**SDO – Service Data Objects – CanOpen**

November 7, 2015

Author: RobinC

Original: <http://www.byteme.org.uk/canopenparent/canopen/sdo-service-data-objects-canopen/>

**Pages**

* [CanOpen](http://www.byteme.org.uk/canopenparent/)
  + [CanOpen](http://www.byteme.org.uk/canopenparent/canopen/)
    - [Emergency Messages – CanOpen](http://www.byteme.org.uk/canopenparent/canopen/emergency-messages-canopen/)
    - [Guard protocol – CanOpen](http://www.byteme.org.uk/canopenparent/canopen/guard-protocol-canopen/)
    - [NMT Protocol – Network Managment – CanOpen](http://www.byteme.org.uk/canopenparent/canopen/nmt-protocol-network-managment-canopen/)
    - [PDO – Process Data Objects – CanOpen](http://www.byteme.org.uk/canopenparent/canopen/pdo-process-data-objects-canopen/)
    - [SDO – Service Data Objects – CanOpen](http://www.byteme.org.uk/canopenparent/canopen/sdo-service-data-objects-canopen/)
* [PLU File format](http://www.byteme.org.uk/plu-file-format/)
* [Projects](http://www.byteme.org.uk/projects/)
* [Sams4s protocol project](http://www.byteme.org.uk/sams4s-protocol-project/)
  + [Clerk file format](http://www.byteme.org.uk/sams4s-protocol-project/clerk-file-format/)
  + [Sams4s RS232 command format](http://www.byteme.org.uk/sams4s-protocol-project/sams4s-rs232-command-format/)
* [Sync Protocol – CanOpen](http://www.byteme.org.uk/sync-protocol-canopen/)

The SDO or Service Data Objects provide access to the object dictionaries in each device. They are particularly useful for configuration of devices as the SDO protocol is allowed in pre-operation mode. But it is also possible to get “process” data values by polling via SDO the appropriate object dictionary entry.

SDO protocol always confirms the read/write operation.

When performing a SDO Read or Write the Index of the object dictionary entry and the sub index is always specified, the index is 2 bytes and the sub index 1 byte. if the data to be read/written is 32 bits or less then it can be done in an expedited packet thus only one transmit and one confirm receive is necessary.

SDO uses the following COB-IDs

|  |  |
| --- | --- |
| **COB-ID** | **NMT Function** |
| 0x600+node id | SDO Receive |
| 0x580+node id | SDO Transmit |

NB the TX/RX direction are from the point of view of the device. So to query a device on the network you would send a 0x600+nodeid and get back a 0x580+nodeid

The SDO Packet looks like the following :-

| **Can header** | **rtr** | **len** | **Byte 0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x600 + node | 0 | 8 | Command | Index | | Sub Index | Data | | | |

The Can header consists of the COB ID (Function + Node), the RTR bit and 4 bits representing the packet length (0-8) This makes the header 16 bits. So the total packet is 10 bytes.

The SDO packet always contains 8 bytes of data (even if they are not all used). Command specifies the nature of the transfer read/write etc. The Index of the object dictionary being queried is the next 2 bytes (Don’t forget its Little Endian on the wire), followed by 1 byte specifying the sub-index. The remaining 4 bytes contain the data of the transfer (or zero if they are not required)

The command byte bits all have various meanings and the 8 bits of the command can be divided into bits with the following meanings depending on who sent the message. In this context the server is the node initiating the read/write operation and the client is the responding node.

If there is 4 bytes or less than the transfer can be expedited and all the data sent within the command or response packet. This limits the overall SDO transfer to just two packets. If more than 4 bytes of data is required to be transferred then a segmented transfer is used. Where after the first command or response packet, additional packets are sent and requested/acknowledged until all data is complete. There is also a Block transfer where instead of confirming each segment, the entire block is transferred then only one confirmation is made at the end.

As the various flags and handshakes are subtly different, details for each is provided.

**Expedited Read Dictionary Object**

Request Server -> Client  (0x600 + Node ID)

| **Can Header** | **rtr** | **len** | **Byte 0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x600 + node | 0 | 8 | Command | Index | | Sub Index | 0 | | | |

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 010 | CCS – Client Command Specifier |
| 4-0 | 00000 | Not used |

The overall command for this transfer is

|  |  |
| --- | --- |
| **Command Code** | **Meaning** |
| 0x40 | Read Dictionary Object |

Note! at this point you do not actually know it is an expedited transfer you have just requested to read a dictionary object, the server will confirm in the response if it can expedite the transfer. If the total data size is 4 bytes or less, the server will set the appropriate bits in the response and send the data with the response. if you have requested an object that is larger that 4 bytes the reply will have the expedited transfer bit set to 0 and then you must look at the segmented transfer

Response, Client -> Server (0x580 + Node ID)

| **Can Header** | **rtr** | **lrn** | **Byte 0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x580 + node | 0 | 8 | Command | Index | | Sub Index | Data | | | |

The index and Sub index are as you specified and the command byte has the following meaning:-

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 010b | SCS – Server Command Specifier |
| 4 | 0 | (Not Used) Segment toggle bit |
| 3-2 | (n) | Data size n=3 (11b) 1 data bytes sent n=2 (10b) 2 data bytes sent n=1 (01b) 3 data byte sent n=0 (00b) 4 data byte sent |
| 1 | 1 | expedited transfer |
| 0 | 1 | data set size is indicated |

|  |  |
| --- | --- |
| **Command Code** | **Meaning** |
| 0x43 | Read Dictionary Object reply, expedited, 4 bytes sent |
| 0x47 | Read Dictionary Object reply, expedited, 3 bytes sent |
| 0x4B | Read Dictionary Object reply, expedited, 2 bytes sent |
| 0x4F | Read Dictionary Object reply, expedited, 1 bytes sent |

**Expedited Write Dictionary Object**

Request Server -> Client  (0x600 + Node ID)

| **Can Header** | **rtr** | **len** | **Byte 0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x600 + node | 0 | 8 | Command | Index | | Sub Index | Data | | | |

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 001b | SCS – Server Command Specifier |
| 4 | 0 | (Not Used) Segment toggle bit |
| 3-2 | (n) | Data size n=3 (11b) 1 data bytes sent n=2 (10b) 2 data bytes sent n=1 (01b) 3 data byte sent n=0 (00b) 4 data byte sent |
| 1 | 1 | expedited transfer |
| 0 | 1 | data set size is indicated |

The overall command for this transfer is one of :

|  |  |
| --- | --- |
| **Command Code** | **Meaning** |
| 0x23 | Write Dictionary Object reply, expedited, 4 bytes sent |
| 0x27 | Write Dictionary Object reply, expedited, 3 bytes sent |
| 0x2B | Write Dictionary Object reply, expedited, 2 bytes sent |
| 0x2F | Write Dictionary Object reply, expedited, 1 bytes sent |

it is important that the the above data size is correct and the correct command code is used depending on the size of the object dictionary entry you are trying to write to. Can Open will enforce that only a 1 byte write can be performed to a 1 byte entry in the dictionary and the same goes for 1,2,3 and 4 bytes so make sure you use the correct size or an error will be returned

Response, Client -> Server (0x580 + Node ID)

The response packet again contains the index and sub index you specified and should look like:-

| **Can Header** | **rtr** | **len** | **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x580 + node | 0 | 8 | Command | Index | | Sub Index | 00 00 00 00 | | | |

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 011 | SCS – Server Command Specifier |
| 4-0 | 00000 | Not used |

**Read Dictionary Object (segmented)**

The read starts exactly the same way as documented in Expedited Read Dictionary Object :-

Request Server -> Client  (0x600 + Node ID)

| **Can Header** | **rtr** | **len** | **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x600 + node | 0 | 8 | Command | Index | | Sub Index | 00 00 00 00 | | | |

The difference is in the response

Response, Client -> Server (0x580 + Node ID)

| **Can Header** | **rtr** | **len** | **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x580 + node | 0 | 8 | Command | Index | | Sub Index | Len | | | |

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 010b | SCS – Server Command Specifier |
| 4 | 0 | Not Used |
| 3-2 | 00 | Not Used |
| 1 | 0 | expedited transfer |
| 0 | 1 | data set size is indicated |

 Instead of any data being returned with the response, the data bytes contain a 32bit length that specifies the total data size that needs to be transferred.

Then each data “segment” is requested and returned one at a time

Request Server -> Client  (0x600 + Node ID)

| **Can Header** | **rtr** | **len** | **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x600 + node | 0 | 8 | Command | Index | | Sub Index | 00 00 00 00 | | | |

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 011b | CCS – Client Command Specifier |
| 4 | 0/1 | Toggle bit, must be flipped each request (start with a 0) |
| 3-2 | 00 | Not Used |
| 1 | 0 | Not Used |
| 0 | 1 | Not Used |

Response, Client -> Server (0x580 + Node ID)

| **Can Header** | **rtr** | **len** | **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x580 + node | 0 | 8 | Command | Data | | | | | | |

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 000b | SCS – Server Command Specifier |
| 4 | 0/1 | Toggle bit, flipped each request |
| 3-1 | 000 | (number of data bytes ([8-n to 7]) that do NOT contain data) or zero if segment size not specified |
| 0 | 0/1 | 1 = Last segment |

**Write Dictionary Object (segmented)**

If uploading more than 4 bytes to a client a segmented transfer can be used

Request Server -> Client  (0x600 + Node ID)

| **Can Header** | **rtr** | **len** | **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x600 + node | 0 | 8 | Command | Index | | Sub Index | Len | | | |

Instead of sending any data bytes, the total length of the data is sent in the Len Field of the first packet

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 001b | CCS – Client Command Specifier |
| 4 | 0 | Not Used |
| 3-2 | 00 | Not Used |
| 1 | 0 | expedited transfer |
| 0 | 1 | data set size is indicated |

The client then responds with the following packet

| **Can Header** | **rtr** | **len** | **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x580 + node | 0 | 8 | Command | Index | | Sub Index | 00 00 00 00 | | | |

With command as follows:-

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 011 | SCS – Server Command Specifier |
| 4-0 | 00000 | Not used |

Then the handshake of the data packets(segments) begins

| **Can Header** | **rtr** | **len** | **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x600 + node | 0 | 8 | Data | | | | | | | |

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 011 | SCS – Server Command Specifier |
| 4 | 0/1 | toggle (change each packet, start with 0) |
| 3-0 | 0000 | Not used |

And the client confirms each segment with the following :-

| **Can Header** | **rtr** | **len** | **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x580 + node | 0 | 8 | Command | 00 00 00 00 00 00 00 | | | | | | |

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 001 | SCS – Server Command Specifier |
| 4 | 0/1 | toggle, changes each time |
| 3-0 | 0000 | Not used |

**Error/Abort codes**

When something goes wrong, or it is necessary to abort the command specifier in all cases can be changed to Abort Transfer. There are two abort messages one to the client and one from the client.

If the server needs to abort the transfer the packet is

| **Can Header** | **rtr** | **len** | **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x600 + node | 0 | 8 | Command | Index | | Sub index | Additional Info | | Error code | Error class |

If the client aborts the transfer the packet is

| **Can Header** | **rtr** | **len** | **B0** | **B1** | **B2** | **B3** | **B4** | **B5** | **B6** | **B7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x580 + node | 0 | 8 | Command | Index | | Sub index | Additional Info | | Error code | Error class |

The command bits are common to both packets for an abort

|  |  |  |
| --- | --- | --- |
| **Command Code Bits** | **Value** | **Meaning** |
| 7-5 | 100 | SCS/CCS – Abort |
| 4-0 | 00000 | Not used |

The Fields Add Info, Err code and Err class describe the reason for the abort. some errors need an Additional Code

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | **Error Class** | **Error Code** | **Additional Code** |
| Toggle bit not alternated | 5 Service Error | 3 Parameter Inconsistent | 0 |
| Command specifier not valid | 5 Service Error | 4 Illegal Parameter | 0 |
| Object does not exist | 6 Access Error | 2 Object non-existent | 0 |
| Attempt to read a write only Object | 6 Access Error | 1 Object access unsupported | 0 |
| Attempt to write a read only Object | 6 Access Error | 1 Object access unsupported | 0 |
| Index value is reserved for further use (00A0h-0FFFh and A000h-FFFFh) | 6 Access Error | 4 Invalid address | 0 |
| Access failed due to hardware | 6 Access Error | 6 Hardware fault | 0 |
| Sub-index does not exist | 6 Access Error | 9 Object attribute inconsistent | 11h |
| Object length too high | 6 Access Error | 7 Type conflict | 12h |
| Object length too low | 6 Access Error | 7 Type conflict | 13h |
| Data cannot be transferred /  Invalid signature | 8 Other Error | 0 | 20h |
| Parameter value out of range | 6 Access Error | 9 Object attribute inconsistent | 30h |
| Sub-parameter value out of range | 6 Access Error | 9 Object attribute inconsistent | 33h |
| Maximum value < Minimum value | 6 Access Error | 9 Object attribute inconsistent | 36h |
| Object cannot be mapped to PDO | 6 Access Error | 4 Invalid address | 41h |
| PDO length exceeded | 6 Access Error | 4 Invalid address | 42h |
| General internal incomptibility | 6 Access Error | 4 Invalid address | 44h |