CANopenIA-MGR-DLL

CANopen Instant Access to minimal Manager functionality

for version 1.0 or higher

This functionality is also implemented in the MGR firmware of the CANgineBerry.

Rev. 1.04 of 24th April 2019

Published by
Embedded Systems Academy GmbH
Bahnhofstraße 17
D-30890 Barsinghausen, Germany
www.esacademy.com

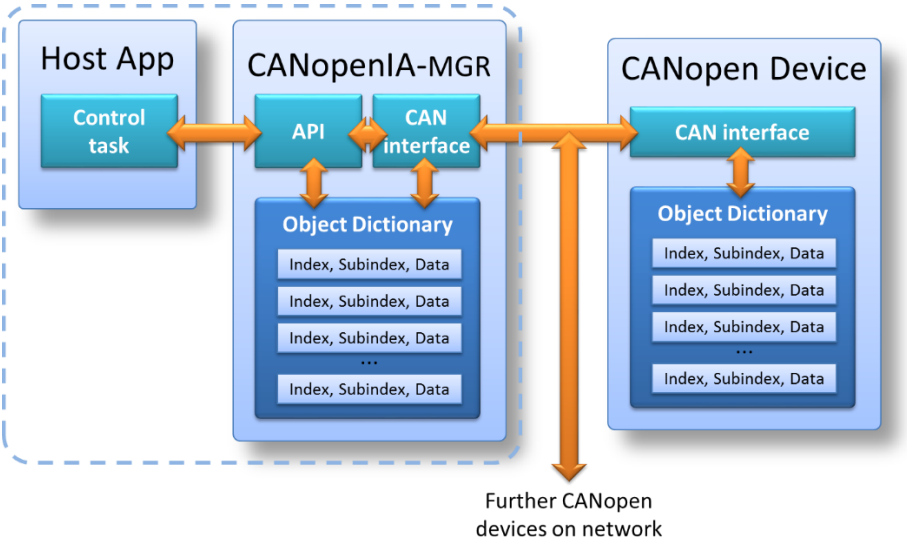
1 Contents

2 System Overview

The CANopenIA-MGR is a minimal CANopen Manager that gives the host quick and easy access to a network of CANopen devices.

2.1 CANopenIA-MGR: Minimal CANopen Manager

This library implements a minimal CANopen Manager. The configuration requirements are minimal, as the Manager auto detects the devices. It then scans all vital communication parameters and configures itself to consume and produce all matching CAN messages (CANopen PDO communication).



CANopenIA Manager Library

The host program receives simple serial indications with new data arriving (indicates from which node ID and which data object by index and subindex) and may use commands to send data (also based on node ID, index and subindex of data object) to the connected devices.

Note that only PEAK USB CAN interfaces are supported. If you need support for another type of CAN interface, please contact us.

2.2 CANopen Object Dictionary

As required by any CANopen device, the CANopenIA-MGR implements a CANopen Object Dictionary (OD) that contains all configurations of the chip itself as well as all the process data communicated. This OD is available to the CANopen network as well as to the host. Which OD entries are present in the CANopenIA depends on its configuration.

2.3 CANopen Manager

In CANopen, managers provide several functionalities. The ones provided by CANopenIA are listed in this section.

SDO Client

The CANopenIA-MGR supports SDO client services. Once such a CANopenIA device is up and running (CANopen state operational), it may send CANopen SDO (Service Data Object) read and write requests to the nodes connected to the CANopen network. This gives the host application read and write access to all the Object Dictionaries of all connected nodes.

Note that in regular CANopen this means that this device uses the regular SDO client channels used by a CANopen Manager. DO NOT use this mode, when another CANopen Manager is present and using these channels at the same time.

NMT Master and Heartbeat monitoring

The CANopenIA-MGR software also provides the CANopen NMT (Network Management) Master functions to control the individual nodes connected. The manager can autostart known devices to facilitate a quick start up of CANopen systems.

A default heartbeat time and timeout monitoring can be automatically activated. If devices are lost (no more heartbeat received), the master automatically transmits a reset request to these nodes for automated recovery support.

Automated PDO handling

The minimal CANopen manager supports an automated PDO (Process Data Object) handling. PDO configurations of connected devices are analyzed and activated. The host is informed about every PDO received from all devices. The information passed on to the host for data received includes node ID, object info (index, subindex) and the data.

3 API Overview

Two APIs are provided, an object-orientated (OO) API for C++ users and a “flat” API for C and Java users. The functionality provided is identical. JNA can be used from Java to access the library functions.

The OO API is divided up into two classes, called CANopenIAMgr and SerialProtocol.

Class	Description
CANopenIAMgr	Handles initialization and de-initialization of the library. Must always be called when starting to use the library and after use of the library is finished.
SerialProtocol	Provides access to the manager functionality, for example accessing the object dictionary of the manager or other nodes on the network, resetting the manager, sending network management requests, etc.

The flat API uses the same division however the class names are part of the function name. Here is an example.

API	Example Function Call
OO	CANopenIAMgr::Start()
Flat	C_CANopenIAMgr_Start()

The flat function names are formatted as follows:

```
C_<classname>_<functionname>
```

This allows any function to be easily transposed from one API to the other.

3.1 Initialization

The following sequence is used to initialize the manager.

C/Java:

```
C_CANopenIAMgr_Init(Bitrate, NodeId);
C_CANopenIAMgr_Start();
C_SerialProtocol_Init();
C_SerialProtocol_Connect();
```

C++:

```
CANopenIAMgr *Mgr = new CANopenIAMgr(Bitrate, NodeId);
Mgr->Start();
SerialProtocol *SerialProto = new SerialProtocol();
SerialProto->Connect();
```

3.2 DeInitialization

The following sequence is used to deinitialize the manager.

C/Java:

```
C_SerialProtocol_Disconnect();
C_SerialProtocol_DeInit();
C_CANopenIAMgr_Stop();
C_CANopenIAMgr_DeInit();
```

C++:

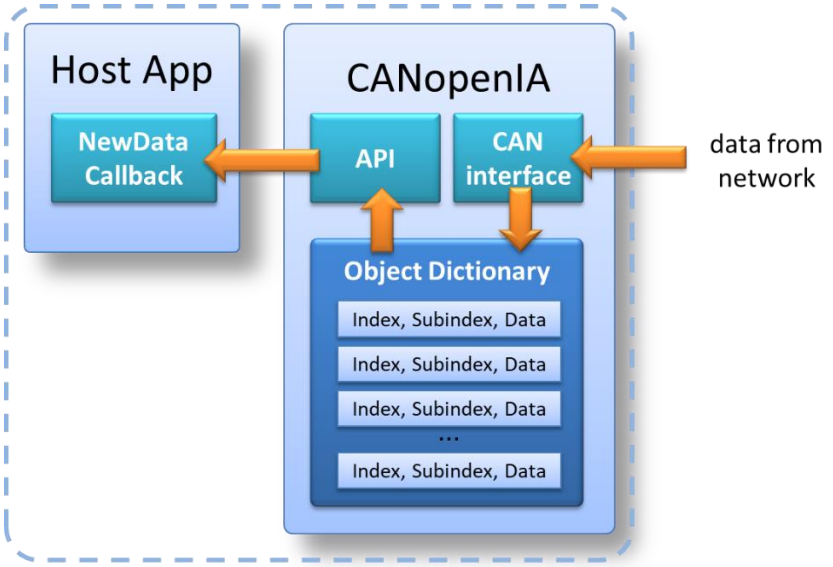
```
SerialProto->Disconnect();
delete SerialProto;
Mgr->Stop();
delete Mgr
```

4 API Function Summary

This section summarizes the functionality provided by the CANopenIA-MGR-DLL. For more details see the SerialProtocol.h header file.

4.1 New Data arrived indication

New process data arrived from the CANopen network. The node ID of the sender (if known), the Object Dictionary entry in question and the new data is part of this indication.



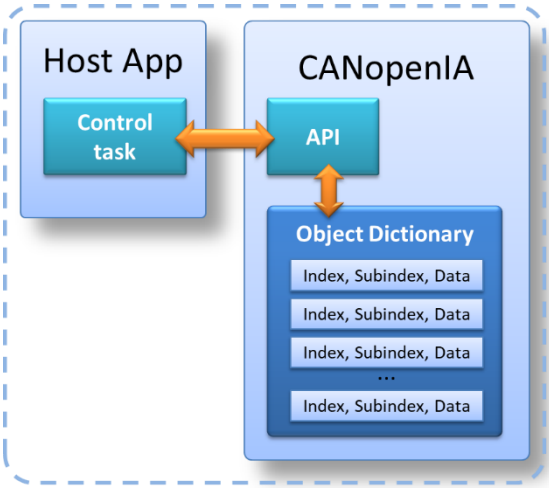
Automatic indication of new data written to Object Dictionary

Language	Prototype
C	`void NewData(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED16 length, UNSIGNED8 *data);`
C++	`static void NewData(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED16 length, UNSIGNED8 *data);`
Java	`public static void NewData(byte NodeID, int Index, byte Subindex, int DataLength, Pointer Data, Pointer Param);`

Parameter	Description
nodeid	The ID of the node sending the data
index	Index of the object dictionary entry in the node
subindex	Subindex of the object dictionary entry in the node
length	Length of data data
data	The data

4.2 Write to a local Object Dictionary entry command

Writes data to one local Object Dictionary entry. Data size is indicated via length field of lower communication layer (see message definition).



Read or write command to local Object Dictionary

Language	Prototype
C	`UNSIGNED32 C_SerialProtocol_WriteLocalOD(UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`
C++	`UNSIGNED32 SerialProtocol::WriteLocalOD(UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`
Java	`long C_SerialProtocol_WriteLocalOD(short index, byte subindex, int datalength, Pointer data)`

Parameter	Description
index	The index of the object dictionary entry to write to
subindex	The subindex of the object dictionary entry to write to
datalength	Number of bytes to write
data	Data to write

The return value is ERROR_NOERROR or an error code, ERROR_xxx.

4.3 Read from a local Object Dictionary entry command

Request to read data from one Object Dictionary entry. Data size is indicated via length field of lower communication layer.

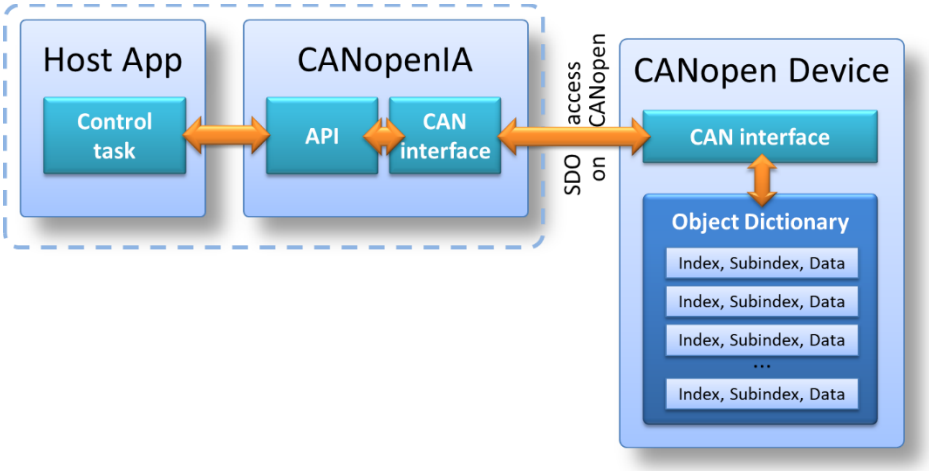
Language	Prototype
C	`UNSIGNED32 C_SerialProtocol_ReadLocalOD(UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 *datalength, UNSIGNED8 *data)`
C++	`UNSIGNED32 SerialProtocol::ReadLocalOD(UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 *datalength, UNSIGNED8 *data)`
Java	`long C_SerialProtocol_ReadLocalOD(short index, byte subindex, Pointer datalength, Pointer data)`

Parameter	Description
index	The index of the object dictionary entry to read from
subindex	The subindex of the object dictionary entry to read from
datalength	When called set to the maximum number of bytes to read. On return holds the number of bytes read
data	Filled with read data

The return value is ERROR_NOERROR or an error code, ERROR_xxx.

4.4 Write to a remote ObjectDictionary entry command

Writes data to one Object Dictionary entry of a remote node (using SDO client access). Data size is indicated via length field of lower communication layer



Read or write request to a remote Object Dictionary of a node on the network

Language	Prototype

C	`UNSIGNED32 C_SerialProtocol_WriteRemoteOD(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`
	
	`UNSIGNED32 C_SerialProtocol_WriteRemoteODExtended(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`

C++	`UNSIGNED32 SerialProtocol::WriteRemoteOD(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`
	
	`UNSIGNED32 SerialProtocol::WriteRemoteODExtended(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`

Java	`long C_SerialProtocol_WriteRemoteOD(byte nodeid, short index, byte subindex, int datalength, Pointer data)`
	
	`long C_SerialProtocol_WriteRemoteODExtended(byte nodeid, short index, byte subindex, int datalength, Pointer data)`

Parameter	Description
nodeid	The ID of the node to write to
index	The index of the object dictionary entry to write to
subindex	The subindex of the object dictionary entry to write to
datalength	The number of bytes to write
data	Data to write

The “extended” versions are non-blocking. They return immediately and on completion of the write the SDO Request Complete callback function is called.

The return value is ERROR_NOERROR or an error code, ERROR_xxx.

Language	Prototype
C	`void SDORequestComplete(UNSIGNED8 nodeid, UNSIGNED32 result);`
C++	`static void SDORequestComplete(UNSIGNED8 nodeid, UNSIGNED32 result);`
Java	`public static void SDORequestComplete(byte NodeID, int Result);`

Parameter	Description
nodeid	The ID of the node that was written to
Result	The result of the write operation, SDOERR_OK or SDOERR_xxx

4.5 Read from a remote Object Dictionary entry command

Request to read data from a remote Object Dictionary entry (using SDO client access, upload).

Language	Prototype
C	`UNSIGNED32 C_SerialProtocol_ReadRemoteOD(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 * datalength, UNSIGNED8 *data)`

	UNSIGNED32 C_SerialProtocol_ReadRemoteODExtended(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 * datalength, UNSIGNED8 *data)
C++	UNSIGNED32 SerialProtocol::ReadRemoteOD(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 *datalength, UNSIGNED8 *data) UNSIGNED32 SerialProtocol::ReadRemoteODExtended(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 *datalength, UNSIGNED8 *data)
Java	long C_SerialProtocol_ReadRemoteOD(byte nodeid, short index, byte subindex, Pointer datalength, Pointer data) long C_SerialProtocol_ReadRemoteODExtended(byte nodeid, short index, byte subindex, Pointer datalength, Pointer data)

Parameter	Description
nodeid	The ID of the node to read from
index	The index of the object dictionary entry to read from
subindex	The subindex of the object dictionary entry to read from
datalength	When called set to the maximum number of bytes to read. On return holds the number of bytes read
data	Filled with read data

The “extended” versions are non-blocking. They return immediately and on completion of the read the SDO Request Complete callback function is called.

The return value is ERROR_NOERROR or an error code, ERROR_xxx.

See “Write to a remote Object Dictionary entry command“ for details of the callback function.

5 Minimal Manager SDO & PDO Handling

The CANopenIA Minimal Manager version simplifies how an application uses CANopen communication. Here all data is only referred to by a node ID and the Object (Index and Subindex) to address ab object in a node's object dictionary.

Using the write and read to a remote object dictionary functions, the host system can read and write all objects in a network.

In addition, the host system receives event notifications, if data came in from a remote object. Again, referred to by the node ID the data comes from and the object dictionary entry (Index and Subindex).

If your application is generic and does not require optimized communication (e.g. to lower the bus load for communication or achieve shorter reaction times), the this is all you need to know.

5.1 Communication options

The default CANopen communication mode used by the CANopenIA-MGR is the SDO communication (Service Data Objects). Here the manager sends one read/write request for a single object of a node and receives one response.

Internally, the manager scans detected devices for their PDO (Process Data Object) configuration. The scanned information is used by the minimal manger to configure itself for receiving all Transmit PDOs transmitted by the devices and for transmission of all Receive PDOs to the devices.

Receiving TPDO data from the devices

The application requires no knowledge about the Transmit PDO configuration of the devices. Once self-configured, the minimal manger receives all PDOs generated by the devices and converts them into the corresponding "New Data" indication events towards the host or application. The application automatically receives all PDO data.

For each object received, the host/application is informed about:

- The node ID which sent the data
- Which Object of that node was received (Index/Subindex)
- The data itself

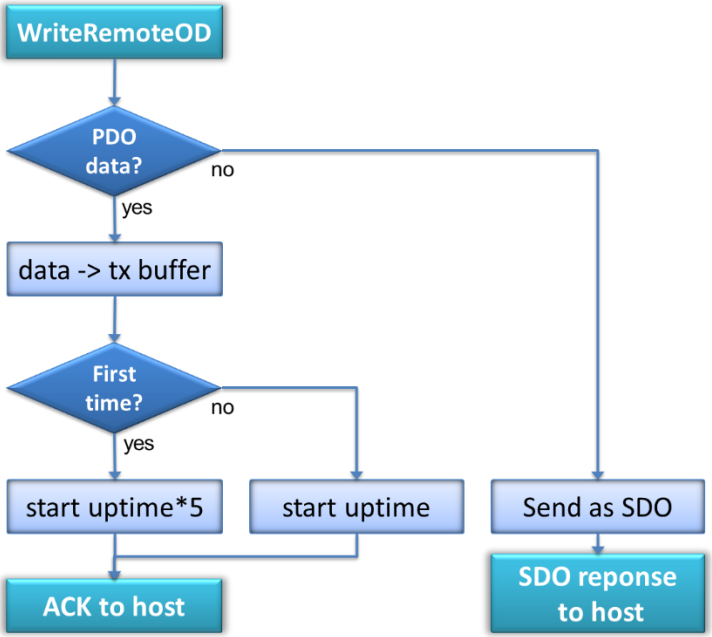
Sending data to the devices

The application addresses the data in the same fashion as for received data. It uses the WriteRemoteOD functionality and informs the CANopenIA Manager about:

- The node ID to which the data needs to be send
- Which Object of that node is it going to (Index/Subindex)
- The data itself

The manager automatically determines if this data can be send by PDO or if a SDO needs to be triggered. As PDOs can have multiple objects mapped (multiple object contained in one CAN message) all mapped items must be written at least once, before the PDO can be transmitted by the manager. This is required to prohibit transmission of uninitialized data/commands to a CANopen device.

We recommend that once the application receives the call-back that a mode has been scanned, it writes once to all objects of that device that can be written to, to ensure all data has been initialized.



PROCESSING REMOTE DATA WRITE REQUESTS

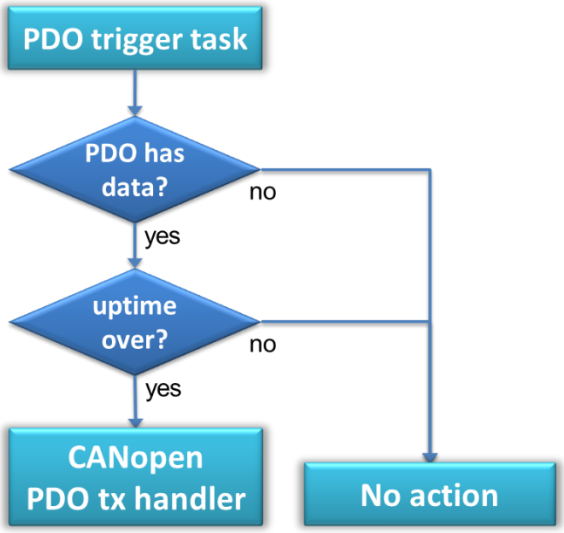
The flow chart above illustrates how the manager processes data write requests to nodes on the network. If the data written is not part of any PDO, it gets written to the

node using a SDO client write access. Once the SDO response comes back from the node, this response is passed on to the host.

Otherwise the data is copied to the appropriate buffer and the PDO update timer is re-started with every data write to this PDO. The PDO gets queued/triggered ready for transmission once the PDO update timer expires.

Internally, the CANopenIA system continuously checks if a PDO requires transmission (see next flow chart “PDO trigger task”.

PDOs are only processed for transmission, if data is available and the update time expired.



PDO TRIGGER TASK

If both the default event and inhibit times (objects [5F01h,05h] and [5F01h,06h], see section 6.2) are zero, then a PDO is triggered for transmission whenever the update time expires (time since last write by the host) or the last mapped entry has been written.

Once set (non-zero), the event and inhibit times work as defined in CANopen:

If the inhibit time is set (non-zero) and the event time is zero, then any repetitive transmission is only transmitted, if the time since last transmission is at least as long as the inhibit time.

If the event time is set (non-zero) and the inhibit time is zero, then the PDO is transmitted cyclically, no matter if the data has been updated by the application or not.

If both times are set, then they are combined. If triggered by host (expiration of update time), then the PDO gets transmitted, observing the inhibit time. Without host triggering, the PDO is transmitted cyclically based on the event time.

6 Object Dictionary entries in the manufacturer specific area

The manufacturer specific area of the Object Dictionary provides direct access to configuration data. These can be accessed using the read and write local commands. Syntax used in listing below:

Name

[index,subindex] (data type, access type)
Description

6.1 CANopenIA Device Status

The entries in this section give the host access to the current state of the local CANopenIA device. All these entries are read-only.

Device status: own node ID

[5F00h,01h] (UNSIGNED8, RO)
The node ID of the local CANopenIA device

Device status: own NMT state

[5F00h,02h] (UNSIGNED8, RO)
The current CANopen state of the local CANopenIA device. See section 0for a list of all defined states.

Device status: own HW state

[5F00h,03h] (UNSIGNED8, RO)
Bit: 0: INIT – set after a completed initialization
1: CERR – set, if a CAN bit or frame error occurred
2: ERPA – set, if a CAN "error passive" occurred
3: RXOR – set, if a receive queue overrun occurred
4: TXOR – set, if a transmit queue overrun occurred
5: CANFD – set, if CAN hardware supports CAN FD
6: TXBSY – set, if Transmit queue is not empty
7: BOFF – set, if a CAN "bus off" error occurred

Device status: own HW/FW mode

[5F00h,04h] (UNSIGNED32, RO)
Bit: 0..7: 00h: Custom hardware
01h: CANgineBerry

02h: CANgineLight
03h: CANgineBT
04h: PCAN-RS232
05h: PCAN-xxx with PCAN-Basic API

Bit: 8..15: 00h: Custom firmware
01h: CANopenIA Device
02h: CANopenIA Manager
03h: CANopenIA 447izer

Bit: 16..23: Firmware major version

Bit: 24..31: Firmware minor version

Chip serial number (where available)

[5F00h,05h] (UNSIGNED128/DOMAIN, RO)

The serial number of the microcontroller hosting the CANopenIA software.

6.2 CANopenIA Device Control

The entries in this section can be written to and allow the host to actively control the local CANopenIA device or manager.

Device control: Reset

[5F01h,01h] (UNSIGNED8,WO)
Reset the CANopenIA chip, module or library. Writing129 issues a soft reset, 130 a hard reset.

Device control 447: Sleep Objection

[5F01h,02h] (UNSIGNED8,RW)
Activate the CiA 447 sleep objection (set to 1 to object).

Device control 447: Ignore PDOs from VD

[5F01h,03h] (UNSIGNED32,RW)
For If a bit is set in this value, then PDOs coming from the corresponding virtual device (see vdfg number in CiA-447) are ignored. For example: set bit 7 to ignore all PDOs coming from GPS devices.

Manager control (manager only)

[5F01h,04h] (UNSIGNED32,RW)

Bit: 0: KEEP_OP - set to keep nodes operational
(will send appropriate NMT command automatically)
1: HB receive all - set to activate automated heartbeat monitoring

(default HB times below are used)
2: PDO receive all - set to activate automated device TPDO handling (scan devices for their transmit PDOs and receive them all)
3: PDO transmit all - set to activate automated device RPDO handling (scan devices for their receive PDOs and produce them all)
4: Use scanned entries – set to activate caching of scanned entries. If requested by host, reply from cache.
5: Enforce remote write SDO – set to enforce SDO write access when. writing to an OD entry of a remote node, do not send as TPDO.
6: Manual TPDO trigger – set to activate manual transmit PDO triggering (use [5F01h,0Ch] to trigger)
7-15: Reserved
16-22: Number of nodes supported for heartbeat monitoring and SDO client handling
23: Reserved
24-30: Number of nodes supported for automated PDO handling
31: Reserved

Default heartbeat producer time (manager only)

[5F01h,05h] (UNSIGNED16,RW)
Use this default event time (in milliseconds) for all PDO transmissions by the manager.

Default heartbeat consumer time (manager only)

[5F01h,06h] (UNSIGNED16,RW)
Use this default event time (in milliseconds) for all PDO transmissions by the manager.

Default PDO update time (manager only)

[5F01h,07h] (UNSIGNED8,RW)
When the manager updates PDO transmission data, this update timeout is started before triggering the PDO for transmission. This allows the application to update all objects of a PDO before its transmission is triggered. Note that this time is not used, when the PDO event time (see below), is non-zero.

Default PDO transmission event time (manager only)

[5F01h,08h] (UNSIGNED16,RW)
Use this default event time (in milliseconds) for all PDO transmissions by the manager.

Default PDO transmission inhibit time (manager only)

[5F01h,09h] (UNSIGNED16,RW)
Use this default inhibit time (in 100th of microseconds) for all PDO transmissions by the manager.

Manager re-scan device (manager only)

[5F01h,0Ah] (UNSIGNED8,WO)
Writing a node ID to this entry re-triggers the auto-scan mechanism for this node. The manger will start a new node scan for this device.

Bit: 0-6: Node ID to scan
7: reserved

Device and Manager generic CAN Rx / Tx

[5F01h,0Bh] (UNSIGNED8,RW)

This feature enables the support of generic CAN messages, not handled by the local CANopen implementation.

Bit: 0-3: Generic CAN transmit
0: disabled
1: Condensed access via object 5F0Ch
4-7: Generic CAN receive
0: disabled
1: Condensed access via object 5F0Ch

CAN messages received, that are not processed by the local CANopen task are passed on to the host as a write to the local Object Dictionary entry 5F0Ch:

- Node ID: 0
- Index: 5F0Ch
- Subindex: Length of CAN message in bytes (0-8)
- Len: 2 + Length of CAN message in bytes (0-8)
- Data: First 2 byte: CAN ID
  Followed by the data bytes of the CAN message

To transmit a generic CAN message, execute a write to the local object dictionary entry 5F0Ch:

- Index: 5F0Ch
- Subindex: Length of CAN message in bytes (0-8)
- Len: 2 + Length of CAN message in bytes (0-8)
- Data: First 2 byte: CAN ID
  Followed by the data bytes of the CAN message

Testing with the COIA utility for the CANgineBerry

Activate the feature by a write to [5F01h,0Bh]:

```
-w 0x5F01,0x0B,0x01,0x11
```

To transmit a generic CAN message, use the “--tx-can” parameter, passing the CAN ID, the length and the data bytes:

```
--tx-can 0x150,4,0x11,0x22,0x33,0x44
```

This produces a 4-byte CAN message with ID 150h and the four data bytes 11h to 44h.

Use the monitoring “-m” parameter to monitor incoming data and messages.

Manual PDO trigger (manager only)

[5F01h,0Ch] (UNSIGNED16,WO)

Manually trigger the transmission of a PDO. This is the only trigger method, if the automatic PDO triggering mechanism is disabled ([5F01h,04h] bit 6).

To trigger a PDO, write node ID and PDO number into this entry. The PDO triggered is the one matching the Receive PDO of that node.

Bit: 0-7: PDO number, starting at 1
8-14: Node ID, starting at 1
14: Reserved

Testing with the COIA utility for the CANgineBerry

Node number 5 is a digital I/O node (CiA 401)

```
-w 0x5F01,4,4,0x0000003F
--node-write 5,0x6000,1,1,0x55
--node-write 5,0x6000,2,1,0x66
--node-write 5,0x6000,3,1,0x77
--node-write 5,0x6000,4,1,0x88
-w 0x5F01,0x10,2,0x0501
```

In first line we enable automatic TPDO handling, the next four lines write data into the PDO and the last line triggers the PDO (node 5, PDO 1).

6.3 Status of all nodes

Only available with CANopenIA-MGR and 447 versions.

Last known state of Node 1

[5F04h,01h] (UNSIGNED8, RO)
The last known state of node 1, see list below for all defined values.

Last known state of Node X

[5F04h,X] (UNSIGNED8, RO)
The last known state of this node (allowed range 1 to 127), see list below for all defined values.

The following values are defined:

NODESTATUS_BOOT	0x00
NODESTATUS_STOPPED	0x04
NODESTATUS_OPERATIONAL	0x05
NODESTATUS_PREOP	0x7F
NODESTATUS_EMCY_NEW	0x80
NODESTATUS_EMCY_OVER	0x81
NODESTATUS_HBACTIVE	0x90
NODESTATUS_HBLOST	0x91
NODESTATUS_SCANSTARTED	0x9F
NODESTATUS_SCANCOMPLETE	0xA0
NODESTATUS_SCANABORTED	0xA8
NODESTATUS_RESETAPP	0xB0
NODESTATUS_RESETCOM	0xB1
NODESTATUS_SLEEP	0xF0
NODESTATUS_BOOTLOADER	0xF1

6.4 NMT Master Message

Only available with CANopenIA-MGR version. An NMT Master message can be triggered by writing to [5F0Ah,01h].

Transmit NMT (manager only)

[5F0Ah,01h] (UNSIGNED16, WO)
The high byte contains the destination node id (1-127) or zero for “all” nodes.
The low byte contains the NMT command:

01h: Switch to operational state
02h: Switch to stopped state
80h: Switch to pre-operational state
81h: Execute an application reset
82h: Execute a communication reset

6.5 Manager: Automatic Node Scan

In CANopen Manager or CiA 447 mode, the device automatically scans nodes found on the network for often used entries. This data is available, as soon as a node's state is reported as NODESTATUS_SCANCOMPLETE.

If caching is enabled in the Manager Control word (Object [5F01h,04h]), then the CANopenIA device will return the pre-scanned entries without re-requesting these from the device via CANopen.

Example: If the host requests the object [1018h,1] (vendor id) from node 3 by sending the ReadRemoteOD command, then the CANopenIA will directly reply with the value, if it is in the local cache.

7 C++ Programming Example

The C++ example can be found in the folder TestHarness-C. The project file provided is for Visual Studio 2017.

The example demonstrates starting the manager, resetting all nodes, reading from the manager object dictionary, reading from a node's object dictionary and transmission of real-time data.

8 Java Programming Example

The Java example can be found in the folder TestHarness-Java. The project provided is for Netbeans 8.2.

The example demonstrates starting the manager and reading from the manager's object dictionary.

The example also demonstrates how JNA can be used to access the library from Java. A complete declaration of the API is provided wrapped in a class called "ManagerLib". Here is an example of calling a function:

```
ManagerLib MgrLib = ManagerLib.INSTANCE;
MgrLib.C_CANopenIAMgr_Init(125, 0x40)
```