Bulk messaging

Ethernet Binary Command Messages (Type 0 – Standard)

The message type is appended to the start of the command message. The message type 0x00 indicates that it is a standard command message.

	<Ethernet binary command syntax>	
	<type>	<base binary command>
Example: ABegin	0x00	0x0083
Example: BSpeed	0x00	0x048A
Example: AVel[2]	0x00	0x0005 0002
Example: ASpeed=888	0x00	0x008A 0000 0378
Example: AGenData[10]=-200	0x00	0x00ED 000A FFFF FF38

Ethernet Binary Reply Messages (Type 0 – Standard)

The message type is appended at the start of the reply message. The message type 0x00 indicates that it is a standard reply message.

The message is the terminated with a > which has a hexadecimal value of 0x3E.

	<Ethernet binary reply syntax>		
	<type>	<base binary reply>	<terminator>
Example: OK>	0x00	-	0x3E
Example: ERR 39>	0x00	0x0027	0x3E
Example: 100000>	0x00	0x0001 86A0	0x3E

Ethernet Binary Command Messages (Type 2 – Bulk)

The message type is appended to the start of the command message. The message type 0x02 indicates that it is a bulk command message.

Before every command, a byte is appended to indicate the length of the command. Bulk messaging supports the concatenation of up to 100 commands.

	<Ethernet binary command syntax>					
	<type>	<size>	<base binary command>	<size>	<base binary command>	…
Example: BSpeed; Vel[2];	0x02	0x02	0x048A	0x04	0x0005 0002	

Ethernet Binary Reply Messages (Type 2 – Bulk)

The message type is appended at the start of the reply message. The message type 0x02 indicates that it is a bulk reply message.

Before every reply, a byte is appended to indicate the size of the command in number of bytes, followed by the reply.

The message is the terminated with a > which has a hexadecimal value of 0x3E.

	<Ethernet binary command syntax>					
	<type>	<size>	<sub binary reply>	…	…	<terminator>
Example: ERR 39>100000>	0x02	0x02	0x0027	0x04	0x0001 86A0	0x3E

6 Examples

6.1 RS232 Communication Example

6.1.1 Connection and configuration of RS232 communication channel

1. Connect a USB-USB Micro cable from the PC to the controller. Alternatively, establish a hardware connection via the RJ45 port.
2. Download a serial terminal program such as Termite, RealTerm, or PuTTY. The following example uses Termite. Launch Termite and configure the software to use the follow settings.
   - Baud rate: 115,200
   - Data bits: 8
   - Stop bits: 1
   - Parity bits: None
   - Transmitted text: Append CR
   - Local Echo: Enable (so that command will be printed in output)

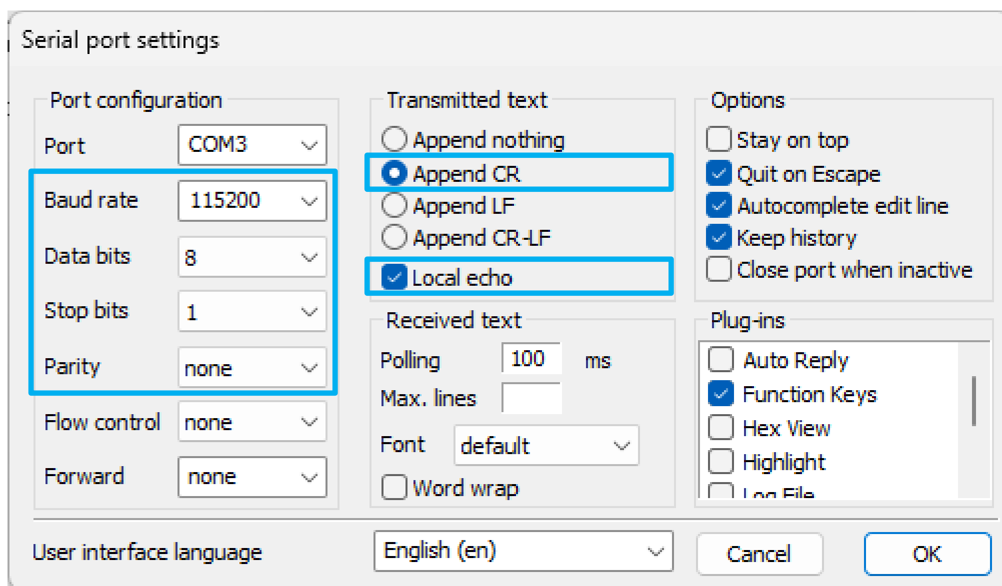

Figure 5.

6.1.2 Sending ASCII commands over RS232 communication channel

Issue the command **APos** and observe the reply.

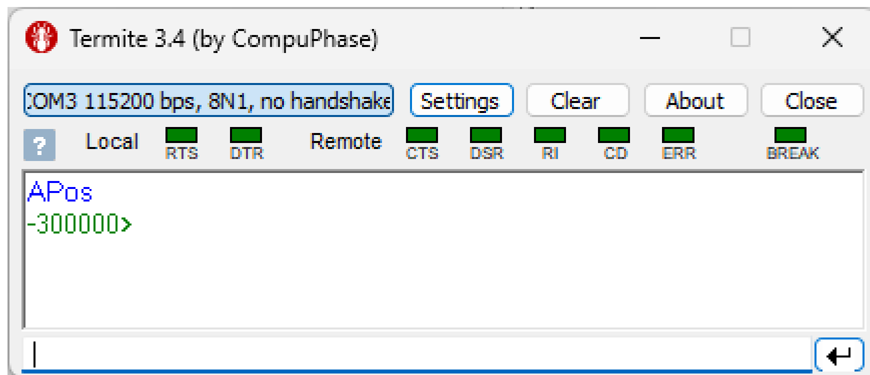

Figure 6.

For more examples, see the section RS232 ASCII Command Messages.

6.2 RS485 Communication Example

6.2.1 Connection and configuration of RS485 communication channel

1. For this example, prepare two controllers. Before an RS485 communication channel can be established, you need to configure the parameter ChainAddress. Connect the controllers over RS232 and set ChainAddress to 0 and 1 respectively. Then, save to flash and reset the controllers.
2. Follow the steps in the chapter Hardware Connections to connect two controllers in an RS485 daisy chain configuration. You need to prepare a USB to RS485 FTDI interface cable to connect the PC to the first controller. The first and second controller can be daisy chained using a regular shielded Ethernet cable.
3. Download a serial terminal program such as Termite, RealTerm, or PuTTY. The following example uses Termite. Launch Termite and configure the software to use the follow settings.
   - Baud rate: 115,200
   - Data bits: 8
   - Stop bits: 1
   - Parity bits: None
   - Transmitted text: Append CR
   - Local Echo: Enable (So that command will be printed in output)

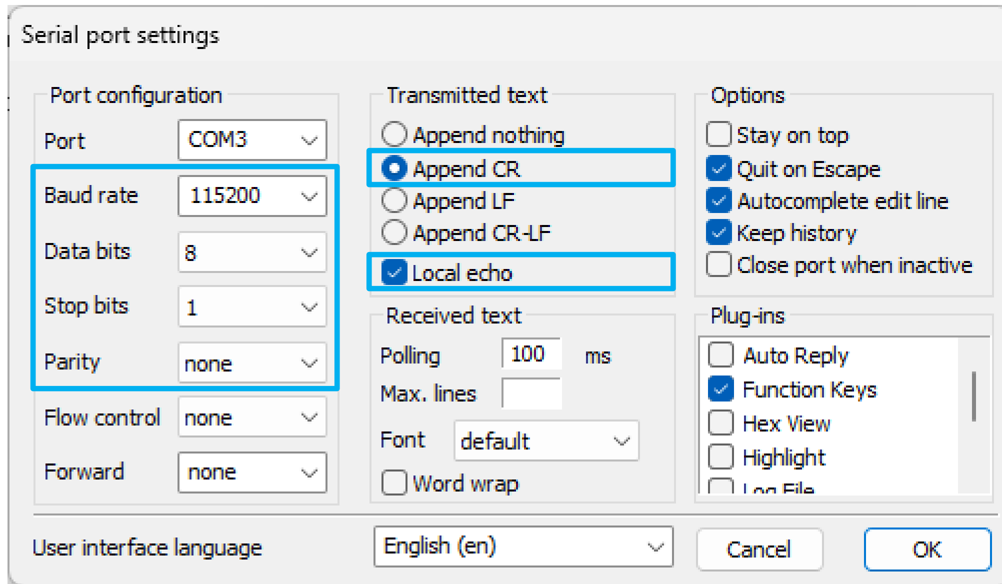

Figure 7.

6.2.2 Sending ASCII commands over RS485 communication channel

Issue the commands **0ASpeed**, **1ASpeed**. Observe the replies.

The prefix character addresses the command to the controller according to the ChainAddress parameter.

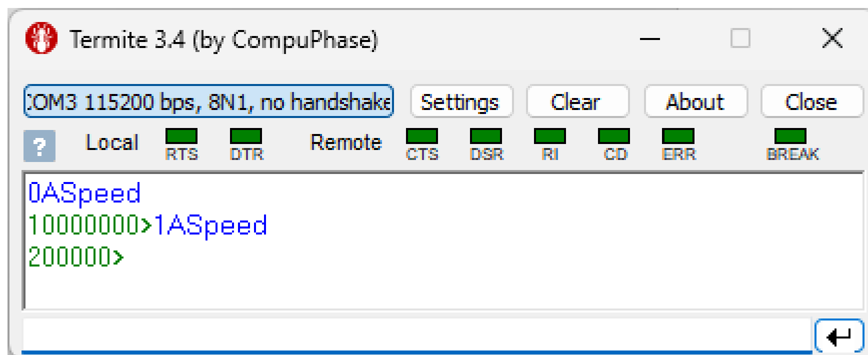

Figure 8.

For more examples, see the section RS485 ASCII Command Messages.

6.3 CAN Communication Example

6.3.1 Connection and configuration of CAN Bus communication channel

1. For this example, prepare two controllers. Before a CAN bus communication channel can be established, you must first configure the address. On the second controller, toggle DIP Switch [1] to ON state to terminate the CAN bus. Also toggle DIP Switch [6] to ON state to add an address offset of 16.
2. Follow the steps in the chapter Hardware Connections to connect two controllers in a CAN bus daisy chain configuration. You need to prepare a USB to CAN adapter to connect to PC to the first controller. The first and second controller can be daisy chained using a regular shielded Ethernet cable. For this example, the Kvaser Leaf Light v2 CAN adapter is used.
3. Download the Kvaser CANKing software and launch the application. Use a 1 CAN channel template.

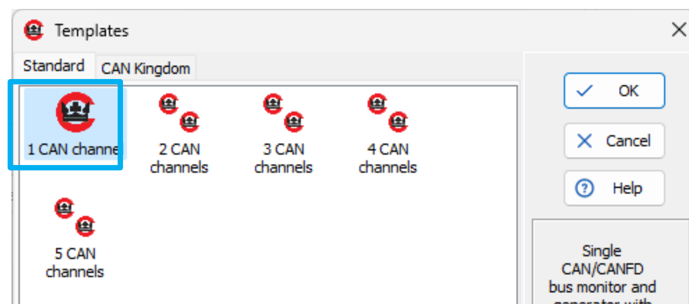

Figure 9.

4. Configure the settings as follows:

   Bus speed: 1000 kbits/s

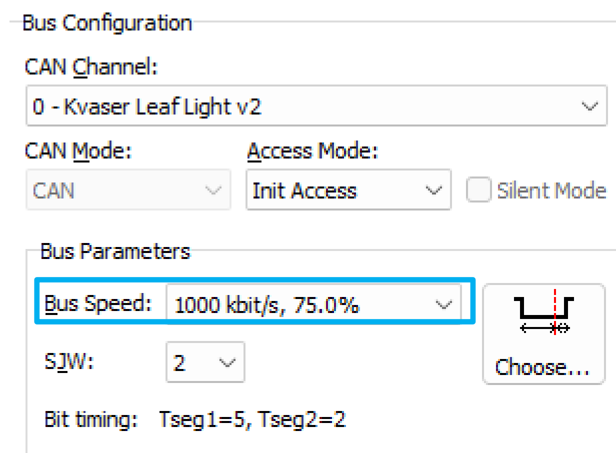

Figure 10.

5. Select **Go on Bus** to establish a connection.

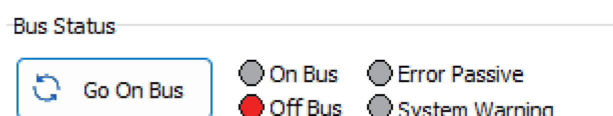

Figure 11.

6.3.2 Sending binary commands over CAN Bus communication channel

1. In the Messages tab, open the universal messenger. Set the CAN Identifier to the address of the controller to which the message will be sent: **0x40** (Decimal:64) for the first controller and **0x50** (Decimal:80) for the second controller.
   - Populate the message data as 0x048A, which is the command to query for ASpeed.
   - Set DLC to 0x02 as the message size is 2 bytes.
   - Click Send, and repeat for the other controller address.

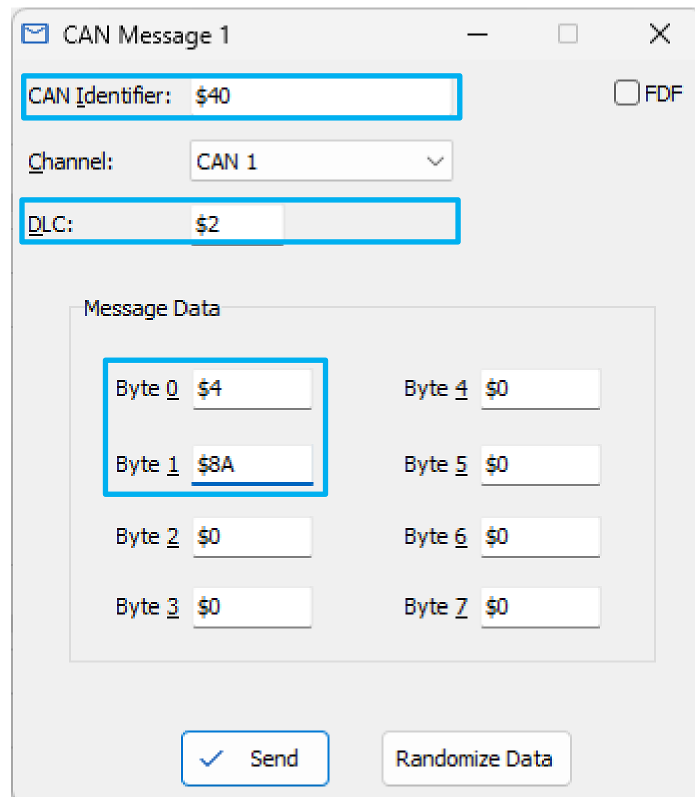

Figure 12.

2. Observe the replies. The controller replies to an address that is its address incremented by 1.

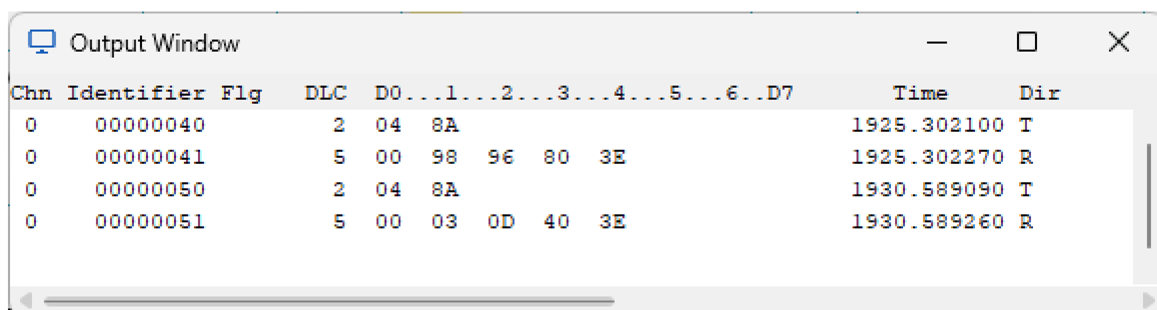

Figure 13.

6.4 Ethernet Communication Example

6.4.1 Connection and configuration of Ethernet communication channel

1. For this example, prepare a single controller. Connect an Ethernet cable from the PC to the controller's Ethernet port.
2. The default IP address of the controller is 172.1.1.101. To establish a connection, the PC's Ethernet port must be on the same subnet (172.1.1).

   Go to Windows > Start > Setting > Network & Internet > Advance Network Settings > More Network Adapter Options, to open the Network Connections explorer.

   Right-click on the Ethernet port and select **Properties**.

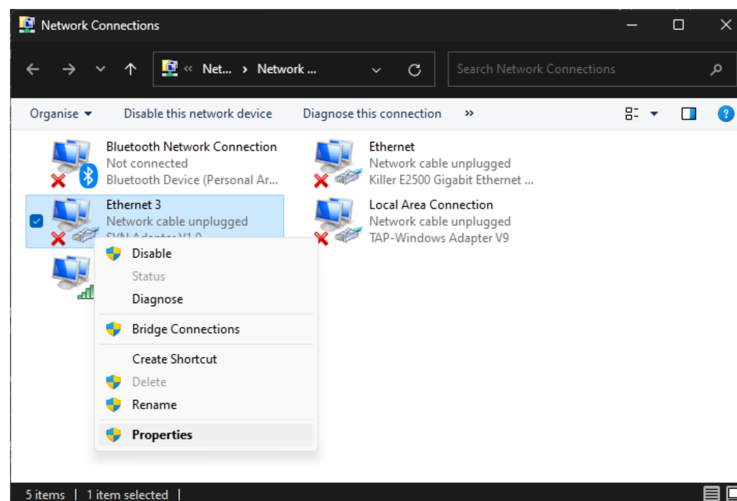

Figure 14.

3. Navigate to the Internet Protocol Version 4 (TCP/IPv4) and select **Properties**.

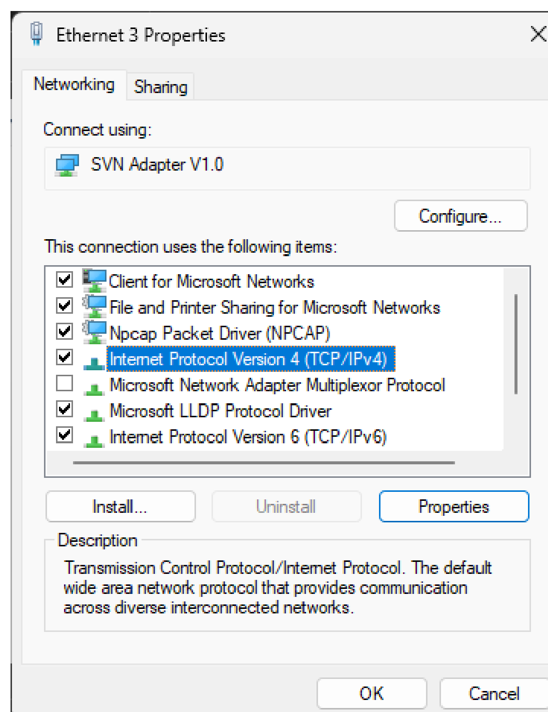

Figure 15.

4. Configure the IP address of the port to use the IP Address 172.1.1.x. The last digit may be any number as long as it is different from the controller's address.

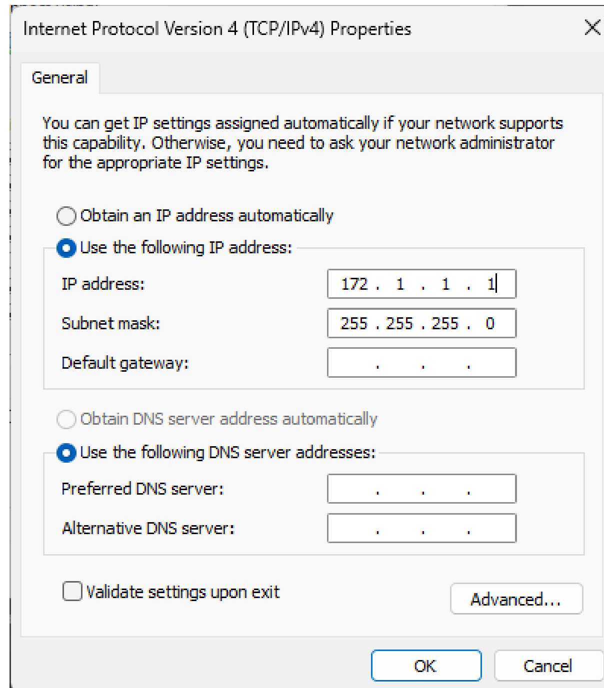

Figure 16.

5. Download software that allows you to establish a TCP/IP socket connection and send packets. For this example, Packet Sender is used.
6. Set the address to 172.1.1.101, which is the default IP address. Use port 50,000, which is the default port.

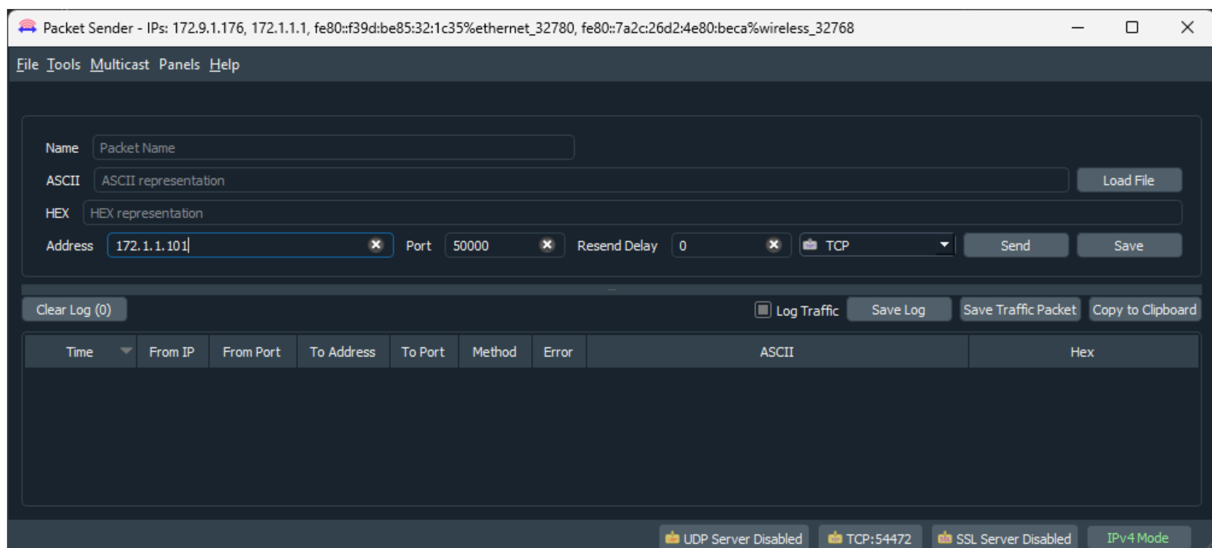

Figure 17.

6.4.2 Sending ASCII commands over Ethernet communication channel

1. In the ASCII field, enter the command **AASpeed\00**, and click Send.

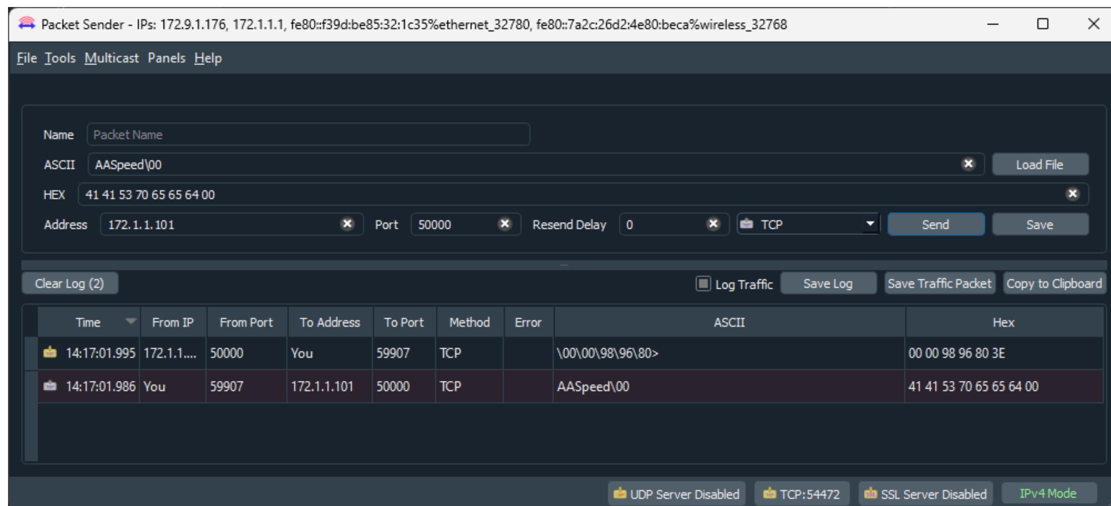

Figure 18.

The prefix character **A** indicates that the message is ASCII Standard type. **ASpeed** is the base command. **\00** is a null character that terminates the message.

Observe that the reply is in binary code.

Refer to the section Ethernet ASCII Command Messages (Standard).

Refer to the sectionEthernet Binary Reply Messages (Type 0 - Standard)

2. In the ASCII field, enter the command **IASpeed\00ABegin\00BSpeed\00**, and click Send.

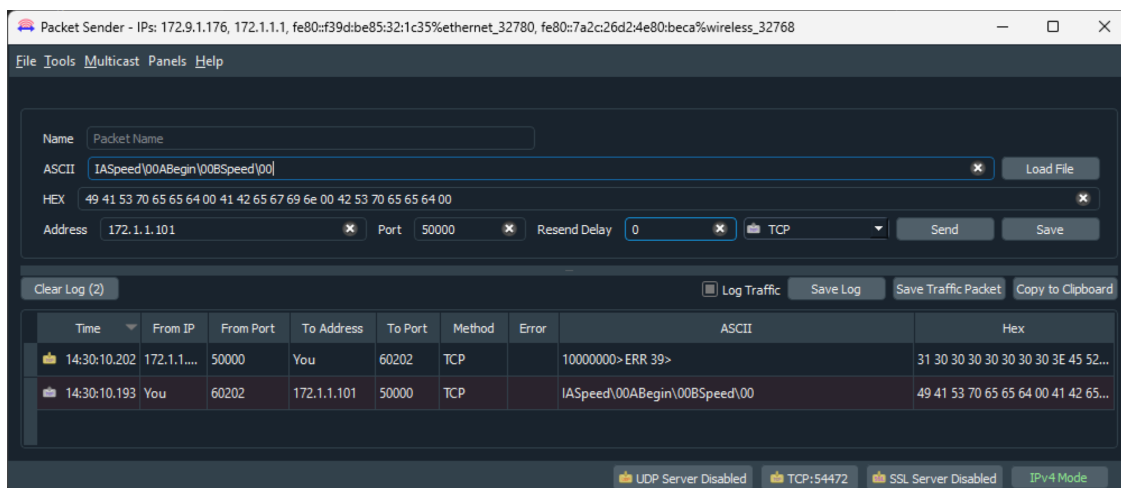

Figure 19.

The prefix character **I** indicates that the message is ASCII Intermittent type. This type of message supports the concatenation of commands that are executed sequentially. If a particular command fails, the subsequent commands will not be executed. Observe from the reply that the second command fails and the remaining commands are not executed.

Refer to the sections Ethernet ASCII Bulk Command/Reply Messages (Intermittent).

3. In the ASCII field, enter the command **LASpeed\00ABegin\00BSpeed\00**, and click Send.

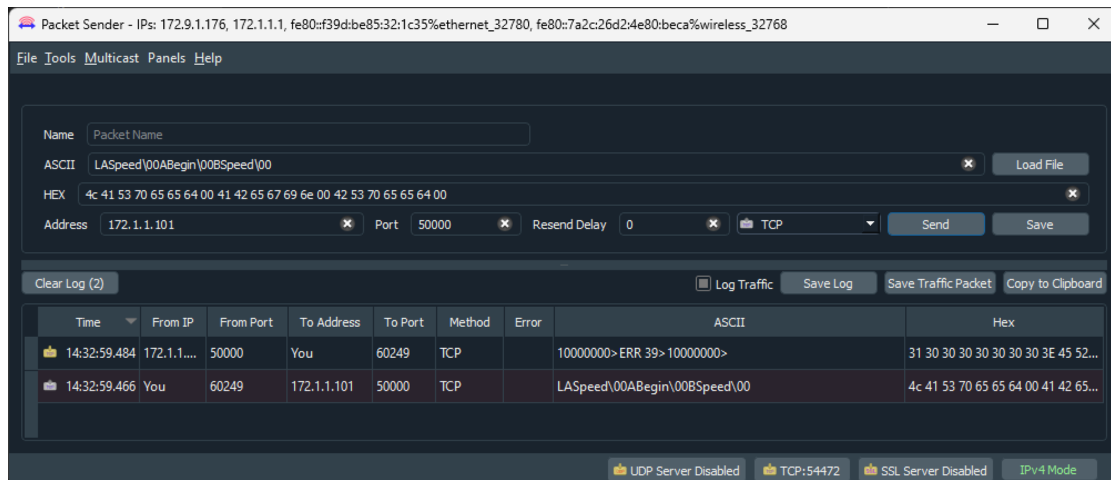

Figure 20.

The prefix character **L** indicates that the message is ASCII Lengthy type. This type of message supports the concatenation of commands which are executed sequentially. Even if a particular command fails, subsequent commands will still be executed.

Refer to the sections Ethernet ASCII Bulk Command/Reply Messages (Lengthy).

6.4.3 Sending binary commands over Ethernet communication channel

1. In the HEX field, enter the command **00 04 8A**, and click Send.

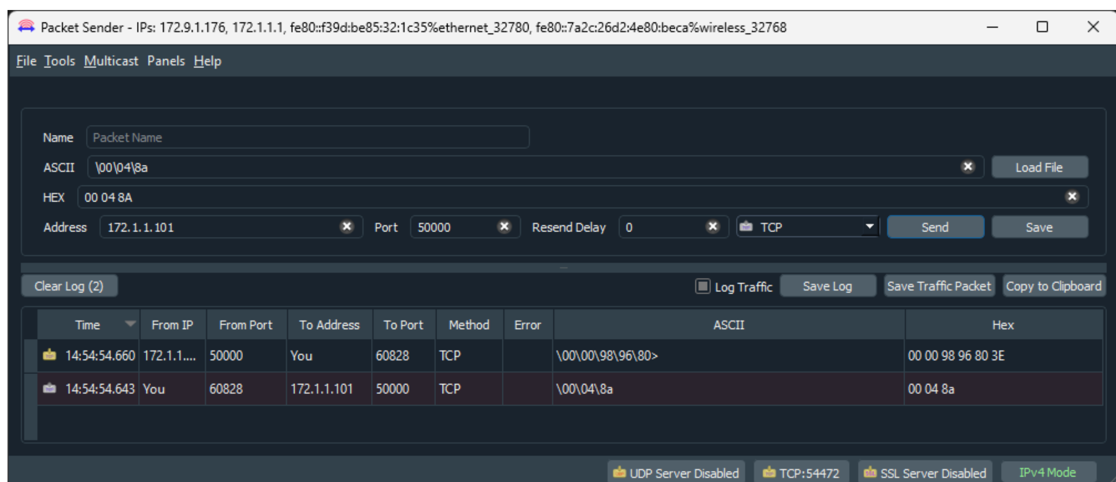

Figure 21.

Observe that the message size is much shorter compared to the ASCII syntax. The first byte 0x00 indicates the message is binary standard type. The remaining two bytes represent the Complex CAN Code for **ASpeed**.

Refer to the section Ethernet Binary Command/Reply Messages (Type 0 - Standard).

2. In the HEX field, enter the command **02 02 04 8A 04 00 05 00 02**, and click Send.

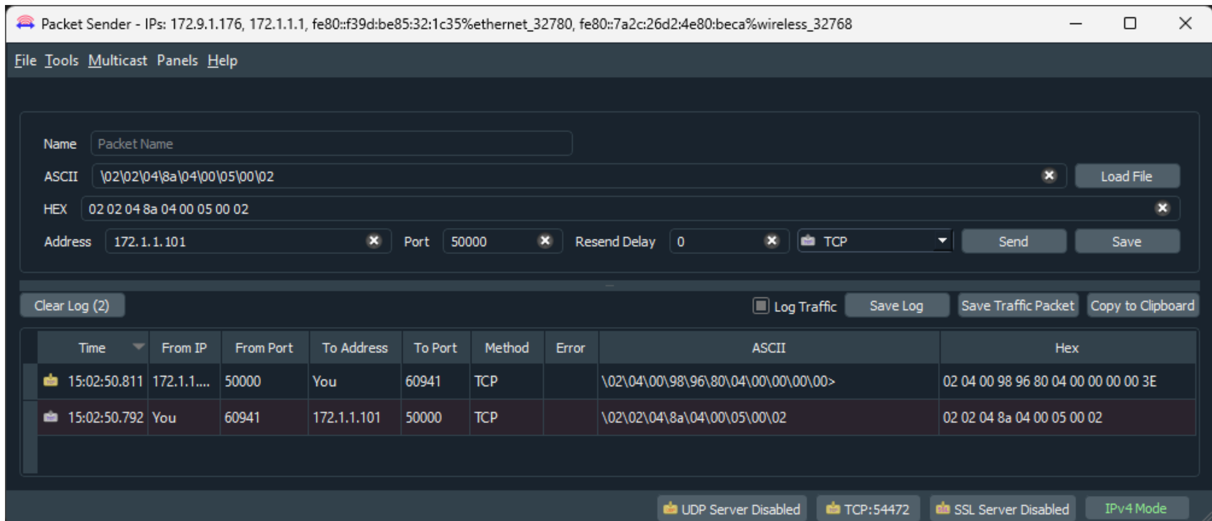

Figure 22.

The first byte 0x02 indicates that the message is binary bulk type. The next byte 0x02 represents the command size, followed by the command 0x04 8A (ASpeed). It is followed by the command size 0x04, followed by the command 0x00 05 00 02 (AVel[2]).

Refer to the section Ethernet Binary Command/Reply Messages (Type 2 - Bulk).