TECH/OPS

SEVCON

Gen4 Application Note

Title	PDO Fundamentals
Filename	App Note - PDO Fundamentals.docx
Date of Creation	27/11/2008 09:34:00 by Paul Shipley
Last Updated	28/09/2010 16:12:00 by Paul Shipley
Revision	2

Contents

Sevcon Ltd
TVTE
Gateshead
Tyne & Wear
England NE11 0QA

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

1. Introduction

Process Data Objects (PDOs) in CANopen form the basis of configuring how internal vehicle systems communicate with the outside world. They allow data to be moved around between the Object Dictionaries of any nodes in a CANopen system.

For instance, consider the throttle input. By looking at the Object Dictionary, we see that the Traction Application expects the throttle input voltage to be written to object 0x2220. However, the analogue input voltages actually appear at objects 0x6C01. What we need is a method of continually copying values from index 0x6C01 to index 0x2220.

The solution is to use PDOs. PDOs can be arranged so that the contents of one Object Dictionary location can be copied into a different Object Dictionary location – even on a different node.

The espAC supports three types of PDO:

- Transmit PDOs (TPDOs) can be configured to periodically copy the contents of a number of Object Dictionary entry to the CANbus.
- Receive PDOs (RPDOs) are used to take information from the CANbus and copy it back to somewhere in the Object Dictionary. By using a combination of TPDOs and RPDOs it is possible to provide a constant stream of information between a number of nodes.
- Virtual PDOs (VPDOs) continually copy information from one Object Dictionary location to another without sending it to the CANbus.

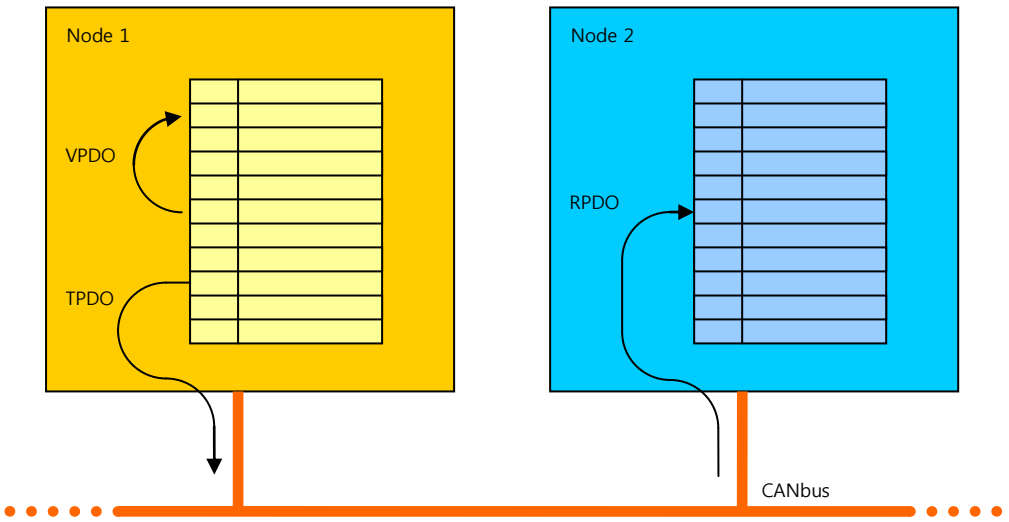

Figure 1 - Illustration showing how different types of PDO move data around the system

So, in the example where we want to set up the throttle input, we can set up PDOs to constantly copy the data into the correct location. The location where the data comes from defines which pin the throttle input should be wired to.

RPDOs and TPDOs are a standard feature of CANopen. VPDOs are specific to Sevcon and are only implemented on espAC controllers.

All PDOs are disabled unless the system is in the operational state. PDO mappings can only be configured if the system is in the pre-operational state.

2. VPDOs

VPDOs copy data from one location in the Object Dictionary to another location without sending anything to the CANbus. They are used to configure I/O on the master node.

VPDOs are the simplest of the PDO mappings to configure. Objects to configure VPDOs are located in the Object Dictionary at 0x3000 to 0x3400.

VPDOs are divided into 5 groups, with an object for each group.

- Motor VPDO mappings, located at 0x3000
- Digital output VPDO mappings, located at 0x3100
- Analogue output VPDO mappings, located at 0x3200
- Digital input VPDO mappings, located at 0x3300
- Analogue input VPDO mappings, located at 0x3400

Each object has the same number of sub-indices as there are physical inputs or outputs. For example, on Shiroko and Mistral controllers, which have 5 analogue inputs, there will be 5 sub-indices in object 0x3400.

The value written to the VPDO mapping objects is the Object Dictionary index of the vehicle application object that is to be associated with the physical input or output.

Examples:

- To make the first analogue output the electrobrake drive, set object 0x3200 sub-index 1 to be 0x2420. Object 0x2420 is where the traction application writes the electrobrake output value. This VPDO map copies whatever is in 0x2420 to 0x6C01 sub-index 1, therefore making the first analogue output the electrobrake drive.
- To make the third digital input the inch forward switch input, set object 0x3300 sub-index 3 to be 0x2129. Object 0x2129 is where the traction application reads in the states of the inch switches. This VPDO map copies whatever is in 0x6800 sub-index 1 bit 3 to 0x2129, therefore making the third digital input the inch switch input.
- To make the local motor run as the pump motor, set object 0x3000 sub-index 1 to be 0x2040. Object 0x2040 is where the pump application motor interface is. By specifying this mapping, the VPDO will copy all the relevant motor control and status data between objects at 0x2040 and the motor control objects at 0x6000-0x67FF.

VPDO mappings are intelligent enough to work out which direction they are supposed to copy the data. For inputs, data is copied from the Generic I/O[1] profile to the application objects. For outputs, data is copied from the application objects to the Generic I/O profile.

[1] Generic I/O profile as defined by CANopen document DS401. In espAC, the profile is located between objects 0x6800 and 0x6FFF.

For motors, there are a number of pieces of information which must be copied in both directions. Control words and demands are copied from the application objects to the Drives and Motion Control[2] profile object, while status words and status information is sent back from the Drives and Motion Control profile back to the application objects. A single VPDO mapping will automatically configure all the required internal mappings.

For any VPDO mapping, sub index zero indicated the size of the object and hence the number of active mappings.

2.1. Available Mappings

The following table defines the values that can be entered into the mapping objects:

Object	Mapping Type	Available Mappings
0x3000	Motors	0x2020, 0x2021, 0x2040 or 0x2060. Set to 0x20FF to disable a mapping.
0x3100	Digital outputs	Any of the digital output application objects between 0x2300-0x23FE. Set to 0x23FF to disable a mapping.
0x3200	Analogue outputs	Any of the analogue output objects between 0x2400-24FE. Set to 0x24FF to disable a mapping.
0x3300	Digital inputs	Any of the digital input objects between 0x2100-0x21FE. Set to 0x21FF to disable a mapping.
0x3400	Analogue inputs	Any of the analogue input objects between 0x2200-0x22FE. Set to 0x24FF to disable a mapping.

2.2. Typical VPDO maps

Setting the following VPDO maps will configure a typical standalone traction controller:

Index	Sub-index	Value
0x3000 - Motors	0	0x01
	1	0x2020 (motor 1)
0x3100 - Digital outputs	0	0x00
0x3200 – Analogue outputs	0	0x01
	1	0x2400 (line contactor)

[2] Drives and Motion Control profile as defined by CANopen document DSP402. In espAC, the profile is located between objects 0x6000 and 0x67FF.

0x3300 – Digital inputs	0	0x04
	1	0x2121 (fwd switch)
	2	0x2122 (rev switch)
	3	0x2123 (FS1 switch)
	4	0x2124 (seat switch)
0x3400 – Analogue inputs	0	0x01
	1	0x2220 (throttle)

Inputs Outputs

The above VPDO maps will configure the following:

- Local motor is mapped to the left motor drive information[3]
- First analogue output is the line contactor drive
- First analogue input is the throttle input
- First digital input is the forward switch
- Second digital input is the reverse switch
- Third digital input is the FS1 switch
- Forth digital input is the seat switch

The above configuration can be extended to add extra functionality if required, such as electrobrake drives, inch switch inputs or other diagnostic devices such as external LEDs or buzzers. Many features, such as the electrobrake, will become active when mapped in to a PDO.

[3] If only one motor drive information object is mapped to a PDO then it will be treat as a single motor system. It doesn't matter if you choose left or right.

3. 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[4] 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:

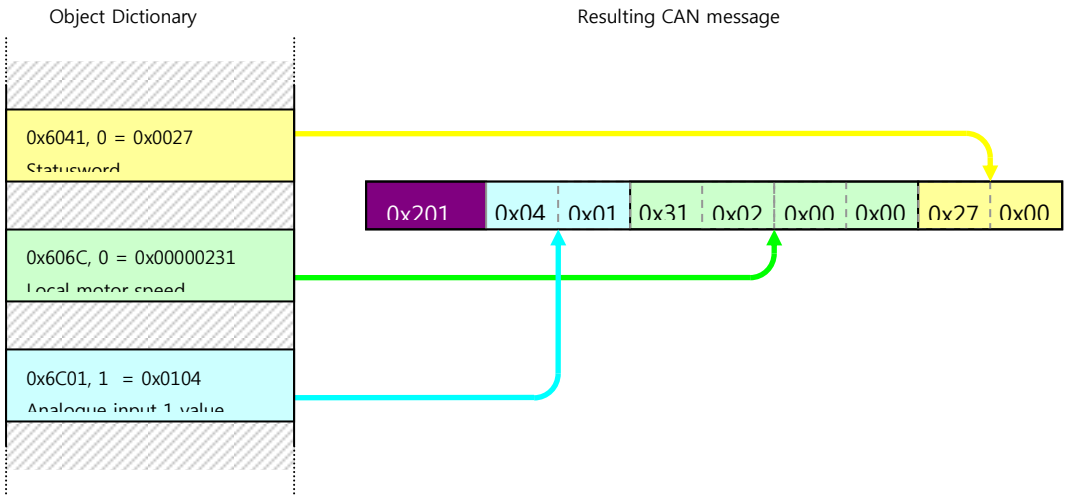

Figure 2 - Illustration showing formation of TDPO messages

The espAC 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.

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

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

3.1. 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 espAC, 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
	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.

3.2. 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, these are not used on espAC. 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[5] 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.

[5] 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.

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 espAC, 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

3.3. 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.
- 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.

4. 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.

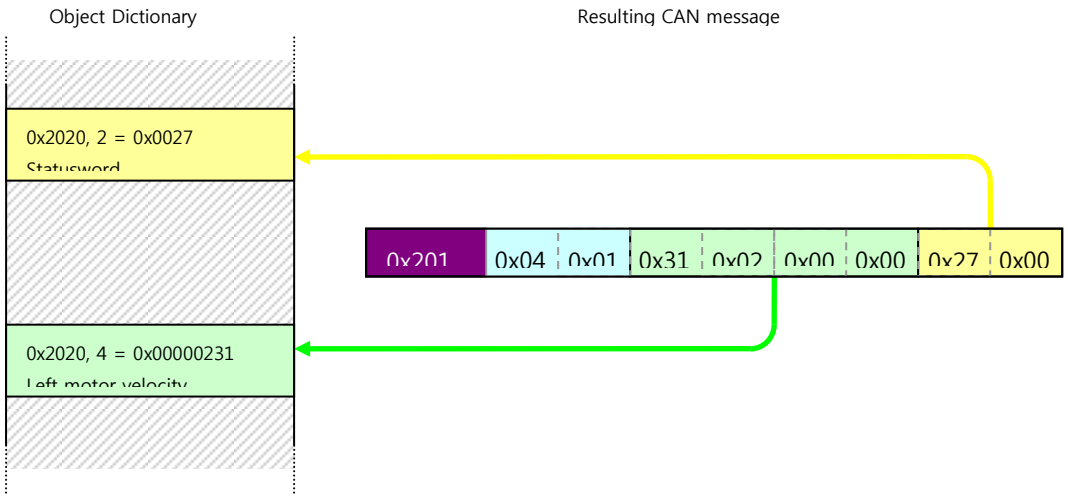

Figure 3 - Illustration showing operation of RPDOs

Using RPDOs we can configure the espAC 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.

4.1. 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

Boolean, 1 bit	0x00010001
Byte, 8 bits	0x00020008
Integer, 16 bits	0x00030010
Integer, 32 bits	0x00040020

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.

4.2. 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 espAC 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

5. Further Information

For more information on configuring PDOs, refer to the following sources:

- CiA specifications, DS301 and DS302. As a member of the CiA, Sevcon is able to download any specification from the CiA website. DS301 and DS302 define all aspects of the CANopen protocol in great detail.
- espAC user manual