Network Manager

June 11, 2024
00DA0209-001, Rev AH

Network Manager Interface

Introduction

This application note describes the implementation details of the Network Manager Interface (NMI). Application note #01, 00DA0109 provides an overview of the Network Manager (NM).

The Network Managers (NM) handle the alarm data management for Senstar's proprietary security networks. The Network Manager is available either as a Windows Application (kit # 00FG0200) or a Windows Service (kit # 00FG0220). There are three variants of the NM Application, one each for the Sentrax, Voice over Ethernet (VoE), and MX networks. The Network Manager Service handles the alarm data management for the Silver, FiberPatrol, CCC, Crossfire, Krypton, Sennet, and Starcom networks.

Note	This document refers to both the Network Manager Application and the Network Manager Service as the Network Manager unless describing a specific feature or function which pertains only to one of them.

The Network Managers function as data servers which collect and distribute alarm point data and control point status for third party Security Management Systems (SMS) via the Network Manager Interface (NMI) or generic text, or legacy Starcom protocol (contact Senstar Customer Service for Starcom protocol details). The third party organization is responsible for writing the software, which establishes communication to the Network Manager and implements the NMI.

Software developers have 2 choices when implementing the NMI:

- establish the TCP/IP communication and process raw NMI messages;

OR

- use an MFC DLL, which provides a higher level integration to the NMI TCP/IP messages.

Both methods are supported by the Network Manager software, to provide developers greater flexibility when interfacing to the NM products. The Network Manager Interface Software Development Kit (SDK) includes the files necessary for developing an interface. It also demonstrates the two methods through sample programs, written in C++ for Windows MFC framework. The programs serve as examples and test applications, and all source code is included.

If a developer is using raw NMI messages and redundant Network Managers, the application can connect to only the active NM. Therefore, when trying to connect to redundant Network Managers initially, or after losing the connection, you must hunt between the two specified IP addresses for the active NM.

Network Manager Simulators

There are two Network Manager Simulators, one for the Network Manager Application (NMSimul) and one for the Network Manager Service (NMS Simul). The Network Manager Simulator is used in place of a Network Manager to simulate the operation of a Network Manager with a connected network of security devices. Use a simulator to test system databases before installing and connecting the network devices. Developers should also use a simulator to test and verify the interfaces to third party display applications.

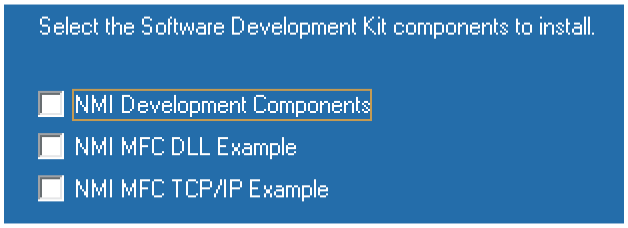
Note	Refer to the online help for information about using the Network Manager Simulator.

Network Manager SDK installation

Use the Network Manager installation program (Setup.exe) to install the NMI development components, and the sample programs that demonstrate its use.

During the initial installation:

On the Network Manager Installshield Wizard dialog, check the boxes beside the components that you want to install.



Select the Software Development Kit components to install.

- ☐ NMI Development Components
- ☐ NMI MFC DLL Example
- ☐ NMI MFC TCP/IP Example

Or to modify an existing installation:

Select **Modify**, and then select the appropriate components from the Network Manager Interface tree structure.

This selection creates a folder C:\Senstar\Network Manager\SDK, which contains the folders and files necessary for a developer to use the Network Manager Interface. Examples are included with the installation to demonstrate the NMI usage, and to test the Network Manager and connected equipment.

IP address configuration

To prevent unauthorized access, you must configure the Network Manager program with the IP addresses of all computers with which it will communicate. A Network Manager will not allow connections from any computer that is not on its registered address list.

Note	Consult the Network Manager’s online documentation for additional information.

Network Manager Interface TCP/IP

To write an interface that establishes its own TCP/IP connection, and then processes raw NMI messages, requires the NMTcpip.h header file. The header file includes message type definitions, Silver definitions, FiberPatrol definitions, CCC definitions, Crossfire definitions, Sennet definitions, Sentrax definitions, VoE definitions, MX definitions, Starcom definitions, and other sensor specific definitions. The NMTcpip.h file is included as part of the Network Manager Interface Development Components installation.

The connection is a client-server asynchronous relationship with the NM being the server. The client attaches to port 849 + the NM's Unit ID (1-10) at the NM Computer's IP address. If the client's IP address is in the Network Manager's list, and is not already in use, the connection is accepted.

Conventions:

- Bytes are in hexadecimal.
- Words and other multibyte types are stored least significant byte first.
- Bit 0 is the least significant bit in a byte or word.

All messages have a common format for the first 5 bytes:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	01	XX	XX

- Bytes 1 & 2 are constant values.
- Bytes 3 & 4 are the length of the message from the message type to the end of the data.
- Byte 5 is the message type.
- Byte 6 is the start of the data, if required.

NMI message summary

The client sends one of the following message types to the NM. The NM typically responds with the same message type. The NM can also send unsolicited messages. These are noted below.

Note See Recommended NMI Message Use Summary on page 59 for a list of commonly used messages.

NM_MT_NULL

This message has no data bytes, requires no response, and should be ignored by the recipient. The Windows O/S may be slow to report a TCP/IP connection loss unless a message is being sent. The NM periodically sends this message to check the status of a connection to an SMS client. An SMS client may use this message for the same purpose.

Message format:

Byte #	1	2	3	4	5
Data	E0	31	01	00	00

- Byte 5, the message type, is 00.

NM_MT_LOOPBACK

Two bytes of data are loaded by the client. The inverted values are returned. The client can use this to verify the TCP/IP connection as Windows can be slow to report TCP/IP link failures.

TIP	Use the NM_MT_LOOPBACK message as a client initiated heartbeat.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	01	AA	FF

- Byte 5, the message type, is 01.
- Bytes 6 & 7 are user defined and each may be any value between 00-FF.

Server response:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	01	55	00

- Bytes 6 & 7 are the inverted values of the user's input.

NM_MT_VERSION

This message is used to retrieve the software version of the Network Manager Service.

Client query:

Byte #	1	2	3	4	5
Data	E0	31	01	00	18

- Byte 5, the message type, is 24.

Server response:

Byte #	1	2	3	4	5	6	7	8	9
Data	E0	31	05	00	18	02	03	02	06

- Byte 5, the message type, is 24.
- Bytes 6, 7 & 8 are the 3 digits of the software version (X.YZ).
- Byte 9 is the software build number.

In this example, the software version is 2.32 and the software build number is 6.

NM_MT_DEVC_TYPE

This message is used to request the identity of a specific device, indicated by its address, or of all devices.

TIP	Use the NM_D_DEVC_TYPE structure available in the NMTcpip.h header to simplify parsing the response message.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	02	01	00

- Byte 5, the message type, is 02.
- Bytes 6 & 7 are the device address. This shows device 1.

Server response:

The server sends this message in response to the client query or unsolicited when a new device is detected.

Byte #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Data	E0	31	0B	00	02	01	00	01	01	03	00	00	01	00	01

- Byte 5, the message type, is 02.
- Bytes 6 & 7 are the device address. An address of 0 is the Network Controller.
- Bytes 8 & 9 are the device type.
- Bytes 10 & 11 are the total number of diagnostic alarms (3).
- Bytes 12 & 13 are the total number of alarms (256).
- Bytes 14 & 15 are the total number of control points (256).

If the device address was -1 (FF FF) in the query, each device's data would be appended to this message and the message length would increase by 10 bytes per device.

NM_MT_DB_VERSION

This message is used to retrieve the database version of a device. Database versions are currently supported for Silver based devices.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	01	00	1E	01	00

- Byte 5, the message type, is 30.
- Bytes 6 & 7 are the device address. This shows device 1.

Server response:

The server sends this message in response to the client query, or unsolicited, when a device database change is detected, and any point status changes have been retrieved from the device.

Note Database version may change and in-use statuses may not be impacted.

Byte #	1	2	3	4	5	6	7	8	9
Data	E0	31	01	00	1E	01	00	0A	01

- Byte 5, the message type, is 30.
- Bytes 6 & 7 are the device address. This shows device 1.
- Bytes 8 & 9 are the database version. This shows database version 266.

A database version of 0 indicates a device factory default database.

A database version of FFFF hex indicates that a database version is not applicable for the device (i.e. Database changes not reported by the device or detectable by NMS).

NM_MT_COMM_STAT

This message is used to retrieve the communications status for a device, or for all devices.

TIP Use the NM_D_COMM_STAT structure available in the NMTcpip.h header to simplify parsing the response message.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	03	01	00

- Byte 5, the message type, is 03.
- Bytes 6 & 7 are the device address.

Server response:

The server sends this message in response to the client query or unsolicited upon a change in communication status.

Byte #	1	2	3	4	5	6	7	8	9
Data	E0	31	05	00	03	01	00	01	02

- Byte 5, the message type, is 03.
- Bytes 6 & 7 are the device address.
- Byte 8 indicates if the device is connected (True 1, False 0).
- Byte 9 is the communication status (bit 0: Side A Fault, bit 1: Side B Fault).

If the device address was -1 (FF FF) in the query, each device's data would be appended to this message and the message length would increase by 4 bytes per device.

NM_MT_DIAG_ALARM

The server sends this message unsolicited when a diagnostic alarm changes state.

TIP Use the NM_D_DIAG_ALARM structure available in the NMTcpip.h header to simplify parsing the response message.

Server Response:

Byte #	1	2	3	4	5	6	7	8	9	10
Data	E0	31	06	00	04	03	00	01	00	01

- Byte 5, the message type, is 04.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the point number.
- Byte 10 is the new state for the diagnostic point. Active is 1 and Inactive is 0.

NM_MT_DIAG_ALARMS

This message is used to retrieve diagnostic alarm information from a device.

TIP Use the NM_D_DIAG_ALARMS structure available in the NMTcpip.h header to simplify parsing the response message.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	05	01	00

- Byte 5, the message type, is 05.
- Bytes 6 & 7 are the device address.

Server Response:

Byte #	1	2	3	4	5	6	7	8	9	10
Data	E0	31	06	00	05	01	00	08	00	02

- Byte 5, the message type, is 05.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the number of diagnostic alarms defined for the device.
- Bytes 10 to the number required; there are enough bytes to contain 1 status bit for each diagnostic alarm that has been defined.

NM_MT_SENSOR_ALARM

This message is used to retrieve the status of a sensor alarm.

TIP Use the NM_D_SENSOR_ALARM structure available in the NMTcpip.h header to simplify parsing the response message.

Client query:

Byte #	1	2	3	4	5	6	7	8	9
Data	E0	31	05	00	06	01	00	05	00

- Byte 5, the message type, is 06.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the sensor alarm number.

Server response/unsolicited message:

The server sends this message unsolicited when a sensor alarm changes state.

Byte #	1	2	3	4	5	6	7	8	9	10	11	12
Data	E0	31	08	00	06	01	00	05	01	01	00	00

- Byte 5, the message type, is 06.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the sensor alarm number.
- Byte 10 is the status - bit 0 is alarm and bit 1 is tamper.
- Bytes 11 & 12 are the number of alarm location structures that follow.

Location Structure

Bytes 13 to the number required are alarm location information. This applies only for sensors that provide target locating information, such as OmniTrax, FlexZone-20 / -60, the FiberPatrol Sensor Unit (SU) and the redundant FiberPatrol Sensor Unit (RSU). Location information is sent for each location that changed state and for all active locations.

For OmniTrax & FlexZone-20 / -60

TIP Use the NM_AG_OTRX_LOCN structure available in the NMTcpip.h header to simplify parsing the OmniTrax alarm location information.

TIP Use the NM_AG_FLXZN_LOCN structure available in the NMTcpip.h header to simplify parsing the FlexZone alarm location information.

Byte #	13	14
Data	0A	C0

- Bytes 13 & 14, Bits 0-13 are the distance from the processor along the cable, in meters.
- Bit 14 indicates the side, Side A is 0 and Side B is 1.
- Bit 15 indicates if the location is active (1) or has just turned inactive (0).

This example indicates an active location on Side B, 10 m from the processor.

For FiberPatrol SU & RSU

There are two levels of alarm location details provided depending on whether “Enable Alarm Classification” is checked on the NMS Front Panel’s SMS TCP/IP Configuration tab.

TIP Use the NM_FP_SU_LOCN or NM_FP_SU_LOCN2 structure available in the NMTcpip.h header to simplify parsing the FiberPatrol SU alarm location information.

Byte #	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Data	01	00	C0	07	44	91	45	35	42	A5	FE	97	C2	00	00	C8	42

- Byte 13, Bit 0 indicates if the location is active (1) or has just turned inactive (0). Bit 1 indicates if location is for Alarm (0) or Tamper (1) condition. For FiberPatrol, the Tamper condition is used to report that a Fiber Cut has occurred in the zone.

TIP Select (check) the checkbox on the NMS configuration to enable reporting of the cut location. This does not affect the tamper alarm reporting.

- Bytes 14-17 are a float value containing the distance of the target along the fiber. The units are meters or feet depending on the FiberPatrol configuration setting.
- Bytes 18-21 are a float value giving the latitude of the target.
- Bytes 22-25 are a float value giving the longitude of the target.
- Bytes 26-29 are a float value giving the altitude of the target.

Values for latitude, longitude and altitude may be 0 if the value is not available from FiberPatrol.

This example indicates an active location at 543 meters (or feet) at latitude 45.317936, longitude -75.997354, altitude 100.

The following additional bytes are provided if “Enable Alarm Classification” is checked:

Byte #	30	31	32	33	34	35	36
Data	1E	37	19	00	00	00	00

- Byte 30 is a numeric value for the Alarm Classification code. (Contact Senstar FiberPatrol Product Management for the current list of Alarm Classification codes.)
- Byte 31 is an unsigned value containing the magnitude of the alarm.
- Byte 32 is a signed value containing the speed and direction the target is moving. The units are kph or mph depending on the FiberPatrol configuration setting.
- Bytes 33-36 are reserved for future expansion.

The way that ranging sensors report alarms is configurable on the SMS TCP/IP configuration tab of the NMS Front Panel via the Enable Location Reporting and Enable Location Tracking checkboxes:

Example 1:

- **Enable Location Reporting** is not checked. **Enable Location Tracking** is not applicable. (i.e., report as non-ranging sensor).
  - No location information is included in the alarm reports.
  - An alarm report is sent at the initial alarm detection and when the alarm ends.

Example:

Device Addr.	Alarm #	Status	# Locations
5	2	0x1	0
5	2	0x0	0

Example 2:

- **Enable Location Reporting** is checked. **Enable Location Tracking** is not checked.
  - Location information is included in the initial alarm report.
  - An alarm report is sent at the initial alarm detection and when the alarm ends.

Example: Target crossing at 524 meters (**Alarm Hold** time defined).

Device Addr.	Alarm #	Status	# Locations	Location	
				Active	Position
5	2	0x1	1	1	524
5	2	0x0	0		

Example 3:

- **Enable Location Reporting** is checked. **Enable Location Tracking** is checked.
  - Location information is included in the alarm reports.
  - An alarm report is sent at the initial alarm detection, when the location status changes, and when the alarm ends.

Example: Targets Crossing at 524 and 550 meters (**Alarm Hold** time defined).

Device Addr.	Alarm #	Status	# Locations	Location		Location	
				Active	Postion	Active	Position
5	2	0x1	1	1	524		
5	2	0x1	2	1	524	1	550
5	2	0x1	2	1	550	0	524
5	2	0x1	1	0	550		
5	2	0x0	0				

NM_MT_SENSOR_ALARMS

The client uses this message to query a device for sensor alarms.

TIP	Use the NM_D_SENSOR_ALARMS structure available in the NMTcpip.h header to simplify parsing the response message.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	07	01	00

- Byte 5, the message type, is 07.
- Bytes 6 & 7 are the device address.

Server response:

Byte #	1	2	3	4	5	6	7	8	9	10	11
Data	E0	31	07	00	07	03	00	06	00	AA	01

- Byte 5, the message type, is 07.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the number of sensor alarm points.
- Byte 10 to the number required; enough bytes to contain 2 bits for each sensor alarm point; bit 0: alarm, bit 1: tamper.

In this example, there are 6 alarms, therefore, 12 bits are required. Sensor alarm points 1-4 are in tamper; sensor alarm point 5 is in alarm.

NM_MT_FILTER_ALARM

This message is used to retrieve the status of a sensor alarm filtered by the point's shunt status.

TIP	Use the NM_D_FILTER_ALARM structure available in the NMTcpip.h header to simplify parsing the response message.

Client query:

Byte #	1	2	3	4	5	6	7	8	9
Data	E0	31	05	00	10	01	00	05	00

- Byte 5, the message type, is 16.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the sensor alarm number.

Server response/unsolicited message:

The server sends this message unsolicited when a sensor alarm's filtered state changes.

Byte #	1	2	3	4	5	6	7	8	9	10	11	12
Data	E0	31	08	00	10	01	00	05	00	01	00	00

- Byte 5, the message type, is 16.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the sensor alarm number.
- Byte 10 is the status; bit 0: alarm and bit 1: tamper.
- Bytes 11 & 12 are the number of alarm location structures that follow (See NM_MT_SENSOR_ALARM for location structure description.)

This example indicates an alarm on sensor point 5 of device 1.

NM_MT_FILTER_ALARMS

The client uses this message to query a device for sensor alarms filtered by the shunt states.

TIP	Use the NM_D_FILTER_ALARMS structure available in the NMTcpip.h header to simplify parsing the response message.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	11	01	00

- Byte 5, the message type, is 17.
- Bytes 6 & 7 are the device address.

Server response/unsolicited message:

The server sends this message unsolicited when a sensor alarm's filtered state changes.

Byte #	1	2	3	4	5	6	7	8	9	10	11
Data	E0	31	07	00	11	01	00	06	00	55	02

- Byte 5, the message type, is 17.
- Bytes 6 & 7 are the device address.

- Bytes 8 & 9 are the number of sensor alarm points.
- Byte 10 to the number required; enough bytes to contain 2 bits each for each sensor alarm point; bit 0: alarm and bit 1: tamper.

In this example, there are 6 alarms, therefore, 12 bits are required. Sensor points 1 - 4 are in alarm; sensor point 5 is in tamper.

NM_MT_SHUNT

The client sends this message to change the shunt state for a sensor point's alarm and tamper states. Shunting blocks the reporting of the state in the NM_MT_FILTER_ALARM message.

TIP	Use the NM_D_SHUNT structure available in the NMTcpip.h header to simplify parsing the response message.

The server will respond with a NM_MT_FILTER_ALARM message if the shunt state is changed.

Client request:

Byte #	1	2	3	4	5	6	7	8	9	10	11
Data	E0	31	07	00	0E	01	00	05	00	01	00

- Byte 5, the message type, is 14.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the sensor alarm number.
- Byte 10 is a mask of which shunt bits to change; bit 0: alarm and bit 1: tamper.
- Byte 11 is the shunt states to set (1 = shunted); bit 0: alarm and bit 1: tamper.

This example clears shunting the alarm state of sensor point 5 on device 1.

NM_MT_SHUNTS

The client uses this message to query or set the shunt status of all the sensor points on a device.

TIP	Use the NM_D_SHUNTS structure available in the NMTcpip.h header to simplify parsing the response message.

The server sends this message in response to a query. The server will respond with a NM_MT_FILTER_ALARMS message if the shunt state of at least one point is changed.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	0F	01	00

- Byte 5, the message type, is 15.
- Bytes 6 & 7 are the device address.

Client request/Server response:

Byte #	1	2	3	4	5	6	7	8	9	10	11
Data	E0	31	07	00	0F	01	00	06	00	55	03

- Byte 5, the message type, is 15.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the number of sensor alarm points.
- Byte 10 to the number required; enough bytes to contain 2 bits each for each sensor point; bit 0: alarm and bit 1: tamper.

In this example, there are 6 sensor points, therefore, 12 bits are required. Sensor points 1 - 4 alarm condition is shunted; sensor point 5 alarm and tamper conditions are shunted.

NM_MT_PRE_ALARMS

The client sends this message to query a device for sensor pre-alarms. This message is supported only by the NMS, and only when enabled (see NMS Front Panel > Configure dialog > SMS tab > TCP/IP tab to enable). Pre-alarms are currently supported for Silver based FlexZone, FiberPatrol SU, and redundant FiberPatrol SU, zone alarms.

TIP	Use the NM_D_PRE_ALARMS structure available in the NMTcpip.h header to simplify parsing the response message.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	15	01	00

- Byte 5, the message type, is 21.
- Bytes 6 & 7 are the device address.

Server response/unsolicited message:

The server sends this message unsolicited when a sensor pre-alarm changes state.

Byte #	1	2	3	4	5	6	7	8	9	10
Data	E0	31	06	00	15	01	00	08	00	28

- Byte 5, the message type, is 21.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the number of sensor alarm points.
- Byte 10 to the number required; enough bytes to contain 1 bit for each sensor alarm point; a 1 in a bit indicates the corresponding point is in pre-alarm.

In this example, there are 8 alarms. Therefore, 8 bits (1 byte) are required. Sensor alarm points 4 & 6 are in pre-alarm.

NM_MT_CONTROL

The client sends this message to change the state of an output point.

TIP	Use the NM_D_CONTROL structure available in the NMTcpip.h header to simplify parsing the response message.

The server sends this message when a device reports a change of output state.

Client request:

Byte #	1	2	3	4	5	6	7	8	9	10
Data	E0	31	06	00	08	03	00	01	00	01

- Byte 5, the message type, is 08.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the point number.
- Byte 10 is the new state for the point. Active is 1; Inactive is 0.

Server response:

Byte #	1	2	3	4	5	6	7	8	9	10
Data	E0	31	06	00	08	03	00	01	00	01

The format of the server message is identical to the client message.

NM_MT_CONTROLS

The client uses this message to query or set the state of the output points of a device.

TIP	Use the NM_D_CONTROLS structure available in the NMTcpip.h header to simplify parsing the response message.

The server sends this message in response to a client query.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	09	03	00

- Byte 5, the message type, is 09.
- Bytes 6 & 7 are the device address.

Client request/Server Response:

Byte #	1	2	3	4	5	6	7	8	9	10	11
Data	E0	31	07	00	09	03	00	0C	00	03	04

- Byte 5, the message type, is 09.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the number of control points.
- Bytes 10 to the end are enough bytes to contain 1 status bit for each control point. The status points are packed starting with the least significant bit of the first byte.

In this example, there are 12 control points. The 2 bytes required to contain the data indicate that control points 1, 2 and 12 are active. The message length indicates the length from the message type to the end of the data bytes.

NM_MT_STANDBY

The client uses this message to force a redundant NM into standby mode. This message has no effect on a redundant NM when the alternate NM is not available to go online, or on a non-redundant NM.

Client request:

Byte #	1	2	3	4	5
Data	E0	31	01	00	0A

- Byte 5, the message type, is 10.

Server Response:

None, NM goes off-line, Client must make the connection to the alternate NM.

NM_MT_MATE_STAT

In a redundant configuration, the client uses this message to query the state of the mate NM (the standby NM).

Client request:

Byte #	1	2	3	4	5
Data	E0	31	01	00	0B

- Byte 5, the message type, is 11.

Server Response:

Byte #	1	2	3	4	5	6
Data	E0	31	02	00	0B	01

The server sends this message in response to the client query, or unsolicited upon a change in communication status.

- Byte 5, the message type, is 11.
- Byte 6 is the connection status - 0 = unconnected, 1 = connected.

NM_MT_DESC

This message is used to query the Network Manager’s description for a Device or Comm/Diagnostic/ Sensor/Control Point.

TIP Use the NM_D_DESC structure available in the NMTcpip.h header to simplify parsing the response message.

Client request:

Byte #	1	2	3	4	5	6	7	8	9	10
Data	E0	31	06	00	08	03	00	01	00	01

- Byte 5, the message type, is 18.
- Bytes 6 & 7 are the device address (n/a for Network descriptions).
- Bytes 8 & 9 are the point number (n/a for Device and Network descriptions).
- Byte 10 is the description type requested (Bits 0-3: 0-Device, 1-Comm Point, 2-Diagnostic Point, 3-Sensor Point, 4-Control Point, 5-Summary Point, 6-Network). Set bit 4 to retrieve a user-defined description if available. Currently applicable for Silver ALE, 16I/160, and UltraLink I/O Sensor and Control points, Silver Audio MUX Control points, FiberPatrol Sensor points, Krypton Device description.

Server response:

Byte #	1	2	3	4	5	6	7	8	9	10	11-n
Data	E0	31	06	00	08	03	00	01	00	01	01

- Byte 5, the message type, is 18.
- Bytes 6 & 7 are the device address (n/a for Network descriptions).
- Bytes 8 & 9 are the point number (n/a for Device and Network descriptions).
- Byte 10 is the description type requested (Bits 0-3: 0-Device, 1-Comm Point, 2-Diagnostic Point, 3-Sensor Point, 4-Control Point, 5-Summary Point, 6-Network). Bit 4 set if this is a user-defined description.
- Byte 11 to the number required is a null terminated string containing the description (string is in little endian Unicode).

NM_MT_DEVC_SMRY

This message is used to retrieve a summary of the communication and diagnostic status for a device.

TIP Use the NM_D_DEVC_SMRY structure available in the NMTcpip.h header to simplify parsing the response message.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	13	01	00

- Byte 5, the message type, is 19
- Bytes 6 & 7 are the device address

Server response:

The server sends this message in response to the client query or unsolicited upon a change in status.

Byte #	1	2	3	4	5	6	7	8
Data	E0	31	04	00	13	01	00	18

- Byte 5, the message type, is 19
- Bytes 6 & 7 are the device address
- Byte 8 is the device status summary
  - Bit 0: Comm. fail active or mismatch between configured and connected device type
  - Bit 1: 1 or more comm. side faults active
  - Bit 2: 1 or more critical diagnostic alarms active
  - Bit 3: 1 or more warning diagnostic alarms active
  - Bit 4: Enclosure tamper alarm active

NM_MT_SENSOR_TRBL

Client query:

The client uses this message to query a device for those sensor alarms whose alarm reporting capability might be compromised due to active Comm. fail, Device mismatch or Diagnostic alarms.

TIP	Use the NM_D_SENSOR_TRBL structure available in the NMTcpip.h header to simplify parsing the response message.

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	14	01	00

- Byte 5, the message type, is 20
- Bytes 6 & 7 are the device address

Server response:

The server sends this message in response to the client query or unsolicited upon a change in status.

Byte #	1	2	3	4	5	6	7	8	9	10	11
Data	E0	31	05	00	20	01	00	0A	00	18	00

- Byte 5, the message type, is 20
- Bytes 6 & 7 are the device address
- Byte 8 & 9 are the number of sensor alarm points
- Byte 10 to the number required; enough bytes to contain 1 bit for each sensor alarm point; A 1 in a bit indicates the corresponding point's alarm reporting capability may be compromised.

In this example there are 10 alarms, therefore 10 bits (2 bytes) are required. Sensor alarm points 4 & 5 are in trouble and as a result may not report alarms.

NM_MT_SENSOR_INUSE

The client uses this message to query a device for the optional sensor points available for reporting (in-use) and the alarm states supported for each point. Examples of optional points include points from ranging sensor zones that may be configured and points from optional auxiliary input cards.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	16	01	00

- Byte 5, the message type, is 22.
- Bytes 6 & 7 are the device address.

Server response:

Byte #	1	2	3	4	5	6	7	8	9	10	11	12
Data	E0	31	08	00	16	01	00	06	00	BB	99	19

- Byte 5, the message type, is 22.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the number of sensor alarm points.
- Byte 10 to the number required; enough bytes to contain 4 bits for each sensor alarm point; bit 0: alarm state supported, bit 1: tamper state supported, bit 2: pre-alarm state supported, bit 3: in-use.

In this example, there are 6 alarms; therefore, 24 bits are required. Sensor alarm points 1-2 are in-use and support alarm and tamper states; sensor alarm points 3-5 are in-use and support only the alarm state; point 6 is not in-use (if it was, it would support only the alarm state).

NM_MT_SMRY_INUSE

The client uses this message to query a device for the diagnostic summary points that are in-use.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	17	01	00

- Byte 5, the message type, is 23.
- Bytes 6 & 7 are the device address.

Server response:

Byte #	1	2	3	4	5	6	7	8
Data	E0	31	08	00	17	01	00	1F

- Byte 5, the message type, is 22.
- Bytes 6 & 7 are the device address.
- Byte 8 contains a bit for each diagnostic summary point in-use. Bit 0: Comm Fail/Mismatch, bit 1: Comm Side Fault, bit 2: Critical Diagnostic, bit 3: Warning Diagnostic, bit 4: Enclosure Tamper.

NM_MT_CONTROL_INUSE

The client uses this message to query a device for the optional control points available (in-use). Examples of optional points include controls on expansion cards.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	19	01	00

- Byte 5, the message type, is 25.
- Bytes 6 & 7 are the device address.

Server response:

Byte #	1	2	3	4	5	6	7	8	9	10	11
Data	E0	31	07	00	19	01	00	0E	00	FF	03

- Byte 5, the message type, is 25.
- Bytes 6 & 7 are the device address.
- Bytes 8 & 9 are the number of control points.
- Byte 10 to the number required; enough bytes to contain 1 bit for each control point indicating that the control point is in-use.

In this example, there are 14 controls; therefore, 2 bytes are required. Control points 1-10 are available for use; points 11-14 are not available.

NM_MT_LRANGE_ALARMS

The client uses this message to query a device for the linear range coverage of the device's alarm points.

TIP	Use the NM_D_LRANGE_ALARMS and NM_D_LRANGE_POINT structure available in the NMTcpip.h header to simplify parsing the response message.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	1A	01	00

- Byte 5, the message type, is 26
- Bytes 6 & 7 are the device address. This shows device 1.

Server response:

Byte #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Data	E0	31	03	00	1A	01	00	01	00	0B	00	00	4B	00	00	00	04	01	00	00

- Byte 5, the message type, is 26
- Bytes 6 & 7 are the device address. This shows device 1.
- Bytes 8 & 9 are the number of point range structures that follow. A value of -1 indicates a non-ranging sensor. A value of 0 indicates the NM has no range information for a ranging sensor. This may be because the NM has not yet retrieved the information or because the sensor does not have any ranging alarm points configured. This shows 1 range structures follow.

Range Structure:

Bytes 10 to the number required are alarm point range information structure. The structure consists of the alarm point number, start and end points of the linear distance the alarm represents, and if applicable, on which side of the device the range exists. An Alarm point's coverage may span the sides of a device in which case 2 range structures will be reported, one for each side. The start and end point locations are dimensionless in the message, they may be interpreted as meters, feet or heads depending on the sensor type.

Byte #	10	11	12	13	14	15	16	17	18	19	20
Data	0B	00	00	4B	00	00	00	04	01	00	00

- Bytes 10-11, the alarm point number. This shows 11.
- Byte 12, the side of the device the range is on. (0 = Side A, 1 = Side B, 0xFF = not applicable) This shows side A.

- Bytes 13-16, start of the range. This shows 75.
- Bytes 17-20, end of the range. This shows 260.

NM_MT_GPS_PATH_ALARMS

The client uses this message to query a device for the GPS Path covered by the device’s alarm points.

TIP	**TIP** Use the NM_D_GPS_PATH_ALARM and NM_D_GPS_COORD structures available in the NMTcpip.h header to simplify parsing the response message.

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	1D	01	00

- Byte 5, the message type, is 29.
- Bytes 6 & 7 are the device address. This shows device 1.

Server response:

The NMS will generate a separate message for each alarm point ith a defined GPS Path.

Byte #	1	2	3	4	5	6	7	8	9	10	11	
Data	E0	31	1F	00	1D	01	00	03	00	02	00	
Byte #	12	13	14	15	16	17	18	19	20	21	22	23
Data	48	F8	D1	C1	B9	B0	E1	41	00	00	00	00
Byte #	24	25	26	27	28	29	30	31	32	33	34	35
Data	48	F8	D1	C1	B9	B0	E1	41	00	00	00	00

- Byte 5, the message type, is 29.
- Bytes 6 & 7 are the device address. This shows device 1.
- Bytes 8 & 9 are the alarm point number. This shows alarm point 3.
- Bytes 10 & 11 are the number of path co-ordinate structures that follow. A value of -1 indicates a non- ranging sensor or a ranging sensor that doesn’t support GPS. A value of 0 indicates the NM has no GPS path information for a ranging sensor. This may be because the NM has not yet retrieved the information or because the sensor does not have any GPS information configured. When a path is available, at a minimum, there will be 2 structures representing the start and end of the path. This shows 2 co-ordinate structures follow.

Co-ordinate Structure:

Bytes 12 to the number required are alarm point path co-ordinate information structures. The structure consists of the latitude and longitude in decimal degrees and the altitude. The altitude is dimensionless in the message, it may be interpreted as meters or feet depending on the sensor type.

Byte #	12	13	14	15	16	17	18	19	20	21	22	23
Data	48	F8	D1	C1	B9	B0	E1	41	00	00	00	00

- Bytes 12-15 are a float value giving the latitude of the co-ordinate.
- Bytes 16-19 are a float value giving the longitude of the co-ordinate.
- Bytes 20-23 are a float value giving the altitude of the co-ordinate.

Values for altitude may be 0 if the sensor does not provide an altitude.

This example indicates a start point at latitude -26.246231, longitude 28.211290.

NM_MT_SET_CONFIG

The client sends this message to change to a temporary config for a sensor.

TIP	Use the NM_D_CONFIGSET structure available in the TCPIP.h header to simplify parsing the response message.

kooClient request:

Byte #	1	2	3	4	5	6	7	8	9	10	11
Data	E0	31	06	00	20	01	04	00	01	08	02

- Byte 5, the message type, is 0X20.
- Byte 6 the state is on.
- Bytes 7 & 8 the device address is 4.
- Byte 10 the hours value is 8.
- Byte 11 the zone value is 2

The server sends this message in response to a client query.

Server Response:

Byte #	1	2	3	4	5	6	7	8	9	10	11
Data	E0	31	06	00	20	01	04	00	01	08	02

VoE NM only

NM_MT_AUDIO_FOLDER

This message is used to set and retrieve the folder used to store audio recordings. (The folder pathname is specified in UNC syntax as a NULL terminated, little endian Unicode string.)

Client query:

Byte #	1	2	3	4	5	6	7
Data	E0	31	03	00	0C	01	00

- Byte 5, the message type is 12.
- Bytes 6 & 7 are the audio channel.

Client command/server response:

Byte #	1	2	3	4	5	6	7	8	9	10	11	12	13
Data	E0	31	1F	00	0C	01	00	5C	00	5C	00	53	00
Byte #	14	15	16	17	18	19	20	21	22	23	24	25	26
Data	52	00	56	00	5C	00	43	00	5C	00	41	00	75
Byte #	27	28	29	30	31	32	33	34	35				
Data	00	64	00	69	00	6F	00	00	00				

- Byte 5, the message type is 12.
- Bytes 6 & 7 are the audio channel.
- Bytes 8 to the number required are the folder pathname.

In this example, Audio channel 1 stores it’s audio recordings in \\SRV\C\Audio.

NM_MT_AUDIO_EVENT

The server sends this message unsolicited when an audio recording ends. All strings are NULL terminated, little endian Unicode strings.

TIP	Use the NM_D_AUDIO_EVENT structure available in the NMTcpip.h header to simplify parsing the response message.

Byte #	1	2	3	4	5	6	7	8	9	10	11	...
Data	E0	31	??	??	0D	01	00	??	??	??	??	...

- Byte 5, the message type, is 13.
- Bytes 6 & 7 are the audio channel.
- Bytes 8 & 9, length of recording file pathname in wide characters (WCHAR) including terminating NULL.
- Bytes 10 & 11, length of recording description in wide characters (WCHAR) including terminating NULL.
- Byte 12 to N, recording file pathname.
- Byte N+1 to number required, recording description.

The recording filename is based on the time the recording started and has the format "**Ch# C# N# start_time.wav**" for Intercom channels and
"**Ch# C# PA start_time.wav**" for Public Address channels.

The recording description has the format
"**Ch_desc, Intercom, Ctrl_Id, Target_Id,**" for Intercom channels and
"**Ch_desc, PA, Ctrl_Id, start_time-end_time**" for Public Address channels.

The **start_time** & **end_time** is specified in the format "**YYYYMMDD_hhmmss**".

Where:

Ch#	Identifies the channel (e.g., Ch1 is Channel 1)
C#	Identifies the Control Node (e.g., C4 is Control Node 4)
N#	Identifies the Target Node (e.g., N5 is Target Node 5)
Ch_desc	Label assigned to audio channel in VoNM configuration dialog
Ctrl_id	Identification label assigned to Control node on UCM IPCC Config tab
Target_id	Identification label assigned to Target node on UCM IPCC Config tab
YYYY	4 digit year
MM	2 digit month
DD	2 digit day
hh	2 digit hour in 24 hour format
mm	2 digit minute
ss	2 digit second

NMI DLL

The NMI DLL was written in Visual Studio C++ with MFC. To include the DLL in your software, four files are required:

- NMTcpip.h contains message type definitions, Silver definitions, CCC definitions, Crossfire definitions, Sennet definitions, Sentrax definitions, VoE definitions, MX definitions and other sensor specific definitions

- NMDll.h contains the function prototypes,
- NM.lib is required by the linker, and
- NM.dll is the dynamically linked library.

These files are included as part of the Network Manager Interface Development Components installation. An application can poll for TCP/IP messages from a Network Manager by periodically calling NM_GetMessage (recommended once every 10 ms). Alternately, the application can specify a message (nRxMsg parameter of NM_OpenNetwork) to post to a window (handle specified in NM_Set_HWND) when a TCP/IP message is received from a Network Manager. The corresponding message handler function in the application then calls NM_GetMessage when a message is available to be processed.

The following are the descriptions of the exported functions:

NM_Set_HWND - This function designates the Window that will receive messages. Use when one window handles all connected networks. Otherwise, see optional parameter in NM_OpenNetwork.

```
void NM_Set_HWND( HWND hWnd);
hWnd - handle to Window that will receive messages
```

NM_StatusWindow - This function is used to open and close the Status Information window.

```
void NM_StatusWindow( BOOL bShow );
bShow - true = Open, false = Close
```

NM_OpenNetwork - NM_OpenNetwork opens the connection to the Network Manager.

Returned:

HANDLE	- Handle to Network Manager

Notes:
Use inet_ntoa to convert in_addr to dotted address ascii string
Use inet_addr to convert dotted address ascii string to in_addr

```
HANDLE NM_OpenNetwork( in_addr addrNM1, in_addr addrNM2, UINT nUnitId,
UINT nRxMsg = WM_APP, HWND hWnd = NULL );
```

addrNM1	- IP address of primary network manager
addrNM2	- IP address of alternate network manager
nUnitID	- Unit Id of network manager
nRxMsg	- Message number to use to signal message received for processing
hWnd	- Handle to Window that will receive messages (optional, use if separate Windows used for each connected network)

NM_OpenNetworkTLS - NM_OpenNetworkTLS opens the connection to the Network Manager.

Returned:

HANDLE	- Handle to Network Manager

Notes:
Use inet_ntoa to convert in_addr to dotted address ascii string
Use inet_addr to convert dotted address ascii string to in_addr

There are 2 files required for TLS encryption, a Certificate file and a Key file. The NMS ships with default versions of both of these files located in a “Certificates” folder in the Network Manager Installation directory (default file-path C:\Senstar\Network Manager\Certificates). The Certificate file is named MyCertificate.pem and the Key file is named MyKey.pem. These can be replaced with site specific Certificate and Key files as long as they have the same names.

HANDLE NM_OpenNetworkTLS(in_addr addrNM1, in_addr addrNM2, UINT nUnitId,
UINT nRxMsg = WM_APP, HWND hWnd = NULL);

addrNM1	- IP address of primary network manager
addrNM2	- IP address of alternate network manager
nUnitID	- Unit Id of network manager
nRxMsg	- Message number to use to signal message received for processing
hWnd	- Handle to Window that will receive messages (optional, use if separate Windows used for each connected network)
bTLS1	- Boolean to enable TLS on the primary NMS connection
bTLS2	- Boolean to enable TLS on the secondary NMS connection

NM_CloseNetwork - This function closes the connection to the Network Manager.

void NM_CloseNetwork(HANDLE hNetManager);

hNetManager	- Handle to Network Manager returned by OpenNetwork

NM_SetStandby - This function sets the active Network Manager to standby, thereby transferring control to the alternate Network Manager and making it active. This applies only to redundant NM configurations.
void NM_SetStandby(HANDLE hNetManager)

hNetManager	- Handle to Network Manager returned by OpenNetwork

NM_DeviceType - Get the Device type for a specified device.

Returned:

int	- Device Type (-1 = No device, 0 = Network dependent device type)

int NM_DeviceType(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number

NM_LinkStat - Get the specified Network Manager’s Link State.
Returned:

bool	- Status of link to Network Manager (true = connected)

bool NM_LinkStat(HANDLE hNetManager);

hNetManager	- Handle to Network Manager returned by OpenNetwork

NM_GetMessage - Get the next message from the specified Network Manager.

Returned:

bool	- true = Success
	- false = No message available

bool NM_GetMessage(HANDLE hNetManager, int& nSrcAddr, int& nMsgType);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nSrcAddr	- Storage for source device # from current message
nMsgType	- Storage for Message type from current message (see NMTcpip.h for message type definitions)

NM_RqstMate - Send a Mate NM Connection Status request message to a Network Manager.

void NM_RqstMate(HANDLE hNetManager)

hNetManager	- Handle to Network Manager returned by OpenNetwork

NM_MateStat - Get the Mate NM Connection Status from the current NM_MT_MATE_STAT message.
Returned:

bool	- false = Unconnected
	- true = Connected

bool NM_MateStat(HANDLE hNetManager)

hNetManager	- Handle to Network Manager returned by OpenNetwork

NM_RqstDevcSmry - Sends the Device Summary request message to a Network Manager

void NM_RqstDevcSmry(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device Number

NM_GetDevcSmry - Get the Device Summary from the current NM_MT_DEVC_SMRY message.

bool NM_GetDevcSmry(HANDLE hNetManager, bool& bCommFail, bool& bCommFault, bool& bDiagCrit, bool& bDiagWarn, bool& bEnclTamp);

hNetManager	- Handle to Network Manager returned by OpenNetwork
bCommFail	- storage for comm. fail summary status
bCommFault	- storage for comm. fault summary status
bDiagCrit	- storage for critical diagnostic alarm summary status
bDiagWarn	- storage for warning diagnostic alarm summary status
bEnclTamper	- storage for enclosure tamper alarm status

NM_RqstComm - Send a Communication Status request message to a Network Manager.

void NM_RqstComm(HANDLE hNetManager);

hNetManager	- Handle to Network Manager returned by OpenNetwork

NM_GetComm - Get a device's communication status from the current NM_MT_COMM_STAT message.

Returned:

bool	- true = Success
	- false = Failure, invalid for current message type, or device #

bool NM_GetComm(HANDLE hNetManager, int nDevice, bool& bConnected, UINT* uSideFault = NULL);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number
bConnected	- Storage for Comm status (true = connected)
uSide	- Pointer to Storage for side fault status (optional) bit 0 = X side, bit 1 = Y side (true = fault)

NM_RqstDiagAlrm - Send the Diagnostic Alarm request message to a Network Manager.

void NM_RqstDiagAlrm(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number

NM_RqstSensAlrm - Send Sensor Alarm request for a single alarm point on a device.

void NM_RqstSensAlrm(HANDLE hNetManager, int nDevice, int nPoint);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number
nPoint	- Alarm point number

NM_RqstSensAlrm - Send the Sensor Alarm request message for all the alarm points on a device.

void NM_RqstSensAlrm(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number

NM_RqstFiltAlarm - Send Filtered Sensor Alarm request for a single alarm point on a device.

void NM_RqstSensAlrm(HANDLE hNetManager, int nDevice, int nPoint);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number
nPoint	- Alarm point number

NM_RqstFiltAlarm - Send Filtered Sensor Alarm request for all the alarm points on a device.

void NM_RqstSensAlrm(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number

NM_RqstPreAlarm - Send Sensor Pre-Alarm request for all the alarm points on a device.

void NM_RqstPreAlrm(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number

NM_RqstSensTrbl - Sends the Sensor Trouble request message to a Network Manager

void NM_RqstSensTrbl(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device Number

NM_GetAlarm - Get the point status from the Current NM_MT_DIAG_ALARM(S), NM_MT_SENSOR_ALARM(S), NM_MT_PRE_ALARMS, or NM_MT_SENSOR_TRBL message.

Returned:

bool	- true = Success
	- false = Failure, invalid for current message type, device type or point #

bool NM_GetAlarm(HANDLE hNetManager, int& nPoint, UINT& uStatus);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nPoint	- Storage for point number
	- filled by function for NM_MT_SENSOR_ALARM messages
	- specified for NM_MT_SENSOR_ALARMS messages
uStatus	- Storage for point status
	NM_MT_DIAG_ALARM(S): 0 = secure, 1 = alarm
	NM_MT_SENSOR_ALARM(S): 0 = secure, 1 = alarm, 2 = tamper, 3 = alarm & tamper
	NM_MT_PRE_ALARMS: 0 = secure, 1 = pre-alarm
	NM_MT_SENSOR_TRBL: 0 = secure, 1 = trouble

NM_GetFiltAlarm - Get the point status from the current NM_MT_FILTER_ALARM or NM_MT_FILTER_ALARMS message.

Returned:

bool	- true = Success
	- false = Failure, invalid for current message type, device type or point #

bool NM_GetSensAlarm(HANDLE hNetManager, int& nPoint, UINT& uStatus);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nPoint	- Storage for point number
	- filled by function for NM_MT_FILTER_ALARM messages
	- specified for NM_MT_FILTER_ALARMS messages
uStatus	- Storage for point status (0 = secure, 1 = alarm, 2 = tamper, 3 = alarm & tamper)

NM_GetLocn - Get the alarm location from the current NM_MT_SENSOR_ALARM or NM_MT_FILTER_ALARM message. Returned: bool - true = Success (always fails for non-ranging sensors) - false = Failure, invalid for current message type, device type or point # bool NM_GetLocn(HANDLE hNetManager, int& nPoint, int& nLocn, HGLOBAL &hLocn, int nSize); hNetManager - Handle to Network Manager returned by OpenNetwork nPoint - Storage for point number - filled by function for NM_MT_..._ALARM messages nLocn - Storage for number of alarm locations hLocn - Storage for handle to alarm locations (delete after use with GlobalFree()) - Storage structure dependent on device type (see NMTcpip.h) nSize - Storage for size of memory block pointed to by hLocn
NM_SetShunt – Set the shunt state of a sensor point on a device. void NM_RqstSetShunt(HANDLE hNetManager, int nDevice, int nPoint, BYTE uMask, BYTE uState); hNetManager - Handle to Network Manager returned by OpenNetwork nDevice - Device number nPoint - Alarm point number uMask - Shunts to change (Bit 0: Alarm shunt, Bit 1: Tamper shunt) uState - State to set (Bit 0: Alarm shunt state, Bit 1: Tamper shunt state. 1 = Shunt active)
NM_SetShunts – Set the shunt state for all sensor points on a device. void NM_RqstSetShunt(HANDLE hNetManager, int nDevice, int nPoints, BYTE* pState); hNetManager - Handle to Network Manager returned by OpenNetwork nDevice - Device number nPoint - Number of Alarm points uState - Pointer to array of bytes containing shunt states, 2 bits per point (Bit 0: Alarm shunt state, Bit 1: Tamper shunt state. 1 = Shunt active) e.g., Byte 0 bit 0 & 1 (least significant bits) contain shunt state for sensor point 1 e.g., Byte 0 bit 6 & 7 contain shunt state for sensor point 3
NM_RqstShunt – Send Shunt status request for all the alarm points on a device. void NM_RqstShunt(HANDLE hNetManager, int nDevice); hNetManager - Handle to Network Manager returned by OpenNetwork nDevice - Device number
NM_GetShunt – Get the shunt status from the current NM_MT_SHUNTS message. Returned: bool - true = success - false = failure, invalid for current message type, device type or point # bool NM_GetShunt(HANDLE hNetManager, int& nPoint, UINT& uStatus); hNetManager - Handle to Network Manager returned by OpenNetwork nPoint - point number uStatus - storage for shunt status (Bit 0: Alarm shunt state, Bit 1 Tamper shunt state. 1 = Shunt active)
NM_SetControl - Set the state of a control point on a device. void NM_SetControl(HANDLE hNetManager, int nDevice, int nControl, BYTE bState); hNetManager - Handle to Network Manager returned by OpenNetwork nDevice - Device number nControl - Control point number bState - State to set (Active = 1, Inactive = 0)

NM_SetControls - This function sets all control points for a device.

void NM_SetControls(HANDLE hNetManager, int nDevice, int nControls, BYTE* pState);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number
nControls	- Number of controls
pState	- Pointer to array of bytes containing control states, 1 bit per control (Active = 1, Inactive = 0)
	- e.g. Byte 0 bit 0 (least significant bit) contains state for control 1
	- e.g. Byte 0 bit 7 contains state for control 8

NM_RqstControl - Send a Control status request message to a Network Manager.

void NM_RqstControl(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number

NM_GetControl - Get the Control status from the current NM_MT_CONTROL or NM_MT_CONTROLS message.

Returned:

bool	- true = Success
	- false = Failure, invalid for current message type, device type or point #

bool NM_GetControl(HANDLE hNetManager, int& nControl, UINT& uStatus);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nControl	- Control number
	- filled by function for NM_MT_CONTROL message
	- specified for NM_MT_CONTROLS messages
bStatus	- Storage for Control status (false = clear, true = active)

NM_RqstAudioFolder - Send Audio Folder request message

void NM_RqstAudioFolder(HANDLE hNetManager, int nChannel);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nChannel	- Channel number

NM_SetAudioFolder - Send Audio Folder message

void NM_SetAudioFolder(HANDLE hNetManager, int nChannel, wchar_t* pPathname);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nChannel	- Channel number
pPathname	- Pointer to null terminated wide character string containing pathname of folder used to store audio recordings for specified channel.

Note: String is in little endian Unicode format

NM_GetAudioFolder - Get Audio Folder from current message

Returned:

bool	- true = Success
	- false = Failure, invalid for current message type

void NM_GetAudioFolder(HANDLE hNetManager, int nChannel, HGLOBAL& hPathname);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nChannel	- Channel number
	- Filled by function for NM_MT_AUDIO_FOLDER message
hPathname	- Storage for handle of null terminated wide character string containing pathname of folder used to store audio recordings for specified channel. (Delete after use with GlobalFree())
	- Filled by function for NM_MT_AUDIO_FOLDER message

Note: String is in little endian Unicode format

NM_GetAudioEvent - Get Audio Event information from current message

Returned:

bool	- true = Success
	- false = Failure, invalid for current message type

void NM_GetAudioEvent(HANDLE hNetManager, int nChannel, HGLOBAL& hFilename, HGLOBAL& hDesc);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nChannel	- Channel number
	- Filled by function for NM_MT_AUDIO_EVENT message
hFilename	- Storage for handle of null terminated wide character string containing file pathname of audio recording for specified channel. (Delete after use with GlobalFree())
	- Filled by function for NM_MT_AUDIO_EVENT message
hDesc	- Storage for handle of null terminated wide character string containing description of audio recording for specified channel. (Delete after use with GlobalFree())
	- Filled by function for NM_MT_AUDIO_EVENT message

Note: Strings are in little endian Unicode format

NM_RqstPointDesc - Send a Point description request message

void NM_RqstPointDesc (HANDLE hNetManager, int nDevice, int nPoint, int nType);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number (n/a for Network descriptions)
nPoint	- Point number (n/a for Device descriptions, n/a for Network descriptions)
nType	- Description Type (0-Device, 1-Comm Point, 2-Diagnostic Point, 3-Sensor Point, 4-Control Point, 5-Summary Point, 6-Network)

NM_GetPointDesc - Get Point description from current message

Returned:

bool	- true = Success
	- false = Failure, invalid for current message type

bool NM_RqstPointDesc (HANDLE hNetManager, int& nPoint, int nType, HGLOBAL& hDesc);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nPoint	- Point number from message (n/a for Device descriptions, n/a for Network descriptions)
nType	- Description Type from message
hDesc	- Storage for handle of null terminated wide character string containing description of device/point. (Delete after use with GlobalFree())

Note: String is in little endian Unicode format

NM_RqstSensInUse - Send the Sensor point in-use request message for the sensor points on a device.

void NM_RqstSensInUse(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number

NM_RqstSmryInUse - Send the Diagnostic Summary point in-use request message for a device.

void NM_RqstSmryInUse(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number

NM_RqstCtrlInUse - Send the Control point in-use request message for the control points on a device.

void NM_RqstCtrlInUse(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number

NM_GetInUse - Get the point status from the current NM_MT_SENSOR_INUSE or NM_MT_SMRY_INUSE message

Returned:

bool	- true = Success
	- false = Failure, invalid for current message type, device type or point #

bool NM_GetAlarm(HANDLE hNetManager, int& nPoint, UINT& uStatus);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nPoint	- Point number
uStatus	- Storage for point status

NM_MT_SENSOR_INUSE
Bit 0: Alarm state supported
Bit 1: Tamper state supported
Bit 2: Pre-alarm state supported
Bit 3: Point in-use

NM_MT_SMRY_INUSE
Bit 0: Point in-use

NM_MT_CONTROL_INUSE
Bit 0: Point available for use

NM_SocketInit - Helper function for when DLL is being used from a secondary thread. Call before using NM_OpenNetwork.

void NM_SocketInit();

NM_RqstVersion - Send the NMS Version request message

void NM_RqstVersion(HANDLE hNetManager);

hNetManager	- Handle to Network Manager returned by OpenNetwork

---

NM_Version - Send Database Version request message for a device the version and build number from the current NM_MT_VERSION message

Returned:

bool	- true = Success
	- false = Failure, invalid for current message type

bool NM_VERSION(HANDLE hNetManager, int& nVer1, int& nVer2, int& nVer3, int& nBuild);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nVer1	- Storage for 1st version digit
nVer2	- Storage for 2nd version digit
nVer3	- Storage for 3rd version digit
nBuild	- Storage for Build number

NM_RqstDbVersion - Send the Database Version request message for a device

void NM_RqstDbVersion(HANDLE hNetManager, int nDevice);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nDevice	- Device number

NM_GetDbVersion - Get the Database Version from the current NM_MT_DB_VERSION message Returned: bool - true = Success - false = Failure, invalid for current message type bool NM_GetDbVersion(HANDLE hNetManager, WORD&wVersion); hNetManager - Handle to Network Manager returned by Open Network wVersion - Storage for database version
NM_RqstLRanges - Send the Linear Range request message to a Network Manager void NM_RqstLRanges(HANDLE hNetManager, int nDevice); hNetManager - Handle to Network Manager returned by OpenNetwork nDevice - Device number
NM_GetLRanges - Get the point location information from the NM_MT_LRANGE_ALARMS message bool NM_GetLRanges(HANDLE hNetManager, int& nRanges, HGLOBAL &hRanges, int& nSize); hNetManager- Handle to Network Manager returned by OpenNetwork hNetManager - Handle to Network Manager returned by OpenNetwork nRanges - Storage for number of point ranges hRanges - Storage for handle to point ranges (delete after use with GlobalFree()) - NM_D_LRANGE_POINT structs (see NMTcpip.h) nSize - Storage for size of memory block pointed to by hRanges
NM_RqstGpsPath - Send the GPS Path request message to a Network Manager void NM_RqstGpsPath(HANDLE hNetManager, int nDevice); hNetManager - Handle to Network Manager returned by OpenNetwork nDevice - Device number
NM_GetGpsPath - Get the path coordinate information from the NM_MT_GPS_PATH_ALARMS message bool NM_GetGpsPath(HANDLE hNetManager, int& nPoint, int& nPathPnts, HGLOBAL &hPath, int& nSize); hNetManager - Handle to Network Manager returned by OpenNetwork nPoint - Storage for alarm point number nPathPnts - Storage for number of path points hPath - Storage for handle to path points (delete after use with GlobalFree()) - NM_D_GPS_COORD structs (see NMTcpip.h) nSize - Storage for size of memory block pointed to by hPath
NM_SetTempConfig - Set the state of a control point on a device void NM_SetTempConfig(HANDLE hNetManager,bool bState int nDevice, int nConfig, int nHours, int nZone); hNetManager - Handle to Network Manager returned by OpenNetwork bState - True applies config for specified duration on specified device, false returns config to normal before the time is up. nDevice - Device Number nControl - Temporary Config Number nHours - Integer number of hours to apply setting. nZone - Integer number representing zone to be adjusted.

NM_GetTempConfig - Get the status from the current NM_MT_TEMP_CONFIG
Returned:

Bool	-True = Success
	-False = Failure, invalid for current message type, device type or point #

bool NM_GetTempConfig (HANDLE hNetManager, int& nConfig, UINT& uStatus , int &nZone);

hNetManager	- Handle to Network Manager returned by OpenNetwork
nConfig	- Config number

- filled by function for NM_MT_TEMP_CONFIG message

nHours	- Integer Amount of hours temp config is scheduled for.
bStatus	- Storage for Config status (false = clear, true = active)
nZone	- int representing zone config has changed for.

NM DLL Test - NMI test application

The NM DLL Test application (TestDLL) is a windows based Visual C++ MFC application, developed using Microsoft Visual Studio 2012. It uses the NM DLL functions to interface with a Network Manager. It is one of the programs used to verify the Network Manager TCP/IP display interface. A second sample application, NM TCP/IP Test (TestTcpIp) is available to demonstrate using the TCP/IP messages to interface with a Network Manager. The following screen shot, shows the main window of the NM DLL Test application. Network Manager connection status and messages are reported in the display space. Menu items to Pause, Resume and Clear the display space are under the Display menu. Also, the NM DLL Status Info dialog can be displayed or hidden from the Display menu. Menu items to connect to a Network Manager and to test available NMI DLL functions are available under the Misc menu.

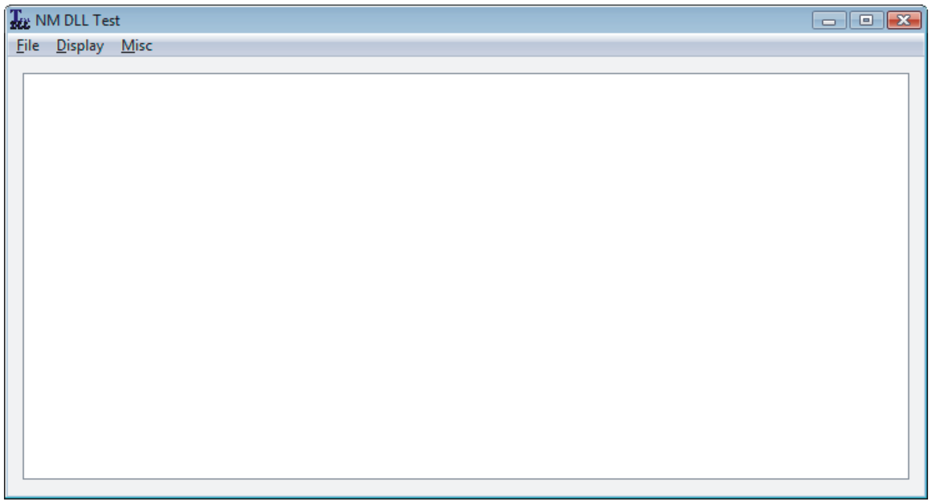
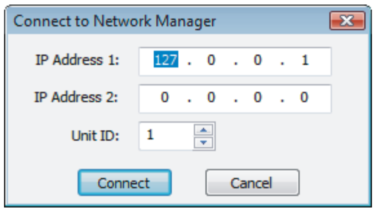


Figure 1: NM DLL Test window

To connect to the Network Manager, select **Misc** > **Connect**. The following popup appears.:



IP Address 1: 127 . 0 . 0 . 1
IP Address 2: 0 . 0 . 0 . 0
Unit ID: 1

Figure 2: NM connection window

1. Enter the **IP address** of the computer running the Network Manager. For redundant Network Managers, enter the **IP address** of each computer. The NMI DLL will search for the active Network Manager using these addresses.

Note	The Network Manager software must be configured with the IP address of the computer running the NM DLL Test program, or the connection will be rejected.

2. Use the **Unit ID** to select a specific Network Manager.
3. Select **Connect** and TestDLL uses the **NM_OpenNetwork** function to make the connection.

The following menu items on the Misc menu become active when connected to a Network Manager:

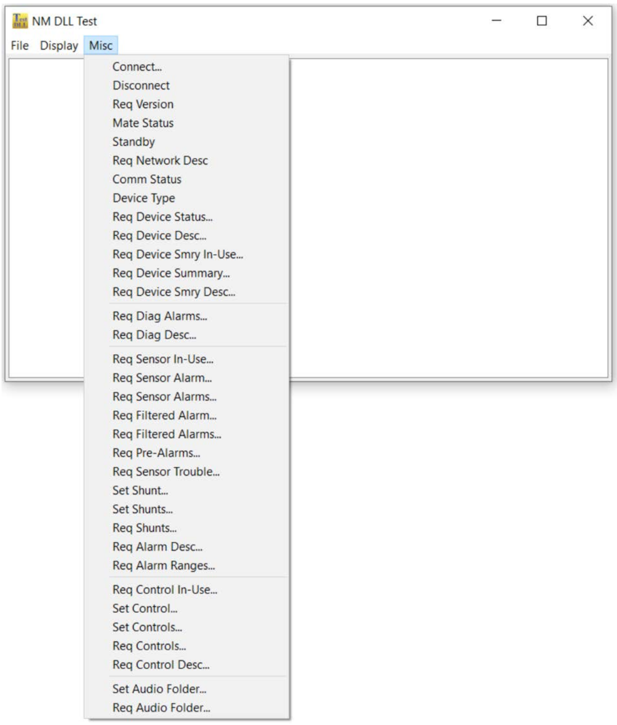


Figure 3: NM DLL Test window - Misc menu

Disconnect

The **NM_CloseNetwork** function is used to disconnect from the Network Manager.

Req Version

The **NM_RqstVersion** function is used to request the version of the Network Manager. The Network Manager responds with the **NM_MT_VERSION** message and TestDLL uses the **NM_Version** function to retrieve the version and reports it in the display area.

Mate Status

The **NM_RqstMate** function is used to request the status of the connection between the active and standby Network Managers. The Network Manager responds with the **NM_MT_MATE _STAT** message and TestDLL uses the **NM_MateStat** function to retrieve the connection status and reports it in the display area. A status of unconnected (false) is always returned from a non-redundant Network Manager.

Standby

The **NM_SetStandby** function is used to request the active Network Manager to enter standby mode and transfer control to the alternate Network Manager. The NMI DLL will automatically connect to the newly active Network Manager and the connection loss and new connection will be reported in the display area. This only applies to redundant NM configurations and will have no effect if the alternate NM is not available to go online.

Req Network Desc

The **NM_RqstPointDesc** function is used to request the description for the network. The Network Manager responds with the **NM_MT_DESC** message and TestDLL uses the **NM_GetPoint Desc** function to retrieve the description.

Comm Status

The **NM_RqstComm** function is used to request the communication status between the Network Manager and all of its nodes. The Network Manager responds with the **NM_MT_COMM_STAT** message and TestDLL uses the **NM_GetComm** function to retrieve the communication status for each device and reports it in the display area.

In addition, the Network Manager sends unsolicited **NM_MT_COMM_STATUS** messages when a device's communication status changes. When this occurs, TestDLL uses **NM_GetComm** function to retrieve the device number and communication status and reports it in the display area.

Req Device Status

Req Device Status performs a combined Req Diag Alarms, Req Sensor Alarms, Req Filtered Alarms, Req Shunts, and Req Controls.

Req Device Desc

The **NM_RqstPointDesc** function is used to request the description for the device. The Network Manager responds with the **NM_MT_DESC** message and TestDLL uses the **NM_GetPointDesc** function to retrieve the description.

Req Device Smry In-Use

The **NM_RqstSmryInUse** function is used to request the summary points in use for the device. The Network Manager responds with the **NM_MT_SMRY_INUSE** message and TestDLL uses the **NM_GetInUse** function to retrieve the in-use points.

Req Device Summary

The **NM_RqstDevcSmry** is used to request a summary of the communication and diagnostic status for a device. The Network Manager responds with the **NM_MT_DEVC_SMRY** message and TestDLL uses the **NM_GetInUse** function to retrieve the in-use points.

In addition, the Network Manager sends unsolicited **NM_MT_DEVC_SMRY** messages when a summary status point changes state, which TestDLL reports in the display area.

Req Device Smry Desc

The **NM_RqstPointDesc** function is used to request the description for the summary point. The Network Manager responds with the **NM_MT_DESC** message and TestDLL uses the **NM_GetPointDesc** function to retrieve the description.

Req Diag Alarms

The **NM_RqstDiagAlrm** function is used to request the Diagnostic alarm status for a device. The Network Manager responds with the **NM_MT_DIAG_ALARMS** message and TestDLL uses the **NM_GetAlarm** function to retrieve the status for a diagnostic alarm point.

In addition, the Network Manager sends unsolicited **NM_MT_DIAG_ALARM** messages when diagnostic alarm points change state. When this occurs, TestDLL uses the **NM_GetAlarm** function to retrieve the point number and state and reports it in the display area.

Req Diag Desc

The **NM_RqstPointDesc** function is used to request the description for the diagnostic point. The Network Manager responds with the **NM_MT_Desc** message and TestDLL uses the **NM_GetPointDesc** function to retrieve the description.

Req Sensor In-Use

The **NM_RqstSensInUse** function is used to request the Sensor alarm points in use. The Network Manager responds with the **NM_MT_SENSOR_INUSE** message and TestDLL uses the **NM_GetInUse** function to retrieve the in-use points.

Req Sensor Alarm (single point variant)

The **NM_RqstSensAlarm** function is used to request the Sensor alarm status for a single sensor alarm point on a device. The Network Manager responds with the **NM_MT_SENSOR_ALARM** message and TestDLL uses the **NM_GetAlarm** function to retrieve the status for sensor alarm point and reports it in the display area. For a ranging sensor (e.g., OmniTrax) TestDll uses the **NM_GetLocn** function to retrieve location information for the alarm and also reports the information in the display area. See NMTcpip.h for how the device type’s location information is provided.

In addition, the Network Manager sends unsolicited **NM_MT_SENSOR_ALARM** messages when a sensor alarm point changes state, which TestDLL reports in the display area

Req Sensor Alarms (all points variant)

The **NM_RqstSensAlrm** function is used to request the status for all sensor alarm points on a device. The Network Manager responds with the **NM_MT_SENSOR_ALARMS** message and TestDLL uses the **NM_GetAlarm** function to retrieve the status for each sensor alarm point and reports them in the display area.

Req Filter Alarm (single point variant)

The **NM_RqstFiltAlarm** function is used to request the Filtered alarm status for a single sensor alarm point on a device. The Network Manager responds with the **NM_MT_FILTER_ALARM** message and the TestDLL uses the **NM_GetFiltAlarm** function to retrieve the status for the sensor alarm point and reports it in the display area. For a ranging sensor (e.g., OmniTrax) TestDll uses the **NM_GetLocn** function to retrieve location information for the alarm, and also reports the information in the display area. See NMTcpip.h for details about how the device type’s location information is provided.

In addition, the Network Manager sends unsolicited **NM_MT_FILTER_ALARM** messages when a sensor alarm point changes state and the new state is not shunted. The Network Manager also sends **NM_MT_FILTER_ALARM** messages in response to changes to the sensor point’s shunt state using the **NM_SetShunt** function (**NM_MT_SHUNT** message). The TestDLL reports the contents of these messages in the display area.

Req Filter Alarms (all points variant)

The **NM_RqstFiltAlarm** is used to request the Filtered alarm status for all sensor alarm points on a device. The Network Manager responds with the **NM_MT_FILTER_ALARMS** message and the TestDLL uses the **GetFiltAlarm** function to retrieve the status for each sensor alarm point and reports them in the display area.

In addition the Network Manager sends **NM_MT_FILTER_ALARMS** messages in response to changes to a device’s shunt status using the **NM_SetShunts** function (**NM_MT_SHUNTS** message). TestDLL reports the contents of these messages in the display area.

Req Pre Alarms

The **NM_RqstPreAlarm** function is used to request the pre-alarm status for all sensor alarm points on a device. The Network Manager responds with the **NM_MT_PRE_ALARMS** message and the TestDLL uses the **NM_GetAlarm** function to retrieve the status for each sensor alarm point and reports them in the display area.

In addition, the Network Manager sends unsolicited **NM_MT_PRE_ALARMS** messages when a sensor alarm point pre-alarm changes state, which TestDLL reports in the display area.

Req Sensor Trouble

The **NM_RqstSensTrbl** is used to request which sensor alarms whose alarm reporting capability might be compromised due to active Comm. fail, Device mismatch or Diagnostic alarms. The Network Manager responds with the **NM_MT_SENSOR_TRBL** message and TestDLL uses the **NM_GetAlarm** function to retrieve the status for each sensor alarm point and reports them in the display area.

In addition, the Network Manager sends unsolicited **NM_MT_SENSOR_TRBL** messages when a sensor alarm point trouble status changes state, which TestDLL reports in the display area.

Set Shunt

The **NM_SetShunt** function is used to set the shunt state for a sensor alarm point on a device. If the message results in a change of state, the Network Manager responds with a **NM_MT_FILTER_ALARM** message containing the resulting filtered alarm state. TestDLL uses the **NM_GetFiltAlarm** function to retrieve the status for the sensor alarm point and reports it in the display area.

Set Shunts

The **NM_SetShunts** function is used to set the shunt state for all sensor alarm points on a device. If the message results in a change of state of a sensor alarm point, the Network Manager responds with a **NM_MT_FILTER_ALARMS** message containing the filtered alarm state of all sensor alarm points on the device. TestDLL uses the **NM_GetFiltAlarm** function to retrieve the status for the sensor alarm point and reports it in the display area.

Req Shunts

The **NM_RqstShunt** function is used to request the shunt status for all sensor alarm points on a device. The Network Manager responds with a NM_MT_SHUNTS message and TestDLL uses the **GetShunt** function to retrieve the shunt status for the sensor alarm points and reports them in the display area.

Req Alarm Desc

The **NM_RqstPointDesc** function is used to request the description for the alarm. The Network Manager responds with the **NM_MT_Desc** message and TestDLL uses the **NM_GetPoint Desc** function to retrieve the description.

Req Alarm Ranges

The **NM_RqstLRanges** function is used to request the linear ranges for a device’s alarm points. The Network Manager responds with the **NM_MT_LRANGE_ALARMS** message and TestDLL uses the **NM_GetLRanges** function to retrieve the alarm point ranges.

Req Alarm Paths

The **NM_RqstGpsPath** function is used to request the GPS path for a device’s alarm points. The Network Manager responds with the **NM_MT_GPS_PATH_ALARMS** messages (one per alarm point with a path defined) and TestDLL uses the **NM_GetGpsPath** function to retrieve the path co-ordinates.

Req Control In-Use

The **NM_RqstCtrlInUse** function is used to request the Control points available for use. The Network Manager responds with the **NM_MT_CONTROL_INUSE** message and TestDLL uses the **NM_GetInUse** function to retrieve the in-use points.

Set Control

The **NM_SetControl** function is used to set the state for a control point on a device. If the message results in a change of state of the control point, the Network Manager responds with a **NM_MT_CONTROL** message containing the updated state. TestDLL uses the **NM_GetControl** function to retrieve the point number and its status and reports it in the display area.

In addition, the Network Manager sends unsolicited **NM_MT_CONTROL** messages when control points change state due to other Display or UCM applications. Silver Network devices will also send control point state changes for points under local control of the device.

Set Controls

The **NM_SetControls** function is used to set the state for all the control points on a device. If the message results in a change of state of a control point, the Network Manager responds with a **NM_MT_CONTROLS** message containing all control point states for the device. TestDLL uses the **NM_GetControl** function to retrieve the statuses and reports them in the display area.

Req Controls

The **NM_RqstControl** function is used to request the status for all the control points on a device. The Network Manager responds with a **NM_MT_CONTROLS** message and TestDLL uses the **NM_GetControl** function to retrieve the status for the control points and reports them in the display area.

Req Control Desc

The **NM_RqstPointDesc** function is used to request the description for the control. The Network Manager responds with the **NM_MT_Desc** message and TestDLL uses the **NM_GetPoint Desc** function to retrieve the description.

Set Audio Folder

The **NM_SetAudioFolder** function is used to specify the folder to be used to store audio recordings for an audio channel.

Req Audio Folder

The **NM_RqstAudioFolder** function is used to request the folder used by an audio channel for recordings. The Network Manager responds with an **NM_MT_AUDIO_FOLDER** message containing the folder name and TestDLL uses the **GetAudioFolder** function and reports it in the display area. (Applies only to VoE Network Manager.)

In addition, the VoE NM sends unsolicited **NM_MT_AUDIO_EVENT** messages when an audio recording ends. TestDLL uses the **NM_GetAudioEvent** function to retrieve the audio channel, the pathname of the recording and the description of the event.

Silver point assignments

The Silver Network defines valid device addresses ranging from 1 to 60.

#define NM_AG_MIN_ADDR 1
#define NM_AG_MAX_ADDR 60

Silver Device defines:

#define NM_AG_OTRX 0X201 // OmniTrax
#define NM_AG_XFLD 0X203 // XField
#define NM_AG_XFLD_LT 0X20A // XField LT
#define NM_AG_IO16 0X204 // 16I/16O
#define NM_AG_4100 0X206 // MPS-4100
#define NM_AG_FLXPS 0X207 // FlexPS
#define NM_AG_UWAVE 0X208 // UltraWave
#define NM_AG_FLXZN 0X20F // FlexZone-60
#define NM_AG_FLXZ20 0X20B // FlexZone-20
#define NM_AG_FLXZ4 0X20D // FlexZone-4
#define NM_AG_AMUX 0X20E // Audio MUX (virtual device)
#define NM_AG_ULIO 0X209 // UltraLink I/O
#define NM_AG_RBX510 0X210 // rBOX510 (embedded controller for NMS)
#define NM_AG_ALE 0X211 // Alarm Logic Engine (virtual device)
#define NM_AG_LM100 0X212 // Senstar LM100
#define NM_AG_FP400 0X213 // FiberPatrol FP400
#define NM_AG_BR100 0X214 // BR100
#define NM_AG_E5000 0x215 // E5000 Physical Security Appliance

The following tables include the point definitions for Silver compatible devices:

OmniTrax input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	Enclosure tamper	9	2 V 5 / 1 V 2 rail fault
2	Program flash error	10	Battery fault
3	RAM error	11	Input power fail
4	Processor boot fail	Cable faults	
5	Option card fail	12	Side A cable supervision
6	8 V rail fault	13	Side B cable supervision
7	3 V 3 rail fault	14	Side A interference (jam)
8	+5 V / - 5 V rail fault	15	Side B interference (jam)
Sensor alarms			
1	AUX input 1 (bit 0 alarm, bit 1 supervision)	7	Option card input Opt5 (bit 0 alarm, bit 1 supervision)
2	AUX input 2 (bit 0 alarm, bit 1 supervision)	8	Option card input Opt6 (bit 0 alarm, bit 1 supervision)
3	Option card input Opt1 (bit 0 alarm, bit 1 supervision)	9	Option card input Opt7 (bit 0 alarm, bit 1 supervision)
4	Option card input Opt2 (bit 0 alarm, bit 1 supervision)	10	Option card input Opt8 (bit 0 alarm, bit 1 supervision)

OmniTrax input point mapping			
Point	Description	Point	Description
5	Option card input Opt3 (bit 0 alarm, bit 1 supervision)	11 to 60	Cable zone #n (bit 0 alarm, bit 1 unused)
6	Option card input Opt4 (bit 0 alarm, bit 1 supervision)		

OmniTrax output point mapping (controls)			
Point	Description	Point	Description
1	Processor relay 1	7	Option card relay Opt3
2	Processor relay 2	8	Option card relay Opt4
3	Processor relay 3	9	Option card relay Opt5
4	Processor relay 4	10	Option card relay Opt6
5	Option card relay Opt1	11	Option card relay Opt7
6	Option card relay Opt2	12	Option card relay Opt8

XField/XField LT input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	Enclosure tamper	10	Battery fault
2	Program flash error	11	Input power fail
3	RAM error	12	Side A supervision
4	Processor boot fail	13	Side B supervision
5	Option card fail	14	Side A spoof
6	8 V rail fault	15	Side B spoof
7	3 V 3 rail fault	16	Jam
8	+5 V / - 5 V rail fault	17	Side A service required
9	2 V 5 / 1 V 2 rail fault	18	Side B service required
Sensor alarms			
1	AUX input 1 - (bit 0 alarm, bit 1 supervision)	7	Option card input Opt5 - (bit 0 alarm, bit 1 supervision)
2	AUX input 2 - (bit 0 alarm, bit 1 supervision)	8	Option card input Opt6 - (bit 0 alarm, bit 1 supervision)
3	Option card input Opt1 - (bit 0 alarm, bit 1 supervision)	9	Option card input Opt7 - (bit 0 alarm, bit 1 supervision)
4	Option card input Opt2 - (bit 0 alarm, bit 1 supervision)	10	Option card input Opt8 - (bit 0 alarm, bit 1 supervision)
5	Option card input Opt3 - (bit 0 alarm, bit 1 supervision)	11	Side A alarm (bit 0 alarm, bit 1 unused)
6	Option card input Opt4 - (bit 0 alarm, bit 1 supervision)	12	Side B alarm (bit 0 alarm, bit 1 unused)

XField/XField LT output point mapping (controls)			
Point	Description	Point	Description
1	Processor relay 1	8	Option card relay Opt4
2	Processor relay 2	9	Option card relay Opt5

XField/XField LT output point mapping (controls)			
Point	Description	Point	Description
3	Processor relay 3	10	Option card relay Opt6
4	Processor relay 4	11	Option card relay Opt7
5	Option card relay Opt1	12	Option card relay Opt8
6	Option card relay Opt2	13	Side A self-test
7	Option card relay Opt3	14	Side B self-test

16I/16O input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	Enclosure tamper	5	3V3 Rail Fault
2	Program flash error	6	5V Rail Fault
3	RAM error	7	Battery fault
4	8V Rail Fault	8	Input power fail
Sensor alarms			
1 - 16	processor inputs 1 - 16 (bit 0: Alarm; bit 1: Tamper)		

16I/16O output point mapping (controls)	
Points	Description
1 - 16	processor relays 1 - 16

MPS-4100 input point mapping			
Point	Description	Point	Description
Sensor alarms			
1	Microwave 1 (bit 0: Alarm, bit 1: tamper)	2	Microwave 2 (bit 0: Alarm, bit 1: tamper)

MPS-4100 output point mapping (controls)			
Points	Description	Points	Description
1	Audio Select 1	3	Self-Test 1-2 + Self-Test LED 1
2	Audio Select 2	4	Self-Test LED 2

FlexPS input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	Enclosure tamper	5	3V3 Rail Fault
2	Program flash error	6	Battery fault
3	RAM error	7	Input power fail
4	8V Rail Fault		
Sensor alarms			
1 - 2	processor Aux inputs 1 - 2 (bit 0: Alarm; bit 1: Tamper)		
3 - 4	Side A/B (bit 0: Alarm; bit 1: Tamper)		

FlexPS output point mapping (controls)	
Points	Description
1 - 4	processor relays 1 - 4
5 - 6	audio select Side A/B
7 - 8	self-test Side A/B

UltraWave input point mapping			
Point	Description	Point	Description
Diagnostic alarms (receiver)			
1	Enclosure tamper	6	5V5 Rail fault
2	Program flash error	7	3V3 Rail fault
3	RAM error	8	5V Rail fault
4	Transmitter link fault	9	Input power fail
5	Transmitter mismatch		
Diagnostic alarms (transmitter)			
10	Enclosure tamper	13	5V5 Rail fault
11	Program flash error	14	3V3 Rail fault
12	Ram error	15	5V Rail fault
		16	Input power fail
Sensor alarms			
1	Receiver Aux input (bit 0: Alarm; bit 1: Tamper)		
2	Microwave (bit 0: Alarm; bit 1: unused)		

UltraWave output point mapping (controls)	
Points	Description
1 - 2	Receiver relays 1 - 2
3	self-test

FlexZone-60 input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	Enclosure tamper	16	Processor offline
2	Program flash error	17	Gate Module Rcvr Fault
3	RAM error	18	Gate Module Rcvr Interference
4	Processor boot fail	19	Gate Module 1 Comm Fail
5	8V Rail fault	20	Gate Module 1 Encosure tamper
6	3V Rail fault	21	Gate Module 1 Program flash error
7	1V8 Rail fault	22	Gate Module 1 I2C Bus fault
8	1V2 Rail fault	23	Gate Module 1 2V7 Rail fault
9	Input power fail	24	Gate Module 1 Low power
7	1V8 Rail fault	25 - 30	Gate Module 2 (see Gate Module 1)
8	1V2 Rail fault	31 - 36	Gate Module 3 (see Gate Module 1)
9	Input power fail	37 - 42	Gate Module 4 (see Gate Module 1)
		43	Option card fail
Cable faults			

FlexZone-60 input point mapping			
Point	Description	Point	Description
10	Side A power fault	13	Side B supervision
11	Side B power fault	14	Side A interference
12	Side A supervision	15	Side B interference
Sensor alarms			
1 - 2	Processor Aux inputs 1 - 2 (bit 0: Alarm; bit 1: Tamper)		
3 - 62	Cable Zone n (bit 0: Alarm; bit 1: unused)		
Gate Modules 1 - 4 (bit 0: Alarm; bit 1: Tamper) NOTE: Alarm and Tamper states are not mutually exclusive for points 63, 65, 67 & 69			
63	GM1 Gate status	67	GM3 Gate status
64	GM1 Aux input	68	GM3 Aux input
65	GM2 Gate status	69	GM4 Gate status
66	GM2 Aux input	70	GM4 Aux input
71 - 74	Option card inputs Opt 1 - Opt 4 (bit 0: Alarm; bit 1: Tamper)		

FlexZone-60 output point mapping (controls)	
Points	Description
1 - 4	Processor relays 1 - 4
5 - 8	Option card relays Opt 1 - Opt 4

FlexZone-20 input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	Enclosure tamper	16	Processor offline
2	Program flash error	17	Gate Module Rcvr Fault
3	RAM error	18	Gate Module Rcvr Interference
4	Processor boot fail	19	Gate Module 1 Comm Fail
5	8V Rail fault	20	Gate Module 1 Encosure tamper
6	3V Rail fault	21	Gate Module 1 Program flash error
7	1V8 Rail fault	22	Gate Module 1 I2C Bus fault
8	1V2 Rail fault	23	Gate Module 1 2V7 Rail fault
9	Input power fail	24	Gate Module 1 Low power
7	1V8 Rail fault	25 - 30	Gate Module 2 (see Gate Module 1)
8	1V2 Rail fault	31 - 36	Gate Module 3 (see Gate Module 1)
9	Input power fail	37 - 42	Gate Module 4 (see Gate Module 1)
		43	Option card fail
Cable faults			
10	Side A power fault	13	Side B supervision
11	Side B power fault	14	Side A interference
12	Side A supervision	15	Side B interference
Sensor alarms			
1 - 2	Processor Aux inputs 1 - 2 (bit 0: Alarm; bit 1: Tamper)		
3 - 22	Cable Zone n (bit 0: Alarm; bit 1: unused)		

FlexZone-20 input point mapping			
Point	Description	Point	Description
Gate Modules 1 - 4 (bit 0: Alarm; bit 1: Tamper) NOTE: Alarm and Tamper states are not mutually exclusive for points 23, 25, 27 & 29			
23	GM1 Gate status	27	GM3 Gate status
24	GM1 Aux input	28	GM3 Aux input
25	GM2 Gate status	29	GM4 Gate status
26	GM2 Aux input	30	GM4 Aux input
31 - 34	Option card inputs Opt 1 - Opt 4 (bit 0: Alarm; bit 1: Tamper)		

FlexZone-20 output point mapping (controls)	
Points	Description
1 - 4	Processor relays 1 - 4
5 - 8	Option card relays Opt 1 - Opt 4

FlexZone-4 input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	Enclosure tamper	16	Processor offline
2	Program flash error	17	Gate Module Rcvr Fault
3	RAM error	18	Gate Module Rcvr Interference
4	Processor boot fail	19	Gate Module 1 Comm Fail
5	8V Rail fault	20	Gate Module 1 Encosure tamper
6	3V Rail fault	21	Gate Module 1 Program flash error
7	1V8 Rail fault	22	Gate Module 1 I2C Bus fault
8	1V2 Rail fault	23	Gate Module 1 2V7 Rail fault
9	Input power fail	24	Gate Module 1 Low power
7	1V8 Rail fault	25 - 30	Gate Module 2 (see Gate Module 1)
8	1V2 Rail fault	31 - 36	Gate Module 3 (see Gate Module 1)
9	Input power fail	37 - 42	Gate Module 4 (see Gate Module 1)
		43	Option card fail
Cable faults			
10	Side A power fault	13	Side B supervision
11	Side B power fault	14	Side A interference
12	Side A supervision	15	Side B interference
Sensor alarms			
1 - 2	Processor Aux inputs 1 - 2 (bit 0: Alarm; bit 1: Tamper)		
3 - 6	Cable Zone n (bit 0: Alarm; bit 1: Unused)		
Gate Modules 1 - 4 (bit 0: Alarm; bit 1: Tamper) NOTE: Alarm and Tamper states are not mutually exclusive for points 7, 9, 11 & 13			
7	GM1 Gate status	11	GM3 Gate status
8	GM1 Aux input	12	GM3 Aux input
9	GM2 Gate status	13	GM4 Gate status
10	GM2 Aux input	14	GM4 Aux input

FlexZone-4 input point mapping			
Point	Description	Point	Description
15 - 18	Option card inputs Opt 1 - Opt 4 (bit 0: Alarm; bit 1: Tamper)		

FlexZone-4 output point mapping (controls)	
Points	Description
1 - 4	Processor relays 1 - 4
5 - 8	Option card relays Opt 1 - Opt 4

Audio MUX-60 output point mapping (controls)	
Points	Description
1 - 100	Audio select

UltraLink input point mapping			
Point	Description	Point	Description
Diagnostic alarms (Processor card)			
1	Program flash error	3	5V5 Rail fault
2	RAM error	4	3V3 Rail fault
Diagnostic alarms (expansion card 1)			
5	Card fail	8	RAM error
6	Card mismatch	9	3V3 Rail fault
7	Program flash error		
Expansion card 2 (10 - 14) see expansion card 1 description			
Expansion card 3 (15 - 19) see expansion card 1 description			
Expansion card 4 (20 - 24) see expansion card 1 description			
Expansion card 5 (25 - 29) see expansion card 1 description			
Expansion card 6 (30 - 34) see expansion card 1 description			
Expansion card 7 (35 - 39) see expansion card 1 description			
Expansion card 8 (40 - 44) see expansion card 1 description			
Sensor alarms			
1 - 8	processor inputs 1 - 8 (bit 0: Alarm; bit 1: Tamper)		
9 - 40	expansion card 1 inputs 1 - 32 (bit 0: Alarm; bit 1: Tamper)		
41 - 72	expansion card 2 inputs 1 - 32 (bit 0: Alarm; bit 1: Tamper)		
73 - 104	expansion card 3 inputs 1 - 32 (bit 0: Alarm; bit 1: Tamper)		
105 - 136	expansion card 4 inputs 1 - 32 (bit 0: Alarm; bit 1: Tamper)		
137 - 168	expansion card 5 inputs 1 - 32 (bit 0: Alarm; bit 1: Tamper)		
169 - 200	expansion card 6 inputs 1 - 32 (bit 0: Alarm; bit 1: Tamper)		
201 - 232	expansion card 7 inputs 1 - 32 (bit 0: Alarm; bit 1: Tamper)		
233 - 264	expansion card 8 inputs 1 - 32 (bit 0: Alarm; bit 1: Tamper)		

UltraLink output point mapping (controls)	
Points	Description
1 - 8	Processor relays 1 - 8
9 - 40	expansion card 1 relays/open collector outputs 1 - 32
41 - 72	expansion card 2 relays/open collector outputs 1 - 32

UltraLink output point mapping (controls)	
Points	Description
73 - 104	expansion card 3 relays/open collector outputs 1 - 32
105 - 136	expansion card 4 relays/open collector outputs 1 - 32
137 - 168	expansion card 5 relays/open collector outputs 1 - 32
169 - 200	expansion card 6 relays/open collector outputs 1 - 32
201 - 232	expansion card 7 relays/open collector outputs 1 - 32
233 - 264	expansion card 8 relays/open collector outputs 1 - 32

rBOX510 embedded controller input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
None			
Sensor alarms			
1 - 8	DIO inputs 1 - 8		

rBOX510 embedded controller output point mapping (controls)	
Points	Description
1 - 8	DIO outputs 1 - 8

Alarm Logic Engine input point mapping (virtual device)	
Point	Description
Diagnostic alarms	
None	
Sensor alarms	
1 - 400	Logic Elements (bit 0: Alarm, bit 1: tamper) NOTE: Alarm and tamper states are not mutually exclusive.

Alarm Logic Engine output point mapping (controls) (virtual device)	
Points	Description
1 - 400	Fanout points

LM100 input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	Enclosure tamper	6	8V Rail fault
2	Program flash error	7	3V Rail fault
3	RAM error	8	3V9 Rail fault
4	Head Net fail	9	Input power fail
5	Option card fail		
10, 12 ... 108	Zone 1 - 50 Critical Head Diagnostics	11, 13 ... 109	Zone 1-50 Warning Head Diagnostics
110	Gate Module Receiver fault	116	Gate Module 1 2V7 Rail fault
111	Gate Module Receiver interference	117	Gate Module 1 Low power
112	Gate Module 1 Comm fail	118 - 123	Gate Module 2 (see Gate Module 1)
113	Gate Module 1 Enclosure tamper	124 - 129	Gate Module 3 (see Gate Module 1)
114	Gate Module 1 Program Flash error	130 - 135	Gate Module 4 (see Gate Module 1)

LM100 input point mapping			
Point	Description	Point	Description
115	Gate Module 1 I2C Bus fault		
Sensor alarms			
1 - 10	Gateway Aux inputs 1 - 10 (bit 0: Alarm; bit 1: Tamper)		
11 - 14	Option card inputs Opt 1 - Opt 4 (bit 0: Alarm; bit 1: Tamper)		
15 - 64	Zone n alarms (bit 0: Alarm; bit 1: Tamper) NOTE: Alarm and Tamper states are not mutually exclusive		
Gate Modules 1 - 4 (bit 0: Alarm; bit 1: Tamper) NOTE: Alarm and Tamper states are not mutually exclusive for Gate Status			
65	GM1 Gate status	69	GM3 Gate status
66	GM1 Aux input	70	GM3 Aux input
67	GM2 Gate status	71	GM4 Gate status
68	GM2 Aux input	72	GM4 Aux input
73 - 122	Zone n Aux alarms (bit 0: Alarm; bit 1: Tamper) NOTE: Alarm and Tamper states are not mutually exclusive		

LM100 output point mapping (controls)	
Points	Description
1 - 10	Gateway relays 1 - 10
11 - 14	Option card relays Opt 1 - Opt 4
15 - 64	Zone n lights - activate at lighting level 1
65 - 114	Zone n lights - activate at lighting level 2
115 - 164	Zone n lights - activate at alarm lighting level
165 - 214	Zone n lights - force OFF (override any other lighting triggers)

FP400 input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	Program flash error	5	6V5 rail fault
2	RAM error	6	3V3 rail fault
3	Laser fault	7	Input power fail
4	Option card fail		
Sensor alarms			
1 - 2	Processor AUX inputs 1 - 2 (bit 0: Alarm; bit 1: Tamper)		
2	Option card inputs Opt 1 - Opt 4 (bit 0: Alarm; bit 1: Tamper)		
3	Zone 1 - 4 (bit 0: Alarm; bit 1: Tamper)		

FP400 output point mapping (controls)	
Point	Description
1 - 6	Processor relays 1 - 6
7 - 10	Option card relays 1 - 4

BR100 input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	Enclosure Tamper	7	3V3 regulator fault
2	Program flash error	8	Ethernet regulator fault
3	Flash error	9	RF regulator fault
4	Synth Lock fail	10	RS422 A regulator fault
5	I^2C bus fault	11	RS422 B regulator fault
6	3V6 rail fault	12	Input power fail
Sensor alarms			
1	Processor AUX input 1 (bit 0: Alarm; bit 1: Tamper)		
2	Microwave, Monostatic & Bistatic RX modes only (bit 0: Alarm; bit 1: unused)		

BR100 output point mapping (controls)	
Point	Description
1 - 2	Processor relays 1 - 2
3	Self-test

E5000 input point mapping	
Point	Description
Diagnostic alarms	
None	
Sensor alarms	
1 - 8	GPIO Inputs 1 - 8

E5000 output point mapping (controls)	
Points	Description
1 - 8	GPIO Outputs 1 - 8

FiberPatrol point assignments

FiberPatrol defines valid device addresses ranging from 1 to 10.

#define NM_FP_MIN_ADDR 1

#define NM_FP_MAX_ADDR 10

FiberPatrol Device defines:

#define NM_FP_SU   0X700 // FiberPatrol Sensor Unit

#define NM_FP_RSU   0X701 // FiberPatrol Redundant Sensor Unit (virtual device)

The FiberPatrol Redundant Sensor Unit (RSU) is a virtual device which combines the Sensor (Zone) alarms reported by two identical FiberPatrol Sensor Units and reports the alarms as a single SU. The RSU is configured by designating two Sensor Units with one as the Primary Sensor Unit (PSU) and the second as the Alternate Sensor Unit (ASU). The two SUs and the virtual RSU must be defined in the same FiberPatrol Network Manager service (i.e., 2 SUs and 1 RSU defined on 1 network manager). The RSU reports a Sensor alarm from either the PSU or ASU depending on the point's current critical trouble status reported by the PSU and ASU. A critical trouble status is reported for a zone if there is a communication or diagnostic alarm reported that could block the detection or reporting of a Sensor alarm. For example, if the PSU reports a critical trouble condition in Zone 3, the RSU will use the Zone 3 sensor alarm data from the ASU.

The following table outlines which source's Sensor Alarm status is reported by the RSU depending on the reported supervision and trouble states of the PSU and ASU.

PSU Status	ASU Status	RSU Zone Alarm	RSU Zone Trouble
OK	OK	PSU status	PSU status (i.e., OK)
OK	Trouble	PSU status	PSU status (i.e., OK)
Trouble	OK	ASU status	ASU status (i.e., OK)
Trouble	Trouble	PSU status	PSU status

The RSU does not combine and report the Device Summary, Communication and Diagnostic alarms of the PSU and ASU. Monitoring of device summary (NM_MT_DEVC_SMRY), communication (NM_MT_COMM_STAT) and diagnostic alarms (NM_MT_DIAG_ALARM(S)) should be done via the 2 physical Sensor Units (PSU and ASU).

For Sensor Units designated as the Primary or Alternate source for an RSU:

1. Alarm reporting via NM_MT_SENSOR_ALARM(S) and NM_MT_FILTER_ALARM(S) is suppressed as the alarms are reported by the RSU. If an alarm point status is queried the NMS will respond with an inactive status.
2. The Sensor In-use message (NM_MT_SENSOR_INUSE) will report all points as not in-use. The associated RSU will report the points as in-use.
3. Primary and Alternate Sensor Units must be configured so that zones are physically aligned (identical zone layouts).

The following tables include the point definitions for FiberPatrol compatible devices:

FiberPatrol SU input point mapping	
Point	Description
Diagnostic Alarms	
1	Processor Fault
2	COM Fault (Processor to Controller Communication)
3	Transmitter Fault (Controller)
4	Receiver Fault (Controller)
5	ADC Fault (Processor Signal Digitizer)
6	Environmental Fault (Ambient Temperature)
7	Sensor 1 Fault (Channel 1 Signal)
8	Sensor 2 Fault (Channel 2 Signal)
9	Optical Power Fault (Low in both Channels)
10	Fiber Cut
Sensor Alarms	
1 - 1440	Zones 1 - 1440 (bit 0: Alarm, bit 1: cut)

FiberPatrol RSU input point mapping (virtual device)	
Point	Description
Diagnostic Alarms (see physical SU assigned)	
None	
Sensor Alarms	
1 - 1440	Zones 1 - 1440 (bit 0: Alarm, bit 1: cut)

CCC point assignments

The CCC network defines valid device addresses ranging from 0 to 127 (0 to 120 when using “Boris” Card).

#define NM_C3_MIN_ADDR 0

#define NM_C3_MAX_ADDR 127

CCC Device defines:

#define NM_C3_FOST 0X600 // Innofence Fiber Optic Sensor Transducer

#define NM_C3_GPRU 0X601 // General Purpose Reporting Unit

#define NM_C3_SPRU 0X602 // DTR Sensor Post Reporting Unit

#define NM_C3_VPRU 0X603 // Barricade Vibration Processing Reporting Unit

#define NM_C3_YAEL 0X604 // YAEL-16 Piezo Taut Wire Processor

The following tables include the point definitions for CCC compatible devices:

FOST input point mapping	
Point	Description
Diagnostic Alarms	
1	Enclosure tamper
Sensor Alarms	
1 - 2	Aux Inputs 1 - 2 (bit 0: Alarm, bit 1: unused)
3 - 4	Fiber 1 - 2 (bit 0: Alarm, bit 1: Tamper)

FOST output point mapping (controls)	
Point	Description
1	Self Test

GPRU input point mapping	
Point	Description
Sensor Alarms	
1 - 8	Inputs 1 - 8 (bit 0: Alarm, bit 1: unused)

GPRU output point mapping (controls)	
Point	Description
1 - 9	Relays 1 - 9
10	Self Test

SPRU input point mapping	
Point	Description
Diagnostic Alarms	
1	Enclosure Tamper
Sensor Alarms	
1	Taut-wire group 1 lower (bit 0: Alarm, bit 1: unused)
2	Taut-wire group 2 lower (bit 0: Alarm, bit 1: unused)
3	Taut-wire group 3 lower (bit 0: Alarm, bit 1: unused)

SPRU input point mapping	
Point	Description
4	Taut-wire group 4 lower (bit 0: Alarm, bit 1: unused)
5	Taut-wire group 5 lower (bit 0: Alarm, bit 1: unused)
6	Taut-wire group 1 upper (bit 0: Alarm, bit 1: unused)

SPRU output point mapping (controls)	
Point	Description
1	Self Test

VPRU input point mapping	
Point	Description
Diagnostic Alarms	
1	Enclosure Tamper
2	Input Power Fault
Sensor Alarms	
1	Sensor Line 1 (bit 0: Alarm, bit 1: Tamper)
2	Sensor Line 2 (bit 0: Alarm, bit 1: Tamper)
3	Sensor Line 3 (bit 0: Alarm, bit 1: Tamper)
4	Sensor Line 4 (bit 0: Alarm, bit 1: Tamper)

VPRU output point mapping (controls)	
Point	Description
1 - 2	Relays 1 - 2
3	Self Test

YAEL input point mapping	
Point	Description
Diagnostic Alarms	
1	Enclosure Tamper
2	Input Power Fault
3	Detector Current Fault
4	Detector Amplifier Alarm
Sensor alarms	
1	Fence (bit 0: Alarm, bit 1: Tamper)

YAEL output point mapping (controls)	
Point	Description
1 - 3	TTL Output 1 - 3
4	Self Test

Crossfire point assignments

The Crossfire network defines valid device addresses ranging from 0 to 127.

#define NM_XF_MIN_ADDR 0

#define NM_XF_MAX_ADDR 127

Crossfire Device defines:

#define NM_XF_410   0X001 // PLC-410

#define NM_XF_420   0X002 // PLC-420

#define NM_XF_430   0X003 // PLC-430

#define NM_XF_4100   0X004 // MPS-4100 (Intelli-WAVE)

#define NM_XF_IFLX   0X005 // Intelli-FLEX

#define NM_XF_IFLD   0X006 // Intelli-FIELD

The following tables include the point definitions for Crossfire compatible devices:

PLC-410 input point mapping	
Point	Description
Sensor Alarms	
1 - 64	Inputs 1 - 64 (bit 0: Alarm)

PLC-410 output point mapping (controls)	
Point	Description
1 - 128	Controls 1 - 128

PLC-420 input point mapping	
Point	Description
Sensor Alarms	
1 - 16	Card 1 IO-101/102 inputs 1 - 16 (bit 0: Alarm, bit 1: unused)
17 - 32	Card 1 IO-101 inputs 17 - 32 (bit 0: Alarm, bit 1: unused)
	IO-102 inputs 1 - 16 (bit 0: Tamper, bit 1: unused)
33 - 48	Card 2 IO-101/102 inputs 1 - 16 (bit 0: Alarm, bit 1: unused)
49 - 64	Card 2 IO-101 inputs 17 - 32 (bit 0: Alarm, bit 1: unused)
	IO-102 inputs 1 - 16 (bit 0: Tamper, bit 1: unused)

PLC-420 output point mapping (controls)	
Point	Description
1 - 16	Card 1 IO-201 outputs 1 - 16
	Card 1A IO-202 outputs 1 - 16
17 - 32	Card 1 IO-201 outputs 17 - 32
	Card 1B IO-202 outputs 1 - 16
33 - 48	Card 2 IO-201 outputs 1 - 16
	Card 2A IO-202 outputs 1 - 16
49 - 64	Card 2 IO-201 outputs 17 - 32
	Card 2B IO-202 outputs 1 - 16

PLC-430 input point mapping	
Point	Description
Sensor Alarms	
1 - 8	Inputs 1 - 8 (bit 0: Alarm, bit 1: Tamper)

PLC-430 output point mapping (controls)	
Point	Description
1 - 8	Outputs 1 - 8

MPS-4100 input point mapping	
Point	Description
Sensor Alarms	
1 - 2	Microwave 1 - 2 (bit 0: Alarm, bit 1: Tamper)

MPS-4100 output point mapping (controls)	
Point	Description
1	Audio Select 1 (Audio 2 LED)
2	Audio Select 2 (Audio 1 LED)
3	Self-Test 1 & 2 (Self-Test LED 1)
4	Self-Test LED 2

Intelli-FLEX input point mapping	
Point	Description
Diagnostic Alarms	
1	Enclosure Tamper
2	Low Power Fail
Sensor alarms	
1 - 2	Aux Inputs 1 - 2 (bit 0: Alarm, bit 1: Tamper)
3 - 4	Side A/B (bit 0: Alarm, bit 1: Tamper)

Intelli-FLEX output point mapping (controls)	
Point	Description
1 - 2	Relays 1 - 2
3 - 4	Audio Select Side A/B
5 - 6	Self-Test Side A/B

Intelli-FIELD input point mapping	
Point	Description
Diagnostic Alarms	
1	Enclosure Tamper
Sensor Alarms	
1 - 2	Aux Inputs 1 - 2 (bit 0: Alarm, bit 1: Tamper)
3 - 4	Side A/B (bit 0: Alarm, bit 1: Tamper)

Intelli-FIELD output point mapping (controls)	
Point	Description
1 - 6	Relays 1 - 6 (labelled Suprv A/B, Alarm A/B, Enclosure, Power Fail)
7 - 8	Self-Test Side A/B

Sennet point assignments

The Sennet Network defines valid device addresses ranging from 0 to 62, with address 0 reserved for the Network Controller:

#define NM_SN_MIN_ADDR 0

#define NM_SN_MAX_ADDR 62

Sennet Device defines:

#define NM_SN_TU 0X100 // TU

#define NM_SN_LTU 0X101 // LTU

#define NM_SN_SM 0X103 // Perimitrax SM

#define NM_SN_IF 0X105 // IntelliFlex

#define NM_SN_NC 0X106 // Network Controller

The following tables include the point definitions for Sennet compatible devices:

Network Controller input point mapping	
Point	Description
Diagnostic alarms	
1	ROM Error
2	RAM Error
3	Enclosure Tamper
4	AC Power Fail

Transponder Unit input point mapping	
Point	Description
Diagnostic alarms	
1	ROM Error
2	RAM Error
3	Enclosure Tamper
4	AC Power Fail
Sensor alarms	
1 - 16	Inputs 1 - 16 (bit 0: Alarm, bit 1: Tamper)

Transponder Unit output point mapping (controls)	
Point	Description
1 - 8	Relays 1 - 8

Large Transponder Unit input point mapping	
Point	Description
Diagnostic alarms	
1	ROM Error
2	RAM Error
3	Enclosure Tamper
Sensor alarms	
1 - 256	Inputs 1 - 256 (bit 0: Alarm, bit 1: Tamper)

Large Transponder Unit output point mapping (controls)

Point	Description
1 - 256	Relays 1 - 256

Perimitrax Sensor Module input point mapping			
Point	Description		
Diagnostic alarms			
1	ROM Error	5	RX Cable Fault
2	RAM Error	6	TX Cable Fault
3	Enclosure Tamper	7	Voltage Fault
4	EEPROM Error	8	Temperature Fault
Sensor alarms			
1 - 8	Aux Inputs 1 - 8 (bit 0: Alarm, bit 1: Tamper)		
9 - 10	Side A/B (bit 0: Alarm, bit 1: Unused)		

Perimitrax Sensor Module output point mapping (controls)	
Point	Description
1 - 4	Relays 1 - 4
5 - 6	Self-Test Side A/B

Intelli-FLEX input point mapping	
Point	Description
Diagnostic alarms	
1	RAM Error
2	Enclosure Tamper
3	Low Power Fail
Sensor alarms	
1 - 2	Aux Inputs 1 - 2 (bit 0: Alarm, bit 1: Tamper)
3 - 4	Side A/B (bit 0: Alarm, bit 1: Tamper)

Intelli-FLEX output point mapping (controls)	
Point	Description
1 - 2	Relays 1 - 2
3 - 4	Audio Select Side A/B
5 - 6	Self-Test Side A/B

Sentrax point assignments

The Sentrax Network defines valid device addresses ranging from 0 to 16, with address 0 reserved for the Sentrax Control Module:

#define NM_SX_MIN_ADDR 0

#define NM_SX_MAX_ADDR 16

Sentrax Device defines:

#define NM_SX_CM 0X500 // Control Module

#define NM_SX_LTM 0X501 // Transceiver Module

The following tables include the point definitions for Sentrax compatible devices:

Control Module input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	RAM Error	6	TX Cable Fault
2	ROM Error	7	RX Cable Fault
3	Enclosure Tamper	8	TX Cable Short
4	TM Power OFF	9	RX Cable short
5	CM Offline	10	TX Cable Open
		11	RX Cable Open

Control Module output point mapping (controls)			
Point	Description	Point	Description
1 - 32	Option Board Relays 1 - 32	33	Global Self-Test

Transceiver Module input point mapping			
Point	Description		
Diagnostic alarms			
1	Enclosure Tamper		
Sensor alarms			
Note: Aux input Alarm and Tamper states are not mutually exclusive			
1	Aux input Side A:X (bit 0: Alarm, bit 1: Tamper)	4	Aux input Side B:Y (bit 0: Alarm, bit 1: Tamper)
2	Aux input Side B:X (bit 0: Alarm, bit 1: Tamper)	5 - 6	Side A/B (bit 0: Alarm, bit 1: Unused)
3	Aux input Side A:Y (bit 0: Alarm, bit 1: Tamper)		
Controls			
1 - 4	Aux Outputs 1 - 4		

Voice over Ethernet point assignments

The VoE Network defines valid device addresses ranging from 1 to 200.

#define NM_VO_MIN_ADDR 1

#define NM_VO_MAX_ADDR 200

VoE Device defines:

#define NM_VO_IPCC   0X400 // IP Cell Call

The following tables include the point definitions for VoE compatible devices:

IPCC input point mapping	
Point	Description
Diagnostic alarms	
1	3V3 rail fault
2	1V8 rail fault
3	5V rail fault
4	2V5 rail fault
Sensor alarms	
1 - 4	Processor inputs 1 - 4 (bit 0: Alarm, bit 1: Tamper)

IPCC output point mapping (controls)	
Point	Description
1 - 4	Processor relays 1 - 4 (UCM programmed state for relays)
5 - 8	Processor relays 1 - 4 (latch relay ON overriding programmed state)
9	Audio Channel 1 Ctrl IPCC Slct
10	Audio Channel 1 Dest IPCC Slct
11	Audio Channel 2 Ctrl IPCC Slct
12	Audio Channel 2 Dest IPCC Slct
13	Audio Channel 3 Ctrl IPCC Slct
14	Audio Channel 3 Dest IPCC Slct
15	Audio Channel 4 Ctrl IPCC Slct
16	Audio Channel 4 Dest IPCC Slct
17	Audio Channel 5 Ctrl IPCC Slct
18	Audio Channel 5 Dest IPCC Slct

MX point assignments

The MX network defines valid device addresses ranging from 0 to 120, with address 0 reserved for the MX-5000/MX-7000 Controller.

#define NM_MX_MIN_ADDR 0

#define NM_MX_MAX_ADDR 120

MX Device defines:

#define NM_MX_5000   0X300 // MX-5000

#define NM_MX_7000   0X300 // MX-7000

#define NM_MX_ZN   0X301 // Zone

The following tables include the point definitions for MX compatible devices:

MX-5000/MX-7000 input point mapping			
Point	Description	Point	Description
Diagnostic Alarms			
1	Fiber Fault	3	Input Power Fault
2	Battery Fault		

MX-5000/MX-7000 output point mapping (controls)	
Point	Description
1	Global Self-Test (Note: Test takes approximately 15 seconds to execute. Only one Global or Zone test may run at one time.)

Zone input point mapping	
Point	Description
Diagnostic Alarms	
1	Trouble
Sensor Alarms	
1	Status (bit 0: alarm, bit 1: tamper)

Zone output point mapping (controls)	
Points	Description
1	Audio Select (Note: Audio may only be selected one zone at a time.)
2	Alarm Access (Note: Invalid if alarm condition active.)
3	VPRU output point mapping (controls)

Starcom point assignments

Note: To date all implementations of Starcom have only supported device # 00 (device controller).

#define NM_SC_MIN_ADDR 0

#define NM_SC_MAX_ADDR 0

Starcom Device defines:

#define NM_SC_DEV   0X900 // Starcom Device

The following tables include the point definitions for Starcom devices:

Starcom input point mapping			
Point	Description	Point	Description
Diagnostic alarms			
1	ROM Error	4	Device Fault
2	RAM Error	5	Power Fault
3	Data Error		
Sensor Alarms			
1 - 1024	Input Points 0 - 1023 (bit 0: Alarm, bit 1: Tamper)		

Starcom output point mapping (controls)	
Point	Description
1 - 1024	Output Points 0 - 1023

Recommended NMI Message Use Summary

The following table is a summary of the recommended and most commonly used messages defined in the Network Manager Interface document. Refer to the appropriate section in the Network Manager Interface application note for complete message details.

The => and <= symbols in the description column indicate information flow to (=>) and from (<=) the Network Manager.

Message	Description	Usage
NM_MT_LOOPBACK	• Originator sends 2 data bytes. • Recipient replies with inverted value of bytes.	Can be used as a heartbeat to confirm the sanity of the recipient.
NM_MT_DEVC_TYPE	=> Query NM for the device type of 1 or all sensors. <= NM Reply identifies the device type and the number of diagnostic alarms, sensor alarms and control points supported by the device.	Used to identify the sensors that are being monitored by a Network Manager
NM_MT_DEVC_SMRY	=> Query NM for a summary of comm and diagnostic alarms for a sensor. <= NM Reply identifies a critical and warning level report of communication and self diagnostic alarms for each sensor. • NM sends unsolicited in the event of a status change.	Recommend using this message in place of the NM_MT_COMM_STAT, NM_MT_DIAG_ALARM and NM_MT_DIAG_ALARMS messages. It provides an operator appropriate summary of the detailed alarms reported in those messages. Currently 5 summary alarm points are defined.
NM_MT_SMRY_INUSE	=> Request NM for which summary points are valid for a sensor. <= NM Reply identifies the summary points supported by a sensor.	Not all summary bits are valid for all sensor types. This can be used to identify the applicable summary bits in a system configuration dialog.
NM_MT_SENSOR_ALARM	=> Query NM for raw status of a sensor's alarm point. <= NM Reply provides intrusion and tamper status for an alarm point. For sensors supporting target location, the target(s) location is appended. • NM sends unsolicited in the event of a status change. • NM can be configured to send unsolicited message if target location changes.	Provides raw status of an alarm point Monitor for the alarm point's changes of states. If using NM alarm shunt capability, use the NM_MT_FILTER_ALARM message.
NM_MT_SENSOR_ALARMS	=> Query NM for raw status of all of a sensor's alarm points. <= NM Reply provides intrusion and supervision status for all of the sensor's alarm points.	Useful for initial poll of the current raw status of a sensor's alarm points. If using NM alarm shunt capability use the NM_MT_FILTER_ALARMS message.

NM_MT_SHUNT	=> Request NM to set the shunt status of a sensor alarm point. => Query NM for the shunt status of a sensor alarm point. <= Reply NM provides shunt status of alarm point.	Used to shunt an alarm point so the NM will not report the point's alarms. Usually done to suppress intrusion alarm reporting due to authorized activity (e.g., landscape maintenance). Tamper alarms are not usually shunted as they are considered to be the result of malicious action rather than authorized activity.
NM_MT_FILTER_ALARM	=> Query NM for filtered status of a sensor's alarm point. <= NM Reply provides intrusion and tamper status for alarm point. For sensors supporting target location, the location(s) is appended. • NM sends unsolicited in the event of a status change. • Changing state of associated shunt state can result in a status change if the raw alarm state is active (cause unsolicited message). • NM can be configured to send unsolicited message if the target location changes.	Provides the status of alarm points filtered by shunt state. Monitor for alarm point state changes.
NM_MT_FILTER_ALARMS	=> Query NM for filtered status of all of a sensor's alarm points. <= NM Reply provides intrusion and supervision status for all of the sensor's alarm points.	Useful for initial poll of current filtered status of a sensor's alarm points.
NM_MT_SENSOR_TRBL	=> Query NM for the trouble state of all of a sensor's alarm points. <= NM Reply provides trouble status for all the sensor's alarm points. • NM sends unsolicited in event of status change.	A critical communication or diagnostic alarm on a sensor may affect the ability of the sensor to report alarms for one or more alarm points. An active trouble state for an alarm point indicates the point's operation may be compromised.
NM_MT_PRE_ALARMS	=> Query NM for the pre-alarm state of all of a sensor's alarm points. <= NM Reply provides pre-alarm status for all of the sensor's alarm points. • NM sends unsolicited in the event of a status change.	Some sensors can report a pre-alarm state before an alarm is reported. This can be used to direct PTZ cameras to enable pre-alarm video recording before an alarm is reported.
NM_MT_SENSOR_INUSE	=> Request NM which sensor alarms are valid for a sensor. <= NM Reply identifies the sensor alarm points that a sensor currently supports.	Not all sensor alarm points support all alarm states (intrusion, tamper and pre-alarm). In addition, some points may not be applicable due to missing optional hardware or the sensor's configuration settings. The information in this message can be used to identify the available points and alarm states in a system configuration dialog.

NM_MT_CONTROL	=> Request NM to set the active/inactive state of a sensor's control point. => Query NM for state of a sensor's control point. <= NM Reply reports current state of sensor output. • NM sends unsolicited in the event of a status change.	Used to set the status of a sensor's output. A control point could be a physical output like a relay, or it could trigger a sensor alarm self-test.
NM_MT_CONTROLS	=> Request NM to set or clear multiple sensor control points. => Query NM for state of all of a sensor's controls. <= NM Reply provides status for all of the sensor's controls.	Useful for initial poll of the current status of a sensor's control points or for setting the initial state for all of the sensor controls.
NM_MT_CONTROL_INUSE	=> Request NM which sensor control points are valid for a sensor. <= NM Reply identifies the sensor control points that a sensor currently supports.	Some control points may not be currently applicable due to missing optional hardware or the sensor's configuration settings. The information in this message can be used to identify the available controls in a system configuration dialog.
NM_MT_VERSION	=> Request current NM software version. >= NM Reply contains software version.	Useful for checking if the NM supports a feature introduced at a particular software version.
NM_MT_DB_VERSION	=> Request the database version of a device <= NM Reply contains database version • NM sends unsolicited in the event of a detected version change.	If SMS makes use of point In-use information and wants to dynamically respond to changes made to a sensor; this can be used to detect when such changes occur.
NM_MT_DESC	=> Request NM name for a device or summary/sensor/control point. <= NM Reply contains Unicode description for the requested device/ point.	Useful for retrieving description for use in a system configuration dialog.
NM_MT_MATE_STAT	=> Request the status of the standby NM. <= NM Reply contains the status of the standby NM. • NM sends unsolicited in the event of a status change.	Used in redundant NM situations to monitor the availability of the backup NM.
NM_MT_STANDBY	=> Request Active NM transfer control to Standby NM.	Used to swap which NM is active in a redundant situation.
NM_MT_LRANGE_ALARMS	=> Request the linear distance linear range coverage of the device's alarm points. <= NM Reply contains the linear ranges of configured ranging points.	Used to identify which points support ranging and the extent of linear locations that can be expected.
NM_MT_GPS_PATH_ALARMS	=> Request the GPS path points of the device's alarm points. <= NM Reply contains the GPS path points of configured ranging points.	Used to identify which points support GPS alarm location and the path along which the GPS location will be reported.