

ASI CAN Setup, Configuration and Communication for BAC Controllers

Created by: Nick Oudyk

Based on ASI CANOpen Communication Protocol & Implementation (v1.1.0 2018/01/23) documentation by Evin Ballantyne and Terry Stone, and CAN setup and feature documentation by Muhammad Ikhlas.

This document is confidential and may not be reproduced, transmitted or stored without the express written permission of Accelerated Systems Inc.

Contents

ASI CAN Setup, Configuration and Communication for BAC Controllers	0
Contents	1
Document Revision	3
Overview	4
Definitions	4
CAN Settings.....	5
Feature Descriptions	6
Extended 29 bit ID.....	6
CAN Faults	6
Enable Sleep on Sync Loss.....	6
Enable Auto-operational.....	7
Auto set CANOpen COB ID for TPDOs.....	7
Auto set CANOpen COB ID for RPDOs.....	7
Enable CAN.....	7
Disable CAN termination resistor	7
Enable Auto CAN ID adjustments.....	7
Enable RTR	7
ASI CANOpen Implementation.....	8
Implementation Overview	8
CANOpen Supported Communication Calls.....	8
CANOpen Message Format.....	8
Message Prioritization	9
Endianness	9
Service Data Objects (SDO)	10
SDO-Expedited Read	11
SDO-Expedited Write	12
SDO-Expedited Error Responses	13
Non-Data Communication Protocols	14
Network Management Protocol (NMT).....	14

SYNC Message.....	14
Heartbeat Protocol	15
Emergency Protocol (EMCY)	15
Process Data Objects (PDO)	19
Transmit PDO (TPDO).....	19
Receive PDO (RPDO)	19
CAN Customization and Setup	20
General Settings.....	20
CAN Sync Loss Timeout.....	20
CAN Heartbeat Period.....	20
CAN quiet after sleep command.....	20
Adjusting Transmit PDO (TPDO).....	20
Adjusting Receive PDO (RPDO)	23
Step-by-step guides and Examples	25
Setting up the CAN Parameters	25
TPDO Configuration	26
RPDO Configuration	29
Custom Addresses.....	31
Battery RPDO Setup	32
Display TPDO Setup.....	33
ASI CAN Object Dictionary	34
CAN Object Dictionary Blocks	34

Document Revision

Version	Date	Author	Change Description
1.0.0	4/26/2021	Nick Oudyk	Initial Release
1.0.1	8/3/2021	Nick Oudyk	Updates based on feedback from P. Desrosiers, and updates to CAN emergency messages
1.0.11	12/10/2021	Martin Kunze	Spelling / formatting

Overview

This document covers ASI's CAN communication protocol, based on CANOpen, as well as the various customization features and extra options available to BAC Controllers with CAN enabled. This document is intended for BAC Firmware versions 6.020 or greater.

This document is intended for customers to use when configuring the controller for communicating with various peripherals or building peripherals that communicate with the controller. An ASI Object Dictionary should be supplied with this document and used as a reference.

Definitions

The following definitions are used throughout the document.

Term	Description
CAN	Controller Area Network
Bus	The transmission line carrying the signal (bits) between all connected devices.
CANOpen	The default protocol used to communicate with the CAN Device
Bit	0 or 1
Byte	8 Sequential bits
Word	2 Sequential Bytes
Message	A series of bytes with an identifying header
Header	Metadata identifying a message
Object Dictionary	List of parameters you can access in the CAN device
Object Dictionary Index	An address for a block of objects in the Object Dictionary
Object Dictionary Sub Index	An address within a block of objects in the Object Dictionary
Dictionary Object	A single value in the dictionary.
Node	A device on the CAN network
Node Id	A unique id given to the node to identify itself on the network.
COB-ID	Can Object Identifier used in the header to uniquely identify the type of message. Often used in conjunction with the Node Id.
Client	The node making the request in a transmit/receive protocol
Server	The node making the response in a transmit/receive protocol
Little Endian	Byte sequence in multibyte variables is interpreted from Least Significant Byte to Most Significant Byte

CAN Settings

The following functions are standard settings required for CANOpen communication. Most of ASI's settings are configurable on a per-device basis.

Function	Description
Baud rate	1000Kb/s, 500Kb/s, 250Kb/s, or 125Kb/s. Other non-standard baud rates can be configured upon customer request.
Node ID	Ranging from 0x01 to 0x7F – Default of 0x2A (42)
CAN Heartbeat Period (ms)	Ranging from 0x0010 to 0x7FFF (or 0x0 to disable)
CAN Sync Loss Timeout (ms)	Ranging from 0x0001 to 0x7FFF (or 0x0 to disable)

In addition, our controller firmware supports additional customization features controlled by the following settings:

Function	Description
Communications Configuration Vector	16 bit communications vector that allows for enabling/disabling various features.
CAN quiet after sleep (ms)	Ranging from 0x0001 to 0x7FFF (or 0x0 to disable)
CAN quit after sync loss (ms)	Ranging from 0x0001 to 0x7FFF (or 0x0 to disable)

Communications Configuration Vector	
Bit	Feature – 1=enabled, 0=disabled
0	Extended 29bit ID
1	CAN Faults
2	Bit utilized for other communications (does not affect CAN configuration)
3	Enable sleep on sync loss
4	Enable auto operational
5	Auto set CANOpen COB ID for TPDO's
6	Auto set CANOpen COB ID for RPDO's
7	Enable CAN
8	Reserved
9	Reserved
10	Bit utilized for other communications (does not affect CAN configuration)
11	Bit utilized for other communications (does not affect CAN configuration)
12	Disable CAN termination resistor (Setting this bit disconnects the internal termination resistor from the CAN bus)
13	Enable Auto CAN ID adjustments
14	Enable RTR
15	Reserved

To set or modify any of these settings, connect to the controller using BACDoor™. Select the Communications Tab, and then the "Serial and CAN Communications Settings" Inner tab. Change the corresponding parameter in the Write/Save box, hit "Enter" on the keyboard after every parameter entry to confirm the parameter is sent to the controller. Then from the Parameter drop-down menu, click Save to Flash. After saving to flash, turn off the controller. Reboot and reconnect to the controller and verify the parameters are updated. If not, repeat the above steps.

Alternatively, each parameter can be accessed using an SDO read/write to the parameter's CANOpen index and subindex based on its dictionary address. Converting between dictionary address and CANOpen indices will be described later in this document.

Peripheral Selection Peripheral Configuration Protection and Fault Thresholds **Communications** Controller Debug Custom Tab

CAN RPDO 11,12 CAN TPDO1,2 CAN TPDO 3,4 CAN TPDO 5,6

Serial and CAN Communications Settings CAN RPDO1,2 CAN RPDO 3,4

Write/Save

Name	Value	Units	Scale	Modbus Address	CAN Address	Access Level
Display protocol	0		enum	66	0x2001(03)	0
communications configuration ve	0000000000000000		bit vect	58	0x2000(3B)	0
CAN sync loss timeout	0	ms	1	255	0x2003(40)	0
CAN quiet after sleep command	0	ms	1	1864	0x201D(00)	0
CAN quiet after sync loss	0	ms				
CAN heartbeat period	0	ms	1	1866	0x201D(0B)	0
CAN ID	0		1	57	0x2000(3A)	0
CAN baud rate	0	kbps	1	56	0x2000(39)	0
Slave ID	0	Node #	1	5	0x2000(06)	0
Slave ID port2	0	Node #	1	65	0x2001(02)	0
Baud rate	0	Baud	0.01	4	0x2000(05)	0
Baud rate port2	0	Baud	0.01	64	0x2001(01)	0
Command timeout threshold	0	ms	1	32	0x2000(21)	0
Average Command timeout thresh	0	ms	1	49	0x2000(32)	0

[Standard CAN Parameter settings in BACDoor™](#)

Feature Descriptions

Extended 29 bit ID

While CANOpen address space is limited to 11 bits (0x000-0x7FF), if this feature is enabled, it converts all CANOpen addresses to the corresponding 29 bit extended ID for transmission and reception.

E.g. Address 0x034 would be converted to 0x00000034

CAN Faults

This bit enables/disables generating faults and warnings based on CAN communication errors. If the device goes into Error Passive CAN state, the CAN Fault bit will be set, and if the Error Warning CAN State is entered, the CAN Warning bit will be set. The Warning bit will be cleared automatically upon exiting the CAN State, but the fault bit requires an explicit, clear fault request.

Enable Sleep on Sync Loss

This bit enables sleeping the controller for the duration saved in the "CAN quiet after sync loss" command on a sync timeout, before resetting the controller. If this bit is disabled, the controller will simply fault after a sync timeout, and stay in faulted state until the controller is power cycled or a fault clear request is received.

Enable Auto-operational

Enabling this bit tells the controller to automatically move from pre-operational to operational network status upon boot without waiting for a sync pulse.

Auto set CANOpen COB ID for TPDOs

Automatically configures each TPDO COBID to the corresponding CANOpen COB ID based on the TPDO number and saved device Node ID. **Only valid for TPDOs 1-4, as CANOpen does not support over 4 TPDOs.**

Auto set CANOpen COB ID for RPDOs

Automatically configures each RPDO COBID to the corresponding CANOpen COB ID based on the RPDO number and saved device Node ID. **Only valid for RPDOs 1-4, as CANOpen does not support over 4 RPDOs.**

Enable CAN

Enable CAN Communication on BAC Controllers. **Required to be set for any CAN communication.**

Disable CAN termination resistor

Set this bit to disable the built-in termination resistor on the CAN network. Use if another Node will be terminating the bus.

Enable Auto CAN ID adjustments

This feature allows modifying the CAN Node ID on boot based on digital inputs, allowing two controllers with the same parameter file and firmware to have different CAN Node IDs. Thus, being able to operate on the same bus. If this feature is enabled, the CAN Node ID saved in parameter 57 will be incremented by 1 if the Cruise flag is high on boot, incremented by 2 if the PFS flag is high, and incremented by 3 if both Cruise and PFS flags are high on boot. This Node ID will be used in all standard CANOpen communications and can be validated by reading parameter 1676.

Enable RTR

This flag enables RTR functionality in the controller firmware. If this feature is enabled, custom TPDO's can be configured to automatically transmit upon a remote frame request at the TPDO address instead of on a sync frame or at a time interval.

ASI CANOpen Implementation

Implementation Overview

ASI uses the standard CANOpen protocol (see <https://en.wikipedia.org/wiki/CANOpen>) to communicate over a CAN network, with a few small exceptions allowing customization of features.

To allow for other communication methods, such as serial or a Bluetooth connection to control and configure parameter settings, ASI maps its custom Modbus dictionary addresses to CANOpen indices in the 0x2000 to 0x2024 range. The specific conversion method between addresses listed in the ASI Object Dictionary XML file is as follows:

Index	Sub-Index
FLOOR (Modbus address / 64) + 8192	REMAINDER(Modbus address / 64) + 1

E.g. Communications configuration vector at Modbus address 58 corresponds to CANOpen index 8192 (0x2000), with subindex 59 (0x3B). TPDO1 size(words) is Modbus address 1824, which corresponds to CANOpen index 8220 (0x201C), with subindex 33 (0x21).

In addition, ASI controllers support manually customizing TPDO and RPDO addresses. They can send and receive groups of parameters at specific addresses to integrate with other communication protocols.

CANOpen Supported Communication Calls

The following are messaging protocols that ASI CAN-enabled devices support. More detail on these protocols and their implementations are given in later sections.

Function	Description
NMT	Network Management Protocols
SDO	Service Data Objects
PDO	Process Data Objects
SYNC	Synchronization Object
EMCY	Emergency Objects

CANOpen Message Format

The following is the typical representation for all CANOpen messages, transmitting or receiving.

Bits 0 to 10	Bit 11	Bits 12 to 15	Variable Length (0 to 8 Bytes)
COB-ID	Rtr bit (generally not used)	Data length	Data

Message Prioritization

If multiple messages are sent on the bus simultaneously, the message with the lowest COB-ID takes priority.

Endianness

CANOpen stores and transfers its data bytes in little-endian order. When reading this document, any variable that requires more than one byte should be interpreted as little-endian.

For example: Consider the following table.

Byte ...	Byte 1	Byte 2	Byte ...
...	Index		...

If we have an Index of 0x1122, **byte 1** would be 0x22, and **byte 2** would be 0x11 because it is stored as little endian.

Service Data Objects (SDO)

SDO is a basic transmit & response protocol. Used for one-time read & writes to Data Objects in the Object Dictionary. All Dictionary objects are four bytes or less in length (typically two bytes), so only SDO-Expedited Messages are necessary. SDO objects can be transmitted while the node is in a pre-operational state or an operational state (see NMT commands).

Item	Description
CAN Header	Contains CAN Network Instruction & Node ID
Rtr	Remote Transmit bit
Length	Data Length
Command	Server Command
Index	Dictionary Object Block location
Sub Index	Dictionary Object Sub-Block Location
Data	Transmitted Data

SDO-Expedited Read

The following is the message format to read from the CAN Object Dictionary.

Client Transmit Message:

CAN Header	Rtr	Length	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x600 + Node Id	0	8	Command	Index		Sub Index	0			

Valid Commands:

Command Code	Meaning
0x40	Read Dictionary Object.

Server Response Message:

CAN Header	Rtr	Length	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x580 + Node Id	0	8	Command	Index		Sub Index	Data			

Valid Response Commands:

Command Code	Meaning
0x43	Read Dictionary Expedited Response. 4 bytes returned.
0x47	Read Dictionary Expedited Response. 3 bytes returned.
0x4B	Read Dictionary Expedited Response. 2 bytes returned.
0x4F	Read Dictionary Expedited Response. 1 byte returned.

SDO-Expedited Write

The following is the message format to write to the CAN Object Dictionary.

Client Transmit Message:

CAN Header	Rtr	Length	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x600 + Node Id	0	8	Command	Index		Sub Index	Data			

Valid Commands:

Command Code	Meaning
0x23	Write Dictionary Expedited Request. 4 bytes to write.
0x27	Write Dictionary Expedited Request. 3 bytes to write.
0x2B	Write Dictionary Expedited Request. 2 bytes to write.
0x2F	Write Dictionary Expedited Request. 1 byte to write.

Server Response Message:

CAN Header	Rtr	Length	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x580 + Node Id	0	8	Command	Index		Sub Index	Data			

Valid Response Commands:

Command Code	Meaning
0x60	Write Complete.

SDO-Expedited Error Responses

If something should go wrong during an SDO Read or write, the server-side will respond with the following message.

Server Response Message:

CAN Header	Rtr	Length	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x580 + Node Id	0	8	0x80	Index		Sub Index	Error Code			

Valid Error Codes:

Error Code	Meaning
0x05040001	Client/Server command specifier not valid or unknown.
0x06010000	Unsupported access to an object.
0x06020000	The object does not exist in the object dictionary.
0x06070010	Data type does not match length. Length of Service Parameter does not match.
0x06090011	Sub Index does not exist.

Non-Data Communication Protocols

The following protocols and message formats are supported.

Network Management Protocol (NMT)

The following message will set the operational state of the node. The node's operational state can be viewed in the heartbeat message.

Client Transmit Message:

CAN Header	Rtr	Length	Byte 0	Byte 1
0x000	0	2	Command	Node Id

Valid Commands:

Command Code	Meaning
0x01	Enter Operational State
0x02	Enter Stopped State
0x80	Enter Pre-Operational State
0x81	Reset Node
0x82	Reset Communication

SYNC Message

The Sync message is used to keep nodes in an operational state and tell nodes to process or transmit their PDOs. If the SYNC message is not broadcast before a Node's SYNC Loss timeout setting, the node will go from an operational state to a Pre-operational state.

Client Transmit Message:

CAN Header	Rtr	Length	Bytes
0x080	0	0	(None)

Heartbeat Protocol

Usually, a node will periodically send out a message while operational to signify it still exists on the network. It is not required that a node does this.

Server Transmit Message:

CAN Header	Rtr	Length	Byte 0
0x700 + Node Id	0	1	State

Valid States:

State Code	Meaning
0x00	Boot up / Initializing
0x04	Stopped
0x05	Operational
0x7f	Pre-Operational

Emergency Protocol (EMCY)

Whenever there is a change in the Faults, Faults2, or Warnings, the node will broadcast the following message. The controller will only broadcast this message once per change.

Server Transmit Message (Type 1):

CAN Header	Rtr	Length	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x080 + Node Id	0	8	Emergency Code (Always 0x1000)		Error Register	Faults 2(Low Byte)	Faults 1		Warnings	

Server Transmit Message (Type 2):

CAN Header	Rtr	Length	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x080 + Node Id	0	8	Emergency Code (Always 0x2000)		Error Register	Faults 2(High Byte)	Warnings2		Undefined	

Valid Emergency Codes:

Emergency Codes	Meaning
0x1000	Manufacturer Defined Error Codes – Message Type 1

0x2000	Manufacturer Defined Error Codes – Message Type 2
--------	---

Error Register Bit Structure:

Error Register Bits	Meaning
0	Generic Error
1	Current
2	Voltage
3	Temperature
4	Communication Error
5	Device Profile Specific
6	Peripheral Error
7	Manufacturer Specific

Faults 2 Low byte Bit Structure:

Faults 2 Bits	Meaning
0	Parameter CRC
1	Current Scaling
2	Voltage Scaling
3	Headlight Under Voltage
4	Parameter3 CRC
5	CAN Bus
6	Hall Stall
7	Bootloader

Faults 1 Bit Structure:

Faults 1 Bits	Meaning
0	Controller Over Voltage
1	Phase Over Current
2	Current Sensor Calibration
3	Current Sensor Over Current
4	Controller Over Temperature

5	Motor Hall Sensor Fault
6	Controller Under Voltage
7	POST Static Gating Test
8	Network Communication Timeout
9	Instantaneous Phase Over Current
10	Motor Over Temperature
11	Throttle Voltage Outside Range
12	Instantaneous Controller Over Voltage
13	Internal Error
14	POST Dynamic Gating Test
15	Instantaneous Under Voltage

Warnings Bit Structure:

Warnings Bit	Meaning
0	Communication Timeout
1	Hall Sensor
2	Hall Stall
3	Wheel Speed Sensor
4	CAN Bus
5	Hall Illegal Sector
6	Hall Illegal Transition
7	Vdc Low Foldback
8	Vdc High Foldback
9	Motor Temperature Foldback
10	Control Temperature Foldback
11	Low SOC Foldback
12	Hi SOC Foldback
13	I2t Foldback
14	<i>Low Temp Battery/Controller Foldback</i>
15	BMS timeout

Faults 2 High byte Bit Structure:

Faults 2 (High Byte) Bits	Meaning
0	Parameter2 CRC
1	Motor position fault
2	Dyname torque sensor voltage outside range
3	Dyname torque sensor static voltage fault
4	Remote CAN fault
5	Accelerometer Side tilt fault
6	Open phase fault
7	Reserved for future use

Warnings2 Bit Structure:

Warnings Bit	Meaning
0	Throttle out of range
1	Dual speed sensor missing pulses
2	Dual speed sensor no pulses
3	Dynamic Flash Full
4	Dynamic Flash Read Error
5	Reserved for future use
6	Reserved for future use
7	Reserved for future use
8	Reserved for future use
9	Reserved for future use
10	Reserved for future use
11	Reserved for future use
12	Reserved for future use
13	Reserved for future use
14	Reserved for future use
15	Reserved for future use

Process Data Objects (PDO)

Process Data objects are an efficient means of transferring dictionary objects between different nodes. PDOs can only be transmitted and received while the node is in operational mode (see NMT commands).

Transmit PDO (TPDO)

A node will transmit a TPDO depending on its settings. The message structure is as follows:

Node Transmit Message:

CAN Header	Rtr	Length	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
TPDO Id	0	TPDO Length	TPDO Data 1 (If set)		TPDO Data 2 (If set)		TPDO Data 3 (If set)		TPDO Data 4 (If set)	

TPDO Id: Standard CANOpen TPDO Id's follow the pattern $0x[X]80 + \text{Node id}$, where **[X]** is the TPDO number that is being transmitted. For example: if the Node ID of the controller is $0x2a$, and it is broadcasting its second TPDO, the TPDO Id will be $0x2AA$. These IDs will be used when the "Auto set TPDO COB-ID" feature is enabled. In addition, customizing the TPDO address to transmit to a specific, non-standard address is possible; this will be discussed later.

TPDO Length: Any length between 2 to 8 bytes.

TPDO Data: The Data can represent anywhere between one to four dictionary objects, depending on how the TPDO has been mapped.

Receive PDO (RPDO)

Anytime a node receives a message that matches one of its assigned RPDOs, it will write the data values of that message to the mapped dictionary objects of that RPDO. The message structure the node will receive is as follows:

Node Receive Message:

CAN Header	Rtr	Length	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
RPDO Id	0	RPDO Length	RPDO Data 1 (If set)		RPDO Data 2 (If set)		RPDO Data 3 (If set)		RPDO Data 4 (If set)	

RPDO Id: Standard CANOpen RPDO Id's follow the pattern $0x[X+1]00 + \text{Node id}$, where **[X]** is the RPDO number that is being transmitted. For example: if the Node ID of the controller is $0x2a$, and it is broadcasting its second RPDO, the RPDO Id will be $0x3AA$. These are the IDs that will be used when the "Auto set RPDO COB-ID" feature is enabled. In addition, customizing the RPDO address to receive a specific, non-standard address is possible; this will be discussed later.

RPDO Length: Any length between 2 to 8 bytes.

RPDO Data: The Data can be written to one to four dictionary objects, depending on how the RPDO has been mapped.

CAN Customization and Setup

ASI's implementation of adjusting the PDOs is done indirectly through the object dictionary. While this is not entirely in line with CANOpen, it means that the PDOs can be changed in BacDoor™ via TTL, RS485, or Bluetooth LE, in addition to CAN. The following sections list the name of the parameter, as well as expected values. You can find these in PC BacDoor™ under the "Communications" tab. Up to 12 TPDOs and 12 RPDOs are available to configure. However, TPDOs and RPDOs after the 4 will require custom addressing, as they are not supported in the CANOpen specification.

General Settings

CAN Sync Loss Timeout

The CAN Sync Loss Timeout command controls the maximum amount of time allowed between sync messages received by the controller before the controller generates a fault condition. If the "Enable sleep on sync loss" feature bit is enabled, after faulting due to sync loss, the controller will sleep for an amount of time determined by the "CAN quiet after sync loss" parameter before resetting the controller. The valid range for this setting is 0x10 to 0x7FFF milliseconds, or 0 to disable this feature.

CAN Heartbeat Period

This parameter controls the interval at which the controller transmits the heartbeat messages described earlier in this document. The valid range for this setting is 0x10 to 0x7FFF milliseconds, or 0 to disable transmitting heartbeat messages.

CAN quiet after sleep command

This parameter controls the duration for which the controller will sleep before resetting after receiving a sleep command over the CAN network. The valid range for this setting is 0x0 to 0xFFFF ms; a value of 0 will cause the controller to immediately reset upon receiving a sleep command.

Adjusting Transmit PDO (TPDO)

The following are the parameters used to adjust a TPDO. All objects are two bytes in size.

TPDO COBID (high word): This parameter is used to set control bits for the TPDO if the "Autoset TPDO COBID" feature is disabled.

Bit Number	Control Bit	Description
15	PDO Disable	Enabling this bit (1) will disable the PDO.
14	RTR Enable	Enabling this bit will tell the TPDO to transmit upon receiving a RTR frame instead of using the Event Time interval or Sync Frame. The Event Time interval or Sync Frame count will be used to determine the RTR data update rate.
13	Extended ID Set	Configures the TPDO address to use extended mode, setting the COBID low word to the last 11 bits of the extended address.
12-0	Unused	Unused bits

TPDO COBID (low word): This parameter is used to control the broadcast address of the TPDO directly, without offset by the Node ID parameter. When using this parameter to customize the TPDO broadcast address, care must be taken to ensure no network conflicts occur, as this does not follow CANOpen spec. Valid values for this parameter are 0x0001 to 0x07FF.

TPDO Size (words): Number of objects being broadcast.

TPDO Transmission Type:

Value	Meaning
0	Asynchronous: The TPDO will only transmit based on an event time.
1...240	Cyclic Synchronous: TPDO will transmit on a SYNC Message. If value is set higher than one, TPDO will wait that many SYNC messages before transmitting. Example: if the value is set for 4, the TPDO will transmit every fourth SYNC message.
241...244	<i>(Reserved: Not Used)</i>

TPDO Event Time: The time between an asynchronous TPDO transmission in milliseconds. Minimum time is 10ms and maximum time is 32767ms. To disable asynchronous TPDO transmission, set this parameter to 0.

Each TPDO can broadcast up to 4 Dictionary objects.

TPDO Map Index:

Byte 0	Byte 1
Dictionary Index	

TPDO Map Sub Index & Size:

Byte 0	Byte 1
Dictionary Sub Index	Object Size (in bits)

BacDoor

File Parameter Scope Help

Data Datalog

Peripheral Selection Peripheral Configuration Protection and Fault Thresholds **Communications** Controller Debug Custom Tab

Serial and CAN Communications Settings

CAN RPDO 1,2	CAN RPDO 3,4	CAN RPDO 5,6	CAN RPDO 7,8	CAN RPDO 9,10
CAN RPDO 11,12	CAN RPDO 1,2	CAN TPDO 3,4	CAN TPDO 5,6	CAN TPDO 7,8
				CAN TPDO 9,10
				CAN TPDO 11,12

Write/Save

Name	Value	Units	Scale	Modbus Address	CAN Address	Access Lev
TPDO1 COBID (high word)	2000	hex	hex	1873	0x201D(12)	
TPDO1 COBID (low word)	0182	hex	hex	1872	0x201D(11)	
TPDO1 size (words)	4		1	1824	0x201C(21)	
TPDO1 transmission type	0		1	1825	0x201C(22)	
TPDO1 event time	500	ms	1	1826	0x201C(23)	
TPDO1 map1 index	201A	hex	hex	1817	0x201C(1A)	
TPDO1 map1 sub index and size	1B10	hex	hex	1816	0x201C(19)	
TPDO1 map2 index	201A	hex	hex	1819	0x201C(1C)	
TPDO1 map2 sub index and size	1C10	hex	hex	1818	0x201C(1B)	
TPDO1 map3 index	201A	hex	hex	1821	0x201C(1E)	
TPDO1 map3 sub index and size	1D10	hex	hex	1820	0x201C(1D)	
TPDO1 map4 index	201A	hex	hex	1823	0x201C(20)	
TPDO1 map4 sub index and size	1E10	hex	hex	1822	0x201C(1F)	
TPDO2 COBID (high word)	2000	hex	hex	1875	0x201D(14)	
TPDO2 COBID (low word)	018A	hex	hex	1874	0x201D(13)	
TPDO2 size (words)	4		1	1836	0x201C(2D)	

Controller:

Read Write

Display Only

Name	Value
faults	000000000000
vars2_1690	
vars2_1691	
vars2_1692	
vars2_1693	
vars2_1694	
vars2_1695	
vars2_1696	
vars2_1697	
vars2_1698	
vars2_1699	
vars2_1700	
vars2_1701	

Controller:

Read Poll

Write (Temp)

Network

Port: COM9

Baud: 115200

Address: 1

Connect ☐ Connected

Activity: ☐ Rx ☐ Tx

Checksum Errors: 0

Timeout Errors: 3

Status

Reset

Read

Name	Value
user access level	0
Test build	0
software revision	0
motor input pow	0
battery power	0
motor current	0
motor speed	0
motor rpm	0
vehicle speed	0

In the above picture, TPDO1 has 4 mapped objects, all with the 0x201A index, sub-indices of 0x1B, 0x1C, 0x1D and 0x1E, and sizes of 0x10 or 16 bits (1 word) each.

This corresponds to the Modbus address' of $(0x201A - 0x2000) * 64 + (\text{sub-index} - 1)$, or 1690, 1691, 1692 and 1693.

Adjusting Receive PDO (RPDO)

The following are the Dictionary objects used to adjust an RPDO. All objects are two bytes in size.

RPDO COBID (high word): This parameter is used to set control bits for the RPDO if the “Autoset RPDO COBID” feature is disabled.

Bit Number	Control Bit	Description
15	PDO Disable	Enabling this bit (1) will disable the PDO.
14	Reserved	Bit reserved for future use.
13	Extended ID Set	Configures the RPDO address to use extended mode, setting the COBID low word to the last 11 bits of the extended address.
12-0	Unused	Unused bits

RPDO COBID (low word): This parameter is used to control the receiving address of the RPDO directly, without offset by the Node ID parameter. When using this parameter to customize the RPDO receive address, care must be taken to ensure no network conflicts occur, as this does not follow CANOpen spec. Valid values for this parameter are 0x0001 to 0x07FF.

RPDO Size: Number of objects being received.

RPDO Transmission Type:

Value	Meaning
0	RPDO Disabled
1...256	Asynchronous: The RPDO is processed as soon as the message is received.

RPDO Timeout: Parameter reserved for future use.

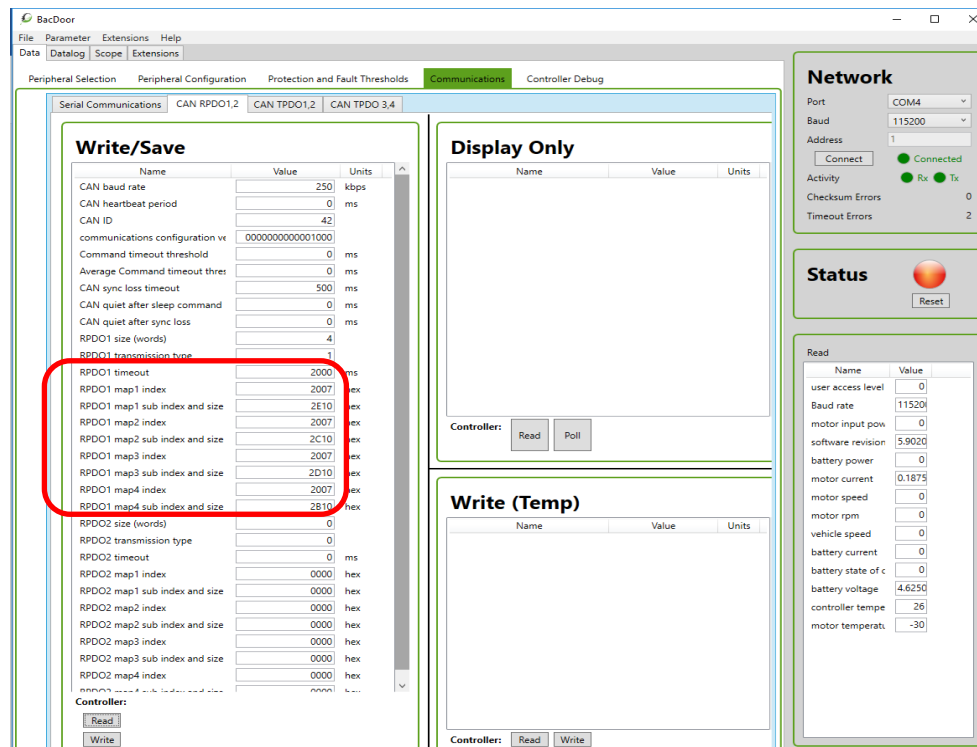
Each RPDO can receive up to 4 Dictionary objects.

RPDO Map Index:

Byte 0	Byte 1
Dictionary Index	

RPDO Map Sub Index & Size:

Byte 0	Byte 1
Dictionary Sub Index	Object Size (in bits)



The screenshot shows the BacDoor software interface. The 'Write/Save' tab is active, displaying a list of parameters. A red box highlights the 'RPDO1 map1 index' and 'RPDO1 map1 sub index and size' entries, which are set to 2007 and 2E10 respectively. The 'Display Only' tab is also visible, showing a list of parameters. The 'Network' and 'Status' panels are on the right side of the interface.

In this example picture, you can see the RPDO mappings use the 0x2007 index and are using 2B-2E subindices with a 1 word size.

These mappings correspond to Modbus addresses 490-493, which control the Remote speed command, Remote maximum motoring current, Remote maximum braking current and Remote state command parameters.

Step-by-step guides and Examples

Note: For the following examples, if a tab or parameter is missing from BACDoor™, due to an outdated application layout file, the user can right click in any Write/Save window to add any missing parameters. If the parameter is not present in the “Parameter Selection” options, contact ASI for an updated Object Dictionary file.

Setting up the CAN Parameters

1. Open BACDoor™ and Connect to the controller through a serial port.
2. Open up the “Communications” tab, and the “Settings” sub-tab.
3. Set the “CAN ID” parameter to 42, the “CAN baud rate” parameter to 250, and ensure that the “CAN syn loss timeout” and “CAN heartbeat period” parameters are set to 0.
4. In the “Communications Configuration Vector”, set bits 4, 5, 6 and 7 and confirm all other bits are cleared. This will enable CAN communications on the controller, and automatically configure addresses for TPDOs and RPDOs used in later examples.
5. Saving changes:
 - 5.1. Click “Write” at the bottom of the Write/Save section, and then click “Read” to confirm that all of the above changes have been sent to the controller.
 - 5.2. Then click the “Save to Flash” option from the Parameter menu at the top left of the application.
6. Wait for the parameters to complete saving to flash, then power cycle the controller. After power cycling the controller, you should be able to connect to the controller using the CAN version of BACDoor™ with a CAN-to-USB adapter, such as a PeakCAN dongle, by selecting the “CAN” port, a baud rate of 250, and an address of 42. Opening up an instance of PCAN-View will also allow you to observe the communication between the controller and BACDoor™.

Step 5.2: Saving parameters to flash.

Write/Save

Name	Value	Units	Scale	Modbus Address	CAN Address	Access Level
CAN Auto ID	0		1	1676	0x201A(0D)	0
Communications Configuration W	0000000011110000	bit vect		58	0x2000(3B)	0
CAN sync loss timeout	0	ms	1	255	0x2003(40)	0
CAN quiet after sleep command	0	ms	1	1864	0x201D(09)	0
CAN quiet after sync loss	0	ms	1	1865	0x201D(0A)	0
CAN heartbeat period	0	ms	1	1866	0x201D(0B)	0
CAN ID	42		1	57	0x2000(3A)	0
CAN baud rate	250	kbps	1	56	0x2000(39)	0
Slave ID	2	Node #	1	5	0x2000(0B)	0
Slave ID port2	2	Node #	1	65	0x2001(02)	0
Baud rate	115200	Baud	0.01	4	0x2000(05)	0
Baud rate port2	115200	Baud	0.01	64	0x2001(01)	0
Command timeout threshold	1000	ms	1	32	0x2000(21)	0
Average Command timeout thresh	250	ms	1	49	0x2000(32)	0

Display Only

Name	Value	Units	SI
modbus1 CRC error count	0	counts	
modbus1 HW errors	0	counts	
modbus2 CRC error count	0	counts	

Write (Temp)

Name	Value	Units	SI
------	-------	-------	----

Network

Port: COM12
Baud: 115200
Address: 1
Connect: ☒ Connected
Activity: ☐ Rx ☐ Tx
Checksum Errors: 0
Timeout Errors: 0

Read

Name	Value
user access level	0
Test build	0
software revision	16.017
motor input pow	0
battery power	0
motor current	0
motor speed	0
motor rpm	0
vehicle speed	0
battery current	0
battery state of c	100
battery voltage	31.750
controller temper	22
motor temperat	0

Controller:

Write (Temp)

Step 5.1 - Make sure to write these parameters before saving to flash

PCAN-View

File CAN Edit Transmit View Trace Window Help

Receive / Transmit Trace PCAN-USB

CAN-ID	Type	Length	Data	Cycle Time	Count
62Ah		8	40 04 20 2C 00 00 00 00	2.8	980
5AAh		8	4B 04 20 2C 00 00 00 00	2.0	980

Step 6: Confirm that the controller is communicating using CAN

Network

Port: CAN
Baud: 250
Address: 42
Connect: ☒ Connected
Activity: ☒ Rx ☒ Tx
Checksum Errors: 0
Timeout Errors: 0

Status

Read

Name	Value
------	-------

TPDO Configuration

- After connecting to the controller in BACDoor™, select the “Communications” tab, and the “CAN TPDO 1,2” subtab.
- Set the following parameters to configure the TPDO structure and timing:
 - TPDO1 size(words) to 4
 - TPDO1 transmissions type to “0” for asynchronous only.
 - TPDO1 event time to 100 (ms)
- Set the following parameters to configure the TPDO mapping:

- 3.1. TPDO1 map1 index to "2004", and TPDOmap1 subindex and size to "0A10"
- 3.2. TPDO1 map2 index to "2003", and TPDOmap2 subindex and size to "2410"
- 3.3. TPDO1 map3 index to "2003", and TPDOmap3 subindex and size to "2310"
- 3.4. TPDO1 map4 index to "2001", and TPDOmap4 subindex and size to "0710"
4. Save the parameter settings to flash, then power cycle the controller and observe the messages being sent over the CAN network.
 - 4.1. As the TPDO addresses are being automatically mapped, the TPDO1 CAN address should be $0x180 + \text{CAN Node ID}$, which is 42, or 0x2A. Thus, the messages being transmitted by the controller at a 100ms interval should have address 0x1AA. This can be observed by checking the TPDO1 COBID parameters in BACDoor™.
 - 4.2. The following information should be transmitted in the message based on the above configuration:
 - 4.2.1. Battery Voltage (x32)
 - 4.2.2. Wheel diameter (in mm)
 - 4.2.3. Gear ratio (x256)
 - 4.2.4. Rated System Voltage
 - 4.3. Open BACDoor™ and check the corresponding parameters and confirm that they match after converting the CAN data into decimal format and dividing by the scaling factor.

Write/Save

Should be automatically set due to setting the "Autoset CANopen COBID for TPDOs" flag in the first example.

Name	Value	Units	Scale	Modbus Address	CAN Address	Access
TPDO1 COBID (high word)	0000	hex	hex	1873	0x201D(12)	
TPDO1 COBID (low word)	01AA	hex	hex	1872	0x201D(11)	
TPDO1 size (words)	4		1	1824	0x201C(21)	
TPDO1 transmission type	0		1	1825	0x201C(22)	
TPDO1 event time	100	ms	1	1826	0x201C(23)	
TPDO1 map1 index	2004	hex	hex	1817	0x201C(1A)	
TPDO1 map1 sub index and size	0A10	hex			0x201C(19)	
TPDO1 map2 index	2003	hex			0x201C(1C)	
TPDO1 map2 sub index and size	2410	hex			0x201C(1B)	
TPDO1 map3 index	2003	hex	hex	1821	0x201C(1E)	
TPDO1 map3 sub index and size	2310	hex	hex	1820	0x201C(1D)	
TPDO1 map4 index	2001	hex	hex	1823	0x201C(20)	
TPDO1 map4 sub index and size	0710	hex	hex	1822	0x201C(1F)	

Steps 2 & 3: Configure the TPDO1 Parameters, then save to flash.

PCAN-View

File CAN Edit Transmit View Trace Window Help

Receive / Transmit Trace PCAN-USB

CAN-ID	Type	Length	Data	Cycle Time	Count
1AAh		8	F8 03 0A 00 00 01 24 00	100.2	473
62Ah					
5AAh					

TPDO1 Message, with ID 0x1AAh:

Map1 value: 0x03F8 (remember the data is little endian)

Map2 value: 0x000A

Map3 value: 0x0100

Map4 value: 0x0024

Display Only

Name	Value	Units	Scale	Modbus Address	CAN Address	Access Level
battery voltage	31.7500	Volts	32	265	0x2004(0A)	0
Wheel diameter	10	mm	1	227	0x2003(24)	0
Gear ratio	1	N:1	256	226	0x2003(23)	0
Rated system voltage	36	Volts	1	70	0x2001(07)	0

Observed values in BACDoor™ (get the decimal result multiplying by scale, and convert to hexadecimal value to compare with CAN message)

31.75V * Scale(32) = 1016 = 0x3F8 for map 1

10mm * Scale(1) = 10 = 0xA for map 2

1:1 ratio * Scale(256) = 256 = 0x100 for map 3

36V * Scale(1) = 36 = 0x24 for map 4

RPDO Configuration

1. After connecting to the controller in BACDoor™, select the “Communications” tab, and the “CAN RPDO 1,2” subtab.
2. Set the following parameters to configure the TPDO structure and timing:
 - 2.1. RPDO1 size(words) to 4
 - 2.2. RPDO1 transmission type to 1 to enable the RPDO.
3. Set the following parameters to configure the RPDO mapping:
 - 3.1. RPDO1 map1 index to “2007”, and TPDOmap1 subindex and size to “3010”
 - 3.2. RPDO1 map2 index to “2003”, and TPDOmap2 subindex and size to “2410”
 - 3.3. RPDO1 map3 index to “2003”, and TPDOmap3 subindex and size to “2310”
 - 3.4. RPDO1 map4 index to “2001”, and TPDOmap4 subindex and size to “0710”
4. Save the parameter settings to flash, then power cycle the controller.
 - 4.1. As the RPDO addresses are configured to be automatically mapped, the RPDO1 CAN address should be 0x200 + CAN Node ID, which is 42, or 0x2A. Thus, the controller will be monitoring the address 0x22A. This can be observed by checking the RPDO1 COBID parameters in BACDoor™.
 - 4.2. The following information should be updated on the controller when the corresponding RPDO message is received, based on the above configuration:
 - 4.2.1.Remote Throttle Voltage (x4096)
 - 4.2.2.Wheel diameter (in mm)
 - 4.2.3.Gear ratio (x256)
 - 4.2.4.Rated System Voltage
5. To test this configuration, we need to send a message to the controller matching the expected RPDO address.
 - 5.1. In PCAN-View, created a new transmit message, with the following settings:
 - 5.1.1.ID: 0x22A
 - 5.1.2.Length: 8
 - 5.1.3.Data Bytes:
 - 5.1.3.1. 0 – 0x00
 - 5.1.3.2. 1 – 0x30
 - 5.1.3.3. 2 – 0x58
 - 5.1.3.4. 3 – 0x02
 - 5.1.3.5. 4 – 0x66
 - 5.1.3.6. 5 – 0x02
 - 5.1.3.7. 6 – 0x30
 - 5.1.3.8. 7 – 0x00
 - 5.1.4. Make sure the Extended Frame, and Remote Request boxes are unchecked.
 - 5.2. Configure back door to observe the parameters that we are transmitting to.
 - 5.2.1.Right click any “Display Only” box and select “Add Parameter”.
 - 5.2.2.Add the “Remote Throttle Voltage” Parameter.
 - 5.2.3.Repeat the previous two steps for “Gear ratio”, “Rated system voltage” and “Wheel Diameter”.

5.2.4. Click the “Poll” button, which will tell BACDoor™ to keep the values in the display box up to date.

5.3. Transmit the message by double-clicking the CAN-ID of the message in PCAN-View.

5.4. Observe the results in BACDoor™.

5.4.1. Remote throttle voltage should change to 3 Volts ($0x3000 = 12288 / \text{Scale } (4096) = 3$)

5.4.2. Wheel diameter should change to 600 mm ($0x258 = 600 / \text{Scale } (1) = 600$)

5.4.3. Gear ratio should change to ~ 2.4 ($0x266 = 614 / \text{Scale } (256) = 2.3984$)

5.4.4. Rated system voltage should change to 48 Voltage ($0x30 = 48 / \text{Scale } (1) = 48$).

5.5. In addition, if you set up the previous TPDO example, you should see the last 6 bytes of the message change to match the last 6 bytes of the RPDO data packet, as those 3 mappings are targeting the same parameters.

Serial and CAN Communications Settings

CAN RPDO1,2

CAN RPDO 3,4

Write/Save

Name	Value	Units	Scale	Media Address	CAN Address	Access Level
Communications Configuration V	0000000011110000					0
RPDO1 COBID (high word)	0000					0
RPDO1 COBID (low word)	022A	hex	hex	1868	0x201D(0D)	0
RPDO1 size (words)	4		1	1800	0x201C(09)	0
RPDO1 transmission type	1		1	1801	0x201C(0A)	0
RPDO1 timeout	0	ms	1			0
RPDO1 map1 index	2007	hex	hex			0
RPDO1 map1 sub index and size	3010	hex	hex	1792	0x201C(01)	0
RPDO1 map2 index	2003	hex	hex	1795	0x201C(04)	0
RPDO1 map2 sub index and size	2410	hex	hex	1794	0x201C(03)	0
RPDO1 map3 index	2003	hex	hex	1797	0x201C(06)	0
RPDO1 map3 sub index and size	2310	hex	hex	1796	0x201C(05)	0
RPDO1 map4 index	2001	hex	hex	1799	0x201C(08)	0
RPDO1 map4 sub index and size	0710	hex	hex	1798	0x201C(07)	0

RPDO1 Address based on Autoset CANopen COBID feature

RPDO1 Parameter Configuration

New Transmit Message

ID: (hex) 22A Length: 8 Data: (hex) 00 30 58 02 66 02 30 00 ...

Cycle Time: 0 ms

Message Type

☐ Extended Frame

☐ Remote Request

Comment: RPDO1 Example Message

OK Cancel ? Help

Message to send that will update the parameters defined by RPDO1 Mappings.

Display Only

Name	Value	Units	Scale	Modbus Address
Remote Throttle Voltage	3	Volts	4096	495
Gear ratio	2.3984	N:1		
Rated system voltage	48	Volts		
Wheel diameter	600	mm	1	227

Results in BACDoor™, and updated TPDO1 Message

Receive / Transmit	Trace	PCAN-USB			
CAN-ID	Type	Length	Data	Cycle Time	Count
1AAh		8	F8 03 58 02 66 02 30 00	100.3	14877

Custom Addresses

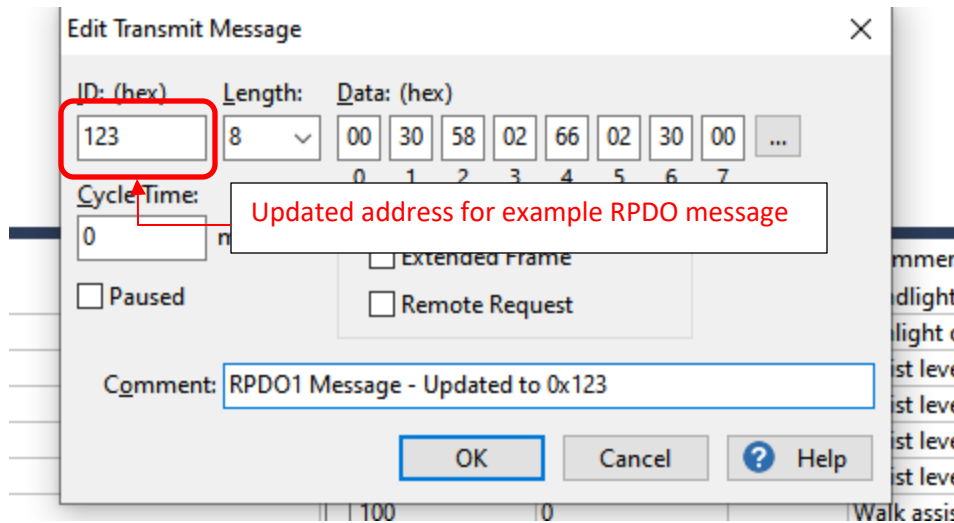
To change the addresses in the above examples, to read from a specific address, or transmit to a specific node ID, simply disable the “Autoset CANOpen COBID” flags, for both TPDO and RPDO (bits 5 and 6 in the Communications Configuration Vector), and then change the PDO low word to a custom address of your choice in a valid range of 11 bit can addresses (0-0x7FF). After saving the update to flash and rebooting, the updated PDO should receive or transmit at that specific address. In the example below, after disabling the two autoset flags, the RPDO1 COBID (low word) is set to 0x0123. Then, simply editing the example RPDO message to address 0x0123 instead of 0x022A will allow for testing.

For further detail regarding specific features, and their usage, please contact Accelerated Systems for support.

Write/Save

Name	value	Units	Scale	Modbus Address	CAN Address	Access Level
Communications Configuration Vector	0000000010010000					
RPDO1 COBID (high word)	0000	hex				
RPDO1 COBID (low word)	0123	hex				
RPDO1 size (words)	4					
RPDO1 transmission type	1		1	1801	0x201C(0A)	0
RPDO1 timeout	0	ms	1	1802	0x201C(0B)	0
RPDO1 map1 index	2007	hex	hex	1793	0x201C(02)	0
RPDO1 map1 sub index and size	3010	hex	hex	1792	0x201C(01)	0
RPDO1 map2 index	2003	hex	hex	1795	0x201C(04)	0
RPDO1 map2 sub index and size	2410	hex	hex	1794	0x201C(03)	0
RPDO1 map3 index	2003	hex	hex	1797	0x201C(06)	0
RPDO1 map3 sub index and size	2310	hex	hex	1796	0x201C(05)	0
RPDO1 map4 index	2001	hex	hex	1799	0x201C(08)	0
RPDO1 map4 sub index and size	0710	hex	hex	1798	0x201C(07)	0
RPDO2 COBID (high word)	0000	hex	hex	1871	0x201D(10)	0
RPDO2 COBID (low word)	032A	hex	hex	1870	0x201D(0F)	0

Modified Configuration vector to disable autosetting PDO addresses, and updated RPDO1 Address



Dialog box: Edit Transmit Message

ID: (hex): 123

Length: 8

Data: (hex): 00 30 58 02 66 02 30 00

Cycle Time: 0 ms

☐ Paused

☐ Extended Frame

☐ Remote Request

Comment: RPDO1 Message - Updated to 0x123

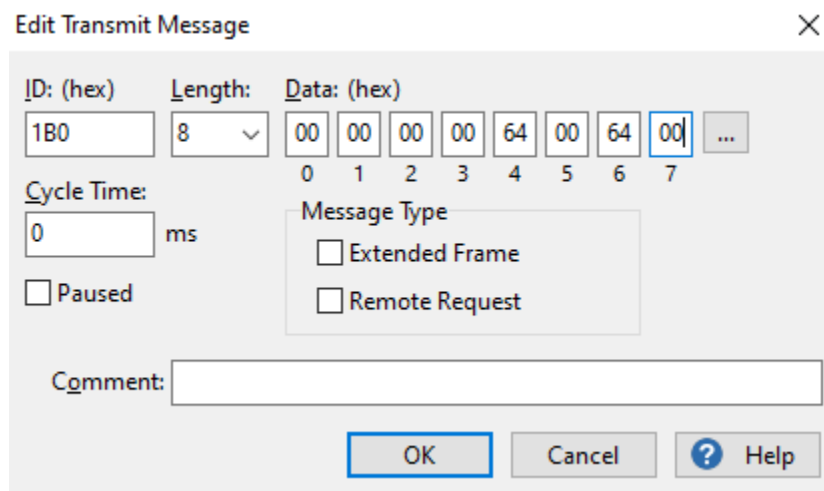
Buttons: OK, Cancel, Help

Battery RPDO Setup

Ensure that the “Autoset RPDO for CANOpen” bit is cleared in the communication configuration vector. Then, configure RPDO1 with the following settings:

1. RPDO1 COBID (high word): 0x0000
2. RPDO1 COBID (low word): 0x01B0
3. RPDO1 size (words): 4
4. RPDO1 transmission type: 1
5. RPDO1 map3 index: 0x2007
6. RPDO1 map3 subindex and size: 0x3810
7. RPDO1 map4 index: 0x2007
8. RPDO1 map4 sub index and size: 0x3910

After saving to flash, and reloading, you can test by sending the following CAN message to simulate the CAN battery in PCAN with a State of charge of 100% and a 100 degree temperature, and observing the Remote battery SOC and Remote battery temperature (addresses 503 & 504) update.



Dialog box: Edit Transmit Message

ID: (hex): 1B0

Length: 8

Data: (hex): 00 00 00 00 64 00 64 00

Cycle Time: 0 ms

☐ Paused

☐ Extended Frame

☐ Remote Request

Comment:

Buttons: OK, Cancel, Help

Display TPDO Setup

While various displays expect various formats/data, an example TPDO that can be set up to report Controller Status, Vehicle Speed, Motor Power, and Trip Odometer for use in a display is as follows:

- | | |
|------------------------------------|---|
| 1. TPDO1 size (words): | 4 |
| 2. TPDO1 event time: | 100 (ms) |
| 3. TPDO1 map1 index: | 0x2004 |
| 4. TPDO1 map1 sub index and size: | 0x0210 |
| 5. TPDO1 map2 index: | 0x2004 |
| 6. TPDO1 map2 sub index and size: | 0x0510 |
| 7. TPDO1 map3 index: | 0x2005 |
| 8. TPDO1 map3 sub index and size: | 0x0F10 |
| 9. TPDO1 map4 index: | 0x2006 |
| 10. TPDO1 map4 sub index and size: | 0x1910 |
| 11. TPDO1 COBID (high word): | User choice – (0x2000 for extended, 0x0 for standard) |
| 12. TPDO1 COBID (low word): | User choice – (CAN Address display is expecting) |

The user can also enable the “Autoset CANOpen COBIDs for TPDOs” feature to have the controller automatically transmit this message at the CAN open address (TPDO1 Base address of 0x180 + Node ID).

To confirm this, CAN messages can be observed and the data packet can be converted as follows:

- Byte 1: Enumeration:
 - 0 = Pre-operational
 - 1 = Idle
 - 2 = Post Operational
 - 3 = Run
 - 4 = Fault
- Byte 2 – Unused
- Byte 3&4 – Vehicle Speed in little endian format
 - Convert to decimal, and divide by 256 to get km/hr
 - Ie. Byte 3: 0x00 Byte 4: 0x19 = 6400 decimal/256 = 25km/hr
- Byte 5&6 – Motor Power in little endian format
 - Convert to decimal to get power in Watts
 - Ie. Byte 3: 0xFA Byte 4: 0x00 = 250 decimal = 250W
- Byte 7&8 – Trip odometer in little endian format
 - Convert to decimal, and divide by 100 to get km
 - Eg. Byte 7: 0x90 Byte 8: 0x1A = 6800 decimal/100 = 68km

ASI CAN Object Dictionary

This section has a very brief overview of the object dictionary used for ASI controllers. Most parameters in the BAC Controller are accessed using the ASI Specific Object Dictionary entries by converting the Modbus address value in the ASI Object Dictionary file to an index and sub-index. See ASI CAN Implementation Overview above for details on the conversion process.

CAN Object Dictionary Blocks

The following is a quick reference of the Object Dictionary Blocks used in the controller. For a complete description of the ASI Specific Object Dictionary entries, please refer to the ASI Object Dictionary. In general, when customizing a controller setup, the ASI Specific Object dictionary should be used. Not all indices are supported by all controllers.

Index	Entry
0x1000	Device Type
0x1001	Error Register
0x1005	COB-ID SYNC
0x1007	Synchronous Window Length
0x1009	Manufacturer Hardware Version
0x100A	Manufacturer Software Version
0x1011	Restore Default Parameters
0x1017	Producer Heartbeat Time
0x1018	Identity Object
0x1029	Error behavior
0x1200	Server SDO Parameter 1
0x1201	SDO server parameter 1
0x14XX	RPDO communication parameter
0x16XX	RPDO mapping parameter
0x18XX	TPDO communication parameter
0x1AXX	TPDO mapping parameter
0x1F51	Program Control
0x1F57	Flash status identification
0x2000 – 0x2024	ASI Specific Object Dictionary Entries
0x6000	Supported virtual devices
0x6001	Device control
0x6002	Device status

0x6003	Device capability
0x6006	Device electronic production date
0x6008	Device rated voltage
0x6009	Device alarm status
0x600A	Device alarm capability
0x600B	Device warning status
0x600C	Device warning capability
0x6024	Device maximum continuous input current
0x6025	Device maximum continuous output current
0x6026	Device maximum voltage
0x6027	Device minimum voltage
0x603A	Device peak input current
0x603B	Device peak input current time
0x603C	Device peak output current
0x603D	Device peak output current time
0x603E	Device actual current
0x6040	Device actual voltage
0x6042	Device electronic temperature
0x605A	Device password
0x6200	MCU – Manufacturer Name
0x62D0	MCU Motor controller capability
0x62D8	MCU – Motor limitation capability
0x62E8	MCU – Actual motor values
0x6306	MCU – Set motor assistance level
0x6308	MCU – Set maximum vehicle Speed