TECH/OPS

SEVCON

Application Note

Title	Controlling AC Via CANOpen
Filename	Controlling AC via CAN.docx
Date of Creation	12/03/2008 by Dave Conboy
Last Updated	26/10/2012 by Matthew Karas

Sevcon Ltd
TVTE
Gateshead
Tyne & Wear
England NE11 0QA

Tel +44 (0)191 4979000
Fax +44 (0)191 4824223
www.sevcon.com

1. Initial Setup

1.1. Configuring Baud Rate, Node Id and NMT role.

This configuration can be done via sdo write through DVT, Drivewizard or another CANOpen software tool.

- Configure the **Node Id** for each node under OD object 0x5900, 1. This can be any number from 1 to 127.
- Configure the **baud rate** for each Sevcon Node via the OD object 0x5900, 2.
- Configure if the node is a NMT master or a slave node via the OD object 0x5800 1. There should be only one NMT master on the network. 0x5800 sub 1 set 0 for Slave 1 for Master.
- Most third party devices have the default configuration of NMT slave.

2. CAN Isolation

For any application over 60V the EU vehicle safety standards require isolation of the traction higher voltage battery from the chassis and other 12V electrical systems. This is no problem for Gen4 Size 8 or GpAC as these controllers have isolated CAN circuits. On Gen 4, however no isolation is offered and therefore it may be necessary to use a CAN isolator such as an ICP DAS 7531 http://www.icpdas.com/products/Remote_IO/can_bus/i-7531.htm Product manual is embedded below. When connecting this isolator it advised to connect both CAN network grounds and ensure that each network has a total bus resistance of 60Ω.

CAN isolator
i-7531-manual.pdf

3. Creating a Safety Critical System

The recommended CANopen system for monitoring the health of attached node is to use the Heartbeat system. CANopen heart beats have the COB ID 0x700 + node id so that 0x701 is the heartbeat from node id 1.

Each node produces[1] its own unique heartbeat and consumes[2] a variety of other heartbeats on the bus as required. A CANbus analyser tool (such as the DVT) will reveal what messages are on the CAN bus.

If it not desired to have a node to render the vehicle inoperable should it fail, such as a display, it should not be registered as a slave with the master.

3.1. Configuring the Heartbeat Protocol

- For each node which is protected by the heartbeat protocol the following steps are required, these can be found under the CANopen/NMT error control/Heartbeat tree node of Drive Wizard.
- Set the **heartbeat producer time** to something agreed appropriate for that system, a typical value is 500ms. This is set via OD index 0x1017, 0.
- Set the **heartbeat consumer time** for each of the received heartbeats that this node is to raise a fault should one cease to be broadcast. These objects are a 32 bit values and have a the following format 0x00NNTTTT. These are set via OD index 0x1016, Where NN is the node id of the producer and TTTT is the timeout value in ms. ***NB Drive wizard separates these out automatically (if the DW version (0001.00013+) and EDS (shiroko 0007.0011+) support and use bit-splitting) such that all that is required is to enter the node id and heartbeat timeout in separately.*** It is recommended to double the producer time for the timeout, ie if the producer has 500ms heartbeat period then the consumer time should be set to 1000ms. Eg. On another node heartbeats produced by node id 1 should be checked and a fault raised if one is not seen for 1s would be 0x000103E8. Set all other unused entries to 0x00000000.
- To ensure that each slave responds to the boot up message, cobid 0x0000 it is important to make sure that all unused RPDO COB ids (in the range 0x1400 to 0x1408), unused sdo server cob ids (0x1201,1 & 2) and slave SYNC cob ids (0x1005) are all non zero and disabled in the following way :
- Unused RPDO COB ids should have bit 15 set ie 0x80000XXX this disables the RPDOs.
- Unused SDO server COB ids should have bit 15 set ie 0x80000XXX this disables the SDO.
- Slave SYNC message cob IDs should be set to 0x00000080 and the master set to 0x40000080, in OD object 0x1005

NB To store CANopen parameters the controller must be logged in, in pre operational.

3.2. Register all Protected Nodes with the NMT master

The NMT master needs to know what nodes are present on the bus in order to tell each of them to go to operational.

This is done in the CANopen/General/Set up section of Drive Wizard or via the OD entry 0x2810 where sub index 0 represents the number of slaves. The following sub indices must be set to the node ids of the protected nodes in the system. Eg

- 0x2810, 00 = 0x02 (nodes on system)
- 0x2810, 01 = 0x02 (node 2)
- 0x2810, 02 = 0x03 (and node 3)

The system should now power up without any CANbus faults, should any of the attached nodes be unplugged (which were registered for heart-beating) from the CANbus the effected nodes will now report a CANbus Heartbeat error.

[1] CANopen terminology referring to a heartbeat producer, this reference can be found in the object dictionary and the DS301 CANopen communication profile.

[2] CANopen terminology referring to a heartbeat consumer, ie the node is "listening" in for other nodes presence

4. Controlling the Gen 4 as a Master via CAN

It is possible to configure multiple Sevcon Master Controllers on one CANbus and utilise the higher level traction control to provide speed\torque ramping, controller\motor fault detection and the additional features associated with a Sevcon master within the object ranges 0x2000-0x2FFF. This setup method allows the user to send via CAN direction switches for drive and a throttle and\or brake demand via CAN. This also removes the need for the user ECU to carry out ramp rates and the controlword\statusword motor control start up sequence needed if the controller was configured as a pure motor slave controller compliant with CANOpen DSP402. If this control is a requirement then section 5 covers this in more detail.

When setting up the system it is advised that the following items are configured and designed into the system.

- Each controller should control its own line contactor to allow it react to faults and open the line contactor if need be.
- Only one master on the entire CANbus transmits Synchronisation ID's 0x80. This can be disabled in object 0x1005 by setting it to 0x80000080. If more than one master transmits this ID then the CANbus may become overloaded as each controller will send TPDO's on each SYNC message.
- Each controller emergency message should be set accordingly in 0x1014.
- Each controller should have unique TPDO COB ID's set in 0x1800- 0x1804 sub 1.

Once these are setup then it should be possible to configure RPDO's in each drive for direction inputs and throttle demand accordingly. Transmission rates of TPDO's should be as fast as possible, ideally 20ms.

5. Controlling Slaves Directly from 3rd Party CAN Nodes

It must be advised that if configuring the controller as a pure motor slave then the user ECU is expected to carry out the majority of the core vehicle control, this will include:

- Ramping of torque demands transmitted via CAN to the controller. The controller has one speed ramp rate when configured as a slave 0x6083, but this is used for both acceleration and deceleration. A torque slope rate is also possible in 0x6087, again used in both acceleration and deceleration.
- Fault monitoring and taking appropriate safe action such as opening the main isolating contactor. The controller will set a fault and turn off its motor control but has no immediate control over contactors etc.

To control a Sevcon slave node directly from a 3rd party CAN node the following steps must be carried out:

- Login to the required node by writing the password and user ID to 0x5000 subs 2 & 3 respectively. Contact Sevcon for this information.
- Set the slave or all slaves to operational by sending “0x00 0x01 0x00” where these bytes refer to :

  0x00 = NMT ID Command monitored by all nodes.

  0x01 = set operational, 0x80 = set pre-operational.

  0x00 = node id to be set, if set to 0 all nodes will be set.

e.g. In DVT this would be can send “0x00 0x01 0x02” to set node 2 to operational and can send “0x00 0x80 0x02” to set node 2 to pre-operational.

The following steps assume that SDO’s will be used to write to objects with a client->server cob id of 0x0601. These are set in object 0x1200. If PDO’s are to be used a different format will be required.

If a line contactor or brake is fitted enable this by writing to the correct analogue output drive. In the case of a line contactor on lCont 1 on Nano this would be object 0x6c11 sub 1. Writing a value of 0x7fff will apply full battery voltage across the coil. In DVT this can be done by

can send "0x0601 0x2B 0x11 0x6c 0x01 0xFF 0x7F 0x00 0x00" where

0x0601 SDO Client cob ID

0x2B 16 bit message This data is built up from the SDO protocol bits 7-0. It specifies 16 bit in this packet as the data is expedited and the n bytes containing no data.

0x11 & 0x6c, object to write to 0x6c11

0x01 sub index of 0x6c11

0xff & 0x7f, value to write 0x7ff

0x00 & 0x00 are not used but must be contained in packet.

To open the line contactor it would be done by setting 0v output voltage

can send "0x0601 0x2B 0x11 0x6c 0x01 0x00 0x00 0x00 0x00"

Enable the motor control model; this can be done by first writing 0x06 (disable) to the control word object 0x6040 followed by writing 0x07 & then 0x0f(enable) The model should now be enabled.

can send "0x0601 0x2B 0x40 0x60 0x00 0x06 0x00 0x00 0x00" (Disables model)

can send "0x0601 0x2B 0x40 0x60 0x00 0x07 0x00 0x00 0x00"

can send "0x0601 0x2B 0x40 0x60 0x00 0x0f 0x00 0x00 0x00" (Enables model)

The motor can now be controlled by writing a target speed to object 0x60ff (target speed). This value is a signed integer with a positive value denoting a forwards direction movement in rpm and a negative value denoting reverse direction in rpm. Or alternatively if running in torque mode a target torque can be sent to object 0x6071.

can send "0x0601 0x23 0xFF 0x60 0x00 0x00 0x01 0x00 0x00" sets a target speed of +256rpm to 0x60ff

can send "0x0601 0x23 0xFF 0x60 0x00 0x00 0x00 0x00 0x00" sets a target speed of 0rpm to 0x60ff

can send "0x0601 0x23 0xFF 0x60 0x00 0xff 0xfe 0xff 0xff" sets a target speed of -256rpm to 0x60ff

can send "0x0601 0x2B 0x71 0x60 0x00 0x00 0x0f 0x00 0x00" sets a target torque of 15 to 0x6071

5.1. Controlling via PDO’s

The commands and sequence above primarily describe using an SDO command on the default client\server channels 601 & 581. to write to the command word and torque\speed demand. This is ok for quick testing of CAN control, however for a safety critical system PDO’s should be used. The bare minimum for PDO’s would involve setting up RPDO’s in the controller for the following objects:

Control Word – 0x6040

Target Speed 0x60ff or Target Torque 0x6071

Any contactor drivers driven directly by writing to 0x6c11

Setup an RPDO in the controller which will handle these objects sent via TPDO from a third party ECU. RPDO setup is carried out in 0bject 0x1400-1x1404 for setup and in 0x1600-0x1604 for associated data. See Appendix for more details

For example to setup an RPDO in a slave to receive controlword and torque on a cob id of 191 in DVT is as follows:

sdo_wnx 1 0x1400 1 0x00000191

sdo_wnx 1 0x1400 2 0x01

sdo_wnx 1 0x1600 0 0x00

sdo_wnx 1 0x1600 1 0x60400010

sdo_wnx 1 0x1600 2 0x60710010

sdo_wnx 1 0x1600 0 0x02

.A TPDO must now be received via CAN with a cob ID of 0x0191 containing 32bits of data of which the first 16 are the control word and the second 16 are target torque.

It is also advisable that TPDO's are setup and monitored by the master ECU, as a minimum the statusword 0x6041 should be monitored. To complete the power up process successful it is advisable to use the statusword and controlword sequence as below:

1. At power up, the controlword and statusword are both zero.
2. Motor slave will perform power up tests automatically and, if successful, will proceed to the "switch on disabled" state, reporting a statusword of 0x0040.
3. Master sends control word of 0x0006 to request shutdown.
4. Slave responds with statusword of 0x0021, ready to switch on.
5. Master sends control word of 0x0007 to request switch on.
6. Slave responds with statusword of 0x0023, switched on. The bridge will now also be enabled.
7. Master sends control word of 0x000f to request enable operation.
8. Slave responds with statusword of 0x0027, operation enabled. Slave will now respond to demands for speed or torque.

Controlword and Statusword may have other bits set when running in a live system. This is normal as extra manufacturer specific bits will be in use, However, these can normally be ignored for experimental purposes.

5.2. Safety Critical Parameters

Once PDO communication is established and working a number of safety objects can be set to ensure the system will respond safely should communication fail. These are listed below:

RPDO timeout. The controller software incorporates an RPDO timeout scheme which will carry out a defined action should an RPDO not be received by the ECU for 500ms. This is set in object 0x5902 as below:

The current available setups are:

0x5902,0 = 0 - None

0x5902,0 = 1 - Warn and limit drive only - This indicates a warning level fault and applies a driveability profile (Drv Sel 2) to slow vehicle speed. This must only be used on vehicles where no safety critical inputs are received over CANbus

0x5902,0 = 2 - Inhibit drive - Sets a drive inhibit fault causing the vehicle to stop. Line remains closed.

0x5902,0 = 3 - Immediate stop - Sets a severe fault causing the line to open and the

EBrake to be applied immediately.

Note: Some of these schemes will only work correctly if a Sevcon master is configured in the system. If an ECU is used to drive Sevcon devices configured as pure slaves then it is advised that a severe fault scheme is used in conjunction with the setting to control the contactor drives in error mode, object 0x6c43, see below.

- Contactor Drivers Error mode. Each of the contactor drivers have a voltage setting in 0x6c43 which the drive will revert to in the case of a communication fault, including RPDO timeout, being set. Setting this object to an appropriate voltage will ensure that when such a fault is set the line contactor will open and any auxiliary drives such as ebrakes will be reverted to a safe state.
- Additional monitoring of CANopen emergency faults and statusword should also be carried out by the master ECU.

6. SDO Protocol

This protocol is used to implement the Initiate SDO Download service for SDOs.

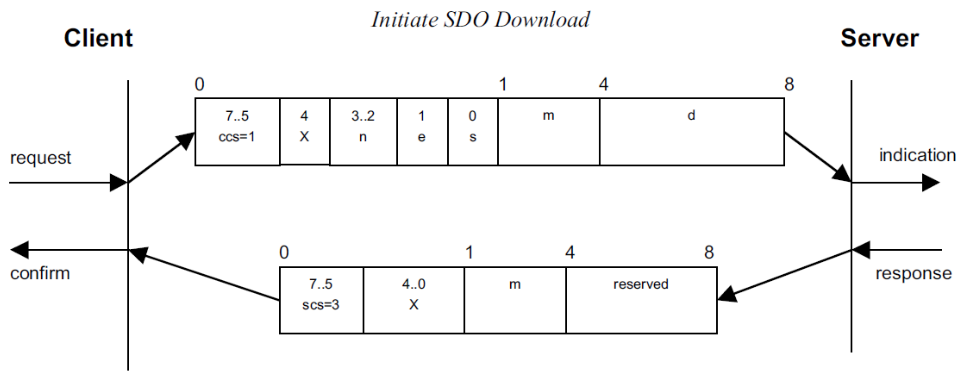


Figure 17: Initiate SDO Download Protocol

- **ccs:** client command specifier
  1: initiate download request
- **scs:** server command specifier
  3: initiate download response
- **n:** Only valid if e = 1 and s = 1, otherwise 0. If valid it indicates the number of bytes in d that do not contain data. Bytes [8-n, 7] do not contain data.
- **e**: transfer type
  0: normal transfer
  1: expedited transfer
- **s:** size indicator
  0: data set size is not indicated
  1: data set size is indicated
- **m**: multiplexor. It represents the index/sub-index of the data to be transfer by the SDO.
- **d:** data
  e = 0, s = 0: d is reserved for further use.
  e = 0, s = 1: d contains the number of bytes to be downloaded.
  Byte 4 contains the lsb and byte 7 contains the msb.
  e = 1, s = 1: d contains the data of length 4-n to be downloaded,
  the encoding depends on the type of the data referenced
  by index and sub-index
  e = 1, s = 0: d contains unspecified number of bytes to be downloaded
- **X:** not used, always 0
- **reserved:** reserved for further use, always 0

This protocol is used to implement the Abort SDO Transfer Service.

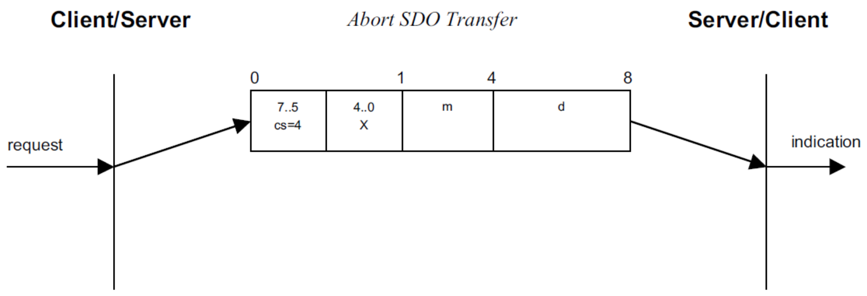


Figure 22: Abort SDO Transfer Protocol

- **cs:** command specifier
  4: abort transfer request
- **X:** not used, always 0
- **m**: multiplexor. It represents index and sub-index of the SDO.
- **d:** contains a 4 byte abort code about the reason for the abort.

The abort code is encoded as UNSIGNED32 value.

Table 20: SDO abort codes

Abort code	**Description**
0503 0000h	Toggle bit not alternated.
0504 0000h	SDO protocol timed out.
0504 0001h	Client/server command specifier not valid or unknown.
0504 0002h	Invalid block size (block mode only).
0504 0003h	Invalid sequence number (block mode only).
0504 0004h	CRC error (block mode only).
0504 0005h	Out of memory.
0601 0000h	Unsupported access to an object.
0601 0001h	Attempt to read a write only object.
0601 0002h	Attempt to write a read only object.
0602 0000h	Object does not exist in the object dictionary.
0604 0041h	Object cannot be mapped to the PDO.
0604 0042h	The number and length of the objects to be mapped would exceed PDO length.
0604 0043h	General parameter incompatibility reason.
0604 0047h	General internal incompatibility in the device.
0606 0000h	Access failed due to an hardware error.
0607 0010h	Data type does not match, length of service parameter does not match
0607 0012h	Data type does not match, length of service parameter too high
0607 0013h	Data type does not match, length of service parameter too low
0609 0011h	Sub-index does not exist.

0609 0030h	Value range of parameter exceeded (only for write access).
0609 0031h	Value of parameter written too high.
0609 0032h	Value of parameter written too low.
0609 0036h	Maximum value is less than minimum value.
0800 0000h	general error
0800 0020h	Data cannot be transferred or stored to the application.
0800 0021h	Data cannot be transferred or stored to the application because of local control.
0800 0022h	Data cannot be transferred or stored to the application because of the present device state.
0800 0023h	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error).

The abort codes not listed here are reserved.

7. TPDOs

TPDOs are used to copy information from the Object Dictionary to the CANbus. The data being sent by the TPDO will appear on the CANbus as a series of CAN messages.

Each message on the CANbus consists of an 11-bit COB-ID[3] and up to 8 bytes of data. When using the DVT, CAN messages can be seen to be scrolling up the CAN window.

TPDOs will repeatedly send CAN messages using a specific COB-ID. The system designer should assign a unique COB-ID for the TPDO to use. The message body used by the TPDO is able to store up to 8 bytes of data.

For example, I may wish to create a TPDO that sends out the following information:

- The voltage read by the first analogue input, available from 0x6C01, 1
- The speed of the local motor, available from 0x606C, 0
- The statusword of the local motor, available from 0x6041, 0

From the Object Dictionary, we see that the analogue input voltage and statusword are 2 bytes each, and the local motor speed is 4 bytes, giving us 8 bytes in total. This should completely fill one TPDO.

In this example a COB-ID of 0x201 has been chosen as this will not interfere with anything else on the CANbus.

The data could be arranged in the CAN message as follows:

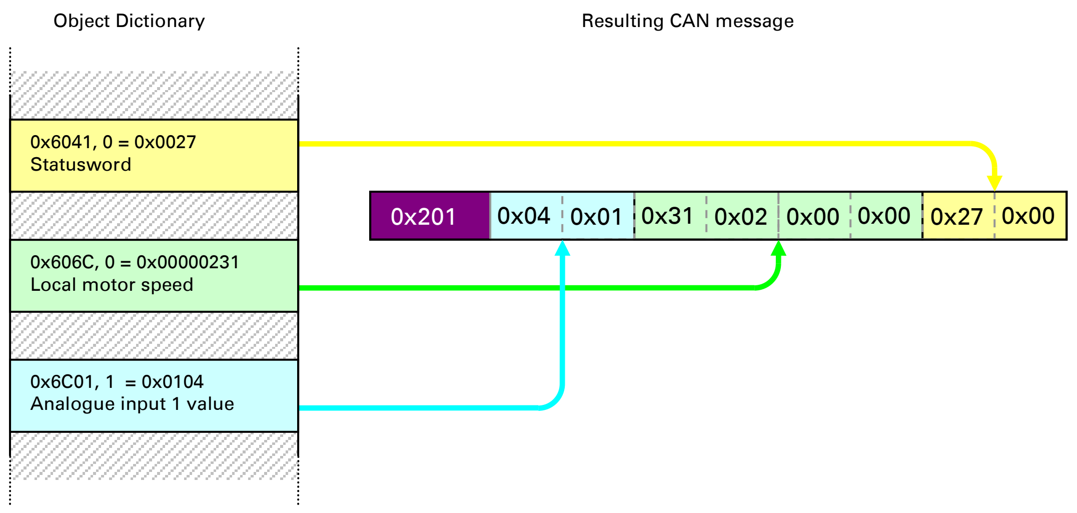


Figure 1 - Illustration showing formation of TDPO messages

The inverter can then be configured to send this CAN message repeatedly to the CANbus. The contents of the message will be updated, but it will always have a COB-ID of 0x201. Therefore it is possible for other nodes on the system to receive this data.

It is possible to configure up to 9 TPDOs on the inverter.

7.1. Defining the *Contents* of the TPDOs

The contents of each TPDO is defined in the Object Dictionary at locations 0x1A00. Objects 0x1A00 – 0x1A08 are supported by the inverter, a total of 9 possible TDPO. Each object has 8 sub-indices, meaning that each TDPO can have up to 8 items mapped into it.

Each of the mapping parameters is a 32-bit number. In the number, we encode the index, sub-index and length of each item that it to be mapped into the TDPO. The first 16-bits represent the index, the next 8 bits represent the sub-index and the last 8 bits represent the length of the object being mapped. These values are best expressed in hex.

In the above example of statusword, speed and analogue input value, we would set up the first TPDO mapping as follows:

Index	Sub Index	Value
0x1A00	0	0x03

[3] COB-ID = Communication Object Identifier. This is the CANopen term for the CAN message identifier.

	1	0x6C010110
	2	0x606C0020
	3	0x60410010

The above objects define what information is to be sent in the TPDO. However it does not define what COB-ID the TPDO will use, or how often the data is to be sent. This is discussed in the next step.

7.2. TPDO Transmission Parameters

TPDO transmission parameters are defined in objects 0x1800-0x1808. The transmission parameters correspond to the TPDO contents that are defined at locations 0x1A00-0x1A08.

Each transmission type object has 5 sub-indices.

Sub index 1 defines the COB-ID of the TPDO. This is 32 bits long as it was designed to support the 29 bit COB-IDs available with CAN v2.0B. However, 29 bit COB-IDs are not used at present. This entry may also be set to 0x80000000, which will disable the TPDO.

Sub index 2 defines the transmission type. Set this to zero to have asynchronous TPDO transmission. In most circumstances you would set it to a value between 1 and 240, to specify how many SYNC messages[4] must be received before the TPDO is sent, so it is possible to specify the TPDO is sent, say, every 5th SYNC message. Setting this sub-index to 255 indicates the TPDO will be sent when there is a change to any of the values mapped to the TPDO.

Sub index 3 specified an inhibit time. For example, if the inhibit time is set to 10ms, the TPDO will never be sent more than once every 10ms. This is useful when the transmission type is set to 255, as it will prevent the CANbus being clogged with data from a sensor whose value may change many times in a short period

Sub index 4 does not exist on the inverter, or on many other CANopen devices. This sub-index is reserved for future use.

Sub index 5 is an event timer. A time in ms may be specified here. If a time is specified, then the TPDO will be sent on every time interval in addition to any other triggers that may exist. Set this sub index to zero to disable the event timer.

In the above example of sending the first TPDO using COB-ID 0x201 on every SYNC message, we would set the communication parameters as follows:

Index	Sub-index	Value
0x1800	0	0x05
	1	0x00000201
	2	0x01
	3	0x00
	5	0x00

7.3. Additional Notes

Some miscellaneous notes to bear in mind when setting up TPDOs:

- TPDOs do not need to be acknowledged. There is no problem if a device is configured to send TPDOs and there are no other devices on the bus to do anything with these messages. In fact, this is a useful situation as it is common to find nodes with a set of TPDOs configured for use with monitoring tools such as the DVT which are not always connected.
- The system must be in the pre-operational state in order to change any PDO mappings. In addition, sub index 0 of any of the mapping parameter objects must be set to zero before the mappings can be changed.
- Changes to TPDO and RPDO mappings are not stored to EEPROM automatically. They will only be written to EEPROM when the store command is sent. To send the store command from the DVT, simply type 'store' and press return. If you key off before entering the store command you will lose your PDO setup.
- A DVT script, pdo_config.tcl, is available as part of the DVT package. It sets up a default set of TPDOs that can be used when logging data with the DVT. Studying this script will provide more information on how TPDOs are configured.

[4] SYNC messages are produced by the master node. They usually have a COB-ID of 0x80, but not always. The frequency at which SYNC messages are sent out can be adjusted in the Object Dictionary of the master node at index 0x1006.

- Strictly speaking, you should only map objects which are read only to TPDOs. It is possible to map some which are read/write, but this may cause problems. For instance, when transmitting the state of digital inputs, it is better to transmit object 0x6800 than objects 0x21**.
- CANopen data objects are written in the CAN packets in a little-endian form ie back to front.

8. RPDOs

RPDOs perform the opposite action to TPDOs in that they take information from the CANbus and copy it back into the Object Dictionary. Each time a CAN message is received, the RPDOs are checked to see if they contain any information that needs to be copied back to the Object Dictionary.

The example below follows on from the example used in the TPDO section. The statusword and motor speed items are being copied into the left motor drive information object at 0x2020, sub-indices 2 and 4. The first 2 bytes are ignored however. This may be because they are intended for a different node.

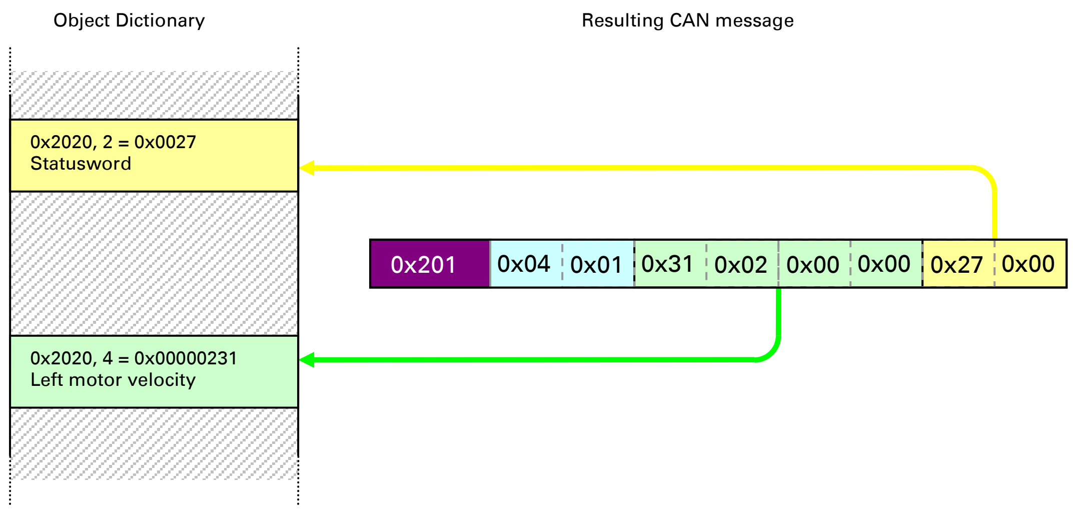


Figure 2 - Illustration showing operation of RPDOs

Using RPDOs we can configure the inverter to look out for CAN messages with COB-ID 0x201, and, when one is received, copy the contents of the message into the Object Dictionary where they can be accessed by the local application.

Limitations regarding the number of RPDOs that can be configured and the maximum number of items that can be mapped into an RPDO are identical to TPDO. There are a maximum of 9 RPDOs available, each RPDO can have a maximum of 8 objects mapped into it.

8.1. Defining the Contents of an RPDO

The mapping parameters for RPDOs are identical to those for TPDOs, except they are located in the Object Dictionary at index 0x1600 – 0x1608. The format is the same as for a TPDO, where the 32 bit number has the index, sub-index and length encoded into it, but dummy maps are also available.

In the above example where we are ignoring the first two bytes, we would have to insert a dummy mapping to fill the gap. Dummy maps are special cases that map to Object Dictionary indices 0x0001 to 0x0004. The following mappings can be used depending on how much padding space you want to insert:

Data Type	Value to use in mapping parameter	
	IXXAT stack (e.g. espAC, GpAC)	Sevcon stack (e.g. Nano, Gen4, EVo5)
Boolean, 1 bit	0x00010001	0x00010001
Byte, 8 bits	0x00020008	0x00050008
Integer, 16 bits	0x00030010	0x00060010
Integer, 32 bits	0x00040020	0x00070020

So, to handle the above RPDO mapping, including the 16 bit padding at the start of the message, our mapping parameters would be as follows:

Index	Sub Index	Value
0x1600	0	0x03
	1	0x00030010
	2	0x20200420
	3	0x20200210

It should be noted however that these dummy mappings are only valid for RPDOs. They cannot be used on TPDOs as the TPDO must have some data to send.

8.2. RPDO Reception Parameters

The reception parameters of an RPDO are very easy to specify. The only information that is required is the COB-ID of the message that will be used by the RPDO. Each time a message is received with the specified RPDO, the data is automatically copied into the object dictionary and acted upon.

The reception parameters are specified in objects 0x1400-0x1408. The format is similar to the transmission parameters of TPDOs. Sub index 1 of these objects specifies the COB-ID, which is expressed as a 32 bit number. Setting this to 0x80000000 disables the RPDO. Sub-index 2 is not used by RPDOs, but on the inverter is usually left set to 255.

In our example, we are expecting the RPDO to arrive with COB-ID 0x201. Therefore, we would have the following setup:

Index	Sub Index	Value
0x1400	0	0x02
	1	0x00000201
	2	0xFF

9. Precharge

Gen 4 controllers size 2, 4 & 6 have inbuilt precharge capability. If the application needs the Gen4's to precharge their caps the following procedure must be done.

Master node has to write an sdo to 0x5180 on each controller.

1. Write the value 1 to 0x5180 to each controller
2. Monitor Capacitor Voltage and see that it gets to the desired percentage of battery voltage (e.g. 90%).
3. Close the line contactor
4. Proceed with remainder of control/word statusword to enable drive.

5180h	Capacitor Precharge activate	Write a non-zero value to this object to start the capacitor precharge cycle.

NOTE: Activate precharge only once per key cycle for each controller. (ie – only precharge once – do not precharge again until power is shut down). Multiple precharges while the controller is on may damage the drive.

NOTE: Gen 4 size 8 & 10 do not have precharge functionality and as such precharge must be carried out by external means.

10.Cob ID Configuration

This is a guideline for configuring COB ids on a canopen bus. It is not required – but may help in setting up a system to easily understand which messages are which.

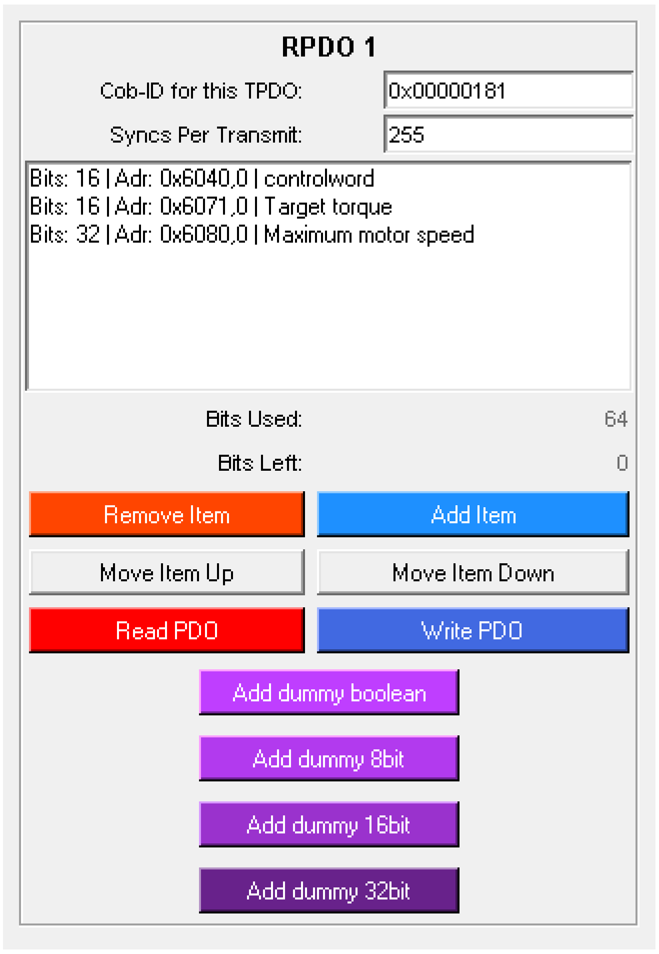
Each controller has five channels of communication.

X in the 2nd column should be the node id of the controller (allowing up to 16 nodes). The fifth channel is kept close to the forth channel due to limited standard COB IDs.

NOTE: Cob Ids that are lower have priority on the can bus – torque control messages should have lower cob-ids than status or debug messages.

PDO	**COB ID**
TPDO1	18Xh
RPDO1	20Xh
TPDO2	28Xh
RPDO2	30Xh
TPDO3	38Xh
RPDO3	40Xh
TPDO4	48Xh
RPDO4	50Xh
TPDO5	49Xh
RPDO5	51Xh

11.PDO Contents Example in DVT



RPDO 1

Cob-ID for this TPDO: 0x00000181

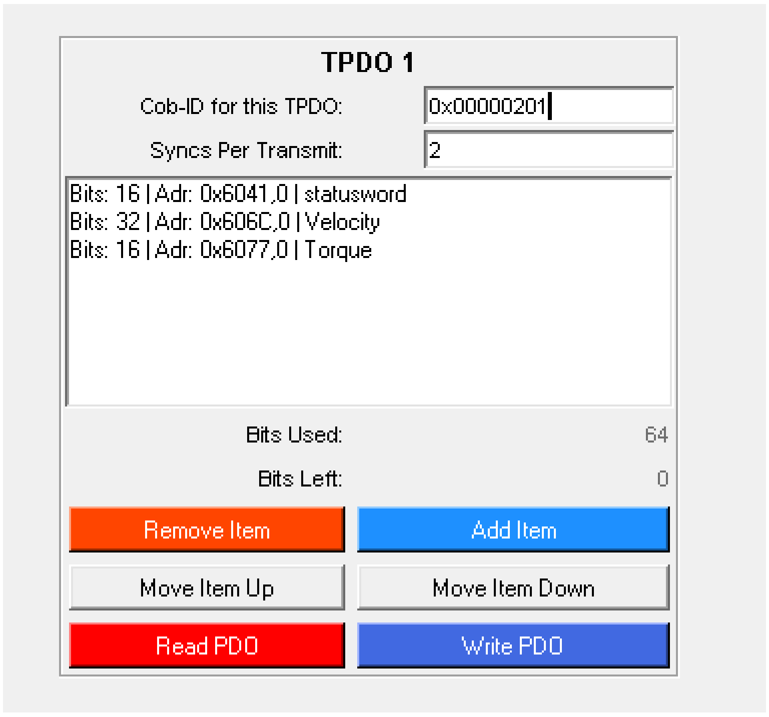
Syncs Per Transmit: 255

```
Bits: 16 | Adr: 0x6040,0 | controlword
Bits: 16 | Adr: 0x6071,0 | Target torque
Bits: 32 | Adr: 0x6080,0 | Maximum motor speed
```

Bits Used: 64

Bits Left: 0

Figure 3 The controller gets the controlword, speed limit, and target torque from the VCU



TPDO 1

Cob-ID for this TPDO: 0x00000201

Syncs Per Transmit: 2

```
Bits: 16 | Adr: 0x6041,0 | statusword
Bits: 32 | Adr: 0x606C,0 | Velocity
Bits: 16 | Adr: 0x6077,0 | Torque
```

Bits Used: 64

Bits Left: 0

Figure 4 Here the drive transmits basic information on drive status, torque produced, and actual velocity.

12. Summary of Torque Control Objects in OBD

Index	Name	Sub-Index	Scaling	Units	Data Type	Low Limit	High Limit	Description	Unit
6040h	controlword	0	N/A	N/A	Unsigned16	0	FFFFh	Controls the drive state and operational mode.	Receives
6041h	statusword	0	N/A	N/A	Unsigned16	0	FFFFh	Indicates the current state of the drive.	Transmits
606Ch	Velocity	0	1	RPM	Integer32	-20000	20000	Actual velocity in RPM.	Transmits
6071h	Target torque	0	0.1	% of peak	Integer16	-1000	1000	Target Torque in 1/1000th's of Motor Rated Torque (6076h). Calculated by the Application Master.	Receives
6077h	Torque	0	0.1	% of peak	Integer16	-1000	1000	Actual Motor Torque in 1/1000th's of Motor Rated Torque (6076h).	Transmits
6080h	Maximum motor speed	0	1	RPM	Unsigned32	0	20000	Maximum motor speed (ωnom) in RPM. It is unsigned and applies in both directions.	Receives

13.Statusword

The status word is provided by DSP402 and provides useful information about the state of the drive. It can be read from object dictionary index 0x6041 sub index 0.

Drive State:

Bits 0-3, 5 and 6 represent the drive state:

Bits	Function	State
0x0000	Not ready to switch on	Off
0x0040	Switch on disabled	Self test
0x0021	Ready to switch on	Waiting for params
0x0023	Switched on	Power off
0x0027	Operation enabled	Under control
0x0007	Quick stop active	Reset
0x000F	Fault reaction active	Fault
0x0008	Fault	Flash programming

Other status bits:

The purpose of the other status bits is largely application specific. Not all of these are assigned a function by the DSP402 standard. However, they have the following function on our controllers:

Bits	Function
0x0010	High voltage on power frame
0x0200	Remote active (only set when in "not ready to switch on")
0x0100	Motor is generating
0x0400	Target is reached
0x0800	Internal limit active
0x4000	State change is delayed (e.g. waiting for start up checks to complete)

Note: On size 8 & 10 controllers “High Voltage On” bit will be set permanently as these controllers have no battery voltage measurement circuits and hence cannot know when to set the bit.

14.Control Word and Status Word on Power Up

A sample start up procedure with regard to controlword and statusword values would be as follows:

1. At power up, the controlword and statusword are both zero.
2. Motor slave will perform power up tests automatically and, if successful, will proceed to the "switch on disabled" state, reporting statusword of 0x0040.
3. Master sends control word of 0x0006 to request shutdown.
4. Slave responds with statusword of 0x0021, ready to switch on.
5. Master sends control word of 0x0007 to request switch on.
6. Slave responds with statusword of 0x0023, switched on. The bridge will now also be enabled.
7. Master sends control word of 0x000f to request enable operation.
8. Slave responds with statusword of 0x0027, operation enabled. Slave will now respond to demands for speed or torque.

'''Notes:'''

Controlword and statusword may have other bits set when running in a live system. This is normal as extra manufacturer specific bits will be in use. However, these can normally be ignored for experimental purposes.

It is possible to omit steps 5 and 6 of the above steps, and proceed directly from the shutdown to operation enabled state.

15.Emergency Message

EMCY is short for Emergency Telegram. This is a CANopen message which nodes use to indicate a fault has been set or cleared. It has 8-bytes of data which are used as follows:

Byte	0	1	2	3	4	5	6	7
Assignment	CANopen bytes			Manufacturer specific bytes				
Function	Error code		Error reg	Fault ID		Data bytes		

where:

- Bytes 0-1 are the CANopen Error Code. These are defined in the [DS301], [DS401] and [DSP402] specifications. This is different to the 16-bit fault ID which is specific to SEVCON. If the Error Code is set to 0x0000, this indicates that ''%3cfault ID>'' has been cleared.
- Byte 2 is the CANopen Error Register. This is a bit array, each of which indicates the presence of one or more type of fault. There are bits for Generic Errors (0), Current (1), Voltage (2), Temperature (3), Communication Error (4), Device Profile Specific (5), Manufacturer Specific (7). Bit 6 is reserved.
- Bytes 3-4 are the fault ID. These are SEVCON specific and are defined further in Host Software Fault Messages.
- Bytes 5-7 are the 3 fault data bytes that are logged when the fault is set. Note, these will be 0 if the EMCY message is being retransmitted at power up. If the fault is being cleared (i.e. error code = 0x0000) then bytes 5 and 6 contain the ID of the next most severe fault on that node.

16. ControlWord & StatusWord

The *controlword* consist of bits for:

- the controlling of the state,
- the controlling of operating modes and
- manufacturer specific options.

OBJECT DESCRIPTION

INDEX	6040_h
Name	Controlword
Object Code	VAR
Data Type	UNSIGNED16
Category	Mandatory

ENTRY DESCRIPTION

Access	rw
PDO Mapping	Possible
Value Range	UNSIGNED16
Default Value	No

DATA DESCRIPTION

The bits of the *controlword* are defined as follows:

15 11	10 9	8	7	6 4	3	2	1	0
manufacturer specific	reserved	halt	Fault reset	Operation mode specific	Enable operation	Quick stop	Enable voltage	Switch on
O	O	O	M	O	M	M	M	M

MSB LSB

O - Optional M - Mandatory

BITS 0 – 3 AND 7:

Device control commands are triggered by the following bit patterns in the *controlword*:

Command	Bit of the *controlword*					Transitions
	Fault reset	Enable operation	Quick stop	Enable voltage	Switch on	
Shutdown	0	X	1	1	0	2,6,8
Switch on	0	0	1	1	1	3*
Switch on	0	1	1	1	1	3**
Disable voltage	0	X	X	0	X	7,9,10,12
Quick stop	0	X	0	1	X	7,10,11
Disable operation	0	0	1	1	1	5
Enable operation	0	1	1	1	1	4,16
Fault reset	↑	X	X	X	X	15

Table 4: Device control commands **(bits marked X are irrelevant, * ... In the state SWITCHED ON the drive executes the functionality of this state., ** ... It exists no functionality in the state SWITCHED ON. The drive does not do any in this state.)**

BITS 4, 5, 6 AND 8:

These bits are operation mode specific. The description is situated in the chapter of the special mode. The following table gives an overview:

Bit	**Operation mode**					
	Velocity mode	**Profile position mode**	**Profile velocity mode**	**Profile torque mode**	**Homing mode**	**Interpolation position mode**
4	rfg enable	New set-point	reserved	reserved	Homing operation start	Enable ip mode
5	rfg unlock	Change set immediately	reserved	reserved	reserved	reserved
6	rfg use ref	abs / rel	reserved	reserved	reserved	reserved
8	Halt	Halt	Halt	Halt	Halt	Halt

Table 5: Mode specific bits in the *controlword*

BITS 9, 10:

These bits are reserved for further use. They are inactive by setting to zero. If they have no special function, they must be set to zero.

BITS 11, 12, 13, 14 AND 15:

These bits are manufacturer specific.

10.3.2 Object 6041$_h$: *Statusword*

The *statusword* indicates the current state of the drive. No bits are latched. The *statusword* consist of bits for:

- the current state of the drive,
- the operating state of the mode and
- manufacturer specific options.

OBJECT DESCRIPTION

INDEX	6041$_h$
Name	Statusword
Object Code	VAR
Data Type	UNSIGNED16
Category	Mandatory

ENTRY DESCRIPTION

Access	ro
PDO Mapping	Possible
Value Range	UNSIGNED16
Default Value	No

DATA DESCRIPTION

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

MSB LSB

Bit	Description	M /O
0	Ready to switch on	M
1	Switched on	M
2	Operation enabled	M
3	Fault	M
4	Voltage enabled	M
5	Quick stop	M
6	Switch on disabled	M
7	Warning	O
8	Manufacturer specific	O
9	Remote	M
10	Target reached	M
11	Internal limit active	M
12 - 13	Operation mode specific	O
14 - 15	Manufacturer specific	O

Table 6: Bits in the *statusword*

BITS 0 – 3, 5 AND 6:

The following bits indicate the status of the device:

Value (binary)	State
xxxx xxxx x0xx 0000	Not ready to switch on
xxxx xxxx x1xx 0000	Switch on disabled
xxxx xxxx x01x 0001	Ready to switch on
xxxx xxxx x01x 0011	Switched on
xxxx xxxx x01x 0111	Operation enabled
xxxx xxxx x00x 0111	Quick stop active
xxxx xxxx x0xx 1111	Fault reaction active
xxxx xxxx x0xx 1000	Fault

Table 7: Device state bits (x ... irrelevant for this state)

BIT 4: VOLTAGE ENABLED

High voltage is applied to the drive when this bit is set to 1.

BIT 5: QUICK STOP

When reset, this bit indicates that the drive is reacting on a quick stop request. Bits 0, 1 and 2 of the *statusword* must be set to 1 to indicate that the drive is capable to regenerate. The setting of the other bits indicates the status of the drive (e.g. the drive is performing a quick stop as result of a reaction to a non-fatal fault. The fault bit is set as well as bits 0, 1 and 2).

BIT 7: WARNING

A drive warning is present if bit 7 is set. The cause means no error but a state that has to be mentioned, e.g. temperature limit, job refused. The status of the drive does not change. The cause of this warning may be found by reading the fault code parameter. The bit is set and reset by the device.

BIT 8:

This bit may be used by a drive manufacturer to implement any manufacturer specific functionality.

BIT 9: REMOTE

If bit 9 is set, then parameters may be modified via the CAN-network, and the drive executes the content of a command message. If the bit remote is reset, then the drive is in local mode and will not execute the command message. The drive may transmit messages containing valid actual values like a *position actual value*, depending on the actual drive configuration. The drive will accept accesses via SDO in local mode.

BIT 10: TARGET REACHED

If bit 10 is set by the drive, then a set-point has been reached. The set-point is dependent on the operating mode. The description is situated in the chapter of the special mode. The change of a target value by software alters this bit.

If *quick stop option code* is 5, 6, 7 or 8, this bit must be set, when the quick stop operation is finished and the drive is halted.

If halt occurred and the drive has halted then this bit is set too.

BIT 11: INTERNAL LIMIT ACTIVE

This bit set by the drive indicates, that an internal limitation is active (e.g. *position range limit*).

BIT 12 AND 13:

These bits are operation mode specific. The description is situated in the chapter of the special mode The following table gives an overview:

Bit	**Operation mode**					
	vl	**pp**	**pv**	**tq**	**hm**	**ip**
12	reserved	Set-point acknowledge	Speed	reserved	Homing attained	ip mode active
13	reserved	Following error	Max slippage error	reserved	Homing error	reserved

Table 8: Mode specific bits in the *statusword*

BIT 14 AND 15:

These bits may be used by a drive manufacturer to implement any manufacturer specific functionality.