TECHNOTES - Technology Note

March 2008

CANopen, die Qual der Wahl (the agony of choice)

CAN is rugged, reliable and cheap to embed in devices. The whole world uses CAN, often even without knowing it. You can find Controller Area Networks in rail, road and off-road vehicles, ships, medical devices, production machines, vending machines, theaters... everywhere. In 'closed' systems, CAN can be tuned to do whatever you want. However, if you want to connect devices from different manufacturers together, they need to speak the same CAN language, or Higher Layer Protocol (HLP). There are many of those, and some have local dialects as well.

Omron has chosen to promote CAN with the DeviceNet HLP as its main protocol. This HLP was developed specifically for Industrial Automation. There is no protocol named CANbus, although many people use the term as if arbitrary CAN devices can be connected to a single bus and exchange meaningful data. There is a protocol called CANopen. The "open" does not mean that a CANopen device is able to communicate to any other CAN device. CANopen is just another HLP similar to DeviceNet, but with its own message definitions, device profiles and regulations.

CANopen has a lot of fans but mainly in the area where dedicated special made controllers are used. The popularity of CANopen in this area causes others to start looking at it and sometimes state that it is their choice of network, without really understanding the differences between CANopen and other CAN-based HPLs, such as DeviceNet. In this article, I would like to explain why CANopen is popular, but also point out the pitfalls.

CANopen specification.
The **CANopen** specification is managed by CAN in Automation, or **CiA** (www.can-cia.de), which is the international users' and manufacturers' group that develops and supports CANopen and other CAN-based, higher-layer protocols. The non-profit group was founded in 1992 to provide CAN-based technical, product and marketing information. Approximately 500 companies are member of the non-profit group that is headquartered in Erlangen, Germany.
For CANopen, CiA manages the specification, which is described in so called profiles. They also provide conformance testing and carry out promotional activities.

Device and Application profiles.
The behaviour of a CANopen device is described in so-called profiles. It is not the case that a complete device is described with one profile. There is, for instance, a profile for the cabling, a profile for the connectors to use, a profile for the basic communication, and a profile for the device behaviour.
If we take, for instance, a standard drive, it adheres to the profiles:

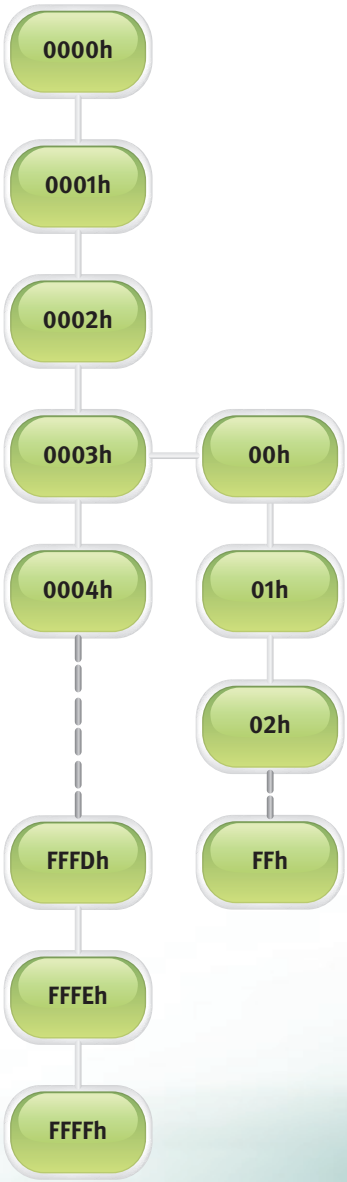
- CiA 102 DS V2.0 CAN physical layer for industrial applications.
- CiA 301 DSP V4.02 CANopen application layer and communication profile.
- CiA 402-1 DSP V3.0 CANopen drives and motion control device profile-Part1: General definition.
- CiA 402-2 DSP V3.0 CANopen drives and motion control device profile-Part2: Operation modes and application data.
- CiA 402-3 DSP V3.0 CANopen drives and motion control device profile-Part3: PDO mapping.

You're still with me? And if you wonder what the DS and DSP stand for, DS is Draft Standard and DSP is Draft Standard Proposal. Yes, it all is still in draft. If you look at all of the specifications none of them is final.

The heart of a CANopen device.
All the parameters and application values in the device are stored in an Object Dictionary. It is a tree-structure that can be browsed with an Index. An object can be a single place for holding data but it can also be an array or a structure. This means that there are other objects behind the initial ones. These other objects can be addressed through a sub-Index.

Specific areas have a specific function. One shows the device information while another initializes the CAN-communication. If the device does not comply with a defined profile then there is a manufacturer-specific area to store application-specific variables. And then there are the device-profile and interface-profile specific areas as they are specified in all the different profiles.

Index	Description
1000h - 1FFFh	Communication object area
2000h - 5FFFh	Manufacturer specific area
6000h - 9FFFh	Device profile specific area
A000h - BFFFh	Interface profile specific area



Reading and writing to the Object Dictionary

To manipulate the data in an object in the Object Dictionary, Service data objects (SDOs) are defined. One SDO consists of two CAN data frames with different CAN identifiers. This is a confirmed communication service. With an SDO, a peer-to-peer client-server communication between two CANopen devices can be established. The owner of the accessed object dictionary acts as a server of the SDO. The device that accesses the object dictionary of the other device is the SDO client. SDO communication is what is commonly called message communication. The addressed object is defined in the SDO message whether it is a read or a write command. And sometimes the data transferred is more than a single frame can hold. In which case the transfer is done in segments.

I/O-communication or PDO.

The difference between message and I/O communication is that the former has the complete information about what object is addressed in the message. With I/O communication on the other hand, both the sender and the receiver know exactly what bit, byte or word represents what. There is only data in the message. This makes this I/O-communication very efficient. In CANopen this I/O communication is called Proces Data Objects or PDO

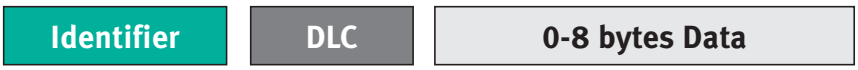
But with PDO communication, CANopen has some nice features. The nature of CAN communication is that it is a broadcast network. A node sending a message is heard by all the other nodes. With one message you can reach all of them. This works very nicely if you want to, let's say, have a couple of drives and want them to start at the same moment. You just set all the drives to react to the same message. Sounds easy, doesn't it?

It is here where the nice features of CANopen begin, but you must understand them well and do some bookkeeping to keep track of what data is going from which source to which destination. For those who are used to the master/slave principle of network operation, this may be difficult to grasp. But for the not faint of heart it is a lovely way to tune a network.

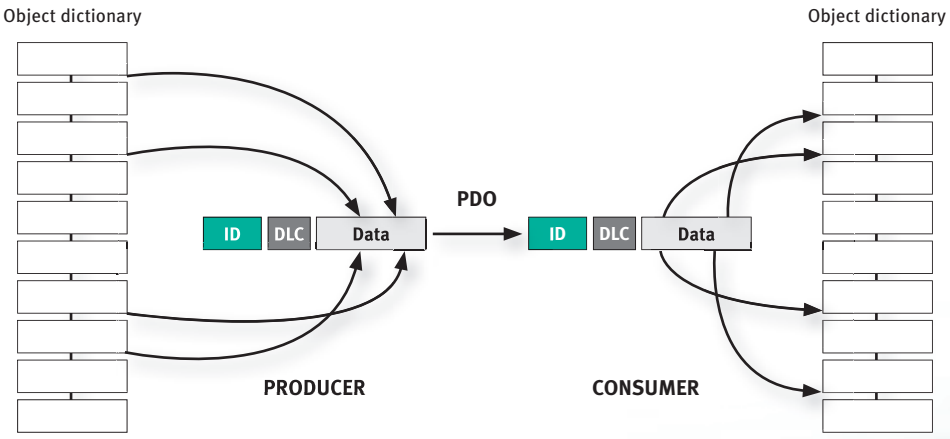
Some principles

To explain how this works first a little CAN theory. A CAN message consists of three parts. There are more, but they are for the protocol handling and do not carry any application data. The three parts are:

1 Identifier. This uniquely defines a message. The golden CAN rule is that an identifier may only be produced by one node.
2 Data Length Code or DLC. It is a number from 0 to 8 telling the number of data-bytes in the data part. And yes, also a message with no data, as DLC is 0, is a valid message.
3 Data. 0 to 8 bytes of data.



There is a mechanism in each CANopen device that composes the message from the application data in the Object Dictionary to a CAN message to be produced on the network. And vice-versa, a message consumed is taken apart to transfer the proper data to the Object Dictionary application objects.



This mechanism is called mapping of objects. But before this works, it must be set up. Also the producer and the consumer must be set to use the same identifier. Note that it is not defined whether or not a value coming from a certain object in a producer is mapped to an equivalent object in a consumer.

Maybe it feels strange that there is no mentioning of master or slave, but only producer and consumer. But this is actually what the CANopen concept is all about. One node sends out a message (produces) and one or more nodes can be interested in this message (consumes). And in each and every device that consumes, it can use or not use parts of the message received. If it only needs the last byte of the data, it is allowed to do so. It consumes the whole message, but discards the first 7 bytes of data.

The data can now be composed, put on the network, received and unpacked. But when is a PDO send out? Again there are several settings possible. In general there are four possibilities:

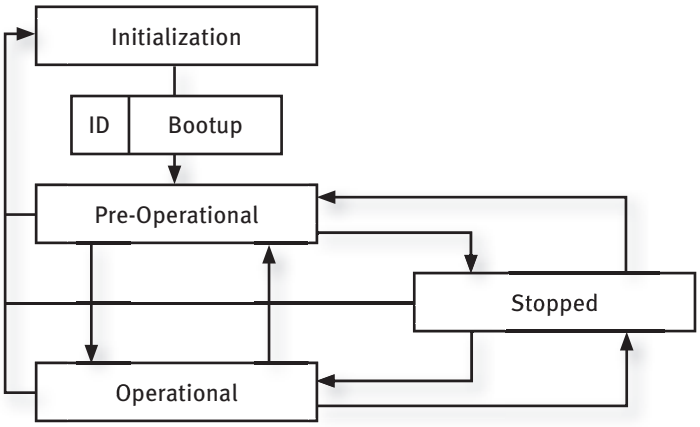
1. Event-driven PDO. The value of one of the objects changes and the device will produce the data on the network. There is a setting for an inhibit time to prevent that a rapidly changing value will cause a flood of messages on the network.
2. Timer-driven PDO. If time elapses the data is produced.
3. Remotely requested PDO. CAN has defined a so-called Remote Transmission Request (RTR). With this RTR frame, the owner of an equal ID to the RTR frame is requested to produce its data. However, the implementation of RTR by CAN-controller vendors is often in such a way that it is not guaranteed that the PDO is really valid. Therefore the CiA discourages the use of RTR.
4. Synchronous PDO transmission. The device produces the PDO on a so-called Sync message. This is a message with ID 80h and no data (DLC=0).

And of course all of this has to be set up properly.

An example of this Producer/Consumer communication is an encoder and a motion controller. The encoder produces the position on the network. And the motion controller just picks it up and uses it as input for its motion control. And maybe another device is also interested in the position. It could be a display that just shows the actual position.

Fire up the network.

With other networks, we are used to them working when power is applied. This is not the case in CANopen. Every device has its own mandatory CANopen state machine. It is controlled by so-called Network Management (NMT) messages. It has a fixed ID and the data defines whether it is directed to one device or to all devices and to what state the device(s) should go.



When powered up, the device comes directly in the initialization state. It checks its internals and configures the CAN controller. Then it sends out a boot-up message telling the other devices on the network “Hello, I’m alive”. It is now in the Pre-Operational state and is ready to be configured through SDO communication. Maybe this configuration is not needed and the device can be brought to the Operational state by another device. It needs to receive a “Start node” or “Start all nodes” command. In the Operational mode, the device will produce and consume PDO’s.

Help, I have a problem!

If something is wrong with a device, it has the capability to send out an emergency message (EMCY). This EMCY message is a broadcast and another device then has to consume it to take countermeasures. EMCY–message data consists of a fixed part of three bytes and a five byte manufacturer-specific part. That leaves the vendor of the device enough space to implement specific error signals.

An EMCY message is only sent out once at its occurrence and as long as there are no changes, in which case no other EMCY messages are sent out. Also a return to normal will cause an EMCY message to be sent out signaling that all is normal again.

EMCY is a broadcast and is not confirmed. So the device with the problem will not know that its cry for help has been heard and that countermeasures have been taken. . It could also be that there is no response and the device is left alone on its island in the Pacific.

Are you still there?

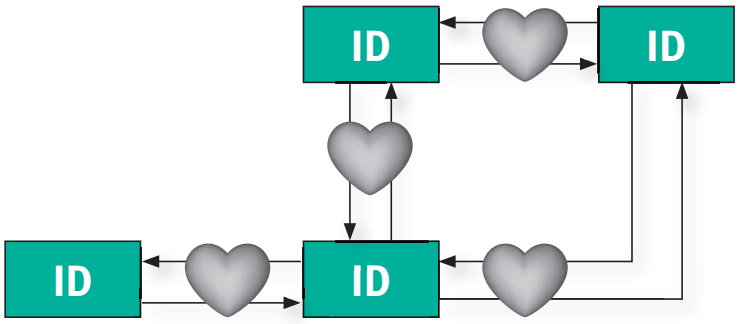
In firing up the network, we saw that a device sends out a boot-up message telling the other devices it is there. From there it goes into the Pre-Operational state and later is put into Operational state. But when in the Pre-Operational state or when a device is in the Operational state and it is set to send out data on change, there is no way to check it is still alive as it does not produce data all the time.

Therefore, two mechanisms are defined to let other nodes know that the device is still alive. These are:

1 Node/Lifeguarding
2 Heartbeat.

Node/Lifeguarding uses the RTR frame to trigger the device to send out its status. But, as mentioned before, the implementation of the RTR principle in CAN controllers is sometimes not how it should be. CiA therefore strongly discourages use of this method and recommends using Heartbeat instead. However, as the first versions of the spec only described Node/Lifeguarding, there are still a lot of devices that use Node/Lifeguarding. It could also be your device of choice. So be aware.

Heartbeat is the other principle. Here the device just starts to send out its heartbeat message at a set interval. This interval needs to be set up in the Object Dictionary. If the value is 0, then no heartbeat messages will be transmitted by the device.



A device can be a heartbeat producer. But it can also be a heartbeat consumer. For instance, two devices cooperate together and they should be constantly aware the other is still active. Although I/O-data is sent when a change occurs, with a regular heartbeat message the state of the other is always known by the device. And of course, being a heartbeat consumer must also be configured. What should be done if a heartbeat is not received anymore depends on the application. It is not described in the CANopen specifications.

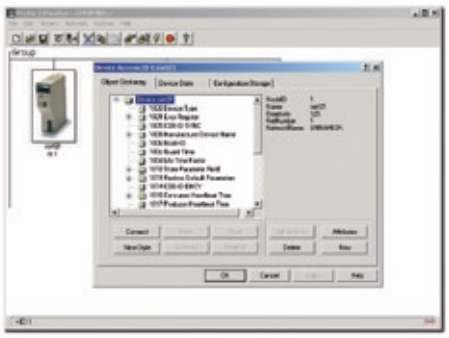
A real live example of a device that would only send out PDO data on a change was a joystick. By default, the heartbeat was not enabled. So what would happen if the joystick was set to full-speed ahead and then somebody pulls out the cable? The device controlled by the joystick, for instance a crane, will go forward at full speed till it hits its barriers. There is no message reaching the crane controller that it should slowdown, although the crane operator is now pulling the joystick fully backward in order to brake. Do you see it happening?

But if heartbeat was implemented then the crane controller would have a time-out on it and would say: “Hey, where is the joystick. Better hit the brakes right now.” And everything would come safely to a stop.

Another option in this case would be that the joystick sends out its PDO’s in a timely fashion. Then it would be obvious that the crane controller should stop if PDO’s from the joystick didn’t come in anymore.

Configuration Manager

It is frequently mentioned that something needs to be configured. Configuration is done by writing to the Object Dictionary of the device. Reading and writing is done through SDO. The device itself is the SDO server for a client, which can be anything. Obviously it could be a PC with software that knows how to communicate SDO to a CANopen device.
For the software tool to know what objects a device has available in its Object Dictionary, a so-called EDS file is needed. And yes, the name and the extension are the same as a DeviceNet EDS file, but the content is completely different.



However, there are many cases were a PC is not available. Think of a big printing machine where there are only a few operator panels and the machine control is done by a dedicated machine controller. There are several options here. Probably there are more, but these are mostly used:

1. Prepare the configuration of the CANopen devices with a software tool. The basis for the tool are the CANopen EDS files. From these EDS files, so-called DCF files can be made. A DCF file is then nothing more than an EDS file with the values for every object filled in. This DCF file is then transferred to the machine controller that functions as a configuration manager. As soon as a device comes alive, this is seen by the machine controller. It then sends down the parameters from the DCF file to the device. From there, the device is configured and brought into the Operational mode.
2. Again, the software tool is used to prepare the configuration. But it is now the tool that downloads the configuration through SDO-communication to the device. The next action is to store the configuration in the device. There is a special procedure for this, i.e. by writing “SAVE” in a certain object through SDO communication. There is also a “RESTORE” to bring the device back into its default state. However, the devices in the network must support this storage of parameters. And what if a device is broken and needs to be replaced?
3. The machine controller does not use a DCF file, but each and every parameter that needs to be set in the devices is programmed in the machine-controller’s program. This is a lot of work. And needs a lot of testing and error checking. However, it still does the job of a configuration manager.

Typically, a configuration manager does not only download configurations to devices, but also controls the CANopen states of all the devices. And it checks to see if it receives all heartbeats from all device. If not it should signal it somehow to an operator and take countermeasures. For those devices that handle their PDO communication on a SYNC-message, it is also the SYNC producer.
There are a lot of things to do for a device and often the whole control program of the machine is also involved. And in this configuration setup, again, all I/O-information is stored locally. We are back to the Master/Slave network.
So what is the difference between the CANopen solution and a solution with DeviceNet?

My settings are gone!
Another real life example of what can happen. It somehow shows how vendors interpret backward compatibility. And also how loose the implementation of CANopen is.

We needed some CANopen I/O-devices for training purposes. In the past we had purchased some of these devices, but needed some more. So we placed a purchase order and received a call from the distributor. The devices ordered were no longer available, but there was a direct replacement with the same price and the same functionality.

I received the products and did a test drive. No problem. All parameters still seemed to be the same, till I wanted to store the configuration in the device. This option was removed by the vendor, probably because of cost. What if your application relied on option 2 in the Configuration Manager section?

Bottom-line.
I hope that you now understand why CANopen is so attractive to so many people. With all the options you have in PDO communication and the device-to-device communication, you can tune it as you like. But you must know exactly what you are doing. Not only regarding the PDO-communication side. But also regarding how you control all the devices and keeping track of them still being alive. And maybe parameters have to be downloaded to a device first before it can operate properly. Also CANopen is a very “open” specification. It leaves a lot of room for interpretation and a lot of things are optional. Therefore it is very difficult to say that a device will function in a certain application. It could be the case that special measures need to be taken. And that parameters need to be set before the application functions reliably.

And sometimes a device does not work as expected or the functionality has changed. What is the solution? Who is to blame?

But based on the experience as European Product Engineer for Omron’s CANopen products, what strikes me most in the field is that a lot of applications is again in Master/Slave style. The setup is one central controller and devices with which it communicates. So what is the difference compared with a DeviceNet network? This is also master/slave and uses CAN as the hardware layer.

And with DeviceNet all of the above, like heartbeat, CANopen state-machines and emergency messages, are already taken care of. And DeviceNet is specified in such a way that there is not much room for different interpretations.

Is CANopen then a bad network? Not at all! CANopen has a lot more to offer then a pre-defined network such as DeviceNet. But you must know exactly what you are doing. Plan your network well and know what devices you are going to use, then you will benefit from what CANopen has to offer.

For all who plan to use CANopen, I wish them a lot of fun configuring. And do your bookkeeping properly.

René Heijma
Product Engineer Industrial Communication
Omron Europe