Protocol manual

*Device**Net*** **Slave**

Hilscher Gesellschaft für Systemautomation mbH
Rheinstraße 78
D-65795 Hattersheim
Germany

Tel. +49 (0) 6190 / 9907 - 0
Fax. +49 (0) 6190 / 9907 - 50

Sales: +49 (0) 6190/9907-0
Hotline and Support: +49 (0) 6190/9907-99

E-mail: hilscher@hilscher.com
Homepage: http://www.hilscher.com

Index	Date	Version	Chapter	Revision
1	30.03.99	1.000	all	creation

Although this protocol implementation has been developed with great care and intensively tested, Hilscher Gesellschaft für Systemautomation mbH cannot guarantee the suitability of this protocol implementation for any purpose not confirmed by us in writing.

Guarantee claims shall be limited to the right to require rectification. Liability for any damages which may have arisen from the use of this protocol implementation or its documentation shall be limited to cases of intent.

We reserve the right to modify our products and their specifiactions at any time in as far as this contributes to technical progress. The version of the manual supplied with the protocol implementation applies.

1 Introduction to the system

The processing of closed loop and open loop control information for machinery and systems with the aid of powerful microchips is widespread in modern automation technology. As a rule, such intelligent units of an automation complex are connected together by one or more communications networks. Within the production facility as a whole, the networks are incorporated in various hierarchical levels, starting with the administration network for the works and ranging up to the network of sensors and actuators in the manufacturing process.

The growing degree of automation and the demand for more and more decentralised units has caused a considerable increase in the quantity of data exchanged in all areas. The establishment of a new concept to accommodate this large flow of data therefore appeared unavoidable.

The lowest hierarchical level in particular, that of sensors and actuators, required the industry to establish a communications network in which the data generated there simultaneously could be transmitted qualitatively and quantitatively as rapidly as possible to the higher level process without the already time critical process sequences noticeably suffering from the transmission speed of the bus.

In the search for a suitable bus system in the field bus area, the parallel transmission process in which each sensor or actuator is connected individually to the higher level controller in a network with star topography was found unsuitable as soon as cost effectiveness calculations revealed that the costs of installation and maintenance exceeded the benefits derived from rapid data transfer.

The only solution here was single cable transmission with serial bus systems, meeting the modern demands for flexible process adaptation and reconfiguration or system expansions.

2 DeviceNet fundamentals

2.1 The Requirements

The connection between the decentralized process sequence and the centralized control system established by the communications system takes place on the lowest hierchical level via the field or process bus.
On this level, the primary requirement is that for a simple protocol sequence and short data transfer times in communications. This guarantees the shortest possible system reaction time to dynamic peripheral conditions.

Together with classical I/O data exchange, acyclical transmission of parameter, diagnostic and configuration data must be possible without decisively impeding the real time capabilities of the bus. Only in this way can a good diagnostic concept be fulfilled and operational reliability be ensured.

2.2 Characteristics of CAN Protocol and DeviceNet

The main function of the DeviceNet is the transmission of process data from the controller system to the peripherals and vice versa in a predefined profile. It uses the Controller Area Network technology for its physically signaling and to transport these data. The access is done in the client-server principle. One or many master are controlling their assigned node devices on the bus in polling, cyclic, change of state or event operation. A simultanous access of two nodes is possible, but intercepted and dissolved in the CAN bus protocol itself. DeviceNet can be used in the baudrates 125kBaud, 250kBaud and 500kBaud.

DeviceNet is a connection based network. It supports up to 64 nodes. The CAN protocol in general solves bus conflicts while simultanous bus access of two devices by message priority. This priority is specified by the header identifier of the message itself, which is always leading the message on the bus as a header information. The lower the whole header identifier is, the higher is the priority of the message.
The header identifier is divided by the DeviceNet specification into 4 message groups, each has always 5 bits reserved in there for the node address to address the right receiver in the message This address is called Media Access Control Identifier short MAC-ID in DeviceNet. This value distinguishes a node among all other nodes on the same link. Because the MAC-ID is directly propotional to the header identifier then, a node device with a lower address has in comparison with a device of a higher address a higher priority and herewith a better chance to send its messages. A device which was interrupted in sending a message by a message higher priority, starts its send attempt again after the high priority message has ended. If the device do not succeed in sending its message after defined retries, the device goes into a defined error state. The location of the 5 address bits is different in the serveral groups whereby other priorities are resulting per group. The lower the group itself the higher is its priority. The other bits serves to distinguish possible multiple assigned communication channels to one node device.

2.3 Introduction of DeviceNet

2.3.1 Philosophy of DeviceNet

DeviceNet is a low-level network that provides connections between simple industrial devices like sensors and actuators and higher-level devices like controller. It is possible to establish Master/Slave connections as well as Peer-To-Peer communications between them. Our devices integrate themself into Master/Slave category only and have the aim to send and receive process data in the simpliest manner.

2.3.2 The DeviceNet Object Model

The DeviceNet uses an abtract object modeling. In there it is desribed how a device shall behave or which communication services are available and can be accessed for example. A node in DeviceNet has a collection of *Objects*. An Object provides an abstract representation of a particular component within a product. The realization of this abstract object model within a product is implementation dependent.
A DeviceNet *Class* can be understood as a superior term for a set of objects that all represent the same kind of system component. An *Instance* follows in the object model hierachie to the class. Each instance is the actual representation of a particular component within a class. *Attributes* in DeviceNet are descriptions of an external visible characteristic or feature of an instance.

The following table shall illustrate the coherence between the several terms:

Class	Instances	Attributes	Attribute Values
Town	Frankfurt	Street	Mainzer Landstraße
		Inhabitants	500.000
	Chicago	Street	Elm Street
		Inhabitants	600.000

2.3.3 DeviceNet connections

DeviceNet defines a connection-based scheme to facilitate all application communications. A connection is always referenced by a unique *Connection ID* when it is established, so that the two end-point devices which hold the connection can associate it.
DeviceNet's connection-based scheme defines a dynamic means by which the following two types of connections can be established:

2.3.3.1 Explicit Messaging Connections

Explicit Messaging Connections - multi-purpose communication paths between two devices, with the aim to command the performance of a particular task and to report the result of performing task, like it is used to configure a module for example. Explicit messages provide the typical request/response-oriented network communications.
DeviceNet itself makes use of the explicit message protocol to announce explicit messaging connections. It defines a fixed procedure and services with predefined CAN messages to get such connections run and to established the first contact to the other participant. Once established applications typically use explicit message connections to transfer big data blocks, like used in a parameter download procedure for example or to transfer low priorty read and write services between the two end-points.

2.3.3.2 I/O connections

I/O connections - communication path between a producing application and one or more consuming applications, with the aim to move application-specific I/O data through these ports. Normally this data is traded as real time proccess data.
I/O connection based messages are transferred with little overhead to reach high performance while transferring it. DeviceNet supports a fragmentation protocol when the CAN message limitation of 8 byte per message is reached, to support larger data package. The limitations in the number of the data bytes that can be transferred is device specific then.

I/O connections are established, enabled and configured via explicit messageing before, so that both connection end-points are preconfigured right and have the knowledge of the intended use and meaning of the I/O message when data is transferred through this line. In DeviceNet a simple device can support a socalled *Predefined Master/Slave connection* set. This is a general model manufactors of devices can use as a basis for the definition of a set of connections which facilitate communications typically seen in a Master/Slave relationship. In there many of steps involved in the creation and configuration of an application to application connection have been removed. This guarantees a short time to develop the device communication software itself as well as compatibility to other devices in the network using the same predefined messages. Established once these I/O connections need no further parameterization in their behaviour. An other possibility is to establish such I/O connections dynamically. They have to be created while the startup procedure of the device and all there parameters must be configured through this connection first till the I/O transfer can happen. Our devices do not support the dynamic creation of such I/O connections at the moment.
There are a wide variety of functions that can be accomplished using I/O connections. DeviceNet distinguishes 4 main I/O connections that can be established.

• *polled I/O connections* - One poll command from the master sends any amount of output data to a specific slave device. This can be done non-fragmented or fragmented to the destination slave device. The slave device consumes the poll command and processes the output data. If it has input data configured for this poll connection it reacts by sending back any amount of input data and/or status information to the master. Before a polled I/O message is initiated by the master, he reads the *consumed* and *produced connection size* of the data from the slave first and compares each value with the internally configured one. If he detects differences the connection cannot be established. Sending a poll command can happen at any time the master want to and has nor timer nor event dependencies.A Slave device has to respond if it has consumed and understood the poll command request of the master, even if it has no input data. Else the master will report a timeout error. Polling data to many slave devices has the disadvantage that the network traffic rate is very high and most data which is transferred has not changed since the last transmission. Furthermode the more the bus loading the more communication errors can occur if the bus is disturbed by external influences.

• *bit stobe I/O connections* - Bit strobe command and response messages rapidly move small amounts of I/O data between a master and one or many slaves. The bit strobe message contains a bit string of 64 bits of output data, one output bit per possible device. Each bit in there is assigned to one device address in the network. Herewith this service has broadcast functionality that means more than one slave device can be addressed by one command. Because all addressed slave devices get this command at the same time, this command is normally used to sychronize data transfer to more slave devices. A slave device can take its corresponding output bit as a real output information to give it to the peripheral connections and/or use the bit as a trigger to send back its input data with a poll response message. The data that can be send back from each slave after a bit strobe command was received is limited to 8 bytes in length. Bit strobe usage causes therefore a reduced bus loading than poll messaging.

• *change of state / cyclic I/O connections* - The master's change of state/cyclic message sends any amount of output data to the destination slave device. Data production is triggered by either a determined changed value in the output data or the cyclic timer expiration. Depending on how the slave's behavior is configured, the slave can send back an acknowlegde message, containing any amount of input data and/or status information. The slave's change of state/cyclic message sends any amount of output data to the master, if the data is either changed or the cyclic timer has expired. The master itself can acknowledge this message with output data if configured. Change of state only production of data hold down the bus loading as small as possible, while data than can be trasnmitted as fast as possible by each device because bus conflicts get unprobable. So you can get high performance data transmission with in comparison low baud rates.

2.4 Network Management

The network management and configuration of the slave devices is normally done by a socalled DeviceNet Manager tool. In most cases a configuration of the slave devices is not necessary at all, especially not if predefined master/slave supporting devices are involved. Our card in extended version will support a configureable network management too, so that there is no need for further external tools and cards.

2.5 DeviceNet Profiles

To provide interoperability and promote interchangeability by like device types, there must be some consitency between devices of the same type. The formal definition of this information is known as *device profile*. The object model and format contents for a device are fixed in there manufacturer independent. Herewith devices following the same device profile are exchangeable. Although many parameters are predefined in the DeviceNet, space is left for manufacturer to extend the existing profile by their own services based on the format contained in the DeviceNet specification.

2.6 Device Guarding

To detect if an existing device has gone off the bus, our master card supervise all configured device with a socalled heartbeat message. So the master can recognize if a device is present on the bus or not. Each message comming from the device reports presence and therefore the master resets the heartbeat timer for this device. So heartbeat messages will only be sent, if the contact is interrupted a defined timer value to hold the bus loading a small as possible.

3 Supported features

Supported baurates: 125K,250K,500KBaud, autobaud detection

Connections, also fragmented

- pollup to 255 byte input and output process data
- Explicit messaging connection

group 2 only server.

4 Supported objects

4.1 Identity Object 0x01

Services implemented: Get_Attribute_Single, Reset

Instance Attributes	ID	Value Limit	Get	Set
Vendor	1	283	*	
Device Type	2	12	*	
Product Code	3	2	*	
Revision	4	1.001	*	
Status	5		*	
Serial Number	6		*	
Product Name	7	COM-DNS, CIF 104-DNS CIF 30-DNS	*	

4.2 Message Router Object 0x02

Instance Attributes	ID	Value Limit	Get	Set
non supported				

4.3 Devicenet Object 0x03

Implemented Services: Get_Attribute_Single,Set_Attribute_Single,Allocate M/S connection set, Release M/S connection set

Instance Attributes	ID	Value Limit	Get	Set
MAC ID	1	0-63	*	
Baud rate	2	0,1,2	*	
Bus-off Interrupts	3	0	*	
Bus-Off-Counter	4	0-255	*	
Allocation-Information	5	x	*	

4.4 Connection Object 0x05

Implemented Connection Instances:

- Explicit Messaging: Instance 0
- Polled: Instance 1

Implemented Service: Get_Attribute_Single, Set_Attribute_Single,Reset

Instance Attributes	ID	Value Limit	Get	Set
state	1	0,1,3,4,5	*	
instance_type	2	0,1	*	
transportClass_Tigger	3	0x82 =i/O,0x83 = expli.	*	
produced_connection_id	4	x	*	
consumed_connection_id	5	x	*	
inital_comm_characteristics	6	x	*	
produced_connection_size	7	0-255	*	
consumed_connection_size	8	0-255	*	
expected_packet_rate	9	0-65535	*	*
watchdog_timeout_action	12	0,1,2,3	*	*
produced_connection_path_length	13	6	*	
produced_connection_path	14	4,1,3	*	
consumed_connection_path_length	15	6	*	
consumed_connection_path	16	4,1,3	*	