molex
one company › a world of innovation

SST DeviceNet Scanner Module

Firmware/Windows DLL Reference Guide

Document Edition: 3.8

Document #: 717-0001

Document Edition: 3.8

Date: November 5, 2010

This document applies to the SST DeviceNet Scanner Module and 32-Bit DLL API.

At Woodhead, we strive to ensure accuracy in our documentation. However, due to rapidly evolving products, software or hardware changes occasionally may not be reflected in our documents. If you notice any inaccuracies, please contact us (see Appendix A).

Written and designed at Woodhead Software & Electronics, 50 Northland Road, Waterloo, Ontario, Canada N2V 1N3.

Hardcopies are not controlled.

Contents

Preface

Preface Contents:

Purpose of this Guide

This document is a reference guide for the DeviceNet™ Scanner Module firmware (DNSCAN), an application module for the SST-DN family of interface cards.

Conventions

This guide uses stylistic conventions, special terms, and special notation to help enhance your understanding.

Style

The following stylistic conventions are used throughout this guide:

Bold	indicates field names, button names, tab names, and options or selections
Underlining	indicates a hyperlink
Italics	indicates keywords (indexed) or instances of new terms and/or specialized words that need emphasis
CAPS	indicates a specific key selection, such as ENTER, TAB, CTRL, ALT, DELETE
`Code Font`	indicates command line entries or text that you would type into a field
“>” delimiter	indicates how to navigate through a hierarchy of menu selections/options

Special Terms

The following special terms are used throughout this guide:

SST-DN Card	SST interface cards/products such as SST-DN3, SST-DN4 and SST-EDN
Firmware	the software running on the card
Module	a synonym for *firmware*
DNSCAN	the DeviceNet Scanner Module

Special Notation

The following special notations are used throughout this guide:

Note

A note provides additional information, emphasizes a point, or gives a tip for easier operation. Notes are accompanied by the symbol shown, and follow the text to which they refer.

1
Introduction

Chapter Contents:

1.1 DNSCAN Overview

The DeviceNet Scanner Module allows an SST-DN interface card to function as a node on a DeviceNet network. The module performs all communications functions, abstracting I/O and explicit messaging behaviors via high-speed, memory-mapped interfaces. The module can also provide simple, generic interfaces that allow CAN packets to be transmitted and received on a CAN-based network.

General Capabilities:

- UCMM capable (supports dynamic explicit messaging)
- Quick Connect capable for master only (supports Quick Connect allocation procedure when scanning on a device-by-device basis)
- Application objects in the host may be accessed from the network via explicit messaging
- Host Get Attribute access to default DeviceNet objects on the card via explicit messaging interface

Master/Scanner (Group 2 Client) Capabilities:

- Supports Strobe, Poll, Change-of-State and/or Cyclic I/O messaging
- Supports explicit messaging (I/O connection not required)
- Supports Group 2 Only Slaves & UCMM-capable devices
- Automatic verification of device identity
- Automatic reconnection of timed-out or faulted Slaves

Server/Slave (Group 2 Server) Capabilities:

- Supports Strobe, Poll, Change-of-State and/or Cyclic I/O messaging
- Supports explicit messaging

1.2 Distribution Files

DNSCAN is shipped with the following distribution files:

DNSCAN.SS3

This file contains the SST DeviceNet Scanner module firmware for use with the SST-DN3 family of DeviceNet interface cards.

DNSCAN.SS4

This file contains the SST DeviceNet Scanner module firmware for use with the SST-DN4 family of DeviceNet interface cards.

DNSCAN_USB.SS4

This file contains the SST DeviceNet Scanner module firmware for use with the SST-DN4-USB DeviceNet USB Interface.

DNSCAN32.DLL

This file contains the exported APIs (documented in Chapter 6, DeviceNet Scanner DLL API).

2
DeviceNet Compliance

Chapter Contents:

2.1 Introduction

This chapter defines the DeviceNet object model for the scanner. For background information on the object model, refer to the DeviceNet Specification.

Note

Blank entries in tables indicate that no predefined limits, options or fixed values apply.

2.2 Specification Revision

DNSCAN is compatible with the DeviceNet Specification:

- Volume I, Release 2.0
- Volume II, Release 2.0

2.3 General Communication Capabilities

DNSCAN's general communication capabilities are as follows:

Data Rate

- 125, 250 and 500 kbps

Predefined Master/Slave Connection Set

- Group 2 Client (Master)
- Group 2 Only Client (Master)
- Group 2 Server (Slave)

Dynamic Connections

- UCMM Capable
- Dynamic connections supported in groups 1, 2 and 3

Fragmentation

- Supports fragmented explicit messages
- Supports fragmented I/O messages

2.4 Identity Object, Class 0x01

There is only one instance of this object.

2.4.1 Class Attributes

Id	Description	Get	Set	Limits
1	Revision	○	○	
2	Max Instance	○	○	
6	Max ID of class attributes	○	○	
7	Max ID of instance attributes	○	○	
● Supported ○ Not Supported				

2.4.2 Class Services

Service		Param. Options
Get_Attributes_All	○	
Reset	○	
Get_Attribute_Single	○	
Find_Next_Object_Instance	○	
● Supported ○ Not Supported		

2.4.3 Instance Attributes

Id	Description	Get	Set	Limits
1	Vendor	●	○	
3	Device Type	●	○	0x0C
4	Product Code	●	○	12
5	Revision[1]	●	○	The revision number is “x.x”, in which the first digit represents the firmware Major revision, and the second represents the Minor revision.
6	Status	●	○	
7	Serial Number	●	○	
8	Product Name	●	○	
9	State	○	○	

● Supported ○ Not Supported

1 DeviceNet specifies that Major and Minor Revision may not be 0; the lowest valid Minor revision is 1.

2.4.4 Instance Services

Service		Param. Options
Reset	●	
Get_Attribute_Single	●	

● Supported ○ Not Supported

2.5 Message Router Object, Class 0x02

There is only one instance of this object.

2.5.1 Class Attributes

Id	Description	Get	Set	Limits
1	Revision	○	○	
4	Optional Attribute List	○	○	
5	Optional Service List	○	○	
6	Max ID of class attributes	○	○	
7	Max ID of instance attributes	○	○	

● Supported ○ Not Supported

2.5.2 Class Services

Service		Param. Options
Get_Attributes_All	○	
Get_Attribute_Single	○	

● Supported ○ Not Supported

2.5.3 Instance Attributes

Id	Description	Get	Set	Limits
1	Object List	○	○	
2	Maximum connections supported	○	○	
3	Number of active connections	○	○	
4	Active connections list	○	○	

● Supported ○ Not Supported

2.5.4 Instance Services

Service		Param. Options
Get_Attributes_All	○	
Get_Attribute_Single	○	

● Supported ○ Not Supported

2.6 DeviceNet Object, Class 0x03

There is only 1 instance of this object.

2.6.1 Class Attributes

Id	Description	Get	Set	Limits
1	Revision	●	○	2

● Supported ○ Not Supported

2.6.2 Class Services

Service		Param. Options
Get_Attribute_Single	●	

● Supported ○ Not Supported

2.6.3 Instance Attributes

Id	Description	Get	Set	Limits
1	MAC ID	●	○	
2	Baud Rate	●	○	
3	BOI	○	○	
4	Bus-off counter	○	○	
5	Allocation information	●	○	
6	MAC ID switch changed	○	○	
7	Baud rate switch changed	○	○	
8	MAC ID switch value	○	○	
9	Baud rate switch value	○	○	

● Supported ○ Not Supported

2.6.4 Instance Services

Service		Param. Options
Get_Attribute_Single	●	
Allocate M/S connection set	●	
Release M/S connection set	●	

● Supported ○ Not Supported

2.7 Connection Object, Class 0x05

There can be a maximum of 255 instances of the connection object in any combination of connection type.

2.7.1 Class Attributes

Id	Description	Get	Set	Limits
1	Revision	○	○	

● Supported ○ Not Supported

2.7.2 Class Services

Service		Param. Options
Reset	○	
Create	○	
Delete	○	
Get_Attribute_Single	○	
Find_Next_Object_Instance	○	

● Supported ○ Not Supported

2.7.3 Instance Attributes

Id	Description	Get	Set	Limits[1]
1	State	●	○	
2	Instance Type	●	○	
3	Transport class trigger	●	○	
4	Produced connection ID	●	○	
5	Consumed connection ID	●	○	
6	Initial comm. characteristics	●	○	
7	Produced connection size	●	○	
8	Consumed connection size	●	○	
9	Expected packet rate	●	●	
12	Watchdog timeout action[2]	●	●	
13	Produced connection path length	●	○	
14	Produced connection path	●	○	
15	Consumed connection path length	●	○	
16	Consumed connection path	●	○	
17	Production Inhibit Time[3]	●	●	

● Supported ○ Not Supported

[1] See limits for each connection type listed below.

[2] Set attribute service supported only by the server explicit messaging connection.

[3] Set attribute service supported on client trigger connections only.

2.7.3.1 Explicit Client Connection Attribute Limits

Id	Description	Limits/Fixed Value
1	State	
2	Instance Type	0
3	Transport class trigger	0x23
4	Produced connection ID	
5	Consumed connection ID	
6	Initial comm. characteristics	
7	Produced connection size	0xFFFF
8	Consumed connection size	0xFFFF
9	Expected packet rate	2500
12	Watchdog timeout action	0x01
13	Produced connection path length	0
14	Produced connection path	-
15	Consumed connection path length	0
16	Consumed connection path	-
17	Production Inhibit Time	-

2.7.3.2 Explicit Server Connection Attribute Limits

Id	Description	Limits/Fixed Value
1	State	
2	Instance Type	0
3	Transport class trigger	0x83
4	Produced connection ID	
5	Consumed connection ID	
6	Initial comm. characteristics	
7	Produced connection size	0xFFFF
8	Consumed connection size	0xFFFF
9	Expected packet rate	2500
12	Watchdog timeout action	0x01 or 0x03
13	Produced connection path length	0
14	Produced connection path	-
15	Consumed connection path length	0
16	Consumed connection path	-
17	Production Inhibit Time	0xFFFF

2.7.3.3 Poll Client Connection Attribute Limits

Id	Description	Limits/Fixed Value
1	State	
2	Instance Type	1
3	Transport class trigger	0x22
4	Produced connection ID	
5	Consumed connection ID	
6	Initial comm. characteristics	0x10
7	Produced connection size	
8	Consumed connection size	
9	Expected packet rate	250
12	Watchdog timeout action	0
13	Produced connection path length	
14	Produced connection path	
15	Consumed connection path length	
16	Consumed connection path	
17	Production Inhibit Time	-

2.7.3.4 Poll Server Connection Attribute Limits

Id	Description	Limits/Fixed Value
1	State	
2	Instance Type	1
3	Transport class trigger	0x83
4	Produced connection ID	
5	Consumed connection ID	
6	Initial comm. characteristics	0x01
7	Produced connection size	
8	Consumed connection size	
9	Expected packet rate	
12	Watchdog timeout action	0
13	Produced connection path length	
14	Produced connection path	
15	Consumed connection path length	
16	Consumed connection path	
17	Production Inhibit Time	-

2.7.3.5 Strobe Client Connection Attribute Limits

Id	Description	Limits/Fixed Value
1	State	
2	Instance Type	1
3	Transport class trigger	0x80
4	Produced connection ID	
5	Consumed connection ID	
6	Initial comm. characteristics	0xF0
7	Produced connection size	
8	Consumed connection size	
9	Expected packet rate	250
12	Watchdog timeout action	0
13	Produced connection path length	
14	Produced connection path	
15	Consumed connection path length	
16	Consumed connection path	
17	Production Inhibit Time	-

2.7.3.6 Strobe Server Connection Attribute Limits

Id	Description	Limits/Fixed Value
1	State	
2	Instance Type	1
3	Transport class trigger	0x83
4	Produced connection ID	
5	Consumed connection ID	
6	Initial comm. characteristics	0x02
7	Produced connection size	
8	Consumed connection size	
9	Expected packet rate	
12	Watchdog timeout action	0
13	Produced connection path length	
14	Produced connection path	
15	Consumed connection path length	
16	Consumed connection path	
17	Production Inhibit Time	-

2.7.3.7 COS/Cyclic Client Connection Attribute Limits (Acknowledged)

Id	Description	Limits/Fixed Value
1	State	
2	Instance Type	1
3	Transport class trigger	0x03 or 0x13
4	Produced connection ID	
5	Consumed connection ID	
6	Initial comm. characteristics	0x01
7	Produced connection size	
8	Consumed connection size	
9	Expected packet rate	250
12	Watchdog timeout action	0
13	Produced connection path length	
14	Produced connection path	
15	Consumed connection path length	
16	Consumed connection path	
17	Production Inhibit Time	-

2.7.3.8 COS/Cyclic Server Connection Attribute Limits (Acknowledged)

Id	Description	Limits/Fixed Value
1	State	
2	Instance Type	1
3	Transport class trigger	0x03 or 0x13
4	Produced connection ID	
5	Consumed connection ID	
6	Initial comm. characteristics	0x01
7	Produced connection size	
8	Consumed connection size	
9	Expected packet rate	250
12	Watchdog timeout action	0
13	Produced connection path length	
14	Produced connection path	
15	Consumed connection path length	4
16	Consumed connection path	0x20 0x2B 0x24 0x01
17	Production Inhibit Time	-

2.7.3.9 COS/Cyclic Client Connection Attribute Limits (Unacknowledged)

Id	Description	Limits/Fixed Value
1	State	
2	Instance Type	1
3	Transport class trigger	0x00 or 0x10
4	Produced connection ID	
5	Consumed connection ID	0xFFFF
6	Initial comm. characteristics	0x0F
7	Produced connection size	
8	Consumed connection size	
9	Expected packet rate	250
12	Watchdog timeout action	0
13	Produced connection path length	
14	Produced connection path	
15	Consumed connection path length	0
16	Consumed connection path	-
17	Production Inhibit Time	-

2.7.3.10 COS/Cyclic Server Connection Attribute Limits (Unacknowledged)

Id	Description	Limits/Fixed Value
1	State	
2	Instance Type	1
3	Transport class trigger	0x00 or 0x10
4	Produced connection ID	
5	Consumed connection ID	0xFFFF
6	Initial comm. characteristics	0x0F
7	Produced connection size	
8	Consumed connection size	
9	Expected packet rate	250
12	Watchdog timeout action	0
13	Produced connection path length	
14	Produced connection path	
15	Consumed connection path length	0
16	Consumed connection path	-
17	Production Inhibit Time	-

2.7.4 Instance Services

Service		Param. Options
Reset	○	
Delete	○	
Apply_Attributes	○	
Get_Attribute_Single	●	
Set_Attribute_Single	●	

● Supported ○ Not Supported

2.8 Ack Handler Object, Class 0x2B

The Ack Handler object class is only visible when one or more instances have been created.

2.8.1 Class Attributes

Id	Description	Get	Set	Limits
1	Revision	○	○	
2	Max Instance	○	○	
3	Number of Instances	○	○	
6	Max ID of class attributes	○	○	
7	Max ID of instance attributes	○	○	

● Supported ○ Not Supported

2.8.2 Class Services

Service		Param. Options
Get_Attribute_Single	○	
Create	○	
Delete	○	

● Supported ○ Not Supported

2.8.3 Instance Attributes

Id	Description	Get	Set	Limits
1	Acknowledge Timer	●	●	1 to 65535, Default=16
2	Retry Limit	●	○	1
3	COS Producing Connection Instance	●	○	
4	Ack List Size	○	○	
5	Ack List	○	○	
6	Data with Ack Path List Size	○	○	
7	Data with Ack Path List	○	○	

● Supported ○ Not Supported

2.8.4 Instance Services

Service		Param. Options
Get_Attribute_Single	●	
Set_Attribute_Single	●	
Delete	○	
Add_AckData_Path	○	
Remove_AckData_Path	○	

● Supported ○ Not Supported

3
DNSCAN Firmware Operation

Chapter Contents:

3.1 Module Features

DNSCAN performs all of the tasks necessary to establish and maintain messaging connections to DeviceNet devices. The main features of the module are:

Flexible Host Interface

- Supports both simple ladder logic and high-level language host applications

DeviceNet Client

- Support for up to 63 server devices (one MAC ID is used by the scanner)
- Device list may be modified at run-time
- Supports Group 2 Only Slaves
- Supports Group 2 Slaves (UCMM capable)
- Supports UCMM-capable servers without the Group 2 Master/Slave Connection Set (explicit messaging only)

DeviceNet Server

- UCMM-Capable Group 2 Server
- Supports Strobe, Poll, Change-of-State and Cyclic I/O connections
- Objects in the host application are accessible from DeviceNet

3.2 Terminology

As the DNSCAN firmware module supports both client and server capabilities, the following terms will be defined.

Inputs / Outputs

Throughout this document, inputs and outputs are named according to the process view.

Inputs are data that originate in the field and are transmitted to the process controller.
Outputs are data that originate in the process controller and are transmitted to the field devices.

Producer / Consumer

DeviceNet is a connection-based network, so all communications take place across connections.

A *connection* is a communication path between a *Producer* (message generator) and a *Consumer* (message recipient). Connections may be one-way (Producer at one end, Consumer at the other), or two-way (Producer and Consumer at each end).

When a connection sends a message, it is said to *produce* the message. When a connection receives a message, it is said to *consume* the message.

See the DeviceNet Specification for a detailed description of DeviceNet's connection-based architecture.

Client / Server

A *client* initiates communication on a connection. A *server* reacts to messages received on a connection. The server's reaction may cause it to send a response to the client.

Master / Slave Roles

A *Master* is a client end-point of a Group 2 Master/Slave connection. Masters transmit outputs and receive inputs from Slaves. Masters send explicit requests to Slaves.

A *Slave* is a server end-point of a Group 2 Master/Slave connection. Slaves receive outputs and transmit inputs to a Master. Slaves receive and respond to explicit request messages from a Master.

Note

A single device can be both a Master and Slave simultaneously.

3.3 Host Interface

All interaction between the host application and DNSCAN takes place in the shared RAM's Host Interface Block. There is an interrupt signal from the card to the host application and an interrupt from the host application to the card. These interrupts request action based on changes in the shared RAM.

3.3.1 Application Module Header

The Application Module Header is a section of the Host Interface Block that is common to all application modules for the card. This block contains module identification and runtime status information. See Section 4.3.

3.3.2 Command Buffer

The Command Buffer provides a command-response interface to the scanner.

Commands are written to the Command Buffer and an interrupt to the card triggers execution of the command. Command results are returned in the Command Buffer.

The command buffer interface is used to configure DNSCAN. See Section 4.5.

3.3.3 Client Status Block

The Client Status Block provides status information related to the scanner's client function. See Section 4.6.

3.3.4 Client Control Block

The Client Control Block provides control functions related to the scanner's client function. See Section 4.7.

3.3.5 Server Status Block

The Server Status Block provides status and event notification related to the scanner's server function. See Section 4.8.

3.3.6 Server Control Block

The Server Control Block provides event triggering and control related to the scanner's server function. See Section 4.9.

3.3.7 Device Control Event Table

The Device Control Event Table contains a flag for each MAC ID. The device's flag must be set by the host if any event flags in the Device Control Table are pending for the device.
See Section 4.10.

3.3.8 Device Status Table

The Device Status Table contains a status structure for each MAC ID. Each status structure provides event notification and runtime status for the device. See Section 4.11.

3.3.9 Device Control Table

The Device Control Table contains a control structure for each MAC ID. Each control structure provides event triggering and control for the device. See Section 4.12.

3.3.10 Memory Pool

The Memory Pool is the remainder of the Application Block that is not used by the pre-defined memory areas listed above. This memory may be freely allocated by the application for communication buffers and device I/O data buffers as required.

3.4 Server Configuration

The scanner can act as a Group 2 Server (Slave) and as an explicit messaging server (via the UCMM). The following sections describe the parameters that configure the operation of the scanner's server function.

Server configuration parameters are set using the DN_ONLINE command in the Command Buffer interface.

3.4.1 Server Connection Configuration

These parameters select the type of server communication supported, and configure each connection as required.

Item	Description
ConnectionFlags	This word contains flags that select the type of server connections supported. Up to 2 I/O connections and 1 explicit messaging connection may be supported.
ExplicitRequestSize[1] ExplicitRequestOffset[2,3]	The size and offset of the explicit request message buffer. This parameter is ignored if the explicit messaging connection is not used.
ExplicitResponseSize[1] ExplicitResponseOffset[2,3]	The size and offset of the explicit response message buffer. This parameter is ignored if the explicit messaging connection is not used.
Output1Size[1] Output1Offset[2,3]	The size and offset of the output data area for the first I/O connection. This parameter is ignored if no I/O connections are used or if the first I/O connection does not consume.
Output1PathOffset[2,3]	Offset to the I/O connection consumed path buffer.
Input1Size[1] Input1Offset[2,3]	The size and offset of the input data area for the first I/O connection. This parameter is ignored if no I/O connections are used or if the first I/O connection does not produce.
Input1PathOffset[2,3]	Offset to the I/O connection produced path buffer
Output2Size[1] Output2Offset[2,3]	The size and offset of the output data area for the second I/O connection. This parameter is ignored if no I/O connections are used or if the second I/O connection does not consume.
Output2PathOffset[2,3]	Offset to the I/O connection consumed path buffer.
Input2Size[1] Input2Offset[2,3]	The size and offset of the input data area for the second I/O connection. This parameter is ignored if no I/O connections are used or if the second I/O connection does not produce.
Input2PathOffset[2,3]	Offset to the I/O connection produced path buffer.

1 Size in bytes.

2 Offset in bytes from the start of the shared memory.

3 Must be within the memory pool host interface area.

3.4.1.1 Server ConnectionFlags

The connection flags parameter selects the types of connections supported by the application.

Offset	Bit							
	7	6	5	4	3	2	1	0
0	Res.	AKS	CYC	COS	RES	ST	P	EX
1	Reserved							

Reserved:

Shaded areas are reserved for future use and must be 0.

Group 2 only I/O Connection Flags:

AKS Acknowledge suppress
CYC Cyclic I/O connection
COS Change-of-state I/O connection
ST Bit-strobed I/O connection
P Polled I/O connection

Explicit Connection Flags:

EX Explicit Messaging Connection(s), G2 pre-defined and G3 dynamic explicit connections

3.4.1.2 Supported Server Connection Flag Combinations

The following table lists all supported combinations of connection flags and indicates the connections allocated to each I/O area.

Flags	Explicit	I/O 1	I/O 2
0x0001	G3 Dynamic[2]	N/A	N/A
0x0002	No	G2 M/S Poll	N/A
0x0003	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	N/A
0x0004	No	G2 M/S Strobe	N/A
0x0005	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	N/A
0x0006	No	G2 M/S Poll	G2 M/S Strobe
0x0007	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S Strobe
0x0010	No	G2 M/S COS	N/A
0x0011	G2 M/S or G3 Dynamic[1]	G2 M/S COS	N/A
0x0012	No	G2 M/S Poll	G2 M/S COS
0x0013	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S COS
0x0014	No	G2 M/S Strobe	G2 M/S COS
0x0015	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S COS
0x0020	No	G2 M/S Cyclic	N/A
0x0021	G2 M/S or G3 Dynamic[1]	G2 M/S Cyclic	N/A
0x0022	No	G2 M/S Poll	G2 M/S Cyclic
0x0023	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S Cyclic
0x0024	No	G2 M/S Strobe	G2 M/S Cyclic
0x0025	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S Cyclic
0x0050	No	G2 M/S COS /Ack[4]	N/A
0x0051	G2 M/S or G3 Dynamic[1]	G2 M/S COS /Ack[4]	N/A
0x0052	No	G2 M/S Poll	G2 M/S COS /Ack[4]
0x0053	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S COS /Ack[4]
0x0054	No	G2 M/S Strobe	G2 M/S COS /Ack[4]
0x0055	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S COS /Ack[4]
0x0060	No	G2 M/S Cyclic /Ack[4]	N/A
0x0061	G2 M/S or G3 Dynamic[1]	G2 M/S Cyclic /Ack[4]	N/A
0x0062	No	G2 M/S Poll	G2 M/S Cyclic /Ack[4]
0x0063	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S Cyclic /Ack[4]
0x0064	No	G2 M/S Strobe	G2 M/S Cyclic /Ack[4]
0x0065	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S Cyclic /Ack[4]

1 Both explicit connection types are supported.

2 Master/slave connection set is not supported (no I/O connections).

3 Scanner allocates either Cyclic or COS at its discretion.

4 Both acknowledged and unacknowledged supported at master's discretion.

3.5 Client Configuration

The scanner can support multiple client connections to multiple devices. The following sections describe the parameters that configure the operation of each client connection.

Client configuration parameters are set using the ADD_DEVICE command in the Command Buffer interface.

3.5.1 Client Connection Configuration

The device list contains client connection configuration parameters. These parameters select the type of communication to be used with the device and configure each connection as required.

Item	Description
ConnectionFlags	This word contains flags that select the type of connections to be opened with the device. Up to 2 I/O connections and 1 explicit messaging connection may be opened to each device.
ExplicitSize[1] ExplicitOffset[2,3]	The size and offset of the explicit message buffer. This parameter is ignored if the explicit messaging connection is not used.
Output1Size[1] Output1Offset[2,3]	The size and offset of the output data area for the first I/O connection. This parameter is ignored if no I/O connections are used or if the first I/O connection does not produce.
Output1LocalPathOffset[2,3]	Offset to the local I/O connection produced path buffer.
Output1RemotePathOffset[2,3]	Offset to the remote I/O connection produced path buffer.
Input1Size[1] Input1Offset[2,3]	The size and offset of the input data area for the first I/O connection. This parameter is ignored if no I/O connections are used or if the first I/O connection does not consume.
Input1LocalPathOffset[2,3]	Offset to the local I/O connection consumed path buffer.
Output1RemotePathOffset[2,3]	Offset to the remote I/O connection produced path buffer.
Output2Size[1] Output2Offset[2,3]	The size and offset of the output data area for the second I/O connection. This parameter is ignored if no I/O connections are used or if the second I/O connection does not produce.
Output2LocalPathOffset[2,3]	Offset to the local I/O connection produced path buffer.
Output2RemotePathOffset[2,3]	Offset to the remote I/O connection produced path buffer
Input2Size[1] Input2Offset[2,3]	The size and offset of the input data area for the second I/O connection. This parameter is ignored if no I/O connections are used or if the second I/O connection does not consume.
Input2LocalPathOffset[2,3]	Offset to the local I/O connection consumed path buffer.
Input2RemotePathOffset[2,3]	Offset to the remote I/O connection consumed path buffer.

1 Size in bytes.

2 Offset in bytes from the start of the shared memory.

3 Must be within the memory pool host interface area.

3.5.1.1 Client Connection Flags

The connection flags parameter selects the types of connections to be used with the device.

Offset	Bit							
	7	6	5	4	3	2	1	0
0	Res	AKS	CYC	COS	Res	ST	P	EX
1	Res	IID	QCR	QC	Reserved		G3P	G3

Reserved:

Shaded areas are reserved for future use and must be 0.

Group 2 only I/O Connection Flags:

AKS Acknowledge suppress

CYC Cyclic I/O connection

COS Change-of-State I/O connection

ST Bit-strobed I/O connection

P Polled I/O connection

Explicit Connection Flags:

EX Explicit Messaging connection

G3 Group 3 Dynamic Explicit Messaging connection only

G3P Group 3 Parent Explicit Messaging connection

Predefined Master/Slave Connection Set Allocation Procedure:

QC Connect to device using Quick Connect allocation procedure

QCR Enable use of reconnect timer for QC enabled connections

Miscellaneous Connection Flags:

IID Ignore current device identifier settings when establishing a connection

3.5.1.2 Supported Client Connection Flag Combinations

The following table lists all supported combinations of connection flags and indicates the connections allocated to each I/O area.

Flags[4]	Explicit	I/O 1	I/O 2
0x0001	G2 M/S or G3 Dynamic[1]	N/A	N/A
0x0002	No	G2 M/S Poll	N/A
0x0003	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	N/A
0x0004	No	G2 M/S Strobe	N/A
0x0005	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	N/A
0x0006	No	G2 M/S Poll	G2 M/S Strobe
0x0007	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S Strobe
0x0010	No	G2 M/S COS	N/A
0x0011	G2 M/S or G3 Dynamic[1]	G2 M/S COS	N/A
0x0012	No	G2 M/S Poll	G2 M/S COS
0x0013	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S COS
0x0014	No	G2 M/S Strobe	G2 M/S COS
0x0015	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S COS
0x0020	No	G2 M/S Cyclic	N/A
0x0021	G2 M/S or G3 Dynamic[1]	G2 M/S Cyclic	N/A
0x0022	No	G2 M/S Poll	G2 M/S Cyclic
0x0023	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S Cyclic
0x0024	No	G2 M/S Strobe	G2 M/S Cyclic
0x0025	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S Cyclic
0x0050	No	G2 M/S COS /Ack[3]	N/A
0x0051	G2 M/S or G3 Dynamic[1]	G2 M/S COS /Ack[3]	N/A
0x0052	No	G2 M/S Poll	G2 M/S COS /Ack[3]
0x0053	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S COS /Ack[3]
0x0054	No	G2 M/S Strobe	G2 M/S COS /Ack[3]
0x0055	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S COS /Ack[3]
0x0060	No	G2 M/S Cyclic /Ack[3]	N/A
0x0061	G2 M/S or G3 Dynamic[1]	G2 M/S Cyclic /Ack[3]	N/A
0x0062	No	G2 M/S Poll	G2 M/S Cyclic /Ack[3]
0x0063	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S Cyclic /Ack[3]
0x0064	No	G2 M/S Strobe	G2 M/S Cyclic /Ack[3]
0x0065	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S Cyclic /Ack[3]
0x0100	G3 Dynamic[3]	N/A	N/A
0x0203	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	N/A
0x0205	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	N/A

Flags[4]	Explicit	I/O 1	I/O 2
0x0207	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S Strobe
0x0211	G2 M/S or G3 Dynamic[1]	G2 M/S COS	N/A
0x0213	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S COS
0x0215	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S COS
0x0221	G2 M/S or G3 Dynamic[1]	G2 M/S Cyclic	N/A
0x0223	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S Cyclic
0x0225	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S Cyclic
0x0251	G2 M/S or G3 Dynamic[1]	G2 M/S COS /Ack[3]	N/A
0x0253	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S COS /Ack[3]
0x0255	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S COS /Ack[3]
0x0261	G2 M/S or G3 Dynamic[1]	G2 M/S Cyclic /Ack[3]	N/A
0x0263	G2 M/S or G3 Dynamic[1]	G2 M/S Poll	G2 M/S Cyclic /Ack[3]
0x0265	G2 M/S or G3 Dynamic[1]	G2 M/S Strobe	G2 M/S Cyclic /Ack[3]

1 Both explicit connection types are supported.

2 Attempts allocation of G3 Dynamic explicit connection only.

3 Slave device must support unacknowledged behaviors.

4 - To Enable Quick Connect feature, logically OR 0x1000 to any of the above supported connections.
- To enable QCR feature, logically OR 0x2000 to any of the above supported connections.
- To enable IID feature, logically OR 0x4000 to any of the above supported connections.

3.5.2 Device Identification

The device list contains device identification parameters. These parameters are optional and each may be used independently, as the application requires.

Item	Description
MacId	Device MAC ID (0-63)
VendorId	Device Vendor ID
DeviceType	Device Type
ProductCode	Device Product Code

3.5.2.1 Device Identification – Connection Flag IID

Clear (0 = Default)

If the IID Flag is clear (0), the scanner will verify any identification parameters that are non-zero. Any mismatch with the value returned by the device while the connection is being established will prevent connection to the device and the scanner will report the appropriate error in the device status table.

If any of the identification parameters are zero, the scanner reads them from the device and updates the device list. The updated device list may be read by the application and stored for future use. This feature permits a read-back of device identification parameters. Any reconnection attempts made by the scanner module to the device will verify these parameters, based on the values read by the scanner and updated in the device list when the initial connection establishment occurred.

This behavior is consistent with the behavior of previous DNSCAN firmware releases and is the default.

Set (1)

If the IID flag is set, the scanner will not verify any of the corresponding device's identification attributes, regardless of what value they were set to when the device was added to the scan list. The scanner will always read them from the device while the connection is being established and update the device list. The updated device list may be read by the application and stored for future use. This feature permits a read-back of device identification parameters.

3.6 Master Configuration

The scanner can act as a Group 2 Master/Slave Connection Set client (Master). The following sections describe the parameters that configure the operation of the scanner's Master function.

Master configuration parameters are set using the DN_ONLINE command in the Command Buffer interface.

Item	Description
ScanInterval	Group 2 Master/Slave Connection Set scan interval in ms. If this value is zero, the scan is as fast as the network parameters and traffic permit.
ReconnectTime	Specifies the interval (in ms) the scanner waits between reconnection attempts for devices that are faulted (timeout, connection attempt errors).

3.7 Scanner Configuration

The scanner has several distinct capabilities, all of which rely on a DeviceNet network connection. The following sections describe the general scanner configuration parameters that affect all functions.

Scanner configuration parameters are set using the DN_ONLINE command in the Command Buffer interface.

Item	Description
MAC ID	DeviceNet MAC ID for the scanner
Baud Rate	DeviceNet Baud Rate

3.7.1 COMM LED

The COMM LED indicates the communication status of the scanner application.

COMM LED	Status
Off	Offline
Flashing Green	Online - no connections active
Solid Green	Online - at least 1 connection is active
Solid Red	Bus Off

3.8 Master Operation

This section applies to Group 2 Master/Slave I/O connections.

3.8.1 Group 2 Master/Slave I/O Scan Timing

Group 2 Master/Slave Connection Set scan timing is dynamically controlled by DNSCAN and is affected by several factors (in decreasing order of significance):

- Scan Interval parameter (minimum interval)
- The number of devices being scanned
- Network Baud Rate
- Other I/O traffic on the network (peer-to-peer, other scanners)
- Device timeout
- Group 2 Only Slave overhead (proxy management)
- Explicit messaging traffic

3.8.1.1 Scan Interval Parameter

This parameter is part of the DN_ONLINE command to the command buffer interface. This value specifies the minimum scan interval. If the minimum scan interval (due to network loading factors listed in Section 3.8.1) is greater than the Scan Interval parameter, the Scan Interval parameter has no effect.

Note

The scan interval parameter can limit the I/O scan rate. It cannot speed up the I/O scan faster than network factors allow.

3.8.1.2 Network Baud Rate

The network baud rate has a large impact on the scan rate, especially if another Master is on the same bus. At slower baud rates, the network time necessary to exchange all I/O messages can exceed the time the scanner requires to process the messages.

3.8.1.3 Number of Devices

As more devices are added to the device list, the network scanning overhead increases.

3.8.1.4 Other I/O Traffic

I/O traffic uses the highest priority network messages. If there is peer-to-peer I/O traffic or another scanner on the same network, I/O messages may be delayed if higher priority messages are being sent. The effect of other I/O traffic is more significant at lower baud rates.

3.8.1.5 Device Timeout

When a device response is not received, the current scan cycle is extended, allowing extra time for the expected reply. If the device fails to respond for three consecutive scan cycles, it is flagged as not present and the reconnection sequence is initiated.

3.8.1.6 Group 2 Only Slave Overhead

Group 2 Only Slaves are simple devices, such as photo-detectors and limit switches, which have limited processing power. The scanner must provide explicit messaging proxy services on behalf of these devices. Each explicit message (i.e., from a configuration tool) sent to a Group 2 Slave that is being scanned must pass through the scanner. This introduces a small “hiccup” into the scan cycle. This effect is typically undetectable unless a large number of explicit messages are being transmitted, or a bulk upload or download of configuration data is being performed.

3.8.1.7 Explicit Message Traffic

Explicit message traffic to the scanner, or from the scanner to server devices, requires CPU processing time. Each explicit message introduces a small “hiccup” into the scan cycle. This effect is typically undetectable unless a large number of explicit messages are being transmitted, or a bulk upload or download of configuration data is being performed.

3.9 Client Operation

This section applies to all client connections, including Group 2 Master/Slave client connections.

3.9.1 Initialization

Client connections are created when the scanner is activated with the START_SCAN command.

3.9.2 Device Initialization

When the scanner is started, device initialization is performed sequentially for each device in the device list:

1. Open explicit messaging connection.
2. Read and verify device identification.
3. Create I/O connections.
4. Read and verify I/O connection sizes.

3.9.3 Device Timeout / Live Insertion of Devices

When a device not configured for Quick Connect stops responding, the scanner tries to reconnect at regular intervals. This reconnection process is the same as the device initialization process.

By default, when a device configured for Quick Connect stops responding, the scanner will wait for the device to be added to the network before attempting to reconnect. However, if the device was also configured with the QCR connection flag set, the scanner will try to reconnect at regular intervals. This reconnection process is the same as the device initialization process.

However, enabling reconnection (using the QCR flag) on Quick Connect-enabled devices may cause connection times to increase as much as 1 second above normal Quick Connection times. This increase will be seen when the device becomes active on the network while a reconnection attempt is already in progress. Therefore, increasing the reconnect interval time will reduce the likelihood of this delay. Automatic reconnection allows devices to be removed and reconnected while the scanner is running, without disrupting communications with other devices on the network.

3.9.4 Application Triggered Poll I/O

The DeviceNet Scanner Module gives the user greater control over system prioritization by allowing the application to trigger Poll I/O connection production for devices configured with a non-zero I/O Interval. Production on the Poll I/O connection can be triggered by setting the Output Update flag in the appropriate Device Control Table entry.

3.9.5 I/O Active & I/O Idle

DeviceNet supports “idle” I/O messages (messages with zero data bytes). Receipt of a zero-length message indicates a “receive_idle” condition, which may trigger a special response in a device. The behavior of devices under these conditions is vendor-specific.

An application may optionally send a heartbeat (see Section 4.5.9) to the scanner, which must be repeated at regular intervals or the I/O messages are forced to zero-length. This mechanism forces I/O messages to zero length if the host application terminates.

Client I/O connections report the receipt of zero length I/O messages (receive_idle condition) via flags in the Device Status Table.

3.9.6 I/O Data Interlocks

I/O data access interlocks are used to prevent access collisions between the scanner and host application. An access collision can result in reading partially updated data. This condition can have serious side effects for data types larger than 1 byte. I/O data interlocks can be handled in two ways:

1. Lock and access each device I/O data independently (Device Status/Control Table I/O Interlock flags).

2. Lock all device I/O data simultaneously (Client Status/Control Block I/O Interlock Flags).

The scanner always uses both methods, first requesting the Global I/O Data Interlock, and then requesting each individual Device I/O Data Interlock. This allows the application to use either approach and still guarantees I/O data integrity.

3.9.6.1 Individual Device I/O Interlock Access Procedure

If multi-byte data integrity is required, the following steps must be performed to avoid access collisions between the host application and scanner module:

1. Host checks the I/O interlock flag in the Device Status Table that corresponds to the I/O area to be accessed. If the interlock flag is set, the host must not access the affected I/O area.

2. Host sets the I/O interlock flag in the Device Control Table that corresponds to the I/O area to be accessed, preventing the scanner from accessing the I/O area.

3. Host re-checks the I/O interlock flag in the Device Status Table that corresponds to the I/O area to be accessed. If the interlock flag is set, the host should clear the flag set in step 2 and go back to step 1.

4. Host accesses I/O data.

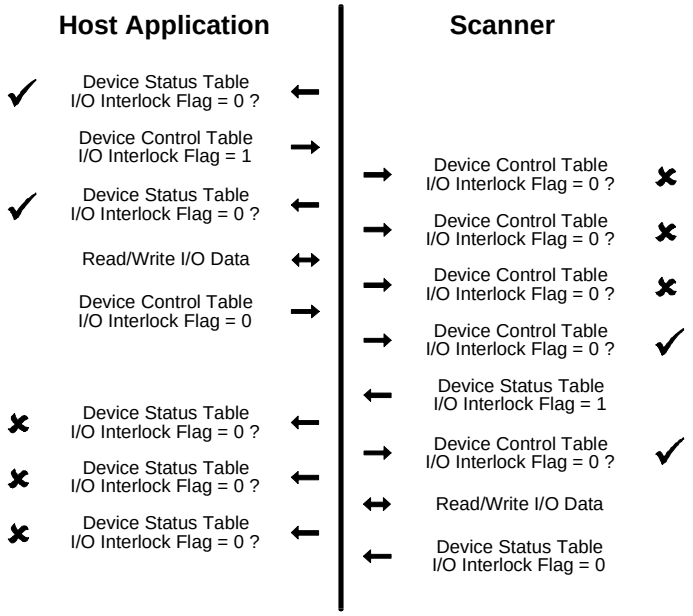
5. Host clears the I/O interlock flag in the Device Control Table that corresponds to the I/O area to be accessed, allowing the scanner to access the I/O area.

   The scanner follows the same rules (with the roles of the Device Control Table and Device Status Table reversed). This process ensures that only one application accesses an I/O area at a time.

Note

While the host has any I/O data Interlock, the scanner will be unable to read or write any I/O data associated with the corresponding device. It is extremely important that the host access the necessary I/O data and release the lock in as short a time a possible to prevent hiccups or lost connections.

Individual Device I/O Interlock Example:



* Each I/O area has an independent interlock flag

3.9.6.2 Global Device I/O Interlock Access Procedure

If multi-byte data integrity is required, the following steps must be performed to avoid access collisions between the host application and the scanner module:

1. Host checks the Input and/or Output data interlock flag in the Client Status Table. If the interlock flag is set, the scanner is accessing at least one device I/O data area.

2. Host sets the Input and/or Output data interlock flag in the Client Control Table.

3. Host re-checks the Input and/or Output data interlock flag in the Client Status Table. If the interlock flag is set, the host should clear the flag set in step 2 and go back to step 1.

4. Host accesses I/O data.

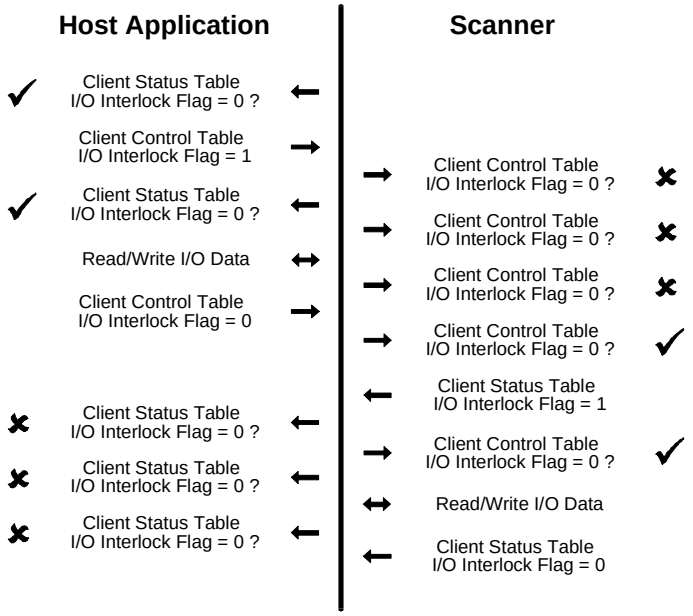
5. Host clears the Input and/or Output data interlock flag in the Device Control Table.

   The scanner follows the same rules (with the roles of the Client Control Table and Client Status Table reversed). This process ensures that only one application accesses any I/O data at a time.

Note

While the host has the Global I/O Interlock, the scanner will be unable to read or write any I/O data. It is extremely important that the host access the necessary I/O data and release the lock in as short a time a possible to prevent hiccups or lost connections.

Global I/O Interlock Example:



3.9.6.3 I/O Interlock Faults

While any I/O interlock flag is set, the scanner is unable to continue I/O communications with the affected device(s). If this condition persists, the device connection may time out or hiccups in the scan may occur. The host application must ensure that the I/O interlock flags are set only when necessary and are cleared as soon as possible.

3.9.7 Client Explicit Messaging

The scanner is able to send explicit message requests from the host application to a device or itself (see Section 3.11, Local Explicit Messaging via the Host). Explicit requests and responses are exchanged with the host application through the explicit message buffer assigned in the device list.

3.9.7.1 Client Explicit Messaging Procedure

Explicit messaging transactions are performed as follows:

1. Host checks the Explicit Message Buffer Interlock flag in the Device Control Table. If it is 0, a previous explicit request is not complete and the explicit message buffer must not be accessed (optional).

2. Host writes explicit request into buffer.

3. Host clears the Explicit Message Buffer Interlock flag in the Device Control Table to indicate that an explicit request is in progress (optional).

4. Host sets the Explicit Message Event flag in the Device Control Table.

5. Host sets the event flag within the Device Control Event Table to send the request.

6. Scanner clears the Explicit Message Event flag in the Device Control Table and sends the request message.

7. When an explicit response is received, or an error is encountered with the connection, the scanner writes the response to the Explicit Message Buffer and sets the Explicit Message Event flag in the Device Status Table.

8. Host clears the Explicit Message Event flag in the Device Status Table.

9. Host sets the Explicit Message Buffer Interlock flag in the Device Control Table (optional).

10. Host reads the response from the Explicit Message Buffer.

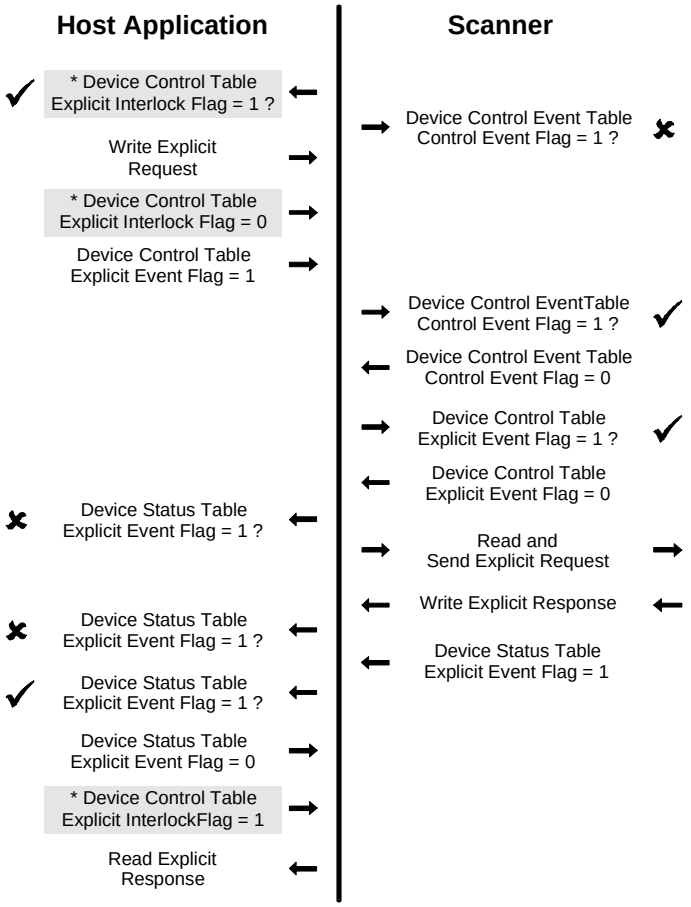
Note

An application using explicit interlocks must initialize (set) the Explicit Message Buffer interlock flag in the Device Control Table prior to the first explicit messaging sequence, as described above.

Note

If the explicit messaging connection times out, the device transitions to the timed out state, which is reported in the Device Status Table. Reconnection is attempted at regular intervals.

Client Explicit Messaging Example:



* Not required if host implements a different explicit messaging synchronization/interlock technique. (interlock flags are not used by the scanner module)

3.9.7.2 Client Explicit Request Format

The host uses this message format to send explicit requests to devices in the device list. This message is written into the device's explicit message buffer.

Offset	Name	Data Type	Description
0	Size	word	Number of Service Data bytes
2	Service	word	DeviceNet service code
4	Class [1]	word	The object class to which this is requested.
6	Instance [1]	word	The specific instance of the object class to which this request is directed.
8	Service Data	-	Optional data, as required by the service.

1 If the explicit message connection body format cannot represent the request, an internally generated explicit error response is returned.

Client Explicit Request Buffer Example:

The format of the client explicit buffer for a Get_Attribute_Single request to a device is shown below.

Offset	Name	Data Type	Value	Description
0	Size	word	0x0001	1 valid byte of Service Data
2	Service	word	0x000E	Get attribute single request
4	Class	word	0x0001	Class 0x01 = Identity object
6	Instance	word	0x0001	Instance 0x01
8	Service Data	-	0x07	Attribute 7 = Product Name

3.9.7.3 Client Explicit Response Format

The scanner uses this message format to send explicit responses to the host. This message is written into the device's explicit message buffer.

Offset	Name	Data Type	Description
0	Size	word	Number of Service Data bytes
2	Service/ Result [1]	word	DeviceNet service code / internal result code
4+	Service Data	-	Optional data, as required by the service.

1 Result codes > 0FFh are reserved for scanner internal errors.

Client Explicit Response Buffer Example:

A successful client explicit response for the previous Get_Attribute_Single request is shown below.

Offset	Name	Data Type	Value	Description
0	Size	word	0x0005	5 valid bytes of Service Data
2	Service	word	0x008E	Get attribute single response
4	Service Data	-	“HELLO”	Product Name

3.9.7.4 Internal Error Codes

Error codes are returned in the Service/Result field of an explicit response message. These errors are detected internally by the scanner.

Code	Description
0100h	Explicit connection is not established.
0101h	Explicit body format cannot represent requested class (i.e. class > 255 and connection body format is 8/8 or 8/16).
0102h	Explicit body format cannot represent requested instance (i.e. instance > 255 and connection body format is 8/8 or 16/8).
0103h	Resources unavailable to send explicit message.
0104h FFFFh	Reserved

3.10 Server Operation

This section applies to all server connections, including Group 2 Master/Slave Server connections.

3.10.1 Initialization

Server connections are established by a client device via allocation of the Group 2 Master/Slave Connection Set or though the UCMM.

The allocation status of the Group 2 Master/Slave Connection Set is reported in the Server Status Block.

3.10.2 I/O Active & I/O Idle

DeviceNet supports “idle” I/O messages (messages with zero data bytes). Receipt of a zero-length message indicates a “receive_idle” condition, which may trigger a special response in a device. The actual behavior of devices under these conditions is vendor-specific.

An application may optionally send a heartbeat (see Section 4.5.9) to the scanner at regular intervals. If the heartbeat is lost, the I/O messages are forced to zero-length. This mechanism forces the I/O messages to zero length if the host application is terminated.

Server I/O connections report the receipt of zero length I/O messages (receive_idle condition) via flags in the Server Status Block.

3.10.3 I/O Data Interlocks

I/O data access interlocks are used to prevent access collisions between the scanner and host application. An access collision can result in reading partially updated data. This condition can have serious side effects for data types larger than 1 byte.

3.10.3.1 Server I/O Access Procedure

If multi-byte data integrity is required, the following steps must be performed to avoid access collisions between the host application and the scanner module:

1. Host checks the I/O interlock flag in the Server Status Block that corresponds to the I/O area to be accessed. If the interlock flag is set, the host should not access the affected I/O area.

2. Host sets the I/O interlock flag in the Server Control Block that corresponds to the I/O area to be accessed, preventing the scanner from accessing the I/O area.

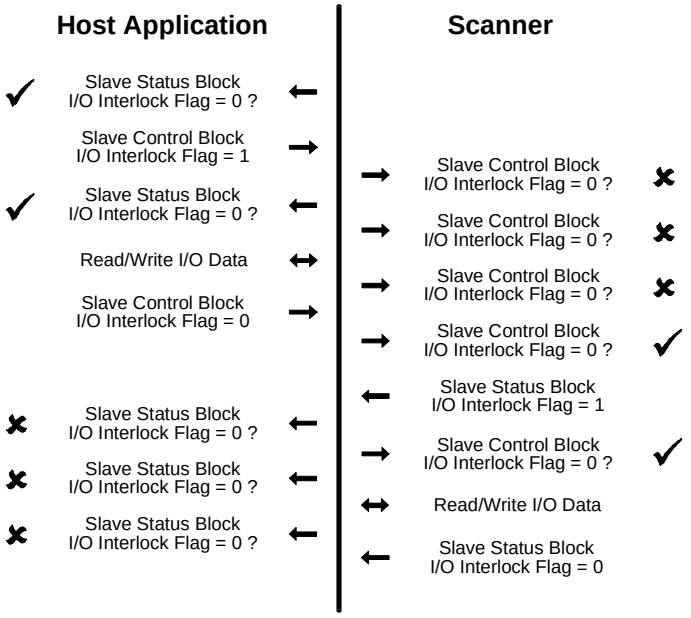
3. Host re-checks the I/O interlock flag in the Server Status Block that corresponds to the target I/O area. If the interlock flag is set, the host should clear the flag set in step 2 and go back to step 1.

4. Host accesses I/O data.

5. Host clears the I/O interlock flag in the Server Control Block that corresponds to the target I/O area, allowing the scanner to access the I/O area.

   The scanner follows the same rules with the roles of the Server Control Block and Server Status Block reversed. This process ensures that only one application accesses a given I/O area at a time.

Server I/O Interlock Example:



* Each I/O area has an independent interlock flag

3.10.3.2 Server I/O Interlock Fault

While a Server Control Block I/O Data Interlock Flag is set, the scanner is unable to transmit I/O response messages. If this condition persists, the I/O connection may time-out. The host application must ensure that the I/O Data Interlock flags are set only when necessary and cleared as soon as possible.

3.10.4 Server Explicit Messaging

The scanner can forward explicit message requests to the host application for service. Explicit requests are forwarded to the host application through the explicit request message buffer assigned in the server configuration data. Explicit responses are returned in the explicit response message buffer assigned in the server configuration data.

As requests and responses use independent message buffers, multiple requests may be in service by the host application at one time. Responses need not be returned in the order that requests are received.

The following explicit messages are forwarded to the host application for service if the server explicit messaging connection is configured:

- Message Router Requests
- Requests addressed to any object other than:
    - Connection Class (class & all instances)
    - DeviceNet Class (class & all instances)
    - Identity Object (instance 1)

If server explicit messaging is supported, the host must handle all Message Router Object requests. If server explicit messaging is not supported, the default Message Router Object in the scanner, which does not support any services, will return an error to all explicit requests.

Explicit requests received by the Group 2 Master/Slave Connection Set explicit connection and dynamic explicit connections opened through the UCMM are forwarded to the server explicit message buffer.

The host application is responsible for servicing all explicit requests and returning a valid response or error response in a timely manner.

If the explicit server messaging connection is not enabled, all unsupported explicit requests return an appropriate DeviceNet error response.

3.10.4.1 Server Explicit Request Procedure

The server explicit request procedure is as follows:

1. Scanner receives an explicit request that is not serviced by internal objects.

2. Scanner checks Explicit Request Interlock Flag in the Server Status Block. If the Explicit Request Interlock Flag is 0, a previous explicit request has not been read by the host. The current explicit request is deferred until the pending request has been read.

3. Scanner writes the explicit request to the Explicit Request Message Buffer. If the Explicit Request Message Buffer is too small for the message, the scanner returns a "too_much_data" error response.

4. Scanner clears the Explicit Request Interlock Flag in the Server Status Block and sets the Explicit Request Event Flag in the Server Status Block.

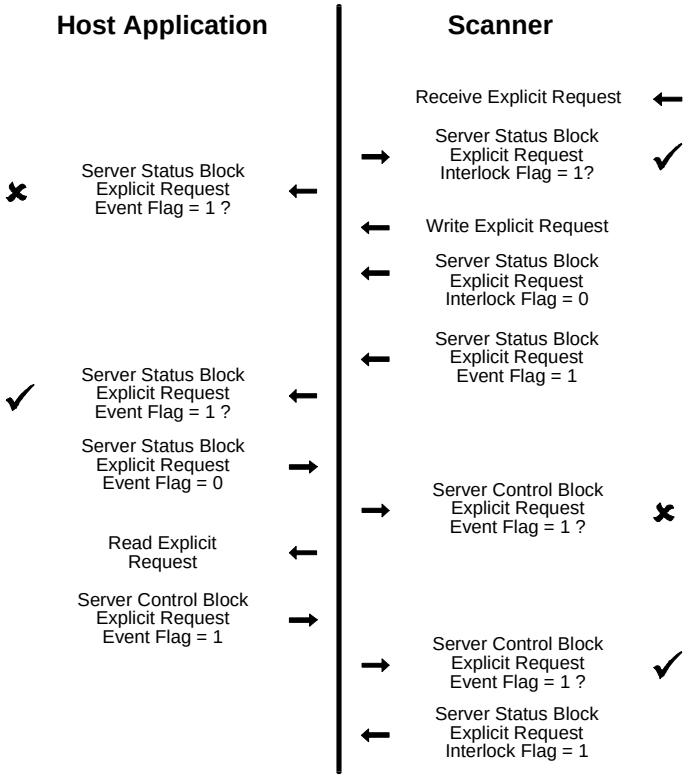
5. Host clears the Explicit Request Event Flag in the Server Status Block and reads the message from the Explicit Request Message Buffer.

6. Host sets the Explicit Request Event Flag in the Server Control Block.

7. Scanner clears the Explicit Request Event Flag in the Server Control Block.

8. Scanner sets the Explicit Request Interlock Flag in the Server Status Block.

Server Explicit Request Example:

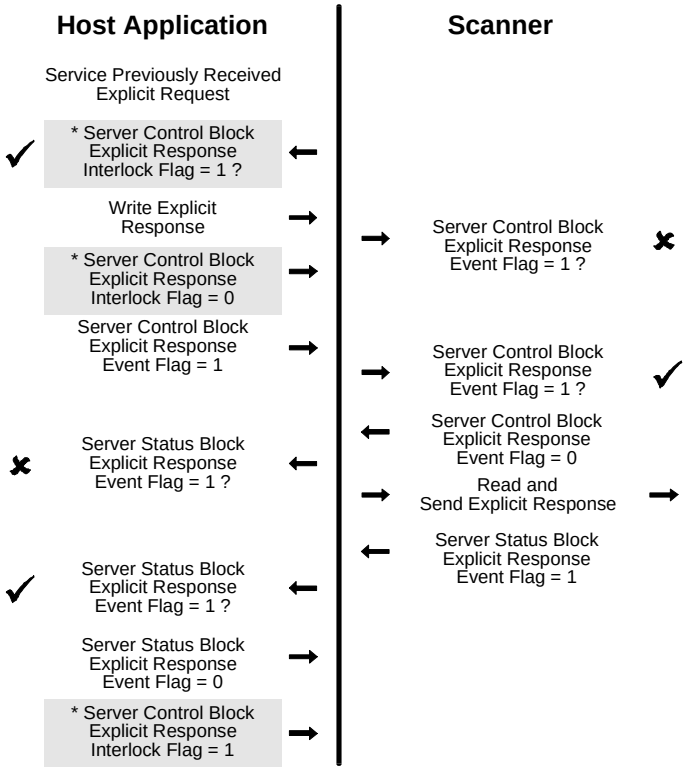


3.10.4.2 Server Explicit Response Procedure

The server explicit response procedure is as follows:

1. Host completes service of a previously received explicit request.
2. Host checks Explicit Response Interlock Flag in the Server Control Block (optional). If the Explicit Response Interlock Flag is 0, a previous explicit response has not been read by the scanner. The current explicit response must be deferred until the pending response has been read.
3. Host writes the explicit response to the Explicit Response Message Buffer.
4. Host clears the Explicit Response Interlock Flag in the Server Control Block (optional).
5. Host sets the Explicit Response Event Flag in the Server Control Block.
6. Scanner clears the Explicit Response Event Flag in the Server Control Block and reads the message from the Explicit Response Message Buffer.
7. Scanner sets the Explicit Response Event Flag in the Server Status Block.
8. Host clears the Explicit Response Event Flag in the Server Status Block.
9. Host sets the Explicit Response Interlock Flag in the Server Control Block (optional).

Server Explicit Response Example:



* Not required if host implements a different explicit messaging synchronization/interlock technique. (interlock flags are not used by the scanner module)

3.10.4.3 Server Explicit Request Format

The scanner uses this message format to forward unserviced explicit requests to the host. This message is written into the device's explicit message buffer. Since multiple concurrent requests may be in service by the host application, each request includes the connection ID to match responses with requests.

Offset	Name	Data Type	Description
00h	ConnectionId	word	Server connection ID, required for request / response matching.
02h	Size	word	Number of Service Data bytes
04h	Service	word	DeviceNet service code
06h	Class	word	The object class to which this is requested.
08h	Instance	word	The specific instance of the object class to which this request is directed.
0ah	Service Data	-	Optional data, as required by the service.

Server Explicit Request Buffer Example:

The format of the server explicit buffer upon receipt of a Get_Attribute_Single request is shown below.

Offset	Name	Data Type	Value	Description
00h	ConnectionId	word	0x0123	Connection ID, G3 UCMM-capable explicit connection.
02h	Size	word	0x0001	1 byte of Service Data
04h	Service	word	0x000E	Get_Attribute_Single Service
06h	Class	word	0x0064	Vendor specific Class
08h	Instance	word	0x0001	Instance 0x01
0ah	Service Data	-	0x01	Attribute 0x01

3.10.4.4 Server Explicit Response Format

The host uses this message format to send explicit responses to the scanner. This message is written into the server explicit response message buffer. Since multiple concurrent requests may be in service by the host application, each response includes the connection ID associated with the request.

Offset	Name	Data Type	Description
00h	ConnectionId	word	Server connection ID, required for request / response matching.
02h	Size	word	Number of Service Data bytes
04h	Service	word	DeviceNet service code
06h+	Service Data	-	Optional data, as required by the service.

Server Explicit Response Buffer Example:

A successful server explicit response for the previous Get_Attribute_Single request is shown below.

Offset	Name	Data Type	Value	Description
00h	ConnectionId	word	0x0123	Connection ID echoed from request
02h	Size	word	0x0002	2 bytes of Service Data
04h	Service	word	0x008E	Get_Attribute_Single response
06h+	Service Data	-	0xAA55	Attribute data

3.11 Local Explicit Messaging via the Host

The DeviceNet scanner firmware allows the host to send explicit requests to its own default DeviceNet Objects. This functionality allows access to the following DN objects on the card:

- DeviceNet Object
- Identity Object
- Connection Object
- Ack Handler Object
- Router Object

3.11.1 Local Explicit Messaging Procedure

The procedure for sending local explicit requests to the card is identical to that for sending client explicit messages (see Section 3.9.7), except that the host must configure its own Mac ID in the scan list with the explicit option enabled. Subsequent client explicit requests to the host's configured Mac ID will result in a response from the host card's default DeviceNet objects, and not from another device on the network.

The following applies to Local Explicit Messaging:

- If the host intends to use this feature in a server-only application, it is still required to configure itself in the scan list. The Start Scan command is not required.
- Only Get Attribute access is allowed for all attributes that support it (see Sections 2.4, 2.5, 2.6, 2.7 and 2.8 for supported Get Attribute access).
- No network interference is created for a local explicit request.

3.11.2 Device Status and Device Control Table Operation

When using Local Explicit Messaging via the host, the Device Status Table (see Section 4.11) and Device Control Table (see Section 4.12) view Host device information differently (device with the host's own MAC ID). The following tables show the differences in usage:

Device Status Table

Byte Offset	Access Rule	Name	Description
0	R	Status Code	0x03 Device timed out
1	R	Status Flags	N/A
2	R/W	I/O 1 Event Flags	N/A
3	R/W	I/O 2 Event Flags	N/A
4	R/W	Explicit Event Flags	Normal operation
5	R	Connection Allocation Flags	N/A
7-15			Reserved

Device Control Table

Byte Offset	Access Rule	Name	Description
0	W	Control Flags	Device Control Flags
1	R/W	I/O 1 Event Flags	N/A
2	R/W	I/O 2 Event Flags	N/A
3	R/W	Explicit Event Flags	Event flags related to the explicit message buffer.
4-13	R	Reserved	Reserved
14 15	W	Event Queue Enable Flags	Event notification queue enable flags

3.12 Connection Path Buffer

A connection path buffer is used to contain Connection Paths associated with client and/or server connections. An independent connection path buffer is used for each server I/O connection (if supported) and each supported I/O connection for each device. These buffers are allocated within the memory pool in the same manner as I/O data buffers.

3.12.1 Connection Path Format

This format is used to represent variable-length connection path strings.

Offset	Name	Data Type	Description
0	MaximumSize	word	Connection path buffer size. Maximum number of bytes in connection path.
2	Size	word	Number of bytes in connection path
4+	Path	byte...	Connection path

The minimum connection path buffer size is 4 bytes. In this case, both MaximumSize and Size must be 0 (no bytes available for Path).

3.13 Event Notification Queue

The event notification queue minimizes the overhead required to pass scanner events to the host application. Each event may be individually enabled for notification, reducing the number of interrupts that must be processed by the host application to the actual number of events being monitored.

The event notification queue is optional and the default behavior of all events is notification disabled.

3.13.1 Event Queue Operation

The event queue operates as follows:

1. Scanner processes an internal event, which results in a host interface event.
2. Scanner checks appropriate Control Block/Table event queue enable flag. If the flag is false, the scanner takes no further action.
3. Scanner writes event source and ID to the Event Queue Entry referenced by the QueueIn offset.
4. Scanner updates the local copy of QueueIn to point to the next queue entry wrapping around at the end of the queue area. If the new value for QueueIn is equal to QueueOut, the queue is full. The scanner asserts the Queue Overflow flag in Event Queue Status, does not update QueueIn and takes no further action.
5. Scanner writes the updated QueueIn and generates an Event Queue interrupt.
6. Host Application receives the event queue interrupt or some other trigger mechanism, such as a cyclic timer.
7. Host Application compares the QueueIn and QueueOut offsets in the Event Queue. If they are equal, the event queue is empty.
8. Host Application reads the event source and ID from the Event Queue Entry referenced by the QueueOut offset.
9. Host Application updates QueueOut to point to the next queue entry wrapping around at the end of the queue area.
10. Host may optionally repeat steps 7-9 until the queue is empty.

3.14 Event Trigger Queue

The event trigger queue minimizes the overhead required for the scanner to detect and respond to control notifications set by the host application.

The trigger notification queue is optional and has no effect until one or more triggers have been posted.

3.14.1 Trigger Queue Operation

To send a trigger to DNSCAN:

1. Host sets the appropriate event flags in the Device/Server Control Table/Block, as well as the Device Control Event Table (if applicable).

2. Host writes trigger source to the Trigger Queue Entry referenced by the QueueIn offset.

3. Host updates a local copy of QueueIn to point to the next queue entry, wrapping around at the end of the queue area. If the new value for QueueIn is equal to QueueOut, the queue is full and the application shall not update QueueIn.

4. Host writes the updated QueueIn and sends an interrupt to the card. Refer to the appropriate Hardware Reference Guide for details on how to interrupt the card.

5. Scanner module receives the card interrupt and checks the Trigger Queue.

6. Scanner compares the QueueIn and QueueOut offsets in the Trigger Queue. If they are equal, the Trigger Queue is empty.

7. Scanner reads the trigger Source from the queue. If the Source is invalid, the scanner writes the Invalid Trigger Event flag to Queue Status Bytes A and B.

8. Scanner checks the Device/Server Control Table/Block, Device Control Event Table, or Command Buffer, as appropriate to the Source.

3.15 CAN Receive Queue

The scanner can forward received CAN 2.0A messages to the host application. The host creates the receive queue in the memory pool by sending the CAN_SETRXQ command to the scanner, then enables individual CAN IDs for reception by sending CAN_SETFILTER commands.

3.15.1 Receive Queue Operation

The receive queue operates as follows:

1. Scanner receives a message whose CAN ID is enabled by the filter.
2. Scanner writes CAN message to the queue at the QueueIn offset.
3. Scanner updates a local copy of QueueIn to point to the next entry, wrapping around at the end of the queue area. If the new value for QueueIn is equal to QueueOut, the queue is full and the scanner asserts the Queue overrun flag in the QueueStatus A/B bytes, does not update QueueIn, and takes no further action.
4. Scanner writes the updated QueueIn value and generates a CAN Rx Event interrupt.
5. Host application receives the CAN Rx Event interrupt or some other trigger mechanism, such as a cyclic timer.
6. Host compares the QueueIn and QueueOut offsets in the CAN receive queue. If they are equal, the receive queue is empty.
7. Host reads the CAN message from the queue at the QueueOut offset.
8. Host updates QueueOut to point to the next queue entry, wrapping around at the end of the queue area.
9. Host repeats steps 6 through 8 until the queue is empty.

4

Shared Memory Interface

Chapter Contents:

4.1 Introduction

The interface between DNSCAN and the host application is implemented using shared memory and one interrupt signal in each direction. The following sections describe the nature of this interface.

4.2 Host Interface Memory

The SST-DN card uses 16 Kbytes of shared RAM to communicate with the host application. The layout of this memory is illustrated in the following table.

Address		Description
Hex	Decimal	
0000h	0	Application Module Header
007Fh	127	
0080h	128	Application Host Interface
3FFFh	16383	

Each type of SST-DN card has 16K of RAM, conforming to this memory map. However, the way this memory is mapped into the host system's address space is different for each type of card. Please refer to the Hardware Reference Guide for memory access details.

4.3 Application Module Header

Each application for the SST-DN card is based on an event-driven kernel. This kernel handles the self-diagnostics and module startup.

The kernel reserves the first 128 bytes of the Host Interface Block for loader interface and run-time status information. The majority of this header area is defined and maintained by the application kernel. The following table provides information on the application-specific portions of the header.

All offsets in the table are from the start of the Host Interface Block. Items without an asterisk are common to all SST-DN application modules. Only the items with an asterisk have certain attributes specific to the DNSCAN application module.

Application Module Header

Offset	Name	Data Type	Description
0000h	ModuleType	CHAR[2]	"DN" (0444eh) = card OK "ER" (04552h) = fatal error
0002h	WinSize	UINT2	0000h
0004h	CardId	UINT2	For host application use
0006h	Kernel Id	UINT2	Kernel identification. 0x03 = CAN 2.0 A/B kernel
0008h	Kernel Rev	UINT2	Kernel Revision
000ah	ModuleId	UINT2	Module Id, 0x14 = DNSCAN
000ch	ModuleRev	UINT2	Module revision
000eh	NetSerial	UINT4	DeviceNet serial number
0012h	CardType	CHAR[16]	Card type (i.e. "SST-DN3-104").
0022h	CardSerial	CHAR[8]	Card Serial number
002ah	* IrqControl	UINT2	Card interrupt control
002ch	* IrqStatusA	UINT1	Card interrupt status
002dh	* IrqStatusB	UINT1	
002eh	* MainCode	UINT2	Main Application Error Code
0030h	CanStatus	UINT2	CAN status word
0032h	CanTx	UINT2	CAN transmit counter. Incremented when messages are submitted to the CAN controller.
0034h	CanAck	UINT2	CANAck error counter. Incremented when a transmit message is aborted due to lack of acknowledgment from other stations. When CanAck is incremented, CanTx is decremented to compensate for messages not actually transmitted.
0036h	* CanRx	UINT2	CAN receive counter. Incremented when messages are received. Messages that fail the receive filter still increment CanRx.
0038h	CanError	UINT2	CAN communication error counter. Incremented when a CAN frame error is detected.
003ah	* CanLost	UINT2	CAN lost messages counter. Incremented when a CAN message is received before the previous one is queued.
003ch	* CanOverrun	UINT2	CAN receive queue overrun counter. Incremented when a CAN message is lost due to a full receive queue.
003eh	* AddCode	UINT2	Additional Application Error Code
0040h	Message	CHAR[60]	When ModuleType is "DN", contains the module identification string. When ModuleType is "ER", contains the kernel error string.
007ch	MajorTickInterval	UINT2	Number of milliseconds per Major Tick
007eh	MinorTickCount	UINT2	Number of Minor Ticks per Major Tick

4.3.1 IRQ Control / Status

The IRQ control and status areas are used to implement up to 8 logical interrupts using only 1 physical interrupt signal.

4.3.1.1 IRQ Control Word (002Ah)

The IRQ Control Word contains 8 bits, which are used to enable the generation of a physical interrupt for each of up to 8 logical interrupt sources. This word is written by the host and not written by the scanner.

Offset	Bit							
	7	6	5	4	3	2	1	0
002Ah	Reserved				CE	QE	BS	CM
002Bh	Reserved							

CE CAN Rx Event (see Section 4.15).

QE Queue Event (see Section 4.12.5).

BS Bus status changed (see Section 4.3.2).

CM Command acknowledge (see Section 4.5).

4.3.1.2 IRQ Status Byte A/B (002Ch / 002Dh)

The IRQ Status Bytes each contain 8 bits, which are set when each of up to 8 logical interrupts occur. The IRQ Status Bytes reflect logical interrupts, even if the physical interrupt has been disabled via the IRQ Control Word.

These bytes are written by the scanner and the host application. The IRQ access protocol described below must be followed to avoid missed interrupts.

Offset	Bit							
	7	6	5	4	3	2	1	0
002Ch	Reserved				CE	QE	BS	CM
002Dh	Reserved				CE	QE	BS	CM

CE CAN Rx Event (see Section 4.15).

QE Queue Event (see Section 4.12.5).

BS Bus status changed (see Section 4.3.2).

CM Command acknowledge (see Section 4.5).

4.3.1.3 IRQ Status Access Protocol

To avoid inadvertently clearing interrupt status flags, the host application must:

1. Read IrqStatusA and store the result in temporary variable ‘A’.
2. Clear relevant bits in IrqStatusA.
3. Read IrqStatusB and store the result in temporary variable ‘B’.
4. Clear relevant bits in IrqStatusB.
5. Logically OR temporary variables ‘A’ and ‘B’ to determine IRQ status flags.

Note

The only possible result of an access collision between the scanner and the host application is an interrupt status flag remaining set after the host has attempted to clear it.

4.3.2 CAN Bus Status Word (0030h)

The CAN Bus Status Word contains bit flags that indicate the current status of the CAN bus interface and low-level interrupt handlers. When the CAN Status word changes, a Bus Status interrupt is generated, providing it is enabled in the IRQ Control Word.

Offset	Bit							
	7	6	5	4	3	2	1	0
0030h	ML	RO	TO	TA	A	BO	BW	OL
0031h	SA	O5	O2	O1	Reserved		BP	ER

4.3.2.1 BP - Bus Power Detect

The BP bit indicates the presence or absence of network power. The BP bit is clear if the physical bus interface in not powered.

4.3.2.2 ER - CAN Bus Error

ER is set each time a CAN communication error is detected.

An excessive number of errors indicates a faulty physical medial component (cable, connector etc.) or excessive noise from external sources (check cable routing and shield connection).

4.3.2.3 ML - Message Lost

ML is set when a message is received from the bus while the previous message is still in the receive buffer.

ML indicates a lower-layer application error (in the kernel interrupt handler). Report this condition to Technical Support.

4.3.2.4 RO - Receive Buffer Overrun

RO is set when messages are received from the bus faster than the application can process them.

RO indicates an upper-layer application error (in the application module). Report this condition to Technical Support.

4.3.2.5 TO - Transmit Timeout

TO is set when a pending transmission is incomplete within 25-50ms.

TO indicates excessive message traffic at a higher priority than the aborted message.

4.3.2.6 TA - Transmit Ack Error

The TA bit is not supported. Transmit acknowledge errors will be indicated using the TO bit.

4.3.2.7 A - Network activity detected

A is set when any message is transmitted or received.

4.3.2.8 BO - Bus off

BO is set when an excessive number of communication errors are detected and the CAN chip automatically goes off-line. BO is cleared when the CAN interface is re-initialized.

BO indicates a serious communication fault such as incorrect baud rate or physical layer error (short, open etc). To recover from a bus off condition the application can issue the following command sequence: STOP_SCAN, OFFLINE, ONLINE, START_SCAN.

4.3.2.9 BW - Bus warning

BW is set when an abnormal number of communication errors is detected and the CAN chip stops transmitting error frames. BW is cleared when the error count returns to normal levels or the CAN interface is re-initialized.

BW indicates a potentially serious communication fault, such as out-of-tolerance baud rate or physical layer error (electrical noise, signal attenuation, intermittent connections, etc.).

4.3.2.10 OL - Online

OL indicates that the CAN interface has been initialized and is ready to communicate.

4.3.2.11 SA – Scanner Active

SA indicates that the scanner is active on the network.

4.3.2.12 O5, O2, O1 – Baud indicators

O5, O2 or O1 indicate that the device is online at 500k, 250k or 125k respectively.

4.4 Application Host Interface

The kernel reserves the first 128 bytes of the Host Interface Block for loader interface and run-time status information. The remainder of the Host Interface Block is defined by the application module.

The following table defines the format of the Host Interface Block's application-specific portion. Note that all offsets are from the start of the Host Interface Block.

Offset	Name	Description
0080h 013Fh	CmdBuf	Command buffer See Section 4.5.
0140h 014Fh	ClientStatus	Client Status Block See Section 4.6.
0150h 015Fh	ClientControl	Client Control Block See Section 4.7.
0160h 016Fh	ServerStatus	Server Status Block. See Section 4.8.
0170h 017Fh	ServerControl	Server Control Block See Section 4.9.
0180h 01BFh	DeviceControlEvent	Device Control Event Table See Section 4.10.
01C0h 05BFh	DeviceStatus	Device Status Table See Section 4.11.
05C0h 09BFh	DeviceControl	Device Control Table See Section 4.12.
09C0h 0DBFh	EventQueue	Event Notification Queue See Section 4.13.
0DC0h 0FBFh	EventTriggerQueue	Event Trigger Queue See Section 4.14.
0FC0h 0FFFh	Reserved	
1000h 3FFFh	MemPool	Free memory

4.5 Command Buffer

Offset: 0080h - 013Fh

The Command Buffer is the main initialization interface between the host application and DNSCAN. Commands are written to the Command Buffer to initialize, start and shut down the scanner. The parameter area of the Command Buffer differs for each command.

Command results and reply data (if any) are returned in the command buffer. The following sections describe the format of the command and reply for each command.

4.5.1 Sending a Command to DNSCAN

To send a command to DNSCAN:

1. Clear the Command Acknowledge flag (CM) in IRQStatus bytes A and B.
   See Section 4.3.1.

2. Write the command and parameters (if required) to the Command Buffer (0080h).

3. Interrupt the card to execute the command. Refer to the Hardware Reference Guide for details on how to interrupt the card.

4. Wait for the CM flag to appear in IRQStatus bytes A or B (see Section 4.3.1). This may be implemented with a physical interrupt or by polling the card interrupt flag.

5. Read the command status and reply data (if any) from the Command Buffer.
   See Section 4.5.7.

The following table lists all the commands supported by DNSCAN.

DNSCAN-Supported Commands

Command	Value	Description	Timeout
DN_ONLINE	01h	Initialize server parameters and go on-line.	3 sec.
DN_OFFLINE	02h	Go offline.	1 sec.
ADD_DEVICE	03h	Add a device to the device list.	1 sec.
GET_DEVICE	04h	Retrieve device configuration.	1 sec.
DELETE_DEVICE	05h	Remove a device from the device list.	1 sec.
START_SCAN	06h	Establish client connections and start scanning devices in the device list.	1 sec.
STOP_SCAN	07h	Stop scanning and close client connections.	1 sec.
IO_ACTIVE	08h	Set I/O active and start heartbeat timer.	1 sec.
IO_IDLE	09h	Set I/O idle and stop heartbeat timer.	1 sec.
SCAN_ACTIVE	0Ah	Start the scan heartbeat timer, using the value specified.	1 sec.
CAN_TRANSMIT	0Bh	Transmit a CAN Message.	1 sec.
CAN_SETFILTER	0Ch	Set the CAN Message receive filter.	1 sec.
CAN_SETRXQ	0Dh	Set the CAN Receive queue memory location and size of the CAN receive queue.	1 sec.

After the execution of any command, the Command Buffer contains either the command reply data, as described in the following sections, or an error status block. The format of the Error Status Block is:

Offset	Name	Data Type	Description
0080h	Command	word	Command with bit 15 set
0082h	ErrCode	word	Error Code
0084h 013Fh			Reserved

If any command fails, the Command Buffer contains the original command with bit 15 set and ErrCode indicates the nature of the error.

4.5.1.1 Command Errors

The following ErrCode values are defined:

Error Code	Value	Description	Command
	0000h	Reserved	
ERR_CMD	0001h	Invalid command	
ERR_MAC	0002h	Invalid MAC ID	DN_ONLINE
ERR_BAUD	0003h	Invalid baud rate	DN_ONLINE
ERR_DUPMAC	0004h	Duplicate MAC ID	DN_ONLINE
ERR_DUPDEV	0005h	Duplicate device	ADD_DEVICE
ERR_NODEV	0006h	Device not found	GET_DEVICE DELETE_DEVICE
ERR_OFF	0007h	Bus offline	START_SCAN CAN_TRANSMIT
ERR_ACTIVE	0008h	Scanner is active	DN_OFFLINE
ERR_NOTOFF	0009h	Bus is not offline	DN_ONLINE
ERR_SCAN	000Ah	Scanner is running	START_SCAN
ERR_NOTSCAN	000Bh	Scanner not running	STOP_SCAN SCAN_ACTIVE
ERR_SCANOFF	000Ch	Scanner is stopping	START_SCAN STOP_SCAN SCAN_ACTIVE
ERR_OFFSET	000Dh	Invalid memory pool offset	ADD_DEVICE CAN_SETRXQ
ERR_BUSOFF	000Eh	Bus fault	DN_ONLINE
ERR_CONNECTION_FLAGS	000Fh	Invalid connection flags combination	ADD_DEVICE DN_ONLINE
ERR_EXP_BUFFER	0010h	Invalid explicit buffer size	ADD_DEVICE DN_ONLINE
ERR_STROBE_BUFFER	0011h	Invalid strobe buffer size (output must be 1).	ADD_DEVICE DN_ONLINE
ERR_PATH_BUFFER	0012h	Invalid path buffer (not initialized).	ADD_DEVICE DN_ONLINE
ERR_ACK_FAULT	0013h	Ack Fault (no acknowledge received during Duplicate MAC ID Sequence).	DN_ONLINE
ERR_INVALID_SIZE	0014h	Invalid size parameter	CAN_TRANSMIT CAN_SETRXQ
ERR_INVALID_CANID	0015h	Invalid CAN ID	CAN_TRANSMIT CAN_SETFILTER
	0016h - 7FFFh	Reserved for future use	
ERR_USER	8000h - FFFFh	Reserved for host application use	

4.5.2 DN_ONLINE Command

The DN_ONLINE command initializes the DeviceNet interface. Due to the duplicate MAC ID check, this command may take over 2 seconds to complete.

The format of the command buffer for the DN_ONLINE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. DN_ONLINE = 01h.
0082h	MacId	word	Scanner MAC ID (0-63). Must not duplicate any other station.
0084h	BaudRate	word	DeviceNet baud rate. 0 = 125 kbaud 1 = 250 kbaud 2 = 500 kbaud
0086h	ScanInterval	word	Default I/O update interval. (Used for Strobed I/O and Poll I/O connections with an I/O interval = 0).
0088h	ReconnectTime	word	Reconnection timeout period. 0 – Defaults to 10 seconds Valid Range 100 – 65535ms
008Ah	Flags	word	Server connection flags
008Ch	ExplicitRequestSize	word	Explicit request buffer size (bytes)
008Eh	ExplicitRequestOffset	word	Offset to explicit request buffer
0090h	ExplicitResponseSize	word	Explicit response buffer size (bytes)
0092h	ExplicitResponseOffset	word	Offset to explicit response buffer
0094h	Reserved	word	Future use, set to 0
0096h	Output1Size	word	I/O connection 1 output size (bytes)
0098h	Output1Offset	word	Offset to I/O connection 1 output data
009Ah	Output1PathOffset	word	Offset to the I/O connection consumed path buffer
009Ch	Input1Size	word	I/O connection 1 input size (bytes)
009Eh	Input1Offset	word	Offset to I/O connection 1 input data
00A0h	Input1PathOffset	word	Offset to the I/O connection produced path buffer
00A2h	Reserved	word	Future use, set to 0
00A4h	Output2Size	word	I/O connection 2 output size (bytes)
00A6h	Output2Offset	word	Offset to I/O connection 2 output data
00A8h	Output2PathOffset	word	Offset to the I/O connection consumed path buffer
00AAh	Input2Size	word	I/O connection 2 input size (bytes)
00ACh	Input2Offset	word	Offset to I/O connection 2 input data
00AEh	Input2PathOffset	word	Offset to the I/O connection produced path buffer
00B0h - 013Fh			Reserved

The format of the command reply data for the DN_ONLINE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. DN_ONLINE = 01h Contains 8001h on error. See Section 4.5.1.1.
0082h - 013Fh			Reserved

4.5.2.1 Duplicate MAC Id Check

During execution of the DN_ONLINE command, DNSCAN performs a duplicate MAC ID check. If there is another station already using the same MAC ID, DNSCAN returns error code 0004h, ERR_DUPMAC. See Section 4.5.1.1.

4.5.2.2 Bus Off Fault

If a fatal bus error (bus off) is detected during execution of the duplicate MAC ID check, DNSCAN returns error code 000Eh, ERR_BUSOFF. See Section 4.5.1.1.

The most common causes of a fatal bus error are lack of network power or an incorrect baud rate.

4.5.3 DN_OFFLINE Command

The DN_OFFLINE command shuts down the DeviceNet interface. This command is not allowed when the scanner is active. The scanner must be shut down prior to going offline to ensure that all client connections are closed.

The format of the command buffer for the DN_OFFLINE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. DN_OFFLINE = 02h
0082h - 013Fh			Reserved

The format of the command reply data for the DN_OFFLINE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. DN_OFFLINE = 02h Contains 8002h on error. See Section 4.5.1.1.
0082h - 013Fh			Reserved

4.5.4 ADD_DEVICE Command

The ADD_DEVICE command adds a device configuration to the device list. See Section 3.5 for device list parameter details. If the scanner is active, client connections are created and the device is scanned.

The format of the command buffer for the ADD_DEVICE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. ADD_DEVICE = 03h
0082h	MacId	word	Device MAC ID (0-63).
0084h	VendorId	word	Device Vendor ID
0086h	DeviceType	word	Device Type
0088h	ProductCode	word	Device Product Code
008Ah	ProductionInhibitTime	word	For use with COS I/O connection only
008Ch	Reserved	word	Future use, set to 0
008Eh	Reserved	word	Future use, set to 0
0090h	Flags	word	Connection Flags
0092h	ExplicitSize	word	Explicit message buffer size (bytes).
0094h	ExplicitOffset	word	Offset to the explicit message buffer.
0096h	Io1Interval	word	Poll/COS/Cyclic I/O connection update rate
0098h	Output1Size	word	I/O connection 1 output size (bytes)
009Ah	Output1Offset	word	Offset to I/O connection 1 output data
009Ch	Output1LocalPathOffset	word	Offset to the local I/O connection produced path buffer.
009Eh	Output1RemotePathOffset	word	Offset to the remote (slave's) I/O connection consumed path buffer.
00A0h	Input1Size	word	I/O connection 1 input size (bytes).
00A2h	Input1Offset	word	Offset to I/O connection 1 input data
00A4h	Input1LocalPathOffset	word	Offset to the local I/O connection consumed path buffer.
00A6h	Input1RemotePathOffset	word	Offset to the remote (slave's) I/O connection produced path buffer.
00A8h	Io2Interval	word	Poll/COS/Cyclic I/O connection update rate
00AAh	Output2Size	word	I/O connection 2 output size (bytes)
00Ach	Output2Offset	word	Offset to I/O connection 2 output data
00AEh	Output2LocalPathOffset	word	Offset to path buffer containing the local I/O connection produced path.
00B0h	Output2RemotePathOffset	word	Offset to the remote (slave's) I/O connection consumed path buffer.
00B2h	Input2Size	word	I/O connection 2 input size (bytes).
00B4h	Input2Offset	word	Offset to I/O connection 2 input data.
00B6h	Input2LocalPathOffset	word	Offset to the local I/O connection consumed path buffer.
00B8h	Input2RemotePathOffset	word	Offset to the remote (slave's) I/O connection produced path buffer.
00BAh - 013Fh			Reserved

The format of the command reply data for the ADD_DEVICE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. ADD_DEVICE = 03h Contains 8003h on error. See Section 4.5.1.1.
0082h - 013Fh			Reserved

4.5.5 GET_DEVICE Command

The GET_DEVICE command retrieves a device configuration from the device list. The format of the command buffer for the GET_DEVICE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. GET_DEVICE = 04h
0082h	MacId	word	Device MAC ID (0-63).
0084h - 013Fh			Reserved

See Section 3.5 for device list parameter details.

The format of the command reply data for the GET_DEVICE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. GET_DEVICE = 04h. Contains 8004h on error. See Section 4.5.1.1.
0082h	MacId	word	Device MAC ID (0-63).
0084h	VendorId	word	Device Vendor ID
0086h	DeviceType	word	Device Type
0088h	ProductCode	word	Device Product Code
008Ah	ProductionInhibitTime	word	Product Inhibit Time for use with COS I/O connection only.
008Ch	Reserved	word	Future use
008Eh	Reserved	word	Future use
0090h	Flags	word	Connection Flags
0092h	ExplicitSize	word	Explicit message buffer size (bytes)
0094h	ExplicitOffset	word	Offset to explicit message buffer
0096h	Io1Interval	word	Poll/COS/Cyclic I/O connection update rate
0098h	Output1Size	word	I/O connection 1 output size (bytes)
009Ah	Output1Offset	word	Offset to I/O connection 1 output data.
009Ch	Output1LocalPathOffset	word	Offset to the local I/O connection produced path buffer.
009Eh	Output1RemotePathOffset	word	Offset to the remote I/O connection consumed path buffer.
00A0h	Input1Size	word	I/O connection 1 input size (bytes)
00A2h	Input1Offset	word	Offset to I/O connection 1 input data
00A4h	Input1LocalPathOffset	word	Offset to the local I/O connection consumed path buffer.
00A6h	Input1RemotePathOffset –	word	Offset to the remote I/O connection consumed path buffer.
00A8h	Io2Interval	word	Poll/COS/Cyclic I/O connection update rate
00AAh	Output2Size	word	I/O connection 2 output size (bytes)
00ACh	Output2Offset	word	Offset to I/O connection 2 output data
00AEh	Output2LocalPathOffset	word	Offset to path buffer containing the local I/O connection produced path.
00B0h	Output2RemotePathOffset	word	Offset to the remote I/O connection consumed path buffer.
00B2h	Input2Size	word	I/O connection 2 input size (bytes)
00B4h	Input2Offset	word	Offset to I/O connection 2 input data
00B6h	Input2LocalPathOffset	word	Offset to the local I/O connection consumed path buffer.
00B8h	Input2RemotePathOffset –	word	Offset to the remote I/O connection consumed path buffer.
00Bah - 013Fh			Reserved

4.5.6 DELETE_DEVICE Command

The DELETE_DEVICE command closes all related client connections and removes a device configuration from the device list.

The format of the command buffer for the DELETE_DEVICE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. DELETE_DEVICE = 05h
0082h	MacId	word	Device MAC ID (0-63).
0084h - 013Fh			Reserved

The format of the command reply data for the DELETE_DEVICE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. DELETE_DEVICE = 05h. Contains 8005h on error. See Section 4.5.1.1.
0082h - 013Fh			Reserved

4.5.7 START_SCAN Command

The START_SCAN command starts the scanner's client function. The format of the command buffer for the START_SCAN command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. START_SCAN = 06h
0082h - 013Fh			Reserved

The format of the command reply data for the START_SCAN command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. START_SCAN = 06h. Contains 8006h on error. See Section 4.5.1.1.
0082h - 013Fh			Reserved

Note

If the scanner is active or is in the process of shutting down, DNSCAN returns a command error. See Section 4.5.1.1.

4.5.8 STOP_SCAN Command

The STOP_SCAN command stops the scanner's client function. The format of the command buffer for the STOP_SCAN command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. STOP_SCAN = 07h
0082h 013Fh			Reserved

The format of the command reply data for the STOP_SCAN command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. STOP_SCAN = 07h. Contains 8007h on error. See Section 4.5.1.1.
0082h - 013Fh			Reserved

Note

If the scanner is stopped or is in the process of shutting down, DNSCAN returns a command error. See Section 4.5.1.1.

4.5.9 IO_ACTIVE Command

The IO_ACTIVE command sets the I/O mode to active and starts the application heartbeat timer. The format of the command buffer for the IO_ACTIVE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. IO_ACTIVE = 08h
0082h	Timeout	word	IoActive timeout delay (ms) (zero = infinite timeout).
0084h - 013Fh			Reserved

The format of the command reply data for the IO_ACTIVE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. IO_ACTIVE = 08h. Contains 8008h on error. See Section 4.5.1.1.
0082h - 013Fh			Reserved

Note

Once the IO_ACTIVE command has been executed, it must be executed at least once within the specified timeout period. If the timeout heartbeat timer expires before the next IO_ACTIVE command, the I/O mode is set to idle. See Section 4.5.10. The next IO_ACTIVE command returns the I/O mode to active.

4.5.10 IO_IDLE Command

The IO_IDLE command sets the I/O mode to idle and stops the application heartbeat timer. The format of the command buffer for the IO_IDLE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. IO_IDLE = 09h
0082h - 013Fh			Reserved

The format of the command reply data for the IO_IDLE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. IO_IDLE = 09h. Contains 8009h on error. See Section 4.5.1.1.
0082h - 013Fh			Reserved

Note

In I/O idle mode, all client and server I/O messages are forced to zero-length (transmit_idle). The device's reaction to zero-length messages is defined by the device vendor.

4.5.11 SCAN_ACTIVE Command

The SCAN_ACTIVE command sets the scan heartbeat to the value specified by timeout. If the application fails to issue a subsequent SCAN_ACTIVE command within the specified timeout period, the module will release connection with all devices in the scan list and cease communications. The format of the command buffer for the SCAN_ACTIVE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. SCAN_ACTIVE = 0Ah
0082h	Timeout	word	The timeout interval
0084h - 013Fh			Reserved

The format of the command reply data for the SCAN_ACTIVE command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. SCAN_ACTIVE = 0Ah. Contains 800Ah on error. See Section 4.5.1.1
0082h - 013Fh			Reserved

4.5.12 CAN_TRANSMIT Command

The CAN_TRANSMIT command allows the application to transmit CAN packets on the network. The format of the CAN_TRANSMIT command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. CAN_TRANSMIT = 0Bh
0082h	CAN_ID	word	CAN_ID of message to be transmitted
0084h	Length	byte	Number of valid data bytes within the Data field of the CAN packet. Range: 0 – 8
0085h-008Ch	Data[8]	byte	Data byte array. Maximum length of 8 bytes, specified by the Length parameter.
008Dh - 013Fh			Reserved

The format of the command reply data for the CAN_TRANSMIT command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. CAN_TRANSMIT = OBh. Contains 800Bh on error. See Section 4.5.1.1.
0082h - 013Fh			Reserved

Note

The scanner module must be online to allow successful transmission of CAN messages.

4.5.13 CAN_SETFILTER Command

The CAN_SETFILTER command enables/disables the receipt of CAN messages. This command can be issued when the scanner module is online or offline. The format of the CAN_SETFILTER command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. CAN_SETFILTER = 0Ch
0082h	CAN_ID	word	CAN_ID of message to be Enabled/Disabled. CANID 0xFFFF with Enable = TRUE/FALSE enables/ disables all CANIDs. Valid CAN IDs range from 0-2031, inclusive.
0084h	Enable	BOOL word	1 = Enable 0 = Disable
0085h - 013Fh			Reserved

The format of the command reply data for the CAN_SETFILTER command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. CAN_SETFILTER = 0Ch. Contains 800Ch on error. See Section 4.5.1.1.
0082h - 013Fh			Reserved

Note

The scanner module may internally use received messages to implement the DeviceNet protocol for communication with G2- and UCMM-capable devices. You will need to ensure that the host application does not violate this protocol while using CAN functionality.

4.5.14 CAN_SETRXQ Command

The CAN_SETRXQ command initializes the size and location of the CAN receive queue. This command can be issued when the scanner module is online or offline. The memory address is chosen from the free memory pool in the host interface memory area.

A CAN receive queue does not exist unless this command is successfully issued once. The Offset and QueueSize values should be such that Rx Queue does not exceed 0x3FFFhex, which marks the end of the free memory pool. See Section 4.15.

The format of the CAN_SETRXQ command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command word. CAN_SETRXQ= 0Dh
0082h	QueueOffset	word	The memory address of the CAN Rx Queue. Its value can be in the range 0x1000h - 0x3FEEh, inclusive. See Section 4.15.
0084h	QueueSize	word	Indicates the number of CAN message buffers within the queue. The minimum value the QueueSize can have is 1. The maximum value depends on the queue offset, but the absolute maximum is 0x045Ch.
0086h - 013Fh			Reserved

The format of the command reply data for the CAN_SETRXQ command is:

Offset	Name	Data Type	Description
0080h	Command	word	Command ack. CAN_SETRXQ = 0Dh. Contains 800Dh on error. See Section 4.5.1.1.
0082h - 013Fh			Reserved

Note

When specifying the CAN Receive queue's address and size, avoid overlapping other host application-defined buffers, like explicit messaging, path or I/O buffers.

4.6 Client Status Block

Offset: 0140h - 014Fh

The Client Status Block provides status and event notification related to the scanner's client function.

4.6.1 Client Status Block

Byte Offset	Access Rule	Name	Description
0140h	R	Status Code	Client Status Code
0141h	R	Status Flags	Client Status Flags
0142h	R/W [1]	Scan Event Flags	Event flags related to the Group 2 Master/Slave I/O scan.
0143h - 014Fh			Reserved

1 Host may only write zero to this byte, and only if the byte is non-zero.

4.6.1.1 C Prototype

```
typedef struct
{
   unsigned char StatusCode;
   unsigned char StatusFlags;
   unsigned char ScanEventFlags;
   unsigned char Reserved[13];
} CLIENT_STATUS;
```

4.6.2 Status Code

The Status Code indicates the current status of the scanner's client function. This byte is written by the scanner and must not be written by the host.

Bit Flag	Meaning
0	0 – Idle (not scanning) 1 – Active (scanning)
1	Tx Idle – If this bit is set, the scanner is transmitting Idle output messages (zero-length, no data) to all devices in the scan list. All server input messages (data sent to client) are also Idle (zero-length, no data). See Sections 4.5.9 and 4.5.10.
2 – 7	Reserved

4.6.3 Status Flags

The Status Flags report the status of the scanner's client function. This byte is written by the scanner and must not be written by the host.

Bit	Flag	Description
0	IL	Global I/O data input area interlock. The scanner sets this flag when it is writing **any** device I/O data input areas.
1	OL	Global I/O data output area interlock. The scanner sets this flag when it is reading any device I/O data output areas.
2-7	Reserved	Reserved

4.6.4 Scan Event Flags

The Scan Event Flags inform the host of event occurrences related to the scanner's client function. The host may only write 0 to this byte, and only if it is non-zero.

Bit	Flag	Description
0	S	Start of scan. This event is generated once per I/O scan. This event is asynchronous to I/O data updates and does not replace the I/O data interlocks.
1-7		Reserved

4.7 Client Control Block

Offset: 0150h - 015Fh

The Client Control Block provides control functions related to the scanner's client function.

4.7.1 Client Control Block

Byte Offset	Access Rule	Name	Description
0150h	R/W	Control Flags	Client Control Flags
0151h - 015Dh			Reserved
015Eh 015Fh	R/W	Event Queue Enable Flags	Event notification queue enable flags

4.7.1.1 C Prototype

```
typedef struct
{
   unsigned char ControlFlags;
   unsigned char Reserved[13];
   unsigned short EventQueueEnableFlags;
} CLIENT_CONTROL;
```

4.7.2 Control Flags

The control flags control the operation of the scanner's client function. This byte is written by the host and is not written by the scanner.

Bit	Flag	Description
0	IL	Global I/O data Input Interlock. The scanner will not write any device input data areas when this bit is set.
1	OL	Global I/O data output Interlock. The scanner will not read any device output data areas when this bit is set.
2-7		Reserved, set to 0

4.7.3 Event Queue Enable Flags

The Event Queue Enable Flags determine which client status events are posted into the event notification queue. When a client status event occurs, the event is posted into the event notification queue if the corresponding event queue enable flag is set.

Bit	Flag	Description
0	C	Client status code changed event enable
1		Reserved, set to 0
2	S	Client scan event enable
3-15		Reserved, set to 0

4.8 Server Status Block

Offset: 0160h - 016Fh

The Server Status Block provides status and event notification related to the scanner's server function.

Note

The explicit message server functions in the Server Status Block support explicit request messages from either the Group 2 Master/Slave Connection Set or dynamic explicit scanner connections.

4.8.1 Server Status Block

Byte Offset	Access Rule	Name	Description
0160h	R	Status Code	Server Status Code
0161h	R	Status Flags	Server Status Flags
0162h	R/W [1]	I/O 1 Event Flags	Event flags related to the first I/O area.
0163h	R/W [1]	I/O 2 Event Flags	Event flags related to the second I/O area.
0164h	R/W [1]	Explicit Request Event Flags	Event flags related to the explicit request message buffer.
0165h	R/W [1]	Explicit Response Event Flags	Event flags related to the explicit response message buffer.
0166h	R	AllocatedG2Connections	Group 2 Server Connection Status
0167h - 016Fh			Reserved

1 Host may only write zero to this byte, and only if the byte is non-zero.

4.8.1.1 C Prototype

```
typedef struct
{
   unsigned char StatusCode;
   unsigned char StatusFlags;
   unsigned char Io1EventFlags;
   unsigned char Io2EventFlags;
   unsigned char ExplicitRequestEventFlags;
   unsigned char ExplicitResponseEventFlags;
   unsigned char AllocatedG2Connections;
   unsigned char Reserved[9];
} SERVER_STATUS;
```

4.8.2 Status Code

The Status Code reports the scanner's server function. This byte is written by the scanner and must not be written by the host.

Status	Meaning
00h	Idle (Group 2 Master/Slave Connection Set not allocated)
01h	Active (Group 2 Master/Slave Connection Set allocated)
02h - FFh	Reserved

4.8.3 Status Flags

The Status Flags report the status of the scanner's server function. This byte is written by the scanner and must not be written by the host.

Bit	Flag	Description
0	EX	Explicit Request Message Buffer Interlock. This flag is set when the scanner is writing the explicit request message buffer.
1	I1	Input Area 1 Interlock. This flag is set when the scanner is reading the first input area.
2	I2	Input Area 2 Interlock. This flag is set when the scanner is reading the second input area.
3	O1	Output Area 1 Interlock. This flag is set when the scanner is writing the first output area.
4	O2	Output Area 2 Interlock. This flag is set when the scanner is writing the second output area.
5	Z1	Output Area 1 receive_idle condition. This flag is set when the first I/O connection receives a zero-length message and is cleared when a non-zero length message is consumed.
6	Z2	Output Area 2 receive_idle condition. This flag is set when the second I/O connection receives a zero-length message and is cleared when a non-zero length message is consumed.
7		Reserved

4.8.4 I/O Event Flags

The I/O Event Flags inform the host of event occurrences related to the scanner's server function. Each I/O area has an independent event flag byte. The host may only write 0 to this byte, and only if it is non-zero.

Bit	Flag	Description
0	U	Output data update. This bit is set every time a valid I/O message is received from the I/O connection.
1	Z	Receive_Idle. This bit is set every time a zero-length I/O message is received from the I/O connection.
2-7		Reserved

4.8.5 Explicit Request Event Flags

The Explicit Request Event Flags inform the host of event occurrences related to the server's Explicit Request Message Buffer. The host may only write 0 to this byte, and only if it is non-zero.

Note

Explicit requests forwarded to the server explicit message buffer may have been received by the Group 2 Master/Slave Connection Set explicit connection or a dynamic explicit connection opened through the UCMM.

Bit	Flag	Description
0	R	Received explicit request. This bit is set when a received explicit request is addressed to an object class not supported by the scanner. This bit is also set when an explicit request is addressed to an instance of the Identity Object other than 1. The host application must service the request and send a response.
1-7		Reserved

4.8.6 Explicit Response Event Flags

The Explicit Response Event Flags are used to inform the host of event occurrences related to the server's Explicit Response Message Buffer. The host may only write 0 to this byte, and only if it is non-zero.

Bit	Flag	Description
0	S	Explicit response sent. This bit is set when the scanner has submitted the message in the explicit response buffer for transmission. This event signals the host that the scanner is ready for the next explicit response.
1-7		Reserved

4.8.7 Allocated G2 Connection Flags

The Allocated G2 Connection Flags indicate the currently allocated G2 connections for the scanner's server function.

Offset	Bit							
	7	6	5	4	3	2	1	0
0	Res.	AKS	CYC	COS	Res.	ST	P	EX

Group 2 only I/O Connection Flags

AKS Acknowledge suppress enabled
CYC Cyclic I/O connection allocated
COS Change-of-state I/O connection allocated
ST Bit-strobed I/O connection allocated
P Polled I/O connection allocated

Group 2 Explicit Connection Flag

EX Explicit connection allocated

4.9 Server Control Block

Offset: 0170h - 017Fh

The Server Control Block provides event trigger and control functions related to the scanner's server function.

Note

The explicit message server functions in the Server Control Block support explicit request messages from either the Group 2 Master/Slave Connection Set or dynamic explicit connections to the scanner.

4.9.1 Server Control Block

Byte Offset	Access Rule	Name	Description
0170h	W	Control Flags	Server Control Flags
0171h	R/W [1]	I/O 1 Event Flags	Event flags related to the first I/O area.
0172h	R/W [1]	I/O 2 Event Flags	Event flags related to the second I/O area.
0173h	R/W [1]	Explicit Request Event Flags	Event flags related to the explicit request message buffer.
0174h	R/W [1]	Explicit Response Event Flags	Event flags related to the explicit response message buffer.
0175h - 017Dh			Reserved
017Eh 017Fh	R/W	Event Queue Enable Flags	Event notification queue enable flags

1 Host may not write to this byte if it is non-zero.

4.9.1.1 C Prototype

```
typedef struct
{
   unsigned char ControlFlags;
   unsigned char Io1EventFlags;
   unsigned char Io2EventFlags;
   unsigned char ExplicitRequestFlags;
   unsigned char ExplicitResponseFlags;
   unsigned char Reserved[9];
   unsigned short EventQueueEnableFlags;
} SERVER_CONTROL;
```

4.9.2 Control Flags

The Control Flags control the scanner's server function. This byte is written by the host and is not written by the scanner.

Bit	Flag	Description
0	EX	Explicit Response Message Buffer Interlock. The scanner will not access the explicit response message buffer when this bit is set.
1	I1	Input Area 1 Interlock. The scanner will not read the first input area when this bit is set.
2	I2	Input Area 2 Interlock. The scanner will not read the second input area when this bit is set.
3	O1	Output Area 1 Interlock. The scanner will not write the first output area when this bit is set.
4	O2	Output Area 2 Interlock. The scanner will not write the second output area when this bit is set.
5-7		Reserved

4.9.3 I/O Event Flags

The I/O Event Flags are used to inform the scanner of event occurrences related to its server function. Each I/O area has an independent event flag byte. The host may only write to this byte if it is zero.

Bit	Flag	Description
0	C	Input Data Changed. The host sets this bit to notify the scanner of changed input data. This event may trigger a message production depending on the I/O connection configuration.
1-7		Reserved

4.9.4 Explicit Request Event Flags

The Explicit Request Event Flags inform the scanner of event occurrences related to the server's Explicit Request Message Buffer. The host may only write to this byte if it is zero.

Bit	Flag	Description
0	R	Explicit request received. This bit is set when the host application has read the message in the explicit request buffer. This event signals the scanner that the host is ready for the next explicit request.
1-7		Reserved

4.9.5 Explicit Response Event Flags

The Explicit Response Event Flags inform the scanner of event occurrences related to the server's Explicit Response Message Buffer. The host may only write to this byte if it is zero.

Bit	Flag	Description
0	S	Send Explicit Response. The application sets this flag after it has processed an explicit request event and written an appropriate response to the explicit response message buffer.
1-7		Reserved

4.9.6 Event Queue Enable Flags

The event queue enable word determines which server status events are posted into the event notification queue. When a server status event occurs, the event is posted into the event notification queue if the corresponding event queue enable flag is set.

Bit	Flag	Description
0	C	Server status code changed event enable
1		Reserved, set to 0
2	1	I/O area 1 event enable
3	2	I/O area 2 event enable
4	Q	Explicit request event enable
5	R	Explicit response event enable
6	CF	Server G2 Connection Flags status event enable
7-15		Reserved, set to 0

4.10 Device Control Event Table

Offset: 0180h - 01BFh

The Device Control Event Table is an array of event flag bytes, one for each MAC ID. This table notifies the scanner of which devices have pending control events.

Byte Offset	Access Rule	Description
0180h	R/W [1,2]	Device Control Event Flag for MAC ID 0
0181h	R/W [1,2]	Device Control Event Flag for MAC ID 1
-	-	-
01BFh	R/W [1,2]	Device Control Event Flag for MAC ID 63

1 Host only writes '1', and only if byte is zero.

2 Scanner only writes zero, and only if byte is non-zero.

The host must write a '1' to the Device Control Event Table when any event flags are written in the Device Control Table for the corresponding device. The host must not write to the Device Control Event Table if the value is already non-zero.

The scanner writes zero to the Device Status Table before handling the event and clearing the event flag(s) in the Device Control Table.

4.11 Device Status Table

Offset: 01C0h - 05BFh

The Device Status Table provides status and event notification related to the scanner's client function.

The Device Status Table is an array of DeviceStatus structures, one for each MAC ID. The first element corresponds to MAC ID 0.

4.11.1 Device Status Structure

Byte Offset	Access Rule	Name	Description
0	R	Status Code	Device Status Code
1	R [2]	Status Flags	Device Status Flags
2	R/W [1, 2]	I/O 1 Event Flags	Event flags related to the first I/O area.
3	R/W [1, 2]	I/O 2 Event Flags	Event flags related to the second I/O area.
4	R/W [1]	Explicit Event Flags	Event flags related to the explicit message buffer.
5	R [2]	Connection Allocation Flags	Flags reflecting the currently allocated connections.
7-15			Reserved

1 Host may only write zero to this byte, and only if the byte is non-zero.

2 Not Applicable for Local Explicit Messaging via Host (see Section 3.11).

4.11.1.1 C Prototype

```
typedef struct
{
   unsigned char StatusCode;
   unsigned char StatusFlags;
   unsigned char Io1EventFlags;
   unsigned char Io2EventFlags;
   unsigned char ExplicitEventFlags;
   unsigned short ConnectionAllocationFlags;
   unsigned char Reserved[9];
} DEVICE_STATUS;
```

4.11.2 Status Code

The status code indicates the status of the client's connection to the device. This byte is written by the scanner and must not be written by the host.

Status	Meaning
00h	Device not in device list
01h	Device idle (not being scanned)
02h	Device being scanned
03h	Device timed-out
04h	UCMM connection error
05h	Master/Slave connection set is busy
06h	Error allocating Master/Slave connection set
07h	Invalid vendor ID
08h	Error reading vendor ID
09h	Invalid device type
0Ah	Error reading device type
0Bh	Invalid product code
0Ch	Error reading product code
0Dh	Invalid I/O connection 1 input size
0Eh	Error reading I/O connection 1 input size
0Fh	Invalid I/O connection 1 output size
10h	Error reading I/O connection 1 output size
11h	Invalid I/O connection 2 input size
12h	Error reading I/O connection 2 input size
13h	Invalid I/O connection 2 output size
14h	Error reading I/O connection 2 output size
15h	Error setting I/O connection 1 packet rate
16h	Error setting I/O connection 2 packet rate
17h	M/S connection set sync fault
18h	Error setting Production Inhibit Time
19h - FFh	Reserved

4.11.3 Status Flags

The Status Flags inform the host of the device connection status. This byte is written by the scanner and must not be written by the host.

Bit	Flag	Description
0		Reserved
1	I1	Input Area 1 Interlock. The scanner sets this flag when it is writing the first input area.
2	I2	Input Area 2 Interlock. The scanner sets this flag when it is writing the second input area.
3	O1	Output Area 1 Interlock. The scanner sets this flag when it reading the first output area.
4	O2	Output Area 2 Interlock. The scanner sets this flag when it is reading the second output area.
5	Z1	Input Area 1 receive_idle condition. This flag is set when the first I/O connection receives a zero-length message, and is cleared when a non-zero length message is consumed.
6	Z2	Input Area 2 receive_idle condition. This flag is set when the second I/O connection receives a zero-length message, and is cleared when a non-zero length message is consumed.
7		Reserved

4.11.4 I/O Event Flags

The I/O Event Flags inform the host of event occurrences related to the client I/O areas. Each I/O area has an independent event flag byte.

Bit	Flag	Description
0	U	Input data update. This bit is set every time a valid I/O message is received from the I/O connection.
1	Z	Receive_Idle. This bit is set every time a zero-length I/O message is received from the I/O connection.
2-7		Reserved

4.11.5 Explicit Messaging Event Flags

The Explicit Messaging Event Flags inform the host of event occurrences related to the client's Explicit Message Buffer.

Bit	Flag	Description
0	R	Received explicit response. This bit is set when a response to a previously transmitted explicit request is received.
1	RX	Received excessive length explicit response. This bit has the same function as the R flag, below, except it also indicates that the received explicit response has been truncated to fit the buffer.
2-7		Reserved

4.11.6 Connection Allocation Flags

The Connection Allocation Flags inform the host of the device connections currently allocated by the scanner.

Offset	Bit							
	7	6	5	4	3	2	1	0
5	Res.	AKS	CYC	COS	Res.	ST	P	EX
6	Reserved							3X

AKS	Acknowledge suppress enabled
CYC	Cyclic I/O connection established
COS	Change-of-state I/O connection established
ST	Bit-strobed I/O connection established
P	Polled I/O connection established
EX	Group 2 explicit messaging connection established
3X	Group 3 dynamic explicit messaging connection established

The Connection Allocation Flags are updated whenever the Status Code (see Section 4.11.2) changes.

4.12 Device Control Table

Offset: 05C0h - 09BFh

The Device Control Table provides event trigger and control functions related to the scanner's client function.

The Device Control Table is an array of DeviceControl structures, one for each MAC ID. The first element corresponds to MAC ID 0.

4.12.1 DeviceControl Structure

Byte Offset	Access Rule	Name	Description
0	W	Control Flags	Device Control Flags
1	R/W [1, 2]	I/O 1 Event Flags	Event flags related to the first I/O area.
2	R/W [1, 2]	I/O 2 Event Flags	Event flags related to the second I/O area.
3	R/W [1]	Explicit Event Flags	Event flags related to the explicit message buffer.
4-13	R	Reserved	Reserved
14 15	W	Event Queue Enable Flags	Event notification queue enable flags.

1 Host may not write to this byte if it is non-zero.

2 Not Applicable for Local Explicit Messaging via Host (see Section 3.11).

4.12.1.1 C Prototype

```
typedef struct
{
   unsigned char ControlFlags;
   unsigned char Io1EventFlags;
   unsigned char Io2EventFlags;
   unsigned char ExplicitEventFlags;
   unsigned char Reserved[10];
   unsigned short EventQueueEnableFlags;
} DEVICE_CONTROL;
```

4.12.2 Control Flags

The Control Flags control the scanner's client function. This byte is written by the host and is not written by the scanner.

Bit	Flag	Description
0	EX	Explicit Message Buffer Interlock. The scanner will not access the explicit message buffer when this bit is set.
1	I1	Input Area 1 Interlock. The scanner will not write the first input area when this bit is set.
2	I2	Input Area 2 Interlock. The scanner will not write the second input area when this bit is set.
3	O1	Output Area 1 Interlock. The scanner will not read the first output area when this bit is set.
4	O2	Output Area 2 Interlock. The scanner will not read the second output area when this bit is set.
5-7		Reserved

4.12.3 I/O Event Flags

The I/O Event Flags inform the scanner of client I/O event occurrences. Each I/O area has an independent event flag byte.

Bit	Flag	Description
0	C	Output Data changed. The host sets this bit to notify the scanner of changed output data. This event may trigger a message production, depending on the I/O connection configuration. Modification of this bit affects both COS connections and Poll I/O connections with a non-zero I/O Interval.
1-7		Reserved, set to 0.

4.12.4 Explicit Messaging Event Flags

The Explicit Messaging Event Flags are used to inform the scanner of event occurrences related to the Explicit Message Buffer associated with the scanner's client function.

Bit	Flag	Description
0	S	Send explicit request. The application sets this flag to transmit the explicit request in the explicit message buffer.
1-7		Reserved, set to 0.

4.12.5 Event Queue Enable Flags

The Event Queue Enable Word determines which device status events are posted into the event notification queue. When a device status event occurs, the event is posted into the event notification queue if the corresponding Event Queue Enable Flag is set.

Bit	Flag	Description
0	C	Device status code changed event enable (includes Connection Allocation Flags update).
1		Reserved, set to 0.
2	1	I/O area 1 event enable
3	2	I/O area 2 event enable
4	R	Explicit response event enable
5-15		Reserved, set to 0.

4.13 Event Notification Queue

Offset: 09C0h - 0DBFh

The event notification queue minimizes the overhead required to pass scanner events to the host application. Each event may be individually enabled for notification, reducing the number of interrupts that must be processed by the host application to the actual number of events being monitored.

Offset	Name	Data Type	Description
0x09C0	QueueStatusA	byte	Queue status
0x09C1	QueueStatusB	byte	
0x09C2	QueueIn[1,2]	word	Queue in offset, updated by the scanner only (0x09C6 initially).
0x09C4	QueueOut[1,2]	word	Queue out offset, updated by the host application only (0x09C6 initially).
0x09C6 - 0x0DBF	Event Queue	Event[509]	Event queue entries

1 If QueueIn is equal to QueueOut, the queue is empty.

2 If QueueIn + 2 is equal to QueueOut, the queue is full.

4.13.1 Queue Status

The queue status bytes contain flags that indicate various error conditions. The scanner sets status bits and the host must clear the bits to detect the next occurrence.

Bit	Flag	Description
0	O	Queue overrun. The scanner discarded an event because the event queue was full.
1-7		Reserved

4.13.1.1 Queue Status Access Rules

The scanner writes QueueStatusB first, followed by QueueStatusA. The host application must:

1. Read QueueStatusA and store the result in temporary variable ‘A’.
2. Clear relevant bits in QueueStatusA.
3. Read QueueStatusB and store the result in temporary variable ‘B’.
4. Clear the relevant bits in QueueStatusB.
5. Logically OR temporary variables ‘A’ and ‘B’ to determine queue status flags.

Note

The only possible result of an access collision between the scanner and the host application is a queue status flag remaining set after the host has attempted to clear it.

4.13.2 QueueIn

The QueueIn word contains the memory offset of the next event entry to be written by the scanner.

The scanner updates QueueIn. The host must not write to QueueIn.

4.13.3 QueueOut

The QueueOut word contains the memory offset of the next event entry to be read by the host application.

The scanner initializes QueueOut when the module is loaded and does not write to QueueOut at any other time.

4.13.4 Event Structure

The Event Structure contains one event in the event notification queue.

Offset	Name	Data Type	Description
0	Source	byte	Event source
1	EventId	byte	EventId

4.13.4.1 Event Source

Source	Description
0	Device 0 Event
-	-
63	Device 63 Event
64	Client Event
65	Server Event
66-255	Reserved

4.13.4.2 EventId

The EventId is the byte offset from the start of the corresponding status block to the event flag byte.

Event	ID
Device Status Code Change Event	0
Device I/O 1 Event	2
Device I/O 2 Event	3
Device Explicit Messaging Event	4
Client Status Code Change Event	0
Client Scan Event	2
Server Status Code Change Event	0
Server I/O 1 Event	2
Server I/O 2 Event	3
Server Explicit Request Event	4
Server Explicit Response Event	5
Server Group 2 Connection Flags Status Change Event	6

4.14 Event Trigger Queue

Offset: 0DC0h – 0FBFh

The Event Trigger Queue minimizes the overhead required to pass application events to the scanner module. Using the event trigger queue reduces processing for the scanner module, providing quicker event response and more stable repeatability.

Offset	Name	Data Type	Description
0x0DC0	QueueStatusA	byte	Queue status
0x0DC1	QueueStatusB	byte	
0x0DC2	QueueIn[1,2]	Word	Queue in offset, updated by the host application only (0x0DC6 initially).
0x0DC4	QueueOut[1,2]	Word	Queue out offset, updated by the scanner only (0x0DC6 initially).
0x0DC6 - 0x0FBF	Event Queue	Event[506]	Event queue entries

1 If QueueIn +1 is equal to QueueOut, the queue is full.

2 If QueueIn is equal to QueueOut, the queue is empty

4.14.1 Queue Status

The Queue Status Bytes contain flags that indicate various error conditions. The scanner sets status bits and the host must clear the bits to detect the next occurrence.

Bit	Flag	Description
0	I	Invalid trigger event
1-7		Reserved

4.14.1.1 Queue Status Access Rules

The scanner writes QueueStatusB first, followed by QueueStatusA. The host application must:

1. Read QueueStatusA and store the result in temporary variable ‘A’.
2. Clear relevant bits in QueueStatusA.
3. Read QueueStatusB and store the result in temporary variable ‘B’.
4. Clear relevant bits in QueueStatusB.
5. Logically OR temporary variables ‘A’ and ‘B’ to determine queue status flags.

Note

The only possible result of an access collision between the scanner and the host application is a queue status flag remaining set after the host has attempted to clear it.

4.14.2 QueueIn

The QueueIn word contains the memory offset of the next event entry to be written by the host application.

The host application updates QueueIn. The scanner initializes QueueIn when the module is loaded and does not write to QueueIn at any other time.

4.14.3 QueueOut

The QueueOut word contains the memory offset of the next event entry to be read by the scanner module.

The host application must not modify QueueOut.

4.14.4 Event Structure

The event structure contains one event in the event notification queue.

Offset	Name	Data Type	Description
0	Source	Byte	Event source

4.14.4.1 Event Source

Source	Description
0	Device 0 Event
-	-
63	Device 63 Event
-	Reserved for future use
65	Server Event
66	Command Event
67-255	Reserved

4.15 CAN Rx Queue

The CAN Rx Queue allows the host application to receive CAN packets. Only packets whose CAN ID matches the CAN IDs enabled by the CAN_SETFILTER command will be available in the CAN Rx Queue.

The end of an RxQueue is given by:

```
RxQueue End = QueueOffset + 6 + 11 x QueueSize
```

Offset	Name	Data Type	Description
QueueOffset[1]	QueueStatusA	byte	Queue status
QueueOffset + 1	QueueStatusB	byte	
QueueOffset + 2	QueueIn[2,3]	word	Queue in offset. Updated by the scanner only.
QueueOffset + 4	QueueOut[2,3]	word	Queue out offset. Updated by the host application only.
QueueOffset + 6	CANMessage	CANMessage[QueueSize[4]]	CAN Rx entries

1 Offset for CAN Rx Queue is determined by the CAN_SETRXQ command. See Section 4.5.14.

2 If QueueIn is equal to QueueOut, the queue is empty.

3 If QueueIn + 11 (size of CANMessage data type) is equal to QueueOut, the queue is full.

4 Queue Size is determined by the CAN_SETRXQ command. See Section 4.5.14.

4.15.1 Queue Status

The Queue Status bytes contain flags that indicate various error conditions. The scanner sets status bits and the host must clear the bits to detect the next occurrence.

Bit	Flag	Description
0	O	Queue overrun. The scanner discarded a message because the CAN Rx Queue was full.
1-7		Reserved

4.15.1.1 Queue Status Access Rules

The scanner writes QueueStatusB first, followed by QueueStatusA. The host application must:

1. Read QueueStatusA and store the result in temporary variable ‘A’.
2. Clear relevant bits in QueueStatusA.
3. Read QueueStatusB and store the result in temporary variable ‘B’.
4. Clear relevant bits in QueueStatusB.
5. Logically OR temporary variables ‘A’ and ‘B’ to determine queue status flags.

Note

The only possible result of an access collision between the scanner and the host application is a queue status flag remaining set after the host has attempted to clear it.

4.15.2 QueueIn

The QueueIn word contains the memory offset of the next Rx Queue buffer to be written by the scanner.

The scanner updates QueueIn. The host must not write to QueueIn.

4.15.3 QueueOut

The QueueOut word contains the memory offset of the next Rx Queue buffer to be read by the host application.

The scanner initializes QueueOut when the module is loaded and does not write to QueueOut at any other time.

4.15.4 CANMessage Structure

The CAN Message structure contains the received CAN Message.

Offset	Name	Data Type	Description
0	CANID	word	CAN ID of message received
1	Length	byte	Number of valid data bytes in the packet
2	Data[8]	byte	Data portion of packet

4.15.5 CAN ID

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved for future use					CAN ID										

5

DeviceNet Scanner Module 32-Bit DLL

Chapter Contents:

5.1 Introduction

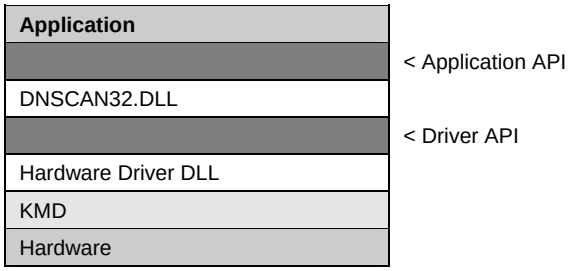
DNSCAN32.DLL is a 32-bit DLL that provides a common interface that abstracts physical hardware to allow one application or DLL to interface with the DeviceNet Scanner module on different hardware platforms.

5.2 Services

DNSCAN32.DLL provides the following services:

- DLL revision information
- Hardware driver management
- Card management (multiple cards, multiple clients)
- Card abstraction using handles to eliminate the need for the application to be aware of memory or I/O addresses
- Event notification
- I/O data access with interlock handling
- Explicit message interface with interlock handling
- Thread-safe APIs

5.3 Application Stack



5.4 Typical Application Operation

An application using the DLL typically follows this sequence for startup:

1. Load the hardware driver DLL (SSDN32.DLL).
2. Open the card and retrieve a CardHandle.
3. Put the scanner module online.
4. Add devices to the scan list.
5. Start scanning.
6. Monitor device status for connection establishment.
7. In Normal Operation Mode, the user can call other functions provided by the DLL. For example, sending explicit messages.
8. When an error occurs, it is reported by calling the standard WIN32 API GetLastError.

To terminate properly, the application should typically follow this sequence:

1. Stop scanning.
2. Wait for All Device Status to indicate “Idle – not being scanned”.
3. Go offline.
4. Close the card connection.
5. Unload the hardware driver DLL.

DLL Thread Safety

A thread-safe DLL allows you to develop a multi-threaded application without having to synchronize the DLL API calls to guarantee data and functional integrity. The following list outlines the key considerations when using the DLL:

- The thread-safe release of Dnscan32.dll (2.48.xx.xx) is backward-compatible with the previous DLL releases
- You do not need to call any special synchronization functions, as all synchronization in handled in the DLL
- Using the thread-safe implementation, you can break down system tasks in a modular way, distributing them among various threads. For example, independent threads can be created for dealing with explicit messaging, and additional threads for I/O transactions with any slave device.
- The application is responsible for maintaining any application-level variables

Note

For more details, refer to the DNSCAN32.DLL Thread Safety Overview Technical Note (document number 716-0010).

Figure 1: Single-Threaded Application Example:

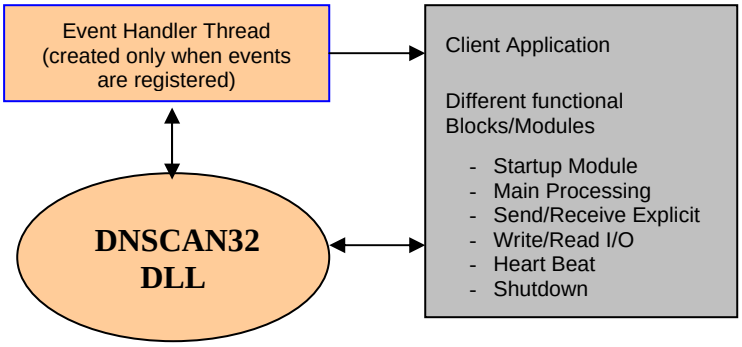
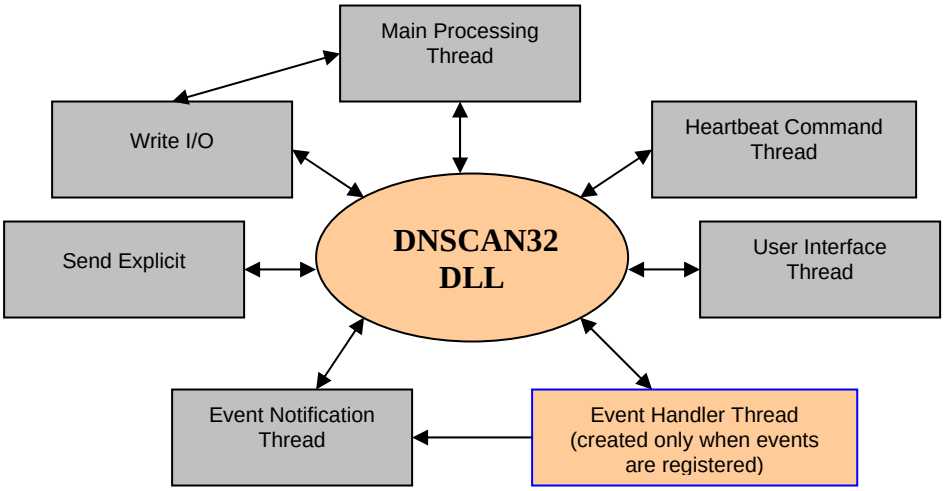


Figure 2: Multi-Threaded Application Example



5.4.1 Event Polling

```
//---------------------------------------------------------------------
// Pseudo code for an application using event polling
//---------------------------------------------------------------------

Main()
{
   DNS_LoadDriver( ... ) // load the hardware driver DLL (SSDN32.DLL)
   DNS_OpenCard( ... ) // open the card and retrieve a CardHandle
   DNS_AddDevice( ... ) // add a device to the scan list
   DNS_Online( ... ) // put the scanner module online
   DNS_StartScan( ... ) // start scanning devices within the scan list

   Loop
   {
        DNS_GetDeviceStatus( ... ) // retrieve status of device being scanned

        if Device Status = Active
        {
             DNS_GetDeviceEvent( ... ) // retrieve the event flags for the
                                       // required I/O event area
             if I/O event is present
             {
                  DNS_ReadDeviceIo( ... ) // read the I/O data for the device
             }
        }
   }

   DNS_StopScan( ... ) // stop scannning
   DNS_Offline( ... ) // go offline
   DNS_DeleteDevice( ... ) // remove the device from the scanlist
   DNS_CloseCard( ... ) // close the card connection
   DNS_FreeDriver() // unload the hardware driver DLL
}
```

5.4.2 Event Notification

```
//-----------------------------------------------------------------------
// Pseudo code for an application using event notification
//-----------------------------------------------------------------------
Main()
{
   DNS_LoadDriver( ... ) // load the hardware driver DLL (SSDN32.DLL)
   DNS_OpenCard( ... ) // open the card and retrieve a CardHandle

   CreateThread( ... ) // thread used to receive event notification messages

   DNS_RegisterDeviceEvent( ... )  // register for device event notification

   DNS_AddDevice( ... ) // add a device to the scan list
   if AddDevice fails
   {
        DNS_UnRegisterDeviceEvent( ... ) // unregister device events
   }

   DNS_RegisterClientEvent( ... ) // register for client events

   DNS_Online( ... ) // put the scanner module online
   if Online fails
   {
        DNS_UnRegisterClientEvent( ... ) // unregister client events
   }

   DNS_StartScan( ... ) // start scanning devices within the scan list

   Loop
   {
        loop until application is to be terminated
        TerminateThread( ... ) // close the event thread
   }

   DNS_StopScan( ... ) // stop scannning
   DNS_Offline( ... ) // go offline
   DNS_DeleteDevice( ... ) // remove the device from the scanlist
   DNS_UnRegisterDeviceEvent( ... ) // unregister all registered device events
   DNS_UnRegisterClientEvent( ... ) // unregister all registered client events
   DNS_CloseCard( ... ) // close the card connection
   DNS_FreeDriver() // unload the hardware driver DLL
}
```

```
//-----------------------------------------------------------------------------

        // Pseudo code for event notification thread
        // Handles event messages from DNSCAN32.DLL
        //-------------------------------------------------------------------------------------
        ---
        EventThread()
        {
           GetMessage( ... )
           {
                switch( MessageId )
                {
                     case DeviceIo1Message:
                          DNS_ReadDeviceIo( ... ) // read device I/O 1 data
                          break;

                     case DeviceIo2Message:
                          DNS_ReadDeviceIo( ... ) // read device I/O 2 data
                          break;

                     case DeviceStatusMessage:
                          DNS_GetDeviceStatus( ... ) // retrieve the device status
                          break;

                     case ClientStatusMessage:
                          DNS_GetClientStatus( ... ) // retrieve the client status
                          break;
                }
           }
        }
```

5.4.3 Client Explicit Messaging

```
//---------------------------------------------------------------------------
// Pseudo code for an application using client explicit messaging
//---------------------------------------------------------------------------
Main()
{
   DNS_LoadDriver( ... ) // load the hardware driver DLL (SSDN32.DLL)
   DNS_OpenCard( ... ) // open the card and retrieve a CardHandle

   DeviceCfg.MacId = 1;
   DeviceCfg.VendorId = 0;
   DeviceCfg.DeviceType = 0;
   DeviceCfg.ProductCode = 0;
   DeviceCfg.Flags = 1; // explicit only connection
   DeviceCfg.ExplicitSize = 32;
   DeviceCfg.ExplicitOffset = 0x1000;

   DNS_AddDevice( ... ) // add a device to the scan list

   DNS_Online( ... ) // put the scanner module online

   DNS_StartScan( ... ) // start scanning devices within the scan list

   While(?)
        {
         // set somewhere by something ???
         if( bQueryDevice == TRUE )
                {
                    DNS_SendDeviceExplicit(….);

                    Do
                        {
                          // query device explicit event byte looking for a response event
                          DNS_GetDeviceEvent(…..);
                        }While( no explicit response event);
                }
        }

   DNS_StopScan( ... ) // stop scannning
   DNS_Offline( ... ) // go offline
   DNS_DeleteDevice( ... ) // remove the device from the scanlist
   DNS_CloseCard( ... ) // close the card connection
   DNS_FreeDriver() // unload the hardware driver DLL
}
```

5.4.4 Server Explicit Messaging

```
//-----------------------------------------------------------------------
// Pseudo code for an application using server explicit messaging
//-----------------------------------------------------------------------
Main()
{
   DNS_LoadDriver( ... ) // load the hardware driver DLL (SSDN32.DLL)
   DNS_OpenCard( ... ) // open the card and retrieve a CardHandle

   ServerCfg.MacId = 1;
   ServerCfg.BaudRate = BAUD_125K;
   ServerCfg.ScanInterval = 0;
   ServerCfg.Flags = 1; // explicit only connection
   ServerCfg.ExplicitRequestSize = 32;
   ServerCfg.ExplicitRequestOffset = 0x1000;
   ServerCfg.ExplicitResponseSize = 32;
   ServerCfg.ExplicitResponseOffset = 0x1100;

   DNS_Online( ... ) // put the scanner module online

   DNS_StartScan( ... ) // start scanning devices within the scan list

   While(?)
        {
         if( GetServerStatus(….) == ACTIVE )
         {
                   // check for server explicit messaging request events
              DNS_GetServerEvent(….);
                   if ( ServerEvent == Receive Explicit Request )
                   {
                   // retrieve the server explicit request
                   DNS_RecieveServerExplicit(….);

                             // handle the explicit request (may be implemented as an
                             event or a direct
                             // function call etc….)
                    HandleRequest(….);
                   }

                       // if an explicit response is ready and the explicit response
                       buffer is available // send it
                   if( ExplicitResponse == READY )
                       {
                        if( DNS_SendServerExplicit(….) == EXPLICIT_MESSSAGE_PENDING )
                             {
                                    // if a previous response has not yet been handled
                                    send this
                                    // response later
                             }
                             else
                             {
                                    // clear explicit response flag (semaphore access
                                    may be
```

```
                                        // required)
                                        ExplicitResponse = NOT_READY;
                                }
                        }
                }
        }

    DNS_StopScan( ... ) // stop scannning
    DNS_Offline( ... ) // go offline
    DNS_CloseCard( ... ) // close the card connection
    DNS_FreeDriver() // unload the hardware driver DLL
}
```

6
DeviceNet Scanner DLL API

Chapter Contents:

6.1 Introduction

This section defines the API *(Application Programming Interface)* for the DeviceNet Scanner DLL. All functions and data types are listed alphabetically.

Note

For Remote Link Devices (such as Remote Ethernet and USB), certain API calls can return before the command is actually processed on the remote server, while others require information from the remote server prior to returning. Therefore, the standard DnScan32 APIs are divided into two groups: “local”, and “remote”.

Remote: the API call will block (does not return) until the command executes and a response is received from the remote server.

Local: the API does not block waiting for a response from the remote server. Data is retrieved from a local buffer, or the parameters of the call can be validated locally.

The DnScan32 APIs for remote cards will be implemented as either “local” or “remote”, as defined in the following APIs.

6.2 API Reference

6.2.1 DNS_AddDevice

6.2.1.1 Description

Adds a device to the scan list. If the connection path offsets are non-zero, they must be initialized prior to executing this command.

6.2.1.2 Related Topics

See Sections 4.5.4, ADD_DEVICE Command, and 6.2.33, DNS_InitializePathBuffer.

6.2.1.3 Prototype

```
BOOL WINAPI DNS_AddDevice( DWORD CardHandle, DNS_DEVICE_CFG *DeviceCfg )
```

6.2.1.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceCfg	Device configuration block (see Section 3.4).

6.2.1.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.1.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0202h	Invalid MacId
2000 0205h	Duplicate device
2000 020dh	Invalid Offset
2000 020fh	Unknown Error
2000 021ah	Unsupported connection type
2000 021bh	Invalid connection flags
2000 0220h	Invalid explicit size
2000 0221h	Invalid strobe size
2000 0225h	Invalid path buffer

6.2.1.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.2 DNS_CAN_COUNTERS Data Type

6.2.2.1 Declaration

```
typedef struct
{
   WORD CanTx;              // CAN transmit counter
   WORD CanAck;             // CAN ack error counter
   WORD CanRx;              // CAN receive counter
   WORD CanError;           // CAN communication error Counter
   WORD CanLost;            // CAN lost messages counter
   WORD CanOverrun;         // CAN receive queue overrun counter
} DNS_CAN_COUNTERS;
```

6.2.3 DNS_CAN_MESSAGE Data Type

6.2.3.1 Description

Used to transmit and receive CAN packets via the CAN APIs provided by DNSCAN32.DLL.

6.2.3.2 Declaration

```
typedef struct
{
 WORD CANID;
 BYTE Length;
 BYTE Data[8];
}DNS_CAN_MESSAGE;
```

6.2.3.3 Related Topics

See Section 4.15.4, CANMessage Structure.

6.2.4 DNS_CANGetRxQStatus

6.2.4.1 Description

Retrieves the current CAN Rx Queue status. Calling this function will clear CAN Rx Queue status bytes A an B in shared memory.

6.2.4.2 Related Topics

See Section 4.13.1, Queue Status.

6.2.4.3 Prototype

```
BOOL WINAPI DNS_CANGetRxQStatus( DWORD CardHandle, BYTE *QueueStatus )
```

6.2.4.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
QueueStatus	The byte containing the CAN Rx Queue status

6.2.4.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.4.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 0229 h	Can Rx Queue not created

6.2.4.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.5 DNS_CANReceive

6.2.5.1 Description

Retrieves a message from the CAN Rx Queue.

6.2.5.2 Related Topics

See Section 6.2.3, DNS_CAN_MESSAGE Data Type.

6.2.5.3 Prototype

```
BOOL WINAPI DNS_CANReceive( DWORD CardHandle, DNS_CAN_MESSAGE *CANMsg )
```

6.2.5.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
CANMsg	Pointer to receive the CAN Message

6.2.5.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.5.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 0007h	Null pointer
2000 0229 h	CAN Rx Queue not created
2000 022a h	No Messages in the CAN Receive Queue

6.2.5.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.6 DNS_CANSetFilter

6.2.6.1 Description

Enables/disables the receipt of the specified CAN ID.

6.2.6.2 Related Topics

See Section 4.5.13, CAN_SETFILTER Command.

6.2.6.3 Prototype

```
BOOL WINAPI DNS_CANSetFilter( DWORD CardHandle, WORD CANID, BOOL Enable )
```

6.2.6.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
CANID	CAN ID to be enabled or disabled. CANID 0xFFFF with Enable = TRUE/FALSE enables/disables all CANIDs.
Enable	Boolean value to enable/ disable the CANID

6.2.6.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.6.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0228h	Invalid CAN ID
2000 020fh	Unknown Error

6.2.6.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.7 DNS_CANSetRXQ

6.2.7.1 Description

Specifies the CAN Rx Queue offset and size.

6.2.7.2 Related Topics

See Section 4.5.14, CAN_SETRXQ Command.

6.2.7.3 Prototype

```
BOOL WINAPI DNS_CANSetRXQ( DWORD CardHandle, WORD QueueOffset, WORD QueueSize )
```

6.2.7.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
QueueOffset	The offset in shared RAM where the CAN Rx Queue is to be created. Its value can be in the range 0x1000h – 0x3FEE h, inclusive. See Section 4.15.
QueueSize	Indicates the number of CAN message buffers within the queue. The minimum value is 1. The maximum value depends on the queue offset and available memory within the host interface.

6.2.7.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.7.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000020dh	Invalid Offset
2000 020fh	Unknown Error
20000227h	Invalid Queue Size

6.2.7.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.8 DNS_CANTransmit

6.2.8.1 Description

Transmits a single CAN Message.

6.2.8.2 Related Topics

See Sections 4.5.12, CAN_TRANSMIT Command, and 6.2.3, DNS_CAN_MESSAGE Data Type.

6.2.8.3 Prototype

```
BOOL WINAPI DNS_CANTransmit( DWORD CardHandle, DNS_CAN_MESSAGE *CANMsg )
```

6.2.8.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
CANMsg	Pointer that contains the transmit CAN packet

6.2.8.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.8.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0007 h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0207 h	Bus Offline
2000 020fh	Unknown Error
2000 0227 h	Invalid CAN Message length
2000 0228 h	Invalid CAN ID

6.2.8.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.9 DNS_ClearDeviceEvent

6.2.9.1 Description

Clears the specified device I/O event flags.

6.2.9.2 Prototype

```
BOOL WINAPI DNS_ClearDeviceEvent( DWORD CardHandle, WORD DeviceId, BYTE EventId, BYTE
EventMask )
```

6.2.9.3 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies device to be accessed
EventId	Indicates the event to be retrieved (see EventId table below)
EventMask	Client Status bit to clear

6.2.9.4 EventId

Value	Name	Description
0	Reserved	
1	DNS_IO1_EVENT	I/O area 1 events
2	DNS_IO2_EVENT	I/O area 2 events
3-0xff	Reserved	

Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.9.5 Errors

Value	Description
2000 0003_{hex}	Driver not loaded
2000 0102_{hex}	Invalid CardHandle
2000 $011a_{hex}$	Card Access Timeout
2000 0211_{hex}	Device not configured
2000 0212_{hex}	Invalid Event Id
2000 $021f_{hex}$	Invalid DeviceId
2000 0222_{hex}	Connection not configured

6.2.9.6 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

Note

For Remote Link Devices, this API always returns “True”.

6.2.10 DNS_ClearServerEvent

6.2.10.1 Description

Clears the specified server I/O event status flags.

6.2.10.2 Prototype

```
BOOL WINAPI DNS_ClearServerEvent( DWORD CardHandle, BYTE EventId, BYTE EventMask )
```

6.2.10.3 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
EventId	Indicates the event to be retrieved (see EventId table below)
EventMask	Server Status bit to clear

6.2.10.4 EventId

Value	Name	Description
0	Reserved	
1	DNS_IO1_EVENT	I/O area 1 events
2	DNS_IO2_EVENT	I/O area 2 events
5-0xff	Reserved	

6.2.10.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.10.6 Errors

Value	Description
2000 0003_{hex}	Driver not loaded
2000 0102_{hex}	Invalid CardHandle
2000 $011a_{hex}$	Card Access Timeout
2000 0212_{hex}	Invalid Event Id
2000 0213_{hex}	Server not configured
2000 0222_{hex}	Connection not configured

6.2.10.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

Note

For Remote Link Devices, this API always returns “True”.

6.2.11 DNS_CloseCard

6.2.11.1 Description

Closes a card connection. The specified CardHandle will be invalidated and the card will be shut down and disabled.

6.2.11.2 Prototype

```
BOOL WINAPI DNS_CloseCard( DWORD CardHandle )
```

6.2.11.3 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard

6.2.11.4 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.11.5 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout

6.2.11.6 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.12 DNS_DeleteDevice

6.2.12.1 Description

Removes a device from the scan list. If the device is actively being scanned on the network, all connections with the specified device are closed.

6.2.12.2 Related Topics

See Section 4.5.6, DELETE_DEVICE Command.

6.2.12.3 Prototype

```
BOOL WINAPI DNS_DeleteDevice( DWORD CardHandle, WORD DeviceId )
```

6.2.12.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies device to be removed from the scan list

6.2.12.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.12.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0206h	Device not found
2000 020fh	Unknown Error
2000 0211h	Device not configured
2000 021fh	Invalid DeviceId

6.2.12.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.13 DNS_DEVICE_CFG Data Type

6.2.13.1 Description

Contains all the configuration data necessary to add a device to the DeviceNet module scan list. The structure can uniquely identify a device and specify the I/O connection requirements and behaviors to be used when allocating and maintaining a connection with the device.

6.2.13.2 Declaration

```
typedef struct
{
   WORD MacId;                    // Device MacId
   WORD VendorId;                 // Device Vendor Id
   WORD DeviceType;               // Device Type
   WORD ProductCode;              // Device Product Code
        WORD ProductionInhibitTime;          // Production Inhibit Time for use with COS
        I/O
                              // connections[2]
   WORD Reserved1;            // Reserved for future use
   WORD Reserved2;            // Reserved for future use
   WORD Flags;                     // Connection flags
   WORD ExplicitSize;              // Device explicit buffer size
   WORD ExplicitOffset;            // Device explicit buffer offset
   WORD Io1Interval;               // I/O 1 connection interval
   WORD Output1Size;               // I/O 1 output buffer size
   WORD Output1Offset;             // I/O 1 output buffer offset
   WORD Output1LocalPathOffset;         // I/O 1 local output buffer path
                              // offset[1]
   WORD Output1RemotePathOffset;        // I/O 1 remote output buffer
                              // path offset[1]
   WORD Input1Size;                // I/O 1 input buffer size
   WORD Input1Offset;              // I/O 1 input buffer offset
   WORD Input1LocalPathOffset;          // I/O 1 local input buffer path
                              // offset[1]
   WORD Input1RemotePathOffset;         // I/O 1 remote input buffer path
                              // offset[1]
   WORD Io2Interval;               // I/O 2 connection interval
   WORD Output2Size;               // I/O 2 output buffer size
   WORD Output2Offset;             // I/O 2 output buffer offset
   WORD Output2LocalPathOffset;         // I/O 2 local output buffer path
                              // offset[1]
   WORD Output2RemotePathOffset;        // I/O 2 remote output buffer path
                              // offset[1]
   WORD Input2Size;                // I/O 2 input buffer size
   WORD Input2Offset;              // I/O 2 input buffer offset
   WORD Input2LocalPathOffset;          // I/O 2 local input buffer path
                              // offset[1]
   WORD Input2RemotePathOffset;         // I/O 2 remote input buffer path
                              // offset[1]
} DNS_DEVICE_CFG;
```

[1] Path offsets must point to an initialized path buffer (see Section 6.2.33).

[2] This parameter is valid with change-of-state I/O connections only.

6.2.14 DNS_DetectNIOSCard

6.2.14.1 Description

For backward compatibility, generation 4 or SST-DN4 interface cards will install and look like generation 3 or DN3 cards in the operating system. Use this function to detect whether a card is DN3 or DN4. This function is not for use with earlier platforms such as DNP or DN. Results in this case will be undefined.

6.2.14.2 Prototype

```
BOOL WINAPI DNS_DetectNIOSCard( TCHAR *pchName, DWORD *pdwNIOSType )
```

6.2.14.3 Arguments

Argument	Description
pchName	The name assigned to the SST-DN card. example “DN4-104-0002”
pdwNIOSType	Contains the card type upon return 0 = DN3, Non-zero = DN4

6.2.14.4 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.14.5 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null Pointer (pchName)
2000 0008h	Connection exists
2000 0009h	Resource unavailable
2000 000ah	Permission denied
2000 0100h	General failure
2000 0101h	Card handle not available
2000 0102h	Invalid card handle
2000 0109h	Card not found
2000 0110h	Invalid size (dwLength)
2000 011Ah	Card Access Timeout

6.2.14.6 Remote Link Device API Behavior

This is not supported by the Remote Ethernet product. An error of 20000003h or 20000009h is always returned.

6.2.15 DNS_Driver

6.2.15.1 Description

Retrieves hardware driver version information. The driver version is returned in both numeric and human-readable string format.

6.2.15.2 Prototype

```
BOOL WINAPI DNS_Driver( TCHAR *Buffer, WORD *Version, DWORD Size )
```

6.2.15.3 Arguments

Argument	Description
Buffer	Driver identification string buffer
Version	Minor revision (LSB) Major revision (MSB)
Size	Buffer size. The Identification string is truncated to fit.

6.2.15.4 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.15.5 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer

6.2.15.6 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.16 DNS_EnumDrivers

6.2.16.1 Description

Enumerates available driver configurations from the registry, allowing an application to determine the system's available SST-DN card configurations. This function can be called multiple times to determine all SST-DN cards configured on the system.

6.2.16.2 Prototype

```
BOOL WINAPI DNS_EnumDrivers( DWORD Index, char *lpName, DWORD *Len )
```

6.2.16.3 Arguments

Argument	Description
Index	Index number of the driver to enumerate
lpName	The driver name set by the function call
Len	Length of the lpName string

6.2.16.4 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.16.5 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0100h	General failure

6.2.16.6 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.17 DNS_FreeDriver

Unloads the hardware driver DLL. If no other card connections (CardHandles) are active, the Driver DLL is unloaded.

6.2.17.1 Prototype

```
BOOL WINAPI DNS_FreeDriver( void )
```

6.2.17.2 Arguments

This function has no arguments.

6.2.17.3 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.17.4 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0008h	Connection exists

6.2.17.5 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.18 DNS_GetBusStatus

6.2.18.1 Description

Gets the CAN bus status word.

6.2.18.2 Related Topics

See Section 4.3.2, CAN Bus Status Word (0030h).

6.2.18.3 Prototype

```
BOOL WINAPI DNS_GetBusStatus( DWORD CardHandle, WORD *BusStatus )
```

6.2.18.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
BusStatus	Pointer to word to receive bus status flags

6.2.18.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.18.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout

6.2.18.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.19 DNS_GetCANCounters

6.2.19.1 Description

Gets the CAN bus counters.

6.2.19.2 Related Topics

See Sections 4.3, Application Module Header, and 6.2.2, DNS_CAN_COUNTERS Data Type.

6.2.19.3 Prototype

```
BOOL WINAPI DNS_GetCANCounters( DWORD CardHandle, DNS_CAN_COUNTERS *CANCounters )
```

6.2.19.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
CANCounters	Buffer to contain CAN counters (see Section 6.2.2).

6.2.19.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.19.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout

6.2.19.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction

6.2.20 DNS_GetCardStatus

6.2.20.1 Description

Retrieves the card status.

6.2.20.2 Prototype

```
BOOL WINAPI DNS_GetCardStatus( DWORD CardHandle, BOOL *CardOk )
```

6.2.20.3 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
CardOk	Contains card status upon return True = OK, False = Error

6.2.20.4 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.20.5 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout

6.2.20.6 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.21 DNS_GetClientEvent

6.2.21.1 Description

Retrieves the specified client status block event flags.

6.2.21.2 Related Topics

See Section 4.6, Client Status Block.

6.2.21.3 Prototype

```
BOOL WINAPI DNS_GetClientEvent( DWORD CardHandle, BYTE EventId, BYTE *ClientEvent )
```

6.2.21.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard.
EventId	Indicates the event to be retrieved (see the EventId table below)
ClientEvent	Buffer to contain client event flags (see Section 4.6.4).

6.2.21.5 EventId

Value	Name	Description
0	Reserved	
1	DNS_SCAN_EVENT	Scan events
2-0xff	Reserved	

6.2.21.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.21.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0212h	Invalid Event Id

6.2.21.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.22 DNS_GetClientStatus

6.2.22.1 Description

Gets the status information for the scanner's client function.

6.2.22.2 Related Topics

See Section 4.6, Client Status Block.

6.2.22.3 Prototype

```
BOOL WINAPI DNS_GetClientStatus( DWORD CardHandle, DNS_STATUS *ClientStatus )
```

6.2.22.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
ClientStatus	Buffer to contain client status information (see Section 4.11.2).

6.2.22.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.22.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout

6.2.22.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.23 DNS_GetDevice

6.2.23.1 Description

Reads a device configuration from the scan list.

6.2.23.2 Related Topics

See Section 4.5.5, GET_DEVICE Command.

6.2.23.3 Prototype

```
BOOL WINAPI DNS_GetDevice( DWORD CardHandle, WORD DeviceId, DNS_DEVICE_CFG *DeviceCfg )
```

6.2.23.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies device to be retrieved from the scan list
DeviceCfg	Device configuration block (see Section 3.4). Destination of device configuration data from scan list.

6.2.23.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.23.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0206h	Device not found
2000 020fh	Unknown Error
2000 0211h	Device not configured
2000 021fh	Invalid DeviceId

6.2.23.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.24 DNS_GetDeviceConnectionAllocations

6.2.24.1 Description

Reads the Connection Allocation Flags for a specific device.

6.2.24.2 Related Topics

See Section 4.11.6, Connection Allocation Flags.

6.2.24.3 Prototype

```
BOOL WINAPI DNS_GetDeviceConnectionAllocations( DWORD CardHandle, WORD DeviceId,
   WORD *AllocationFlags )
```

6.2.24.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies device to be accessed
AllocationFlags	Buffer to receive Connection Allocation Flags

6.2.24.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.24.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0211h	Device not configured
2000 021fh	Invalid DeviceId

6.2.24.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.25 DNS_GetDeviceEvent

6.2.25.1 Description

Retrieves the device event flags for the specified event area. See the *DeviceNet Scanner Module Reference Guide* for more information.

6.2.25.2 Related Topics

See Section 4.11, Device Status Table.

6.2.25.3 Prototype

```
BOOL WINAPI DNS_GetDeviceEvent( DWORD CardHandle, WORD DeviceId, BYTE EventId,
       BYTE *DeviceEvent )
```

6.2.25.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies device to be accessed
EventId	Indicates the event to be retrieved (see EventId table below)
DeviceEvent	Buffer to contain device event flags

6.2.25.5 EventId

Value	Name	Description
0	Reserved	
1	DNS_IO1_EVENT	I/O area 1 events
2	DNS_IO2_EVENT	I/O area 2 events
3	DNS_EXP_EVENT	Explicit events
4-0xff	Reserved	

6.2.25.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.25.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0211h	Device not configured
2000 0212h	Invalid Event Id
2000 021fh	Invalid DeviceId
2000 0222h	Connection not configured

6.2.25.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.26 DNS_GetDevicePath

6.2.26.1 Description

Retrieves the local or remote path for the specified device.

6.2.26.2 Related Topics

See Section 3.12, Connection Path Buffer.

6.2.26.3 Prototype

```
BOOL WINAPI DNS_GetDevicePath( DWORD CardHandle, WORD DeviceId, BYTE PathId,
       WORD *MaxPathLength, WORD *PathLength, BYTE *Buffer )
```

6.2.26.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard.
DeviceId	Identifies device to be accessed.
PathId	Indicates the path information to be retrieved (see PathId table below).
MaxPathLength	Contains the maximum path length upon return.
PathLength	Initially contains the size of Buffer. On return contains the path length on success, 0 if path was not configured, or the required size if Buffer is too small (error 2000 0214h).
Buffer	Buffer to contain path

6.2.26.5 PathId

Value	Name	Description
0	DNS_OUTPUT1_PATH	Local output 1 path
1	DNS_INPUT1_PATH	Local input 1 path
2	DNS_OUTPUT2_PATH	Local output 2 path
3	DNS_INPUT2_PATH	Local input 2 path
4	DNS_REMOTE_OUTPUT1_PATH	Remote output 1 path
5	DNS__REMOTE_INPUT1_PATH	Remote input 1 path
6	DNS_REMOTE_OUTPUT2_PATH	Remote output 2 path
7	DNS_REMOTE_INPUT2_PATH	Remote input 2 path
8-0xff	Reserved	

6.2.26.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.26.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0211h	Device not configured
2000 0214h	Invalid data buffer size
2000 021fh	Invalid DeviceId
2000 0223h	Invalid PathId

6.2.26.8 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.27 DNS_GetDeviceStatus

6.2.27.1 Description

Gets the status information for a specific device.

6.2.27.2 Related Topics

See section 4.11, Device Status Table.

6.2.27.3 Prototype

```
BOOL WINAPI DNS_GetDeviceStatus( DWORD CardHandle, WORD DeviceId, DNS_STATUS
*DeviceStatus )
```

6.2.27.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies the device to be accessed
DeviceStatus	Buffer to contain device status information. (See Section 6.2.58, DNS_STATUS Data Type).

6.2.27.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.27.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0211h	Device not configured
2000 021fh	Invalid DeviceId

6.2.27.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.28 DNS_GetModuleHeader

6.2.28.1 Description

Gets the module header information.

6.2.28.2 Related Topics

See Section 4.3, Application Module Header, and 6.2.38, DNS_MODULE_HEADER Data Type.

6.2.28.3 Prototype

```
BOOL WINAPI DNS_GetModuleHeader( DWORD CardHandle, DNS_MODULE_HEADER *ModuleHeader )
```

6.2.28.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
ModuleHeader	Buffer to contain Module Header information. (See Section 6.2.38, DNS_MODULE_HEADER Data Type).

6.2.28.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.28.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout

6.2.28.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.29 DNS_GetServerEvent

6.2.29.1 Description

Retrieves the specified server event flags.

6.2.29.2 Related Topics

See Section 4.8, Server Status Block.

6.2.29.3 Prototype

```
BOOL WINAPI DNS_GetServerEvent( DWORD CardHandle, BYTE EventId, BYTE *ServerEvent )
```

6.2.29.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
EventId	Indicates the event to be retrieved (see EventId table below).
ServerEvent	Buffer to contain server event flags

6.2.29.5 EventId

Value	Name	Description
0	Reserved	
1	DNS_IO1_EVENT	I/O area 1 events
2	DNS_IO2_EVENT	I/O area 2 events
3	DNS_EXP_REQ_EVENT	Explicit request events
4	DNS_EXP_RES_EVENT	Explicit response events
5-0xff	Reserved	

6.2.29.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.29.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0212h	Invalid Event Id
2000 0213h	Server not configured
2000 0222h	Connection not configured

6.2.29.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.30 DNS_GetServerPath

6.2.30.1 Description

Retrieves the specified server path. See the *DeviceNet Scanner Module Reference Guide* for more information.

6.2.30.2 Related Topics

See Section 3.12, Connection Path Buffer.

6.2.30.3 Prototype

```
BOOL WINAPI DNS_GetServerPath( DWORD CardHandle,BYTE PathId, WORD *MaxPathLength,
       WORD *PathLength, BYTE *Buffer )
```

6.2.30.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
PathId	Indicates the path information to be retrieved (see PathId table below).
MaxPathLength	Contains the maximum path length upon return
PathLength	Contains the path length on success, 0 if path was not configured, required size if Buffer is too small. Initially contains the size of Buffer
Buffer	Buffer to contain path

6.2.30.5 PathId

Value	Name	Description
0	DNS_OUTPUT1_PATH	Output 1 path
1	DNS_INPUT1_PATH	Input 1 path
2	DNS_OUTPUT2_PATH	Output 2 path
3	DNS_INPUT2_PATH	Input 2 path
4-0xff	Reserved	

6.2.30.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.30.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0213h	Server not configured
2000 0214h	Invalid data buffer size
2000 0223h	Invalid PathId

6.2.30.8 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.31 DNS_GetServerG2Status

6.2.31.1 Description

Retrieves the server Group 2 connection flag status information. This information provides an indication of the currently active server connections.

6.2.31.2 Related Topics

See Section 4.8, Server Status Block.

6.2.31.3 Prototype

```
BOOL WINAPI DNS_GetServerG2Status( DWORD CardHandle, BYTE *ServerG2Status )
```

6.2.31.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
ServerStatus	Buffer to contain server status.

6.2.31.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.31.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0213h	Server not configured

6.2.31.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.32 DNS_GetServerStatus

6.2.32.1 Description

Retrieves the server status information.

6.2.32.2 Related Topics

See Section 4.8, Server Status Block.

6.2.32.3 Prototype

```
BOOL WINAPI DNS_GetServerStatus( DWORD CardHandle, DNS_STATUS *ServerStatus )
```

6.2.32.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
ServerStatus	Buffer to contain server status (see Section 6.2.58, DNS_STATUS Data Type).

6.2.32.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.32.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0213h	Server not configured

6.2.32.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.33 DNS_InitializePathBuffer

6.2.33.1 Description

Initializes a path buffer for use with local and remote input and output paths (see Section 3.12, Connection Path Buffer). The path buffer must be initialized prior to the DN_ONLINE (see Section 4.5.2, DN_ONLINE Command) or ADD_DEVICE (see Section 4.5.4, ADD_ DEVICE Command) calls.

6.2.33.2 Related Topics

See Section 3.12, Connection Path Buffer.

6.2.33.3 Prototype

```
BOOL WINAPI DNS_InitializePathBuffer( DWORD CardHandle, WORD Offset, WORD MaxPathLength,
       WORD PathLength, BYTE *Path )
```

6.2.33.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
Offset	Offset at which to initialize path buffer
MaxPathLength	Maximum path length
PathLength	Length of current path
Path	Buffer containing path information

6.2.33.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.33.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 020dh	Invalid Offset
2000 020fh	Unknown Error
2000 0225h	Invalid path buffer

6.2.33.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.34 DNS_InvalidateHandles

6.2.34.1 Description

Invalidates all card handles. Any existing handles that were previously returned by DNS_OpenCard for the specified card name will become invalid.

This function can be used to close all connections that are being maintained by a remote link device, even those that may no longer be valid due to an abnormal application termination or a physical cable disconnection (which resulted in no DNS_CloseCard having been called to terminate the connection).

6.2.34.2 Prototype

```
BOOL WINAPI DNS_InvalidateHandles( TCHAR *CardName )
```

6.2.34.3 Arguments

Argument	Description
CardName	The name assigned to the SST-DN card.

6.2.34.4 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.34.5 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 011ah	Card Access Timeout

6.2.34.6 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.35 DNS_IoActive

6.2.35.1 Description

Sets the I/O state to active.

Each time this function is called, a timeout is started. If the timeout expires before the next DNS_IoActive call, the I/O state is set to idle. The next DNS_IoActive call sets the I/O state back to active. The reaction of devices to the I/O idle state (zero length output messages) is determined by the device manufacturer.

6.2.35.2 Related Topics

See Section 4.5.9, IO_ACTIVE Command.

6.2.35.3 Prototype

```
BOOL WINAPI DNS_IoActive( DWORD CardHandle, WORD Timeout )
```

6.2.35.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
Timeout	IoActive timeout delay in milliseconds (zero = no timeout).

6.2.35.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.35.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 020fh	Unknown Error

6.2.35.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.36 DNS_IoIdle

6.2.36.1 Description

Sets the I/O state to idle. In this state, all output messages are set to zero-length. The reaction of devices to the I/O idle state is determined by the device manufacturer.

6.2.36.2 Related Topics

See Section 4.5.10, IO_IDLE Command.

6.2.36.3 Prototype

```
BOOL WINAPI DNS_IoIdle( DWORD CardHandle )
```

6.2.36.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard

6.2.36.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.36.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 020fh	Unknown Error

6.2.36.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.37 DNS_LoadDriver

6.2.37.1 Description

Loads the hardware driver DLL (SSDN32.DLL).

6.2.37.2 Prototype

```
BOOL WINAPI DNS_LoadDriver( TCHAR *DriverName )
```

6.2.37.3 Arguments

Argument	Description
DriverName	Name and optional path to the hardware driver DLL i.e. “c:\windows\system\ssDN32.dll’

6.2.37.4 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.37.5 Error Codes

Value	Description
2000 0000h	Driver loaded
2000 0001h	Driver not found
2000 0002h	Invalid Driver
2000 0007h	Null pointer
2000 000Bh	Corrupted installation – re-install required

6.2.37.6 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.38 DNS_MODULE_HEADER Data Type

6.2.38.1 Declaration

```
typedef struct
{
   WORD ModuleType;     // Module type identifier
   WORD WinSize;        // Host interface window size
   WORD CardId;         // Reserved
   WORD KernelId;       // Kernel identifier
   WORD KernelRev; // Kernel revision
   WORD ModuleId;       // Module identifier
   WORD ModuleRev; // Module revision
   DWORD NetSerial;     // DeviceNet serial number
   BYTE CardType[16];   // Card type (i.e. ôSST-DN3ö)
   BYTE CardSerial[8];  // Card serial number
   WORD IrqControl;     // Card interrupt control word
   BYTE IrqStatusA;     // Card interrupt status byte A
   BYTE IrqStatusB;     // Card interrupt status byte B
   WORD MainCode;       // Main application error code
   WORD CanStatus; // CAN bus status word
   WORD CanTx;          // CAN transmit counter
   WORD CanAck;         // CAN ack error counter
   WORD CanRx;          // CAN receive counter
   WORD CanErr;         // CAN communication error counter
   WORD CanLost;        // CAN lost messages counter
   WORD CanOverrun;     // CAN receive queue overrun counter
   WORD AddCode;        // Additional application error code
   BYTE ModuleString[60];    // Module status string
WORD MajorTickInterval       // Number of milliseconds per Major Tick
WORD MinorTickCount          // Number of Minor Ticks per Major Tick
} DNS_MODULE_HEADER;
```

6.2.38.2 Related Topics

See Section 4.3, Application Module Header.

6.2.39 DNS_Offline

6.2.39.1 Description

Stops DeviceNet communications and goes offline. This function is not allowed while the scanner is active (see Section 6.2.59, DNS_StopScan).

6.2.39.2 Related Topics

See Section 4.5.3, DN_OFFLINE Command.

6.2.39.3 Prototype

```
BOOL WINAPI DNS_Offline( DWORD CardHandle )
```

6.2.39.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard

6.2.39.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.39.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0208h	Command not allowed while bus is active
2000 020fh	Unknown Error

6.2.39.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.40 DNS_Online

6.2.40.1 Description

Sets the DeviceNet communication and server parameters and goes online.

6.2.40.2 Related Topics

See Section 4.5.2, DN_ONLINE Command.

6.2.40.3 Prototype

```
BOOL WINAPI DNS_Online( DWORD CardHandle, DNS_SCANNER_CFG *ScannerCfg )
```

6.2.40.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
ScannerCfg	Scanner configuration data (see Section 6.2.52, DNS_SCANNER_CFG Data Type).

6.2.40.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.40.6 Error Codes

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0202h	Invalid Mac ID
2000 0203h	Invalid baud rate
2000 0204h	Duplicate Mac ID
2000 0209h	Bus is not offline
2000 020dh	Invalid Offset
2000 020eh	Bus fault encountered
2000 020fh	Unknown Error
2000 021bh	Invalid connection flags
2000 0220h	Invalid explicit size
2000 0221h	Invalid strobe size
2000 0225h	Invalid path buffer
2000 0226h	No CAN message acknowledgment

6.2.40.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.41 DNS_OpenCard

6.2.41.1 Description

Opens a card connection. The card handle returned by this function is used by the DLL client to identify the card to all other services.

6.2.41.2 Prototype

```
BOOL WINAPI DNS_OpenCard( DWORD *CardHandle, TCHAR *CardName, void *Module, DWORD Flags )
```

6.2.41.3 Arguments

Argument	Description
CardHandle	Upon successful completion of DNS_OpenCard, CardHandle is used for subsequent API calls.
CardName	The name assigned to the SST-DN card.
Module	Embedded binary application module pointer. (Load module from disk if NULL). {This argument is ignored for remote link devices.}
Flags	See Flags table below {This argument is ignored for remote link devices.}

6.2.41.4 Flags

Bit	Name	Description
0	SS_APPIRQ	Request use of physical interrupts
1	SS_OVERLAP	Enable overlapped mode {This flag has no effect when running on operating systems Windows 2000 and above.}
2-31	Reserved	

6.2.41.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.41.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0008h	Connection exists
2000 0009h	Resource unavailable
2000 000ah	Permission denied
2000 0100h	General failure
2000 0101h	Card handle not available
2000 0103h	Write Access denied
2000 0104h	Card Irq Access denied
2000 0105h	App Irq Access denied
2000 0106h	Hardware Control Access denied
2000 0107h	Overlap conflict
2000 0108h	Configuration not found (invalid CardName)
2000 0109h	Card not found
2000 010ah	Memory conflict
2000 010bh	Invalid / damaged application module
2000 010ch	Card not responding (application did not run)
2000 010dh	Card self-diagnostic failure
2000 0115h	Invalid interrupt level
2000 0116h	Invalid I/O address
2000 0117h	Invalid I/O length
2000 0118h	Invalid memory address
2000 0119h	Invalid memory length
2000 011ah	Card Access Timeout
2000 011bh	Driver access denied
2000 011ch	Memory test failure
2000 011dh	Module load request denied
2000 022dh	Remote Transport Link Error

6.2.41.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.42 DNS_ReadDeviceIo

6.2.42.1 Description

Reads I/O data from a device being scanned by the scanner's master function.

6.2.42.2 Related Topics

See Section 3.5.1, Client Connection Configuration.

6.2.42.3 Prototype

```
BOOL WINAPI DNS_ReadDeviceIo( DWORD CardHandle, WORD DeviceId, BYTE IoArea, void *Buffer,
       WORD Size )
```

6.2.42.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies the device to be read
IoArea	I/O area to be read (see IoArea table below).
Buffer	Buffer for data read
Size	Must contain the exact size, in bytes, of the I/O area to be read.

6.2.42.5 IoArea

Value	Name	Description
0	DNS_INPUT1	I/O input area 1
1	DNS_OUTPUT1	I/O output area 1
2	DNS_INPUT2	I/O input area 2
3	DNS_OUTPUT2	I/O output area 2
4-0xff	Reserved	

6.2.42.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.42.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 0110h	Invalid Size
2000 011ah	Card Access Timeout
2000 0210h	Invalid data area
2000 0211h	Device not configured
2000 0214h	Invalid data buffer size
2000 021fh	Invalid DeviceId
2000 0222h	Connection not configured
2000 0224h	I/O Interlock timeout

6.2.42.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.43 DNS_ReadServerIo

6.2.43.1 Description

Reads I/O data from the scanner's server function.

6.2.43.2 Related Topics

See Section 3.4.1, Server Connection Configuration.

6.2.43.3 Prototype

```
BOOL WINAPI DNS_ReadServerIo( DWORD CardHandle, BYTE IoArea, void *Buffer, WORD Size )
```

6.2.43.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
IoArea	I/O area to be read (see IoArea table below).
Buffer	Buffer for data read
Size	Must contain the exact size, in bytes, of the I/O area to be read.

6.2.43.5 IoArea

Value	Name	Description
0	DNS_INPUT1	I/O input area 1
1	DNS_OUTPUT1	I/O output area 1
2	DNS_INPUT2	I/O input area 2
3	DNS_OUTPUT2	I/O output area 2
4-0xff	Reserved	

6.2.43.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.43.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 0110h	Invalid Size
2000 011ah	Card Access Timeout
2000 0210h	Invalid data area
2000 0213h	Server not configured
2000 0214h	Invalid data buffer size
2000 0222h	Connection not configured
2000 0224h	I/O Interlock timeout

6.2.43.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.44 DNS_ReceiveDeviceExplicit

6.2.44.1 Description

Retrieves an explicit response message from a device being scanned by the scanner's master function. The explicit error event is cleared if an error is detected.

6.2.44.2 Related Topics

See Sections 6.2.53, DNS_SendDeviceExplicit, and 3.9.7, Client Explicit Messaging.

For a Client Explicit Messaging pseudo code example, see Section 5.4.3, Client Explicit Messaging.

6.2.44.3 Prototype

```
BOOL WINAPI DNS_ReceiveDeviceExplicit( DWORD CardHandle, WORD DeviceId, BYTE *Service,
       void *ServiceData, WORD *Size )
```

6.2.44.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies device to retrieve explicit message from
Service	Buffer to contain service code from the explicit response
ServiceData	Buffer to contain explicit response data from device
Size	Size of Message buffer (contains the number of service data bytes read upon return).

6.2.44.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code

6.2.44.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0211h	Device not configured
2000 0214h	Invalid data buffer size
2000 0215h	Connection does not exist
2000 0216h	Invalid Class
2000 0217h	Invalid Instance
2000 0218h	Explicit message not available
2000 021fh	Invalid DeviceId
2000 0222h	Connection not configured

6.2.44.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.45 DNS_ReceiveServerExplicit

6.2.45.1 Description

Retrieves an explicit request message for the scanner's server function. The explicit error event is cleared if an error is detected.

6.2.45.2 Related Topics

See Sections 6.2.54, DNS_SendServerExplicit, and 3.10.4, Server Explicit Messaging.

For a Server Explicit Messaging pseudo code example, see Section 5.4.4, Server Explicit Messaging.

6.2.45.3 Prototype

```
BOOL WINAPI DNS_ReceiveServerExplicit( DWORD CardHandle, WORD *ConnectionId, BYTE
*Service,
       WORD *ClassId, WORD *InstanceId, void *ServiceData, WORD *Size )
```

6.2.45.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
ConnectionId	The connection Id associated with the explicit request.
Service	Buffer to contain service being requested.
ClassId	Buffer to contain Object Class Id to which request is directed.
InstanceId	Buffer to contain the instance of the object to which the request is directed.
ServiceData	Buffer to contain explicit request data from client or master
Size	Size of Message buffer (contains the number of service data bytes read upon return).

6.2.45.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.45.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0213h	Server not configured
2000 0214h	Invalid data buffer size
2000 0215h	Connection does not exist
2000 0218h	Explicit message not available
2000 0222h	Connection not configured

6.2.45.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.46 DNS_RegisterBusStatusEvent

6.2.46.1 Description

Registers for bus status change event notification. Upon notification of a registered event, *wParam* contains the CAN bus status word.

6.2.46.2 Related Topics

See Sections 6.2.61, DNS_UnRegisterBusStatusEvent, and 4.3.2, CAN Bus Status Word (0030h)

6.2.46.3 Prototype

```
BOOL WINAPI DNS_RegisterBusStatusEvent( DWORD CardHandle, DWORD ThreadId, DWORD MsgId,
       LPARAM lParam )
```

6.2.46.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
ThreadId	Thread Id of thread to be notified of events
MsgId	Message Identifier
LParam	Thread-specific data (not modified by DNSCAN32.DLL).

6.2.46.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.46.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 021eh	Event already registered

6.2.46.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.47 DNS_RegisterCANRxEvent

6.2.47.1 Description

Registers for CAN Receive Rx event notification.

6.2.47.2 Related Topics

See Section 6.2.62, DNS_UnRegisterCANRxEvent.

6.2.47.3 Prototype

```
BOOL WINAPI DNS_RegisterCANRxEvent( DWORD CardHandle, DWORD ThreadId, DWORD MsgId,
                    LPARAM lParam )
```

6.2.47.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
ThreadId	Thread Id of thread to be notified of events
MsgId	Message Identifier
LParam	Thread-specific data (not modified by DNSCAN32.DLL).

6.2.47.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code

6.2.47.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0009h	Resource Unavailable – Remote Link Device only
2000 0102h	Invalid CardHandle
2000 021eh	Event already registered

6.2.47.7 Remote Link Device API Behavior

This is not supported. An error of 20000003h or 20000009h is always returned.

6.2.48 DNS_RegisterClientEvent

6.2.48.1 Description

Registers for client event notification. Upon notification of a registered event, *wParam* contains the client event/status flags.

6.2.48.2 Related Topics

See Sections 6.2.63, DNS_UnRegisterClientEvent, and 4.6, Client Status Block.

6.2.48.3 Prototype

```
BOOL WINAPI DNS_RegisterClientEvent( DWORD CardHandle, BYTE EventId, DWORD ThreadId,
        DWORD MsgId, LPARAM lParam )
```

6.2.48.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard.
EventId	Event identifier (see EventId table below).
ThreadId	Thread Id of thread to be notified of events.
MsgId	Message Identifier.
lParam	Thread-specific data (not modified by DNSCAN32.DLL).

6.2.48.5 EventId

Value	Name	Description
0	DNS_STATUS_EVENT	Status events
1	DNS_SCAN_EVENT	Scan events
2-0xff	Reserved	

6.2.48.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code

6.2.48.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 0212h	Invalid Event Id
2000 021eh	Event already registered

6.2.48.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.49 DNS_RegisterDeviceEvent

6.2.49.1 Description

Registers for device event notification. Upon notification of a registered event, *wParam* contains the device event/status flags.

6.2.49.2 Related Topics

See Sections 6.2.64, DNS_UnRegisterDeviceEvent, and 4.11, Device Status Table.

6.2.49.3 Prototype

```
BOOL WINAPI DNS_RegisterDeviceEvent( DWORD CardHandle, WORD DeviceId, BYTE EventId,
       DWORD ThreadId, DWORD MsgId, LPARAM lParam )
```

6.2.49.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies device to be registered
EventId	Event identifier (see EventId table below).
ThreadId	Thread Id of thread to be notified of events
MsgId	Message Identifier
lParam	Thread-specific data (not modified by DNSCAN32.DLL).

6.2.49.5 EventId

Value	Name	Description
0	DNS_STATUS_EVENT	Status events
1	DNS_IO1_EVENT	I/O area 1 events
2	DNS_IO2_EVENT	I/O area 2 events
3	DNS_EXP_EVENT	Explicit events
4-0xff	Reserved	

6.2.49.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.49.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 0212h	Invalid Event Id
2000 021fh	Invalid DeviceId
2000 021eh	Event already registered

6.2.49.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.50 DNS_RegisterServerEvent

6.2.50.1 Description

Registers for server event notification. Upon notification of a registered event, *wParam* contains the event/status flags.

6.2.50.2 Related Topics

See Sections 6.2.65, DNS_UnRegisterServerEvent, and 4.8, Server Status Block.

6.2.50.3 Prototype

```
BOOL WINAPI DNS_RegisterServerEvent( DWORD CardHandle, BYTE EventId, DWORD ThreadId,
       DWORD MsgId, LPARAM lParam )
```

6.2.50.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
EventId	Event identifier (see EventId table below)
ThreadId	Thread Id of thread to be notified of events
MsgId	Message Identifier
lParam	Thread-specific data (not modified by DNSCAN32.DLL).

6.2.50.5 EventId

Value	Name	Description
0	DNS_STATUS_EVENT	Status events
1	DNS_IO1_EVENT	I/O area 1 events
2	DNS_IO2_EVENT	I/O area 2 events
3	DNS_EXP_REQ_EVENT	Explicit request events
4	DNS_EXP_RES_EVENT	Explicit response events
5	DNS_G2FLAGS_STATUS_EVENT	G2 Connection Flags Status Event
5-0xff	Reserved	

6.2.50.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.50.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 0212h	Invalid Event Id
2000 021eh	Event already registered

6.2.50.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.51 DNS_ScanActive

6.2.51.1 Description

Sets the scan heartbeat to the value specified by timeout.

Each time this function is called, a timeout is started. If the timeout expires before the next DNS_ScanActive call, the scanner module will begin a controlled shutdown of all connections and stop scanning the network.

6.2.51.2 Related Topics

See Section 4.5.11, SCAN_ACTIVE Command.

6.2.51.3 Prototype

```
BOOL WINAPI DNS_ScanActive(DWORD CardHandle, WORD Timeout)
```

6.2.51.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
Timeout	The scan timeout interval

6.2.51.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.51.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0207h	Bus Offline
2000 020bh	Scanner is not running
2000 020ch	Scanner is stopping
2000 020fh	Unknown Error

6.2.51.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.52 DNS_SCANNER_CFG Data Type

6.2.52.1 Description

Contains all the configuration data necessary to go online and define the supported connection behaviors and characteristics for the scanner module's server/slave functionality.

6.2.52.2 Declaration

```
typedef struct
{
   WORD MacId;                    // Client/Server MacId
   WORD BaudRate;                 // Client/Server baud rate
   WORD ScanInterval;             // Global Scan interval
   WORD ReconnectTime;       // Reconnection Timeout Interval
   WORD Flags;                    // Connection flags
   WORD ExplicitRequestSize;      // Server explicit request buffer size
   WORD ExplicitRequestOffset;          // Server explicit request buffer
                             // offset
   WORD ExplicitResponseSize;     // Server explicit response buffer
                             // size
   WORD ExplicitResponseOffset;         // Server explicit response buffer
                             // offset
   WORD Io1Interval;              // I/O 1 connection interval
   WORD Output1Size;              // I/O 1 output buffer size
   WORD Output1Offset;            // I/O 1 output buffer offset
   WORD Output1LocalPathOffset;         // I/O 1 local output buffer path
                             // offset[1]
   WORD Input1Size;               // I/O 1 input buffer size
   WORD Input1Offset;             // I/O 1 input buffer offset
   WORD Input1LocalPathOffset;          // I/O 1 local input buffer path
                             // offset[1]
   WORD Io2Interval;              // I/O 2 connection interval
   WORD Output2Size;              // I/O 2 output buffer size
   WORD Output2Offset;            // I/O 2 output buffer offset
   WORD Output2LocalPathOffset;         // I/O 2 local output buffer path
                             // offset[1]
   WORD Input2Size;               // I/O 2 input buffer size
   WORD Input2Offset;             // I/O 2 input buffer offset
   WORD Input2LocalPathOffset;          // I/O 2 local input buffer path
                             // offset[1]
} DNS_SCANNER_CFG;
```

[1] Path offsets must point to an initialized path buffer (see Section 6.2.33, DNS_InitializePathBuffer).

6.2.52.3 Related Topics

See Section 3.4, Server Configuration.

6.2.53 DNS_SendDeviceExplicit

6.2.53.1 Description

Sends an explicit request message to a device in the scan list.

6.2.53.2 Related Topics

See Sections 6.2.44, DNS_ReceiveDeviceExplicit, and 3.9.7, Client Explicit Messaging.

For a Client Explicit Messaging pseudo code example, see Section 5.4.3, Client Explicit Messaging.

6.2.53.3 Prototype

```
BOOL WINAPI DNS_SendDeviceExplicit( DWORD CardHandle, WORD DeviceId, BYTE Service, WORD
       ClassId, WORD InstanceId, void *ServiceData, WORD Size )
```

6.2.53.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies device to retrieve explicit message from.
Service	Service being requested
ClassId	Object Class Id request is directed to
InstanceId	Instance of the object request is directed to
ServiceData	Buffer containing request specific service data
Size	Size of ServiceData buffer

6.2.53.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.53.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0211h	Device not configured
2000 0214h	Invalid data buffer size
2000 0219h	Explicit message pending
2000 021fh	Invalid DeviceId
2000 0222h	Connection not configured
2000022Dh	Remote Transport Link Error

6.2.53.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.54 DNS_SendServerExplicit

6.2.54.1 Description

Sends an explicit response message from the scanner's server function.

6.2.54.2 Related Topics

See Sections 6.2.45, DNS_ReceiveServerExplicit, and 3.10.4, Server Explicit Messaging.

For a Server Explicit Messaging pseudo code example, see Section 5.4.4, Server Explicit Messaging.

6.2.54.3 Prototype

```
BOOL WINAPI DNS_SendServerExplicit( DWORD CardHandle, WORD ConnectionId, BYTE Service,
        void *ServiceData, WORD Size )
```

6.2.54.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
ConnectionId	The connection Id associated with the explicit response
Service	Service code
ServiceData	Buffer containing request-specific service data
Size	Size of ServiceData buffer

6.2.54.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.54.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0213h	Server not configured
2000 0214h	Invalid data buffer size
2000 0219h	Explicit message pending
2000 0222h	Connection not configured

6.2.54.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.55 DNS_SetAccessTimeout

Note

This API applies only to interface cards that share system memory (ISA and 104).

6.2.55.1 Description

Sets the card access timeout delay. Indicates the maximum access time for use with overlapped cards.

6.2.55.2 Prototype

```
BOOL WINAPI DNS_SetAccessTimeout( DWORD CardHandle, DWORD Timeout )
```

6.2.55.3 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
Timeout	The access timeout delay in milliseconds (default 1000 ms).

6.2.55.4 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.55.5 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle

6.2.55.6 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

Note

For Remote Link Devices, this API always returns TRUE.

6.2.56 DNS_SetEventNotificationInterval

6.2.56.1 Description

Sets the event notification interval for all registered events.

6.2.56.2 Prototype

```
BOOL WINAPI DNS_SetEventNotificationInterval( DWORD CardHandle, DWORD Interval )
```

6.2.56.3 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
Interval	Event notification interval in milliseconds (minimum 10ms).

6.2.56.4 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.56.5 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle

6.2.56.6 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.57 DNS_StartScan

6.2.57.1 Description

Starts the I/O scan and attempts to configure and connect to all devices currently in the scan list.

6.2.57.2 Related Topics

See Section 4.5.7, START_SCAN Command.

6.2.57.3 Prototype

```
BOOL WINAPI DNS_StartScan( DWORD CardHandle )
```

6.2.57.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard

6.2.57.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.57.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 0207h	Bus Offline
2000 020ah	Scanner is running
2000 020ch	Scanner is stopping
2000 020fh	Unknown Error

6.2.57.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.58 DNS_STATUS Data Type

6.2.58.1 Declaration

```
typedef struct
{
   BYTE StatusCode;
   BYTE StatusFlags;
} DNS_STATUS;
```

6.2.58.2 Related Topics

See Section 4.11, Device Status Table.

6.2.59 DNS_StopScan

6.2.59.1 Description

Stops the I/O scan.

6.2.59.2 Related Topics

See Section 4.5.8, STOP_SCAN Command.

6.2.59.3 Prototype

```
BOOL WINAPI DNS_StopScan( DWORD CardHandle )
```

6.2.59.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard

6.2.59.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.59.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0004h	Command timeout
2000 0102h	Invalid CardHandle
2000 011ah	Card Access Timeout
2000 020bh	Scanner is not running
2000 020ch	Scanner is stopping
2000 020fh	Unknown Error

6.2.59.7 Remote Link Device API Behavior

The behavior is remote. For more information, see Section 6.1, Introduction.

6.2.60 DNS_TriggerDevicePollIo

6.2.60.1 Description

Triggers immediate production for Poll I/O connections whose I/O Interval parameter is non-zero. This provides the application with greater control of device prioritization on the network.

6.2.60.2 Prototype

```
BOOL WINAPI DNS_TriggerDevicePollIo( DWORD CardHandle, WORD DeviceId )
```

6.2.60.3 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies the device

6.2.60.4 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.60.5 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 0211 h	Device Not Configured
2000 021f h	Invalid DeviceId
2000022Dh	Remote Transport Link Error

6.2.60.6 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.61 DNS_UnRegisterBusStatusEvent

6.2.61.1 Description

Cancels bus status change notification.

6.2.61.2 Related Topics

See Sections 6.2.46, DNS_RegisterBusStatusEvent, and 4.3.2, CAN Bus Status Word (0030h).

6.2.61.3 Prototype

```
BOOL WINAPI DNS_UnRegisterBusStatusEvent( DWORD CardHandle )
```

6.2.61.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard

6.2.61.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.61.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 021dh	Event not registered

6.2.61.7 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.62 DNS_UnRegisterCANRxEvent

6.2.62.1 Description

Cancels CAN Rx event notification.

6.2.62.2 Related Topics

See Section 6.2.47, DNS_RegisterCANRxEvent.

6.2.62.3 Prototype

```
BOOL WINAPI DNS_UnRegisterCANRxEvent( DWORD CardHandle )
```

6.2.62.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard

6.2.62.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.62.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0009h	Resource Unavailable – Remote Link Device only
2000 0102h	Invalid CardHandle
2000 021dh	Event not registered

6.2.62.7 Remote Link Device API Behavior

This is not supported. An error of 20000003h or 20000009h is always returned.

6.2.63 DNS_UnRegisterClientEvent

6.2.63.1 Description

Cancels notification of a previously registered client event.

6.2.63.2 Related Topics

See Sections 6.2.48, DNS_RegisterClientEvent, and 4.6, Client Status Block.

6.2.63.3 Prototype

```
BOOL WINAPI DNS_UnRegisterClientEvent( DWORD CardHandle, BYTE EventId )
```

6.2.63.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
EventId	Event identifier (see EventId table below).

6.2.63.5 EventId

Value	Name	Description
0	DNS_STATUS_EVENT	Status events
1	DNS_SCAN_EVENT	Scan events
2-0xff	Reserved	

6.2.63.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.63.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 0212h	Invalid Event Id
2000 021dh	Event not registered

6.2.63.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.64 DNS_UnRegisterDeviceEvent

6.2.64.1 Description

Cancels notification of a previously registered device event.

6.2.64.2 Related Topics

See Sections 6.2.49, DNS_RegisterDeviceEvent, and 4.11, Device Status Table.

6.2.64.3 Prototype

```
BOOL WINAPI DNS_UnRegisterDeviceEvent( DWORD CardHandle, WORD DeviceId, BYTE EventId )
```

6.2.64.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies device to be registered
EventId	Event identifier (see EventId table below).

6.2.64.5 EventId

Value	Name	Description
0	DNS_STATUS_EVENT	Status events
1	DNS_IO1_EVENT	I/O area 1 events
2	DNS_IO2_EVENT	I/O area 2 events
3	DNS_EXP_EVENT	Explicit events
4-0xff	Reserved	

6.2.64.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.64.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 0212h	Invalid Event Id
2000 021dh	Event not registered
2000 021fh	Invalid DeviceId

6.2.64.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.65 DNS_UnRegisterServerEvent

6.2.65.1 Description

Cancels notification of a previously registered server event.

6.2.65.2 Related Topics

See Sections 6.2.50, DNS_RegisterServerEvent, and 4.8, Server Status Block.

6.2.65.3 Prototype

```
BOOL WINAPI DNS_UnRegisterServerEvent( DWORD CardHandle, BYTE EventId )
```

6.2.65.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
EventId	Event identifier (see EventId table below).

6.2.65.5 EventId

Value	Name	Description
0	DNS_STATUS_EVENT	Status events
1	DNS_IO1_EVENT	I/O area 1 events
2	DNS_IO2_EVENT	I/O area 2 events
3	DNS_EXP_REQ_EVENT	Explicit request events
4	DNS_EXP_RES_EVENT	Explicit response events
5	DNS_G2FLAGS_STATUS_EVENT	G2 Connection Flags Status events
6-0xff	Reserved	

6.2.65.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.65.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0102h	Invalid CardHandle
2000 0212h	Invalid Event Id
2000 021dh	Event not registered

6.2.65.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.66 DNS_UpdateFirmware

6.2.66.1 Description

Updates an .ss4 file to card flash (DN4 interface cards only). This function must only be called when the card is closed with DNS_CloseCard(…) or has not yet been opened with DNS_OpenCard(…). To detect whether a card is DN4, use DNS_DetectNIOSCard(…).

6.2.66.2 Related Topics

See Section 6.2.14, DNS_DetectNIOSCard.

6.2.66.3 Prototype

```
BOOL WINAPI DNS_UpdateFirmware( BYTE *puchBuff, DWORD dwLength, TCHAR *pchName )
```

6.2.66.4 Arguments

Argument	Description
*puchBuff	Pointer to a memory area containing the .ss4 data to be loaded.
dwLength	Length of .ss4 data contained in memory pointed to by puchBuff
*pchName	Pointer to string which contains name of card, example “DN4-104-0002”

6.2.66.5 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.66.6 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null Pointer (puchBuff or pchName)
2000 0008h	Connection exists
2000 0009h	Resource unavailable
2000 000ah	Permission denied
2000 0100h	General failure
2000 0101h	Card handle not available
2000 0102h	Invalid card handle
2000 0109h	Card not found
2000 0110h	Invalid size (dwLength)
2000 011Ah	Card Access Timeout
2000 0232h	ss4 Load failure
2000 0233h	ss4 Revision already exists in flash
2000 0235h	ss4 Page checksum error
2000 0236h	ss4 Load size failure
2000 0237h	ss4 Load checksum failure
2000 023Ah	ss4 Invalid ss4 file

6.2.66.7 Remote Link Device API Behavior

This is not supported by the Remote Ethernet product. An error of 20000003h or 20000009h is always returned.

6.2.67 DNS_Version

6.2.67.1 Description

Retrieves DLL version information. The DLL version is returned in both numeric and human-readable string format.

6.2.67.2 Prototype

```
BOOL WINAPI DNS_Version( TCHAR *Buffer, WORD *Version, DWORD Size )
```

6.2.67.3 Arguments

Argument	Description
Buffer	DLL identification string buffer.
Version	Minor revision (LSB). Major revision (MSB).
Size	Buffer size. The Identification string is truncated to fit.

6.2.67.4 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.67.5 Errors

Value	Description
2000 0007h	Null pointer

6.2.67.6 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.68 DNS_VersionEx

6.2.68.1 Description

Retrieves version information for every component in the card name tree. For example, calling this function on a local card name would return the version information of DnScan32.dll, SSDn32.dll, and the hardware driver; calling it on a remote card name would return the version information of DnScan32.dll, RemoteProxy, the remote server firmware tasks, and the remote server hardware version.

The versions are returned in an array of VersionEx structures, beginning at *Buffer. If all the VersionEx structures fit into “Buffer”, the API will return TRUE, and *Size will be set to the total size (in bytes) of all VersionEx structures in the “Buffer” data. If the VersionEx structures will not all fit into “Buffer” (because *Size is not large enough), the API will return FALSE, and *Size will be set to the buffer size (in bytes) required for the API to succeed.

6.2.68.2 Related Topics

See Section 3.4.1, Server Connection Configuration.

6.2.68.3 Prototype

```
BOOL WINAPI DNS_VersionEx(CHAR* CardName, CHAR *Buffer, DWORD *Size, unsigned short
*VersionCount )
```

6.2.68.4 Arguments

Argument	Description
CardName	The name assigned to the SST-DN card
Buffer	Version information buffer
Size	Buffer Size
VersionCount	Number of VersionEx structures returned in “Buffer”

6.2.68.5 VersionEx Structure

The fields of the VersionEx data structure shall be defined as listed below:

NAME	Data Type	Description
ComponentName	Unsigned char[80]	ASCII human-readable NULL terminated string format
Version	Unsigned short	Upper byte ->Major revision. Lower byte -> Minor revision

6.2.68.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.68.7 Errors

Value	Description
2000 0007h	Null pointer
2000 0109h	Card not Found
2000 0110h	Invalid Size – The length specified by “DWORD *Size” was not large enough to hold all the requested version information

6.2.68.8 Remote Link Device API Behavior

The behavior is local if the card is already open; otherwise, the behavior is remote. For more information, see Section 6.1, Introduction.

6.2.69 DNS_WriteDeviceIo

6.2.69.1 Description

Writes I/O data to a specific device being scanned by the scanner's master function.

6.2.69.2 Related Topics

See Section 3.5.1, Client Connection Configuration.

6.2.69.3 Prototype

```
BOOL WINAPI DNS_WriteDeviceIo( DWORD CardHandle, WORD DeviceId, BYTE IoArea, void *Data,
       WORD Size )
```

6.2.69.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
DeviceId	Identifies the device to be written
IoArea	Data area to be written (see IoArea table below).
Data	Data buffer containing data to be written
Size	Must contain the exact size, in bytes, of the I/O area to be written. Contains the actual I/O size upon failure.

6.2.69.5 IoArea

Value	Name	Description
0	DNS_INPUT1[1]	I/O input area 1
1	DNS_OUTPUT1	I/O output area 1
2	DNS_INPUT2[1]	I/O input area 2
3	DNS_OUTPUT2	I/O output area 2
4-0xff	Reserved	

[1] Read only area

6.2.69.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.69.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 0110h	Invalid Size
2000 011ah	Card Access Timeout
2000 0210h	Invalid data area
2000 0211h	Device not configured
2000 0214h	Invalid data buffer size
2000 021fh	Invalid DeviceId
2000 0222h	Connection not configured
2000 0224h	I/O Interlock timeout
2000022Dh	Remote Transport Link Error

6.2.69.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

6.2.70 DNS_WriteServerIo

6.2.70.1 Description

Writes I/O data to the scanner's server function.

6.2.70.2 Related Topics

See Section 3.4.1, Server Connection Configuration.

6.2.70.3 Prototype

```
BOOL WINAPI DNS_WriteServerIo( DWORD CardHandle, BYTE IoArea, void *Data, WORD Size )
```

6.2.70.4 Arguments

Argument	Description
CardHandle	The handle returned by DNS_OpenCard
IoArea	Data area to be written (see IoArea table below).
Data	Data buffer containing data to be written.
Size	Must contain the exact size, in bytes, of the I/O area to be written. Contains the actual I/O size upon failure.

6.2.70.5 IoArea

Value	Name	Description
0	DNS_INPUT1	I/O input area 1
1	DNS_OUTPUT1[1]	I/O output area 1
2	DNS_INPUT2	I/O input area 2
3	DNS_OUTPUT2[1]	I/O output area 2
4-0xff	Reserved	

[1] Read only area

6.2.70.6 Returns

Value	Description
TRUE	Success
FALSE	Error. Use GetLastError to retrieve error code.

6.2.70.7 Errors

Value	Description
2000 0003h	Driver not loaded
2000 0007h	Null pointer
2000 0102h	Invalid CardHandle
2000 0110h	Invalid Size
2000 011ah	Card Access Timeout
2000 0210h	Invalid data area
2000 0213h	Server not configured
2000 0214h	Invalid data buffer size
2000 0222h	Connection not configured
2000 0224h	I/O Interlock timeout
2000022Dh	Remote Transport Link Error

6.2.70.8 Remote Link Device API Behavior

The behavior is local. For more information, see Section 6.1, Introduction.

A

Warranty and Support

Appendix Contents:

- Warranty
- Technical Support

A.1 Warranty

For warranty information, refer to http://www.mysst.com/warranty.asp.

A.2 Technical Support

Please ensure that you have the following information readily available before calling for technical support:

- Card type and serial number
- Computer's make, model, CPU speed and hardware configuration (other cards installed)
- Operating system type and version
- Details of the problem you are experiencing: firmware module type and version, target network and circumstances that may have caused the problem

A.2.1 Getting Help

Technical support is available during regular business hours by telephone, fax or email from any Woodhead Software & Electronics office, or from http://www.woodhead.com. Documentation and software updates are also available on the website.

Note

If you are using the SST-DN card with a third-party application, refer to the documentation for that package for information on configuring the software for the card.

North America

Canada:
Tel: +1-519-725-5136
Fax: +1-519-725-1515
Email: WoodheadSupportNA@molex.com

Europe

France:
Tel: +33 2 32 96 04 22
Fax: +33 2 32 96 04 21
Email: WoodheadIC.SupportEU@molex.com

Germany:
Tel: +49 7252 9496 555
Fax: +49 7252 9496 99
Email:mailto: WoodheadIC.SupportEU@molex.com

Italy:
Tel: +39 010 5954 052
Fax: +39 02 664 00334
Email: WoodheadIC.SupportIT@molex.com

Other countries:
Tel: +33 2 32 96 04 23
Fax: +33 2 32 96 04 21
Email: WoodheadIC.SupportEU@molex.com

Asia-Pacific

Japan:
Tel: +81 52 221 5950
Fax: +81 46 265 2429
Email: WoodheadIC.SupportAP@molex.com

Singapore:
Tel: +65 6268 6868
Fax: +65 6261 3588
Email: WoodheadIC.SupportAP@molex.com

China:
Tel: +86 21 5835 9885
Fax: +86 21 5835 9980
Email: WoodheadIC.SupportAP@molex.com

For the most current contact details, please visit http://www.woodhead.com.