Protocol API

DeviceNet Slave

V2.7.0

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC060202API18EN | Revision 18 | English | 2020-10 | Released | Public

Table of Contents

1 Introduction

1.1 About this document

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

1.2 List of revisions

Rev	Date	Name	Revisions
16	2018-07-23	RG/HH/TD	Firmware / stack version V2.6.0 Removed section on hardware switch (feature is not supported)
			Document restructured. Sections *Fundamentals* and *Dual-Port Memory* removed as they are part of references [1] and [2].
			Section *References to documents* updated.
			Section *Object model*: Assembly Object added in Figure 4.
			Section *Assembly Object (Class Code 0x04)* added.
			Section *Set Configuration Service*: Config flag MSK_DNS_CFG_FLAG_DISABLE_GET_SET_ATT_IND added.
			Section *Communication state* added.
			Section *Register Class Service*: ulClass and ulAccessTyp updated in Table 52.
			Section *Combined LED (MNS) State Service* added.
			Section *Status information* added. Subsection *Status in the input data area* describes data status in input memory area.
17	2019-04-30	HH/TD	Section *Set Configuration Request*: Parameter ranges updated.
			Section *Local Service Request*: Range of bServiceCode now is 0-49.
			Section *Modify Firmware Parameter* added.
			Section *Separated LED (MS and NS) State Service*: Values for "No change" added.
18	2020-10-12	HHE	Firmware / stack version V2.7.0
			Target for COMX 52.

Table 1: List of Revisions

1.3 System requirements

This software package has the following system requirements:

- netX-Chip as CPU hardware platform
- operating system for task scheduling required

1.4 Intended audience

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

- Knowledge of the programming language C
- Knowledge of the use of the real-time operating system rcX

1.5 Specifications

The data below applies to DeviceNet Slave firmware and stack version V2.7.0.

1.5.1 Technical data

Maximum number of cyclic input data	255 bytes
Maximum number of cyclic output data	255 bytes
Acyclic communication as server	Get_Attribute_Single max. 240 bytes per request Set_Attribute_Single max. 240 bytes per request Other Services max. 248 bytes per request
Baud rates	125 kBits/s, 250 kBit/s, 500 kBit/s Auto-detection mode is not supported.
Connections	Poll Change-of-state Cyclic Bit-strobe
Explicit messaging	supported
Fragmentation	Explicit and I/O

Firmware/stack available for netX

netX 10	yes
netX 50	yes
netX 51	yes
netX 52	yes
netX 100, netX 500	yes

PCI

DMA Support for PCI targets	yes

Slot Number

Slot number supported for	CIFX 50-DN

Configuration

Configuration is done by sending packets to the stack, by using SYCON.net configuration database, or using netX Configuration Tool.

Diagnostic

Firmware supports common diagnostic in the dual-port-memory for loadable firmware.

1.5.2 Limitations

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

The quick connect feature allows the device to save 2 seconds of duplicate MAC ID check. The quick connection slave device can go online directly after power cyclic or soft reset. For further information please refer to ODVA documentation.

1.6 Terms, abbreviations, definitions

Term	Description
AP	Application on top of the Stack
AREP	Application Reference End Point
ASCII	American Standard Code for Information Interchange
BOI	Bus-off interrupt
CAN	Controller Area Network
CIP	Common Industrial Protocol
COS	Change of State
DL	Data Link (Layer)
DNS	DeviceNet Slave
DPM	Dual Port Memory
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

Table 2: Terms, abbreviations and definitions

All variables, parameters and data used in this manual have basically the LSB/MSB (“Intel”) data representation. This corresponds to the convention of the Microsoft C Compiler.

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 4, English, 2020.

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

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

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

[6] Hilscher Gesellschaft für Systemautomation mbH: Operating Instruction Manual, Tag List Editor V1.4.

Table 3: References to documents

2 Getting started

This chapter describes the basics of the Hilscher DeviceNet Slave stack. This includes information about

- use cases and stack types (LFW/LOM)
- configuration methods of the DeviceNet Slave stack

2.1 Stack types

The DeviceNet Slave protocol stack can be used in two different use cases:

- Loadable Firmware (LFW)
- Linkable Object Modules (LOM)

2.1.1 Use case: Loadable Firmware (LFW)

The application and the DeviceNet Slave protocol stack run on different processors. While the host application runs on a computer typically equipped with an operating system (such as Microsoft Windows® or Linux), the DeviceNet Slave protocol stack runs on the netX processor together with a connecting software layer, the AP task. 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. This situation is shown in Figure 1:

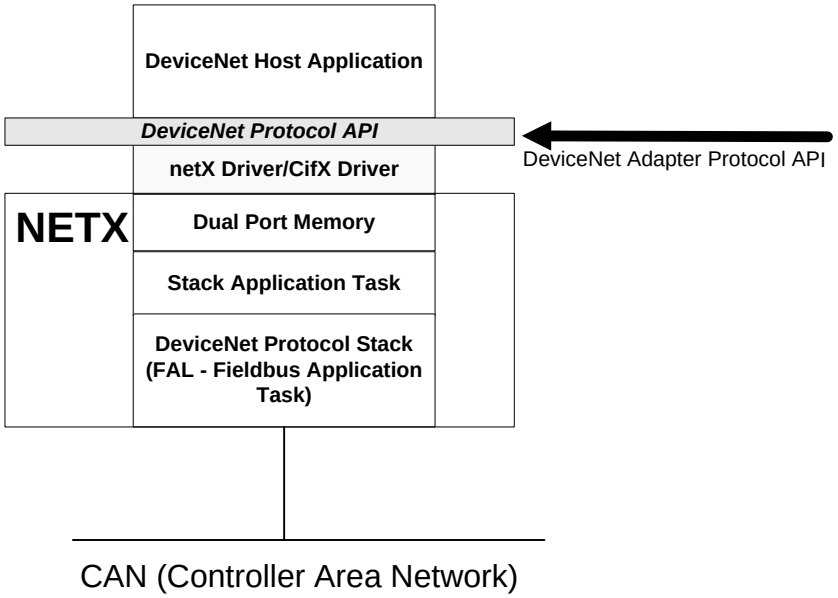


Figure 1: Use case: Loadable Firmware

2.1.2 Use case: Linkable Object Module (LOM)

Both, the application and the DeviceNet Slave protocol stack are executed on netX. There is no need for drivers or a stack-specific AP task. Application and protocol stack are statically linked.

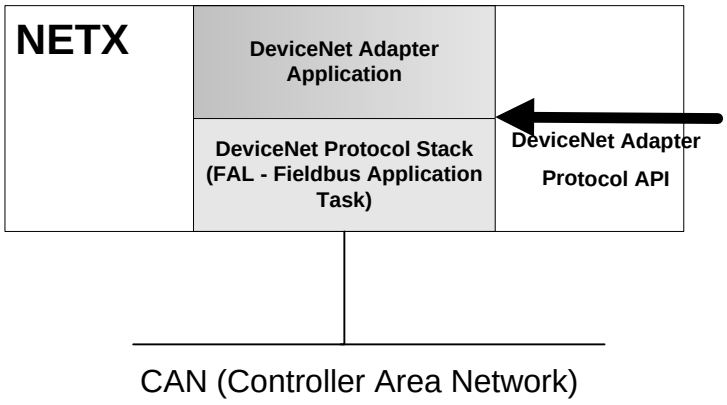


Figure 2: Use case: Linkable Object Modules

If the stack is used as Linkable Object Module, the user has to create its own configuration file (which among others contains task start-up parameters and hardware resource declarations).

2.2 Configuration

2.2.1 Configuration methods

You can use one of the following methods to configure the DeviceNet Slave stack:

- The application can set the configuration parameters using the Set Configuration service to transfer the parameters within a packet to the stack.
- You can use one of the following configuration software to set the configuration parameters.
  - You can use SYCON.net configuration software (which creates a configuration file named CONFIG.NXD). This configuration software described in a separate documentation, see reference [5].
    or
  - You can use the netX Configuration Tool (which creates a file named INIBATCH.NXD)

2.2.2 Application sets the configuration parameters

Configuration sequence

Depending on the use case different services are provided by the DeviceNet Slave protocol stack. The required configuration sequence depends on the use case and on the CIP Object Model required by the device.

Use case	Configuration sequence	Description
Loadable Firmware	*Configuration sequence (Loadable Firmware)* on page 22.	The configuration sequence needed in order to configure and start the DeviceNet Slave stack as Loadable Firmware (LFW) via DPM as well as register the host application for indications generated by the stack.
Linkable Object Module	See *Configuration sequence* (Linkable Object Module) on page 91.	The user application is developed on netX as a task and communicates with DeviceNet Slave stack task via packets. An extended configuration sequence has to be implemented by the user application task in order to configure and start the stack.

Table 4: Configuration sequences and related functions

2.3 Process Data (Input and Output) – Cyclic data exchange

Devicet uses I/O connections to transfer I/O data. A DeviceNet Slave produces and consumes I/O data. The size of produced or consumed data and the connection type is subject of the configuration of the DeviceNet Slave.

I/O data is stored in an input and output area. The input and output data area is divided into the following sections:

I/O Offset	Area	Length (Byte)	Type
0x1000	Output block	Max. 255	Read/Write
0x2680	Input block	Max. 255	Read

Table 5: Input and Output Data

2.4 Acyclic data exchange

For acyclic data exchange, DeviceNet uses the *Explicit Messaging Services* (see page 57). For reading or writing data, the Get_Attribute or Set_Attribute service is to be used. Data is stored in objects, see section *Object model* on page 15.

3 Stack features

3.1 Structure of the DeviceNet Slave stack

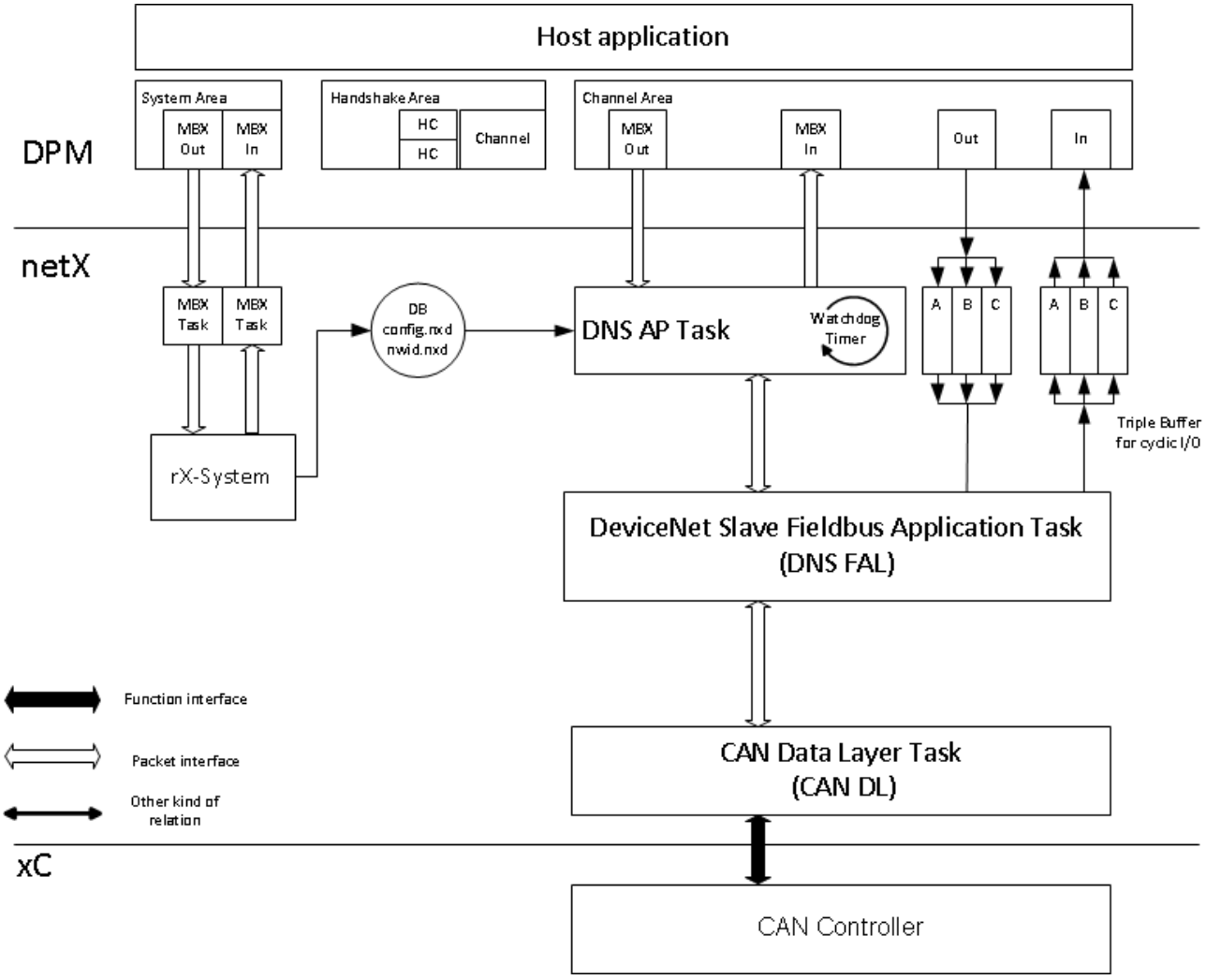


Figure 3: Structure of the DeviceNet Slave stack

Host Application Task (is optional)

The user-implemented application handles further requirements like device profile implementation or some other advanced/customized handling of data exchange.

AP Task (DeviceNet Slave Application Task)

The DNS AP Task contains the process queue which handles all incoming and outgoing packets from the user application. It provides the communication channel for the underlying DNS FAL stack. Once, the stack communication is established, mailbox packets are sent to the process queue of the AP Task. Every packet is evaluated in context of the AP Task and corresponding actions are executed. The AP Task is the user interface to the DNS FAL stack.

DNS FAL Task

The DNS FAL Task is the DeviceNet protocol stack. This task handles all aspects of the DeviceNet protocol. Application packets sent from the AP Task are processed through the internal process queue. Once configured by the AP Task the DNS FAL Task will handle all DeviceNet communications like cyclic I/O message processing and acyclic messaging. This task provides acyclic messaging to the AP Task through mailbox indication packets.

CAN DL Task

The CAN DL Task handlers all process queue information between the DNS FAL Task and the underlying CAN hardware layer. An interface to this task is not available via the AP Task.

3.2 Object model

The device 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 objects are present in the default Hilscher DeviceNet Slave device. The user application is free to define device / manufacturer-specific objects and register them with the DeviceNet protocol stack, see section *Remote Service* on page 75.

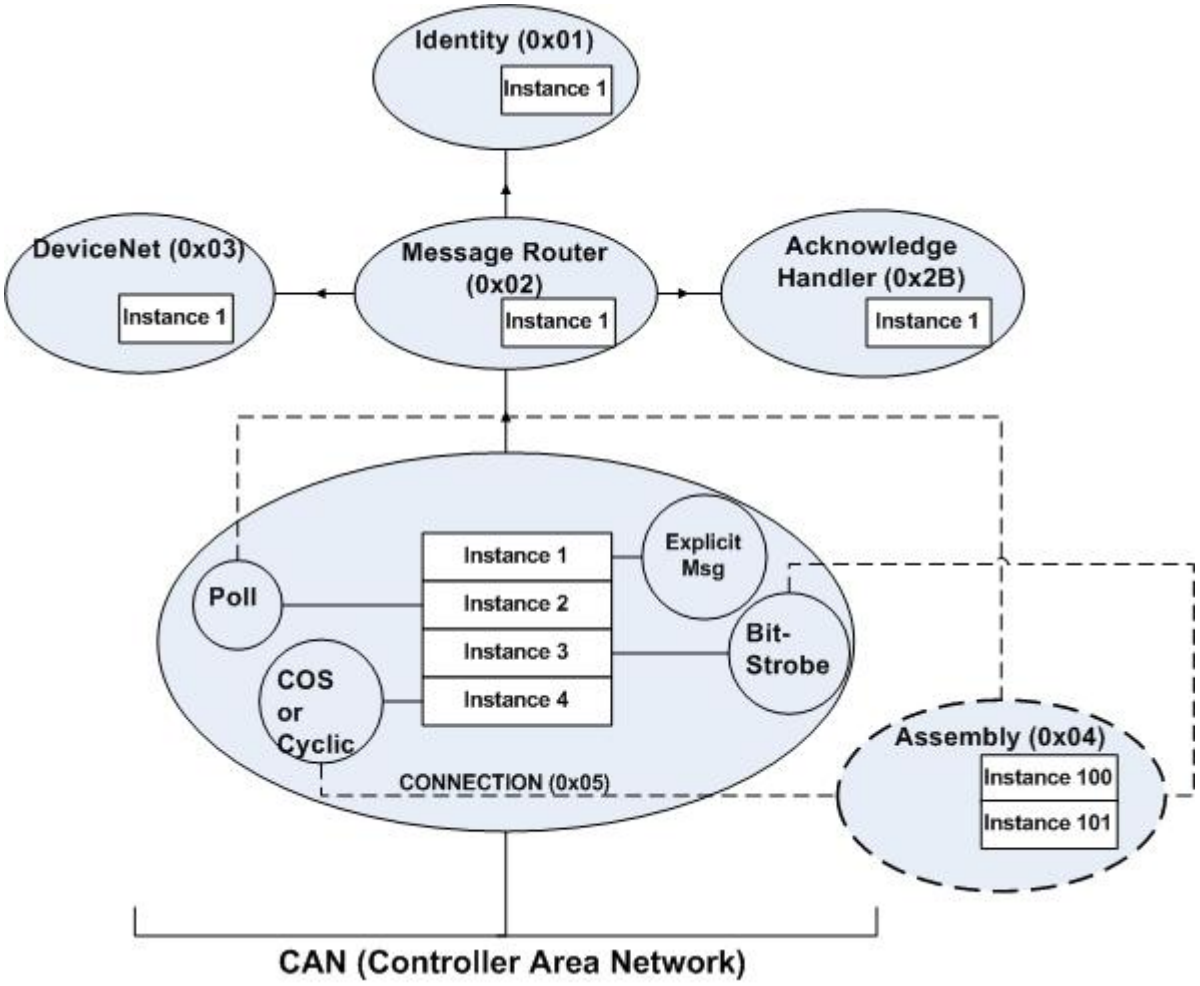


Figure 4: Objects Model of Hilscher DeviceNet Slave stack

For the general description of the Object Model of DeviceNet, see reference [4], Section 1-5 Device Object Model.

3.2.1 Identity Object (Class Code: 0x01)

The Identity Object identifies the device and provides general information about it. The first instance identifies the entire device. It is used for electronic keying and by applications requiring information about the nodes on the network.

Supported features

Instance	Name	Attribute ID	Name	Service supported	Services forwarded to the application
0	Class	-	-	none	none
1	Instance Attributes	1	Vendor ID	Get Attribute Single Get Attribute All Reset	supported
		2	Device Type		
		3	Product Code		
		4	Major Revision		
			Minor Revision		
		5	Status		
		6	Serial Number		
		7	Product Name		
		8 ... 255	-		
2 ... 255	non-existent	non-existent	non-existent	non-existent	non-existent

Table 6: Identity Object supported features

3.2.2 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.

There are no services supported by the Message Router Object.

3.2.3 DeviceNet Object (Class Code: 0x03)

The DeviceNet Object contains information about the configured DeviceNet Slave. Examples of this information include MAC ID, baudrate, etc. as listed in Table 7.

Supported features

Instance	Name	Attribute ID	Name	Service supported	Services forwarded to the application
0	Class	1	Revision	Get Attribute Single	supported
1	Instance Attributes	1	MAC ID	Set Attribute Single Get Attribute Single	
		2	Baudrate	Get Attribute Single	
		3	Bus-off Interrupt	Set Attribute Single Get Attribute Single	
		4	Bus-off Counter	Set Attribute Single Get Attribute Single	
		5	Object Allocation Information	Get Attribute Single	
		6	MAC ID Switch Changed	Get Attribute Single	
		7	Baud Rate Switch Changed	Get Attribute Single	
		8	MAC ID Switch Value	Get Attribute Single	
		9	Baud Rate Switch Value	Get Attribute Single	
		10 ... 255	non-existent	non-existent	
2..255	non-existent	non-existent	non-existent	non-existent	

Table 7: DeviceNet Object supported features

- The attributes 6 and 8 are only present if the MAC ID has been set via a switch.
- The attributes 7 and 9 are only present if the baud rate has been set via a switch.

3.2.4 Assembly Object (Class Code 0x04)

By default, the Assembly objects in Hilscher DeviceNet slave stack are created dynamically. This means, the Assembly objects come only to into existence when one of the cyclic connections is established (poll, change of state/cyclic, bitstrobe). The default instance of the Assembly object is 100 (0x64) for producing data (output) and 101 (0x65) for consuming data (input).

In case, no cyclic connection is configured, all request to Assembly instances will be return with error code 0x16 (object does not exist).

Table 8 lists the default supported attributes and services.

Supported features

Instance	Name	Attribute ID	Name	Service supported	Services forwarded to the application
0	Class	-	-	none	supported
100, 101	Instance Attributes	1	non-existent	none	
		2	non-existent	none	
		3	Data	Get Attribute Single Set Attribute Single	
		4 … 255	non-existent	none	
others	non-existent	non-existent	non-existent	none	

Table 8: Assembly Object supported features

3.2.5 Connection Object (Class Code: 0x05)

The Connection Object consists of multiple objects which provide information about the current connection status. Each instance relates to a different connection type. For example:

- Explicit Messaging connection,
- Bit Strobe,
- Polled,
- Change of State or
- Cyclic.

Supported features

Instance	Name	Attribute ID	Name	Service supported	Services forwarded to the application
0	Class	1	Revision	none	supported
		2 ... 255	-	none	supported
1	Instance Attributes Explicit Messaging Connection	1	Connection State	Get Attribute Single Set Attribute Single Reset	none (all attributes)
		2	Connection Type		
		3	Transport Type		
		4	Produced Connection ID		
		5	Consumed Connection ID		
		6	Initial Com Characteristics		
		7	Produced Connection Size		
		8	Consumed Connection Size		
		9	Expected Packet Rate		
		12	Timeout Action		
		13	Produced Path Length		
		14	Produced Connection Path ID		
		15	Consumed Path Length		
		16	Consumed Connection Path ID		
		17	Inhibit Time		
2	Polled Connection	1 – 17	same as attributes of instance 1		none (all attributes)
3	Bit Strobe Connection	1 – 17	same as attributes of instance 1		none (all attributes)
4	Change of State or Cyclic Connection	1 – 17	same as attributes of instance 1		none (all attributes)
5 ... 255	non-existent	non-existent	non-existent	non-existent	non-existent

Table 9: Connection Object supported features

3.2.6 Acknowledge Handler Object (Class Code: 0x2B)

The Acknowledge Handler Object is responsible for handling acknowledge response messages from the slave. This object supports both Get and Set Attribute Single. Table 10 lists supported attributes.

Supported features

Instance	Name	Attribute ID	Name	Service supported	Services forwarded to the application
0	Class	-	-	none	none
1	Instance Attributes	1	Acknowledge Timer	Get Attribute Single Set Attribute Single	
		2	Acknowledge Handler Retry Limit	Get Attribute Single	
		3	COS Produced ID	Get Attribute Single	
		4 ... 255	non-existent	non-existent	
2..255	non-existent	non-existent	non-existent	non-existent	

Table 10: Acknowledge Handler Object supported features

4 Application interface

This chapter defines the user application interface of the DeviceNet-Slave Stack. There are two different levels the user can access the DeviceNet-Slave stack.

- The Communication Channel Interface via Dual port Memory if Loadable Firmware used
- The DeviceNet Slave - Task Interface if Linkable Object module are used

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. This Interface is also available, when a user develops its own application within the netX system and works with the virtual DPM system intern.

The second level is when a user implements its own application task direct over the DevNetFAL Task. Then a user has to care the configuration process, mapping I/O data and status information by its own. Typically when a user works with its own dual port memory layout, the application itself has to be developed.

4.1 Configuration services

4.1.1 Configuration sequence (Loadable Firmware)

To configure the DeviceNet Slave stack the following packets are necessary:

Packet name	Command code (REQ/CNF)	Page
Set Configuration Service, DNS_AP_CMD_SET_CONFIGURATION_REQ/CNF Set the configuration parameters.	0x4100 0x4101	23
Register Application Service, RCX_REGISTER_APP_REQ/CNF Register Application to the stack in order to receive indications. For a description of this service, see reference [2].	0x2F10 0x2F11	-
Channel Initialization Service, RCX_CHANNEL_INIT_REQ/CNF Perform a Channel Initialization. For a description of this service, see reference [2].	0x2F80 0x2F81	-

Table 11: Configuration sequence (Loadable Firmware)

Figure 5 shows the packet sequence the application has to use. After the application has received the confirmation of a packet, the application then can send the next request.

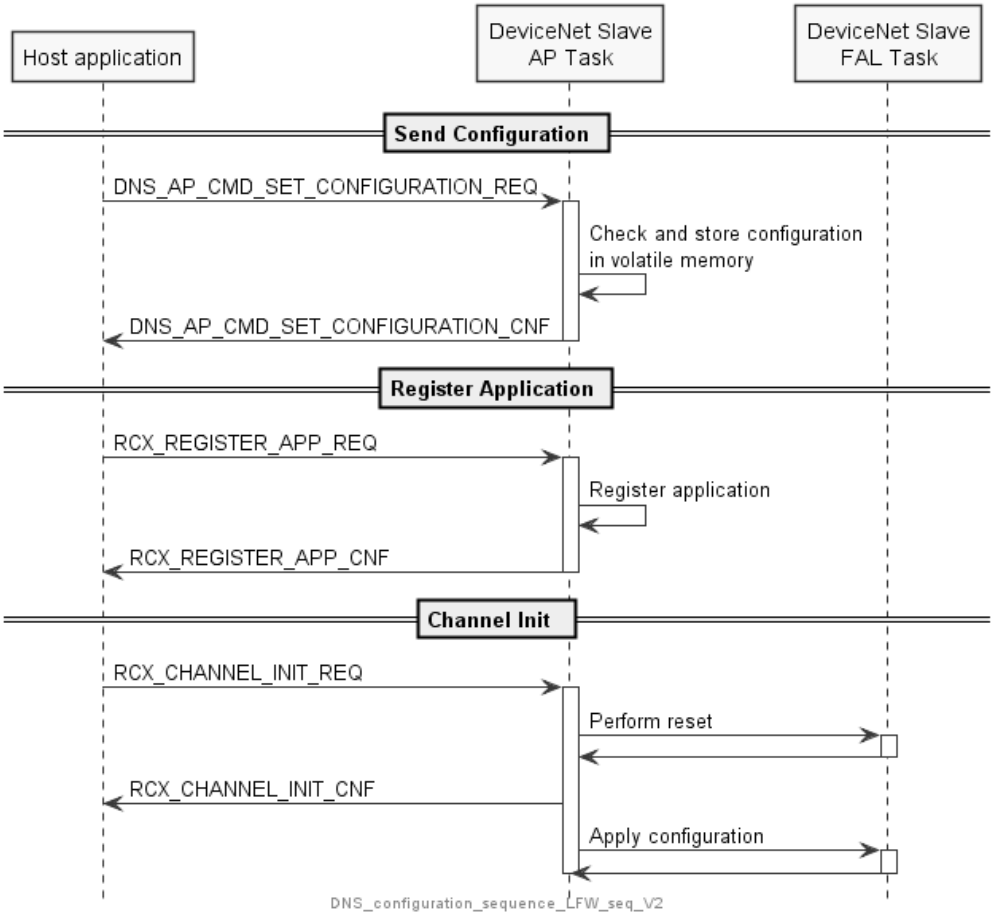


Figure 5: Configuration sequence (Loadable Firmware)

The following rules apply for the behavior of the DeviceNet Slave protocol stack when receiving a set configuration command:

- The configuration data are checked for consistency and integrity.
- In case of failure no data are accepted.
- In case of success the configuration parameters are stored internally (within the RAM).
- The parameterized data will be activated only after a channel init has been performed.
- No automatic registration of the application at the stack happens.
- The confirmation packet `DNS_AP_CMD_SET_CONFIGURATION_CNF` only transfers simple status information, but does not repeat the whole parameter set.

4.1.2 Set Configuration Service

Once the DeviceNet Slave Stack is started, it must be initialized with the appropriate bus-related parameters. The packet `DNS_AP_CMD_SET_CONFIGURATION_REQ` can be used to configure the DeviceNet Slave stack. The AP Task checks the parameters which are send with this packet. If all parameters are valid the AP Task will hold the configuration. To activate the parameters and become effective to network a “Channel Init” must be performed. This can be done by sending the corresponding packet `RCX_CHANNEL_INIT_REQ` or by calling the cifX Driver function `xChannelInit()`.

The bus parameters include

- Bus Startup Mode (Automatic or application-controlled)
- Watchdog Time [ms]
- Baudrate
- MAC ID
- ProducedSize
- ConsumedSize
- Configuration Flags
- Enable Flags
- Vendor ID
- Revision Information
- ProductType
- ProductCode
- Serial number
- Product name and its length

The following applies for this packet:

- Configuration parameters will be stored internally.
- In case of any error no data will be stored at all.
- **A channel init is required to activate the parameterized data**.
- This packet does not perform any registration at the stack automatically. Registering must be performed with a separate packet such as the registration packet described in the netX Dual-Port-Memory Manual (`RCX_REGISTER_APP_REQ`, code `0x2F10`).
- This request will be denied if the configuration lock flag is set (for more information on this topic see reference [1].

4.1.2.1 Set Configuration Request

The application has to send this request to the protocol stack.

Note: The command DNS_AP_CMD_SET_CONFIGURATION_REQ is mapped to the command DNS_FAL_CMD_INIT_STACK_REQ. The differences are in the command number and the behavior when the parameters become effective to the network. The parameters which are sent with DNS_FAL_CMD_INIT_STACK_REQ become effective immediately. The parameters set with the command DNS_AP_CMD_SET_CONFIGURATION_REQ become effective after a following “Channel Init”.

Packet description

Variable	Type	Value / Range	Description
ulDest	UINT32	LFW: 0x00000020, LOM: DNS FAL	LFW: AP task, LOM: DNS FAL Task
ulLen	UINT32	80	sizeof(DNS_FAL_INIT_STACK_REQ_T)
ulCmd	UINT32	0x4100	DNS_AP_CMD_SET_CONFIGURATION_REQ
Data			
ulSystemFlags	UINT32	bitlist	System Flags For a description, see Table 13 on page 27.
ulWdgTime	UINT32	0, 20-65535	Watchdog time within which the device watchdog must be retriggered from the application program while the application program monitoring is activated. When the watchdog time value is equal to 0 respectively the application program monitoring is deactivated. The watchdog time parameter specifies the time in multiples of 1msec. the device has to supervise the host program if it has started the host-watchdog functionality once. Read in manual 'Toolkit General Definitions' how to activate and deactivate the device and host supervision.
ulNodeId	UINT32	0-63	Node id MAC ID of the DeviceNet Slave in the network.
ulBaudrate	UINT32	0 - 2	Baud rate For possible baud rates and values, see Table 16 on page 30.
ulProducedSize	UINT32	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.
ulConsumedSize	UINT32	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.
ulConfigFlags	UINT32	0-31	Configuration flags For a description, see Table 14 on page 29.
ulEnableFlags	UINT32		Enable flags Indicates to the DNS task if the following identity variables usProductType, usProductCode, bMinorRev, bMajorRev and the ProdName are filled with user application specific data. For coding see *Table 15: EnableFlags* 0 denotes: Use settings from stack. For a description, see Table 15 on page 29.

Variable	Type	Value / Range	Description
usVendorId	UINT16	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.
usProductType	UINT16	1-65535 (default 12)	Identification of the general type of product. The Hilscher standard value for this is 12 which is a Communications Adapter.
usProductCode	UINT16	1-…	Identification of a particular product within a defined device type.
bMinorRev	UINT8	1-255 (default 1)	First part of the revision which identifies 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	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	0x00000000 - 0xFFFFFFFF	Unique serial number of the device defined by user.
abReserved[3]	UINT8		Reserved
bProductNameLen	UINT8	1-32	Length of abProdName string. The maximum number of characters in this string is 32.
abProductName[32]	UINT8[]	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.

Table 12: DNS_AP_CMD_SET_CONFIGURATION_REQ_T – Set Configuration Request

Packet structure reference

```
/*****************************************************************************/
/*      System flags with command DNS_AP_CMD_SET_CONFIGURATION _REQ          */
/*****************************************************************************/
#define MSK_DNS_SYS_MANUAL_START                                    0x00000001
#define MSK_DNS_SYS_FLG_ADR_SW_ENABLE                               0x00000010
#define MSK_DNS_SYS_FLG_BAUD_SW_ENABLE                              0x00000020
#define MSK_DNS_SYS_FLG_RESERVED                                    0xFFFFFFCE

/*****************************************************************************/
/*                              DNS Baudrates                                */
/*****************************************************************************/
#define   DNS_FAL_BAUDRATE_500kB                                     0
#define   DNS_FAL_BAUDRATE_250kB                                     1
#define   DNS_FAL_BAUDRATE_125kB                                     2

/*****************************************************************************/
/*        Enable flags in with command DNS_AP_CMD_SET_CONFIGURATION_REQ      */
/*****************************************************************************/
#define MSK_DNS_ENABLE_VENDORID                                     0x00000001
#define MSK_DNS_ENABLE_PRODUCTTYPE                                  0x00000002
#define MSK_DNS_ENABLE_PRODUCTCODE                                  0x00000004
#define MSK_DNS_ENABLE_MAJORMINORREV                                0x00000008
#define MSK_DNS_ENABLE_SERIALNR                                     0x00000010
#define MSK_DNS_ENABLE_PRODUCTNAME                                  0x00000020
#define MSK_DNS_ENABLE_RESERVED                                     0xFFFFFFC0

/*****************************************************************************/
/*      Configuration flags with command DNS_AP_CMD_SET_CONFIGURATION_REQ    */
/*****************************************************************************/
#define MSK_DNS_CFG_FLAG_IGNORE_ADDR_SWITCH                         0x00000001
#define MSK_DNS_CFG_FLAG_CONTINUE_ON_BUSOFF                         0x00000002
#define MSK_DNS_CFG_FLAG_CONTINUE_ON_LOSS_NP                        0x00000004
#define MSK_DNS_CFG_FLAG_RECVIDLE_CLEAR_DATA                        0x00000008
#define MSK_DNS_CFG_FLAG_RECVIDLE_USER_DATA                         0x00000010
```

```
#define MSK_DNS_CFG_FLAG_24VDCINVERT                                0x00000020
#define MSK_DNS_CFG_FLAG_ENABLE_SET_PRODCONS_SIZE_REMOTE            0x00000040
#define MSK_DNS_CFG_FLAG_ENABLE_SET_MACID_REMOTE                    0x00000080
#define MSK_DNS_CFG_FLAG_ENABLE_SET_BAUDRATE_REMOTE                 0x00000100
#define MSK_DNS_CFG_FLAG_ENABLE_DATA_STATUS                         0x00000200
#define MSK_DNS_CFG_FLAG_DISABLE_GET_SET_ATT_IND                    0x00000400

#define MSK_DNS_CFG_FLAG_RESERVED                                   0xFFFFFC00


typedef struct DNS_FAL_INIT_STACK_REQ_Ttag {
  TLR_UINT32 ulSystemFlags;
  TLR_UINT32 ulWdgTime;
  TLR_UINT32 ulNodeId;
  TLR_UINT32 ulBaudrate;
  TLR_UINT32 ulProducedSize;
  TLR_UINT32 ulConsumedSize;
  TLR_UINT32 ulConfigFlags;
  TLR_UINT32 ulEnableFlags;
  TLR_UINT16 usVendorId;
  TLR_UINT16 usProductType;
  TLR_UINT16 usProductCode;
  TLR_UINT8  bMinorRev;
  TLR_UINT8  bMajorRev;
  TLR_UINT32 ulSerialNumber;
  TLR_UINT8  abReserved[3];
  TLR_UINT8  bProductNameLen;
  TLR_UINT8  abProductName[32];
} DNS_FAL_INIT_STACK_REQ_T;

typedef struct DNS_AP_PACKET_SET_CONFIGURATION _REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_INIT_STACK_REQ_T tData;
} DNS_AP_PACKET_SET_CONFIGURATION_REQ_T;
```

SystemFlags parameter

The SystemFlags defines the startup behavior of the stack. It also defines if 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 address switch is activated. This flag is used with packet Address Switch Enable Indication (page 54).
5	Baudrate Switch Enable, MSK_DNS_SYS_FLG_BAUD_SW_ENABLE: If this bit is set, the handling mechanism of baudrate switch is activated. This flag is used with packet Address Switch Enable Indication (page 54).
Others	Reserved.

Table 13: SystemFlags parameter

ConfigFlags parameter

Table 14 list the meaning of the ConfigFlags parameter. If a bit is set to 1, the function will be activated, otherwise deactivated.

Bit	Definition / Description
0	Ignore Address Switch, MSK_DNS_CFG_FLAG_IGNORE_ADDR_SWITCH: This flag indicates whether the address switch is enabled. **Note:** this flag not used anymore since firmware version 2.3.8, instead flag MSK_DNS_SYS_FLG_ADR_SW_ENABLE is used exclusively If set to 1, the DeviceNet MAC-ID provided by the hardware address switches will be ignored. Instead the MAC-ID will be read from the bOwnMacId configuration variable after a warmstart. This applies to devices which feature hardware address switches, only. All other devices will ignore the MSK_DNS_CFG_FLAG_IGNORE_ADDR_SWITCH flag. The MAC address of the device will be evaluated from hardware address switches if this flag set to 0.
1	Continue on Bus-off, MSK_DNS_CFG_FLAG_CONTINUE_ON_BUSOFF: This flag defines the behavior of the device in case a 'bus off' indication from the CAN controller occurs. 0: device will stop operation 1: device will continue its operation
2	Continue on loss of NP, MSK_DNS_CFG_FLAG_CONTINUE_ON_LOSS_NP: Not supported.
3	Clear data on receive idle, MSK_DNS_CFG_FLAG_RECVIDLE_CLEAR_DATA: This flag defines the behavior of the device in case the master sends receive_idle telegrams in an established poll connection. Each master has the possibility to send no input data to the slave as a special poll command. The slave on the other hand is still forced to send back its latest output data and keep the master updated. This is procedure defined in the DeviceNet specification. 0: device will hold the last received input data in the input process image 1: device will clear the input area in the process image with the value 0.
4	User data on receive idle, MSK_DNS_CFG_FLAG_RECVIDLE_USER_DATA: Not supported
5	Inverted handling of Network Power, MSK_DNS_CFG_FLAG_24VDCINVERT: Handles the 24V Network Signal inverted, if the user hardware does not follow the standard (inverted 24V network power supply) or have hardware reference of Hilscher old and obsolete NXSB-100 board.
6	Enable Remote Setting of Producer Size and Consumer Size: Via Remote Service, the DeviceNet Master may request a change of the Producer Size or Consumer Size. 0: disable changes. If the flag is not set and if connection class is not registered by host application (host task) then the Set_Attribute_Single request from client will be answered with general code: service not supported (0x08). When the flag is not set and if connection class is already registered by host application task (host task) using DNS_FAL_CMD_REGISTER_CLASS_REQ then the Set_Attribute_Single request from client will be forwarded to host application task (host task). 1: enable changes. If the flag is set, then the Set_Attribute_Single request from client to attributes 7 and 8 (produced connection size and consumed connection size) of connection object will be automatically handled by stack application task.
7	Enable Remote Setting of MAC ID: Via Remote Service, the DeviceNet Master may request a change of the MAC ID. This flag allows to enable (flag set) or disable (flag cleared) such changes. When the flag is set, then Set_Attribute_Single request from client to attributes 1 (MAC ID) of DeviceNet object will be automatically handled by stack application task. When the flag is not set and if DeviceNet class is not registered by host application (host task) then the Set_Attribute_Single request from client will be answered with general code: service not supported (0x08). When the flag is not set and if DeviceNet class is already registered by host application (host task) using DNS_FAL_CMD_REGISTER_CLASS_REQ then the Set_Attribute_Single request from client will be forward to host application (host task).

8	Enable Remote Setting of Baudrate: Via Remote Service, the DeviceNet Master may request a change of the Baudrate. This flag allows to enable (flag set) or disable (flag cleared) such changes. When the flag is set, then Set_Attribute_Single request from client to attributes 2 (baudrate) of DeviceNet object will be automatically handled by stack application task. When the flag is not set and if DeviceNet class is not registered by host application (host task) then the Set_Attribute_Single request from client will be answered with general code: service not supported (0x08). When the flag is not set and if DeviceNet class is already registered by second-level application task (host task) using DNS_FAL_CMD_REGISTER_CLASS_REQ then the Set_Attribute_Single request from client will be forward to host application (host task).
9	Enable Data Status, MSK_DNS_CFG_FLAG_ENABLE_DATA_STATUS: Via Remote Service, the DeviceNet Master may request a change of the data status. This flag allows to enable (flag set) or disable (flag cleared) these changes. For details, see section *Get Output Image Service* on page 39.
10	Get / Set Attribute Single service handled with Remote Service, MSK_DNS_CFG_FLAG_DISABLE_GET_SET_ATT_IND: 0: The user receives indications ▪ for Get / Set Attribute Single from the *Get / Set Attribute Service* (page 78) and ▪ for all other services from the *Remote Service* (page 75). 1: The user receives indications for all services (including Get / Set Attribute Single) from the *Remote Service* (page 75).
11 … 31	Reserved, set to 0.

Table 14: ConfigFlags parameter

EnableFlags parameter

The EnableFlags byte defines whether the following identity variables ProductType, ProductCode, MinorRev, MajorRev and the ProductName are filled up with valid data, or not. The following table shows the meaning of the EnableFlags byte. If the corresponding bit is set to 1, the variable contains valid data and is enabled, i.e. the appropriate item (Vendor ID, product type, product code, minor and major revision together, serial number and product name) can be set by the user, and otherwise the corresponding parameters will be filled with default values from the stack.

Bit	Description
0	Vendor ID, MSK_DNS_ENABLE_VENDORID: Vendor Identification of the manufacturer according to ODVA. See Licensed Vendor List of ODVA.
1	ProductType, MSK_DNS_ENABLE_PRODUCTTYPE: The variable ProductType is the indication of the general type of product. The Hilscher default value for this is 12 which is a Communications Adapter
2	ProductCode, MSK_DNS_ENABLE_PRODUCTCODE: The variable ProductCode is the identification of a particular product within a device type.
3	Major Minor Revision, MSK_DNS_ENABLE_MAJORMINORREV: The variable MinorRev is one part of the revision which identifies the revision of the DEVICE. The revision attribute consists of Major and Minor Revisions and they are typically displayed as major.minor. The variable MajorRev is the 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.
4	Serial Number, MSK_DNS_ENABLE_SERIALNR: Unique serial number of the device that user want to set. The serial number should be unique for each device of the manufacturer.
5	ProductName, MSK_DNS_ENABLE_PRODUCTNAME: The variable ProductName is a 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 bLength which is the first byte in the structure tProduct.

Table 15: EnableFlags parameter

Table 16 lists the possible baud rates.

Baud rate	Value
500 kBit/s	0
250 kBit/s	1
125 kBit/s	2

Table 16: Baud rates and values

Source Code Example

```
TLR_RESULT DnsApp_SetConfig_Req(DNS_APP_RSC_T FAR* ptRsc)
{
  TLR_RESULT eRslt;
  DNS_AP_PACKET_SET_CONFIGURATION_REQ_T  *ptInitStackReq;

  eRslt = TLR_POOL_PACKET_GET(ptRsc->tLoc.hPool, &ptInitStackReq);
  if( eRslt == TLR_S_OK )
  {
    TLR_QUE_LINK_SET_PACKET_SRC(ptInitStackReq,ptRsc->tLoc.tQueSrcDnsApp);
    ptInitStackReq->tHead.ulCmd = DNS_AP_CMD_SET_CONFIGURATION_REQ;
    ptInitStackReq->tHead.ulLen = sizeof(DNS_FAL_SET_CONFIGURATION_REQ_T);

     /* Set the slave related parameters here */
     ptInitStackReq->tData.ulSystemFlags = 0;
     ptInitStackReq->tData.ulWdgTime = 0;
     ptInitStackReq->tData.ulNodeId = 11;
     ptInitStackReq->tData.ulBaudrate = DNS_FAL_BAUDRATE_125kB;
     ptInitStackReq->tData.ulProducedSize = 8;
     ptInitStackReq->tData.ulConsumedSize = 8;
     ptInitStackReq->tData.ulConfigFlags = 0;

     ptInitStackReq->tData.ulEnableFlags = MSK_DNS_ENABLE_VENDORID |
                                           MSK_DNS_ENABLE_PRODUCTTYPE |
                                           MSK_DNS_ENABLE_PRODUCTCODE |
                                           MSK_DNS_ENABLE_PRODUCTNAME |
                                           MSK_DNS_ENABLE_SERIALNR;

     ptInitStackReq->tData.usVendorId = 283;   /* My VendorId    */
     ptInitStackReq->tData.usProductType = 12; /* My Producttype */
     ptInitStackReq->tData.usProductCode = 1;  /* My Productcode */
     ptInitStackReq->tData.bMajorRev = 0;      /* Not enabled use stack default */
     ptInitStackReq->tData.bMinorRev = 0;      /* Not enabled use stack default */
     TLR_MEMCPY(&ptInitStackReq->tData.abProductName[0],"My netX DNS",11);
     ptInitStackReq->tData.bProductNameLen = 11;
     ptInitStackReq->tData.ulSerialnumber = 0x11223344L;

    eRslt = TLR_QUE_SENDPACKET_FIFO(ptRsc >tLoc.tDnsQue,ptInitStackReq,TLR_FINITE);
    if( eRslt != TLR_S_OK ){
      TLR_POOL_PACKET_RELEASE(ptRsc->tLoc.hPool, ptInitStackReq);}}
    return eRslt;}
```

4.1.2.2 Set Configuration Confirmation

Packet description

Variable	Type	Value / Range	Description
ulLen	UINT32	8	
ulSta	UINT32		ulSta = 0 Initialization OK ulSta != 0 Initialization failed See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x4101	DNS_AP_CMD_SET_CONFIGURATION_CNF
Data			
hPdOutTrpBuf	TLR_HANDLE		Handle for Triple Buffer for Output. For use case Linkable Object Module only.
hPdInTrpBuf	TLR_HANDLE		Handle for Triple Buffer for Input For use case Linkable Object Module only.

Table 17: DNS_AP_CMD_SET_CONFIGURATION_CNF_T – Confirmation of DNS Stack Initialization

Packet structure reference

```
typedef struct DNS_FAL_INIT_STACK_CNF_Ttag {
  TLR_HANDLE hPdOutTrpBuf;
  TLR_HANDLE hPdInTrpBuf;
} DNS_FAL_INIT_STACK_CNF_T;

typedef struct DNS_FAL_PACKET_INIT_STACK_CNF_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_INIT_STACK_CNF_T tData;
} DNS_FAL_PACKET_INIT_STACK_CNF_T;
```

4.1.3 Clear Configuration Service

This service clears the configuration data stored in the device.

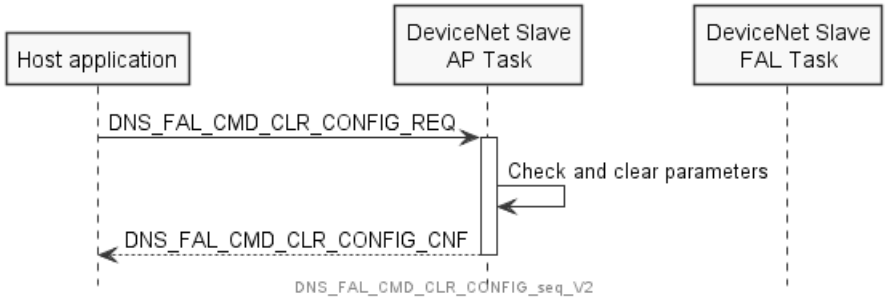


Figure 6: DNS_FAL_CMD_CLR_CONFIG_REQ/CNF sequence diagram

4.1.3.1 Clear Configuration Request

The application has to send this request to the protocol stack.

Packet description

Variable	Type	Value / Range	Description
ulDest	UINT32	LFW: 0x00000020	LFW: AP task
ulLen	UINT32	4	Packet Data Length in bytes
ulCmd	UINT32	0x2D08	DNS_FAL_CMD_CLR_CONFIG_REQ
Data			
ulCfgArea	UINT32	0	0 = Clear all configuration data

Table 18: DNS_FAL_CMD_CLR_CONFIG_REQ_T - Clear Configuration Request

Packet structure reference

```
typedef struct DNS_FAL_SDU_CLR_CONFIG_REQ_Ttag
{
  TLR_UINT32 ulCfgArea;
}DNS_FAL_SDU_CLR_CONFIG_REQ_T;

typedef struct DNS_FAL_PACKET_CLR_CONFIG_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_SDU_CLR_CONFIG_REQ_T tData;
} DNS_FAL_PACKET_CLR_CONFIG_REQ_T;
```

4.1.3.2 Clear Configuration Confirmation

Packet description

Variable	Type	Value / Range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D09	DNS_FAL_CMD_CLR_CONFIG_CNF

Table 19: DNS_FAL_CMD_CLR_CONFIG_CNF_T – Confirmation of Clear Configuration Request

Packet structure reference

```
typedef _struct DNS_FAL_PACKET_CLR_CONFIG_CNF_Ttag {
  TLR_PACKET_HEADER_T tHead;
} DNS_FAL_PACKET_CLR_CONFIG_CNF_T;
```

4.2 Control / Monitor the Stack

4.2.1 Get LED State Service

The LED state can be requested by the host application using DNS_AP_CMD_GET_LED_STATE_REQ packet. Please note, that the LED state at the start-up LED self test sequence cannot be covered with this command. The LED blink sequence is described as following according to the [4]. A LED test is to be performed at power up:

- Turn combined Module/Network Status LED on Green for approximately 0.25 seconds.
- Turn combined Module/Network LED on Red for approximately 0.25 seconds.
- Turn combined Module/Network LED off."

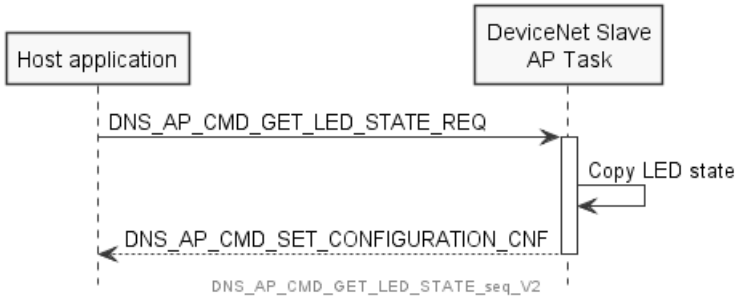


Figure 7: DNS_AP_CMD_GET_LED_STATE_REQ/CNF sequence diagram

The type of the affected LED is indicated by parameter `ulLedType` in the following way:

Value of ulLedType	Meaning
1	MNS LED

Table 20: Meaning of ulLedType

The Device Net Slave Stack can provide only the LED information according to the network access state. The network access state is represented by the Network Status LED (MNS LED). The host application is responsible to combine this information with the Module Status information and control the connected LED's according the "Chapter 9-2 Indicators" of the reference [4].

The mode of the affected LED is indicated by parameter `ulLedMode` according to this table:

Value of ulLedMode	Meaning
1	Static
2	Flash

Table 21: Meaning of ulLedMode

The color of the affected LED is indicated by parameter `ulLedColor` as follows:

Value of `ulLedColor`	Meaning
1	Off
2	Green
3	Red

Table 22: Meaning of ulLedColor

4.2.1.1 Get LED State Request

The application has to send this request to the protocol stack.

Packet description

Variable	Type	Value / Range	Description
ulDest	UINT32	LFW: 0x00000020, LOM: DNS FAL	LFW: AP task, LOM: DNS FAL Task
ulLen	UINT32	4	Packet Data Length in bytes
ulCmd	UINT32	0x4104	DNS_AP_CMD_GET_LED_STATE_REQ
Data			
ulLedType	UINT32	1	LED type (MNS = 1)

Table 23: DNS_AP_CMD_GET_LED_STATE_REQ_T – LED State Request

Packet structure reference

```
/*****************************************************************************/
/*                      DNS_AP_CMD_GET_LED_STATE_REQ Structure                */
/*****************************************************************************/
typedef __PACKED_PRE struct __PACKED_POST DNS_APP_GET_LED_STATE_REQ_DATA_Ttag
{
  TLR_UINT32 ulLedType;                                              /* MNS */
} DNS_APP_GET_LED_STATE_REQ_DATA_T;

#define DNS_APP_GET_LED_STATE_REQ_SIZE  (sizeof(DNS_APP_GET_LED_STATE_REQ_DATA_T))

typedef __PACKED_PRE struct __PACKED_POST DNS_APP_PACKET_GET_LED_STATE_REQ_Ttag
{
  TLR_PACKET_HEADER_T              tHead;
  DNS_APP_GET_LED_STATE_REQ_DATA_T tData;
} DNS_APP_PACKET_GET_LED_STATE_REQ_T;
```

4.2.1.2 Get LED State Confirmation

Packet description

Variable	Type	Value / Range	Description
ulLen	UINT32	12	Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x4105	DNS_AP_CMD_GET_LED_STATE_CNF
Data			
ulLedType	UINT32	1	LED type 1: MNS
ulLedMode	UINT32	1 … 2	LED mode 1: static 2: flash
ulLedColor	UINT32	1 … 3	Color of LED 1: off 2: green 3: red

Table 24: DNS_AP_CMD_GET_LED_STATE_CNF_T – LED State Confirmation

Packet structure reference

```
/*****************************************************************************/
/*                        DNS_AP_CMD_GET_LED_STATE_CNF Structure             */
/*****************************************************************************/
typedef __PACKED_PRE struct __PACKED_POST DNS_APP_GET_LED_STATE_CNF_DATA_Ttag
{
  TLR_UINT32 ulLedType;  /* MNS = 1, DNS_FAL_LED_TYPE_..                     */
  TLR_UINT32 ulLedMode;  /* STATIC = 1, FLASH = 2, DNS_FAL_LED_MODE_..       */
  TLR_UINT32 ulLedColor; /* RED = 3, GRN = 2, OFF= 1, DNS_FAL_LED_COLOR_ ..  */
} DNS_APP_GET_LED_STATE_CNF_DATA_T;


typedef __PACKED_PRE struct __PACKED_POST DNS_FAL_PACKET_GET_LED_STATE_CNF_Ttag
{
  TLR_PACKET_HEADER_T              tHead;
  DNS_APP_GET_LED_STATE_CNF_DATA_T tData;

}DNS_APP_PACKET_GET_LED_STATE_CNF_T;

#define DNS_APP_GET_LED_STATE_CNF_SIZE  (sizeof(DNS_APP_GET_LED_STATE_CNF_DATA_T))
```

4.3 Handle Input / Output Data Image

4.3.1 Set Input Image Service

The command `DNS_FAL_CMD_SET_INPUT_REQ` is used by the user application to update user input data to the DNS task. The input information will appear as input values to the DeviceNet Master. This command provides other mean of setting slave‘s output except the manipulation of output area in the DPM.

The count of the data to be transferred is specified by the value of `ulInLen`. The maximum permitted length is 255 bytes.

The offset address is specified in the parameter `ulInOffSet`. The specified address is interpreted from the device as the relative address to the start address in the send process data or the receive process data. The maximum value is decimal 255 for byte access.

The input data from the DeviceNet Master corresponds to the input data image in the DNS task. It must be specified in the `abInData[]` field of the request packet.

An explanation of how this command is used is as follows.

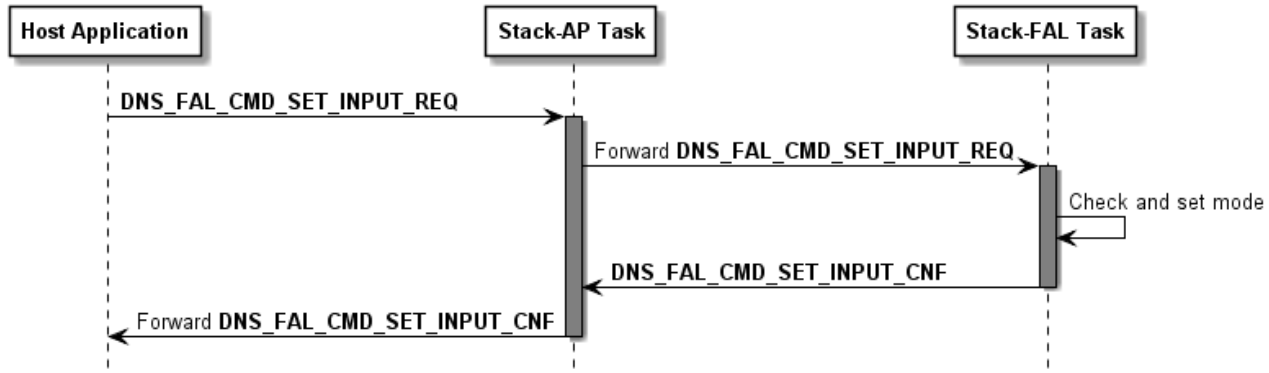


Figure 8: Sequence Diagram for the DNS_FAL_CMD_SET_INPUT_REQ/CNF Packet

4.3.1.1 Set Input Image Request

The application has to send this request to the protocol stack.

Packet Structure Reference

```
typedef struct DNS_FAL_SET_INPUT_REQ_Ttag {
  TLR_UINT32   ulInOffSet;
  TLR_UINT32   ulInLen;
  TLR_UINT32   ulInDataSta;
  TLR_UINT8    abInData[256];
} DNS_FAL_SET_INPUT_REQ_T;

typedef struct DNS_FAL_PACKET_SET_INPUT_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
   DNS_FAL_SET_INPUT_REQ_T tData;
} DNS_FAL_PACKET_SET_INPUT_REQ_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32	LFW: 0x00000020, LOM: DNS FAL	LFW: AP task, LOM: DNS FAL Task
ulLen	UINT32	12 + ulInLen	Packet Data Length in bytes
ulCmd	UINT32	0x2D0E	DNS_FAL_CMD_SET_INPUT_REQ - Command
tData			
ulInOffSet	UINT32	0-255	Offset in the producing Data image in which to place the abInData[].
ulInLen	UINT32	0-255	Input length.
ulInDataSta	UINT32	0	Reserved, set to 0
abInData[]	UINT8	0-255	Input data sent to the DNS task. This information will appear to the master as input data.

Table 25: DNS_FAL_CMD_SET_INPUT_REQ – Request Command for Set Input Image Update

4.3.1.2 Set Input Image Confirmation

This confirmation will be returned to the application.

Packet Structure Reference

```
typedef struct DNS_FAL_SET_INPUT_CNF_Ttag {
  TLR_UINT32   ulInOffSet;
  TLR_UINT32   ulInLen;
  TLR_UINT32   ulInDataSta;
} DNS_FAL_SET_INPUT_CNF_T;

typedef struct DNS_FAL_PACKET_SET_INPUT_CNF_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_SET_INPUT_CNF_T tData;
} DNS_FAL_PACKET_SET_INPUT_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	12	Packet Data Length in bytes.
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D0F	DNS_FAL_CMD_SET_INPUT_CNF - Command
tData			
ulInOffSet	UINT32		Offset in the input data image (specified in bytes).
ulInLen	UINT32		Length of input data to be read (specified in bytes).
ulInDataSta	UINT32		Input data status

Table 26: DNS_FAL_CMD_SET_INPUT_CNF – Confirmation of the DNS Set Input Image Update

4.3.2 Get Output Image Service

The command DNS_FAL_CMD_GET_OUTPUT_REQ is used to obtain the output data from the DNS task. This packet provides another way to get output from the master except reading from input area in the DPM.

The count of the data to be transferred is specified by the value of `ulOutLen`. The maximum permitted length is 255 bytes.

The offset address is specified in the parameter `ulOutOffSet`. The specified address is interpreted from the device as the relative address to the start address in the send process data or the receive process data. The maximum value is decimal 255 for byte access.

The output data from the DeviceNet Master is actually the output data image in the DNS task. It will appear in the `abOutData[]` field of the confirmation packet.

The following status information is delivered in the variable `ulOutDataSta` of the confirmation packet:

ulOutDataSta

Symbolic code	Numeric value	Meaning
DNS_FAL_DS_ZERO	0x00000000	Safe state zero
DNS_FAL_DS_RECV_RUN	0x00000001	Data are valid
DNS_FAL_DS_RECV_IDLE	0x00000002	Data are in receive idle state
DNS_FAL_DS_RECV_IDLE_ZERO	0x00000003	Receive idle zero
DNS_FAL_DS_HOLD_LAST_STATE	0x00000004	Hold last state
DNS_FAL_DS_USER_STATE	0x00000005	User defined state

Table 27: Data Status of produced / consumed data

An explanation of how this command is used is as follows.

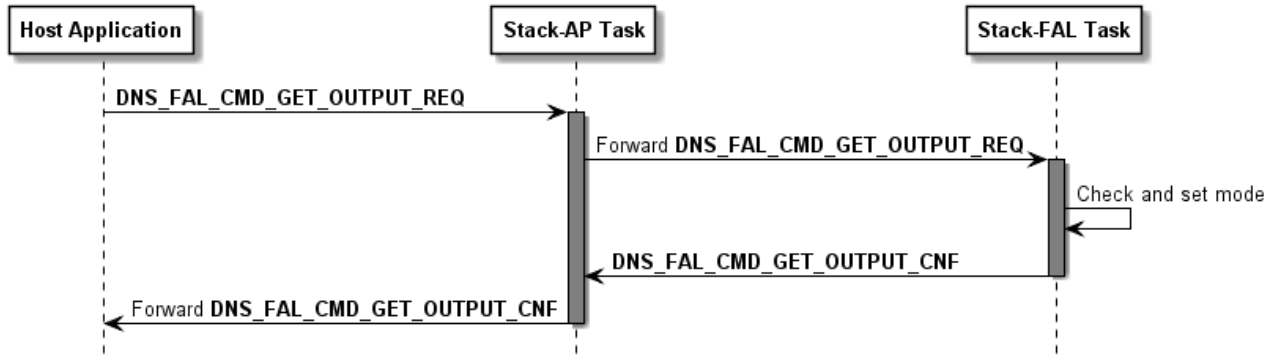


Figure 9: Sequence Diagram for the DNS_FAL_CMD_SET_INPUT_REQ/CNF Packet

4.3.2.1 Get Output Image Request

The application has to send this request to the protocol stack.

Packet Structure Reference

```
typedef struct DNS_FAL_GET_OUTPUT_REQ_Ttag {
  TLR_UINT32   ulOutOffSet;
  TLR_UINT32   ulOutLen;
} DNS_FAL_GET_OUTPUT_REQ_T;

typedef struct DNS_FAL_PACKET_GET_OUTPUT_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
   DNS_FAL_GET_OUTPUT_REQ_T tData;
} DNS_FAL_PACKET_GET_OUTPUT_REQ_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32	LFW: 0x00000020, LOM: DNS FAL	LFW: AP task, LOM: DNS FAL Task
ulLen	UINT32	8	Packet Data Length in bytes
ulCmd	UINT32	0x2D10	DNS_FAL_CMD_GET_OUTPUT_REQ - Command
tData			
ulOutOffSet	UINT32	0-255	Offset in the output Data image.
ulOutLen	UINT32	0-255	Length of output data to be read.

Table 28: DNS_FAL_CMD_GET_OUTPUT_REQ – Request Command for Get Output Image

4.3.2.2 Get Output Image Confirmation

This confirmation will be returned to the application.

Packet Structure Reference

```
typedef struct DNS_FAL_GET_OUTPUT_CNF_Ttag {
  TLR_UINT32  ulOutOffSet;
  TLR_UINT32  ulOutLen;
  TLR_UINT32  ulOutDataSta;
  TLR_UINT8   abOutData[256];
} DNS_FAL_GET_OUTPUT_CNF_T;

typedef struct DNS_FAL_PACKET_GET_OUTPUT_CNF_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_GET_OUTPUT_CNF_T tData;
} DNS_FAL_PACKET_GET_OUTPUT_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	12+ ulOutLen	Packet Data Length in bytes. Length of abInData[]
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D11	DNS_FAL_CMD_GET_OUTPUT_CNF - Command
tData			
ulOutOffSet	UINT32	0-255	Offset in the Input Data image in which to receive the Input data.
ulOutLen	UINT32	0-255	Length of Input data to be read.
ulOutDataSta	UINT32		Data status of produced/consumed data
abOutData[]	UINT8[]		Output data from the DNS task.

Table 29: DNS_FAL_CMD_GET_OUTPUT_CNF – Confirmation of the DNS Get Output Image

4.3.3 Update I/O Image Service

The command DNS_FAL_UPDATE_IO_REQ is used by user application to get the output data as well as set the input data from the DeviceNet Slave stack.

The count of input data to be transferred is specified by the value of ulInLen. The maximum permitted length is 255 bytes.

The input offset address is specified in the parameter ulInOffSet. The specified address is interpreted from the device as the relative address to the start address in the send process data or the receive process data. The maximum value is decimal 255 for byte access.

The count of output data to be transferred is specified by the value of ulOutLen. The maximum permitted length is 255 bytes.

The output offset address is specified in the parameter ulOutOffSet. The specified address is interpreted from the device as the relative address to the start address in the send process data or the receive process data. The maximum value is decimal 255 for byte access.

The output data from the DeviceNet Master is actually the output data image in the DNS task. It will appear in the abOutData[] field of the confirmation packet.

The following status information is delivered in the variable ulOutDataSta of the confirmation packet:

ulOutDataSta

Symbolic code	Numeric value	Meaning
DNS_FAL_DS_ZERO	0x00000000	Safe state zero
DNS_FAL_DS_RECV_RUN	0x00000001	Data are valid
DNS_FAL_DS_RECV_IDLE	0x00000002	Data are in recv idle
DNS_FAL_DS_RECV_IDLE_ZERO	0x00000003	Recv idle zero
DNS_FAL_DS_HOLD_LAST_STATE	0x00000004	Hold last state
DNS_FAL_DS_USER_STATE	0x00000005	User defined state

Table 30: Data Status of Produced/ Consumed Data (Allowed Values for ulOutDataSta)

An explanation of how this command is used is as follows.

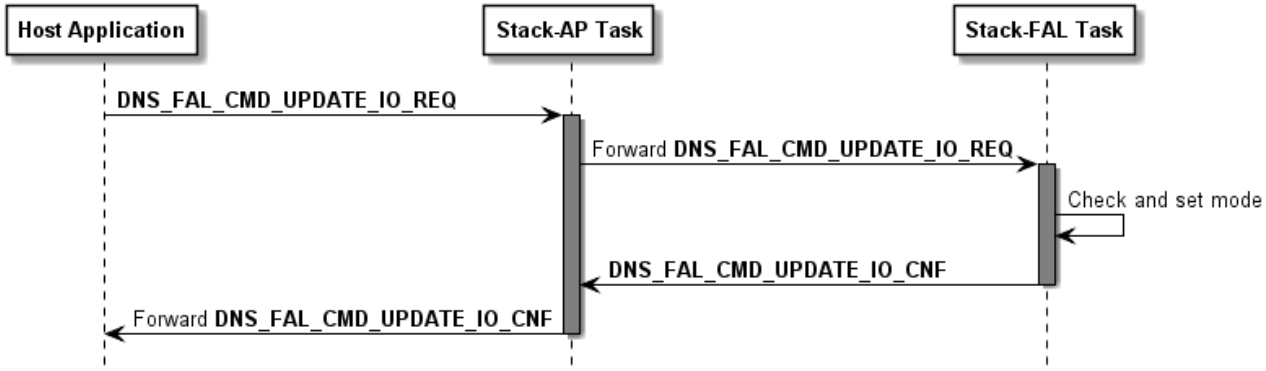


Figure 10: Sequence Diagram for the DNS_FAL_CMD_UPDATE_IO_REQ/CNF Packet

4.3.3.1 Update I/O Image Request

The application has to send this request to the protocol stack.

Packet Structure Reference

```
typedef struct DNS_FAL_UPDATE_IO_REQ_Ttag {
  TLR_UINT32   ulOutOffSet;
  TLR_UINT32   ulOutLen;
  TLR_UINT32   ulOutDataSta;
  TLR_UINT32   ulInOffSet;
  TLR_UINT32   ulInLen;
  TLR_UINT32   ulInDataSta;
  TLR_UINT8    abInData[256];
} DNS_FAL_UPDATE_IO_REQ_T;

typedef struct DNS_FAL_PACKET_UPDATE_IO_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
   DNS_FAL_UPDATE_IO_REQ_T tData;
} DNS_FAL_PACKET_UPDATE_IO_REQ_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32	LFW: 0x00000020, LOM: DNS FAL	LFW: AP task, LOM: DNS FAL Task
ulLen	UINT32	24+ ulInLen	Packet Data Length in bytes
ulCmd	UINT32	0x2D0C	DNS_FAL_CMD_UPDATE_IO_REQ - Command
tData			
ulOutOffSet	UINT32	0-255	Offset in the Output Data.
ulOutLen	UINT32	0-255	Length of outputs.
ulOutDataSta	UINT32	0	Reserved, set to zero
ulInOffSet	UINT32	0-255	Offset in the Input Data image in which to receive the Input data.
ulInLen	UINT32	0-255	Input length.
ulInDataSta	UINT32	0	Reserved, set to zero
abInData[]	UINT8	0-255	Input data sent to the DNS task. This information will appear to the master as input data.

Table 31: DNS_FAL_CMD_UPDATE_IO_REQ – Request Command for I/O Image Update

4.3.3.2 Update I/O Image Confirmation

This confirmation will be returned to the application.

Packet Structure Reference

```
typedef struct DNS_FAL_UPDATE_IO_CNF_Ttag {
  TLR_UINT32   ulOutOffSet;
  TLR_UINT32   ulOutLen;
  TLR_UINT32   ulOutDataSta;
  TLR_UINT32   ulInOffSet;
  TLR_UINT32   ulInLen;
  TLR_UINT32   ulInDataSta;
  TLR_UINT8    abOutData[256];
} DNS_FAL_UPDATE_IO_CNF_T;

typedef struct DNS_FAL_PACKET_UPDATE_IO_CNF_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_UPDATE_IO_CNF_T tData;
} DNS_FAL_PACKET_UPDATE_IO_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	24+ ulOutLen	Packet Data Length in bytes. Length of abInData[]
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D0D	DNS_FAL_CMD_UPDATE_IO_CNF - Command
tData			
ulOutOffSet	UINT32	0-255	Offset in the Output Data.
ulOutLen	UINT32	0-255	Length of outputs.
ulOutDataSta	UINT32	0x00000000 0x00000001 0x00000002 0x00000003 0x00000004 0x00000005	Data status of output (consumed) data DNS_FAL_DS_ZERO - Safe state zero DNS_FAL_DS_RECV_RUN - Data are valid DNS_FAL_DS_RECV_IDLE - Data are in recv idle DNS_FAL_DS_RECV_IDLE_ZERO - Recv idle zero DNS_FAL_DS_HOLD_LAST_STATE - Hold last state DNS_FAL_DS_USER_STATE - User defined state See more on Table 30: Data Status of Produced/ Consumed Data (Allowed Values for ulOutDataSta)
ulInOffSet	UINT32	0-255	Offset in the Input Data image in which to receive the Input data.
ulInLen	UINT32	0-255	Input length.
ulInDataSta	UINT32	0	Reserved zero
abOutData[]	UINT8[]		Output data from DNS task.

Table 32: DNS_FAL_CMD_UPDATE_IO_CNF – Confirmation of the DNS I/O Image Update

4.4 Register Application Services

4.4.1 rcX Register Application Service RCX_REGISTER_APP_REQ/CNF

This packet is used to register with host application. For further information please refer to Dual-Port Memory manual, reference [1].

4.5 Indication of Events

4.5.1 Combined LED (MNS) State Service

This packet indicates a change of the LED state. To receive this indication in the Stack-AP-Task, the corresponding enable flag `DNS_FAL_EVENT_LED_STATE_IND` must be set within the parameter `ulMode` of the command `DNS_FAL_CMD_APP_REQ`. This indication will not be forwarded to host application.

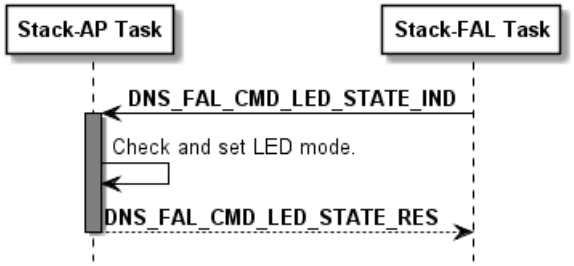


Figure 11: Sequence Diagram for the DNS_FAL_CMD_LED_STATE_IND/RES Packet

The type of the affected LED is indicated by parameter `ulLedType` in the following way:

Value of ulLedType	Meaning
1	MNS LED

Table 33: Meaning of ulLedType

The DNS task itself can provide only the LED information according to the network access state. The network access state is represented by the Network Status LED (MNS LED). The application is responsible to combine this information with the Module Status information and control the connected LED's according the "Chapter 9-2 Indicators" of the reference document [4].

The mode of the affected LED is indicated by parameter `ulLedMode` according to this table:

Value of ulLedMode	Meaning
1	Static
2	Flash

Table 34: Meaning of ulLedMode

The color of the affected LED is indicated by parameter `ulLedColor` as follows:

Value of ulLedColor	Meaning
1	Off
2	Green
3	Red

Table 35: Meaning of ulLedColor

The response packet without parameters needs to be sent to receive further change of state indications of the LED.

Note: The power-up sequence of a DeviceNet LED is not indicated to the AP task, because the stack task does not know about the used DeviceNet LED system, if it is an combined module network status LED (MNS) or if there are two separate LED's (NS and MS) to indicate the device state. This handling must be done in the AP task context.

4.5.1.1 LED State Indication

The stack task will provide three information to the application task: LED type, LED mode and LED color.

Packet Structure Reference

```
typedef struct DNS_FAL_LED_STATE_Ttag
{
  TLR_UINT32 ulLedType;  /* DNS_FAL_LED_TYPE_MNS */
  TLR_UINT32 ulLedMode;  /* DNS_FAL_LED_MODE_STATIC ,.._FLASH */
  TLR_UINT32 ulLedColor; /* DNS_FAL_LED_COLOR_OFF ,..GREEN, ..RED  */
} DNS_FAL_LED_STATE_T;

#define DNS_FAL_LED_STATE_IND_SIZE (sizeof(DNS_FAL_LED_STATE_T))

typedef struct DNS_FAL_PACKET_LED_STATE_IND_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_LED_STATE_T tData;
} DNS_FAL_PACKET_LED_STATE_IND_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	12	Packet Data Length in bytes
ulCmd	UINT32	0x2D1C	DNS_FAL_CMD_LED_STATE_IND - Command
tData			
ulLedType	UINT32	1	LED Type: MNS
ulLedMode	UINT32	1…2	LED Mode: Static = 1, flash = 2.
ulLedColor	UINT32	1…3	Color of LED: Off = 1, green = 2, red = 3.

Table 36: DNS_FAL_CMD_LED_STATE_IND – LED State Indication

4.5.1.2 LED State Response

The response packet to the stack task is a header-only packet. The stack does not consider the error provided in `ulSta` variable.

Packet Structure Reference

```
typedef struct DNS_FAL_PACKET_LED_STATE_RES_Ttag {
  TLR_PACKET_HEADER_T tHead;
} DNS_FAL_PACKET_LED_STATE_RES_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32		Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D1D	DNS_FAL_CMD_LED_STATE_RES

Table 37: DNS_FAL_CMD_LED_STATE_RES – LED State Response

4.5.2 Separated LED (MS and NS) State Service

This packet indicates a change of the LED state. There are two separated LEDs: Module State (MS) and Network State (NS). To receive this indication in the Host application, the corresponding enable flag `DNS_FAL_EVENT_LED_STATE_IND` must be set within the parameter `ulMode` of the command `DNS_FAL_CMD_APP_REQ`.

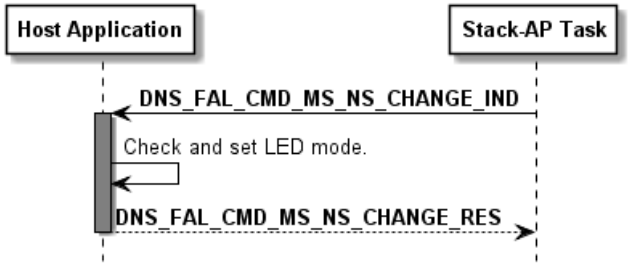


Figure 12: Sequence Diagram for the DNS_FAL_CMD_MS_NS_CHANGE_IND/RES Packet

The module LED is presented by variable `ulModuleStatus`:

Value of ulModuleStatus	Meaning
0x00	No change.
0x01	No Power. The LED should be off.
0x02	Self-Test. Power up testing. The LED should performing the testing blink sequence.
0x04	Standby. The device has not been configured. The LED should flashing green.
0x08	The device is operating and the LED should be steady green.
0x10	Minor fault, recoverable fault. The LED should flashing red.
0x20	Major fault, non recoverable critical fault. Steady red.

Table 38: Meaning of ulModuleStatus

The network LED is presented by variable `ulNetworkStatus`:

Value of ulNetworkStatus	Meaning
0x0000	No change.
0x0100	No Power. Dupplicate MAC ID checking not completed. The LED should be off.
0x0200	No connection. MAC address and baurate is configured but no connection established from master. The LED should flashing green.
0x0400	Connected. At least on connection established. The LED should be steady green.
0x0800	Connection time out. The LED should flashing red.
0x1000	Major fault. The device detected unrecoverable communication error i.e. duplicate MAC ID. The LED should steady red.
0x2000	Self-test. The device is performing its power-on self testing procedure.

Table 39: Meaning of ulNetworkStatus

4.5.2.1 LED State Indication

The stack task will provide two information to the host application: The state of the Module LED and the state of the Network LED

Packet Structure Reference

```
/*****************************************************************************/
/*                  DNS_FAL_MS_NS_CHANGE_IND_T Structure                     */
/*****************************************************************************/
typedef __PACKED_PRE struct DNS_FAL_MS_NS_CHANGE_IND_Ttag
{
  TLR_UINT32  ulModuleStatus;  /* Module Status MS LED
                                  DNS_FAL_LED_MODUL_STATUS_E for more details.*/

  TLR_UINT32  ulNetworkStatus;/*!< Network Status \n
                                 DNS_FAL_LED_NETWORK_STATUS_E for more details*/

}__PACKED_POST  DNS_FAL_MS_NS_CHANGE_IND_T;

#define DNS_FAL_MS_NS_CHANGE_IND_SIZE (sizeof(DNS_FAL_MS_NS_CHANGE_IND_T))

typedef struct DNS_FAL_PACKET_MS_NS_CHANGE_IND_Ttag
{
  TLR_PACKET_HEADER_T             tHead;
  DNS_FAL_MS_NS_CHANGE_IND_T      tData;
} DNS_FAL_PACKET_MS_NS_CHANGE_IND_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	12	Packet Data Length in bytes
ulCmd	UINT32	0x2D30	DNS_FAL_CMD_MS_NS_CHANGE_IND
tData - Structure DNS_FAL_LED_STATE_T			
ulModuleStatus	UINT32		See Table 38 on page 49.
ulNetworkStatus	UINT32		See Table 39 on page 49.

Table 40: DNS_FAL_CMD_MS_NS_CHANGE_IND – LED State Indication

4.5.2.2 LED State Response

The response packet to the stack task is a header-only packet. The stack does not consider the error provided in `ulSta` variable.

Packet Structure Reference

```
#define DNS_FAL_MS_NS_CHANGE_RES_SIZE 0

typedef struct DNS_FAL_PACKET_MS_NS_CHANGE_RES_Ttag
{
  TLR_PACKET_HEADER_T             tHead;
} DNS_FAL_PACKET_MS_NS_CHANGE_RES_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32		Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D31	DNS_FAL_CMD_MS_NS_CHANGE_RES

Table 41: DNS_FAL_CMD_LED_STATE_RES – LED State Response

4.5.3 IO Update Indication

The packet DNS_FAL_CMD_UPDATE_IO_IND indicates whenever the receive data are refreshed or changed e.g. the master has send new data or the receive data are cleared due to connection lost. To receive this indication to the AP task the corresponding enable flag 'DNS_FAL_EVENT_IO_UPDATE_IND' or 'DNS_FAL_EVENT_IO_CHANGED_IND' must be set within the parameter 'ulMode' of the command 'DNS_FAL_CMD_APP_REQ'. This indication can be used from AP task to obtain the latest receive data from stack event driven. Depending on the registered event this indication will be send every time when the master has updated the receive data or the receive data are changed (change of state). The response packet without parameters needs to be sent to receive further update indications of the receive data.

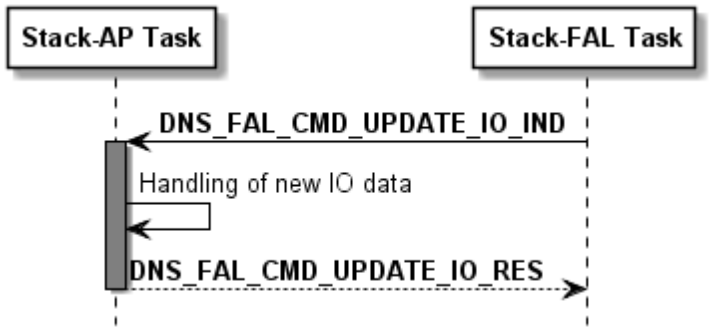


Figure 13: Sequence Diagram for the DNS_FAL_UPDATE_IO_IND/RES Packet

4.5.3.1 IO Update Indication Packet

Packet Structure Reference

```
typedef struct DNS_FAL_PACKET_UPDATE_IO_IND_Ttag {
  TLR_PACKET_HEADER_T tHead;
} DNS_FAL_PACKET_UPDATE_IO_IND_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x2D1A	DNS_FAL_CMD_UPDATE_IO_IND - Command

Table 42: DNS_FAL_CMD_UPDATE_IO_IND – Update IO Indication

4.5.3.2 IO Update Response Packet

The response packet with no data load need to be send back to stack task to acknowledge the preceding indication.

Packet Structure Reference

```
typedef struct DNS_FAL_PACKET_UPDATE_IO_RES_Ttag {
  TLR_PACKET_HEADER_T tHead;
} DNS_FAL_PACKET_UPDATE_IO_RES_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D1B	DNS_FAL_CMD_UPDATE_IO_RES - Command

Table 43: DNS_FAL_CMD_UPDATE_IO_RES – Update IO Response

4.5.4 Address Switch Enable Indication

The AP task indicates the stack task about the enabling of the hardware switches. The AP task should use this command when the hardware (MAC ID or Baudrate) switches are available and the AP task has access to actual value of these switches. That means the AP task should inform the stack task whenever the flag MSK_DNS_SYS_FLG_ADR_SW_ENABLE is set or the command RCX_SET_FW_PARAMETER_REQ is used.

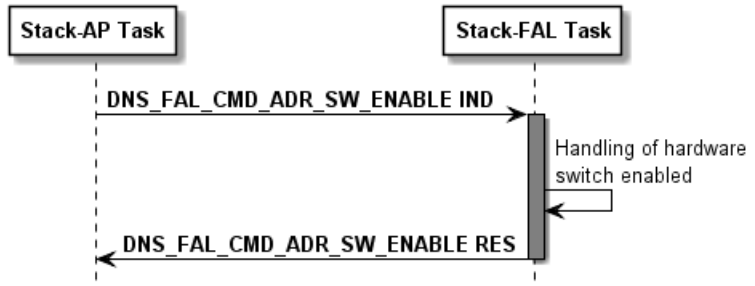


Figure 14: Sequence Diagram for the DNS_FAL_CMD_ADR_SW_ENABLE_IND Packet

4.5.4.1 Address Switch Enable Indication Packet

Packet Structure Reference

```
#define DNS_FAL_SW_TYPE_ADR           (1)
#define DNS_FAL_SW_TYPE_BAUD          (2)
#define DNS_FAL_SW_ENABLE_STANDARD    (1)
#define DNS_FAL_SW_ENABLE_FW_PRM_SET  (2)

typedef __PACKED_PRE struct __PACKED_POST DNS_FAL_ADR_SW_ENABLE_Ttag
{
  TLR_UINT32 ulSwType;                                /* DNS_FAL_SW_TYPE_ .. */
  TLR_UINT32 ulEnable;                                /* 1 = Enabled */
}
DNS_FAL_ADR_SW_ENABLE_T;

#define DNS_FAL_ADR_SW_ENABLE_IND_SIZE (sizeof(DNS_FAL_ADR_SW_ENABLE_T))

typedef __PACKED_PRE struct __PACKED_POST DNS_FAL_PACKET_SW_ENABLE_IND_Ttag
{
  TLR_PACKET_HEADER_T             tHead;
  DNS_FAL_ADR_SW_ENABLE_T         tData;
}
DNS_FAL_PACKET_SW_ENABLE_IND_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32	LFW: 0x00000020, LOM: DNS FAL	LFW: AP task, LOM: DNS FAL Task
ulLen	UINT32	8	Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D26	DNS_FAL_CMD_LED_STATE_IND - Command
tData - Structure DNS_FAL_ADR_SW_ENABLE_T			
ulSwType	UINT32	1..2	1: Address switch (DNS_FAL_SW_TYPE_ADR) 2: Baudrate switch (DNS_FAL_SW_TYPE_BAUD)
ulEnable	UINT32	0..2	0 default. Not enabled. 1 if MSK_DNS_SYS_FLG_ADR_SW_ENABLE is set 2 if RCX_SET_FW_PARAMETER_REQ is used

Table 44: DNS_FAL_CMD_ADR_SW_ENABLE_IND – Hardware Switch Enabled Indication

4.5.4.2 Address Switch Enable Response Packet

Packet Structure Reference

```
typedef __PACKED_PRE struct __PACKED_POST DNS_FAL_ADR_SW_ENABLE_Ttag
{
  TLR_UINT32 ulSwType;                                  /* DNS_FAL_SW_TYPE_ .. */
  TLR_UINT32 ulEnable;                                  /* 1 = Enabled */
}
DNS_FAL_ADR_SW_ENABLE_T;

#define DNS_FAL_ADR_SW_ENABLE_RES_SIZE (sizeof(DNS_FAL_ADR_SW_ENABLE_T))

typedef __PACKED_PRE struct __PACKED_POST DNS_FAL_PACKET_SW_ENABLE_RES_Ttag
{
  TLR_PACKET_HEADER_T             tHead;
  DNS_FAL_ADR_SW_ENABLE_T         tData;
}
DNS_FAL_PACKET_SW_ENABLE_RES_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32		Destination Queue-Handle of DNS Task Process Queue
ulLen	UINT32	12	Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D27	DNS_FAL_CMD_LED_STATE_RES - Command
tData			
ulSwType	UINT32	1..2	1: Address switch (DNS_FAL_SW_TYPE_ADR) 2: Baudrate switch (DNS_FAL_SW_TYPE_BAUD)
ulEnable	UINT32	0..2	0 default. Not enabled. 1 if MSK_DNS_SYS_FLG_ADR_SW_ENABLE is set 2 if RCX_SET_FW_PARAMETER_REQ is used

Table 45: DNS_FAL_CMD_ADR_SW_ENABLE_RES – Hardware Switch Enabled Indication

4.5.5 Bus Event Indication

The Bus Event Indication Service DNS_FAL_CMD_BUS_EVENT_IND/RES is not used.

4.6 Explicit Messaging Services

The Master can require the slave to perform a specific service via explicit message. The explicit message may not be sent on a cyclic basis and therefore to distinguish from I/O messaging. Explicit messages are also named acyclic services.

Service Codes

The following service codes are according to reference [3], Chapter A-3 CIP Common Services, Table A-3.1:

Value of bServiceCode	Service to be executed
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 46: Service Codes

Note: Not every service is available on every object and on every device in the network. Using this packet only makes sense if you check that the selected object exists on the addressed device and supports the service selected by variable `bServiceCode`.

General Status Codes

The Generic Error (variable `bGenErrCode`) of the confirmation/response packet are according to reference [3], Chapter B-1 General Status Codes, Table B-1.1:

Value of bGenError	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 47: Generic Error

Additional Error Codes

The additional error is network device specific. If necessary, manufacturers can individually define an additional code while implementing a service on the device (see section *Remote Service* on page 75 and *Get / Set Attribute Service* on page 78). The pre-defined additional error codes are mentioned in various sections in references [3] and [4].

Class ID

The following table provides the already predefined values of the frequently used class IDs according to the CIP specification. For the full list of class IDs, please refer to [4].

Symbolic constant	Numeric value	Accessed object
DNS_CIP_CLASS_IDENTITY	0x01	Identity Object
DNS_CIP_CLASS_MESSAGE_ROUTER	0x02	Message Router Object
DNS_CIP_CLASS_DEVICENET	0x03	DeviceNet Object
DNS_CIP_CLASS_ASSEMBLY	0x04	Assembly Object
DNS_CIP_CLASS_CONNECTION	0x05	Connection Object
DNS_CIP_CLASS_PARAMETER	0x0F	Parameter Object
DNS_CIP_CLASS_ACKNOWLEDGE_HANDLER	0x2B	Acknowledge Handler Object

Table 48: Predefined Values for the Class ID according to the CIP Specification

Instance ID

If the DeviceNet object is selected, the following predefined values are available for variable `usInstanceID` of the request packet:

Numeric value	Accessed instance	Data type/ Range of values	Symbolic constant / Meaning
0x00	All attributes		DNS_CIP_CLASS_DEVICENET_ALL_ATTRS Not supported
0x01	Revision		DNS_CIP_CLASS_DEVICENET_REVISION Revision
0x01	MAC ID (Node address)	UINT16/ 0…63	DNS_CIP_CLASS_DEVICENET_MACID MAC ID (Node address) of accessed device
0x02	Baudrate	UINT16/ 0…2	DNS_CIP_CLASS_DEVICENET_BAUD Baudrate of accessed device
0x03	Bus-off interrupt (BOI)	BOOLEAN	DNS_CIP_CLASS_DEVICENET_BOI TRUE: BOI occurred FALSE: BOI did not occur
0x04	Bus-off counter	UINT16 0…255	DNS_CIP_CLASS_DEVICENET_BUS_OFF_CNTR Number of bus-off interrupt events since last start-up
0x05	Allocation information	struct	DNS_CIP_CLASS_DEVICENET_ALLOC_INFO Allocation info (contains Allocation Choice Byte and MAC ID of DeviceNet Master.)
0x06	MAC ID switch changed	BOOLEAN	DNS_CIP_CLASS_DEVICENET_MACSWCHANGED TRUE: MAC ID switch changed since last start-up FALSE: MAC ID switch did not change since last start-up
0x07	Baudrate switch changed	BOOLEAN	DNS_CIP_CLASS_DEVICENET_BAUDSWCHANGED TRUE: Baudrate switch changed since last start-up FALSE: Baudrate switch did not change since last start-up
0x08	MAC ID switch value	UINT16/ 0…99	Current value of MAC ID switch
0x09	Baudrate switch value	UINT16/ 0…9	Current value of Baudrate switch
0x0A	Quick_ Connect	BOOLEAN	TRUE: Quick_Connect feature enabled FALSE: Quick_Connect feature disabled (default)

Table 49: Predefined Values for the Instance ID according to the DeviceNet Specification

4.6.1 Request Service from Object of Local Node

This service is used to request a service from an object on a local connected device. The service to be performed is selected by setting the parameter `bServiceCode` of the request packet to the according service code (Table 46: Service Codes).

The class and the instance of the object to be accessed are selected by the variables `usClassId` and `usInstanceID` of the request packet. If an attribute is affected by the service (`Get_Attribute_Single` and `Set_Attribute_Single`), this attribute is selected by variable `usAttribute` of the request packet. Set `usAttribute` to 0 when using other services than these.

The result of the requested service is delivered in array `abSrvData[248]` of the confirmation packet. The number of bytes of array `abSrvData[248]` which will be actually used can be specified in variable `usSrvDatLen` of the request packet.

In case of successful execution, the variables `bGenErrCode` and `bAddErrCode` of the confirmation packet will have the value 0.

In case of an error, the variable `bServiceCode` of the confirmation packet will have the value 0x14 (Error Response), the variables `bGenErrCode` and `bAddErrCode` of the confirmation packet will have non-zero values.

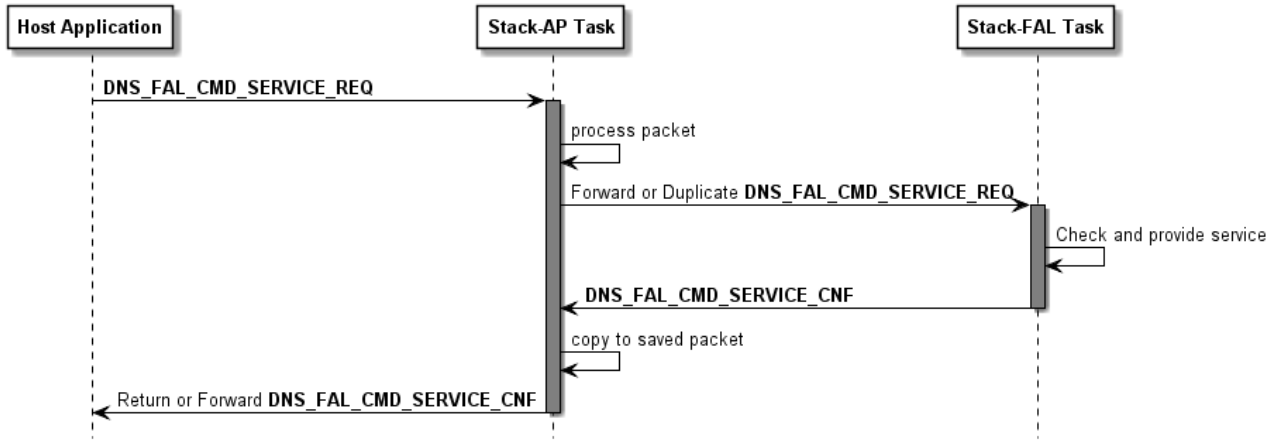


Figure 15: Sequence Diagram for the DNS_FAL_CMD_SERVICE_REQ/CNF Packet

4.6.1.1 Local Service Request

Packet Structure Reference

```
/*****************************************************************************/
/*                       DNS_FAL_CMD_SERVICE_REQ                             */
/*****************************************************************************/
typedef struct DNS_FAL_SERVICE_IND_Ttag{
  /* Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1 */
  TLR_UINT16  usClassId;                              /* Class Id            */
  TLR_UINT16  usInstanceId;    /* Instance Id. See specs of individual object */
  TLR_UINT16  usAttribute;      /* Attribute Id for get/set attr only         */
  TLR_UINT16  usSrvDatLen;     /* Effective Length of Data in abSrvData       */
  TLR_UINT8   bServiceCode;                           /* Service Code      */
  TLR_UINT8   abReserved[2]; /*Reserved. Paddling for bGenErrCode, bAddErrCode*/
  TLR_UINT8   bDevMacId;                        /* reserved, not used, set to 0 */
  TLR_UINT8   abSrvData[DNS_FAL_REMOTE_SERVICE_MAX_DATA];     /* Service data */
}DNS_FAL_SERVICE_REQ_T;

#define DNS_FAL_SERVICE_REQ_SIZE (sizeof(DNS_FAL_SERVICE_REQ_T) \
                                  -DNS_FAL_REMOTE_SERVICE_MAX_DATA)

typedef struct DNS_FAL_PACKET_SERVICE_REQ_Ttag
{
  TLR_PACKET_HEADER_T        tHead;
  DNS_FAL_SERVICE_REQ_T      tData;
}DNS_FAL_PACKET_SERVICE_REQ_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32	LFW: 0x00000020, LOM: DNS FAL	LFW: AP task, LOM: DNS FAL Task
ulLen	UINT32		Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D24	DNS_FAL_CMD_SERVICE_REQ - Command
tData			
usClassId	UINT16	Valid Class ID	Class ID (according to *"The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1" of the reference document [3].*
usInstanceId	UINT16	Valid Instance ID	Instance ID of the class specified by usClassId
usAttribute	UINT16	Valid Attribute ID	Attribute ID of an instance specified by usInstanceId. Attribute of an object is described in details in reference [3] and [4].
usSrvDatLen	UINT16	0..248	Effective Length of Data in abSrvData
bServiceCode	UINT8	0-49	Service Code. See Table 46: Service Codes
abReserved[2]	UINT8		Reserved, Padding of bGenErrCode and bAddErrCode
bDevMacId	UINT8	0	Reserved. Not used. Set to 0.
abSrvData[248]	UINT8		Service data

Table 50: DNS_FAL_PACKET_SERVICE_REQ_T - Remote Service Request

4.6.1.2 Local Service Confirmation

Packet Structure Reference

```
/*****************************************************************************/
/*                       DNS_FAL_CMD_SERVICE_CNF                             */
/*****************************************************************************/
typedef struct DNS_FAL_SERVICE_CNF_Ttag{
  TLR_UINT16  usClassId;                              /* Class Id            */
  TLR_UINT16  usInstanceId;    /* Instance Id. See specs of individual object */
  TLR_UINT16  usAttribute;     /* Attribute Id for get/set attr only          */
  TLR_UINT16  usSrvDatLen;     /* Effective Length of Data in abSrvData       */
  TLR_UINT8   bServiceCode;     /* Service Code. 14 in case of error response */
  TLR_UINT8   bGenErrCode;      /* General error code. Appendix B-1. Volume 1 */
  TLR_UINT8   bAddErrCode;      /* Additional error code. Exp Msg. Volume 3   */
  TLR_UINT8   bDevMacId;                       /* reserved, not used, set to 0 */
  TLR_UINT8   abSrvData[DNS_FAL_REMOTE_SERVICE_MAX_DATA];/* Service data      */
}DNS_FAL_SERVICE_CNF_T;

#define DNS_FAL_SERVICE_CNF_SIZE (sizeof(DNS_FAL_SERVICE_CNF_T) \
                                        -DNS_FAL_REMOTE_SERVICE_MAX_DATA)

typedef struct DNS_FAL_PACKET_SERVICE_CNF_Ttag
{
  TLR_PACKET_HEADER_T          tHead;
  DNS_FAL_SERVICE_CNF_T        tData;
}DNS_FAL_PACKET_SERVICE_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32		Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D25	DNS_FAL_CMD_SERVICE_CNF - Command
tData			
usClassId	UINT16	1 … 0xFFFF	Class ID (according to *"The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1" of the reference document [3].*
usInstanceId	UINT16	Valid instance	Instance ID of the class specified by usClassId
usAttribute	UINT16		Attribute ID of an instance specified by usInstanceId. Attributes of an object is described in details in object profiles of reference [3] and [4].
usSrvDatLen	UINT16	0..248	Effective Length of Data in abSrvData
bServiceCode	UINT8	0-31	Service Code. See Table 46: Service Codes
bGenErrCode	UINT8		General error code. See Table 47: Generic Error
bAddErrCode	UINT8		Additional error code.
bDevMacId	UINT8		Reserved. Set to 0.
abSrvData[248]	UINT8		Service data

Table 51: DNS_FAL_PACKET_SERVICE_CNF_T - Confirmation to Remote Service Request

4.6.2 Register Class Service

The DeviceNet Slave stack contains the functionality to forward explicit services like *Get/Set Attribute* to the user, except the Objects/Class which are handled by the stack itself.

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.

The range of Class ID usable with the Register Class service is: 1, 3 to 42, and 44 to 255.

The stack handles the Message Router Class (0x02) and the Acknowledge Handler Class (0x2B, 43) internally and thus the user cannot register these classes.

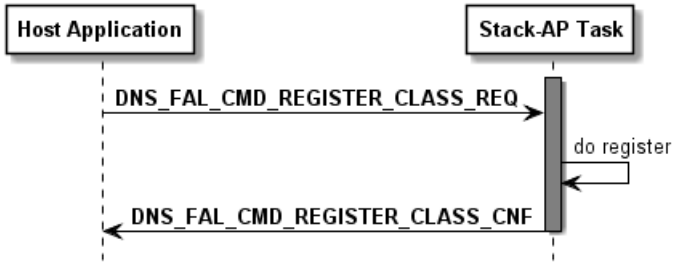


Figure 16: Sequence Diagram for the DNS_FAL_CMD_REGISTER_CLASS_REQ/CNF Packet from Host Application

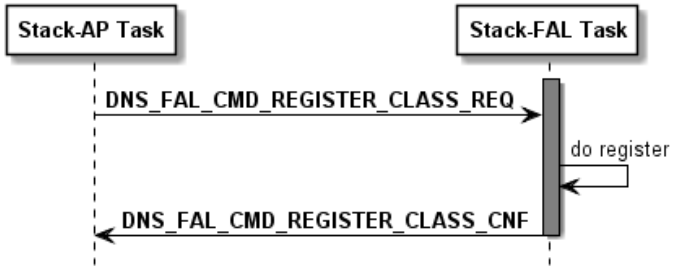


Figure 17: Sequence Diagram for the DNS_FAL_CMD_REGISTER_CLASS_REQ/CNF Packet from AP Application

4.6.2.1 Register Class Request DNS_FAL_PACKET_REGISTER_CLASS_REQ

Packet Structure Reference

```
/*****************************************/
/* DNS_FAL_CMD_REGISTER_CLASS_REQ Structure  */

typedef struct DNS_FAL_REGISTER_CLASS_Ttag {
  TLR_UINT32               ulClass;         /* Class identifier        */
  TLR_UINT32               ulAccessTyp;  /* Instance, Class access  */
} DNS_FAL_REGISTER_CLASS_T;

#define DNS_FAL_REGISTER_CLASS_REQ_SIZE (sizeof(DNS_FAL_REGISTER_CLASS_T))

typedef struct DNS_FAL_PACKET_REGISTER_CLASS_REQ _Ttag {
  TLR_PACKET_HEADER_T              tHead;
  DNS_FAL_REGISTER_CLASS_T       tData;
} DNS_FAL_PACKET_REGISTER_CLASS_REG_T;
```

Packet Description

Variable	Type	Value / range	Description
ulDest	UINT32	LFW: 0x00000020, LOM: DNS FAL	LFW: AP task, LOM: DNS FAL Task
ulLen	UINT32	8	Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D1E	DNS_FAL_CMD_REGISTER_CLASS_REQ - Command
Data			
ulClass	UINT32	1, 3 … 42, 44 ... 255	Class ID (according to reference [3], Chapter 5A, Table 5A-1.1) For class ID 1, 3, and 5, the application can use a bitlist (ulAccessTyp) to select specific services. By default, the stack forwards all services to the application for class ID 4, 6 … 42, and 44 … 255. For the Identity class (class code 0x01), only services to instance 1 will be forwarded to the application. The Message Router (class code 0x02) and the Acknowledge Handler (class code 0x2B) cannot be registered. Assembly (class code 0x04) services to be forwarded to the application is implemented starting with firmware V2.5.0.3. For the Connection class (class code 0x05), only services to instance 0 will be forwarded to the application. Class ID 0 is invalid.
ulAccessTyp	UINT32	 0 bitlist	Services to be registered (forwarded to the application) All services including service codes 0x32 to 0xFF. Registered services according to Table 53 on page 65 to be forwarded to the application.

Table 52: DNS_FAL_CMD_REGISTER_CLASS_REQ – Register Class Request

Service Codes

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 53: Service Codes depending to the bit position

General Case of Using DNS_FAL_CMD_REGISTER_CLASS_REQ

```
TLR_RESULT DnsApp_RegDevNetClx_Req(DNS_APP_RSC_T FAR* ptRsc, TLR_UINT8 ubObjClx)
{
  TLR_RESULT eRslt;
  DNS_FAL_PACKET_REGISTER_CLASS_REQ_T* ptRegObjClxReq:

  eRslt = TLR_POOL_PACKET_GET(ptRsc->tLoc.hPool, &ptRegObjClxReq);
  if( eRslt == TLR_S_OK )
  {
    TLR_QUE_LINK_SET_PACKET_SRC(ptRegObjClxReq,ptRsc->tLoc.tQueSrcDnsApp);
    ptRegObjClxReq ->tHead.ulCmd = DNS_FAL_CMD_REGISTER_CLASS_REQ;
    ptRegObjClxReq ->tHead.ulLen = sizeof(DNS_FAL_PACKET_REGISTER_CLASS_REQ_T)\
                                   - sizeof(TLR_PACKET_HEADER_T);

    ptRegObjClxReq ->tData.ulClass = (TLR_UINT32)ubObjClx;
    ptRegObjClxReq ->tData.ulAccessType = 0;
    eRslt = TLR_QUE_SENDPACKET_FIFO(ptRsc >tLoc.tDnsApQue, ptRegObjClxReq,TLR_FINITE);
    if( eRslt != TLR_S_OK )
    {
      TLR_POOL_PACKET_RELEASE(ptRsc->tLoc.hPool, ptRegObjClxReq);
    }
  }
  return eRslt;
}
```

The diagram below shows the general handling of the remote service request to the Hilscher DeviceNet stack. Please refer to section Object model on page 15 for further information.

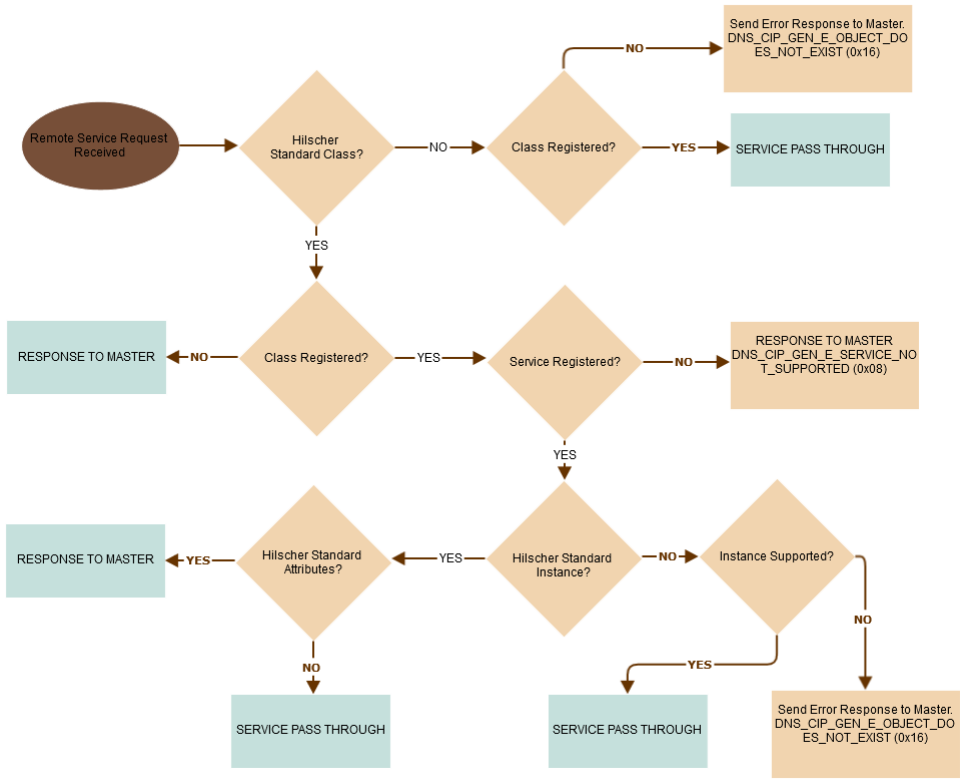


Figure 18: General handling of the remote service request

4.6.2.2 Register Class Confirmation DNS_FAL_PACKET_REGISTER_CLASS_CNF

Packet Structure Reference

```
typedef struct DNS_FAL_REGISTER_CLASS_Ttag
{
          TLR_UINT32 ulClass;            /* Class identifier                      */
          TLR_UINT32 ulAccessTyp;        /* Class access mode, preserved, not used*/
} DNS_FAL_REGISTER_CLASS_T;

typedef struct DNS_FAL_PACKET_REGISTER_CLASS_CNF_Ttag
{
  TLR_PACKET_HEADER_T      tHead;
  DNS_FAL_REGISTER_CLASS_T tData;
} DNS_FAL_PACKET_REGISTER_CLASS_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D1F	DNS_FAL_CMD_REGISTER_CLASS_CNF - Command
tData			
ulClass	UINT32	1, 3 … 42, 44 ... 255	Class ID (according to reference [3], Chapter 5A, Table 5A-1.1) **Note:** The Message Router (class code 0x02) and the Acknowledge Handler (class code 0x2B) cannot be registered. Class 0 is invalid.
ulAccessTyp	UINT32	 0 bitlist	Registered services to be forwarded to the application all services Registered services according to Table 53 on page 65 to be forwarded to the application.

Table 54: DNS_FAL_CMD_REGISTER_CLASS_CNF – Register Class Response

4.6.2.3 Handling of Remote Service Request in case of class/service not registered

In case the class is not registered, the stack will handle the remote service request by itself. The error code will be return accordingly to the master. The handling of the remote service request in case of the class is registered is mentioned in section *Remote Service* on page 75.

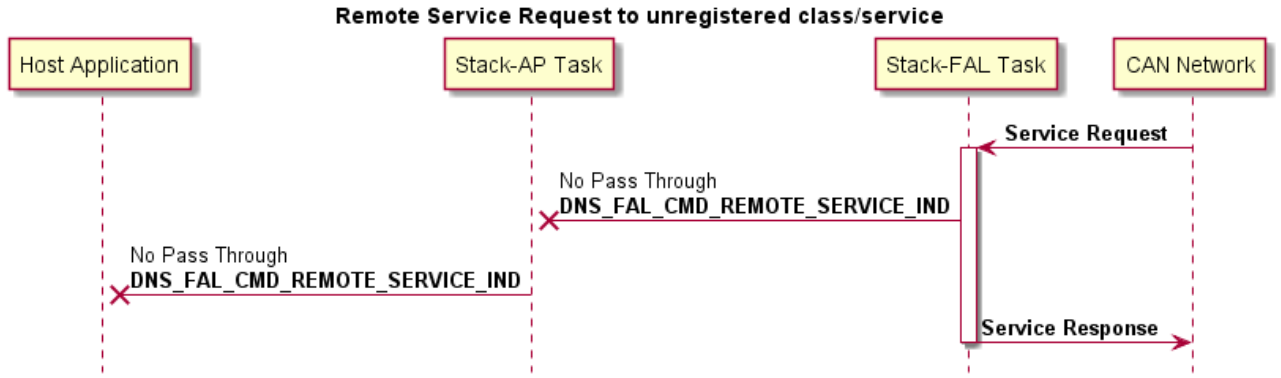


Figure 19: Remote Service Request to unregistered Class or Service

4.6.2.4 Example Register Connection Class for Set/Get_attribute_single

Pseudo code registering get/set_attribute_single of connection class

```
req_packet = DNS_FAL_PACKET_REG_APP_REQ_T()
req_packet.tHead.ulCmd = DNS_FAL_CMD_REG_APP_REQ
req_packet.tHead.ulSrc  = addressof(req_packet)
req_packet.tHead.ulDest = rcX_Public.RCX_PACKET_DEST_DEFAULT_CHANNEL
req_packet.tHead.ulExt  = rcX_Public.RCX_PACKET_SEQ_NONE
req_packet.tHead.ulLen  = sizeof(DNS_FAL_REG_APP_REQ_T)

req_packet.tData.ulMode = 0

dnet_slave.sendPacket(req_packet)
cnf_packet = dnet_slave.waitPacket(DNS_FAL_CMD_REG_APP_CNF)


req_packet = DNS_FAL_PACKET_REGISTER_CLASS_REQ_T()
req_packet.tHead.ulCmd = DNS_FAL_CMD_REGISTER_CLASS_REQ
req_packet.tHead.ulSrc  = addressof(req_packet)
req_packet.tHead.ulDest = rcX_Public.RCX_PACKET_DEST_DEFAULT_CHANNEL
req_packet.tHead.ulExt  = rcX_Public.RCX_PACKET_SEQ_NONE
req_packet.tHead.ulLen  = sizeof(DNS_FAL_PACKET_REGISTER_CLASS_REQ_T)-
sizeof(TLR_PACKET_HEADER_T)

req_packet.tData.ulClass = 5
req_packet.tData.ulAccessTyp =
(0x01<<DEVNET_SRV_CODE_GET_ATTRIBUTE_SINGLE)|(0x01<<DEVNET_SRV_CODE_SET_ATTRIBUTE_SINGLE)

dnet_slave.sendPacket(req_packet)
cnf_packet = dnet_slave.waitPacket(DNS_FAL_CMD_REGISTER_CLASS_CNF)
cnf_packet.checkPacketStatus()
```

4.6.2.5 Example Register Identity Class for Reset Service

Pseudo code for registering reset service of identity class

```
req_packet = DNS_FAL_PACKET_REG_APP_REQ_T()
req_packet.tHead.ulCmd = DNS_FAL_CMD_REG_APP_REQ
req_packet.tHead.ulSrc  = addressof(req_packet)
req_packet.tHead.ulDest = rcX_Public.RCX_PACKET_DEST_DEFAULT_CHANNEL
req_packet.tHead.ulExt  = rcX_Public.RCX_PACKET_SEQ_NONE
req_packet.tHead.ulLen  = sizeof(DNS_FAL_REG_APP_REQ_T)

req_packet.tData.ulMode = 0

dnet_slave.sendPacket(req_packet)
cnf_packet = dnet_slave.waitPacket(DNS_FAL_CMD_REG_APP_CNF)

req_packet = DNS_FAL_PACKET_REGISTER_CLASS_REQ_T()
req_packet.tHead.ulCmd = DNS_FAL_CMD_REGISTER_CLASS_REQ
req_packet.tHead.ulSrc  = addressof(req_packet)
req_packet.tHead.ulDest = rcX_Public.RCX_PACKET_DEST_DEFAULT_CHANNEL
req_packet.tHead.ulExt  = rcX_Public.RCX_PACKET_SEQ_NONE
req_packet.tHead.ulLen  = sizeof(DNS_FAL_PACKET_REGISTER_CLASS_REQ_T)-
sizeof(TLR_PACKET_HEADER_T)

req_packet.tData.ulClass = 1
req_packet.tData.ulAccessTyp = 0x01<<DEVNET_SRV_CODE_RESET

dnet_slave.sendPacket(req_packet)
cnf_packet = dnet_slave.waitPacket(DNS_FAL_CMD_REGISTER_CLASS_CNF)
cnf_packet.checkPacketStatus()
```

Normally, when the identity class is not registered, the stack will handle the reset service by itself. But if the reset service is forwarded using DNS_FAL_CMD_REMOTE_SERVICE_IND, the stack will wait for DNS_FAL_CMD_REMOTE_SERVICE_RES and then act accordingly.

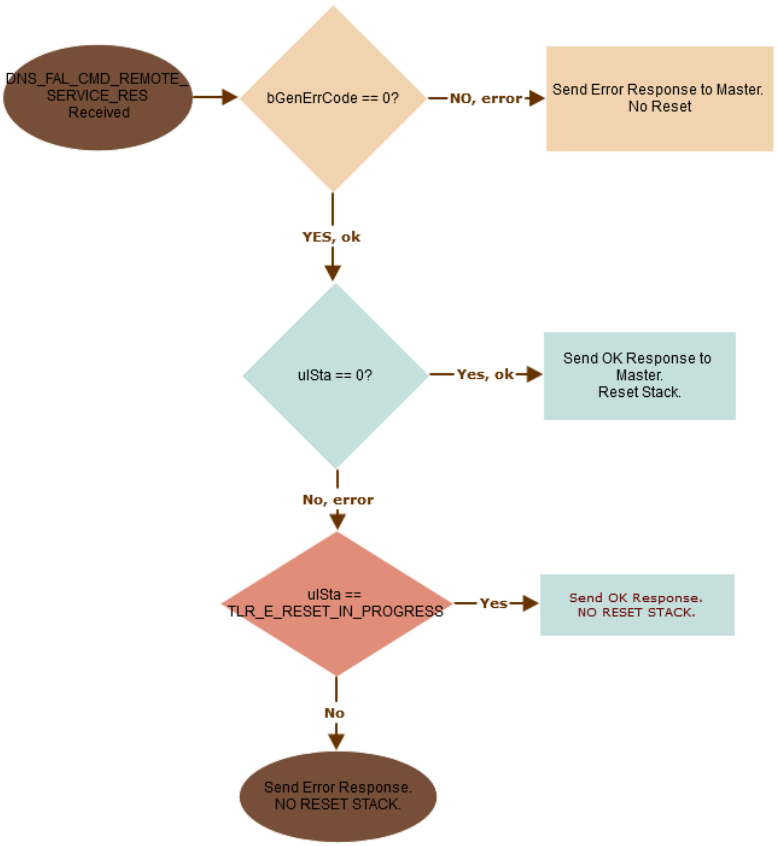
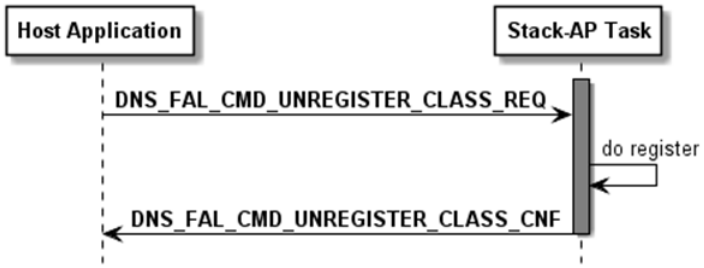


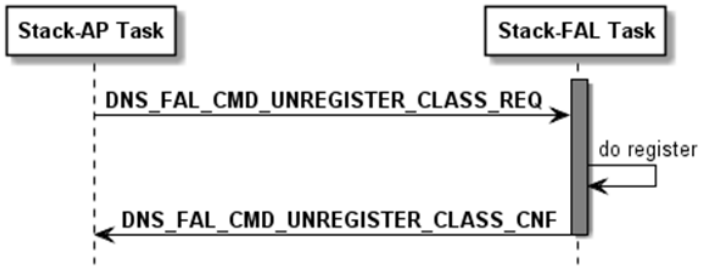
Figure 20: Response to DNS_FAL_CMD_REMOTE_SERVICE_RES

4.6.3 Unregister Class Service

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



Sequence Diagram for the DNS_FAL_CMD_UNREGISTER_CLASS_REQ/CNF Packet from Host Application



Sequence Diagram for the DNS_FAL_CMD_UNREGISTER_CLASS_REQ/CNF Packet from AP Application

4.6.3.1 Unregister Class Request DNS_FAL_CMD_UNREGISTER_CLASS_REQ

Packet Structure Reference

```
typedef struct DNS_FAL_UNREGISTER_CLASS_Ttag
{
 TLR_UINT32 ulClass;     /* Class identifier */
 TLR_UINT32 ulAccessTyp; /* Instance, Class access */
}
DNS_FAL_UNREGISTER_CLASS_T;

#define DNS_FAL_UNREGISTER_CLASS_REQ_SIZE (sizeof(DNS_FAL_UNREGISTER_CLASS_T))

typedef struct DNS_FAL_PACKET_UNREGISTER_CLASS_REQ _Ttag
{
  TLR_PACKET_HEADER_T tHead; DNS_FAL_UNREGISTER_CLASS_T tData;
}
DNS_FAL_PACKET_UNREGISTER_CLASS_REG_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32	LFW: 0x00000020, LOM: DNS FAL	LFW: AP task, LOM: DNS FAL Task
ulLen	UINT32	8	Packet Data Length in bytes
ulCmd	UINT32	0x2D20	DNS_FAL_CMD_UNREGISTER_CLASS_REQ - Command
tData			
ulClass	UINT32		Class ID (according to "The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1" of the reference document [3] or Table 67: Predefined Values for the Class ID according to the CIP Specification.
ulAccessTyp	UINT32	0	Reserved unused, set to 0

Table 55: DNS_FAL_CMD_UNREGISTER_CLASS_REQ – Unregister a DeviceNet Class

4.6.3.2 Unregister Class Confirmation DNS_FAL_CMD_UNREGISTER_CLASS_CNF

Packet Structure Reference

```
Packet Structure Reference
/* DNS_FAL_CMD_UNREGISTER_CLASS_REQ Structure */
typedef struct DNS_FAL_UNREGISTER_CLASS_Ttag
{
  TLR_UINT32 ulClass;     /* Class identifier */
  TLR_UINT32 ulAccessTyp; /* Instance, Class access */
}
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
```

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32	LFW: 0x00000020, LOM: DNS FAL	LFW: AP task, LOM: DNS FAL Task
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x2D21	DNS_FAL_CMD_UNREGISTER_CLASS_CNF
tData			
ulClass	UINT32		Class ID (according to "The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1" of the reference document [3] or Table 67: Predefined Values for the Class ID according to the CIP Specification.
ulAccessTyp	UINT32	0	Reserved unused, set to 0

Table 56: DNS_FAL_CMD_UNREGISTER_CLASS_CNF – Unregister a DeviceNet Class

4.6.4 Remote Service

This packet indicates that the remote DeviceNet Master has requested a service from the Slave. The user receives the service indication only for classes that have been registered as in "Register Class" to underlying task. All service to the registered object will be indicated to the application except two services Get_Attribute_Single and Set_Attribute_Single. These services will be indicated with "Get / Set Attribute Indication", the user handles the service indication using "Get / Set Attribute Response".

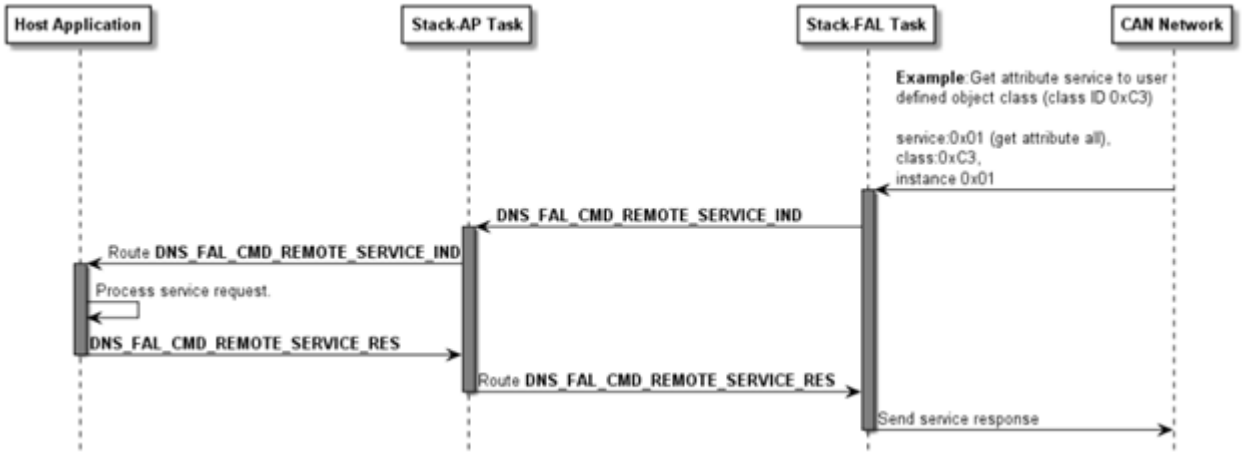


Figure 21: Sequence Diagram for the DNS_FAL_CMD_REMOTE_SERVICE_IND/RES Packet

The services can be selected by the DeviceNet Master by setting the parameter bServiceCode to the according service code as described in Table 65: Service Codes according to [3] Chapter 5, Table 5-1.1.

The response packet provides the generic and additional error numbers. Please refer to Table 66: Generic Error according to General Status Codes [3] Chapter B-1, Table B-1.1 for generic error. The additional error is normally null but in some cases it is not zero and depends on object profile. The user is recommended for further reading in reference [3] and [4].

4.6.4.1 Remote Service Indication DNS_FAL_CMD_REMOTE_SERVICE_IND

Packet Structure Reference

```
typedef struct DNS_FAL_REMOTE_SERVICE_IND_Ttag{
 TLR_UINT16 usClassId;
 TLR_UINT16 usInstanceId;
 TLR_UINT16 usReserved;
 TLR_UINT16 usSrvDatLen;
 TLR_UINT8 bServiceCode;
 TLR_UINT8 abReserved[3];
 TLR_UINT8 abSrvData[DNS_FAL_REMOTE_SERVICE_MAX_DATA];
} DNS_FAL_REMOTE_SERVICE_IND_T;

#define DNS_FAL_REMOTE_SERVICE_IND_SIZE (sizeof(DNS_FAL_REMOTE_SERVICE_IND_T) -
DNS_FAL_REMOTE_SERVICE_MAX_DATA)

typedef struct DNS_FAL_PACKET_REMOTE_SERVICE_IND_Ttag
{
 TLR_PACKET_HEADER_T tHead;
 DNS_FAL_REMOTE_SERVICE_IND_T tData;
} DNS_FAL_PACKET_REMOTE_SERVICE_IND_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	260	Packet Data Length in bytes
ulCmd	UINT32	0x2D22	`DNS_FAL_CMD_REMOTE_SERVICE_IND` - Command
tData			
usClassId	UINT16	1 … 0xFFFF	CIP Class ID having been addressed by the DeviceNet Master. (according to "The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1" of the reference document [3])
usInstanceId	UINT16	Valid instance	CIP Instance ID in this class having been addressed by the DeviceNet Master
usReserved	UINT16	Valid attribute	Reserved. Padding for Attribute Id.
usSrvDatLen	UINT16	1-512	Effective Length of Data in `abSrvData`
bServiceCode	UINT8	0-31	Service Code. See *Table 46*: Service Codes according to [3] Chapter 5, Table 5-1.1.
abReserved[3]	UINT8[]		Reserved. Padding for `bGenErrCode` and `bAddErrCode`
abSrvData[248]	UINT8[]		Array for Service Data

Table 57: DNS_FAL_CMD_REMOTE_SERVICE_IND – Remote Service Indication

4.6.4.2 Remote Service Response DNS_FAL_CMD_REMOTE_SERVICE_RES

Packet Structure Reference

```
typedef struct DNS_FAL_REMOTE_SERVICE_RES_Ttag
{
 TLR_UINT16 usClassId;
 TLR_UINT16 usInstanceId;
 TLR_UINT16 usReserved;
 TLR_UINT16 usSrvDatLen;
 TLR_UINT8 bServiceCode;
 TLR_UINT8 bGenErrCode;
 TLR_UINT8 bAddErrCode;
 TLR_UINT8 bReserved;
 TLR_UINT8 abSrvData[DNS_FAL_REMOTE_SERVICE_MAX_DATA];
} DNS_FAL_REMOTE_SERVICE_RES_T;
#define DNS_FAL_REMOTE_SERVICE_RES_SIZE (sizeof(DNS_FAL_REMOTE_SERVICE_RES_T) -
DNS_FAL_REMOTE_SERVICE_MAX_DATA)

typedef struct DNS_FAL_PACKET_REMOTE_SERVICE_RES_Ttag
{
 TLR_PACKET_HEADER_T tHead;
 DNS_FAL_REMOTE_SERVICE_RES_T tData;
} DNS_FAL_PACKET_REMOTE_SERVICE_RES_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	260	Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D23	DNS_FAL_CMD_REMOTE_SERVICE_RES - Command
tData			
usClassId	UINT16	1 … 0xFFFF	CIP Class ID having been addressed by the DeviceNet Master
usInstanceId	UINT16	Valid instance	CIP Instance ID in this class having been addressed by the DeviceNet Master
usReserved	UINT16	Valid attribute	Reserved. Padding for Attribute Id.
usSrvDatLen	UINT16	1-512	Effective Length of Data in abSrvData
bServiceCode	UINT8	0-31	Service Code. (14 in case of error response.) See Table 65: Service Codes according to [3] Chapter 5, Table 5-1.1.
bGenErrCode	UINT8		General error code. See Table 66: Generic Error according to General Status Codes [3] Chapter B-1, Table B-1.1
bAddErrCode	UINT8		Additional error code.
bReserved	UINT8		Reserved. Padding.
abSrvData[248]	UINT8[]		Array containing the Service Data to be sent to the DeviceNet Master

Table 58: DNS_FAL_CMD_REMOTE_SERVICE_RES – Remote Service Response

In order to get the indication when a service of an object is requested from remote client, the second-level application task (host application task or task above stack application task), the object class must be registered in stack task FAL task using DNS_FAL_CMD_REG_APP_REQ.

Note: If user wants to register an object class, it must be made sure that DNS_FAL_CMD_APP_REQ is sent and a positive confirmation of this command is received before.

4.6.5 Get / Set Attribute Service

If the explicit message channel between master and slave has been established correctly, the user application may receive unsolicited indications from the DeviceNet task as a defined range of accessible class IDs are routed by the device through the mailboxes, if they are accessed.

The Get/Set Attribute Service indication command signals the user application that a Get/Set attribute request was sent by a DeviceNet Master. The user is expected to provide a response to this command to acknowledge its receipt including return data. Otherwise, the user is endangered to be excluded from communication on the DeviceNet as usually Get- and Set Attribute commands are supervised by the master via a timer. The user gets the GetSet Indication only for **Get_Attribute_Single** and **Set_Attribute_Single.** All others service code will be indicated with *Remote Service* on page 75.

Note: To receive this command at the user application, the command `DNS_FAL_CMD_REGISTER_CLASS` must be used to allow the stack to forward this indication to user for each class.
Furthermore, `RCX_REGISTER_APP_REQ` is required to be able to receive indications. The device can only be an Explicit Message Server that means requested Get- and Set Attribute commands of the master device can only be answered passively. An active initiate of Get- and Set Attribute commands by the device cannot be performed.

After registering the classes the user program has to watch cyclically for incoming `DNS_FAL_GET_SET_ATT_IND` indication messages like described above and answer them by the corresponding answer message, see next section (Get / Set Attribute Response).

The various parameters have the following meaning:

The parameters `usClassId`, `usInstId` and `usAttId` are needed for correctly addressing the attribute to be accessed, i.e. for specifying which attribute of which instance of which object should be read or written.

- The range of the parameter `usClassId` is limited to the following numerical values 3 - 42 and 44-255.
- The parameter `usInstId` can be set to all defined instance values for the object chosen with `usClassId.`
- The parameter `usAttId` can be set to all defined attributes values for the chosen combination of `usClassId` and `usInstId`

The parameter `usDataCnt` indicates the number of bytes to read or write which may not exceed 240 bytes.

The parameter `bFunction` decides whether the master requests a read or write operation. The coding is as follows:

Symbolic constant	Assigned numeric value	Assigned function
DNS_FAL_GET_SET_ATT_WRITE	1	Write Function
DNS_FAL_GET_SET_ATT_READ	2	Read Function

Table 59: Allowed Values of bFunction Parameter

The field `abReserved[]` is irrelevant in this context and should be filled with zeroes therefore. This field has only been introduced for correct byte alignment.

The field abData contains the data to be written if the bFunction parameter has been set to DNS_FAL_GET_SET_ATT_WRITE.

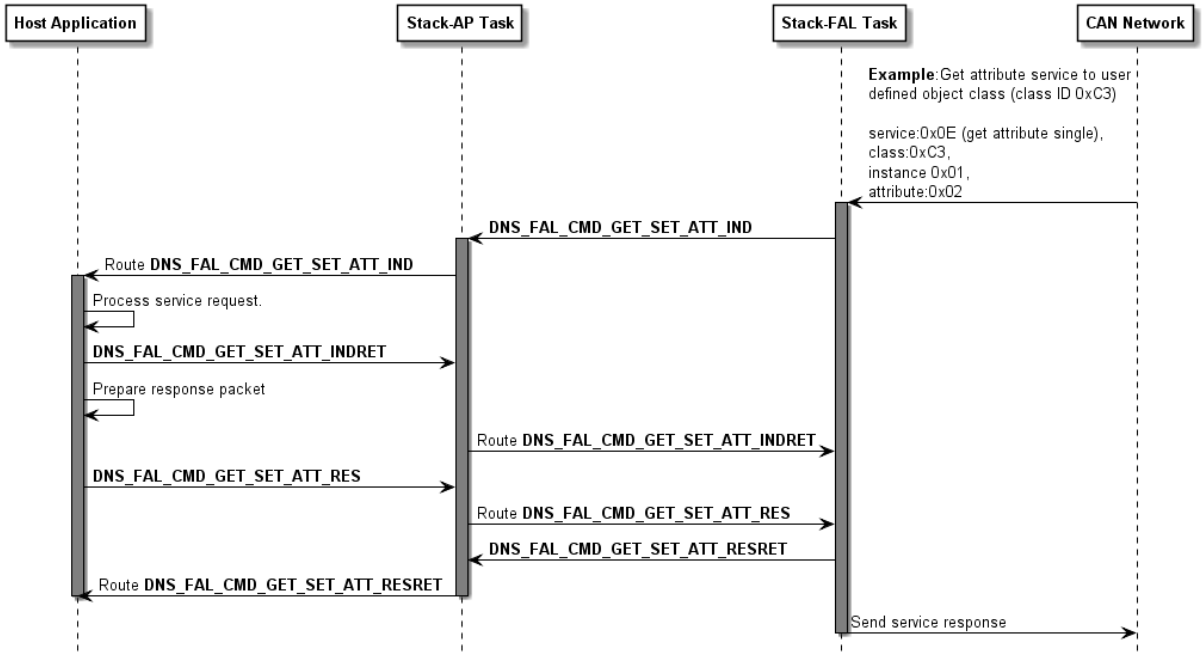


Figure 22: Sequence Diagram for the DNS_FAL_CMD_GET_SET_ATT_IND/RES Packet

4.6.5.1 The Get/Set Attribute Indication Packet

Packet Structure Reference

```
#define DNS_FAL_GET_SET_ATT_MAX_DATA  240

typedef struct DNS_FAL_GET_SET_ATT_IND_Ttag {
  TLR_UINT16  usClassId;
  TLR_UINT16  usInstId;
  TLR_UINT16  usAttId;
  TLR_UINT16  usDataCnt;
  TLR_UINT8   bFunction;
  TLR_UINT8   abReserved[3];
  TLR_UINT8   abData[DNS_FAL_GET_SET_ATT_MAX_DATA];
} DNS_FAL_GET_SET_ATT_IND_T;

typedef struct DNS_FAL_PACKET_GET_SET_ATT_IND_Ttag {
  TLR_PACKET_HEADER_T tHead;
     DNS_FAL_GET_SET_ATT_IND_T tData;
} DNS_FAL_PACKET_GET_SET_ATT_IND_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	12 … 252	
ulCmd	UINT32	0x2D16	`DNS_FAL_CMD_GET_SET_ATT_IND` - Command
tData			
usClassId	UINT16	3…42, 44...255	Class ID of user application object requested by Master. (according to *"The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1" of the reference document [3])*
usInstId	UINT16	0...255	Instance ID of user application object requested by Master. The instance ID in combination with class ID will form a unique identification of an object.
usAttId	UINT16	0...255	Attribute ID of user application object requested by Master. Pre-defined attributes of an object is described in details in object profiles of reference [3] and [4]. Users are also free to define new attribute if they implement the new object himself and the object is not pre-defined according to the *"The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1" of the reference document [3])*
usDataCnt	UINT16	0...240	Data Count of bytes to be read/written by the user application.
bFunction	UINT8	1,2	Function (Write/Read) 1: Write 2: Read
abReserved[3]	UINT8		Reserved
abData[240]	UINT8[]		`abData` array will contain information if the Write function is active.

Table 60: DNS_FAL_CMD_GET_SET_ATT_IND – Indication of Get/Set Attribute Event

4.6.5.2 The Get/Set Attribute Indication Return Packet

This packet is optional.

Packet Structure Reference

```
typedef __PACKED_PRE struct __PACKED_POST DNS_FAL_PACKET_GET_SET_ATT_INDRET_Ttag {
  TLR_PACKET_HEADER_T tHead;
} DNS_FAL_PACKET_GET_SET_ATT_INDRET_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	0	
ulCmd	UINT32	0x2D17	DNS_FAL_CMD_GET_SET_ATT_INDRET - Command

Table 61: DNS_FAL_CMD_GET_SET_ATT_IND – Indication of Get/Set Attribute Event

4.6.5.3 Get / Set Attribute Response

The user application may receive unsolicited indications from the DNS task. The Get / Set Attribute Service signals the user application that a Get/Set attribute request was sent by a DeviceNet Master. As discussed above, the user must then provide a response to this command in order to acknowledge its receipt including return data and to avoid the occurrence of a timeout at the master with negative consequences such as possible exclusion from the DeviceNet network.

That means a timeout error will be generated on the master side if a defined time has expired and no answer was received by the DeviceNet master during this period. Because it is now on the user application to send back the corresponding answer message so that the slave device can route it back to the master, the user application should not waste time and send back the message immediately after having finished up the analysis of the indication message.

The various parameters of this packet have the following meaning:

The parameters `usClassId`, `usInstId` and `usAttId` are needed for correctly addressing the attribute to be accessed, i.e. for specifying which attribute of which instance of which object should be read or written. Use the same values as in the indication packet.

- The range of the parameter `usClassId` is limited to the following values: 3-42 and 44-255.
- The parameter `usInstId` can be set to all defined instance values for the object chosen with `usClassId.`
- The parameter `usAttId` can be set to all defined attributes values for the chosen combination of `usClassId` and `usInstId`

The parameter `bFunction` depends whether a read or write request has been issued within the indication. The coding is as follows:

Symbolic constant	Assigned numeric value	Assigned function
DNS_FAL_GET_SET_ATT_WRITE	1	Write Function
DNS_FAL_GET_SET_ATT_READ	2	Read Function

Table 62: bFunction Parameter

`usDataCnt` indicates the number of bytes to read or write which may not exceed 240 bytes.

- For read access, set it to the value delivered in the indication.
- For write access (`DNS_FAL_GET_SET_ATT_WRITE`, see just below), you should set this value to 0.

`bReserved` should be zero. It has been introduced for correct byte alignment."

`TLR_E_DNS_FAL_GET_STATUS_INVALID_STATUS (0xC0620014L)`

which might otherwise occur. This field has only been introduced for correct byte alignment.

The field `abData` contains the data to be transferred to the master if the `bFunction` parameter has been set to `DNS_FAL_GET_SET_ATT_READ`.

These data should be exactly the contents of the requested attribute belonging to the class and instance that has been specified in the indication.

Additionally, the user application has the possibility to signal back access errors back to master, for example if a requested attribute does not exist. This must be done in the `bGenErrCode` parameter then. See the following table for possible error numbers. DeviceNet allows sending back a general error code and an additional error code in the error response message. All errors in

the next table will result an additional error code of 255 except the error value 31. If this error is returned, you can define an additional error code at the bAddErrCode parameter, which is sent back to the master together with the general error code 31.

The bGenErrCode parameter contains an error code which is determined by the CIP specification, see "Table 63: DNS_FAL_CMD_GET_SET_ATT_RES – bGenErrCode General Error Codes" below or "*The CIP Networks Library, Volume 1, Appendix B,. General Status Codes*", especially table B-1.1 "*CIP General Status Codes"* of the reference [3].

Definition	Value	Description
DNS_FAL_GET_SET_NO_ERROR	0x00	No error
DNS_FAL_GET_SET_RES_NOT_AVAIL	0x02	Resources unavailable, Class ID invalid
DNS_FAL_GET_SET_SERV_NOT_AVAIL	0x08	Service not available, Read or Write not supported to Class
DNS_FAL_GET_SET_ATTRIB_BAD_VALUE	0x09	Invalid attribute value, attribute not supported
DNS_FAL_GET_SET_ATTRIB_NO_SET	0x0E	Attribute not settable, attribute has no write permission.
DNS_FAL_GET_SET_ACCESS_DENIED	0x0F	Access denied, general error to deny access
DNS_FAL_GET_SET_STATE_CONFLICT	0x10	State conflict, host state prohibits the command execution
DNS_FAL_GET_SET_NOT_ENOUGH_DATA	0x13	Not enough data received, master has send too less data
DNS_FAL_GET_SET_ATTRIB_NOT_SUP	0x14	Attribute not supported
DNS_FAL_GET_SET_TOO_MUCH_DATA	0x15	Too much data received, master has send too much data
DNS_FAL_GET_SET_VENDOR_SPEC_CODE	0x1F	Vendor specific error code. An addition error code can be placed in bAddErrCode.

Table 63: DNS_FAL_CMD_GET_SET_ATT_RES – bGenErrCode General Error Codes

Packet Structure Reference

```
typedef struct DNS_FAL_GET_SET_ATT_RES_Ttag
{
  TLR_UINT16    usClassId;
  TLR_UINT16    usInstId;
  TLR_UINT16    usAttId;
  TLR_UINT16    usDataCnt;
  TLR_UINT8     bFunction;
  TLR_UINT8     bGenErrCode;
  TLR_UINT8     bAddErrCode;
  TLR_UINT8     bReserved;
  TLR_UINT8     abData[DNS_FAL_GET_SET_ATT_MAX_DATA];
} DNS_FAL_GET_SET_ATT_RES_T;
typedef struct DNS_FAL_PACKET_GET_SET_ATT_RES_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_GET_SET_ATT_RES_T tData;
} DNS_FAL_PACKET_GET_SET_ATT_RES_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	sizeof (abData)+12	Value depends on the number of bytes transmitted in abData. The minimum value is 12, the maximum value is 252.
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D14	DNS_FAL_CMD_GET_SET_ATT_RES – Command
tData			
usClassId	UINT16	3…42, 44...255	Class ID (according to *"The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1" of the reference document [3])*
usInstId	UINT16	0-255	Instance ID of user application object. The instance ID in combination with class ID will form a unique identification of an object.
usAttId	UINT16	0-255	Attribute ID of user application object requested by Master. Pre-defined attributes of an object is described in details in object profiles of reference [3] and [4]. Users are also free to define new attribute if they implement the new object himself and the object is not pre-defined according to the *"The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1" of the reference document [3])*
usDataCnt	UINT16	0-240	Data Count of bytes to be read/written by the user application.
bFunction	UINT8	1,2	Which service has been performed (Write/Read) 1: Write 2: Read
bGenErrCode	UINT8	0-255	General error code . See Table 47: Generic Error
bAddErrCode	UINT8	0-255	DeviceNet Additional error code.
bReserved	UINT8		Reserved, set to zero
abData[240]	UINT8[]		Data will contain information if the READ function is active.

Table 64: DNS_FAL_CMD_GET_SET_ATT_RES – Get/Set Attribute Response

4.6.5.4 The Get/Set Attribute Response Return Packet

Packet Structure Reference

```
typedef struct DNS_FAL_GET_SET_ATT_RESRET_Ttag {
  TLR_UINT16   usClassId;
  TLR_UINT16   usInstId;
  TLR_UINT16   usAttId;
  TLR_UINT16   usDataCnt;
  TLR_UINT8    bFunction;
  TLR_UINT8    bGenErrCode;
  TLR_UINT8    bAddErrCode;
  TLR_UINT8    bReserved;
} DNS_FAL_GET_SET_ATT_RESRET_T;

typedef struct DNS_FAL_PACKET_GET_SET_ATT_RESRET_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_GET_SET_ATT_RESRET_T tData;
} DNS_FAL_PACKET_GET_SET_ATT_RESRET_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	0	
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D15	DNS_FAL_CMD_GET_SET_ATT_RESRET - Command
tData			
usClassId	UINT8	3…42, 44...255	Class ID (according to *"The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1" of the reference document [3]). It can also be a new user-defined object.*
usInstId	UINT8	0-255	Instance ID of user application object. The instance ID in combination with class ID will form a unique identification of an object.
usAttId	UINT8	0-255	Attribute ID of user application object requested by Master. Pre-defined attributes of an object is described in details in object profiles of reference [3] and [4]. Users are also free to define new attribute if they implement the new object himself and the object is not pre-defined according to the *"The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1" of the reference document [3])*
usDataCnt	UINT8	0-240	Data Count of bytes to be returned by the user application.
bFunction	UINT8	1,2	Which service has been performed DNS_FAL_GET_SET_ATT_WRITE = Write Function DNS_FAL_GET_SET_ATT_READ = Read Function
bGenErrCode	UINT8	0-255	Returned General Error code sent by the application. See Table 47: Generic Error
bAddErrCode	UINT8	0-255	Returned Additional Error code sent by the application.
bReserved	UINT8	0-255	Reserved

Table 65: DNS_FAL_CMD_GET_SET_ATT_RESRET – Get/Set Attribute Response Return

4.7 Modify Firmware Parameter

The host application sends `HIL_SET_FW_PARAMETER_REQ` to set firmware configuration parameters, respectively.

The DeviceNet Slave stack supports the following parameters to modify:

ParameterID	Name	Type	Description
PID_STATION_ADDRESS (0x30000001)	MAC address	UINT32	MAC address of device 0 … 63
PID_BAUDRATE (0x30000002)	Baudrate	UINT32	Baud rate of device 0 = 125 kBit/s 1 = 250 kBit/s 2 = 500 kBit/s

Table 66 HIL_SET_FW_PARAMETER_REQ ParameterID

This is a generic packet, which is not specific to the DeviceNet Slave protocol stack. Therefore, it is not fully covered by this document. Please refer to reference [2] for further information.

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 Communication state

The common status block includes the communication state. This section describes how the communication state is used by DeviceNet Slave Device.

5.1.1.1 Implementation from V2.5.2.0

Starting with DeviceNet Salve V2.5.2.0 the default handling of communication state is as described in the following table:

State	Description
OFFLINE	XC might be loaded and started (After set config and channel initialization).
STOP	"Bus ON" must be set by application to reach this state". The device is configured and is visible for other devices in the system. Eplicit messaging is possible. Connection establishment requests will be accepted by the device. The device has no open connections (poll, bit strobe, change of state)
IDLE	n.a. (Note: this state is not used at all)
OPERATE	The device has at least one established connection (poll, bit strobe, change of state).

Table 67: Communication State (V2.5.2.0 and later)

5.1.1.2 Legacy Implementation (V2.5.1.0 and earlier)

Implementations of version V2.5.1.0 and earlier behave as follows.

State	Description
OFFLINE	No valid configuration. Error occurs.
STOP	No I/O data exchange.
IDLE	n.a. (Note: this state is not used at all)
OPERATE	I/O data exchange was established at least once. Note: If connection is interrupted the state is not leaved to STOP again.

Table 68: Communication State (V2.5.1.0 and earlier)

If needed, the legacy handling can be enabled by setting the AP task startup parameter `DNSAP_STARTUPFLAG_ENABLE_DPM_NETWORK_STATE_LEGACY_BEHAVIOR` in firmware/stack V2.5.2.0 and later.

Note: Legacy behavior could be also enable using Taglist Editor.

The following figure shows the tag to enable the legacy handling of the communication state.

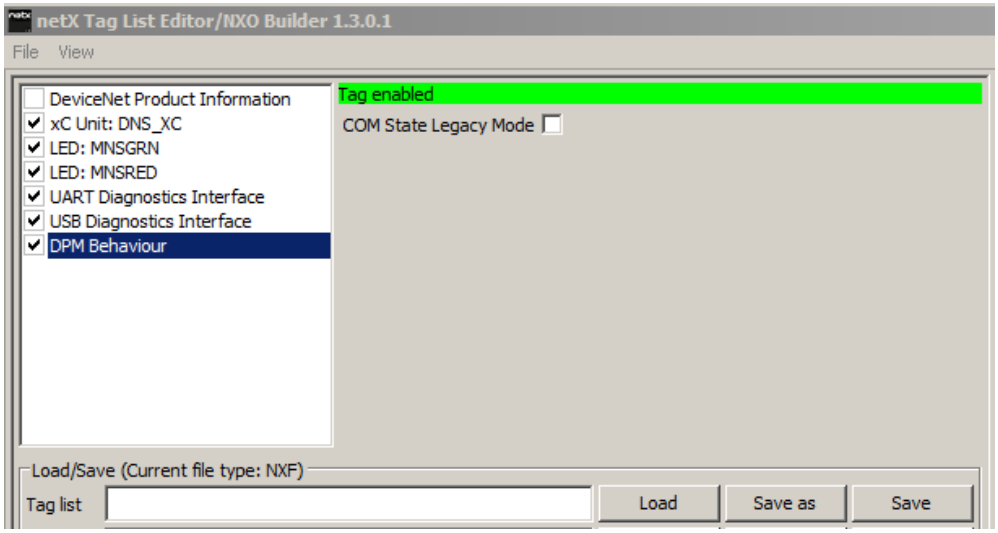


Figure 23: Tag to enable legacy handling of the communication state

5.2 Extended status

For the DeviceNet Slave protocol implementation, the Extended Status Area is structured as DNS_FAL_GEN_STATUS_T. For details, see section *Get Status Service* on page 101.

5.3 Status in the input data area

The status information ulDataSta can be added to the input data (consumed data). To activate the ulDataSta, the flag MSK_DNS_CFG_FLAG_ENABLE_DATA_STATUS has to be set to 1 using the variable ulConfigFlags of the *Configuration sequence (Loadable Firmware)* (page 22).

Value ulDataSta	Meaning
0x00000000 (DNS_FAL_DS_ZERO)	Data are in safe state zero
0x00000001 (DNS_FAL_DS_RECV_RUN)	Data are valid
0x00000002 (DNS_FAL_DS_RECV_IDLE)	Data are in receive idle mode
0x00000003 (DNS_FAL_DS_RECV_IDLE_ZERO)	Data are in receive idle zero
0x00000004 (DNS_FAL_DS_HOLD_LAST_STATE)	Data are in safe state hold last state
0x00000005 (DNS_FAL_DS_USER_STATE)	Data are in safe state user defined
0x00020000 (MSK_DNS_STA_FLAG_ERR_BUS_OFF)	Bus Off

Table 69: Values of ulDataSta

6 Linkable Object Module (LOM)

6.1 Accessing the Protocol Stack by Programming the AP Task's Queue

6.1.1 Getting the Receiver Task Handle of the Process Queue

To get the handle of the process queue of one of the various tasks the Macro TLR_QUE_IDENTIFY() needs to be used. This macro delivers a pointer to the handle of the intended queue to be accessed (which is returned within the third parameter, phQue), if you provide it with the name of the queue (and an instance of your own task). The correct ASCII-queue name for accessing a task which you have to use as current value for the first parameter (pszIdn) is

ASCII Queue name	Description
"QUE_DNS_FAL"	Name of the DNS_FAL task process queue
"QUE_DNSAP"	Name of the DNS_AP task process queue
"CAN_DL_QUE"	Name of the CAN_DL task process queue

Table 70: Queue Names used in DeviceNet Slave

The returned handle has to be used as value ulDest in all initiator packets the AP task intends to send to the DNS_FAL task. This handle is the same handle that has to be used in conjunction with the macros like TLR_QUE_SENDPACKET_FIFO/LIFO() for sending a packet to the respective task.

6.2 Services for the use case of Linkable Object Modules

6.2.1 Configuration sequence (Linkable Object Module)

To configure the DeviceNet Slave Stack the following packets are necessary:

Packet Name	Command Code (REQ/CNF)	Page
Stack Register Application Service, DNS_FAL_CMD_APP_REQ/CNF	0x2D00 0x2D01	92
Init Stack Service, DNS_FAL_CMD_INIT_STACK_REQ/CNF	0x2D02 0x2D03	95
Set Mode Service, DNS_FAL_CMD_SET_MODE_REQ/CNF	0x2D06 0x2D07	98

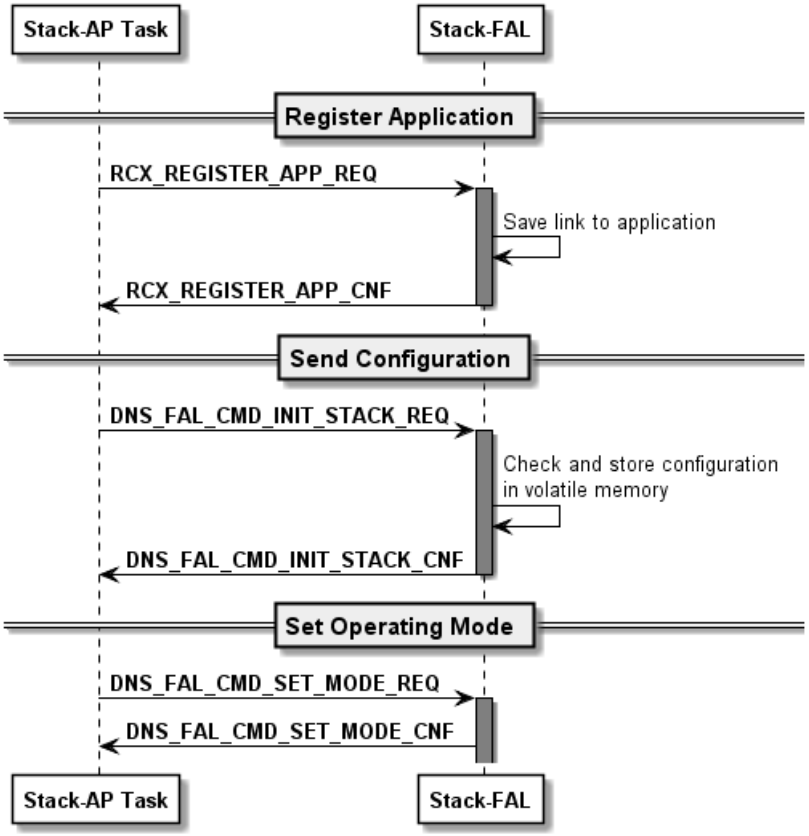


Figure 24: Configuration Sequence Using the Extended Packet Set

6.2.2 Stack Register Application Service

The user application must pass the DeviceNet Slave stack a handle to its queue so unsolicited messages from the stack can be processed and acted upon. Registering the application queue handle must be done first before proceeding with the stack initialization. This queue handle will be used by the stack to notify the user application of GET/SET attribute requests as well as BUS OFF and other events. Details of this command are as follows.

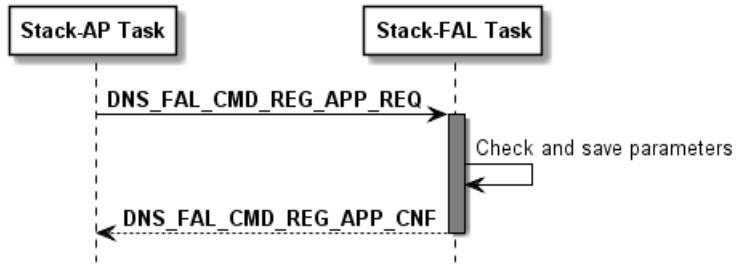


Figure 25: Sequence Diagram for the DNS_FAL_CMD_APP_REQ/CNF Packet

6.2.2.1 Stack Register Application Request

The stack register application request only has one parameter ulMode, which allows the user to choose which event will be forwarded to the stack application task.

Packet Structure Reference

```
/* Event mask for parameter ulMode */
#define DNS_FAL_EVENT_IO_UPDATE_IND            0x00000001
#define DNS_FAL_EVENT_LED_STATE_IND            0x00000002
#define DNS_FAL_EVENT_IO_CHANGED_IND           0x00000004

typedef struct DNS_FAL_REG_APP_REQ_Ttag {
  TLR_UINT32  ulMode;
} DNS_FAL_REG_APP_REQ_T;

typedef struct DNS_FAL_PACKET_REG_APP_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_REG_APP_REQ_T tData;
} DNS_FAL_PACKET_REG_APP_REQ_T;
```

The parameter `ulMode` is a bit field to enable selective different events to indicate to the AP task.

Following events can be activated separately or together:

- DNS_FAL_EVENT_IO_UPDATE_IND

  When this event is activated, then the stack task will send every time an indication message when the master has refreshed the output data to the slave.

- DNS_FAL_EVENT_LED_STATE_IND

  When this event is activated, then the stack task sends every time an indication message when LED operation mode has changed. This message can be used by the application task to control a Network Status LED.

- DNS_FAL_EVENT_IO_CHANGED_IND

  When this event is activated, then the stack task will send every time an indication message when the master has refreshed the output data and the new output data are different then the old one (change of state).

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32	LOM: DNS FAL	LOM: DNS FAL Task
ulLen	UINT32	4	sizeof(DNS_FAL_REG_APP_REQ_T)
ulCmd	UINT32	0x2D00	DNS_FAL_CMD_REG_APP_REQ - Command
tData			
ulMode	UINT32	0x00000001 0x00000002 0x00000004	Bitfield to enable selective different events to indicate to the AP task. DNS_FAL_EVENT_IO_UPDATE_IND DNS_FAL_EVENT_LED_STATE_IND DNS_FAL_EVENT_IO_CHANGED_IND

Table 71: DNS_FAL_CMD_REG_APP_REQ – Request Command for DNS Get Status

Source Code Example

```
TLR_RESULT DnsApp_RegApp_Req(DNS_APP_RSC_T FAR* ptRsc, UINT32 ulMode )
{
  TLR_RESULT eRslt;
  DNS_FAL_PACKET_REG_APP_REQ_T  *ptRegAppReq;

  eRslt = TLR_POOL_PACKET_GET(ptRsc->tLoc.hPool, &ptRegAppReq);
  if( eRslt == TLR_S_OK )
  {
    TLR_QUE_LINK_SET_PACKET_SRC(ptRegAppReq,ptRsc->tLoc.tQueSrcDnsApp);
    ptRegAppReq->tHead.ulDst    = ptRsc->tLoc.hDstQueDnsTsk;
    ptRegAppReq->tHead.ulDestId = 0;
    ptRegAppReq->tHead.ulCmd    = DNS_FAL_CMD_REG_APP_REQ;
    ptRegAppReq->tHead.ulSta    = 0;
    ptRegAppReq->tHead.ulExt    = 0;
    ptRegAppReq->tHead.ulRout   = 0;
    ptRegAppReq->tHead.ulLen    = 4;

      /* Set the mode not used currently */
      ptRegAppReq->tData.ulMode = 0;

    eRslt = TLR_QUE_SENDPACKET_FIFO(ptRsc->tLoc.tDnsFalQue,ptRegAppReq,TLR_FINITE);
    if( eRslt != TLR_S_OK )
    {
      TLR_POOL_PACKET_RELEASE(ptRsc->tLoc.hPool, ptRegAppReq);
    }
     }
     return eRslt;
}
```

6.2.2.2 Stack Register Application Confirmation

The confirmation packet of stack register application service contains no parameter except the error code which has been preset by variable `ulSta` in the packet header.

Packet Structure Reference

```
typedef struct DNS_FAL_PACKET_REG_APP_CNF_Ttag {
  TLR_PACKET_HEADER_T tHead;
} DNS_FAL_PACKET_REG_APP_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	0	
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D01	DNS_FAL_CMD_REG_APP_CNF - Command

Table 72: DNS_FAL_CMD_REG_APP_CNF – Confirmation of DNS Get Status

6.2.3 Init Stack Service

The packet DNS_FAL_CMD_INIT_STACK_REQ can be used to configure the DeviceNet Slave stack from the application task running on netX (Linkable Module scenario). The stack checks the parameters which are sent with the request. If all parameters are valid the DeviceNet Slave stack will directly take over the configuration.

The following applies for this packet:

- Configuration parameters will be checked and become effective directly.
- In case of any error no data will be stored at all.
- Configuration parameters are rejected, if the DeviceNet Slave protocol stack is already configured. In this case the stack must be reset.

Note: The data portion of the command DNS_FAL_CMD_INIT_STACK_REQ is exactly the same as that of the command DNS_AP_CMD_SET_CONFIGURATION_REQ. The differences are in the command number and the behavior when the parameters become effective to the network. The parameters which are send with DNS_FAL_CMD_INIT_STACK_REQ become effective immediately. The parameters set with the command DNS_AP_CMD_SET_CONFIGURATION_REQ become effective after a following “Channel Init”.

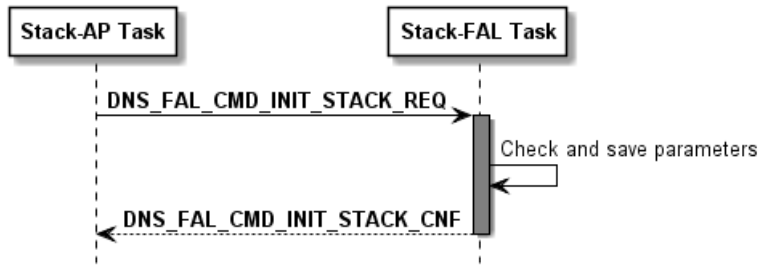


Figure 26: Sequence Diagram for the DNS_AP_CMD_INIT_STACK _REQ/CNF Packet

6.2.3.1 Init Stack Request DNS_FAL_CMD_INIT_STACK_REQ

The application has to send this request to the protocol stack.

Packet Structure Reference

```
/*      System flags                  */
#define MSK_DNS_SYS_FLG_ADR_SW_ENABLE                                0x00000010
#define MSK_DNS_SYS_FLG_BAUD_SW_ENABLE                               0x00000020
#define MSK_DNS_SYS_FLG_RESERVED                                     0xFFFFFFCF

/*                             DNS Baudrates                                 */
#define DNS_FAL_BAUDRATE_500kB                                       0
#define DNS_FAL_BAUDRATE_250kB                                       1
#define DNS_FAL_BAUDRATE_125kB                                       2

/*        Enable flags              */
#define MSK_DNS_ENABLE_VENDORID                                      0x00000001
#define MSK_DNS_ENABLE_PRODUCTTYPE                                   0x00000002
#define MSK_DNS_ENABLE_PRODUCTCODE                                   0x00000004
#define MSK_DNS_ENABLE_MAJORMINORREV                                 0x00000008
#define MSK_DNS_ENABLE_SERIALNR                                      0x00000010
#define MSK_DNS_ENABLE_PRODUCTNAME                                   0x00000020
#define MSK_DNS_ENABLE_RESERVED                                      0xFFFFFFC0

/*      Configuration flags           */
#define MSK_DNS_CFG_FLAG_IGNORE_ADDR_SWITCH                          0x00000001
#define MSK_DNS_CFG_FLAG_CONTINUE_ON_BUSOFF                          0x00000002
#define MSK_DNS_CFG_FLAG_CONTINUE_ON_LOSS_NP                         0x00000004
#define MSK_DNS_CFG_FLAG_RECVIDLE_CLEAR_DATA                         0x00000008
#define MSK_DNS_CFG_FLAG_RECVIDLE_USER_DATA                          0x00000010
#define MSK_DNS_CFG_FLAG_24VDCINVERT                                 0x00000020
#define MSK_DNS_CFG_FLAG_ENABLE_SET_PRODCONS_SIZE_REMOTE             0x00000040
#define MSK_DNS_CFG_FLAG_ENABLE_SET_MACID_REMOTE                     0x00000080
#define MSK_DNS_CFG_FLAG_ENABLE_SET_BAUDRATE_REMOTE                  0x00000100
#define MSK_DNS_CFG_FLAG_ENABLE_DATA_STATUS                          0x00000200
#define MSK_DNS_CFG_FLAG_RESERVED                                    0xFFFFFC00

typedef struct DNS_FAL_INIT_STACK_REQ_Ttag {
  TLR_UINT32 ulSystemFlags;
  TLR_UINT32 ulWdgTime;
  TLR_UINT32 ulNodeId;
  TLR_UINT32 ulBaudrate;
  TLR_UINT32 ulProducedSize;
  TLR_UINT32 ulConsumedSize;
  TLR_UINT32 ulConfigFlags;
  TLR_UINT32 ulEnableFlags;
  TLR_UINT16 usVendorId;
  TLR_UINT16 usProductType;
  TLR_UINT16 usProductCode;
  TLR_UINT8  bMinorRev;
  TLR_UINT8  bMajorRev;
  TLR_UINT32 ulSerialNumber;
  TLR_UINT8  abReserved[3];
  TLR_UINT8  bProductNameLen;
  TLR_UINT8  abProductName[32];
} DNS_FAL_INIT_STACK_REQ_T;

typedef struct DNS_FAL_PACKET_INIT_STACK_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_INIT_STACK_REQ_T tData;
} DNS_FAL_PACKET_INIT_STACK _REQ_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32	LOM: DNS FAL	LOM: DNS FAL Task
ulLen	UINT32	80	sizeof(DNS_FAL_INIT_STACK_REQ_T)
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D02	DNS_FAL_CMD_INIT_STACK_REQ - Command
tData			
See Table 12: DNS_AP_CMD_SET_CONFIGURATION_REQ_T – Set Configuration Request			

Table 73: DNS_FAL_CMD_INIT_STACK_REQ – Request Command for DNS Stack Initialization

6.2.3.2 Init Stack Confirmation DNS_FAL_CMD_INIT_STACK_CNF

This confirmation will be returned to the application.

Packet Structure Reference

```
typedef struct DNS_FAL_INIT_STACK_CNF_Ttag {
  TLR_HANDLE hPdOutTrpBuf;
  TLR_HANDLE hPdInTrpBuf;
} DNS_FAL_INIT_STACK_CNF_T;

typedef struct DNS_FAL_PACKET_INIT_STACK_CNF_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_INIT_STACK_CNF_T tData;
} DNS_FAL_PACKET_INIT_STACK_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	8	sizeof(DNS_FAL_INIT_STACK_REQ_T)
ulSta	UINT32		ulSta = 0 Initialization OK ulSta != 0 Initialization failed See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D03	DNS_FAL_CMD_INIT_STACK_CNF - Command
tData			
hPdOutTrpBuf	TLR_HANDLE		Handle for Triple Buffer for Output
hPdInTrpBuf	TLR_HANDLE		Handle for Triple Buffer for Input

Table 74: DNS_FAL_CMD_INIT_STACK_CNF – Confirmation Command for DNS Stack Initialization

The confirmation packet for stack Initialization returns in the status, whether the stack could successfully be initialized, or not.

6.2.4 Set Mode Service

This packet describes how to set a different mode of operation. The modes are coded as followings:

Operation mode	Numerical value	Description
DNS_FAL_MODE_OFFLINE	0	This mode will set the DeviceNet Stack into a so called Network access state “OFFLINE”. According the “General Network Access State machine” which is defined in the DeviceNet specification, volume 3 chapter 2, section 2-3 of the reference [4], the slave will be from Network point of view not visible and accessible.
DNS_FAL_MODE_STOP	1	Not supported for feature use. (do not use)
DNS_FAL_MODE_IDLE	2	Not supported for feature use. (do not use)
DNS_FAL_MODE_RUN	3	This mode makes the DeviceNet Slave accessible from network point of view. In this state the slave has performed the Duplicate MAC ID procedure which is defined in the DeviceNet norm. In this state the DeviceNet Slave is able to accept IO connection from a master.

Table 75: DeviceNet Operation Modes

The corresponding confirmation packet informs whether the mode could

- be changed as desired, in this case the same value of `ulMode` as requested will be present there.
- not be changed as desired, in this case a different value of `ulMode` as the one requested will be present there indicating the actually active mode of operation.

The packet can be send from the AP application task or host application application (host task) figured as below:

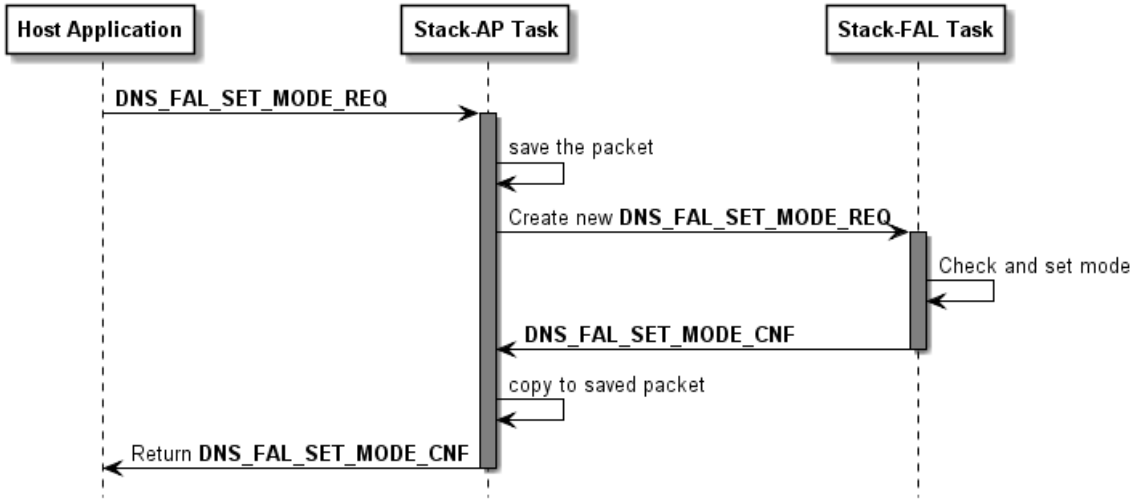


Figure 27: Sequence Diagram for the DNS_FAL_SET_MODE_REQ/CNF Packet

6.2.4.1 Set Mode Request

The application has to send this request to the protocol stack.

Packet Structure Reference

```
typedef struct DNS_FAL_SET_MODE_REQ_Ttag {
  TLR_UINT32  ulMode;
  TLR_UINT32 ulInfo;
} DNS_FAL_SET_MODE_REQ_T;

typedef struct DNS_FAL_PACKET_SET_MODE_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_SET_MODE_REQ_T tData;
} DNS_FAL_PACKET_SET_MODE_REQ_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32	LOM: DNS FAL	LOM: DNS FAL Task
ulLen	UINT32	4	Packet Data Length in bytes
ulCmd	UINT32	0x2D06	DNS_FAL_CMD_SET_MODE_REQ - Command
tData			
ulMode	UINT32	0-3	Mode to set, see above
ulInfo	UINT32		Extra information. Set to zero when sent from host application Task

Table 76: DNS_FAL_CMD_SET_MODE_REQ – Request Command for Setting Operation Mode

6.2.4.2 Set Mode Confirmation

This confirmation will be returned to the application.

Packet Structure Reference

```
typedef struct DNS_FAL_SET_MODE_CNF_Ttag {
  TLR_UINT32 ulMode;
  TLR_UINT32 ulInfo;
} DNS_FAL_SET_MODE_CNF_T;

typedef struct DNS_FAL_PACKET_SET_MODE_CNF_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_SET_MODE_CNF_T tData;
} DNS_FAL_PACKET_SET_MODE_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	4	Packet Data Length in bytes
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D07	DNS_FAL_CMD_SET_MODE_CNF - Command
tData			
ulMode	UINT32	0-3	Mode that has been actually set.
ulInfo	UINT32		Extra Information, contains no information in confirmation packet.

Table 77: DNS_FAL_CMD_SET_MODE_CNF – Request Command for Setting Operation Mode

6.2.5 Get Status Service

The DNS Task can provide status information referring to its current operational state when asked by the DNS_FAL_CMD_GET_STATUS_REQ/CNF command. The answer provided in the confirmation packet contains exactly the extended status block informing about CAN transmissions and Init parameters (such as baud rate or own CAN address) entered during startup. The description of the returned data is given in section *Extended status* on page 89. This command can be used at anytime after the initialization of the queue. The returned status information is delivered in array abData[..] of the confirmation packet. This contains the data of structure DNS_FAL_GEN_STATUS_T.

Parameter	Meaning
ulStatusFlags	Collective flag field to indicate several statuses information. See 6.2.5.2 Get Status Confirmation.
ulRxInt	Counter for received CAN telegrams
ulTxInt	Counter for transmitted CAN telegrams
ulRxOverRun	Counter for detected receive overruns
ulTxOverRun	Counter for detected transmission overrun events.
ulTxAborts	Counter for transmission abort events.
ulErrorInt	Counter for low transmission quality. A certain limit has been exceeded.
usBusOffCnt	Counter for bus off events.
usResetCnt	Reset counter.
ulNodeId	Own address of the DeviceNet Slave device in range 0..63.
ulBaudrate	Chosen baud rate. See *Table 16: Baud rates and values* on page 30.
usVendorId	Vendor ID value
usProducedSize	Number of master input bytes that are produced.
usConsumedSize	Number of master output bytes that are consumed.
usReserved	Reserved for future use

Table 78: Meaning and allowed Values for Status-Parameters

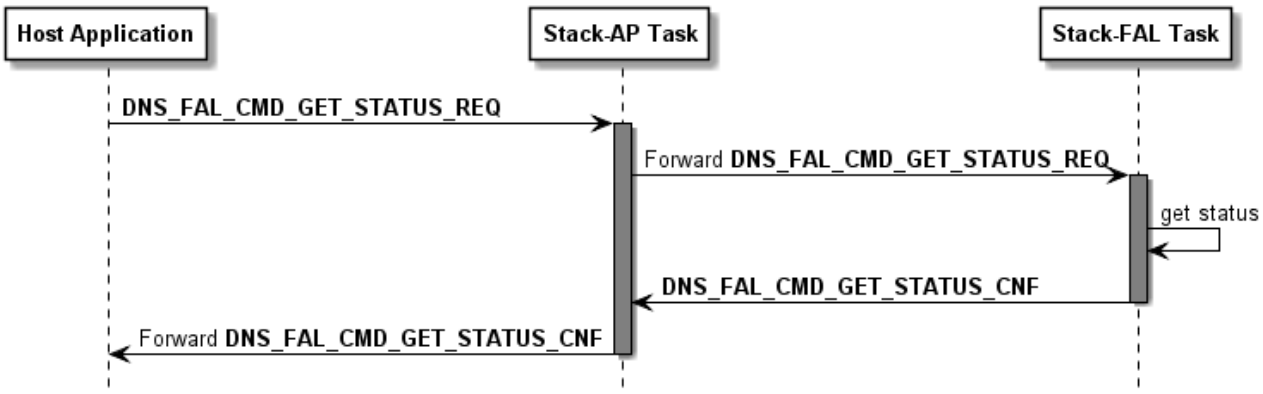


Figure 28: Sequence Diagram for the DNS_FAL_CMD_GET_STATUS_REQ/CNF Packet

6.2.5.1 Get Status Request

Packet Structure Reference

```
typedef struct DNS_FAL_GET_STATUS_REQ_Ttag {
  TLR_UINT16  usArea;
  TLR_UINT16  usSubArea;
} DNS_FAL_GET_STATUS_REQ_T;

typedef struct DNS_FAL_PACKET_GET_STATUS_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_GET_STATUS_REQ_T tData;
} DNS_FAL_PACKET_GET_STATUS_REQ_T;
```

Packet Description

Variable	Type	Value / Range	Description
ulDest	UINT32	LOM: DNS FAL	LOM: DNS FAL Task
ulLen	UINT32	4	sizeof(DNS_FAL_GET_STATUS_REQ_T)
ulCmd	UINT32	0x2D04	DNS_FAL_CMD_GET_STATUS_REQ - Command
tData			
usArea	UINT16	0	Reserved for future use, set to zero
usSubArea	UINT16	0	Reserved for future use, set to zero

Table 79: DNS_FAL_CMD_GET_STATUS_REQ – Request Command for DNS Get Status

6.2.5.2 Get Status Confirmation

This confirmation will be returned to the application.

Packet Structure Reference

```
typedef struct DNS_FAL_GEN_STATUS_T {
  TLR_UINT32 ulStatusFlags;
  TLR_UINT32 ulRxInt;
  TLR_UINT32 ulTxInt;
  TLR_UINT32 ulRxOverRun;
  TLR_UINT32 ulTxOverRun;
  TLR_UINT32 ulTxAborts;
  TLR_UINT32 ulErrorInt;
  TLR_UINT32 ulBusOffCnt;
  TLR_UINT32 ulResetCnt;
  TLR_UINT32 ulNodeId;
  TLR_UINT32 ulBaudrate;
  TLR_UINT16 usVendorId;
  TLR_UINT16 usProducedSize;
  TLR_UINT16 usConsumedSize;
  TLR_UINT16 usReserved;
} DNS_FAL_GEN_STATUS_T;

#define DNS_FAL_GET_STATUS_CNF_MAX_DATA (256)
typedef struct DNS_FAL_GET_STATUS_CNF_Ttag {
  TLR_UINT16  usArea;
  TLR_UINT16  usSubArea;
  TLR_UINT8   abData[DNS_FAL_GET_STATUS_CNF_MAX_DATA];
} DNS_FAL_GET_STATUS_CNF_T;

#define DNS_FAL_GET_STATUS_CNF_SIZE (sizeof(DNS_FAL_GET_STATUS_CNF_T)-
DNS_FAL_GET_STATUS_CNF_MAX_DATA)

typedef struct DNS_FAL_PACKET_GET_STATUS_CNF_Ttag {
  TLR_PACKET_HEADER_T tHead;
  DNS_FAL_GET_STATUS_CNF_T tData;
} DNS_FAL_PACKET_GET_STATUS_CNF_T;
```

The meanings of the ulStatusFlags in structure DNS_FAL_GEN_STATUS_T are described as follows:

```
/* Status flags with command DNS_FAL_CMD_GET_STATUS_REQ */
#define MSK_DNS_STA_FLAG_BUS_PRM_VALID            0x00000001
#define MSK_DNS_STA_FLAG_BUS_START                0x00000002
#define MSK_DNS_STA_FLAG_24V_NETWORK_POWER        0x00000004
#define MSK_DNS_STA_FLAG_NETWORK_STATE_ONLINE     0x00000008
#define MSK_DNS_STA_FLAG_RX_IDLE                  0x00000010
#define MSK_DNS_STA_FLAG_ERR_DUP_MAC_ID           0x00010000
#define MSK_DNS_STA_FLAG_ERR_BUS_OFF              0x00020000
#define MSK_DNS_STA_FLAG_RESERVED                 0xFFFCFFF0
```

- MSK_DNS_STA_FLAG_BUS_PRM_VALID

  This flag means that the stack has received a valid configuration.

- MSK_DNS_STA_FLAG_BUS_START

  This flag indicates that the application task has allowed the stack to start network communication by setting the operate mode to RUN. (see command DNS_FAL_CMD_SET_MODE_REQ).

- MSK_DNS_STA_FLAG_24V_NETWORK_POWER

This Flag indicates if the 24V Network Power is present or not. The 24V Network Power is a basic condition for DeviceNet to start any network activity. If this flag is not set, the device will not be present on the network or start any network activity.

- MSK_DNS_STA_FLAG_NETWORK_STATE_ONLINE

  This flag indicates if the stack has access to the network. This means that the 24V network power is present and the stack was able to send the two duplicate MAC_ID frames to the bus and is in general ready to communicate with the master.

- MSK_DNS_STA_FLAG_RX_IDLE

  This flag indicates the DeviceNet master has sent a receive_idle telegram to the slave. The receive idle telegram is normally a poll request with no data load. That also means a CAN frame with data length of zero.

- MSK_DNS_STA_FLAG_ERR_DUP_MAC_ID

  This flag indicates that the slave has found another device with the same network address on the bus. The slave stops any further operation and must be reconfigured and reset.

- MSK_DNS_STA_FLAG_ERR_BUS_OFF

  This flag indicates that the slave has detected a CAN “BUS_OFF” event. This indicates for heavy physical can errors e.g. bus short circuits. When the slave has detected this event, then this slave stops any further operation and goes into a major fault. The slave must be reset.

Packet Description

Variable	Type	Value / Range	Description
ulLen	UINT32	4 + n	4 = Default length n = number of diagnostic data depending on req. diag in usArea, usSubArea
ulSta	UINT32		See section 7.1 Error Codes of the FAL-Task
ulCmd	UINT32	0x2D05	`DNS_FAL_CMD_GET_STATUS_CNF` - Command
tData			
usArea	UINT16	0	Reserved for future use, zero
usSubArea	UINT16	0-12	0: Whole diagnostic structure. 1: Status Flag `ulStatusFlags.` 2: Counter for received CAN telegrams `ulRxInt.` 3: Counter for transmitted CAN telegrams `ulTxInt.` 4: Counter for transmission abort events `ulTxAborts`. 5: Counter for low transmission quality. A certain limit has been exceeded `ulErrorInt.` 6: Counter for bus off events `usBusOffCnt.` 7 Reset counter `usResetCnt.` 8: Own address of the DeviceNet Slave device in range 0..63. `ulNodeId.` 9: Chosen baud rate `ulBaudrate.` 10: Vendor ID value `usVendorId.` 11: Number of master input bytes that are produced `usProducedSize.` 12: Number of master output bytes that are consumed `usConsumedSize.`
abData[..]	UINT8[]		Array of bytes containing the status data

Table 80: DNS_FAL_CMD_GET_STATUS_CNF – Confirmation of DNS Get Status

7 Status/Error Codes Overview

7.1 Error Codes of the FAL-Task

Hexadecimal value	Definition / Description
0x00000000	TLR_S_OK Status ok
0xC0000001	TLR_E_FAIL Common error, detailed error information optionally present in the data area of packet
0xC0620001	TLR_E_DNS_FAL_DUPLICATE_MAC_ID Duplicate MAC ID found.
0xC0620002	TLR_E_DNS_FAL_INIT_TO_LESS_DATA Too less data for init command.
0xC0620003	TLR_E_DNS_FAL_FUNCTION_NOT_SUPPORTED Function not supported.
0xC0620006	TLR_E_DNS_FAL_PRM_ERR_CODE Invalid parameter in Init Stack request.
0xC0620007	TLR_E_DNS_FAL_BAUDRATE_OUT_RANGE Invalid Baudrate entered in Init Stack request.
0xC0620008	TLR_E_DNS_FAL_MAC_ID_OUT_RANGE Invalid MAC ID entered in Init Stack request.
0xC0620009	TLR_E_DNS_FAL_INVALID_PRODUCT_NAME Invalid Product Name Length entered in Init Stack request.
0xC062000A	TLR_E_DNS_FAL_INVALID_PRODUCED_SIZE Invalid Produced Size entered in Init Stack request.
0xC062000B	TLR_E_DNS_FAL_INVALID_CONSUMED_SIZE Invalid Consumed Size entered in Init Stack request.
0xC062000C	TLR_E_DNS_FAL_INVALID_MAJOR_REV Invalid Major Rev entered in Init Stack request.
0xC062000D	TLR_E_DNS_FAL_INVALID_MINOR_REV Invalid Minor Rev entered in Init Stack request.
0xC062000E	TLR_E_DNS_FAL_INVALID_VENDOR_ID Invalid Vendor ID entered in Init Stack request.
0xC062000F	TLR_E_DNS_FAL_INVALID_PRODUCT_TYPE Invalid Product Type entered in Init Stack request.
0xC0620010	TLR_E_DNS_FAL_INVALID_PRODUCT_CODE Invalid Product Code entered in Init Stack request.
0xC0620011	TLR_E_DNS_FAL_ALREADY_CONFIGURED Slave is already configured.
0xC0620012	TLR_E_DNS_FAL_SET_MODE_INVALID_MODE Invalid operation mode during Set Mode Request.
0xC0620013	TLR_E_DNS_FAL_SET_MODE_ALLREADY_IN_REQUEST Slave is currently in the mode requested.
0xC0620014	TLR_E_DNS_FAL_GET_STATUS_INVALID_STATUS Reserved byte not set to zero.
0xC0620015	TLR_E_DNS_FAL_UPDATE_IO_INVALID_IN_LEN Invalid Input Length specified in Update I/O Command.
0xC0620016	TLR_E_DNS_FAL_UPDATE_IO_INVALID_OUT_LEN Invalid Output Length specified in Update I/O Command.
0xC0620017	TLR_E_DNS_FAL_UPDATE_IO_INVALID_OUT_OFFSET Invalid Output Offset specified in Update I/O Command.
0xC0620018	TLR_E_DNS_FAL_UPDATE_IO_INVALID_IN_OFFSET Invalid Input Offset specified in Update I/O Command.

Hexadecimal value	Definition / Description
0xC0620019	TLR_E_DNS_FAL_SET_INPUT_INVALID_IN_LEN Invalid Input Length specified in Set Input Command.
0xC062001A	TLR_E_DNS_FAL_SET_INPUT_INVALID_IN_OFFSET Invalid Input Offset specified in Set Input Command.
0xC062001B	TLR_E_DNS_FAL_GET_OUTPUT_INVALID_OUT_LEN Invalid Output Length specified in Get Output Command.
0xC062001C	TLR_E_DNS_FAL_GET_OUTPUT_INVALID_OUT_OFFSET Invalid Output Offset specified in Get Output Command.
0xC062001E	TLR_E_DNS_FAL_DOWNLOAD_INVALID_AREA_CODE Invalid download area specified.
0xC062001F	TLR_E_DNS_FAL_DOWNLOAD_INVALID_SEQUENCE Invalid Download Sequence.
0xC0620020	TLR_E_DNS_FAL_DOWNLOAD_TO_MUCH_DATA Too much data received.
0xC0620021	TLR_E_DNS_FAL_DOWNLOAD_TO_LESS_DATA Not enough data received during the download.
0xC0620022	TLR_E_DNS_FAL_NO_CONFIGURATION No configuration.
0xC0620023	TLR_E_DNS_FAL_BUS_OFF_STATE Network error BUS OFF detected.
0xC0620024	TLR_E_DNS_FAL_NO_NETWORK No network access.
0xC0620025	TLR_E_DNS_FAL_BUS_STOP Communication not released by application (BUS Stop).
0xC0620026	TLR_E_DNS_FAL_NO_COMMUNICATION No communication.
0xC0620027	TLR_E_DNS_FAL_SERVICE_DATA_LENGTH_INVALID Invalid length of service data.
0xC0620028	TLR_E_DNS_FAL_USER_OBJ_CONFIGURED User object already configured.
0xC0620029	TLR_E_DNS_FAL_USER_OBJ_LOCKED User object is locked and cannot be passed through.
0xC062002A	TLR_E_DNS_FAL_USER_OBJ_ALREADY_REGISTERED User object has already been registered.
0xC062002B	TLR_E_DNS_FAL_USER_OBJ_NOT_REGISTERED User object has not been registered.
0xC062002C	TLR_E_DNS_FAL_24V_NETWORK_POWER_MISSING 24V Network Power Missing

Table 81: Error Codes of the FAL-Task

7.2 Error codes of the AP task

Hexadecimal value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC0000001	TLR_E_FAIL Common error, detailed error information optionally present in the data area of packet
0xC0630000	TLR_E_DNS_APS_NOTREGISTERED User Application not registered.
0xC0630001	TLR_E_DNS_APS_ALREADY_REGISTERED User Application already registered.
0xC0630003	TLR_E_DNS_APS_ACCESS_FAIL Unregister application queue access failed.
0xC0630004	TLR_E_DNS_APS_CONFIG_LOCK Function not allowed because configuration locked.
0xC0630005	TLR_E_DNS_AP_NO_DATA_BASE No database available.
0xC0630006	TLR_E_DNS_AP_OPEN_DATA_BASE Error open database.
0xC0630007	TLR_E_DNS_AP_IV_DNS_DATA_BASE Not a valid DeviceNet Slave database.
0xC0630008	TLR_E_DNS_AP_READ_DATA_BASE_TBL_GLB Error while reading table GLOBAL.
0xC0630009	TLR_E_DNS_AP_OPEN_DATA_BASE_TBL_GLB Error while open table GLOBAL.
0xC063000A	TLR_E_DNS_AP_OPEN_DATA_BASE_TBL_DNS Error while open table DNS.
0xC063000B	TLR_E_DNS_AP_READ_DATA_BASE_TBL_DNS Error while reading table DNS.

Table 82: Error codes of the AP task

7.3 Error Codes of the CAN_DL-Task

Hexadecimal value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC0000001	TLR_E_FAIL Common error, detailed error information optionally present in the data area of packet
0xC03F0001	TLR_E_CAN_DL_COMMAND_INVALID Invalid command.
0xC03F0002	TLR_E_CAN_DL_CMD_LENGTH_MISMATCH The length code of the command is invalid.
0xC03F0003	TLR_E_CAN_DL_UNKNOWN_PARAMETER_TYPE The parameter type of the command "Set Parameter" is invalid.
0xC03F0004	TLR_E_CAN_DL_SET_MODE_FAILED Within the command "Set Parameter" the function set "CAN Mode" failed.
0xC03F0005	TLR_E_CAN_DL_SET_BAUDRATE_FAILED Within the command "Set Parameter" the function set "Baudrate" failed.
0xC03F0006	TLR_E_CAN_DL_SET_TXABORT_TIME_FAILED Within the command "Set Parameter" the function set "Transmission Abort Timer" failed.
0xC03F0007	TLR_E_CAN_DL_SET_EVENTS_REQUESTED_FAILED Within the command "Set Parameter" the function set "Requested Events" failed.
0xC03F0008	TLR_E_CAN_DL_SET_FILTER_FAILED Within the command "Set Parameter" or "Set Filter the function set "CAN Filter" failed.
0xC03F0009	TLR_E_CAN_DL_SET_ENABLE_DISABLE_RXID_FAILED Within the command Enable or Disable of receive identifiers an error occurred.
0xC03F000A	TLR_E_CAN_DL_TX_FRAME_FAILED At least one CAN frame could not be send. Normally because the send process was aborted by the transmission abort timer.
0xC03F000B	TLR_E_CAN_DL_TX_BUFFER_OVERRUN The send request of CAN frames was rejected because the internal buffer for send requests is full.
0xC03F000C	TLR_E_CAN_DL_UNKNOWN_DIAG_TYPE The diagnostic type of the command "Get Diag" is invalid.
0xC03F000D	TRL_E_CAN_DL_TX_ABORT_ALREADY_IN_REQUEST The command "Transmission Abort" is already requested.
0xC03F000E	TRL_E_CAN_DL_TX_ABORT The send process of CAN frames was aborted by "Transmission Abort" command.
0xC03F000F	TRL_E_CAN_DL_UNKNOWN_APPLICATION The application makes access, is not registered at CAN_DL tasks.
0xC03F0010	TLR_E_CAN_DL_AP_ALREADY_REGISTERED The application is already registered.
0xC03F0011	TLR_E_CAN_DL_CONF_LOCK_FAIL The configuration lock failed.
0xC03F0012	TLR_E_CAN_DL_CONF_LOCKED The configuration is locked.

Table 83: Error Codes of the CAN_DL-Task

8 Appendix

8.1 List of Figures

8.2 List of Tables

8.3 Legal Notes

Copyright

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
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@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
Phone: +91 8888 750 777
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

Korea
Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@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