empowering communication

Protocol API

DeviceNet Slave

V5.4.0

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC210205API05EN | Revision 5 | English | 2024-04 | Released | Public

Table of contents

1 Introduction

1.1 About this document

This manual describes the application interface of the DeviceNet Slave stack V5.4 for netX-based products. The aim of this manual is to support the integration of devices based on the netX chip into own applications.

1.2 List of revisions

Rev	Date	Name	Revisions
1	2020-02-26	MIK	Document created.
2	2021-03-30	MIK	Firmware/stack version V4.1/V5.1 Section *Technical data*: Max. I/O data and max. Acyclic data corrected. Section *DeviceNet Object (Class Code: 0x03)*: More info about attribute 8 and 9. Section *DeviceNet Object (Class Code: 0x03)*: Default value of attribute 5 corrected. Section *DPM Communication status* State Diagram improved.
3	2022-02-16	MIK, HHE	Firmware/stack version V4.2/V5.2 Section *System requirements* updated. Add new section Diagnostic service Section *Reset service*: Support of reset type 2 and user reset types. Add new section *Device data*: Setting the serial number via OEM parameter.
4	2023-02-07	MIK, RGO	Firmware/stack version V5.3 Correct sizeof() len description in service DNS_DIAG_REQ Section ‘Set Configuration request’ allow a configuration of device type value 0 Section ‘Set Configuration request’ new message body formats 8/16; 16/16; 16/8 Section ‘Hilscher-specific CIP services’ add detailed table of attribute option flags Section ‘Hilscher-specific CIP services’ add notify and forward seq. diagram Section ‘Hilscher-specific CIP services’ new service ‘Modify Status’ Section ‘Reset service’ add new reset reason
5	2023-04-04	MIK, HHE	Firmware/stack version V5.4.0 Section *Process data status* new sequence counter. Section *Message Router Object (Class Code: 0x02)* access from network. Section *Set Configuration request* individual disabling of IO connections. Section *Modify Identity Status Attribute* ID Attribute 5 Configured Bit 2 is settable. Section *Attribute Option Flags* Class attributes can be hidden. Section *Acknowledge Handler Object (Class Code 0x2B)* disabling. Section *Get / Set Trigger Type* add new supported trigger type. Section *Cyclic data exchange* new exchange modes and additional description.

Table 1: List of revisions

1.3 System requirements

1.3.1 System requirements for firmware generation V5

The software package has the following system requirements to its environment: netX 90 Chip as CPU hardware platform.

Compatibility between DeviceNet Slave firmware/stack and netX 90

Starting with version 5.2.0, the firmware/stack requires netX 90 with date code 1910 and later. If a NXHX 90-JTAG is used, hardware revision 4 or higher is required.

netX 90 samples (date code before 1910) are no longer supported which is the case for NXHX 90-JTAG hardware revision 3 or lower.

Maintenance Firmware

The firmware/stack requires Maintenance Firmware V1.4.0.0 (or higher).

1.4 Intended audience

This manual is suitable for software developers with the following background:

- Knowledge of the programming language C
- Knowledge of the DeviceNet protocol
- Knowledge of the ODVA's CIP protocol

1.5 Technical data

The data below applies to DeviceNet Slave stack version V5.4.0.

Technical data

Feature	Parameter
Maximum number of cyclic input data	255 bytes
Maximum number of cyclic output data	255 bytes
Acyclic communication as server	Max. 255 bytes per request
Baud rate	125 kBits/s, 250 kBit/s, 500 kBit/s Automatic baud rate detection is not supported
Connection establishment	Predefined Master/Slave Connection Set
IO Connections	Poll ChangeOfState Cyclic Bit-strobe
Explicit messaging	Supported
Fragmentation	Explicit and I/O
Message body format	8/8; 8/16; 16/8; 16/16;
Data transport layer	CAN

Table 2: Technical Data DeviceNet Slave

Firmware available for netX

netX	Available
netX 90	yes (V5)

Table 3: Firmware available for netX

Configuration

- Packet API based configuration by host application
- Data base configuration by configuration tool

Diagnostic

- Common and extended diagnostic via dual port memory
- Stack diagnosis via Packet API

Limitations

- UCMM (Unconnected Message Manager) is not supported
- Quick Connect is not supported

1.6 Terms, abbreviations, definitions

Term	Description
AP	Application on top of the Stack
ASCII	American Standard Code for Information Interchange
BOI	Bus-off interrupt
CAN	Controller Area Network
CIP	Common Industrial Protocol
COS	Change of State
DDP	Device Data Provider
DL	Data Link (Layer)
DNS	DeviceNet Slave
DPM	Dual Port Memory
FDL	Flash Device Label
LFW	Loadable Firmware
LSB	Least Significant Byte
MAC ID	Media Access Control Identifier (i.e. address of a DeviceNet device)
MSB	Most Significant Byte
ODVA	Open DeviceNet Vendors Association
UCMM	Unconnected Message Manager
GRC	General Error Code
ERC	Extended Error Code

Table 4: Terms, abbreviations and definitions

1.7 References to documents

This document refers to the following documents:

[1] Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX Dual-Port Memory Interface, Revision 17, English, 2020.

[2] Hilscher Gesellschaft für Systemautomation mbH: Packet API, Packet-based services (netX 10/50/51/52/100/500), Revision 5, English, 2021.

[3] Hilscher Gesellschaft für Systemautomation mbH: Packet API, netX Dual-Port Memory, Packet-based services (netX 90), Revision 6, English, 2021.

[4] Hilscher Gesellschaft für Systemautomation mbH: Programming reference guide, CIFX API, Revision 09, English, 2020.

[5] ODVA: The CIP Networks Library, Volume 1, “Common Industrial Protocol (CIP™)”, Edition 3.28, April 2020.

[6] ODVA: The CIP Networks Library, Volume 3, “DeviceNet Adaptation of CIP”, Edition 1.15, November 2018.

[7] Hilscher Gesellschaft für Systemautomation mbH: Operating Instruction Manual, DTM for Hilscher DeviceNet Slave Devices, Configuration of Hilscher Slave Devices, Revision 12, English, 2019.

[8] Hilscher Gesellschaft für Systemautomation mbH: Operating Instruction Manual, Tag List Editor V1.5.

Table 5: References to documents

2 DeviceNet Slave features

2.1 Structure of the stack

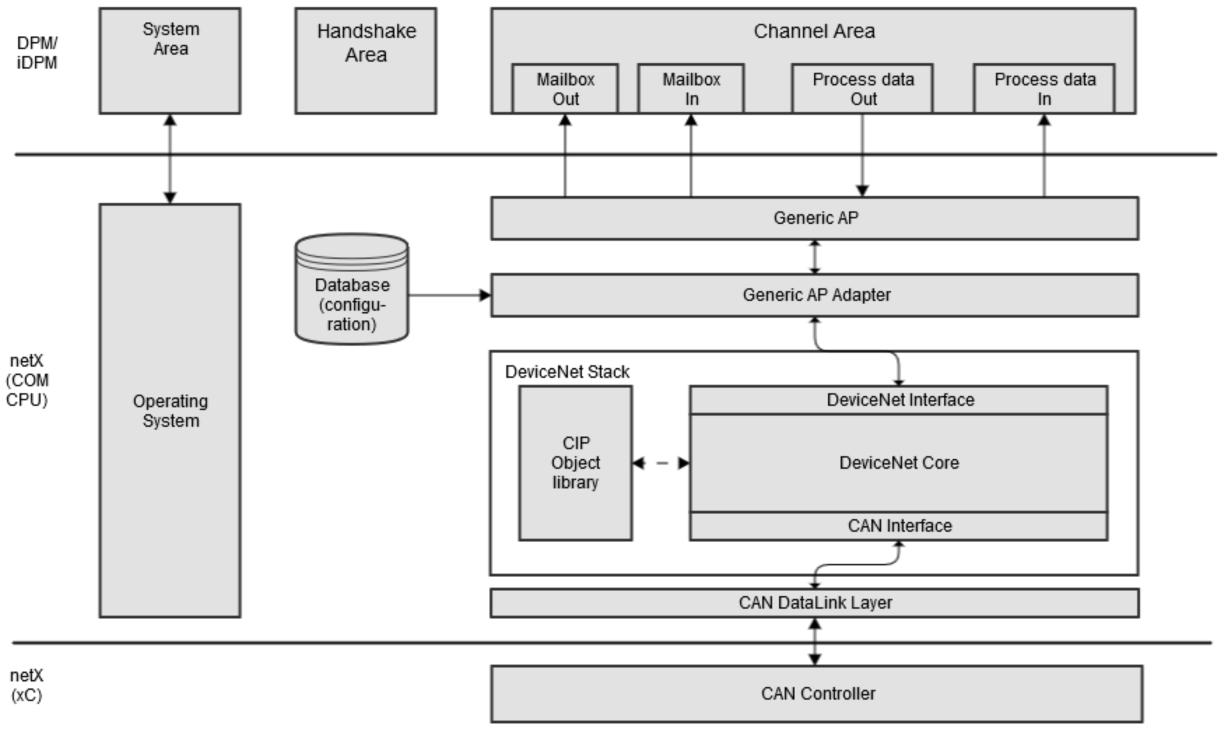

Figure 1: Structure of the DeviceNet Slave stack

Generic AP

The generic AP is a Hilscher common component, which handles the DPM interface on front side. On the backend, it provides a generic communication interface named GCI, which is the common interface where all stacks have to adapt to.

Generic AP adapter

The generic AP adapter binds the ‘DeviceNet’ interface to the backend interface of the ‘Generic AP’. It is responsible for:

- Handling packets received from the application to the stack
- Handling indication from stack to the application
- Provide information about connection state
- Preparation of configuration data

DeviceNet Core

The DeviceNet core component is the main part of the DeviceNet Slave stack. This component is responsible:

- Object handling of the DeviceNet Slave stack according the CIP object model
- Connection management
- Network access state machine

CAN DL

The CAN DL component is responsible for sending and receiving CAN frames to underlying CAN hardware layer.

2.2 CIP Introduction

2.2.1 Object classes

Speaking of CIP object classes means to distinguish between class and instance level. Each object exists at class level and, additionally, may have one or more instances. CIP services address a certain object class or instance by means of a specified Instance ID. An Instance ID value of zero addresses the object class, whereas Instance IDs larger than zero address the corresponding instance of that object class.

Each CIP object class and instance consists of a set of attributes and services. Naturally, the attributes each object class provides at class and instance levels are different from each other. The most common services are the Get_Attribute_Single and Set_Attribute_Single services to read or write the attributes of the addressed object class or instance.

The following sections uses four tables to describe each supported object class:

1. Class attributes.
2. Instance attributes.
3. Services available to the host application.
4. Services available to DeviceNet master over the network.

2.2.2 Class attributes

Class Attributes are defined using the following notation:

Class Attributes (Instance 0)

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	2	3	4	5	6	7

Table 6: Introduction of Class Attribute Description

1. The **Attribute ID** is an integer identification value assigned to an attribute. Use the Attribute ID in the Get_Attributes and Set_Attributes services list. The Attribute ID identifies the particular attribute being accessed.
2. **Name** specifies the name of the class attribute.
3. **Access from Network** specifies the access permission of the attribute when the service is sent from the DeviceNet network. The definitions are:
   - Set (Settable) - The attribute is accessible by at least one of the set services (Set_Attribute_Single/ Set_Attribute_All).
   - Get (Gettable) - The attribute is accessible by at least one of the get services (Get_Attribute_Single/ Get_Attribute_All).
4. **Access from Host** specifies the access permission of the attribute when the service is sent from the Host Application using the DPM/Packet Interface. The definitions for access rules are:
   - Set (Settable) - The attribute is accessible by at least one of the set services (Set_Attribute_Single/ Set_Attribute_All).
   - Get (Gettable) - The attribute is accessible by at least one of the get services (Get_Attribute_Single/ Get_Attribute_All).
5. **Description** contains a descriptive text on the attribute.
6. **Default value** specifies the default value of the attribute.
7. **Supported by default** indicates whether this attribute is supported by the stack in a default configuration.

   In a default configuration, the DeviceNet Slave stack implements certain attributes, which are not accessible from the DeviceNet network. In order to access these attributes via the network, the host application has to activate them using a specific service “Set Attribute Option”. See section *Attribute Option Flags* on page 36 and *Set Attribute Option* on page 40).

   ✓ ➔ The attribute is supported and activated per default.

   ⚠ ➔ The attribute is supported and deactivated per default. The host can activate it.

   ✗ ➔ The attribute is not supported. The host cannot activate it.

2.2.3 Instance attributes

An Instance Attribute is an attribute that is specific to an object class instance. Instance Attributes are defined in the same notation as Class Attributes.

Instance Attributes (Instance [1..N])

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	2	3	4	5	6	7

Table 7: Introduction of Instance Attribute Description

2.2.4 Services

Services can either address the class level (Instance ID 0) or the instance level (Instance ID [1..N]) of a CIP object. Services may be issued either by the host application or by a client on the DeviceNet network.

Correspondingly, we will present services supported by each object class in two tables:

The first table shows common services that can be issued toward the protocol stack over the network, as well as by the host application. The second table shows Hilscher-specific services that are available to the host application only.

Both tables have the same format:

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
1	2	3	4	5

Table 8: Introduction of Service Description

1. **Service Code** is a hexadecimal value to identify the CIP service uniquely. Service codes below the value 255 are defined within the DeviceNet specification. Larger numbers are reserved for Hilscher-specific services. Hilscher-specific services are described separately in section *Hilscher-specific CIP services* on page 36.
2. **Name** specifies the name of the service.
3. Addressing the object's class level

   ✓ ➔ The stack supports this service at object class level (Instance ID 0).

   ✗ ➔ The stack does not support this service at class level.
4. Addressing the object's instance level

   ✓ ➔ The stack supports this service at object instance level (instance 1-n).

   ✗ ➔ The stack does not support this service at instance level.
5. **Description** contains descriptive text on the service.

2.2.5 CIP definitions

2.2.5.1 CIP defined Service Codes

The following service codes are pre-defined by the CIP specification [5], Chapter A-3 CIP Common Services, Table A-3.1.

Service code	Service name
0x00	Reserved
0x01	Get_Attributes_All
0x02	Set_Attributes_All
0x03	Get_Attribute_List
0x04	Set_Attribute_List
0x05	Reset
0x06	Start
0x07	Stop
0x08	Create
0x09	Delete
0x0A	Multiple_Service_Packet
0x0B, 0x0C	Reserved for future use
0x0D	Apply_Attributes
0x0E	Get_Attribute_Single
0x0F	Reserved for future use
0x10	Set_Attribute_Single
0x11	Find_Next_Object_Instance
0x12, 0x13	Reserved for future use
0x14	Error Response
0x15	Restore
0x16	Save
0x17	No Operation (NOP)
0x18	Get_Member
0x19	Set_Member
0x1A	Insert_Member
0x1B	Remove_Member
0x1C	GroupSync
0x1D–0x31	Reserved for additional Common Services

Table 9: Service Codes

Note: Not every service is available on every object and on every device in the network.

2.2.5.2 CIP defined Class IDs

The table below shows the ODVA arranged Class ID numbers according reference [6], Table 4-10.4.

Class range	Meaning
0x00 – 0x63	ODVA pre-defined objects
0x64 - 0xC7	Vendor Specific
0xC8 – 0xEF	Reserved by ODVA for future use
0xF0 – 0x2FF	ODVA pre-defined objects
0x500 – 0xFFFF	Reserved by ODVA for future use

Table 10: Class ID ranges according CIP specification

The following table lists the predefined Class IDs, which are typically used for a simple DeviceNet IO slave.

Class ID	Object class
0x01	Identity Object
0x02	Message Router Object
0x03	DeviceNet Object
0x04	Assembly Object
0x05	Connection Object
0x2B	Acknowledge Handler Object

Table 11: Predefined values for the Class ID according to the CIP specification

2.2.5.3 CIP defined General Status Codes

The following table list the common suitable general status codes to handle confirmation and response packets for explicit massaging services in case of errors. The full list of general status codes is listed in reference [5], Chapter B-1 General Status Codes, Table B-1.1:

General Error (GRC)	Description
0	No error
2	Resources unavailable
8	Service not available
9	Invalid attribute value
11	Already in request mode
12	Object state conflict
14	Attribute not settable
15	A permission check failed
16	State conflict, device state prohibits the command execution
19	Not enough data received
20	Attribute not supported
21	Too much data received
22	Object does not exist
23	Reply data too large, internal buffer to small

Table 12: Generic Status Codes

2.2.5.4 CIP defined Extended Status Codes

Additional Error Codes

The additional error code is network, device or object specific. If necessary, manufacturers can individually define an additional code while implementing a service. The pre-defined additional error codes are mentioned in various sections in references [5] and [6]. The specification says when it has to be used. In default, the additional error code is not used.

2.3 Object classes

The DeviceNet slave stack is modeled as a collection of objects. Object modeling organizes related data and procedures into one entity: the object. An object is a collection of related services and attributes. Services are procedures that an object performs. Attributes are characteristics of objects represented by values or variables. Typically, attributes provide status information or govern the operation of an object. An object's behavior is an indication of how the object responds to particular events.

The following collection of objects represent the object library of a DeviceNet Slave stack. The blue colored are the main objects in the default Hilscher DeviceNet Slave stack. The grey colored objects are Hilscher specific objects. The green colored objects are user or profile specific objects where the user application has the option to register them within the DeviceNet Slave stack and handle these object in the user application.

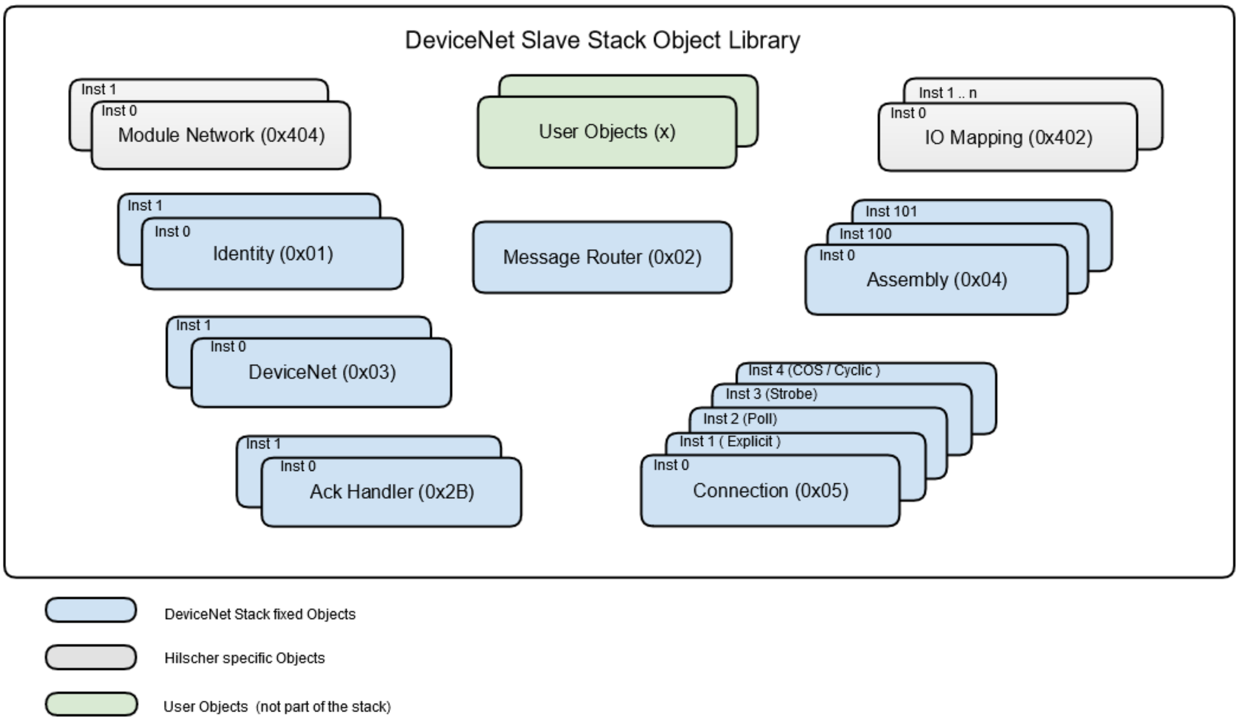

Figure 2: Objects Model of Hilscher DeviceNet Slave stack

For the general description of the Object Model of DeviceNet, see reference [6], Section 1-5 Device Object Model.

2.3.1 Identity Object (Class Code: 0x01)

The Identity Object provides identification and general information about the device. The DeviceNet protocol stack implements the Identity object at class level and a single instance with Instance ID 1. It is used for electronic keying and by applications requiring information about the nodes on the network.

2.3.1.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get/Set	Revision of this object	(1)	✓
2	Max. Instance	Get	Get/Set	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get/Set	The number of Instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get/Set	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get/Set	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(7)	✓

Table 13: Identity Object - Class attributes

2.3.1.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Vendor ID	Get	Get/Set	Vendor Identification	(0x011B) Hilscher	✓
2	Device Type	Get	Get/Set	Indication of general type of product	(1)	✓
3	Product Code	Get	Get/Set	Identification of a particular product of an individual vendor	(1)	✓
4	Revision	Get	Get/Set	Revision of the product	(1.1)	✓
5	Status	Get	Get/Set[1)]	Summary status of device		✓
6	Serial Number	Get	Get/Set	Serial number of device	(1)	✓
7	Product Name	Get	Get/Set	Human-readable identification	“netX”	✓

Table 14: Identity Object - Instance attributes

[1)] The specific service “Modify Status” can be used to modify particular flags of the Status attribute, which is described in chapter 2.4.2.1 Modify Identity Status Attribute on page 43

2.3.1.3 Common services

These services are available to the host application and remote DeviceNet nodes.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x01	Get Attribute All	✓	✓	Retrieve all attribute values
0x05	Reset[1)]	✗	✓	Reset the device
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✓	✓	Modify attribute value
[1)] Refer to chapter reset handling of the device				

Table 15: Identity Object - Common services

2.3.2 Message Router Object (Class Code: 0x02)

The Message Router Object provides a connection point for messaging through which a client can address a service to any object class or instance within the physical device. The message router supports class and instance attributes as described below.

2.3.2.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(1)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of Instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(2)	✓

Table 16: Message Router - Class attributes

2.3.2.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
2	Number Available	Get	Get	Maximum number of connections supported	(4)	✓

Table 17: Message Router - Instance attributes

The default value for attribute 2 “Number Available” is 4 which means that the stack is configured per default to allow all types of IO connections POLL + STROBE + COS/CYC. The stack allows disabling IO connections individual (see section *Set Configuration request* on page 69). For each disabled IO connection the value of “Number Available” is decremented by 1. A value of 1 means that an explicit connection is allowed only.

2.3.2.3 Common services

These services are available to the host application and remote DeviceNet nodes.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value

Table 18: Message Router - Common services

2.3.3 DeviceNet Object (Class Code: 0x03)

The DeviceNet Object contains information about the configured DeviceNet Slave. For example, it holds the MAC ID, the Baud rate, the switch values etc.

2.3.3.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get/Set	Revision of this object	(2)	✓
2	Max. Instance	Get	Get/Set	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get/Set	The number of Instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get/Set	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get/Set	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(9)	✓

Table 19: DeviceNet Object - Class attributes

2.3.3.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	MAC ID	Get/Set	Get/Set	Node ID of the device that it is currently operational on the network	(63)	✓
2	Baudrate	Get/Set	Get/Set	Baud Rate of the device that it is currently operational on the network	0	✓
3	BOI	Get/Set	Get/Set	Bus-Off Interrupt behavior	0	✓
4	Bus-Off Counter	Get/Set	Get/Set	Number of times CAN went to the bus–off state	0	✓
5	Object Allocation Information	Get	Get	Connection allocation information and MAC ID of Master (from Allocate)	0, 255	✓
6	MAC ID Switch Changed	Get	Get	The Node Address Switch has changed since last power – up/reset.	0	⚠
7	Baud Rate Switch Changed	Get	Get	The Baud Rate Switch has changed since last power – up/reset.	0	⚠
8	MAC ID Switch Value	Get	Get/Set	Current value of Node Address Switch	0	⚠
9	Baud Rate Switch Value	Get	Get/Set	Current value of Baud Rate switch	0	⚠

Table 20: DeviceNet Object - Instance attributes

Continued on next page.

- Attribute 6 and 8 are only present if the MAC ID has been set via a switch.
- Attribute 7 and 9 are only present if the baud rate has been set via a switch.
- Attribute 6 and 7 are updated by the stack implicit, when the host modify attribute 8 and 9
- Attribute 8 contains the actual value (position) of the physical ‘Node Address Switch’. At startup, it is the same value as attribute 1. During runtime, the value in attribute 8 can differ from the value in attribute 1 (MAC ID the device is currently online in the network). If the stack controls the ‘Node Address Switch’ (the switch is bound to the netX hardware), then the stack reads the switch value from the hardware at startup. During runtime, the stack updates attribute 8 in case the switch value has been changed. If the host controls the ‘Node Address Switch’ (the switch is bound to the host hardware), then the host application has to update (Set) attribute 8 in case the switch value changes at runtime on the host hardware.
- Attribute 9 contains the current value (position) of the physical ‘Baud Rate Switch’. At startup, it is the same value as attribute 2. During runtime, the value in attribute 9 can differ from the value in attribute 2 (Baud rate the device is currently online to the network). If the stack controls the ‘Baud Rate Switch’ (the switch is bound to the netX hardware), then the stack reads the switch value from the hardware at startup. During runtime, the stack updates attribute 9 in case the switch value has been changed. If the host controls the ‘Baud Rate Switch’ (the switch is bound to the host hardware), then the host application has to update (Set) attribute 9 in case the switch value changes at runtime on the host hardware.

2.3.3.3 Common services

These services are available to the host application and remote DeviceNet nodes.

Service Code	Name	Addressing the object’s		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✓	✓	Modify attribute value

Table 21: DeviceNet Object - Common services

2.3.3.4 Object-specific services

These services are only available for a remote DeviceNet node (master) for connection handling.

Service Code	Name	Addressing the object’s		Description
		Class Level	Instance Level	
0x4B	Allocate_Master/Slave_Connection_Set	✗	✓	Requests the use of the Predefined Master/Slave Connection Set.
0x4C	Release_Master/Slave_Connection_Set	✗	✓	Indicates that the specified Connections within the Predefined Master/Slave Connection Set are no longer desired. These Connections are to be released (Deleted).

Table 22: DeviceNet Object - Specific services

2.3.4 Assembly Object (Class Code 0x04)

The Assembly objects in the Hilscher DeviceNet Slave stack are created at configuration phase. The Hilscher preferred default instance of the Assembly object is 100 (0x64) for consuming data and 101 (0x65) for producing data, but the instance numbers are also configurable.

2.3.4.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get/Set	Revision of this object	(2)	✓
2	Max. Instance	Get	Get/Set	Maximum instance number of an object currently created in this class level of the device	(0)	✓
3	Number of Instances	Get	Get/Set	The number of Instances currently created in this class	(0)	✓
6	Maximum ID Number Class Attributes	Get	Get/Set	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get/Set	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(4)	✓

Table 23: Assembly Object - Class attributes

2.3.4.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Number of Member	✗	Get	Number of members in List	n.a.	✓
2	Member	✗	Get	Member list	n.a.	✓
3	Data	Get/Set	Get/Set	Current process data snapshot	n.a.	✓
4	Size	Get	Get/Set	Process data size in number of bytes	n.a.	✓

Table 24: Assembly Object - Instance attributes

2.3.4.3 Common services

These services are available to the host application and remote DeviceNet nodes.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✓	✓	Modify attribute value
0x18	Get Member	✗	✓	Get a member of instance attribute 2

Table 25: Assembly Object - Common services

2.3.5 Connection Object (Class Code: 0x05)

The Connection Object can have up to four instances. Each instance has a dedicated relation to a specific connection type. The Connection Object holds information about the current connection status, connection-timing parameter and information to the assigned assembly object.

- Instance 1 references the Explicit Messaging connection,
- Instance 2 references the Poll connection
- Instance 3 references the Bit Strobe connection
- Instance 4 references the Change of State / Cyclic connection

2.3.5.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(1)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(4)	✓
3	Number of Instances	Get	Get	The number of Instances currently created in this class	(0 … 4)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get/Set	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(101)	✓

Table 26: Connection Object - Class attributes

2.3.5.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	State	Get	Get	Connection State	0	✓
2	Type	Get	Get	Indicates either I/O or Messaging Connection	0	✓
3	Transport Type	Get	Get	Defines behavior of the Connection	0	✓
4	Produced Connection ID	Get	Get	Placed in CAN Identifier Field when the Connection transmits.	0	✓
5	Consumed Connection ID	Get	Get	CAN Identifier Field value that denotes message to be received.	0	✓
6	Initial Com Characteristics	Get	Get	Defines the Message Group(s) across which productions and consumptions associated with this Connection.	0	✓
7	Produced Connection Size	Get	Get	Maximum number of bytes transmitted across this Connection	0	✓
8	Consumed Connection Size	Get	Get	Maximum number of bytes received across this Connection	0	✓
9	Expected Packet Rate	Get/Set	Get	Defines timing associated with this Connection	0	✓
12	Timeout Action	Get/Set	Get	Defines how to handle Inactivity/Watchdog timeouts	0	✓
13	Produced Path Length	Get	Get	Number of bytes in the Produced Connection Path attribute	0	✓
14	Produced Connection Path	Get	Get	Produced Connection Path	0	✓
15	Consumed Path Length	Get	Get	Number of bytes in the Consumed Connection Path attribute	0	✓
16	Consumed Connection Path	Get	Get	Consumed Connection Path	0	✓
17	Inhibit Time	Get/Set	Get	Defines minimum time between new data production	0	✓
100	Consume Assembly Instance	Get/Set	Get/Set	Hilscher specific attribute pointing to the consuming assembly instance for this connection.	0	✓
101	Produce Assembly Instance	Get/Set	Get/Set	Hilscher specific attribute pointing to the producing assembly instance for this connection.	0	✓

Table 27: Connection Object - Instance attributes

2.3.5.3 Common services

These services are available to the host application and remote DeviceNet nodes.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x05	Reset	✗	✓	Reset a connection
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✓	✓	Modify attribute value

Table 28: Connection Object - Common services

2.3.5.4 Specific attributes

The connection object supports two additional attributes 100 and 101 in the instance level of the object. These are Hilscher specific in the user range of attributes.

These attributes are used to hold the assembly instance that produces or consume the IO data for the associated connection. The sense of this attributes is to allow the user (from host application) or from the network side to bind the connections to different assembly objects.

Connection Object Instance	Default value Attr 100 (consume)	Default value Attr 101 (produce)	Note
1	none	none	This instance relates to the explicit message connection. The explicit message connection has no binding to an assembly object. Therefore the attributes are not existent for instance 1.
2	100 *)	101 *)	This instance relates to the "Poll connection". The value of attribute 100 is the default value of the output assembly instance. The value of attribute 101 is the default input assembly instance.
3	0	101 *)	This instance relates to the "Bit Strobe" connection. The default value for attribute 100 is 0 and cannot be changed. Because the 'Bit Strobe' connection does not has consuming data. The default value for attribute 101 is the same as for the poll connection and produces per default the same data like the poll connection.
4	0	101*)	This instance relates to the "COS / Cyclic" connection. The default value for attribute 100 is 0 and cannot be changed. The consume assembly of the COS / Cyclic connection is always the same like from the poll connection.

Table 29: Predefined Connection Object - Instance attributes default values

*) The attribute default values are the default configured assembly instances which are configured with the set configuration packet.

The attribute values can only be set to existing assembly object. In opposite to the default attributes of the connection object the existence of these Hilscher specific attributes are not bound to the allocation of the connection object itself. This is required to allow a master to write these attributes at configuration phase before allocating the connection.

2.3.6 Acknowledge Handler Object (Class Code 0x2B)

2.3.6.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(1)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of Instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get/Set	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(3)	✓

Table 30: Acknowledge Handler Object - Class attributes

2.3.6.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Acknowledge Timer	Get/Set	Get	Time to wait for acknowledge before resending	16	✓
2	Acknowledge Handler Retry Limit	Get	Get	Number of Ack Timeouts to wait before informing the producing application of a RetryLimit_Reached event.	1	✓
3	COS Producing Connection Instance	Get	Get	Connection Instance which contains the path of the producing I/O application object which will be notified of Ack Handler events.	4	✓

Table 31: Acknowledge Handler Object - Instance attributes

2.3.6.3 Common services

These services are available to the host application and remote DeviceNet nodes.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✓	✓	Modify attribute value

Table 32: Acknowledge Handler Object - Common services

Note: The Acknowledge Handler Object will be disabled automatically when the both connection types COS and CYC are disabled. Disabling the object means, it is not existent and from the network any more. Connection types can be disabled as described in section *Set Configuration service* on page 69.

2.3.7 IO Mapping Object (Class Code: 0x402)

The IO Mapping Object is responsible for partitioning of the DPM I/O input and output areas and mapping of those partitions, i.e. members, to the related instances of the Assembly object. This is a Hilscher-specific CIP object, which is not covered by the CIP specification. For each created assembly object, a corresponding IO mapping object is created automatically within the stack. This object is intended for internal management of the stack. There is no usage for the application of this object.

2.3.7.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	None	Get	Revision of this object	(1)	✓
2	Max. Instance	None	Get	Maximum instance number of an object currently created in this class level of the device	(n)	✓
3	Number of Instances	None	Get	The number of Instances currently created in this class	(n)	✓
6	Maximum ID Number Class Attributes	None	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	None	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(4)	✓

Table 33: IO Mapping Object - Class attributes

2.3.7.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Status	None	Get	Status of I/O data (Data direction)	(0)	✓
2	Length	None	Get	Length of I/O data	(0)	✓
3	Data	None	None	I/O data	(0)	✗ *)
4	Offset	None	None	Offset within the corresponding DPM area	(0)	✗ *)

Table 34: IO Mapping Object - Instance attributes

*) These attribute data cannot be accessed from host application.

2.3.7.3 Common services

These services are available to the host application and remote DeviceNet node.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✗	✗	Modify attribute value

Table 35: IO Mapping Object - Common services

2.3.7.4 Instance attribute values

Value of Status Attribute	Define	Meaning
0x0000	CIP_OBJECT_IO_MAP_STATUS_FREE	The instance is not bound.
0x0001	CIP_OBJECT_IO_MAP_STATUS_PRODUCER	The instance is bound to an input assembly instance and the output image of the DPM.
0x0002	CIP_OBJECT_IO_MAP_STATUS_CONSUMER	The instance is bound to an output assembly instance and the input image of the DPM.

Table 36: IO Mapping Object – 'Status' attribute values

2.3.8 Module and Network Status Object (Class Code: 0x404)

The Module Network status object is a Hilscher specific object. This object reflects the Module Status and the Network Status as it is described in the DeviceNet specification [5] Chapter 9: Indicators & Middle Layers. The host application can use this object to implement its own LED handling by deriving the corresponding LED state from the status information.

2.3.8.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	None	Get	Revision of this object	(1)	✓
2	Max. Instance	None	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	None	Get	The number of Instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	None	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	None	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(2)	✓

Table 37: Module Network Status Object - Class attributes

2.3.8.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Module Status	None	Get	Module State, (according DeviceNet Spec. [5] Chapter 9-2-2)	0	✓
2	Network Status	None	Get	Network State, (according DeviceNet Spec. [5] Chapter 9-2-3)	0	✓

Table 38: Module Network Status Object - Instance attributes

2.3.8.3 Common Services

These services are available to the host application.

Service Code	Name	Addressing the object’s		Description
		Class Level	Instance Level	
0x01	Get Attribute All	✓	✓	Retrieve all attribute value
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✗	✗	Modify attribute value

Table 39: Module Network Status Object - Common services

2.3.8.4 Instance attribute values

Value of Module Status	Meaning	MS LED state
0	**No Power**	The LED should be off.
1	**Self-Test** The device is performing its power-on self-testing procedure.	The LED should performing the testing blink sequence.
2	**Standby** The device has not been configured.	The LED should flashing green.
3	**Operate** The device is operating	The LED should be steady green.
4	**Recoverable Fault** The device has a recoverable fault.	The LED should flashing red.
5	**Unrecoverable Fault** The device has a non-recoverable critical fault.	The LED should steady red.

Table 40: Module Network Status Object – ‘Module Status’ attribute values

Value of Network Status	Meaning	NS LED state
0	No Power	The LED should be off.
1	**No Connection** The node address and baud rate is configured. The duplicate MAC ID check is performed. The device is online to the network. However, the master did not establish any connection.	The LED should flashing green.
2	**Connected** At least one of the connections (explicit and/or IO) to the device is established.	The LED should be steady green.
3	**Connection time out** At least one connection was established and has timed out	The LED should flashing red.
4	**Critical Link Fault** The device detected an unrecoverable communication error i.e. duplicate MAC ID or CAN Bus OFF event.	The LED should be steady red.
5	**Self-Test** The device is performing its power-on self-testing procedure.	The LED should performing the testing blink sequence.

Table 41: Module Network Status Object – 'Network Status' attribute values

The host application can read out this attributes from the stack by sending the command *CIP Service request* as described on page 79.

Alternative the host can register to be notified about state changes by getting CIP service indications from the stack to the host. To register for change indications, the host has to set the attribute option flag 'CIP_FLG_TREAT_NOTIFY' to attribute 1 (Module Status) of the MNS object. Setting attribute option flags is described in section *Attribute Option Flags* on page 36. Registration to attribute 2 of the MNS object is not possible.

Once registered to attribute 1 (Module Status), the stack will sent CIP Service indications to the host with the service code CIP_SERVICE_SET_ATTRIBUTES_ALL with the current value of the module AND the network status attribute.

Note: The stack always sends both attributes 1 and 2 together, even the notify registration is done for attribute 1 only. Both attributes sent always together to the host with a CIP service indication 'Set Attribute All', even if just one data of the attributes is changed.

Note: The initial state of the Module Status and the Network Status attribute without stack configuration is 'No Power'. The state transition starts not before a valid configuration is applied.

2.4 Hilscher-specific CIP services

2.4.1 Common

2.4.1.1 Attribute Option Flags

CIP Attribute Option Flags control the stack on how a certain attribute is to be treated. On the one hand, attribute flags encode access rights and on the other, they control how an attribute is treated by the stack.

The Stack defines the following four levels of access permission with increasing authority:

- Bus access,
- User access,
- Admin access
- No access.

The higher access rights **imply** the lower ones. This means that if, e.g. bus access is permitted for a particular attribute, then also user access and admin access is granted. Access rights are maintained for both data directions, i.e. Read and Write, or, in CIP-terminology, Get and Set.

Name	Description
`CIP_FLG_SET_ACCESS_BUS   (0x0010)`	The attribute's value is modifiable over the network.
`CIP_FLG_SET_ACCESS_USER  (0x0020)`	The attribute's value is modifiable over the host interface.
`CIP_FLG_SET_ACCESS_ADMIN (0x0040)`	The attribute's value is modifiable internally by the stack itself.
`CIP_FLG_SET_ACCESS_NONE  (0x0080)`	The attribute's value is not modifiable.
`CIP_FLG_GET_ACCESS_BUS   (0x0100)`	The attribute's value is readable over the network.
`CIP_FLG_GET_ACCESS_USER  (0x0200)`	The attribute's value is readable over the host interface.
`CIP_FLG_GET_ACCESS_ADMIN (0x0400)`	The attribute's value is readable internally by the stack itself.
`CIP_FLG_GET_ACCESS_NONE  (0x0800)`	The attribute's value is not readable.
`CIP_FLG_TREAT_FORWARD    (0x1000)`	The stack will forward a SET service to this attribute to the host application with a CIP Service Indication. The host application hast to acknowledged the service and can accepted or rejected the SET service to this attribute.
`CIP_FLG_TREAT_NOTIFY     (0x2000)`	The stack will notify the host application about changes in the attribute value by means of a CIP 'SET' Service Indication. The host application cannot reject the service.
`CIP_FLG_TREAT_DISABLE    (0x4000)`	The attribute is disabled and will be treated as an unsupported attribute. It still may be exposed in the response to a Get_Attributes_All CIP request.
`CIP_FLG_TREAT_PROTECTED  (0x8000)`	Attribute is protected (not settable) when protection mode is active. (Not implemented).
`CIP_FLG_TREAT_RESERVED   (0x000F)`	The lower four bits D0..3 of the attribute flags are reserved for internal use of the stack. They could be set, but the meaning is internal to the stack and has to be ignored.

Table 42: Attribute option flags

Attribute Option Flags preset values

The table below shows all default attribute option flags as it is set at startup for each attribute. The host application can change the attribute option flags to a limited extent using the “Set Attribute Option” service.

- the flag `TREAT_NOTIFY` can be set by the host in general
- the flag `TREAT_FORWARD` can be set only for some particular attributes

Each colored cell shows, if the corresponding attribute option flag can be modified by the host or not.

		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D0..3
Class	Attribute	TREAT PROTECTED	TREAT DISABLE	TREAT NOTIFY	TREAT FORWARD	GET NONE	GET ADMIN	GET USER	GET BUS	SET NONE	SET ADMIN	SET USER	SET BUS	RESERVED
1	1	0	0	0	0	0	0	0	1	0	0	1	0	X
	2	0	0	0	0	0	0	0	1	0	0	1	0	X
	3	0	0	0	0	0	0	0	1	0	0	1	0	X
	4	0	0	0	0	0	0	0	1	0	0	1	0	X
	5	0	0	0	0	0	0	0	1	1	0	0	0	X
	6	0	0	0	0	0	0	0	1	0	0	1	0	X
	7	0	0	0	0	0	0	0	1	0	0	1	0	X
2	2	0	0	0	0	0	0	0	1	0	1	0	0	x
3	1	0	0	0	0	0	0	0	1	0	0	0	1	X
	2	0	0	0	0	0	0	0	1	0	0	0	1	X
	3	0	0	0	0	0	0	0	1	0	0	0	1	X
	4	0	0	0	0	0	0	0	1	0	0	0	1	X
	5	0	0	0	0	0	0	0	1	0	1	0	0	X
	6	0	1*	0	0	0	0	0	1	0	1	0	0	X
	7	0	1*	0	0	0	0	0	1	0	1	0	0	x
	8	0	1*	0	0	0	0	0	1	0	0	1	0	X
	9	0	1*	0	0	0	0	0	1	0	0	1	0	X
4	1	0	0	0	0	0	0	1	0	1	0	0	0	X
	2	0	0	0	0	0	0	1	0	1	0	0	0	X
	3	0	0	0	0	0	0	0	1	0	0	1*	1*	X
	4	0	0	0	0	0	0	0	1	1	0	0	0	X
5	1	0	0	0	0	0	0	0	1	0	1	0	0	X
	2	0	0	0	0	0	0	0	1	0	1	0	0	X
	3	0	0	0	0	0	0	0	1	0	1	0	0	X
	4	0	0	0	0	0	0	0	1	0	1	0	0	X
	5	0	0	0	0	0	0	0	1	0	1	0	0	X
	6	0	0	0	0	0	0	0	1	0	1	0	0	X
	7	0	0	0	0	0	0	0	1	0	1	0	0	X
	8	0	0	0	0	0	0	0	1	0	1	0	0	X
	9	0	0	0	0	0	0	0	1	0	0	0	1	X

		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D0..3
Class	Attribute	TREAT PROTECTED	TREAT DISABLE	TREAT NOTIFY	TREAT FORWARD	GET NONE	GET ADMIN	GET USER	GET BUS	SET NONE	SET ADMIN	SET USER	SET BUS	RESERVED
	12	0	0	0	0	0	0	0	1	0	0	0	1	X
	13	0	0	0	0	0	0	0	1	1	0	0	0	X
	14	0	0	0	0	0	0	0	1	0	1	0	0	X
	15	0	0	0	0	0	0	0	1	1	0	0	0	X
	16	0	0	0	0	0	0	0	1	0	1	0	0	X
	17	0	0	0	0	0	0	0	1	0	0	0	1	X
0x2B	1	0	0	0	0	0	0	0	1	0	0	0	1	X
	2	0	0	0	0	0	0	0	1	0	1	0	0	X
	3	0	0	0	0	0	0	0	1	0	1	0	0	X
0x402	1	0	0	0	0	0	0	1	0	0	1	0	0	X
	2	0	0	0	0	0	0	1	0	0	1	0	0	X
	3	0	0	0	0	0	1	0	0	0	1	0	0	X
	4	0	0	0	0	0	1	0	0	1	0	0	0	X
0x404	1	0	0	0	0	0	0	1	0	0	1	0	0	X
	2	0	0	0	0	0	0	1	0	0	1	0	0	X

Table 43: Class Instance Attribute option flags and pre-set values

Note: Class 3 - Attributes 6, 7, 8, 9 are disabled per default. Is enabled, when switch functionality for node and baud rate is enabled.

Note: Class 1 - Attributes 1, 2, 3, 4, 6, 7 can be made settable. This is only for very specific usage e.g. in production phase to program the device identity. To be conform to the norm, these attributes must be gettable only as it is set per default.

Note: Class 4 - Attribute 3 of class 4 is marked settable from bus when it is a consuming assembly and settable from user when it is a producing assembly.

Note: Class 5 - Attribute 17 is only settable for class instance 4 (COS)

The table below shows the attribute option flags of the class attributes.

		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D0..3
Class	Class Attribute	TREAT PROTECTED	TREAT DISABLE	TREAT FORWARD	TREAT NOTIFY	GET NONE	GET ADMIN	GET USER	GET BUS	SET NONE	SET ADMIN	SET USER	SET BUS	RESERVED
1..4	1	0	0	0	0	0	0	0	1	0	0	1	0	X
	2	0	0	0	0	0	0	0	1	0	0	1	0	X
	3	0	0	0	0	0	0	0	1	0	0	1	0	X
	6	0	0	0	0	0	0	0	1	0	0	1	0	X
	7	0	0	0	0	0	0	0	1	0	0	1	0	X
5,0x2B	1	0	0	0	0	0	0	0	1	0	1	0	0	X
	2	0	0	0	0	0	0	0	1	0	1	0	0	X
	3	0	0	0	0	0	0	0	1	0	1	0	0	X
	6	0	0	0	0	0	0	0	1	0	1	0	0	X
	7	0	0	0	0	0	0	0	1	0	0	1	0	X
0x402	1	0	0	0	0	0	0	1	0	0	1	0	0	X
	2	0	0	0	0	0	0	1	0	0	1	0	0	X
	3	0	0	0	0	0	0	1	0	0	1	0	0	X
	6	0	0	0	0	0	0	1	0	0	1	0	0	X
	7	0	0	0	0	0	0	1	0	0	1	0	0	X

Table 44: Class Attribute option flags and pre-set values

Note: The colored attribute option flags can be changed by the host application. By setting bit D8 GET_BUS to 0 and bit D9 GET_USER to 1, it is possible to hide an attribute from the network. You can do this to customize the stack, e.g. to rebuild a legacy device in which this class attribute is not supported.

2.4.1.2 Get Attribute Option Flags

The Hilscher-specific service “Get Attribute Option” (0xFF33) service returns the option flags of the targeted attribute.

Request Service Data Field Parameters:

The service does not accept any parameters.

Success Response Service Data Field Parameters:

Name	Byte Size	Description
Option Flags	2	This is a combination (Boolean OR) of Attribute Flags as described in Table 42.

Table 45: Hilscher Service – Get Attribute Option – Response data parameters

Unsuccessful Response Service Data Field Parameters:

The unsuccessful response does not provide any data.

2.4.1.3 Set Attribute Option Flags

The Hilscher-specific service “Set Attribute Option” (0xFF34) writes the option flags of the targeted attribute.

Request Service Data Field Parameters:

Name	Byte size	Description
Option Mask	2	This is a combination (Boolean OR) of Attribute Flags as described in Table 42. All bits set in this bitmask will be affected by the service: ▪ Attribute Flags not set in this mask will not be changed, despite of the value of the corresponding bit index in “Option Flags” ▪ Attribute Flags set in this mask, which are not set in “Option Flags” will be cleared from the Attribute Flags of the targeted attribute ▪ Attribute Flags set in this mask, which are also set in “Option Flags” will be set in the Attribute Flags of the targeted attribute.
Option Flags	2	This is a combination (Boolean OR) of Attribute Flags as described in Table 42. Attribute Flags that are set in this bit field will be set in the Attribute Flags of the targeted attribute, if the corresponding bit index is also set in “Option Mask” Attribute Flags that are not set in this bit field, but are set in “Option Mask”, will be cleared from the Attribute Flags of the targeted attribute.

Table 46: Hilscher Service – Set Attribute Option – Request data parameters

Note: The operation technically applying to the Flags of the targeted attribute is, in C:

```
usFlags &= ~usOptionMask;
usFlags |= usOptionFlags & usOptionMask;
```

Success Response Service Data Field Parameters

The service has no response parameters.

Unsuccessful Response Service Data Field Parameters

The unsuccessful response does not provide any data.

2.4.1.4 Attribute update Notify vs. Forward

The attribute option flags `CIP_FLAG_TREAT_NOTIFY` and `CIP_FLAG_TREAT_FORWARD` can be used to include the host application in case of an attribute update, which is handled by the CIP object library of the stack. The host application can register with one of these option flags to the stack handled attributes with respect to limitation of registration described in Table 43: Class Instance Attribute option flags and pre-set values on page 38.

Attribute Notify

The attribute flag `CIP_FLAG_TREAT_NOTIFY` can be used to inform the host application, when an attribute value was updated. A CIP service indication ‘Set Attribute’ is sent to the host. The new attribute value is sent to the host only when the stack has validated the attribute data and if it has changed. The host has to respond to the indication positively for completeness. The host cannot reject the Set attribute service that is registered for notification, even if the host sends a negative response to the stack, it will be ignored. The following diagram shows the process when a master writes an attribute. The CIP service indication is also generated when the stack itself updates the attribute internally. A typical use case for this is to register for the MNS object to implement the LED handling on host side.

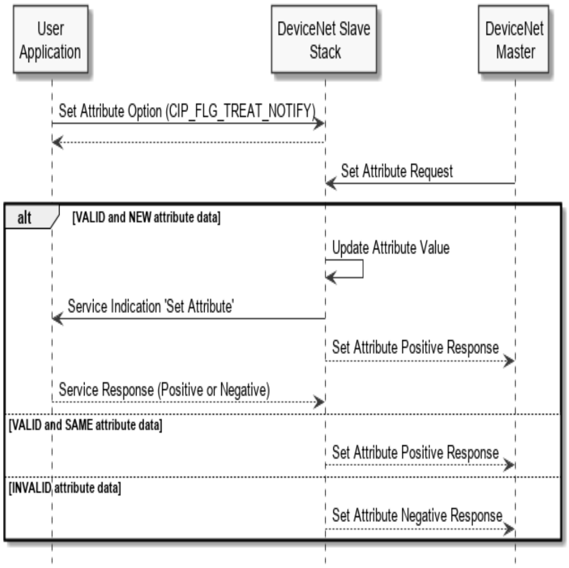
User Application
DeviceNet Slave Stack
DeviceNet Master
Set Attribute Option (CIP_FLG_TREAT_NOTIFY)
Set Attribute Request
alt
[VALID and NEW attribute data]
Update Attribute Value
Service Indication 'Set Attribute'
Set Attribute Positive Response
Service Response (Positive or Negative)
[VALID and SAME attribute data]
Set Attribute Positive Response
[INVALID attribute data]
Set Attribute Negative Response

Figure 3: Sequence Diagram – Attribute Update Notify

Attribute Forward

The flag `CIP_FLAG_TREAT_FORWARD` can be used for a confirmed update of an attribute. The host application can accept or reject the attribute data before the stack applies it. A typical use case for this is the registration for the Expected Packet Rate (EPR) attribute of the connection object when the host wants to allow the establishment of an IO connection as final instance.

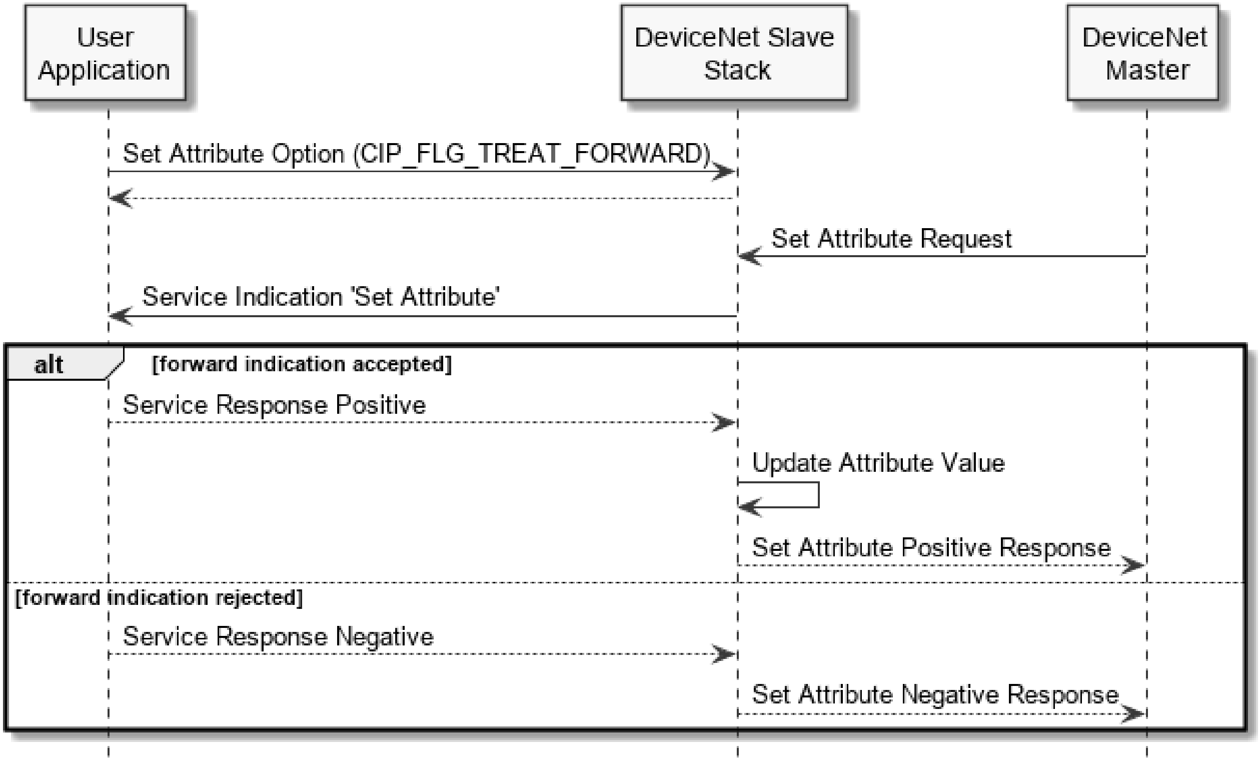
User Application
DeviceNet Slave Stack
DeviceNet Master
Set Attribute Option (CIP_FLG_TREAT_FORWARD)
Set Attribute Request
Service Indication 'Set Attribute'
alt
[forward indication accepted]
Service Response Positive
Update Attribute Value
Set Attribute Positive Response
[forward indication rejected]
Service Response Negative
Set Attribute Negative Response

Figure 4: Sequence Diagram – Attribute Update Forward

2.4.2 Identity Object

2.4.2.1 Modify Identity Status Attribute

The Identity Status Attribute 5 is a collection of 16 status bits representing general status information of a device. Normally, the stack is the owner and controls these status bits. Sometimes, it is required to modify some of these bits by the host application to indicate a device state that depends on the application state.

Following status bits are allowed to be modified by the host application

- Fault bits 8, 9, 10, 11
- Extended Device Status Bits 4, 5, 6, 7
- Configured Bit 2

The Hilscher specific CIP service “CIPHIL_SERVICE_MODIFY_STATUS” (0x0501) must be used for a particular modification of the Identity Status Attribute 5 flags.

The service “CIPHIL_SERVICE_MODIFY_STATUS” has to be send as a CIP service request packet to stack, as described in subsection *CIP Service sent from application* on page 78.

The modification of the status bits is done by a mask / value operation.

Service Data of CIPHIL_MODIFY_STATUS_T

Name	Byte size	Description
usMask	2	The ‘Mask’ decides which status flag of the identity status attribute shall be changed ▪ Status Flags not set in this mask will not be changed in the Identity status attribute, despite of the value of the corresponding bit index in “usValue” ▪ Status Flags set in this mask, which are not set in “usValue” will be cleared from the Identity status attribute ▪ Status Flags set in this mask, which are set in “usValue” will be set in the Identity status attribute
usValue	2	The ‘Value’ defines if the status flag of the identity status attribute shall be set or cleared ▪ Status flags that are set in this bit field will be set in the Status Flags of the Identity Status Attribute 5, if the corresponding bit index is also set in “usMask” ▪ Status Flags that are not set in this bit field, but are set in “usMask”, will be cleared from the Status Flags of the targeted attribute.

Table 47: Hilscher Service ’Modify Status’ – Request data parameters

The source code below shows as an example how to modify the ‘Minor Recoverable Fault’ bit in the Identity Status Attribute. It can be used in the same manner to set or clear the other status attribute bits.

```
uint32_t AppDNS_MinorRecoverableFault(APP_DATA_T* ptAppData, bool fSet)
{
  uint32_t ulRet = CIFX_NO_ERROR;
  DNS_PACKET_CIP_SERVICE_REQ_T* ptReq = AppDNS_GlobalPacket_Init(ptAppData);

  CIPHIL_MODIFY_STATUS_T* ptIdStatus = \
            (CIPHIL_MODIFY_STATUS_T*)&ptReq->tData.abData[0]

  if(true == fSet)
  {
    ptIdStatus->usMask = CIP_CLASS_IDENTITY_ATT_STATUS_FAULT_MINOR_RECOVERABLE;
    ptIdStatus->usValue = CIP_CLASS_IDENTITY_ATT_STATUS_FAULT_MINOR_RECOVERABLE;
  }
  else
  {
    ptIdStatus->usMask = CIP_CLASS_IDENTITY_ATT_STATUS_FAULT_MINOR_RECOVERABLE;
    ptIdStatus->usValue = 0;
  }

  /* prepare packet with CIP service data */
  ptReq->tData.ulService   = CIPHIL_SERVICE_MODIFY_STATUS;
  ptReq->tData.ulClass     = CIP_CLASS_IDENTITY;
  ptReq->tData.ulInstance  = 1;
  ptReq->tData.ulAttribute = CIP_CLASS_IDENTITY_ATT_STATUS;
  ptReq->tData.ulMember    = 0;


  /* Issue CIP Service Request */
  ptReq->tHead.ulCmd  = DNS_CMD_CIP_SERVICE_REQ;
  ptReq->tHead.ulLen  = DNS_CIP_SERVICE_REQ_SIZE + sizeof(CIPHIL_MODIFY_STATUS_T);

  ulRet = AppDNS_GlobalPacket_SendReceive(ptAppData);
  return ulRet;
}
```

Modification of Fault Bits 8 ... 11

CIP defines four types of general fault events, indicated by bit 8, 9, 10, 11 of the Identity Object Status Attribute 5.

Fault Type	Fault Example
Minor recoverable	an analog input device is sensing an input that exceeds the configured maximum input value a rotary switch of a device is moved to a position that it is not operational to the network
Minor unrecoverable	the battery-backed RAM within the device requires a battery replacement. The device will continue to function properly until power is cycled for the first time
Major recoverable	the configuration of the device is incorrect or incomplete.
Major unrecoverable	the device failed its ROM checksum process.

Table 48: List of Identity Status Attribute Fault Bits

The listed examples shows the fault types. Finally, the product developer defines the events, which causes any of these fault events.

Setting a minor / major fault has an implicit effect to the device LEDs. When setting a minor fault, the MNS or MS LED starts to blink red as long the minor fault is present. When setting a major fault the MNS or MS LED becomes solid red as long as the major fault is present.

The stack may also detect a fault condition. Therefore, the stack internally has a copy of each fault bit set by the host application and one fault bit when the stack has to report a fault event. Finally, the stack performs an OR operation and puts the result into the Identity Status Attribute 5. If the host sets a fault bit, it will be present as long as the host clears the bit or a channel init is performed.

Name	Fault bit 8	Fault bit 9	Fault bit 10	Fault bit 11
Fault bit set by host	X	X	X	X
Fault bit set by stack	Y	Y	Y	Y
Effective fault bit in Identity Status Attribute is a OR operation of host set fault bit and stack set fault bit	= X | Y	= X | Y	= X | Y	= X | Y

Table 49: Hilscher Service – Modify Status– stack operation

Modification of Extended Device Status Bits 4 ... 7

Normally, the stack handles these bits per default on its own as described in the CIP Specification Volume 1 Table 5A-2.6 [5]. Alternatively, the CIP specification describes that these bits can be handled in a user-specific way. Therefore, the API of protocol stacks commonly allows modifying these bits by the host application.

The modification is done in the same way as it is described for the fault bit modification with the specific service CIPHIL_SERVICE_MODIFY_STATUS.

Note: Once the host has written one of the bits 4..7 (regardless whether the bit is set or cleared), the stack will switch to handle these flags as user-specific and then the host is responsible for this handling. The only way to go back that the stack takes care about these flags and again handles it according Table 5A-2.6 is a reconfiguration / channel init procedure of the stack.

Modification of Configured Bit 2

Bit number 2 of the status attribute can be set to TRUE to indicate the application of the device has been configured to do something different than the “out-of-box” default. This shall not include configuration of the communications. The stack does not set this bit at all. After reset or reconfiguration, the stack will clear this bit and the application has to set the bit again if required.

3 Getting started

This section provides basic information of the Hilscher DeviceNet Slave stack in terms of configuring the stack, application behavior requirements and remanent data handling. It should be read first before start reading the chapter *Application interface* on page 66.

3.1 Loadable Firmware (LFW)

The DeviceNet Slave stack V5 is available as so called ‘Loadable Firmware’ (LFW). This means the application and the DeviceNet Slave protocol stack are running on different processors. While the host application runs on a computer typically equipped with an operating system (such as Microsoft Windows® or Linux) or an embedded host processor, the DeviceNet Slave protocol stack runs on the netX processor. The connection is accomplished via a driver (Hilscher cifX Driver, Hilscher netX Driver) as software layer on the host side and the AP task as software layer on the netX side. Both communicate via a dual port memory (DPM) into which they both can write and from which they both can read.

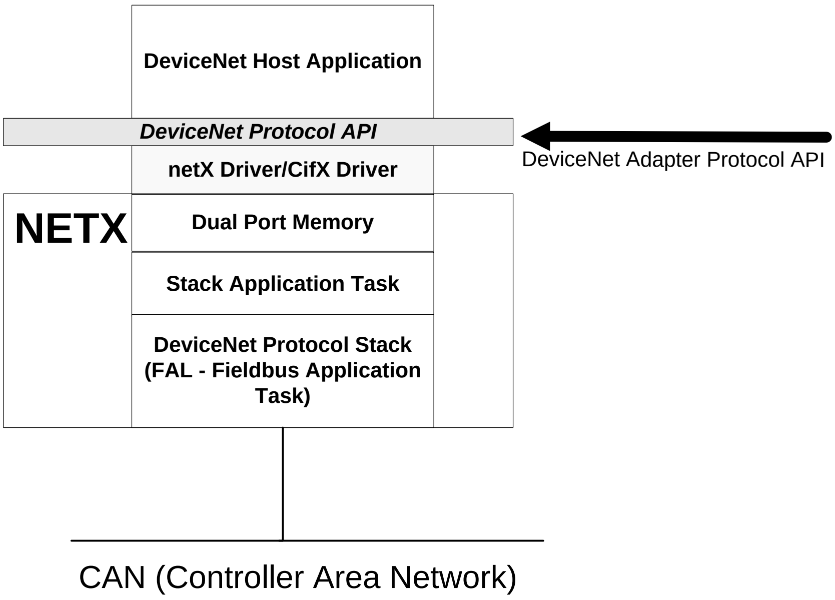
DeviceNet Host Application
DeviceNet Protocol API
DeviceNet Adapter Protocol API
netX Driver/CifX Driver
NETX
Dual Port Memory
Stack Application Task
DeviceNet Protocol Stack (FAL - Fieldbus Application Task)
CAN (Controller Area Network)

Figure 5: Loadable firmware

3.2 Process data direction convention

The definition of process data direction in terms of DeviceNet from perspective of the slave

- ‘produced’ data are process data transferred from the slave to the master
- ‘consumed’ data are process data transferred from the master to the slave

The process data direction in terms of input and output from DPM perspective (user application) is defined as:

- input data are process data received from the network
- output data are process data transferred to the network

DeviceNet	Dual port memory	Description
produced data	The application writes process data to the output image of the dual-port memory	Data sent from DeviceNet Slave to the DeviceNet Master
consumed data	The application reads process data the from input image of the dual-port memory	Data sent from DeviceNet Master to the DeviceNet Slave

Table 50: Process data direction convention

3.3 Cyclic data exchange

The interface for exchanging IO data with the DeviceNet Slave stack is the dual-port memory (DPM) containing the input and output area.

The standard method to exchange the receive data and transmit data between the communication stack and the host application via DPM are the two cifX API functions `xChannelIORead()` and `xChannelIOWrite()`. These functions are documented in [4]. They regulate the access to the input area and output area of the DPM by handshake flags which are described in [1]. Using this functions ensure the data consistency while exchanging the data between application and communication stack.

DeviceNet is using I/O connections to transfer I/O data. A DeviceNet Slave produces and consumes I/O data. The size of produced data or consumed data and the connection type is subject of the configuration of the DeviceNet Slave stack. The I/O data of the connections are stored in the input and output area of the DPM.

DPM area	Length (Byte)	Host access	Comment
Output block	Max. 255	Write/Read	Produced data: data send from the DeviceNet Slave to the master.
Input block	Max. 255	Read	Consumed data: data send from the master to DeviceNet Slave.

Table 51: Input and output data

Note: The application must obey the following rules, when exchanging IO data via DPM with the stack.

- The application always has to continue writing (updating) the DPM Output data regardless of the communications state with valid data.
- The application always has to continue reading (updating) the DPM Input data regardless of the communications state and handle them in the application context.

The DeviceNet slave stack supports several options for exchanging IO data between the application and the stack:

1. IO Exchange – Free Run, see page 49
2. IO Exchange – RX Data Received, see page 51
3. IO Exchange – Application synchronized POLL.Rsp, see page 53

The free run is the default and most common method to exchange IO data with the DeviceNet Slave stack. The other two options are used to reach reaction time synchronization.

3.3.1 IO Exchange – Free Run

The default method of exchanging IO data between a host application and the DeviceNet Slave stack via DPM is the ‘Free Run’ mode. In this mode, the network communication cycle and the application cycle run independently. The stack uses process data buffers for receive data and transmit data to manage the network cycle and the application cycle.

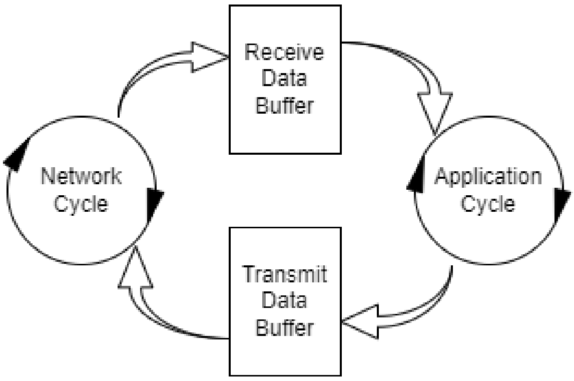

Figure 6: Free Run – data buffers to couple network and application cycle

Exchanging data in ‘Free Run’ between application and DeviceNet Slave stack is the default method and matches to most of all use cases.

The way to exchange the receive and transmit with the DeviceNet Slave stack is calling the cifX API functions `xChannelIORead()` and `xChannelIOWrite()` by the host application.

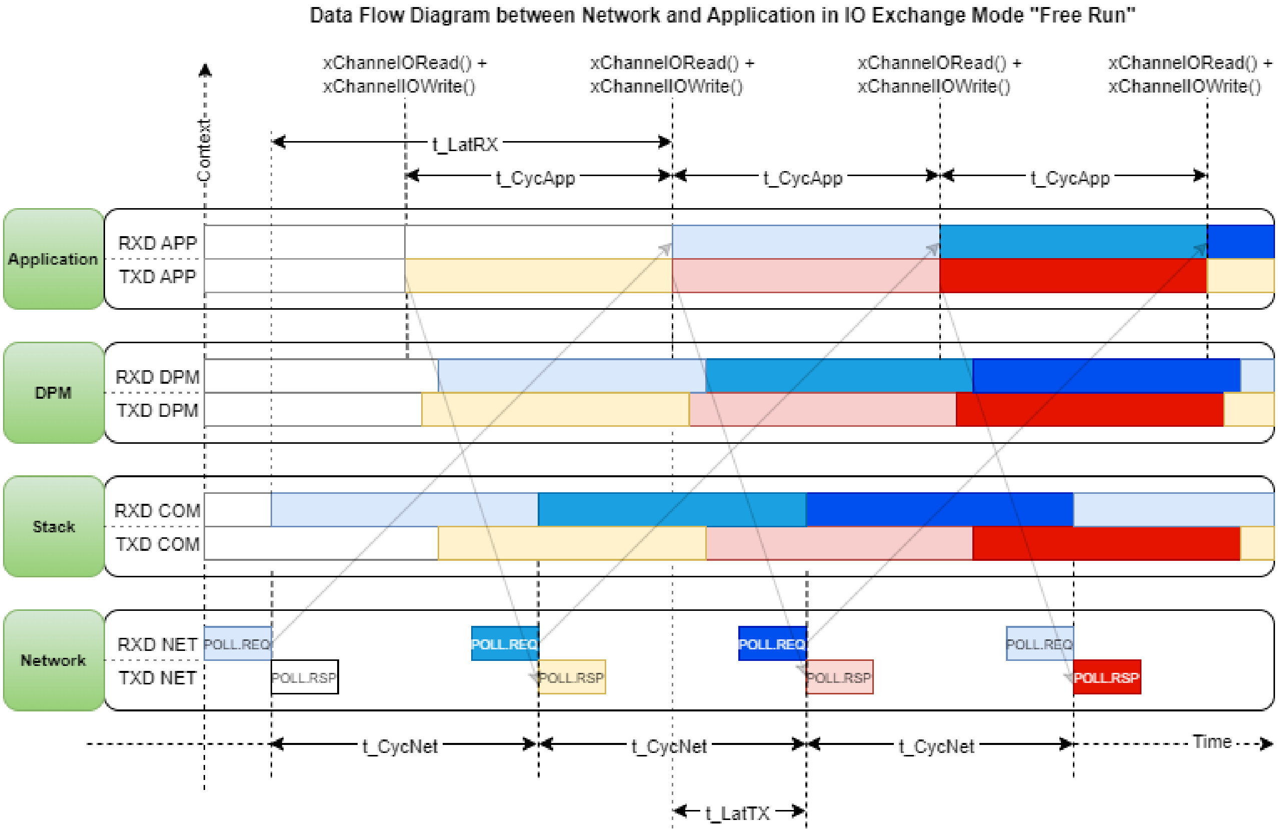
Data Flow Diagram between Network and Application in IO Exchange Mode "Free Run"
xChannelORead() + xChannelIOWrite()
xChannelORead() + xChannelIOWrite()
xChannelORead() + xChannelIOWrite()
xChannelORead() + xChannelIOWrite()
Context
t_LatRX
t_CycApp
t_CycApp
t_CycApp
Application
RXD APP
TXD APP
DPM
RXD DPM
TXD DPM
Stack
RXD COM
TXD COM
Network
RXD NET
TXD NET
POLL.REQ
POLL.RSP
POLL.REQ
POLL.RSP
POLL.REQ
POLL.RSP
POLL.REQ
POLL.RSP
t_CycNet
t_CycNet
t_CycNet
Time
t_LatTX

Figure 7: Free Run – data flow diagram

Time	Description
t_CycApp	Application cycle time: Time the application is calling the API functions to exchange transmit and receive data.
t_CycNet	Network cycle time: Time the master is exchanging data with the slave. In the diagram, the DeviceNet POLL connection is used to show the data transfer.
t_LatRX	Possible time until received data from the network will be available in the application.
t_LatTX	Possible time until application data will be sent to the network.

Table 52: FreeRun – data flow diagram timings

Figure 7 shows the data flow of the default IO Exchange mode ‘Free Run’ through the different layers. It shows the data flow from the network to the application for receiving data (RXD) and the direction from the application to the network for transmitting data (TXD).

The diagram shows an example where the network cycle and the application cycle have the **same** cycle time. It is just a timing snap shot. Both cycles (network and application) are running independently. This means the phase of both cycles can vary and drift over the time to each other.

According this drift, the time that it can take until received data will be available from the network to the application layer can vary. This time is named t_LatRX in the diagram. The max theoretical value for t_LatRX is 2 * tCycApp. The theoretical min value for t_LatRX is 1 * tCycApp.

The time t_LatTX in the diagram indicates the time it can take for the data written by the application to be sent to the network. This time is mostly determined by the network cycle “0 < t_LatTX <= t_CycNet”.

DeviceNet connection timings are configured in expected packet rates (EPR). The smallest configurable EPR unit in DeviceNet is 1 ms, which is the best achievable network cycle time. This depends on several conditions like the baud rate, the amount of IO data per connection or slave, the overall amount of IO data of the network and the response time of a slave.

The DeviceNet stack has a load limiter to restrict how frequently it is allowed to call `xChannelIORead()` and `xChannelIOWrite()` by the application. The reason for this load limiter is not to overload the communication stack with IO exchange calls from application and disturbing the network communication. Calling the functions `xChannelIORead()` or `xChannelIOWrite()` will fail with, when they are called more frequently than this limiter. The reason is that communication stack holds back the handshake flags when falling below this limiter.

For the DeviceNet stack V5.1, V5.2, and V5.3, the load limiter is 1 ms. Beginning with DeviceNet stack V5.4, the load limiter is decreased to 250 µs. This means the application cycle can run four times faster than the expected smallest network cycle of 1 ms.

Configuration

The Free Run mode is the default operation. There is no explicit configuration required reading configuration of the trigger types. The trigger type for `usPdInHskTriggerType` is set to `HIL_TRIGGER_TYPE_PDIN_NONE` and the trigger type for `usPdOutHskTriggerType` is set to `HIL_TRIGGER_TYPE_PDOUT_NONE` per default by the stack.

3.3.2 IO Exchange – RX Data Received

As stated in the description for the ‘Free Run’ mode, there is a possible latency until the receive data from the network become available to the application. Sometimes it is necessary to transfer the network receive data to the application context more quickly. This mode is only useful if the application is working in interrupt mode for the receive data.

Therefore, the DeviceNet stack support the trigger type on “RX DATA RECEIVED”. In this mode, the handshake flags for receive data remain on the stack side until new data is received. This allows the stack to copy and update the RX data immediately into the DPM because the access right to the DPM is by the stack. Once the RX data is updated, the stack returns the RX handshake flags to the application. If the DPM handling on the application side works in IRQ mode, the return of the RX handshake flags results in an IRQ to the application to read out the RX data immediately. The latency until RX data appear from the network to the application mainly exist of the processing time of the RX data by the stack and copy the data to the DPM and IRQ latency and processing time of function `xChannelIORead()` on application side.

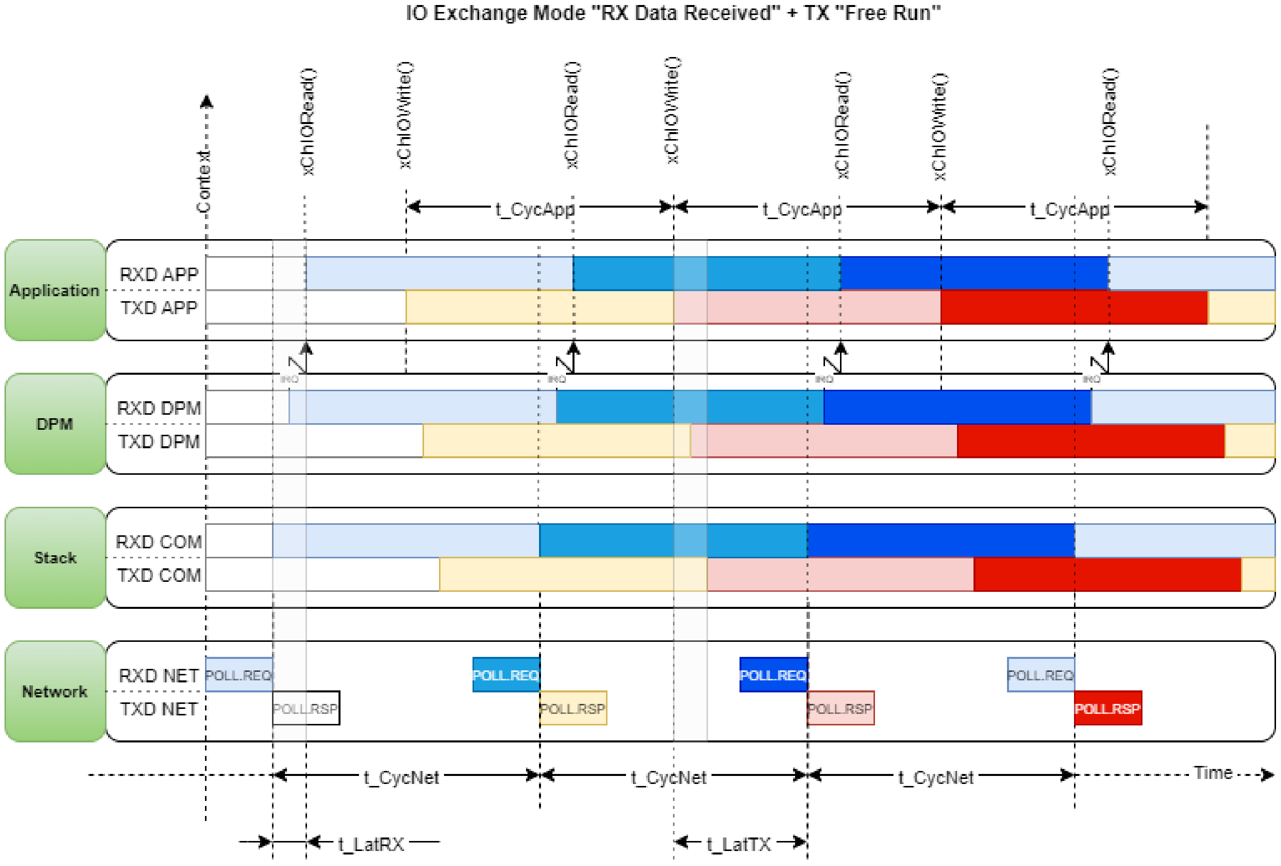

Figure 8: RX Data Received – data flow diagram

Time	Description
t_CycApp	Application cycle time: Time the application is calling the API functions to exchange transmit and receive data.
t_CycNet	Network cycle time: Time the master is exchanging data with the device. In the diagram, the DeviceNet POLL connection is used to show the data transfer.
t_LatRX	Possible time until received data from the network will be available in the application.
t_LatTX	Possible time until application data will be sent to the network.

Table 53: RX Data Received – data flow diagram timings

Configuration

This mode of operation requires to configure the trigger type for the receive data by sending the packet the command `HIL_SET_TRIGGER_TYPE_REQ`. The trigger type for `usPdInHskTriggerType` must be set to `HIL_TRIGGER_TYPE_PDIN_RX_DATA_RECEIVED` and the trigger type for `usPdOutHskTriggerType` must be set to `HIL_TRIGGER_TYPE_PDOUT_NONE.`

Configuring the trigger types (see also section *Get / Set Trigger Type* on page 107)

```
/**************************************************************************************
*! Function to Set the Trigger Type
*   \param ptAppData   pointer to APP_DATA_T structure
**************************************************************************************/
uint32_t AppDNS_SetTriggerType(APP_DATA_T* ptAppData)
{
  uint32_t ulRet = CIFX_NO_ERROR;
  HIL_SET_TRIGGER_TYPE_REQ_T* ptReq = AppDNS_GlobalPacket_Init(ptAppData);

  ptReq->tHead.ulCmd = HIL_SET_TRIGGER_TYPE_REQ;
  ptReq->tHead.ulLen = HIL_SET_TRIGGER_TYPE_REQ_SIZE;
  ptReq->tHead.ulSta = 0;

  ptReq->tData.usPdInHskTriggerType  = HIL_TRIGGER_TYPE_PDIN_RX_DATA_RECEIVED;
  ptReq->tData.usPdOutHskTriggerType = HIL_TRIGGER_TYPE_PDOUT_NONE;
  ptReq->tData.usSyncHskTriggerType  = HIL_TRIGGER_TYPE_SYNC_NONE;

  ulRet = AppDNS_GlobalPacket_SendReceive(ptAppData);

  return ulRet;
}
```

3.3.3 IO Exchange – Application synchronized POLL.Rsp

“Application synchronized POLL.Rsp” is a special IO Exchange variant of “RX DATA RECEIVED” as described in section *IO Exchange – RX Data Received*. This mode is only useful if the application is working in interrupt mode for the receive data.

If the application synchronized poll response is activated, the stack does not return the poll response immediately after processing the poll request by itself anymore. The transmission of the poll response message is bound to the update of the TX data by the application. As soon as the data from the poll request is hand over to the host via DPM, the stack waits for updated TX data provided by the application via DPM invoking `xChannelIOWrite()`. When new TX data are provided by the application, the stack immediately sends the poll response to the master with the data provided by the application. This option allows build an application to send TX data computed by previously received data. Waiting of the stack for updated TX data by the application only affects the poll response. It is the responsibility of the application to always provide updated TX data right in time. The stack does not generate a poll response implicitly no matter how long the application takes providing the updated TX data. The host application has full control over the bus cycle (response time) by controlling the poll response and must be aware of this. If the host application is slower than the configured timeout for the poll connection, the master is expected to repeat the poll request or release the connection, followed by a new connection attempt.

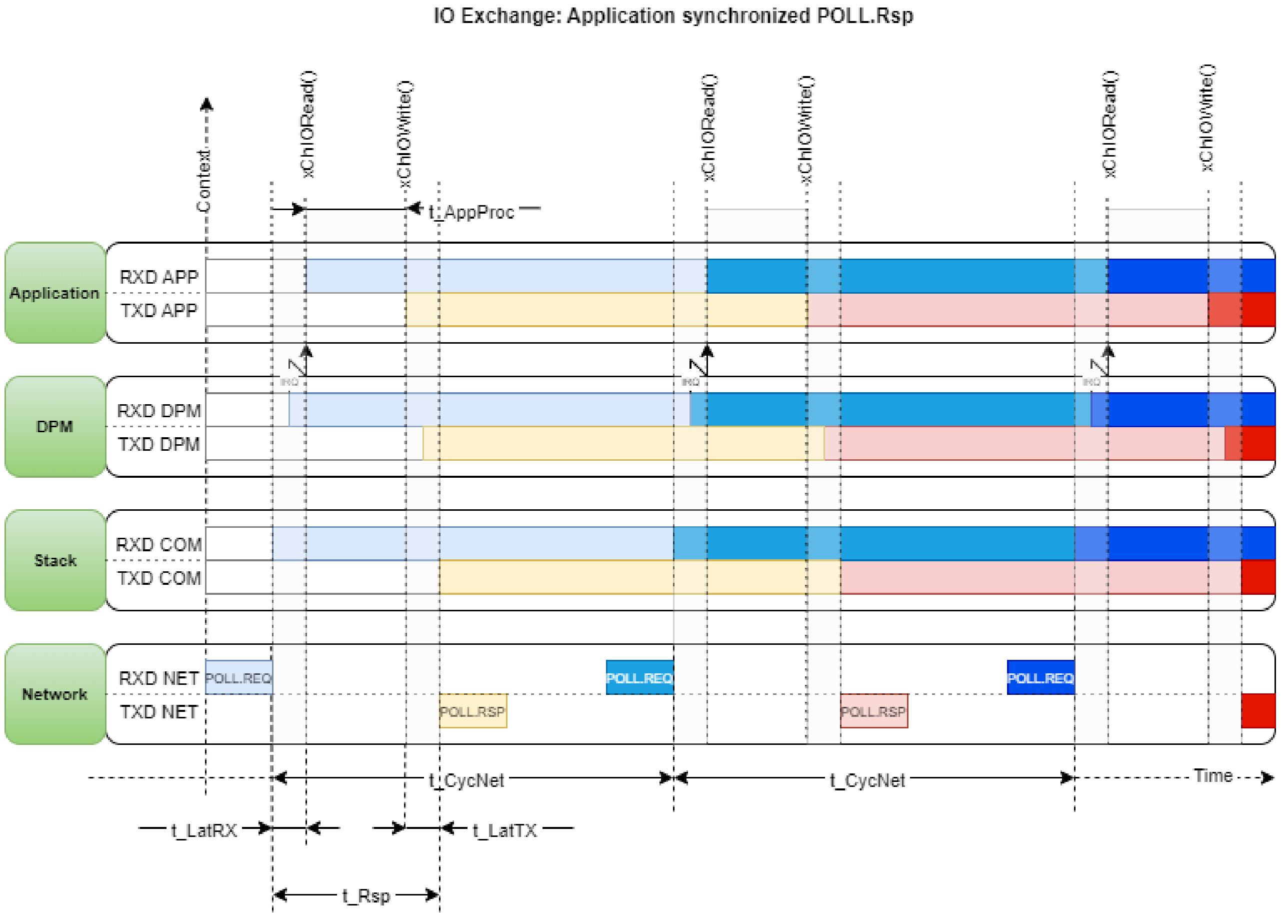

Figure 9: App synchronized poll.rsp – data flow diagram

Time	Description
t_CycNet	Network cycle time: Time the master is exchanging data with the device. In the diagram, the DeviceNet POLL connection is used to show the data transfer.
t_LatRX	Possible time until received data from the network will be available in the application. This time includes the processing of the RX data.
t_AppPoc	Application processing time.
t_LatTX	Possible time until application data will be sent to the network.
t_Rsp	The time t_Rsp in the diagram represents the poll response time from the last bit of a poll request telegram until the first bit of the poll response telegram on the network.

Table 54: App synchronized poll.rsp – data flow diagram timings

Configuration

This mode of operation requires to configure the trigger type for the receive data by sending the packet the command `HIL_SET_TRIGGER_TYPE_REQ`. The trigger type for `usPdInHskTriggerType` must be set to `HIL_TRIGGER_TYPE_PDIN_RX_DATA_RECEIVED` and the trigger type for `usPdOutHskTriggerType` must be set to `HIL_TRIGGER_TYPE_PDOUT_NONE.` Additional the configuration flag `DNS_CFG_FLAG_ENABLE_TXD_UPDATE_CONTROLLED_POLL_RES` must be set within the flag field `ulConfigFlags` of set configuration packet (see section *Set Configuration service* on page 69).

Configure the trigger types (see also section *Get / Set Trigger Type* on page 107).

```
/***************************************************************************************
*! Function to Set the Trigger Type
*   \param ptAppData   pointer to APP_DATA_T structure
***************************************************************************************/
uint32_t AppDNS_SetTriggerType(APP_DATA_T* ptAppData)
{
  uint32_t ulRet = CIFX_NO_ERROR;
  HIL_SET_TRIGGER_TYPE_REQ_T* ptReq = AppDNS_GlobalPacket_Init(ptAppData);

  ptReq->tHead.ulCmd = HIL_SET_TRIGGER_TYPE_REQ;
  ptReq->tHead.ulLen = HIL_SET_TRIGGER_TYPE_REQ_SIZE;
  ptReq->tHead.ulSta = 0;

  ptReq->tData.usPdInHskTriggerType  = HIL_TRIGGER_TYPE_PDIN_RX_DATA_RECEIVED;
  ptReq->tData.usPdOutHskTriggerType = HIL_TRIGGER_TYPE_PDOUT_NONE;
  ptReq->tData.usSyncHskTriggerType  = HIL_TRIGGER_TYPE_SYNC_NONE;

  ulRet = AppDNS_GlobalPacket_SendReceive(ptAppData);

  return ulRet;
}
```

3.4 Acyclic data exchange

For acyclic data exchange, DeviceNet uses the *Explicit Messaging services* (see page 77). For reading or writing acyclic data, the application can use the CIP services Get_Attribute or Set_Attribute service. Data are stored within objects, see section *Object classes* on page 18. All acyclic data exchange with the DeviceNet slave stack takes place via the mailboxes of the DPM.

The basic interface for interaction between the application and the DeviceNet Slave stack is the packet API. The packet API comply the Request (REQ) / Confirmation (CNF) and Indication (IND / Response (RES) principle. This means the application can send a request to the stack and will get a confirmation from the stack. The stack can send an indication to the application and the application has to send a corresponding response.

Note: The application must obey the following rules, when handling with acyclic packets via mailboxes.

- In case the application sends, a request to stack it always has to retrieve and handle the corresponding response. It has to evaluate the status of the response to check if the request was successful or not.
- The application has to retrieve indication and confirmations packets from the receive mailbox fast as possible, to avoid overflow or blocking situations of the mailbox system.
- To receive indications from the stack the application has to register to stack. Without registration, the stack will not send indications to application.
- In case the application has received an indication, it **always** has to send the corresponding response.
- The application has to send a response **soon as possible**, to prevent service timeouts to the network communication.
- The application has to send a response **within 3 seconds**. Otherwise, the stack will generate an error response.

3.5 Configuration methods

The following methods are available to configure the DeviceNet Slave stack 5:

3.5.1 Basic packet configuration set

In case of packet-based configuration, the host sends the configuration data via configuration packets from the host application to the stack. The configuration data are stored in volatile memory of the stack. The application has to send the configuration each time on startup or reconfiguration.

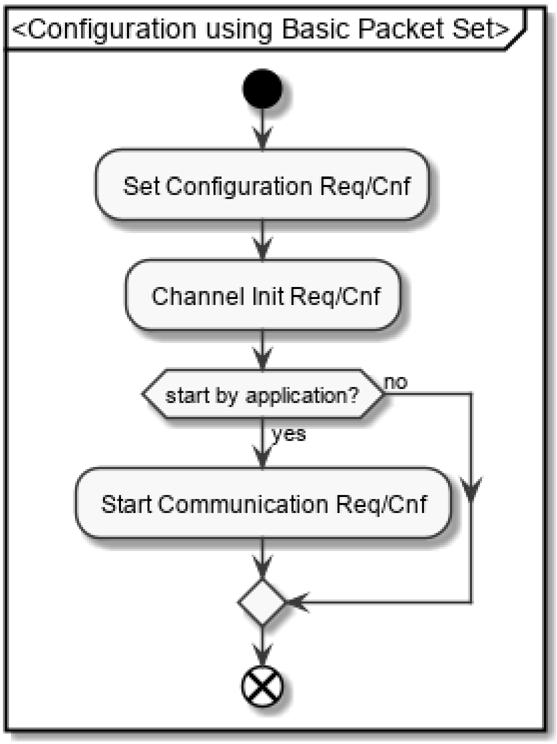
<Configuration using Basic Packet Set>
Set Configuration Req/Cnf
Channel Init Req/Cnf
start by application?
no
yes
Start Communication Req/Cnf

Figure 10: Basic packet configuration sequence

3.5.2 Extended packet configuration set

In case of extended packet-based configuration, the host sends the default set configuration packet. Afterwards, the host can the packets for registration used specific objects and/or registration of additionally assembly objects.

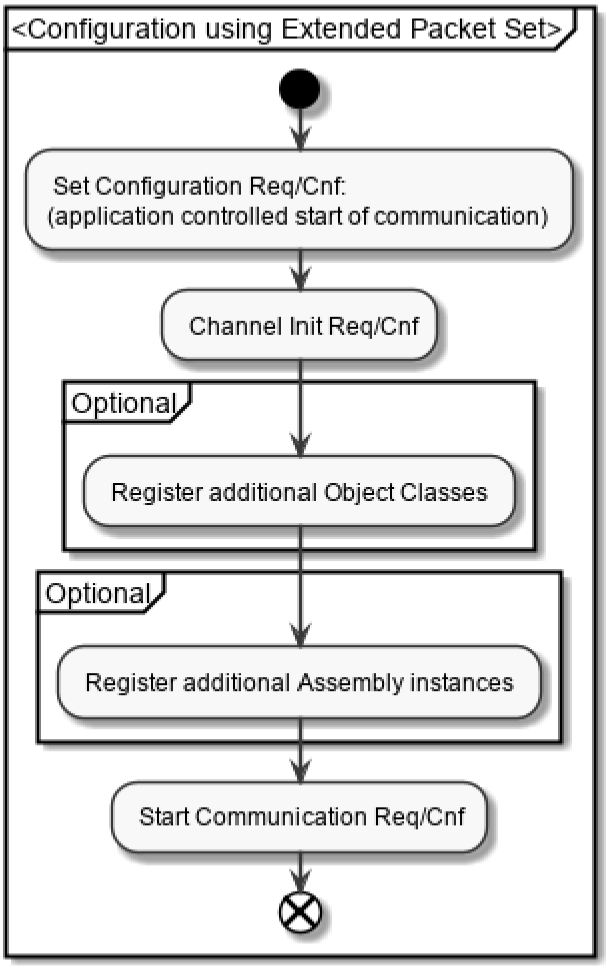
<Configuration using Extended Packet Set>
Set Configuration Req/Cnf: (application controlled start of communication)
Channel Init Req/Cnf
Optional
Register additional Object Classes
Optional
Register additional Assembly instances
Start Communication Req/Cnf

Figure 11: Extended packet configuration sequence

3.5.3 Data base configuration

The data base configuration is a non-volatile configuration. In this case, the stack automatically uploads a configuration file (config.nxd) from the flash file system at the start. The configuration tool generates the configuration file.

Note: When the stack is configured by a data base, it cannot be configured by the packet API.

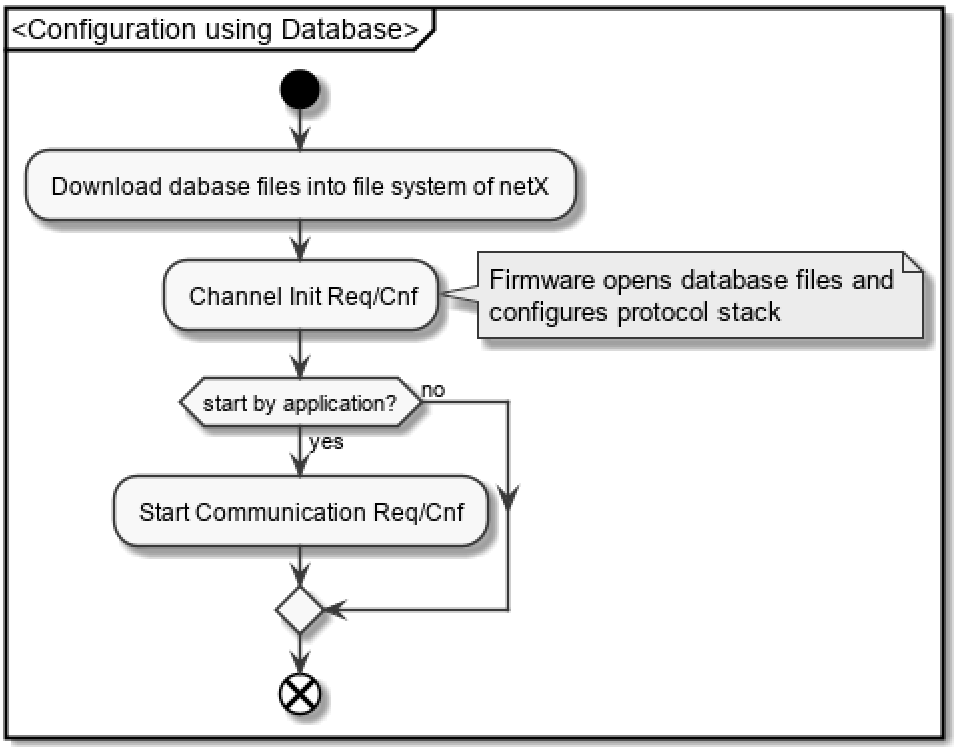
<Configuration using Database>
Download dabase files into file system of netX
Channel Init Req/Cnf
Firmware opens database files and configures protocol stack
start by application?
no
yes
Start Communication Req/Cnf

Figure 12: Data base configuration sequence

3.6 Host application behavior

The following diagram gives an overview of how the host application shall behave in different scenarios.

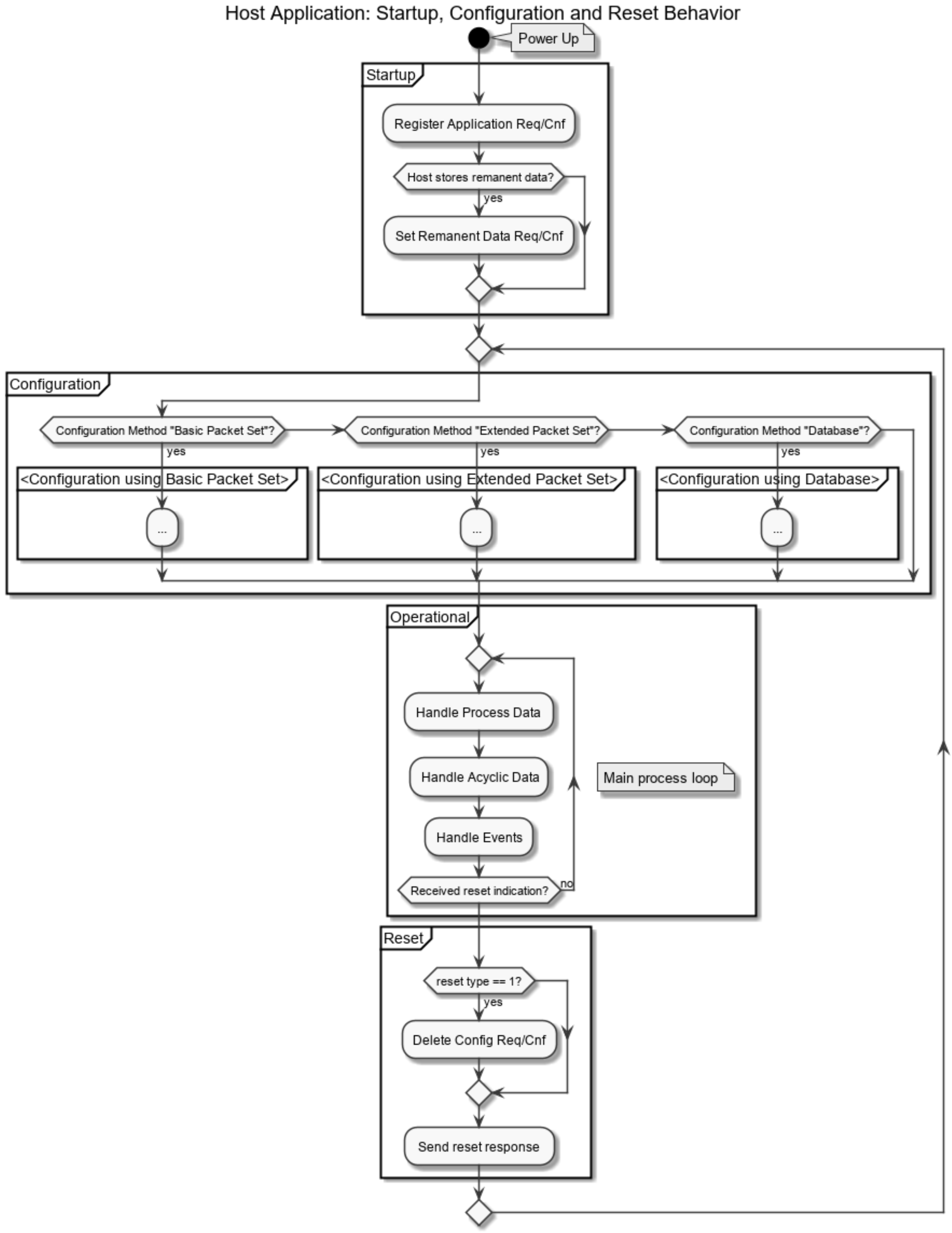
Host Application: Startup, Configuration and Reset Behavior
Power Up
Startup
Register Application Req/Cnf
Host stores remanent data?
yes
Set Remanent Data Req/Cnf
Configuration
Configuration Method "Basic Packet Set"?
yes
Configuration Method "Extended Packet Set"?
yes
Configuration Method "Database"?
yes
<Configuration using Basic Packet Set>
...
<Configuration using Extended Packet Set>
...
<Configuration using Database>
...
Operational
Handle Process Data
Handle Acyclic Data
Handle Events
Main process loop
Received reset indication?
no
Reset
reset type == 1?
yes
Delete Config Req/Cnf
Send reset response

Figure 13: Host application behavior

3.6.1 Startup

At the very first start after power-up, the host application can register itself to the stack. It is mandatory, if the host application wants to receive any indication from the stack.

Conditional it is required to send remanent data at startup. For more information, see section *Remanent data* on page 61

3.6.2 Configuration

The configuration behavior depends on the chosen configuration method as described in section *Configuration methods* on page 56.

- For using the Basic Configuration Packet Set, see section *Basic packet configuration* on page 56.
- For using the Extended Configuration Packet Set see section *Extended packet configuration* on page 57.
- For using a configuration file created by a configuration tool, see section *Data base configuration* on page 58.

3.6.3 Operational

In the operational state, the host application enters its main process loop. This includes IO data handling, protocol stack event handling.

3.6.4 Reset

The reset behavior is independent of the chosen configuration method. For more information regarding the “Reset Indication” and “Delete Config” handling, see section *Reset service* on page 91 and *Delete Config* on page 106.

3.7 Remanent data

3.7.1 Remanent data responsibility

When you design your application, you have to decide whether

- the **protocol stack** stores the remanent data (default) or
- the **application** stores the remanent data

If the system designer decides for application-side storage of remanent data, then the firmware's tag list must be modified as described in section 'Feature configuration via tag list' on page 113.

Note: The Hilscher DeviceNet Slave stack is capable of handling remanent data for the built-in CIP objects only. If the host application implements further CIP objects, which also bear non-volatile attributes, they will have to be handled completely in the scope of the host application. The latter case is not supported by mechanisms of the stack and thus is not subject of this manual.

Remanent data is stored by	Description
Protocol stack	The stack stores the remanent data **Requirements** The protocol stack requires access to non-volatile memory. **Firmware configuration** In the tag list "Remanent Data Responsibility" the tag "Remanent Data stored by Host" has to be set to disabled in the firmware file (`*.nxi or *.nxf`). This is the default setting in a firmware.
Application	The application stores the remanent data In case the host application stores remanent data, the protocol stack no longer accesses the Flash memory, but provides the complete remanent data block towards the host application per indication. The host application has to store the provided data with each indication and has to set this data back to the stack in the (re)configuration process. **Requirement** The application has to use the *Channel Component Information* service (`GENAP_GET_COMPONENT_IDS_REQ`) to get the information about the required size for remanent data of each protocol stack component. The application has to use the *Set Remanent Data* service (`HIL_SET_REMANENT_DATA_REQ`) and to support the *Store Remanent Data* service (`HIL_STORE_REMAMENT_DATA_IND`). **Firmware configuration** In the tag list "Remanent Data Responsibility" the tag "Remanent Data stored by Host" has to be set to **enabled** in the firmware file (`*.nxi`). **Configuration** The application has to use the *Set Remanent Data* service (`HIL_SET_REMANENT_DATA_REQ`) to provide the remanent data to each protocol stack component any time the host application starts up for the first time (e.g. when coming from power up) and before the application sends the *Set Configuration service* (page 69). For a state diagram, see section *Host application* on page 59. **During runtime** The stack component indicates to the application the *Store Remanent Data* service (`HIL_STORE_REMAMENT_DATA_IND`) each time remanent data has been changed. The stack component provides the remanent data as a block towards the application. The application has to store the remanent data with each indication. **Note:** For a detailed description of the Channel Component Information service, the Set Remanent Data, and the Store Remanent Data Indication, see reference [3].

Table 55: Protocol stack or host application stores remanent data

3.7.2 Remanent data state

The remanent data is either available/undeleted or unavailable/deleted. This state is not explicitly observable, but maintained by the protocol stack. This state is stored in the remanent data BLOB itself. If no such BLOB is available, the remanent data counts as unavailable/deleted.

3.7.3 Remanent data handling

At startup with a new configuration, the stack will set its remanent data to its default values or equal to the value submitted from host application. If a remanent value has been set via network, this value will be applied and will get priority against the value from the host configuration. If the host configuration changes, then the remanent data will be discarded and the host configuration will get priority again.

The option to set the MAC ID or Baud rate via network is maybe restricted if the source of configuration is marked as fixed. This typically happens, when rotary switches configure the MAC ID or Baud rate. In this case, the DeviceNet Slave stack will reject a set service via network with an appropriate error response.

3.7.4 Remanent data values

According the DeviceNet specification there are some operational data that may can be modified from network side from a master or commissioning tool that need to be stored remanent.

The DeviceNet Slave stack stores this particular data per default by its own into a non-volatile memory. Currently, three parameters may need to be stored remanent.

Node Address

According the DeviceNet specification Attribute 1 of the DeviceNet Object, the ‘MAC ID’, can be written via network by a master or commissioning tool. In this case, the DeviceNet Slave has to store the new MAC ID remanent and apply them.

Baud Rate

According the DeviceNet specification Attribute 2 of the DeviceNet Object, the Baud rate, can be written via network by a master or commissioning tool. In this case, the DeviceNet Slave has to store the new Baud rate remanent and apply them after a reset.

Bus Off Interrupt

According the DeviceNet specification Attribute 3 of the DeviceNet Object, the ‘BOI’, can be written via network by a master or commissioning tool. In this case, the DeviceNet Slave has to store the new value remanent. The BOI attribute defines the behavior of the stack in case a CAN bus off event appears. This is a heavy critical link fault, which hints to serious network problems. Therefore, the default behavior according to the DeviceNet specification is to go offline from the network into Communication Faulted state. Normally, this state should be recovered by manual intervention like reset or network power resume. In case this attribute is set, the stack will try to resume the communication by itself.

Note: If rotary switches configure the MAC ID and Baud rate, they cannot be set from network. In this case, the DeviceNet Slave stack will reject a set service via network with an appropriate error response.

3.8 Device data

The Flash Device Label (FDL) contains device data. The device-specific data in the Flash Device Label is set during production of the device. During start of the firmware, the firmware reads this data into the Device Data Provider (DDP).

The following table lists the device data and describes how the DeviceNet slave stack maps this data to DeviceNet.

Name	DeviceNet mapping
Manufacturer ID	Not mapped to DeviceNet.
Device class	Not mapped to DeviceNet.
Device number	Not mapped to DeviceNet.
Serial number	Mapped to Identity Object, Attribute 6.
Hardware compatibility number	Not mapped to DeviceNet.
Hardware revision number	Not mapped to DeviceNet.
Production date	Not mapped to DeviceNet.

Table 56: Basic Device Data in the Flash Device Label

The Flash Device Label offers the possibility to store OEM-specific device data. The following table lists the mapping of the OEM-specific device data to DeviceNet.

Name	DeviceNet mapping	DeviceNet coding
OEM data option flags	Each flag determines whether the parameter value from the Basic Device Data or from OEM identification is to be used. Bit 0: If 1, `OEM Serial number` is valid. Bit 1: If 1, `OEM order number` is valid. Bit 2: `OEM hardware revision` is valid. Bit 3: `OEM production date/time` is valid.	-
OEM serial number	Mapped to Attribute 6 of the Identity Object.	Null-terminated C string with decimal values "1" … "4294967295"
OEM order number	Not mapped to DeviceNet.	-
OEM hardware revision	Not mapped to DeviceNet.	-
OEM production date/time	Not mapped to DeviceNet.	-

Table 57: OEM identification in the Flash Device Label

Note: Although the DeviceNet slave stack is using only the serial number from the OEM parameter, the host application has to set all OEM option flags. If OEM-specific device data is required to be used all enabling bits (bits 0 – 3) have to be set. It is not possible to set a single parameter only. All parameters have to be set simultaneously.

3.8.1 Device Serial Number

The device serial number as reflected in the CIP Identity Object, attribute 6, together with the vendor ID, forms a unique identifier for each device on any CIP network. Each vendor is responsible for guaranteeing the uniqueness of the serial number across all of its devices.

Per default, the protocol stack applies the serial number from the underlying Device Data Provider (DDP), which in turn fetches it from either the SecMem or FDL data sources. The host application cannot set the serial number attribute directly. In the `DNS_CMD_SET_CONFIGURATION_REQ` it has to parameterize a value of zero for the serial number. This should be fine for most applications.

Anyway, if the host application seeks to set its own serial number, e.g. if no SecMem is available, the firmware has to be taglist-modified accordingly (refer to section on page 111). Then, it uses the DDP's OEM serial number attribute to set a custom serial number and render this data valid and finally, set the DDP active. The following pseudo code shows this approach:

```
/* optionally when initial DDP state is passive:
   set additional (OEM) DDP base device parameters: serial number
*/
HIL_DDP_SERVICE_SET_REQ_T* ptReq          = (HIL_DDP_SERVICE_SET_REQ_T*)&myPacket;
char*                      szSerialNumber = "76543";

memset(&ptReq->tHead, 0, sizeof(ptReq->tHead));

ptReq->tHead.ulCmd      = HIL_DDP_SERVICE_SET_REQ;
ptReq->tHead.ulLen      = sizeof(ptReq->tData.ulDataType) + strlen(szSerialNumber) + 1;
ptReq->tData.ulDataType = HIL_DDP_SERVICE_DATATYPE_OEM_SERIALNUMBER;

memcpy(ptReq->tData.uDataType.szString, szSerialNumber, strlen(szSerialNumber) + 1);

SendPacket(&myPacket, mychannel);

/* also render the OEM serial number "valid" in the corresponding bit field */
ptReq->tHead.ulCmd             = HIL_DDP_SERVICE_SET_REQ;
ptReq->tHead.ulLen             = sizeof(ptReq->tData.ulDataType)
                                 + sizeof(ptReq->tData.uDataType.ulValue);
ptReq->tData.ulDataType        = HIL_DDP_SERVICE_DATATYPE_OEM_OPTIONS;
ptReq->tData.uDataType.ulValue = 0xF;    /* set all OEM bits valid */

SendPacket(&myPacket, mychannel);

/* required when initial DPP state is passive: Set DDP active now */
ptReq->tHead.ulCmd             = HIL_DDP_SERVICE_SET_REQ;
ptReq->tHead.ulLen             = sizeof(ptReq->tData.ulDataType)
                                 + sizeof(ptReq->tData.uDataType.ulValue);
ptReq->tData.ulDataType        = HIL_DDP_SERVICE_DATATYPE_STATE;
ptReq->tData.uDataType.ulValue = HIL_DDP_SERVICE_STATE_ACTIVE;

SendPacket(&myPacket, mychannel);
```

Note: OEMization is DeviceNet-specific. Other software components will reflect the Hilscher serial number from the base device data anyway instead of the OEM-data, e.g. the netIDENT / EtherNetDeviceConfig subsystem.

4 Application interface

This chapter defines the user application interface of the DeviceNet-Slave Stack via ‘Communication Channel Interface’ of the Dual port Memory.

The ‘Communication Channel Interface’ is the Hilscher’s dual-port memory interface for field buses or other communication stacks. A typical application is when using PC cards or COM modules with a discrete DPM and accessing the DeviceNet Slave via Driver API.

4.1 Service overview

The table below lists all packet-based services supported by the DeviceNet Slave stack.

Topic	Service	Command code	Type	Page
Configuration	Set Configuration service	0xB100 / 0xB101	REQ/CNF	69
	Register Class Service	0xB106 / 0xB107	REQ/CNF	81
	Unregister Class service	0xB108 / 0xB109	REQ/CNF	86
	Create Assembly service	0xB10A / 0xB10B	REQ/CNF	75
Explicit Messaging	CIP Service sent from application	0xB102 / 0xB103	REQ/CNF	78
	CIP Service sent by a master	0xB104 / 0xB105	IND/RES	88
	Reset Service	0xB10C / 0xB10D	IND/RES	91
Diagnosis	Diag service	0xB10E / 0xB10F	REQ/CNF	96
Hilscher Common services	Channel Init	0x2F80 / 0x2F81	REQ/CNF	106
	Register / Unregister Application	0x2F10 / 0x2F11 0x2F12 / 0x2F13	REQ/CNF	106
	Get / Set Watchdog Time	0x2F02 / 0x2F03 0x2F04 / 0x2F05	REQ/CNF	106
	Delete Config	0x2F14 / 0x2F15	REQ/CNF	106
	Start / Stop Communication	0x2F30 / 0x2F31	REQ/CNF	107
	Lock / Unlock Configuration	0x2F32 / 0x2F33	REQ/CNF	107
	Get DPM I/O Information	0x2F0C / 0x2F0D	REQ/CNF	107
	Get / Set Trigger Type	0x2F92 / 0x2F93	REQ/CNF	107
	Firmware Identification	0x1EB6 / 0x1EB7	REQ/CNF	107
	Set Remanent Data	0x2F8C / 0x2F8D	REQ/CNF	-

Table 58: Service overview

4.2 Configuration services

4.2.1 Basic configuration sequence

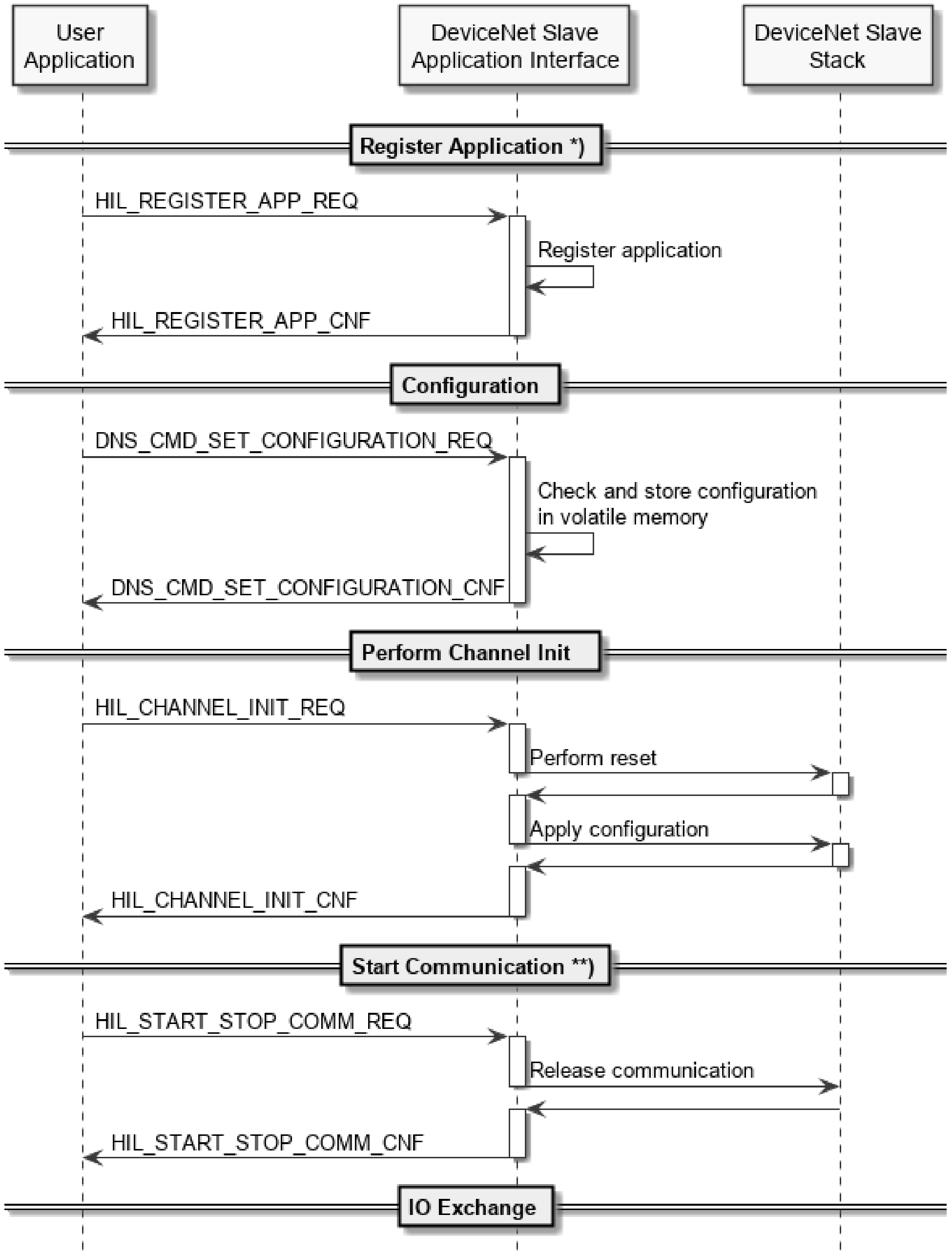

Figure 14: DNS_CMD_SET_CONFIGURATION_REQ/CNF - Basic configuration sequence diagram

Once the DeviceNet Slave stack is started, it must be initialized with appropriate network parameters like Baud rate, MAC ID, Device Ident information, IO Produce/Consumed size etc. The diagram above shows the basic configuration flow to configure the stack.

Step	Description
Register Application *)	At startup, the host application can register itself to the stack in order to receive any indication packets from the stack. This is necessary for example to receive CIP Service Indications. The registration is optional. It is mandatory if the host application wants to receive any indications from stack.
Set Configuration	The set configuration service provides the basic configuration to stack. The stack evaluates the data and stores it into volatile memory. At this point, the new configuration is not yet applied; it is just verified and stored.
Channel Init	This service causes the stack to apply the new configuration.
Start Communication **)	After configuration, the communication must be released with the Start Communication service. This service is required, if the “Application Controlled” startup is configured.
IO Exchange	Finally, the application has to exchange IO data with the stack.

Table 59: Basic configuration steps

4.2.2 Set Configuration service

4.2.2.1 Set Configuration request

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t		sizeof(DNS_SET_CONFIGURATION_V1_REQ)
ulCmd	uint32_t	0xB100	DNS_CMD_SET_CONFIGURATION_REQ
Data			
ulVersion	uint32_t	1	Version of Set Configuration
unCfg.tV1			
ulSystemFlags	uint32_t	bit field	System Flags For a description, see Table 61 on page 72.
ulWdgTime	uint32_t	0, 20-65535	Host Watchdog Time in milliseconds 0: watchdog is disabled min = 20; default = 1000; max = 65535
ulNodeId	uint32_t	0-63	Node id MAC ID of the DeviceNet Slave in the network.
ulBaudrate	uint32_t	0;1;2	Baud rate 0: 125 kBit/s (default) 1: 250 kBit/s 2: 500 kBit/s
ulConfigFlags	uint32_t	0	Configuration flags For a description see Table 62 on page 73
ulObjectFlags	uint32_t	0	Object Configuration flags These flags are not used set 0
usVendorId	uint16_t	1-65535 (default 283)	DeviceNet specific unique number, which is fixed by the ODVA for each DeviceNet manufacturer. The DNS task itself uses this ID during the Duplicate MAC-ID check phase and within each sent Duplicate MAC-Id check response. The value range of this variable is not limited. The Hilscher ID is 283 decimal.
usDeviceType	uint16_t	0-65535 (default 12)	Identification of the general type of product. The Hilscher standard value is 12 denoting a Communications Adapter.
usProductCode	uint16_t	1-…	Identification of a particular product within a defined device type.
bMinorRev	uint8_t	1-255 (default 1)	First part of the revision identifying the revision of the DNS device. The revision attribute consists of Major and Minor Revisions and they are typically displayed as major.minor.
bMajorRev	uint8_t	1-127 (default 1)	Second part of the revision. The Major Revision attribute is limited to 7 bits. The eighth bit is reserved by DeviceNet and must have a default value of zero.
ulSerialnumber	uint32_t	0	Deprecated. This value has to be set to zero. The firmware will apply the serial number as stored in the Device Data Provider (DDP), which in turn fetches it either from the SecMem or from FDL data sources. Refer to section 3.8 / 3.8.1 for details.
abReserved[3]	uint8_t	0	Reserved set to 0
bProductNameLen	uint8_t	1-32	Length of abProdName string. The maximum number of characters in this string is 32.

Variable	Type	Value / Range	Description
abProductName[32]	uint8_t[]	Readable ASCII Characters	ASCII text string that should represent a short description of the product/product family. The maximum number of characters in this string is 32. The number of characters must be set in the variable bProdNameLen. **Note**: The firmware does not check if the set of characters is in the readable range. This is in responsibility of the application.
uProduceAsInstance	uint32_t	1 … 255 Default 101	Instance number of input assembly (Slave to Master) **Note:** The value of ulProduceAsInstance must differ from the value of ulConsumeAsInstance. **Note:** The host application is responsible to choose an assembly ID from a proper range as defined in CIP
ulProduceAsFlags	uint32_t	bit field	0: default For details see: Table 65: Assembly configuration flags on page 76
ulProduceAsSize	uint32_t	0 … 255	Number of input bytes the DNS task shall produce in the view of a master for each established connection. The bytes, which shall be produced then, must be handed over in the send data area of the dual-port memory.
ulConsumeAsInstance	uint32_t	1 … 255 Default 100	Instance number of output assembly (Master to Slave) **Note**: The value of ulProduceAsInstance must differ from the value of ulConsumeAsInstance. **Note**: The host application is responsible to choose an assembly ID from a proper range as defined in CIP
ulConsumeAsFlags	uint32_t	bit field	0: default For the consume assembly optionally bit D8 can be set to map Run / Idle information into input image of the DPM For details see: Table 65: Assembly configuration flags on page 76
ulConsumeAsSize	uint32_t	0 … 255	Number of output bytes the DNS task shall consume in the view of a master for each established connection. The bytes which are received are handed over in the receive data area of the dual-port memory. **Note:** The consumed data will be placed at offset 0 of the input image of the DPM. If bit D8 of ulConsumeAsFlags is the the 4 byte Run/Idle header is placed at offset 0 of the DPM and the assembly data will start at offset 4 of the DPM. See also section *Process data status* on page 111.

Table 60: DNS_CMD_SET_CONFIGURATION_REQ – Set Configuration request

Packet structure reference

```
/******************************************************************************/
/*                             System flags                                   */
/******************************************************************************/
#define DNS_SYS_FLG_MANUAL_START                                     0x00000001
#define DNS_SYS_FLG_ADR_SW_ENABLE                                    0x00000010
#define DNS_SYS_FLG_BAUD_SW_ENABLE                                   0x00000020
#define DNS_SYS_FLG_RESERVED                                         0xFFFFFFCE

/******************************************************************************/
/*                             Baudrates                                      */
/******************************************************************************/
#define   DNS_BAUDRATE_125kB                                                  0
#define   DNS_BAUDRATE_250kB                                                  1
#define   DNS_BAUDRATE_500kB                                                  2


/******************************************************************************/
/*                            Set Configuration Packet                        */
/******************************************************************************/
```

```
#define DNS_CONFIGURATION_V1  1

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_CONFIGURATION_V1_Ttag
{
  uint32_t ulSystemFlags;
  uint32_t ulWdgTime;

  uint32_t ulNodeId;
  uint32_t ulBaudrate;

  uint32_t ulConfigFlags;
  uint32_t ulObjectFlags;

  uint16_t usVendorId;
  uint16_t usDeviceType;
  uint16_t usProductCode;
  uint8_t  bMinorRev;
  uint8_t  bMajorRev;
  uint32_t ulSerialNumber;
  uint8_t  abReserved[3];
  uint8_t  bProductNameLen;
  uint8_t  abProductName[32];

  uint32_t ulProduceAsInstance;
  uint32_t ulProduceAsFlags;
  uint32_t ulProduceAsSize;

  uint32_t ulConsumeAsInstance;
  uint32_t ulConsumeAsFlags;
  uint32_t ulConsumeAsSize;

} DNS_CONFIGURATION_V1_T;


/* Request Packet Data*/
typedef __HIL_PACKED_PRE union __HIL_PACKED_POST DNS_SET_CONFIGURATION_REQ_Utag
{
  DNS_CONFIGURATION_V1_T tV1;

} DNS_SET_CONFIGURATION_REQ_U;


typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_SET_CONFIGURATION_REQ_Ttag
{
  uint32_t ulVersion;
  DNS_SET_CONFIGURATION_REQ_U unCfg;
} DNS_SET_CONFIGURATION_REQ_T;


typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_PACKET_SET_CONFIGURATION_REQ_Ttag
{
  HIL_PACKET_HEADER_T         tHead;
  DNS_SET_CONFIGURATION_REQ_T tData;

} DNS_PACKET_SET_CONFIGURATION_REQ_T;

#define DNS_SET_CONFIGURATION_V1_REQ_SIZE (sizeof(DNS_CONFIGURATION_V1_T)+4)
```

Parameter: ulSystemFlags The System Flags define the startup behavior of the stack. They also define whether the hardware switches are enabled.

Bit	Description
0	Manual Start Enable, MSK_DNS_SYS_FLG_MANUAL_START: 0: Automatic. Network connections are opened automatically regardless of the state of the host application. 1: Application controlled. The firmware is forced to wait for the host application to set the Application Ready flag in the communication change of state register (see reference [1]).
4	Address Switch Enable, MSK_DNS_SYS_FLG_ADR_SW_ENABLE: If this bit is set, the handling mechanism of the address switch is activated and attribute 6 and 8 of the DeviceNet object are enabled.
5	Baud rate Switch Enable, MSK_DNS_SYS_FLG_BAUD_SW_ENABLE: If this bit is set, the handling mechanism of baud rate switch is activated and attribute 7 and 9 of the DeviceNet object are enabled.
Others	Reserved.

Table 61: Set Configuration parameter 'ulSystemFlags

Parameter: ulConfigFlags

These flags configure stack specific behavior.

Bit	Description
0 ... 7	Reserved.
8, 9	Message Body Format Bit 8 and 9 together define the message body format: 00: Message Body Format 8/8 (default) (Class/Instance) 01: Message Body Format 8/16 (Class/Instance) 10: Message Body Format 16/16 (Class/Instance) 11: Message Body Format 16/8 (Class/Instance) The message body format specifies the address range of class and instance. The default of 8/8 allows a highest address of 255 for class and instance where 16/16 allows a highest address of 65535 for classes and instance. **Note:** The DeviceNet master must also support the configured message body format.
10, 11	Reserved.
12	0: POLL connection is allowed (default) 1: POLL connection is not allowed If this bit is set the master cannot open an IO connection of the type POLL
13	0: STROBE connection is allowed (default) 1: STROBE connection is not allowed If this bit is set the master cannot open an IO connection of the type STROBE
14	0: COS connection is allowed (default) 1: COS connection is not allowed If this bit is set the master cannot open an IO connection of the type COS
15	0: CYC connection is allowed (default) 1: CYC connection is not allowed If this bit is set the master cannot open an IO connection of the type CYC
16	0: Application controlled POLL response disabled (default) 1: Application controlled POLL response enabled (Refer to section *IO Exchange – Application synchronized POLL.Rsp* on page 53)
17 … 30	Reserved

31	DNS_CFG_FLAG_INDICATION_NETWORK_POWER When this flag is set, the stack will forward the 24V Network Power state within the reset indication packet. For details, see chapter ‘Reset service’ on page 91.

Table 62: Set Configuration parameter ulConfigFlags

Source code example

```
void DnsUser_SetConfig_Req(DNS_PACKET_SET_CONFIGURATION_REQ_T *ptSetCfgReq)
{
  DNS_CONFIGURATION_V1_T *ptCfg = &ptSetCfgReq->tData.unCfg.tV1;

  /* Set the packet command, length and DNS configuration version */
  ptSetCfgReq->tHead.ulCmd = DNS_CMD_SET_CONFIGURATION_REQ;
  ptSetCfgReq->tHead.ulLen = sizeof(DNS_SET_CONFIGURATION_V1_REQ_SIZE);
  ptSetCfgReq->tData.ulVersion = DNS_CONFIGURATION_V1;

  /* Set the slave related parameters */
  memset(ptCfg,0x00,sizeof(DNS_CONFIGURATION_V1_T));
  ptCfg->ulSystemFlags   = 0;
  ptCfg->tData.ulWdgTime = 0;

  ptCfg->ulNodeId        = 11;
  ptCfg->ulBaudrate      = DNS_BAUDRATE_125kB;

  ptCfg->ulConfigFlags   = 0;
  ptCfg->ulObjectFlags   = 0;

  ptCfg->usVendorId      = 283; /* My VendorId    */
  ptCfg->usDeviceType    = 12;  /* My Device type */
  ptCfg->usProductCode   = 1;   /* My Product code */
  ptCfg->bMajorRev       = 1;   /* My Product major version */
  ptCfg->bMinorRev       = 1;   /* My Product minor version */
  ptCfg->bProductNameLen = 15;  /* Length of product name */
  memcpy(&ptCfg->abProductName[0],"My Product Name",15);

  ptCfg->ulProduceAsInstance = 101; /* My producing assembly instance */
  ptCfg->ulProduceAsSize = 8;       /* My producing assembly size */

  ptCfg->ulConsumeAsInstance = 100; /* My consuming assembly instance */
  ptCfg->ulConsumeAsSize = 8;       /* My consuming assembly size */

  return;
}
```

4.2.2.2 Set Configuration confirmation

Packet description

Variable	Type	Value / Range	Description
ulLen	UINT32	0	DNS_SET_CONFIGURATION_CNF_SIZE
ulSta	UINT32		ulSta = 0, initialization OK ulSta != 0, initialization failed, see section *Status and error codes* on page 114.
ulCmd	UINT32	0xB101	DNS_CMD_SET_CONFIGURATION_CNF

Table 63: DNS_CMD_SET_CONFIGURATION_CNF – Set Configuration confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_PACKET_SET_CONFIGURATION_CNF_Ttag
{
  HIL_PACKET_HEADER_T   tHead;
} DNS_PACKET_SET_CONFIGURATION_CNF_T;

#define DNS_SET_CONFIGURATION_CNF_SIZE   0
```

4.2.3 Create Assembly service

This service allows creating additional assembly objects. Creating additional assembly objects can be used when more than the two default input and output assembly shall be supported. Creating assembly objects is only allowed at configuration phase when the DeviceNet Slave stack is in stop state.

Up to 64 assembly objects are possible:

- The Set Configuration Service creates 2 assembly objects.
- This service can create up to 62 additional assembly objects.

Warning: This service does not cross check for overlapping data with other created assembly instances. It is in hand of the user for partitioning the DPM correctly. The parameter ulOffset is used to place or retrieve the assembly data from the right position within the DPM. If data of different assembly objects overlapping it may lead to process data inconsistency. On the other hand, it is may explicitly wished to overlap assembly data.

4.2.3.1 Create Assembly request

Packet description

Variable	**Type**	**Value / range**	**Description**
ulLen	uint32_t	16	Packet Data Length in bytes `DNS_CREATE_ASSEMBLY_REQ_SIZE`
ulCmd	uint32_t	0xB10A	DNS_CMD_CREATE_ASSEMBLY_REQ
Data			
ulInstance	uint32_t	1 … 255	Assembly instance to created
ulFlags	uint32_t	bit field	Assembly creation flags: D0: Assembly Type D8: Receive Idle option for details see: Table 65: Assembly configuration flags on page 76
ulSize	uint32_t	1 … 255	Size of the assembly to be create in bytes
ulOffset	uint32_t	0 … 254	Offset of the assembly data within the dual port memory **Note**: The sum of ulOffset + ulSize must not exceed 254 **Note**: In case the assembly option flag D8 is set for mapping the Run / Idle information into DPM, then the offset is start of the Run/Idle header. The start of the assembly data is offset + 4. See also section *Process data status* on page 111

Table 64: DNS_CMD_CREATE_ASSEMBLY_REQ – Create Assembly request

Packet structure reference

```
typedef struct DNS_CREATE_ASSEMBLY_Ttag {
    uint32_t ulInstance;
    uint32_t ulFlags;
    uint32_t ulSize;
    uint32_t ulOffset;
} DNS_CREATE_ASSEMBLY_T;

#define DNS_CREATE_ASSEMBLY_REQ_SIZE (sizeof(DNS_CREATE_ASSEMBLY_T))

typedef struct DNS_PACKET_CREATE_ASSEMBLY_REQ _Ttag {
  HIL_PACKET_HEADER_T        tHead;
  DNS_CREATE_ASSEMBLY_T      tData;
} DNS_PACKET_ CREATE_ASSEMBLY_REG_T;
```

Parameter: `ulFlags` (Assembly creation flags)

Bit	Description
D0	Assembly type **0:** If this flag is not set, the created assembly instance is a consuming assembly. The data received from the network are place into the input area of the DPM image. **1:** If this flag is set, the created assembly instance is a producing assembly. The data are transmitted to the network and taken from the output area of the DPM image.
D8	Receive Idle option: **0:** If this flag is zero the "receive / idle" information is not placed into the DPM (**default**) **1:** If this flag is set, "receive / idle" information will be placed additionally in front of the assembly data into the input image of the DPM for this assembly. This flag is only applicable for output assemblies. “Receive / Idle” is a 32-Bit value (4 bytes). This number of bytes must not be included into the assembly size configuration (`ulSize`).
D9	Sequence Counter option: **0:** If this flag is zero the sequence counter is not placed into the DPM (**default**) **1:** If this flag is set a "sequence counter" information is placed additionally in front of the assembly data into the input image of the DPM for this assembly. This flag is only applicable for output assemblies. “Sequence counter” is a 32-Bit value (4 bytes).. For details, refer to chapter Process data status on page 111. This number of bytes must not be included into assembly size configuration (`ulSize`).
Others	Reserved.

Table 65: Assembly configuration flags

4.2.3.2 Create Assembly confirmation

Packet structure reference

```
typedef struct DNS_PACKET_CREATE_ASSEMBLY_CNF _Ttag {
  HIL_PACKET_HEADER_T        tHead;
} DNS_PACKET_CREATE_ASSEMBLY_CNF_T;
```

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet Data Length in bytes
ulSta	uint32_t		See section *Status and error codes* on page 114.
ulCmd	uint32_t	0xB10B	DNS_CREATE_ASSEMBLY_CNF

Table 66: DNS_CMD_CREATE_ASSEMBLY_CNF – Register Class confirmation

4.3 Explicit Messaging services

4.3.1 General

Explicit messaging in terms of DeviceNet means the acyclic communication based on the CIP addressing model “Service / Class / Instance / Attribute”. Two pairs of packets of the DeviceNet Slave stack cover the explicit messaging services.

`DNS_CIP_SERVICE_REQ/CNF` – The host application sends this service to the local DeviceNet Slave stack.

`DNS_CIP_SERVICE_IND/RES` – The DeviceNet Slave stack sends this service to the host application. It is typically issued when a DeviceNet Master is requesting CIP specific data from the DeviceNet slave.

A service is specified by

- the service code to be performed
- the addressed scheme “Class + Instance + Attribute”
- the service specific data
- the General and Extended error decoding (GRC/ERC) in case of a failure

For both packets, the same CIP denotation is used.

4.3.2 CIP Service sent from application

This packet is used to perform a CIP service from the host application to any object of the **local** DeviceNet Slave stack. A list of pre-defined service codes by the CIP specification are listed in section *CIP defined Service Codes* on page 15.

Note: Not every service is available on every object. The list of supported services for each supported object by the DeviceNet Slave stack is described in section *Object classes* on page 18.

Depending on the service the data field `abData[]`of the packet is used to submit service related data (e.g. the attribute value in case of Set_Attribute_Single) or the data field in the confirmation packet contains the requested data (e.g. the attribute value in case of Get_Attribute_Single).

In case of successful execution, the variable `ulSta` of the confirmation packet will have the value SUCCESS_HIL_OK and the CIP specific error codes `ulGRC` and `ulERC` will be 0. In case of an error the variable `ulSta` of the confirmation packet will have value ERR_HIL_FAIL. The variables `ulGRC` and `ulERC` of the confirmation packet will contain CIP specific error codes.

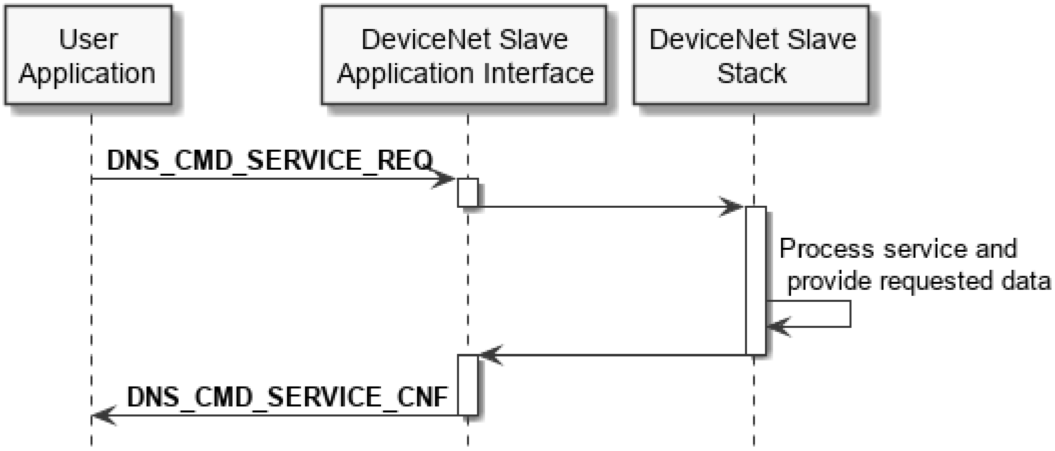
User Application
DeviceNet Slave Application Interface
DeviceNet Slave Stack
DNS_CMD_SERVICE_REQ
Process service and provide requested data
DNS_CMD_SERVICE_CNF

Figure 15: DNS_CMD_SERVICE_REQ/CNF sequence diagram

4.3.2.1 CIP Service request

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	20 + n	Packet Data Length in bytes `DNS_CIP_SERVICE_REQ_SIZE + n`
ulCmd	uint32_t	0xB102	DNS_CMD_CIP_SERVICE_REQ
tData			
ulService	uint32_t	Valid Service Code	Service code according CIP specification
ulClass	uint32_t	Valid Class ID	Class ID See supported objects listed in section *Object classes* on page 18. (also refer to [5], Chapter 5A, Table 5A-1.1)
ulInstance	uint32_t	Valid Instance ID	Instance ID of the class specified by usClassId See supported object instances listed in section *Object classes* on page 18.
ulAttribute	uint32_t	Valid Attribute ID	Attribute ID of an instance specified by usInstanceId. See supported object attributes listed in section *Object classes* on page 18.
ulMember	uint32_t	Valid Member ID	Members ID Set to 0 per default. The only service supporting the MemberID is “Get Member” for the assembly object attribute 2.
abData[]	uint8_t[]		Service data

Table 67: DNS_CMD_CIP_SERVICE_REQ – CIP Service request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_CIP_SERVICE_REQ_Ttag
{
  uint32_t    ulService;
  uint32_t    ulClass;
  uint32_t    ulInstance;
  uint32_t    ulAttribute;
  uint32_t    ulMember;
  uint8_t     abData[DNS_CIP_SERVICE_MAX_DATA_LEN];
} DNS_CIP_SERVICE_REQ_T;


typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_PACKET_CIP_SERVICE_REQ_Ttag
{
  HIL_PACKET_HEADER_T         tHead;
  DNS_CIP_SERVICE_REQ_T       tData;
}DNS_PACKET_CIP_SERVICE_REQ_T;
```

4.3.2.2 CIP Service confirmation

Packet description

Variable	Type	Value / Range	Description
ulLen	UINT32	28 + n	Packet Data Length in bytes
ulSta	UINT32		See section *Status and error codes* on page 114.
ulCmd	UINT32	0xB103	DNS_CMD_CIP_SERVICE_CNF
tData			
ulService	uint32_t	Valid Service Code	Service returned from request
ulClass	uint32_t	Valid Class ID	Class ID returned from request
ulInstance	uint32_t	Valid Instance ID	Instance ID returned from request
ulAttribute	uint32_t	Valid Attribute ID	Attribute ID returned from request
ulMember	uint32_t	Valid Member ID	Member ID returned from request
ulGRC	uint32_t		General error code, see Table 12 on page 16.
ulERC	uint32_t		Additional error code.
abData []	uint8_t[]		Service data

Table 68: DNS_CMD_CIP_SERVICE_CNF – CIP Service confirmation

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_CIP_SERVICE_CNF_Ttag
{
  uint32_t    ulService;
  uint32_t    ulClass;
  uint32_t    ulInstance;
  uint32_t    ulAttribute;
  uint32_t    ulMember;
  uint32_t    ulGRC;
  uint32_t    ulERC;
  uint8_t     abData[DNS_CIP_SERVICE_MAX_DATA_LEN];
} DNS_CIP_SERVICE_CNF_T;


typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_PACKET_CIP_SERVICE_CNF_Ttag
{
  HIL_PACKET_HEADER_T         tHead;
  DNS_CIP_SERVICE_CNF_T       tData;
}DNS_PACKET_CIP_SERVICE_CNF_T;

#define DNS_CIP_SERVICE_CNF_SIZE ((sizeof(DNS_CIP_SERVICE_CNF_T) - \
                                   DNS_CIP_SERVICE_MAX_DATA_LEN))
```

4.3.3 Register Class Service

The DeviceNet Slave stack has the option to forward explicit services like *Get_Attribute or Set_Attribute* or other services to the user application.

Therefore, the user must register the corresponding class within the stack to get these services. This must be done for each class the user wants to have the explicit service indications.

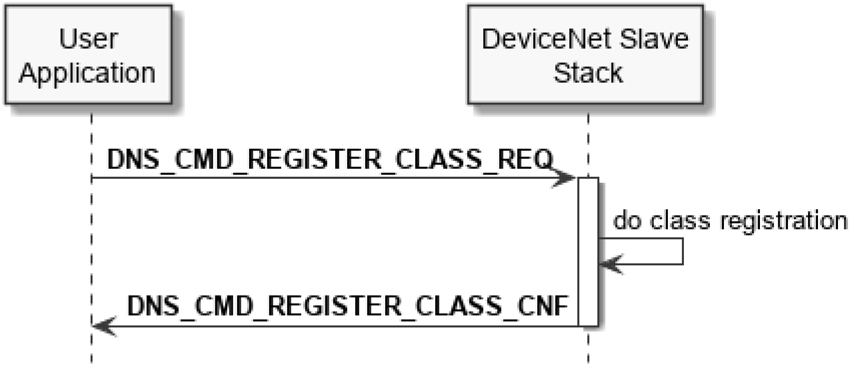

Figure 16: DNS_CMD_REGISTER_CLASS_REQ/CNF - sequence diagram

4.3.3.1 Register Class request

Packet description

Variable	Type	Value / range	Description
ulLen	uint32_t	8	Packet Data Length in bytes sizeof(DNS_REGISTER_CLASS_T)
ulCmd	uint32_t	0xB106	DNS_REGISTER_CLASS_REQ
Data			
ulClass	uint32_t	1, 3 … 42, 44 ... 255	Class ID according reference [5], Chapter 5A, Table 5A-1.1
ulServiceMask	uint32_t	0x00000000 … 0xFFFFFFF	Services registration to be forwarded to the host application 0: All services are forwarded to the application **(default)** n: Only selected services are forwarded to the application The application has to interpret this value as a bit field. Each bit represents a service, according Table 71 on page 84. **Note:** For registration of a user specific Class ID, which are not handled by the stack ‘ulServiceMask’, is not considered by the stack. In this case, the value of ‘ulServiceMask’ must be set to 0. For user specific objects all services are forwarded to the host,

Table 69: DNS_CMD_REGISTER_CLASS_REQ – Register Class request

Packet structure reference

```
typedef struct DNS_REGISTER_CLASS_Ttag {
  uint32_t                ulClass;
  uint32_t                ulServiceMask;
} DNS_REGISTER_CLASS_T;

#define DNS_REGISTER_CLASS_REQ_SIZE (sizeof(DNS_REGISTER_CLASS_T))

typedef struct DNS_PACKET_REGISTER_CLASS_REQ _Ttag {
  HIL_PACKET_HEADER_T        tHead;
  DNS_REGISTER_CLASS_T       tData;
} DNS_PACKET_REGISTER_CLASS_REG_T;
```

Parameter `ulClass`

The table below shows, which object classes the application, can register within the DeviceNet Slave stack.

Class	Description	Registration allowed
0x01	Identity	⚠
0x02	Message Router	✖
0x03	DeviceNet	⚠
0x04	Assembly Object	✖
0x05	Connection Object	⚠
0x06	Connection Manager Object	✖
0x07 – 0x2A	CIP pre-defined Objects	✓
0x2B	Acknowledge Handler	✖
0x2C – 0xFF	CIP pre-defined Objects	✓
0x402	IO Mapping Object	✖
0x404	Module and Network Status Object	✖

Table 70: DNS_CMD_REGISTER_CLASS_REQ – Register Class limitations

✓ - The application can register this object without any limitations, all requests from the network will be forwarded to host application.

⚠ - The application can register this object only with individual limitations, because the stack handles these objects by itself.

- A service is **NOT** forwarded, if the service is directed to an attribute that is handled by the DeviceNet Slave stack itself according section *Object classes* on page 18. This includes all class attributes and instance attributes.
- A service is **NOT** forwarded, if the service was excluded from registration by the parameter `ulServiceMask`.
- A service **IS** forwarded, if the service is directed to an attribute or class instance that is not handled by the DeviceNet Slave stack itself.

✖ - The object cannot be registered.

Parameter `ulServiceMask`

This parameter is a filter to allow forwarding of selective service codes to the host application. The application should interpret this parameter as a bit field. If set to 0, all services are forwarded to the host. As soon as one or more bits are set, only the selected services are sent to the host.

Bit position	Service code	Service name
0	0x00	Reserved
1	0x01	Get_Attributes_All
2	0x02	Set_Attributes_All
3	0x03	Get_Attribute_List
4	0x04	Set_Attribute_List
5	0x05	Reset
6	0x06	Start
7	0x07	Stop
8	0x08	Create
9	0x09	Delete
10	0x0A	Multiple_Service_Packet
11, 12	0x0B, 0x0C	Reserved for future use
13	0x0D	Apply_Attributes
14	0x0E	Get_Attribute_Single
15	0x0F	Reserved for future use
16	0x10	Set_Attribute_Single
17	0x11	Find_Next_Object_Instance
18, 19	0x12, 0x13	Reserved for future use
20	0x14	Error Response
21	0x15	Restore
22	0x16	Save
23	0x17	No Operation (NOP)
24	0x18	Get_Member
25	0x19	Set_Member
26	0x1A	Insert_Member
27	0x1B	Remove_Member
28	0x1C	GroupSync
29-31	0x1D–0x1F	Reserved for additional Common Services

Table 71: Service Codes depending to the bit position

4.3.3.2 Register Class confirmation

Packet structure reference

```
typedef struct DNS_REGISTER_CLASS_Ttag
{
  uint32_t ulClass;
  uint32_t ulServiceMask;
} DNS_REGISTER_CLASS_T;

typedef struct DNS_FAL_PACKET_REGISTER_CLASS_CNF_Ttag
{
  HIL_PACKET_HEADER_T     tHead;
  DNS_REGISTER_CLASS_T    tData;
} DNS_PACKET_REGISTER_CLASS_CNF_T;
```

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	8	Packet Data Length in bytes sizeof(DNS_REGISTER_CLASS_T)
ulSta	uint32_t		See section *Status and error codes* on page 114.
ulCmd	uint32_t	0xB107	DNS_CMD_REGISTER_CLASS_CNF
tData			
ulClass	uint32_t	1, 3 … 42, 44 ... 255	Class ID value returned from the request
ulServiceTyp	uint32_t	0x00000000 .... 0xFFFFFFFF	Service registration value returned from the request

Table 72: DNS_CMD_REGISTER_CLASS_CNF – Register Class confirmation

4.3.4 Unregister Class service

This command will unregister a previously registered class. If unregistering successfully, the service to the class will (no longer) be passed through to the host application.

4.3.4.1 Unregister Class request

Packet description

Variable	Type	Value / Range	Description
ulLen	unit32	8	Packet Data Length in bytes sizeof(DNS_UNREGISTER_CLASS_T)
ulCmd	uint32	0xB108	DNS_CMD_UNREGISTER_CLASS_REQ
tData			
ulClass	uint32_t	1, 3 … 42, 44 ... 255	Class ID (according reference [5], Chapter 5A, Table 5A-1.1)
ulServiceMask	uint32	0	Reserved unused, set to 0

Table 73: DNS_CMD_UNREGISTER_CLASS_REQ – Unregister Class request

Packet structure reference

```
typedef struct DNS_UNREGISTER_CLASS_Ttag
{
  uint32_t ulClass;
  uint32_t ulAccessTyp;
} DNS_UNREGISTER_CLASS_T;

#define DNS_UNREGISTER_CLASS_REQ_SIZE (sizeof(DNS_UNREGISTER_CLASS_T))

typedef struct DNS_PACKET_UNREGISTER_CLASS_REQ _Ttag
{
  HIL_PACKET_HEADER_T    tHead;
  DNS_UNREGISTER_CLASS_T tData;
} DNS_PACKET_UNREGISTER_CLASS_REG_T;
```

4.3.4.2 Unregister Class confirmation

This command confirms the unregistration from the requested class.

Note: The unregistration confirms successfully regardless, whether the class has been registered before, or not.

Packet description

Variable	Type	Value / Range	Description
ulLen	UINT32	8	Packet Data Length in bytes sizeof(DNS_UNREGISTER_CLASS_T)
ulSta	uint32_t		See section *Status and error codes* on page 114.
ulCmd	UINT32	0xB109	DNS_CMD_UNREGISTER_CLASS_CNF
tData			
ulClass	UINT32		Class ID
ulAccessTyp	UINT32	0	Reserved unused, set to 0

Table 74: DNS_CMD_UNREGISTER_CLASS_CNF – Unregister Class confirmation

Packet structure reference

```
Packet Structure Reference
/* DNS_FAL_CMD_UNREGISTER_CLASS_REQ Structure */
typedef struct DNS_FAL_UNREGISTER_CLASS_Ttag
{
  TLR_UINT32 ulClass;
  TLR_UINT32 ulAccessTyp;
}
DNS_FAL_UNREGISTER_CLASS_T;

typedef struct DNS_FAL_PACKET_UNREGISTER_CLASS_CNF_Ttag
{
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_UNREGISTER_CLASS_T tData;
}
DNS_FAL_PACKET_UNREGISTER_CLASS_CNF_T
```

4.3.5 CIP Service sent by a master

This packet indicates that the remote DeviceNet Master has requested a service from the Slave. The user receives the service indication only for those classes that have been registered to the DeviceNet Slave stack.

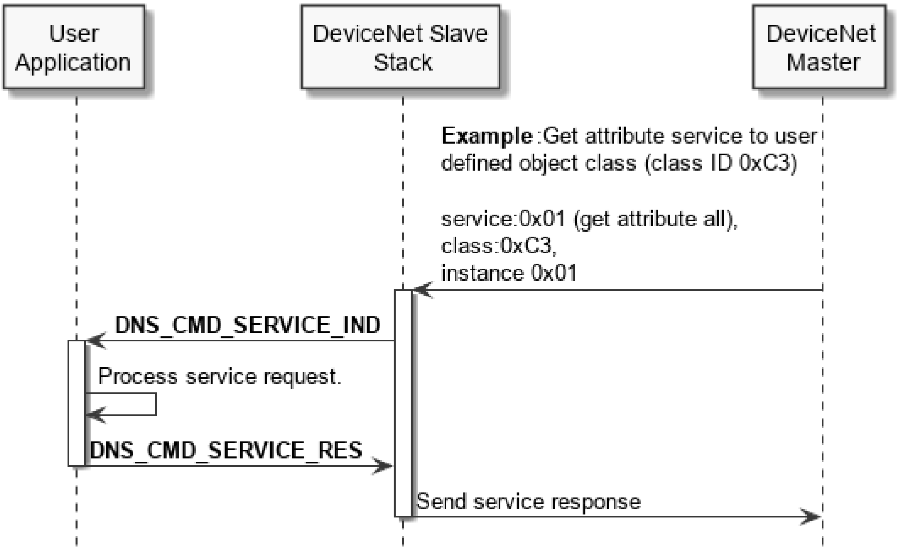

Figure 17: DNS_CMD_SERVICE_IND/RES sequence diagram

This packet is typically used to implement a user- or profile-specific object within the application context.

4.3.5.1 CIP Service indication

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	20 + n	Packet Data Length in bytes (DNS_CIP_SERVICE_IND_SIZE+n)
ulCmd	uint32_t	0xB104	DNS_CMD_CMD_SERVICE_IND
tData			
ulService	uint32_t	Service Code	Service Code; see *CIP defined Service Codes* on page 15.
ulClass	uint32_t	Class ID	Class ID
ulInstance	uint32_t	Instance ID	Class Instance ID
ulAttribute	uint32_t	Attribute ID	Attribute ID
ulMember	uint32_t	Member ID	Member ID
abData[]	uint8_t[]		Service data (n bytes - depending on service)

Table 75: DNS_CMD_SERVICE_IND – CIP Service indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_CIP_SERVICE_REQ_Ttag
{
  uint32_t    ulService;
  uint32_t    ulClass;
  uint32_t    ulInstance;
  uint32_t    ulAttribute;
  uint32_t    ulMember;
  uint8_t     abData[DNS_CIP_SERVICE_MAX_DATA_LEN];
} DNS_CIP_SERVICE_REQ_T;

#define DNS_CIP_SERVICE_IND_T DNS_CIP_SERVICE_REQ_T

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_PACKET_CIP_SERVICE_IND_Ttag
{
  HIL_PACKET_HEADER_T         tHead;
  DNS_CIP_SERVICE_IND_T       tData;
}DNS_PACKET_CIP_SERVICE_IND_T;

#define DNS_CIP_SERVICE_IND_SIZE (sizeof(DNS_CIP_SERVICE_IND_T) - \
                                         DNS_CIP_SERVICE_MAX_DATA_LEN)
```

4.3.5.2 CIP Service response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_CIP_SERVICE_CNF_Ttag
{
  uint32_t    ulService;
  uint32_t    ulClass;
  uint32_t    ulInstance;
  uint32_t    ulAttribute;
  uint32_t    ulMember;
  uint32_t    ulGRC;
  uint32_t    ulERC;
  uint8_t     abData[DNS_CIP_SERVICE_MAX_DATA_LEN];
} DNS_CIP_SERVICE_CNF_T;

#define DNS_CIP_SERVICE_RES_T DNS_CIP_SERVICE_RES_T

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_PACKET_CIP_SERVICE_IND_Ttag
{
  HIL_PACKET_HEADER_T         tHead;
  DNS_CIP_SERVICE_IND_T       tData;
}DNS_PACKET_CIP_SERVICE_IND_T;

#define DNS_CIP_SERVICE_CNF_SIZE (sizeof(DNS_CIP_SERVICE_IND_T) - \
                                         DNS_CIP_SERVICE_MAX_DATA_LEN)
```

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	28 + n	Packet Data Length in bytes (DNS_CIP_SERVICE_RES_SIZE+n) n = 0 in case of error or no response data to the service n != 0 in case of service response data, number of bytes in abData[]
ulSta	uint32_t		Set to SUCCESS_HIL_OK in case of service execution success Set to ERR_HIL_FAIL in case of service execution failed. In this case, additionally ulGRC and ulERC must be set. Other return values are not allowed.
ulCmd	uint32_t	0xB105	DNS_CMD_SERVICE_RES
tData			
ulService	uint32_t	Service Code	Service Code
ulClass	uint32_t	Class ID	return value from request
ulInstance	uint32_t	Instance ID	return value from request
ulAttribute	uint32_t	Attribute ID	return value from request
ulMember	uint32_t	Member ID	return value from request
ulGRC	uint32_t		Generic Error Code Set to 0 in case of service execution success. Set to a valid CIP defined error code in case service execution failed; see section *CIP defined General Status Codes* on page 16.
ulERC	uint32_t		Extended Error Code Set to 0 in case of no additional error code. (default) For additional error code definition and rules, see section *CIP defined Extended Status Codes* on page 17.
abData[]	uint8_t[]		Service data (n bytes - depending on service)

Table 76: DNS_CMD_SERVICE_RES – CIP Service response

4.3.6 Reset service

The stack sends the reset indication to the host whenever a reset of the device is required. It provides a notification about a reset that has been requested. The stack will not perform the reset procedure on its own. The host itself must perform the reset after the reset indication has been received.

When the host receives the reset indication, it must respond to the received packet. The host can either accept the reset or reject it with a corresponding error code. The reset indication can be caused either by a reset service from a remote device to the identity object instance 1 or by a stack internal requirement. In case of a reset requested via network, the stack will return the response to the sender.

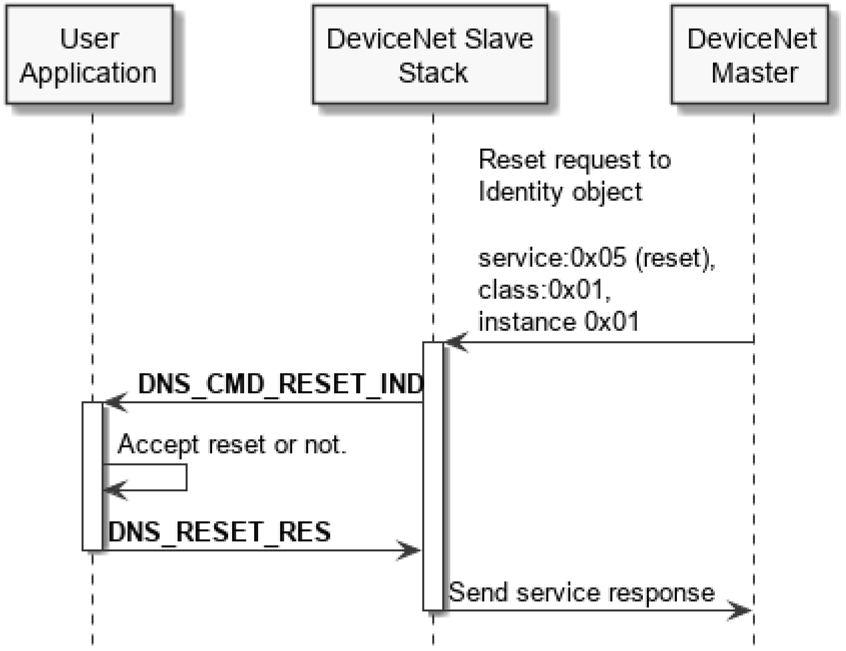

Figure 18: DNS_CMD_RESET_IND/RES sequence diagram

Note: The stack will send a reset indication to the application after the host has used the command `HIL_REGISTER_APP_REQ.` In this case, the host is responsible to perform the reset. Performing a reset by the host means, the application has to start over a complete configuration procedure like after power-on.

Note: A reset request to the identity an object via network to an instance different to 1 or a reset service to other objects is forwarded as CIP_SERVICE_IND.

4.3.6.1 Reset indication

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	8	sizeof(DNS_RESET_IND_T)
ulCmd	uint32_t	0xB10C	DNS_CMD_RESET_IND
tData			
ulReason	uint32_t	0;1;2;3	Reset reason; see Table 78 on page 93.
ulType	uint32_t	0 … 255	Reset type; see Table 79 on page 94.

Table 77: DNS_CMD_RESET_IND – Reset indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_RESET_IND_Ttag
{
 uint32_t ulReason; /*!< Reset reason */
 uint32_t ulType;   /*!< Reset type */
} DNS_RESET_IND_T;


typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_PACKET_RESET_IND_Ttag
{
 HIL_PACKET_HEADER_T tHead;
 DNS_RESET_IND_T     tData;
}DNS_PACKET_RESET_IND_T;

#define DNS_RESET_IND_SIZE (sizeof(DNS_RESET_IND_T)
```

Parameter ulReason

This parameter provides an information about the reason of the reset.

Value	Define / description
0	DNS_RESET_REASON_ID_OBJECT_NET_RESET The reset is requested to instance 1 of the identity object. It could have been received via network from a master or commissioning tool or by the host itself via CIP service request. Additionally, the reset type is indicated. **Note**: If the host itself triggers the reset indication via CIP service request, the reset type is coded into the first byte in the service data field abData[] of the CIP service request.
1	DNS_RESET_REASON_DN_OBJECT_MACID_SET The reset is required because a master or commissioning tool has set the MAC ID attribute of the DeviceNet object. This requires a reset to go online with the new MAC ID.
2	DNS_RESET_REASON_NP_RESUME_SWITCH_CHANGE This reason can only appear when the switch support for MAC ID and/or Baud rate is enabled. The reset is required because one or both of the switch values have been changed at runtime and the 24V network power was released and resumed.
3	DNS_RESET_REASON_NETWOK_POWER_CHANGE This type of reset indication is for informational purposes only. host application does not require any explicit reset activity. The host application can use this type of indication to monitor the 24V network power. The reset indication will be sent when the 24V network power is lost or resumed. **Note**: In order to indicate this reason to the host, it must be **explicitly activated** by setting the flag DNS_CFG_FLAG_INDICATION_NETWORK_POWER in the set configuration packet. The stack has a latch mechanism for the 24V NP indication. It will not send a new indication until the host has sent the response from a previous one. The stack does not send an initial state indication of the 24V trunk power at startup. The application has to start with the assumption that the 24V NP is present. Only when the 24V trunk power is mission at startup a corresponding indication will be send.

Table 78: DNS_RESET_IND – reset reason

Parameter ulType

This parameter provides additional information to the reset reason. The table below lists possible values of ‘ulType’ depending on the reset reason. In case of reset reason 0 (when the reset is received from the network and directed to the identity object), then the parameter ‘ulType’ contains additional information to the reset reason that has to be performed by the host. The host has to perform the reset or can reject with an appropriate error code, e.g. if the type is not supported or invalid or the device is in a state not to allow the reset.

Value of ulType	**Description / define**
Reset Reason: ‘DNS_RESET_REASON_ID_OBJECT_NET_RESET’	
0	Power Cycle Emulate as closely as possible cycling power on the item the Identity Object represents.
1	Factory default Return as closely as possible to the factory default configuration, and then emulate cycling power as closely as possible.
2	Return to Factory Defaults except Communications Parameters **Note:** This type of reset is forwarded to the host application only. The stack does not support or handle this type of reset. If the stack receives this type of reset, the application has to reset its own parameter to factory default values. The communication parameters like node id or baud rate shall not be reset.
3 – 99	Reserved reset types by CIP specification, not forwarded to host application. or
100 - 199	By CIP specification, “User specific” reset types. The master or commissioning tool may send other types of reset, which are user specific. In this case, the host has to perform a user-specific reset procedure or reject with appropriate error code. **Note:** User-specific reset types are forwarded to the host application only. The stack does not handle or support this type of reset.
200 - 255	Reserved reset types by CIP specification, not forwarded to host application.
Reset Reason: ‘DNS_RESET_REASON_DN_OBJECT_MACID_SET’ **Reset Reason: ‘DNS_RESET_REASON_NP_RESUME_SWITCH_CHANGE’**	
0	Power Cycle Emulate as closely as possible cycling power on the item the Identity Object represents.
1..255	Reserved, not used for this reset reason
Reset Reason: ‘DNS_RESET_REASON_NETWOK_POWER_CHANGE’	
3	DNS_RESET_TYPE_NP_MISSING This value indicates that the 24V network power is missing. There is no explicit reset activity required by the host application. The host application can use this type of indication to monitor the 24V network power. It will be sent when the 24V network power is lost.
4	DNS_RESET_TYPE_NP_PRESENT This value indicates that the 24V network is present or is resumed.
0,1,2,5..255	This value indicates that the 24V network power is missing. There is no explicit reset activity required by the host application. The host application can use this type of indication to monitor the 24V network power. It will be sent when the 24V network power is returned back.

Table 79: DNS_RESET_IND – Reset types depending on Reset Reason

4.3.6.2 Reset response

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	16	Packet Data Length in bytes sizeof(DNS_RESET_RES_T)
ulSta	uint32_t	0 or 1	Set to SUCCESS_HIL_OK in case of accepting the service Set to ERR_HIL_FAIL and additional error code (ulGRC) when the reset is not accepted. See section *Status and error codes* on page 114.
ulCmd	uint32_t	0xB10D	DNS_CMD_RESET_RES
tData			
ulReason	uint32_t	0; 1; 2	Reset reason, return value from indication
ulType	uint32_t	0 … 255	Reset type, return value from indication
ulGRC	uint32_t	0 or n	General error code Set to default value 0 in case of success, or set an error code when the reset is not accepted. For appropriate value see CIP defined General Status Codes on page 16
ulERC	uint32_t	0 or n	Additional error code The default value is 0. For more information, see CIP defined Extended Status Codes on page 17

Table 80: DNS_CMD_RESET_RES – Reset response

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_IF_CIP_SERVICE_CNF_Ttag
{
 uint32_t ulReason; /*!< Reset reason */
 uint32_t ulType;   /*!< Reset type */
 uint32_t ulGRC;    /*!< Generic Error Code */
 uint32_t ulERC;    /*!< Extended Error Code */

} DNS_RESET_RES_T;


typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_PACKET_RESET_RES_Ttag
{
 HIL_PACKET_HEADER_T tHead;
 DNS_RESET_RES_T     tData;
}DNS_PACKET_RESET_RES_T;

#define DNS_RESET_RES_SIZE (sizeof(DNS_RESET_RES_T)
```

4.4 Diagnostic service

4.4.1 Diag service

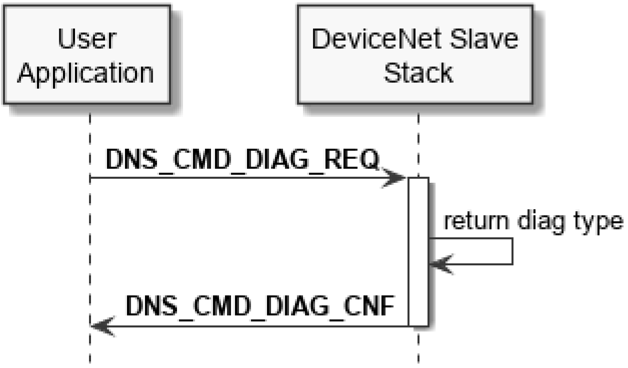

Figure 19: DNS_CMD_DIAG_REQ/CNF - Sequence diagram

The application can use the diagnostic service to retrieve different diagnostic information from the DeviceNet slave stack.

Four types of diagnostic information can be read from stack:

- DNS_DIAG_TYPE_COMMON

  This diagnostic type is a collection of the most useful information about the status of the DeviceNet slave stack.
- DNS_DIAG_TYPE_COMMAND

  This diagnostic structure is a counter diagnostic about services exchanged between stack and host application.
- DNS_DIAG_TYPE_CAN

  This diagnostic structure contains detailed information about the CAN Data Link layer.
- DNS_DIAG_TYPE_RESOURCES

  This diagnostic contains information about some resource allocation of the stack.

4.4.1.1 Diag request

Packet description

Variable	Type	Value / range	Description
ulLen	uint32_t	4	sizeof(DNS_DIAG_REQ_T)
ulCmd	uint32_t	0xB10E	DNS_CMD_DIAG_REQ
Data			
ulDiagType	uint32_t	0,1,2,3	Diagnostic type to request 0: collection of common useful diagnostic information 1: service counter diagnostic between stack and host application 2: CAN network related diagnostic 3: stack resource allocation information

Table 81: DNS_CMD_DIAG_REQ – Diagnostic request

Packet structure reference

```
/*! \brief Diagtypes. */
#define DNS_DIAG_TYPE_COMMON      0
#define DNS_DIAG_TYPE_COMMAND     1
#define DNS_DIAG_TYPE_CAN         2
#define DNS_DIAG_TYPE_RESOURCES   3

/*! \brief Diag Request data. */
typedef struct DNS_DIAG_REQ_Ttag
{
  uint32_t ulDiagType;
}DNS_DIAG_REQ_T;

/*! \brief Diag Request packet. */
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_PACKET_DIAG_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;   /*!< Packet header. */
  DNS_DIAG_REQ_T      tData;   /*!< Diag Request Data. */
}DNS_PACKET_DIAG_REQ_T;
```

4.4.1.2 Diag confirmation

Packet description

Variable	Type	Value / Range	Description

ulLen	uint32_t	4 + n	Packet Data Length in bytes
			For ulDiagType = 0 in request packet DNS_DIAG_REQ:
			`sizeof(uint32_t) + (sizeof(DNS_DIAG_COMMON_T))`
			For ulDiagType = 1 in request packet DNS_DIAG_REQ:
			`sizeof(uint32_t) + (sizeof(DNS_DIAG_COMMAND_T))`
			For ulDiagType = 2 in request packet DNS_DIAG_REQ:
			`sizeof(uint32_t) + (sizeof(DNS_DIAG_CAN_T))`
			For ulDiagType = 3 in request packet DNS_DIAG_REQ:
			`sizeof(uint32_t) + (sizeof(DNS_DIAG_RESOURCE_T))`

ulSta	uint32_t		See section *Status and error codes* on page 114.
ulCmd	uint32_t	0xB10F	DNS_DIAG_CNF
tData			
ulDiagType	uint32_t	0,1,2,3	Diagnostic type returned from request

uDiag	union		Union of four different diagnostic structures:
			For ulDiagType = 0 in request packet DNS_DIAG_REQ:
			`DNS_DIAG_COMMON_T     tCom;`
			For ulDiagType = 1 in request packet DNS_DIAG_REQ:
			`DNS_DIAG_COMMAND_T   tCmd;`
			For ulDiagType = 2 in request packet DNS_DIAG_REQ:
			`DNS_DIAG_CAN_T              tCan;`
			For ulDiagType = 3 in request packet DNS_DIAG_REQ:
			`DNS_DIAG_RESOURCE_T tRsc;`

Table 82: DNS_DIAG_CNF – Diag confirmation

Packet structure reference

```
/*! \brief Diag Confirmation Data union. */
typedef __HIL_PACKED_PRE union __HIL_PACKED_POST DNS_DIAG_Utag
{
  DNS_DIAG_COMMON_T    tCom;
  DNS_DIAG_COMMAND_T   tCmd;
  DNS_DIAG_CAN_T       tCan;
  DNS_DIAG_RESOURCE_T  tRsc;

}DNS_DIAG_U;

/*! \brief Diag Confirmation Data. */
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_DIAG_CNF_Ttag
{
  uint32_t   ulDiagType;  /*!< Diagnostic type */
  DNS_DIAG_U uDiag;       /*!< Union of different diagnostic types  */
}DNS_DIAG_CNF_T;

/*! \brief Diag Confirmation packet. */
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST DNS_PACKET_DIAG_CNF_Ttag
{
 HIL_PACKET_HEADER_T tHead;   /*!< Packet header. */
 DNS_DIAG_CNF_T      tData;   /*!< Diag Confirmation Data. */
}DNS_PACKET_DIAG_CNF_T;

/*! Size of Diag Confirmation. */
#define DNS_DIAG_CNF_SIZE_MIN          (sizeof(uint32_t))

#define DNS_DIAG_CNF_SIZE_COMMON       (sizeof(uint32_t)) + (sizeof(DNS_DIAG_COMMON_T))
#define DNS_DIAG_CNF_SIZE_COMMAND      (sizeof(uint32_t)) + (sizeof(DNS_DIAG_COMMAND_T))
#define DNS_DIAG_CNF_SIZE_CAN          (sizeof(uint32_t)) + (sizeof(DNS_DIAG_CAN_T))
#define DNS_DIAG_CNF_SIZE_RESOURCE     (sizeof(uint32_t)) + (sizeof(DNS_DIAG_RESOURCE_T))
```

Diagnostic Type0 – Common stack information

```
typedef struct DNS_PROTOCOL_INFO_Ttag{
  uint8_t bCurrentNodeId;
  uint8_t bCurrentBaudrate;
  uint8_t bNetwokPower;
  uint8_t bNetworkAccessState;
  uint8_t bModuleStatus;
  uint8_t bNetworkStatus;
  uint16_t usReserved;

}DNS_PROTOCOL_INFO_T;

typedef struct DNS_SWITCH_INFO_Ttag{
  uint8_t bNodeSwitchEnable;
  uint8_t bNodeSwitchValue;
  uint8_t bBaudrateSwitchEnable;
  uint8_t bBaudrateSwitchValue;

}DNS_SWITCH_INFO_T;

typedef struct DNS_CAN_INFO_Ttag{
  uint32_t ulCanState;
  uint32_t ulCanRxCnt;
  uint32_t ulCanTxCnt;
  uint32_t ulCanRxOverRunCnt;
  uint32_t ulCanTxOverRunCnt;
  uint32_t ulCanTxAbortCnt;

  uint32_t ulCanErrWarningCnt
  uint32_t ulCanErrPassiveCnt;
  uint32_t ulCanErrBusOffCnt;

}DNS_CAN_INFO_T;

typedef struct DNS_CONN_INFO_Ttag{
  uint8_t  bAllocChoice;
  uint8_t  bMasterMacId;
  uint16_t usReserved;
}DNS_CONN_INFO_T;

typedef struct DNS_CONN_INST_INFO_Ttag{
  uint16_t usProduceSize;
  uint16_t usProduceAsmInst;
  uint16_t usConsumeSize;
  uint16_t usConsumeAsmInst;

}DNS_CONN_INST_INFO_T;

typedef struct DNS_DIAG_COMMON_Ttag
{
  uint32_t ulStatusFlags;

  DNS_PROTOCOL_INFO_T  tProtocolInfo;
  DNS_SWITCH_INFO_T    tSwitchInfo;
  DNS_CAN_INFO_T       tCanInfo;
  DNS_CONN_INFO_T      tConnInfo;

  #define DNS_CONN_INFO_INST_POLL    0
  #define DNS_CONN_INFO_INST_STROBE  1
  #define DNS_CONN_INFO_INST_COS_CYC 2

  DNS_CONN_INST_INFO_T tConnInfoInst[3];
}DNS_DIAG_COMMON_T;
```

Variable	Type	Value	Description
ulStatusFlags	uint32_t	Bit field	Collective status information of the stack as a bit field MSK_DNS_ … D0: STATUS_FLAG_CONFIG_VALID The stack has a valid configuration. D1: STATUS_FLAG_BUS_START The stack is allowed to release the network communication. (refer to section *Start / Stop Communication* on page 107) D2: STATUS_FLAG_24V_NETWORK_POWER If ‘0’ no network power is present. If ‘1’ 24 V network power is present. D3: STATUS_FLAG_NETWORK_STATE_ONLINE If ‘0’ the stack is not present to the network. If ‘1’ the stack online to the network and has passed the duplicate mac id check procedure. D16: STATUS_FLAG_ERR_DUP_MAC The stack detected a slave with the same mac id and went to the duplicate mac fault. D17: STATUS_FLAG_ERR_CAN_BUS_OFF The stack detected a CAN BUS OFF event. To recover from this state a manual intervention is required. (Power cycle the device or resume the 24 V network power). D18: STATUS_FLAG_ERR_MAJOR_UNRECOVERABLE A major unrecoverable fault has been detected. D4 - D15; D19 - D31 reserved
DNS_PROTOCOL_INFO_T tProtocolInfo			
bCurrentNodeId	uint8_t	0 ... 63	Current node id that the device is online to the network
bCurrentBaudrate	uint8_t	0, 1, 2	Current baud rate that the device is online to the network
bNetwokPower	uint8_t	0, 1	Present of 24V network power 0 – no network power 1 – 24V network power is present
bNetworkAccessState	uint8_t	1 … 4	This value represents the state machine for network access according to the DeviceNet norm. 0 – the device is marked as non-existent 1 – the device is trying to send the duplicate MAC ID request 2 – the device is waiting for a duplicate MAC ID response 3 – the device is online to the network 4 – the device is in the communication fault state
bModuleStatus	uint8_t	0 ... 5	This value contains the module status described in Table 40 (page 34)
bNetworkStatus	uint8_t	0 … 5	This value contains the network status as described in Table 41 (page 35)
usReserved	uint16_t	0	reserved
DNS_SWITCH_INFO_T tSwitchInfo			
bNodeSwitchEnable	uint8_t	0, 1	If ‘1’ the node id is enabled to be configured by a rotary switch
bNodeSwitchValue	uint8_t	0 … 99	Current position of the node id rotary switch
bBaudrateSwitchEnable	uint8_t	0, 1	If ‘1’ the baud rate is enabled to be configured by a rotary switch

Variable	Type	Value	Description
bBaudrateSwitchValue	uint8_t	0 … 99	Current position of the baud rate rotary switch
DNS_CAN_INFO_T tCanInfo			
ulCanState	uint32_t	0, 1, 2, 3	0: Active state 1: Error warning state 2: Error passive state 3: Bus Off state
ulCanRxCnt	uint32_t	n	Number of CAN frames sent from the stack to the CAN controller
ulCanTxCnt	uint32_t	n	Number of CAN frames received by the stack from the CAN controller
ulCanRxOverRunCnt	uint32_t	n	Number of CAN Frame Rx FIFO overrun event.
ulCanTxOverRunCnt	uint32_t	n	Number of CAN Frame Tx FIFO overrun event.
ulCanTxAbortCnt	uint32_t	n	Number of frames aborted from transmission.
ulCanErrWarningCnt	uint32_t	n	Number, how often then CAN controller went to error warning state.
ulCanErrPassiveCnt	uint32_t	n	Number, how often then CAN controller went to error passive state.
ulCanErrBusOffCnt	uint32_t	n	Number, how often then CAN controller went to bus off state.
DNS_CONN_INFO_T tConnInfo			
bAllocChoice	uint8_t	Bit field	Information bit field about the allocated connection D0: Explicit D1: Poll D2: Strobe D3: Multicast poll (not supported) D4: Change of State D5: Cyclic D6: Acknowledge suppress D7: reserved
bMasterMacId	uint8_t	0 … 63, 255	Mac id of the DeviceNet master that has allocated a connection with the slave. 255 means the slave is not allocated by a master.
usReserved	uint16_t	0	Reserved
DNS_CONN_INST_INFO_T atConnInfoInst[0 .. 2] (0 = POLL; 1 = STROBE ; 2 = COS/CYC)			
usProduceSize	uint16_t	0 … 255	Number of bytes produces (send) within this connection.
usProduceAsmInst	uint16_t	n	Assembly object instance assigned for producing to this connect.
usConsumeSize	uint16_t	0 … 255	Number of bytes consumed (received) within this connection.
usConsumeAsmInst	uint16_t	n	Assembly object instance assigned for producing to this connect.

Table 83: Diagnostic Type1 – Common stack information

Diagnostic Type2 – Command counters

```
typedef struct DNS_DIAG_COMMAND_Ttag
{
  uint32_t ulCipReq;
  uint32_t ulCipCnfPos;
  uint32_t ulCipCnfNeg;
  uint32_t ulCipInd;
  uint32_t ulCipResPos;
  uint32_t ulCipResNeg;
  uint32_t ulCreateAsmReq;
  uint32_t ulCreateAsmCnfPos;
  uint32_t ulCreateAsmCnfNeg;
  uint32_t ulRegClassReq;
  uint32_t ulRegClassCnfPos;
  uint32_t ulRegClassCnfNeg;
  uint32_t ulResetInd;
  uint32_t ulResetResPos;
  uint32_t ulResetResNeg;

}DNS_DIAG_COMMAND_T;
```

Variable	Type	Value	Description
ulCipReq	uint32_t	0 … n	Number of CIP Request sent from host to stack
ulCipCnfPos	uint32_t	0 … n	Positive CIP confirmation from stack to host
ulCipCnfNeg	uint32_t	0 … n	Negative CIP confirmation from stack to host
ulCipInd	uint32_t	0 … n	Number of CIP indications sent from stack to host
ulCipResPos	uint32_t	0 … n	Positive CIP response from host to stack
ulCipResNeg	uint32_t	0 … n	Negative CIP response from host to stack
ulCreateAsmReq	uint32_t	0 … n	Number of ‘Create Assembly’ requests sent from host to stack
ulCreateAsmCnfPos	uint32_t	0 … n	Positive ‘Create Assembly’ confirmation from stack to host
ulCreateAsmCnfNeg	uint32_t	0 … n	Negative ‘Create Assembly’ confirmation from stack to host
ulRegClassReq	uint32_t	0 … n	Number of ‘Register Class’ requests sent from host to stack
ulRegClassCnfPos	uint32_t	0 … n	Positive ‘Register Class’ confirmation from stack to host
ulRegClassCnfNeg	uint32_t	0 … n	Negative ‘Register Class’ confirmation from stack to host
ulResetInd	uint32_t	0 … n	Number of ‘Reset’ indications sent from stack to host
ulResetResPos	uint32_t	0 … n	Positive ‘Reset’ response from host to stack
ulResetResNeg	uint32_t	0 … n	Negative ‘Reset’ response from host to stack

Table 84: Diagnostic Type2 – Command counters

Diagnostic Type3 – CAN counters

```
typedef struct DNS_DIAG_CAN_Ttag
{
  uint32_t ulStatus;
  uint32_t ulTxFrameSucceed;
  uint32_t ulTxErrorSummary;
  uint32_t ulRxFrameSucceed;
  uint32_t ulRxErrorSummary;
  uint32_t ulTxErrCnt;
  uint32_t ulRxErrCnt;
  uint32_t ulArbitrationLost;
  uint32_t ulIndDroppedDueFifoFull;
  uint32_t ulConDroppedDueFifoFull;
  uint32_t ulRxStdFramesFilterd;
  uint32_t ulRxExtFramesFilterd;
  uint32_t ulRxStdFramesPassed;
  uint32_t ulRxExtFramesPassed;

}DNS_DIAG_CAN_T;
```

Variable	Type	Value	Description
ulStatus	uint32_t	0 … 3	CAN controller status: 0 – Active state 1 – Error warning state 2 – Error passive state 3 – Bus Off state
ulTxFrameSucceed	uint32_t	0 … n	Successful CAN frames sent to the network
ulTxErrorSummary	uint32_t	0 … n	Summary of transmission error
ulRxFrameSucceed	uint32_t	0 … n	Successful CAN frames received from the network
ulRxErrorSummary	uint32_t	0 … n	Summary of receive error
ulTxErrCnt	uint32_t	0 … n	Transmission error counter (TEC)
ulRxErrCnt	uint32_t	0 … n	Receive error counter (REC)
ulArbitrationLost	uint32_t	0 … n	Number of arbitration lost while trying to send CAN frames.
ulIndDroppedDueFifoFull	uint32_t	0 … n	CAN controller has dropped a can frame due its Rx FIFO overrun. This may can happen, when the software cannot process the frames fast enough.
ulConDroppedDueFifoFull	uint32_t	0 … n	Number of
ulRxStdFramesFilterd	uint32_t	0 … n	Number of 11 bit CAN frames received, but filtered by CAN controller
ulRxExtFramesFilterd	uint32_t	0 … n	Number of 29 bit CAN frames received, but filtered by CAN controller
ulRxStdFramesPassed	uint32_t	0 … n	Number of 11 bit CAN frames received
ulRxExtFramesPassed	uint32_t	0 … n	Number of 29 bit CAN frames received

Table 85: Diagnostic Type3 – CAN counters

Diagnostic Type3 – Resource counters

Diagnostic type 3 contains some counters used by the stack for internal resource tracking. There is not one-to-one relation between their values and the requests that the host sends.

```
typedef struct DNS_DIAG_RESOURCE_Ttag
{
  uint32_t ulIndServiceIndicated;
  uint32_t ulIndServiceResponded;
  uint32_t ulIndServiceRespondedAfterTimeout;
  uint32_t ulIndServiceReleased;
  uint32_t ulIndServiceTimeout;
  uint32_t ulReqServiceAlloc;
  uint32_t ulReqServiceConfirmed;
  uint32_t ulReqServiceReleased;

}DNS_DIAG_RESOURCE_T;
```

Variable	Type	Value	Description
ulIndServiceIndicated	uint32_t	0 … n	Total number of CIP services indications sent to the host
ulIndServiceResponded	uint32_t	0 … n	Total number of CIP services responded from host in time. (Responses sent from in time before the internal 3 seconds timeout for indications has expired)
ulIndServiceRespondedAfterTimeout	uint32_t	0 … n	Total number of CIP services responded from host after internal timeout for indications. (The host has not answered within the specified time of 3 seconds).
ulIndServiceReleased	uint32_t	0 … n	Number of released buffers to indication pool of the stack. If the host has answered all indications correctly this counter should be the same as value of `‘ulIndServiceIndicated’`.
ulIndServiceTimeout	uint32_t	0 … n	Number of timeout events where the host did not respond right in time to an ‘indication’. In this case, the stack responds the service to the network with an appropriate error code by itself.
ulReqServiceAlloc	uint32_t	0 … n	Number of allocations from ‚CIP services request pool ‘.
ulReqServiceConfirmed	uint32_t	0 … n	Number of CIP services that the stack has confirmed to the host.
ulReqServiceReleased	uint32_t	0 … n	Number of releases to the ‚CIP services request pool ‘. If no CIP request is pending, this value should be the same as `‘ulReqServiceAlloc’` number.

Table 86: Diagnostic Type3 – Resource counters

4.5 Hilscher common services

This chapter briefly describes the Hilscher common services and their adaptation to the DeviceNet Slave stack. Since the Hilscher common services are generic packets, their format and commands are described in reference [2] and [3].

4.5.1 Channel Init

The host application has to send a packet `HIL_CHANNEL_INIT_REQ` to trigger a channel initialization at the protocol stack. Channel Initialization causes the stack to perform a reset and to reinitialize with the given configuration.

Anyway, the actions performed during a channel initialization are partly specific to the DeviceNet Slave stack. At least, the protocol stack will perform the following actions:

- Clear READY and RUN bits
- Set BUS_OFF and stop all communication
- Call the object-specific reset functions of all CIP objects
- Unregister all objects and services previously registered with `DNS_CMD_REGISTER_CLASS_REQ`
- Apply configuration from database, if applicable
- Apply configuration from Set Configuration Packet, if applicable
- Reply with `HIL_CHANNEL_INIT_REQ`

4.5.2 Register / Unregister Application

The host application sends the packets `HIL_REGISTER_APP_REQ` and `HIL_UNREGISTER_APP_REQ`, respectively, to register or unregister the host application with the protocol stack. Unless an application has registered, the stack will not generate any indications toward the host application.

4.5.3 Get / Set Watchdog Time

The host application can send the packet `HIL_GET_WATCHDOG_TIME_REQ` to retrieve the currently configured interval of the watchdog timer, or it sends a `HIL_SET_WATCHDOG_TIME_REQ` to set the interval of the watchdog timer, in units of milliseconds.

4.5.4 Delete Config

The host application can send the packet `HIL_DELETE_CONFIG_REQ` to delete the internally stored configuration from RAM or FLASH. For the DeviceNet stack, this will remove all stored remanent data. Database files on the file system are **not** deleted.

4.5.5 Start / Stop Communication

The host application sends the packet HIL_START_STOP_COMM_REQ to instruct the DeviceNet Slave stack to start or stop network communication, i.e. to set or clear the BUS_ON signal of the netX, according to the contained parameter.

4.5.6 Lock / Unlock Configuration

The host application sends the packet HIL_LOCK_UNLOCK_CONFIG_REQ to lock or unlock configuration data, respectively. A locked configuration cannot be altered.

4.5.7 Get DPM I/O Information

The host application sends the packet HIL_GET_DPM_IO_INFO_REQ to obtain the offsets and lengths of the areas used within the DPM I/O blocks.

4.5.8 Get / Set Trigger Type

The host application sends the packet HIL_GET_TRIGGER_TYPE_REQ to request the currently configured trigger type. The trigger type defines the synchronization mode of IO data exchange between host application and network data transfer.

The host application can sends the packet HIL_SET_TRIGGER_TYPE_REQ to the stack in order to configure the data exchange trigger mode for the IO handshakes and Sync handshake. The trigger mode is individual configurable per direction (send and receive). The predefined trigger types and the packet to configuration is described in reference [2].

The DeviceNet slave stack support the following trigger type combinations:

Receive Direction	**Send Direction**	**Note**
HIL_TRIGGER_TYPE_PDIN_NONE	HIL_TRIGGER_TYPE_PDOUT_NONE	*)
HIL_TRIGGER_TYPE_PDIN_RX_DATA_RECEIVED	HIL_TRIGGER_TYPE_PDOUT_NONE	**)

Table 87: Trigger Type combinations

*) This combination is the default mode and is set per default by the stack at startup. This combination is the so called ‘Free Run’ mode which is explained in chapter *IO Exchange – Free Run* (page 49).

**) This combination is explained in chapter *IO Exchange – RX Data Received* (page 51).

4.5.9 Firmware Identification

The host application sends the packet HIL_FIRMWARE_IDENTIFY_REQ to retrieve version information of the Protocol Stack Firmware running on the netX, i.e. its name, version and date.

4.5.10 Set Remanent Data

In case, the application is responsible to store remanent data (section Remanent data on page 61); it has to use this service during startup to provide the remanent data to the firmware.

For a description of this service and the indication and response packet, see reference [3]. For a State diagram, see section Host application on page 59.

Value for ulComponentID

```
#define HIL_COMPONENT_ID_DEVNET_GCI_SLAVE ((uint32_t)0x011B0000L)
```

4.5.11 Store Remanent Data Indication

In case, the application is responsible to store remanent data (section Remanent data on page 61); the application must handle this service. For a description of this service and the indication and response packet, see reference [3].

Value for ulComponentID

```
#define HIL_COMPONENT_ID_DEVNET_GCI_SLAVE ((uint32_t)0x011B0000L)
```

5 Status information

The DeviceNet Slave provides status information in the dual-port memory. The status information has a common block (protocol-independent) and a DeviceNet Slave specific block (extended status).

5.1 Common status

For a description of the common status block, see reference [1].

5.1.1 DPM Communication status

The common status block includes the communication status. This section describes how the DeviceNet Slave stack is adapted to the communication status.

Status	Description
OFFLINE	The device is not configured. No frames are generated.
STOP	The device is configured and Bus OFF is set. The device is not responsive to network communication. In addition, the Network Power (NP) is missing or might be lost, or a CAN Bus OFF event has appeared.
IDLE	The device is configured, Bus ON is set, the Network Power (NP) is available and the Duplicate MAC ID Check has passed. The device is responsive to the network. An explicit connection might exist. An IO connection (poll, strobe cos/cyclic) does not exist.
OPERATE	The device is configured, Bus ON is set and has active IO-Communication. The device has at least one open IO connection poll, strobe and/or cos/cyclic.

Table 88: Communication Status

The figure below shows the communication status transitions depending on specific events.

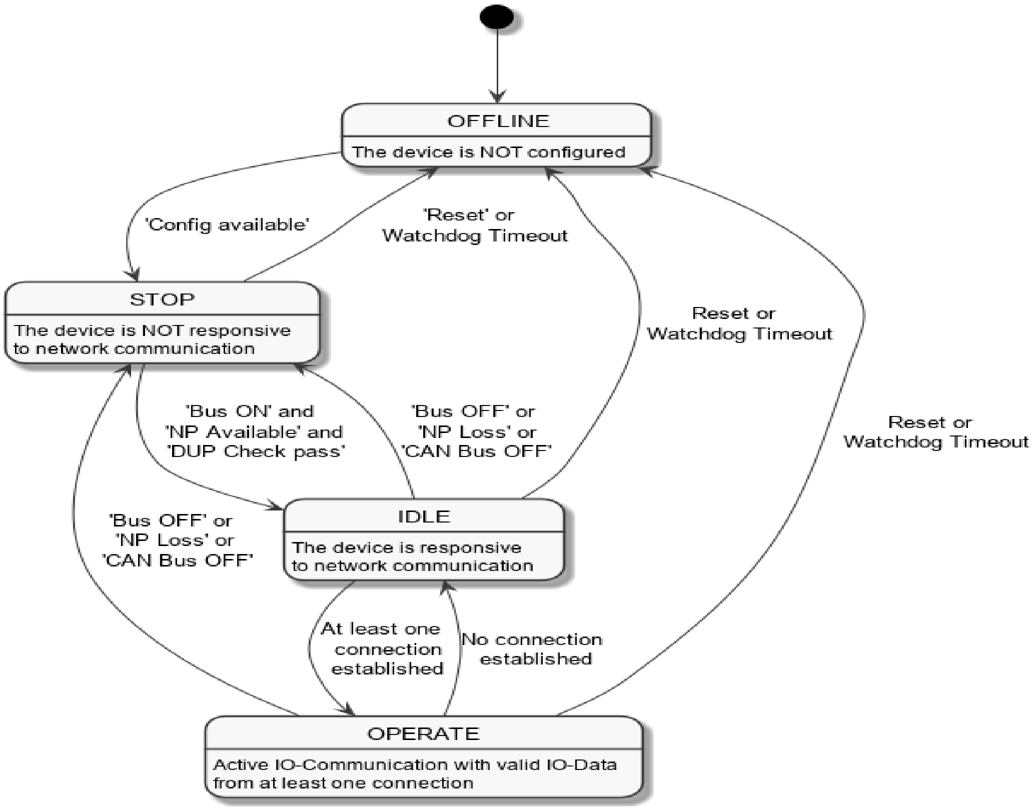
OFFLINE
The device is NOT configured
'Config available'
'Reset' or Watchdog Timeout
STOP
The device is NOT responsive to network communication
Reset or Watchdog Timeout
'Bus ON' and 'NP Available' and 'DUP Check pass'
'Bus OFF' or 'NP Loss' or 'CAN Bus OFF'
Reset or Watchdog Timeout
'Bus OFF' or 'NP Loss' or 'CAN Bus OFF'
IDLE
The device is responsive to network communication
At least one connection established
No connection established
OPERATE
Active IO-Communication with valid IO-Data from at least one connection

Figure 20: Communication status

5.1.2 DPM Status bits

The following table describes the flags of the DPM Status Bits:

Flag	Description
Ready flag	The Ready flag is a basic flag that is set by the communication channel at startup. As soon the Ready flag is set the packet communication with communication channel is possible.
Run flag	The Run flag will be set as soon as the DeviceNet Slave stack has received a valid configuration.
Communication flag	The Communication flag is set for the DPM channel if at least one of the IO connection “poll”, “change of state”, “cyclic” or “bit strobe” is open. It is not set when the explicit connection is open.

Table 89: Flags of the DPM Status Bits

5.2 Extended status

Currently no status data in the Extended Status Area are supported.

5.3 Process data status

Assembly objects represents process data in CIP respective DeviceNet. The data of assembly objects are exchanged with the host application via DPM input and output area.

For output assemblies normally the pure assembly data is located into the DPM input image. Optionally it is possible (by configuration) to map a 4 byte “Run/Idle status” and/or a 4 byte “Sequence Counter” information in front of the assembly data. To enable the Run Idle and “Sequence Counter” mapping refer to Table 65 on page 76.

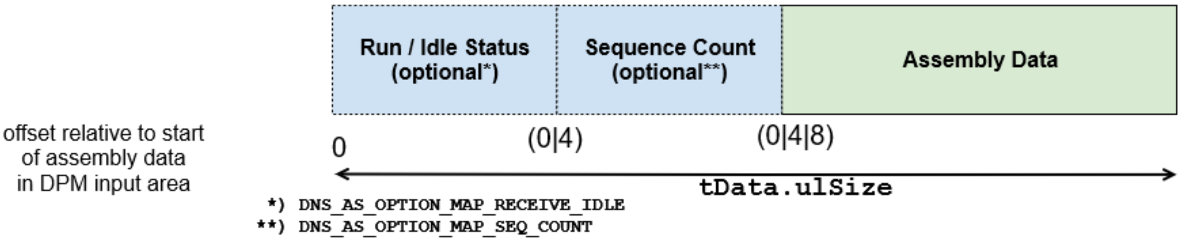
Run / Idle Status (optional*)
Sequence Count (optional**)
Assembly Data
offset relative to start of assembly data in DPM input area
0
(0|4)
(0|4|8)
tData.ulSize
*) DNS_AS_OPTION_MAP_RECEIVE_IDLE
**) DNS_AS_OPTION_MAP_SEQ_COUNT

Figure 21: Run/Idle - Sequence – Assembly Data Layout

5.3.1 Run / Idle status

The Run / Idle header is **not** applicable for producing assembly data, this means for data transfer from host application to the stack.

Value	Define	Description
0x00000000	`DNS_AS_DATA_STATUS_ZERO`	The assembly data are cleared and not valid. This appears typically at startup until no connection is established or when the connection has timed out.
0x00000001	`DNS_AS_DATA_STATUS_RECEIVE_RUN`	The assembly data are valid and can be used by the application.
0x00000002	`DNS_AS_DATA_STATUS_RECEIVE_IDLE`	The master has initiated the “receive idle” mode. This means the master does not send valid data across the connection, but the connection is still established. The slave stack will keep the last received data from the master in assembly data. It is up to the user application to implement an application specific behavior in receive idle state.
0x00000003	`DNS_AS_DATA_STATUS_RECEIVE_IDLE_ZERO`	The same as the state before, but the stack has cleared the assembly data. (not supported)
0x00000004	`DNS_AS_DATA_STATUS_HOLD_LAST_STATE`	The assembly data are in hold last state. (not supported)

Table 90: Assembly data status

5.3.2 Sequence counter

The sequence counter is a counter that is incremented on each message consumption (I/O data reception) from the network. The sequence counter is a counter generated by the stack. It is not part of the data transmitted via network.

- In Run Idle status `DNS_AS_DATA_STATUS_ZERO`, the is counter always zero.
- In Run Idle status `DNS_AS_DATA_STATUS_RECEIVE_RUN`, the counter increments on each poll data consumption.
- In Run Idle status `DNS_AS_DATA_STATUS_RECEIVE_IDLE,` the counter increments on each poll idle consumption.
- On overflow of the counter wraps from 0xFFFFFFFF to 0x00000001.
- In case of disconnecting / closing the connection the counter is reset to zero.
- The counter is NOT a change of data counter.

6 Feature configuration via tag list

Modification of the firmware’s tag list allows controlling of certain behavior, features and resource limits. To modify the tag list, the Hilscher Tag List Editor tool has to be used.

The DeviceNet Slave V5 currently supports at least the following tags:

Tag list parameter

Tag list	Parameter / Tag	Value	Description
Remanent Data responsibility	`HIL_TAG_REMANENT_DATA_RESPONSIBLE`	Disabled	Communication firmware stores remanent data (**default**).
		Enabled	Application stores remanent data. For a description, see section *Remanent data* on page 61.
Initial DDP mode after power on	`HIL_TAG_DDP_MODE_AFTER_STARTUP`	Active	The Device Data Provider is active from the very beginning, providing data from the Flash Device Label (FDL) or SecMem data sources. (**default**)
		Passive	The Device Data Provider is passive until set to “active” by the application, after having set base device information, e.g. serial number.

Table 91: Tag list parameter

7 Status and error codes

7.1 Common status codes

Hexadecimal value	Definition and description
0x00000000	SUCCESS_HIL_OK Operation succeeded.
0xC0000001	ERR_HIL_FAIL Common error, detailed error information optionally present in the data area of packet.
0xC0000002	ERR_HIL_UNEXPECTED Unexpected failure.
0xC0000003	ERR_HIL_OUTOFMEMORY Ran out of memory.
0xC0000004	ERR_HIL_UNKNOWN_COMMAND Unknown Command in Packet received.
0xC0000005	ERR_HIL_UNKNOWN_DESTINATION Unknown Destination in Packet received.
0xC0000006	ERR_HIL_UNKNOWN_DESTINATION_ID Unknown Destination Id in Packet received.
0xC0000007	ERR_HIL_INVALID_PACKET_LEN Packet length is invalid.
0xC0000008	ERR_HIL_INVALID_EXTENSION Invalid Extension in Packet received.
0xC0000009	ERR_HIL_INVALID_PARAMETER Invalid Parameter in Packet found.
0xC000000A	ERR_HIL_INVALID_ALIGNMENT Invalid alignment.
0xC000000C	ERR_HIL_WATCHDOG_TIMEOUT Watchdog error occurred.
0xC000000D	ERR_HIL_INVALID_LIST_TYPE List type is invalid.
0xC000000E	ERR_HIL_UNKNOWN_HANDLE Handle is unknown.
0xC000000F	ERR_HIL_PACKET_OUT_OF_SEQ A packet index has been not in the expected sequence.
0xC0000010	ERR_HIL_PACKET_OUT_OF_MEMORY The amount of fragmented data contained the packet sequence has been too large.
0xC0000011	ERR_HIL_QUE_PACKETDONE The packet done function has failed.
0xC0000012	ERR_HIL_QUE_SENDPACKET The sending of a packet has failed.
0xC0000013	ERR_HIL_POOL_PACKET_GET The request of a packet from packet pool has failed.
0xC0000014	ERR_HIL_POOL_PACKET_RELEASE The release of a packet-to-packet pool has failed.
0xC0000015	ERR_HIL_POOL_GET_LOAD The get packet pool load function has failed.

Hexadecimal value	Definition and description
0xC0000016	ERR_HIL_QUE_GET_LOAD The get queue load function has failed.
0xC0000017	ERR_HIL_QUE_WAITFORPACKET The waiting for a packet from queue has failed.
0xC0000018	ERR_HIL_QUE_POSTPACKET The posting of a packet has failed.
0xC0000019	ERR_HIL_QUE_PEEKPACKET Peeking a packet from queue has failed.
0xC000001A	ERR_HIL_REQUEST_RUNNING Request is already running.
0xC000001B	ERR_HIL_CREATE_TIMER Creating a timer failed.
0xC000001C	ERR_HIL_BUFFER_TOO_SHORT Supplied buffer too short for the data.
0xC000001D	ERR_HIL_NAME_ALREADY_EXIST Supplied name already exists.
0xC000001E	ERR_HIL_PACKET_FRAGMENTATION_TIMEOUT The packet fragmentation has timed out.
0xC0000100	ERR_HIL_INIT_FAULT General initialization fault.
0xC0000101	ERR_HIL_DATABASE_ACCESS_FAILED Database access failure.
0xC0000102	ERR_HIL_CIR_MASTER_PARAMETER_FAILED Master parameter cannot activated at state operate.
0xC0000103	ERR_HIL_CIR_SLAVE_PARAMTER_FAILED Slave parameter cannot activated at state operate.
0xC0000119	ERR_HIL_NOT_CONFIGURED Configuration not available
0xC0000120	ERR_HIL_CONFIGURATION_FAULT General configuration fault.
0xC0000121	ERR_HIL_INCONSISTENT_DATA_SET Inconsistent configuration data.
0xC0000122	ERR_HIL_DATA_SET_MISMATCH Configuration data set mismatch.
0xC0000123	ERR_HIL_INSUFFICIENT_LICENSE Insufficient license.
0xC0000124	ERR_HIL_PARAMETER_ERROR Parameter error.
0xC0000125	ERR_HIL_INVALID_NETWORK_ADDRESS Network address invalid.
0xC0000126	ERR_HIL_NO_SECURITY_MEMORY Security memory chip missing or broken.
0xC0000127	ERR_HIL_NO_MAC_ADDRESS_AVAILABLE No MAC address available.
0xC0000128	ERR_HIL_INVALID_DDP_CONTENT DeviceDataProvider contains invalid data.
0xC0000129	ERR_HIL_FIRMWARE_STARTUP_ERROR Firmware startup failed. Check System logbook for details.

Hexadecimal value	Definition and description
0xC000012A	ERR_HIL_COMM_CHANNEL_STARTUP_ERROR Communication Channel startup failed. Check Communication Channel logbook for details.
0xC000012B	ERR_HIL_FIRMWARE_SPECIFIC_STARTUP_FAILED An error occurred while starting firmware or protocol specific functionality.
0xC000012C	ERR_HIL_INVALID_TAGLIST_CONTENT While evaluating the firmware taglist an invalid taglist parameter was detected.
0xC000012D	ERR_HIL_OPERATION_NOT_POSSIBLE_IN_CURRENT_STATE The requested operation cannot be executed in current state.
0xC000012E	ERR_HIL_REMANENT_DATA_MISSING The requested operation cannot be executed because remanent data was not set correctly.
0xC000012F	ERR_HIL_INVALID_DDP_OEM_SERIALNUMBER_CODING The content of DDPs OEM field SerialNumber cannot be converted for usage with current protocol stack.
0xC0000130	ERR_HIL_INVALID_DDP_OEM_ORDERNUMBER_CODING The content of DDPs OEM field OrderNumber cannot be converted for usage with current protocol stack.
0xC0000131	ERR_HIL_INVALID_DDP_OEM_HARDWAREREVISION_CODING The content of DDPs OEM field HardwareRevision cannot be converted for usage with current protocol stack.
0xC0000132	ERR_HIL_INVALID_DDP_OEM_PRODUCTIONDATE_CODING The content of DDPs OEM field ProductionDate cannot be converted for usage with current protocol stack.
0xC0000140	ERR_HIL_NETWORK_FAULT General communication fault.
0xC0000141	ERR_HIL_CONNECTION_CLOSED Connection closed.
0xC0000142	ERR_HIL_CONNECTION_TIMEOUT Connection timeout.
0xC0000143	ERR_HIL_LONELY_NETWORK Lonely network.
0xC0000144	ERR_HIL_DUPLICATE_NODE Duplicate network address.
0xC0000145	ERR_HIL_CABLE_DISCONNECT Cable disconnected.
0xC0000180	ERR_HIL_BUS_OFF Bus Off flag is set.
0xC0000181	ERR_HIL_CONFIG_LOCK Changing configuration is not allowed.
0xC0000182	ERR_HIL_APPLICATION_NOT_READY Application is not at ready state.
0xC0000183	ERR_HIL_RESET_IN_PROCESS Application is performing a reset.
0xC0000200	ERR_HIL_WATCHDOG_TIME_INVALID Watchdog time is out of range.
0xC0000201	ERR_HIL_APPLICATION_ALREADY_REGISTERED Application is already registered.
0xC0000202	ERR_HIL_NO_APPLICATION_REGISTERED No application registered.

Hexadecimal value	Definition and description
0xC0000203	ERR_HIL_INVALID_COMPONENT_ID Invalid component identifier.
0xC0000204	ERR_HIL_INVALID_DATA_LENGTH Invalid data length.
0xC0000205	ERR_HIL_DATA_ALREADY_SET The data was already set.
0xC0000206	ERR_HIL_NO_LOGBOOK_AVAILABLE Logbook not available.
0xC0001000	ERR_HIL_INVALID_HANDLE No description available - ERR_HIL_INVALID_HANDLE.
0xC0001001	ERR_HIL_UNKNOWN_DEVICE No description available - ERR_HIL_UNKNOWN_DEVICE.
0xC0001002	ERR_HIL_RESOURCE_IN_USE No description available - ERR_HIL_RESOURCE_IN_USE.
0xC0001003	ERR_HIL_NO_MORE_RESOURCES No description available - ERR_HIL_NO_MORE_RESOURCES.
0xC0001004	ERR_HIL_DRV_OPEN_FAILED No description available - ERR_HIL_DRV_OPEN_FAILED.
0xC0001005	ERR_HIL_DRV_INITIALIZATION_FAILED No description available - ERR_HIL_DRV_INITIALIZATION_FAILED.
0xC0001006	ERR_HIL_DRV_NOT_INITIALIZED No description available - ERR_HIL_DRV_NOT_INITIALIZED.
0xC0001007	ERR_HIL_DRV_ALREADY_INITIALIZED No description available - ERR_HIL_DRV_ALREADY_INITIALIZED.
0xC0001008	ERR_HIL_CRC No description available - ERR_HIL_CRC.
0xC0001010	ERR_HIL_DRV_INVALID_RESOURCE No description available - ERR_HIL_DRV_INVALID_RESOURCE.
0xC0001011	ERR_HIL_DRV_INVALID_MEM_RESOURCE No description available - ERR_HIL_DRV_INVALID_MEM_RESOURCE.
0xC0001012	ERR_HIL_DRV_INVALID_MEM_SIZE No description available - ERR_HIL_DRV_INVALID_MEM_SIZE.
0xC0001013	ERR_HIL_DRV_INVALID_PHYS_MEM_BASE No description available - ERR_HIL_DRV_INVALID_PHYS_MEM_BASE.
0xC0001014	ERR_HIL_DRV_INVALID_PHYS_MEM_SIZE No description available - ERR_HIL_DRV_INVALID_PHYS_MEM_SIZE.
0xC0001015	ERR_HIL_DRV_UNDEFINED_HANDLER No description available - ERR_HIL_DRV_UNDEFINED_HANDLER.
0xC0001020	ERR_HIL_DRV_ILLEGAL_VECTOR_ID No description available - ERR_HIL_DRV_ILLEGAL_VECTOR_ID.
0xC0001021	ERR_HIL_DRV_ILLEGAL_IRQ_MASK No description available - ERR_HIL_DRV_ILLEGAL_IRQ_MASK.
0xC0001022	ERR_HIL_DRV_ILLEGAL_SUBIRQ_MASK No description available - ERR_HIL_DRV_ILLEGAL_SUBIRQ_MASK.
0xC0001023	ERR_HIL_DRV_STATE_INVALID Driver is in invalid state.
0xC0001100	ERR_HIL_DPM_CHANNEL_UNKNOWN No description available - ERR_HIL_DPM_CHANNEL_UNKNOWN.

Hexadecimal value	Definition and description
0xC0001101	ERR_HIL_DPM_CHANNEL_INVALID No description available - ERR_HIL_DPM_CHANNEL_INVALID.
0xC0001102	ERR_HIL_DPM_CHANNEL_NOT_INITIALIZED No description available - ERR_HIL_DPM_CHANNEL_NOT_INITIALIZED.
0xC0001103	ERR_HIL_DPM_CHANNEL_ALREADY_INITIALIZED No description available - ERR_HIL_DPM_CHANNEL_ALREADY_INITIALIZED.
0xC0001120	ERR_HIL_DPM_CHANNEL_LAYOUT_UNKNOWN No description available - ERR_HIL_DPM_CHANNEL_LAYOUT_UNKNOWN.
0xC0001121	ERR_HIL_DPM_CHANNEL_SIZE_INVALID No description available - ERR_HIL_DPM_CHANNEL_SIZE_INVALID.
0xC0001122	ERR_HIL_DPM_CHANNEL_SIZE_EXCEEDED No description available - ERR_HIL_DPM_CHANNEL_SIZE_EXCEEDED.
0xC0001123	ERR_HIL_DPM_CHANNEL_TOO_MANY_BLOCKS No description available - ERR_HIL_DPM_CHANNEL_TOO_MANY_BLOCKS.
0xC0001130	ERR_HIL_DPM_BLOCK_UNKNOWN No description available - ERR_HIL_DPM_BLOCK_UNKNOWN.
0xC0001131	ERR_HIL_DPM_BLOCK_SIZE_EXCEEDED No description available - ERR_HIL_DPM_BLOCK_SIZE_EXCEEDED.
0xC0001132	ERR_HIL_DPM_BLOCK_CREATION_FAILED No description available - ERR_HIL_DPM_BLOCK_CREATION_FAILED.
0xC0001133	ERR_HIL_DPM_BLOCK_OFFSET_INVALID No description available - ERR_HIL_DPM_BLOCK_OFFSET_INVALID.
0xC0001140	ERR_HIL_DPM_CHANNEL_HOST_MBX_FULL No description available - ERR_HIL_DPM_CHANNEL_HOST_MBX_FULL.
0xC0001141	ERR_HIL_DPM_CHANNEL_SEGMENT_LIMIT No description available - ERR_HIL_DPM_CHANNEL_SEGMENT_LIMIT.
0xC0001142	ERR_HIL_DPM_CHANNEL_SEGMENT_UNUSED No description available - ERR_HIL_DPM_CHANNEL_SEGMENT_UNUSED.
0xC0001143	ERR_HIL_NAME_INVALID No description available - ERR_HIL_NAME_INVALID.
0xC0001144	ERR_HIL_UNEXPECTED_BLOCK_SIZE No description available - ERR_HIL_UNEXPECTED_BLOCK_SIZE.
0xC0001145	ERR_HIL_COMPONENT_BUSY The component is busy and cannot handle the requested service.
0xC0001150	ERR_HIL_INVALID_HEADER Invalid (file) header. E.g. wrong CRC/MD5/Cookie.
0xC0001151	ERR_HIL_INCOMPATIBLE Firmware does not match device.
0xC0001152	ERR_HIL_NOT_AVAILABLE Update file or destination (XIP-Area) not found.
0xC0001153	ERR_HIL_READ Failed to read from file/area.
0xC0001154	ERR_HIL_WRITE Failed to write from file/area.
0xC0001155	ERR_HIL_IDENTICAL Update firmware and installed firmware are identical.
0xC0001156	ERR_HIL_INSTALLATION Error during installation of firmware.

Hexadecimal value	Definition and description
0xC0001157	ERR_HIL_VERIFICATION Error during verification of firmware.
0xC0001158	ERR_HIL_INVALIDATION Error during invalidation of firmware files.
0xC0001160	ERR_HIL_FORMAT Volume is not formatted.
0xC0001161	ERR_HIL_VOLUME (De-)Initialization of volume failed.
0xC0001162	ERR_HIL_VOLUME_DRV (De-)Initialization of volume driver failed.
0xC0001163	ERR_HIL_VOLUME_INVALID The volume is invalid.
0xC0001164	ERR_HIL_VOLUME_EXCEEDED Number of supported volumes exceeded.
0xC0001165	ERR_HIL_VOLUME_MOUNT The volume is mounted (in use).
0xC0001166	ERR_HIL_ERASE Failed to erase file/directory/flash.
0xC0001167	ERR_HIL_OPEN Failed to open file/directory.
0xC0001168	ERR_HIL_CLOSE Failed to close file/directory.
0xC0001169	ERR_HIL_CREATE Failed to create file/directory.
0xC0001170	ERR_HIL_MODIFY Failed to modify file/directory.
0xC0001171	ERR_HIL_FS_NOT_AVAILABLE File system not available.
0xC0001172	ERR_HIL_FILE_NOT_FOUND File not available.
0xC0001173	ERR_HIL_DIAG_NO_INFO No diagnostic information available.
0xC0001174	ERR_HIL_QUEUE_UNKNOWN Queue is not available.
0xC0001175	ERR_HIL_NAME_UNKNOWN Name is unknown / not available.
0xC0001176	ERR_HIL_UPDATE_ERROR Failed to update firmware.
0xC0001177	ERR_HIL_DDP_STATE_INVALID DDP is in wrong state.
0xC0001178	ERR_HIL_MANUFACTURER_INVALID Manufacturer in file header does not match target.
0xC0001179	ERR_HIL_DEVICE_CLASS_INVALID Device class in file header does not match target.
0xC000117A	ERR_HIL_HW_COMPATIBILITY_INVALID Hardware compatibility index in file header does not match target.
0xC000117B	ERR_HIL_HW_OPTIONS_INVALID Hardware options in file header does not match target.

Hexadecimal value	Definition and description
0x0000F005	SUCCESS_HIL_FRAGMENTED Fragment accepted.
0xC000F006	ERR_HIL_RESET_REQUIRED Reset required.
0xC000F007	ERR_HIL_EVALUATION_TIME_EXPIRED Evaluation time expired. Reset required.
0xC000DEAD	ERR_HIL_FIRMWARE_CRASHED The firmware has crashed and the exception handler is running.

Table 92: Common status codes

7.2 Generic AP

Hexadecimal value	Definition and description
0xC1090001	ERR_GAP_INVALID_COMPONENT_ID Invalid component identifier.
0xC1090002	ERR_GAP_SET_REMANENT_DATA_NOT_ALLOWED Setting remanent data is not allowed.
0xC1090003	ERR_GAP_INVALID_WORKER Invalid worker.
0xC1090004	ERR_GAP_LOGBOOK_CREATE_FAIL The Logbook could not be created.
0xC1090005	ERR_GAP_POOL_CREATE_FAIL The service pool could not be created.
0xC1090006	ERR_GAP_QUEUE_CREATE_FAIL The GAP queue could not be created.
0xC1090007	ERR_GAP_MUTEX_INIT_FAIL The initialization of a mutex failed.
0xC1090008	ERR_GAP_TIMER_INIT_FAIL The initialization of a PS Timer failed.
0xC1090009	ERR_GAP_COMPMGR_REGISTER_FAIL Generic AP could not register at Component Manager.
0xC109000A	ERR_GAP_COMPMGR_DIAG_REGISTER_FAIL Generic AP could not register the diagnostic callback at Component Manager.
0xC109000B	ERR_GAP_WATCHDOG_INIT_FAIL The DPM watchdog initialization failed.
0xC109000C	ERR_GAP_SET_CHANNEL_QUEUE_FAIL The DPM channel queue could not be registered.
0xC109000D	ERR_GAP_GET_CHANNEL_QUEUE_FAIL The DPM channel queue could not be retrieved.
0xC109000E	ERR_GAP_REMANENT_FILE_CREATE_FAIL The remanent data file could not be created.
0xC109000F	ERR_GAP_COMPONENT_INIT_FAIL The component initialization failed.
0xC1090010	ERR_GAP_COMPONENT_START_FAIL The component could not be started.

Hexadecimal value	Definition and description
0xC1090011	ERR_GAP_REMANENT_DATA_NOT_ENOUGH_MEMORY Not enough memory for remanent data.
0xC1090012	ERR_GAP_FRAGMENT_FORWARD_INIT_FAIL The fragment forwarding initialization failed.
0xC1090013	ERR_GAP_DPM_CHANNEL_START_FAIL The DPM channel could not be started.
0xC1090014	ERR_GAP_RESOURCE_ALLOCATION_FAIL Failed to allocate Generic AP resources.
0xC1090015	ERR_GAP_CONFIG_VERSION_INVALID Invalid Generic AP configuration version.
0xC1090016	ERR_GAP_CONFIG_DPM_COMM_CHANNEL_INVALID Invalid DPM communication channel.
0xC1090017	ERR_GAP_CONFIG_DPM_CHANNEL_FW_INFO_INVALID Invalid firmware information for the related DPM communication channel.
0xC1090018	ERR_GAP_REMANENT_FILE_READ_FAIL The remanent data file could not be read.
0xC1090019	ERR_GAP_REMANENT_FILE_WRITE_FAIL The remanent data file could not be written.
0xC109001A	ERR_GAP_REMANENT_DATA_DELETE_FAIL The remanent data could not be deleted.
0x8109001B	WARN_GAP_SET_REMANENT_DATA_DENIED The component did not accept the remanent data.
0x8109001C	WARN_GAP_SERVICE_DENIED The component did not accept the service.
0xC109001D	ERR_GAP_NO_RESPONSE_HANDLER No response service handler could be found.
0x8109001E	WARN_GAP_SERVICE_RETRY The forwarding of a service to the component or application has failed and will be repeated.
0xC109001F	ERR_GAP_QUEUE_INVALID_SERVICE_LEN Queue service length is invalid.
0xC1090020	ERR_GAP_DPM_INVALID_SERVICE_LEN DPM service length is invalid.
0x81090021	WARN_GAP_NO_REMANENT_DATA Remanent data does not exist.
0xC1090022	ERR_GAP_QUEUE_JOB Service job queued failed.

Table 93: Generic AP status codes

7.3 DeviceNet Slave stack

Hexadecimal value	Definition and description
0xC11A0000	ERR_DEVNET_OBJECT_UNKNOWN DeviceNet Object unknown error.
0xC11A0001	ERR_DEVNET_OBJECT_USER_OBJECT_ALREADY_REGISTERED DeviceNet Object already registered.
0xC11A0002	ERR_DEVNET_OBJECT_USER_OBJECT_REGISTER_LIMIT_REACHED The maximum number of objects that can be registered is reached.
0xC11B0000	ERR_DEVNET_GCI_DNS_UNKNOWN DeviceNet GCI Adapter Slave unknown error
0xC11B0001	ERR_DEVNET_GCI_DNS_NETWORK_POWER_LOSS 24V Network Power Missing.
0xC11B0002	ERR_DEVNET_GCI_DNS_DUPLICATE_MAC_DETECTED Duplicate MAC ID found.
0xC11B0003	ERR_DEVNET_GCI_DNS_CAN_BUS_OFF Network error CAN BUS OFF detected.
0xC11B0004	ERR_DEVNET_GCI_DNS_WRONG_OR_MISSING_CONFIGURATION The configuration is missing or not correct.
0xC11B0005	ERR_DEVNET_GCI_DNS_CONFIGURED_BY_DATABASE The device is already configured by a database file.
0xC1190000	ERR_DEVNET_CORE_UNKNOWN DeviceNet core unknown error.
0xC1190001	ERR_DEVNET_CORE_INVALID_PRODUCED_SIZE Invalid produce size.
0xC1190002	ERR_DEVNET_CORE_INVALID_CONSUMED_SIZE Invalid consume size.
0xC1190003	ERR_DEVNET_CORE_NO_BUS_COMMUNICATION No network communication.

Table 94: DeviceNet Slave status Codes

8 Appendix

8.1 List of figures

8.2 List of tables

8.3 Legal Notes

Copyright

The images, photographs and texts in the accompanying materials (in the form of a user's manual, operator's manual, Statement of Work document and all other document types, support texts, documentation, etc.) are protected by German and international copyright and by international trade and protective provisions. Without the prior written consent, you do not have permission to duplicate them either in full or in part using technical or mechanical methods (print, photocopy or any other method), to edit them using electronic systems or to transfer them. You are not permitted to make changes to copyright notices, markings, trademarks or ownership declarations. Illustrations are provided without taking the patent situation into account. Any company names and product designations provided in this document may be brands or trademarks by the corresponding owner and may be protected under trademark, brand or patent law. Any form of further use shall require the express consent from the relevant owner of the rights.

Important notes

Utmost care was/is given in the preparation of the documentation at hand consisting of a user's manual, operating manual and any other document type and accompanying texts. However, errors cannot be ruled out. Therefore, we cannot assume any guarantee or legal responsibility for erroneous information or liability of any kind. You are hereby made aware that descriptions found in the user's manual, the accompanying texts and the documentation neither represent a guarantee nor any indication on proper use as stipulated in the agreement or a promised attribute. It cannot be ruled out that the user's manual, the accompanying texts and the documentation do not completely match the described attributes, standards or any other data for the delivered product. A warranty or guarantee with respect to the correctness or accuracy of the information is not assumed.

We reserve the right to modify our products and the specifications for such as well as the corresponding documentation in the form of a user's manual, operating manual and/or any other document types and accompanying texts at any time and without notice without being required to notify of said modification. Changes shall be taken into account in future manuals and do not represent an obligation of any kind, in particular there shall be no right to have delivered documents revised. The manual delivered with the product shall apply.

Under no circumstances shall Hilscher Gesellschaft für Systemautomation mbH be liable for direct, indirect, ancillary or subsequent damage, or for any loss of income, which may arise after use of the information contained herein.

Liability disclaimer

The hardware and/or software was created and tested by Hilscher Gesellschaft für Systemautomation mbH with utmost care and is made available as is. No warranty can be assumed for the performance or flawlessness of the hardware and/or software under all application conditions and scenarios and the work results achieved by the user when using the hardware and/or software. Liability for any damage that may have occurred as a result of using the hardware and/or software or the corresponding documents shall be limited to an event involving willful intent or a grossly negligent violation of a fundamental contractual obligation. However, the right to assert damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

It is hereby expressly agreed upon in particular that any use or utilization of the hardware and/or software in connection with

- Flight control systems in aviation and aerospace;
- Nuclear fission processes in nuclear power plants;
- Medical devices used for life support and
- Vehicle control systems used in passenger transport

shall be excluded. Use of the hardware and/or software in any of the following areas is strictly prohibited:

- For military purposes or in weaponry;
- For designing, engineering, maintaining or operating nuclear systems;
- In flight safety systems, aviation and flight telecommunications systems;
- In life-support systems;
- In systems in which any malfunction in the hardware and/or software may result in physical injuries or fatalities.

You are hereby made aware that the hardware and/or software was not created for use in hazardous environments, which require fail-safe control mechanisms. Use of the hardware and/or software in this kind of environment shall be at your own risk; any liability for damage or loss due to impermissible use shall be excluded.

Warranty

Hilscher Gesellschaft für Systemautomation mbH hereby guarantees that the software shall run without errors in accordance with the requirements listed in the specifications and that there were no defects on the date of acceptance. The warranty period shall be 12 months commencing as of the date of acceptance or purchase (with express declaration or implied, by customer's conclusive behavior, e.g. putting into operation permanently).

The warranty obligation for equipment (hardware) we produce is 36 months, calculated as of the date of delivery ex works. The aforementioned provisions shall not apply if longer warranty periods are mandatory by law pursuant to Section 438 (1.2) BGB, Section 479 (1) BGB and Section 634a (1) BGB [Bürgerliches Gesetzbuch; German Civil Code] If, despite of all due care taken, the delivered product should have a defect, which already existed at the time of the transfer of risk, it shall be at our discretion to either repair the product or to deliver a replacement product, subject to timely notification of defect.

The warranty obligation shall not apply if the notification of defect is not asserted promptly, if the purchaser or third party has tampered with the products, if the defect is the result of natural wear, was caused by unfavorable operating conditions or is due to violations against our operating regulations or against rules of good electrical engineering practice, or if our request to return the defective object is not promptly complied with.

Costs of support, maintenance, customization and product care

Please be advised that any subsequent improvement shall only be free of charge if a defect is found. Any form of technical support, maintenance and customization is not a warranty service, but instead shall be charged extra.

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterruptable or the hardware and/or software is fault-free.

It cannot be guaranteed that patents and/or ownership privileges have not been infringed upon or violated or that the products are free from third-party influence. No additional guarantees or promises shall be made as to whether the product is market current, free from deficiency in title, or can be integrated or is usable for specific purposes, unless such guarantees or promises are required under existing law and cannot be restricted.

Confidentiality

The customer hereby expressly acknowledges that this document contains trade secrets, information protected by copyright and other patent and ownership privileges as well as any related rights of Hilscher Gesellschaft für Systemautomation mbH. The customer agrees to treat as confidential all of the information made available to customer by Hilscher Gesellschaft für Systemautomation mbH and rights, which were disclosed by Hilscher Gesellschaft für Systemautomation mbH and that were made accessible as well as the terms and conditions of this agreement itself.

The parties hereby agree to one another that the information that each party receives from the other party respectively is and shall remain the intellectual property of said other party, unless provided for otherwise in a contractual agreement.

The customer must not allow any third party to become knowledgeable of this expertise and shall only provide knowledge thereof to authorized users as appropriate and necessary. Companies associated with the customer shall not be deemed third parties. The customer must obligate authorized users to confidentiality. The customer should only use the confidential information in connection with the performances specified in this agreement.

The customer must not use this confidential information to his own advantage or for his own purposes or rather to the advantage or for the purpose of a third party, nor must it be used for commercial purposes and this confidential information must only be used to the extent provided for in this agreement or otherwise to the extent as expressly authorized by the disclosing party in written form. The customer has the right, subject to the obligation to confidentiality, to disclose the terms and conditions of this agreement directly to his legal and financial consultants as would be required for the customer's normal business operation.

Export provisions

The delivered product (including technical data) is subject to the legal export and/or import laws as well as any associated regulations of various countries, especially such laws applicable in Germany and in the United States. The products / hardware / software must not be exported into such countries for which export is prohibited under US American export control laws and its supplementary provisions. You hereby agree to strictly follow the regulations and to yourself be responsible for observing them. You are hereby made aware that you may be required to obtain governmental approval to export, reexport or import the product.

8.4 Contacts

Headquarters

Germany
Hilscher Gesellschaft für Systemautomation mbH
Rheinstraße 15
D-65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-mail: info@hilscher.com
Support
Phone: +49 (0) 6190 9907-990
E-mail: hotline@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-mail: info@hilscher.cn
Support
Phone: +86 (0) 21-6355-5161
E-mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69800 Saint Priest
Phone: +33 (0) 4 72 37 98 40
E-mail: info@hilscher.fr
Support
Phone: +33 (0) 4 72 37 98 40
E-mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai, Bangalore
Phone: +91 8888 750 777
E-mail: info@hilscher.in
Support
Phone: +91 8108884011
E-mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-mail: info@hilscher.it
Support
Phone: +39 02 25007068
E-mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-mail: info@hilscher.jp
Support
Phone: +81 (0) 3-5362-0521
E-mail: jp.support@hilscher.com

Republic of Korea
Hilscher Korea Inc.
13494, Seongnam, Gyeonggi
Phone: +82 (0) 31-739-8361
E-mail: info@hilscher.kr
Support
Phone: +82 (0) 31-739-8363
E-mail: kr.support@hilscher.com

Austria
Hilscher Austria GmbH
4020 Linz
Phone: +43 732 931 675-0
E-mail: sales.at@hilscher.com
Support
Phone: +43 732 931 675-0
E-mail: at.support@hilscher.com

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-mail: info@hilscher.ch
Support
Phone: +41 (0) 32 623 6633
E-mail: support.swiss@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-mail: info@hilscher.us
Support
Phone: +1 630-505-5301
E-mail: us.support@hilscher.com