



LONMARK[®]

Functional Profile:

Node Object

SFPTnodeObject

Overview

The Node Object functional profile describes a special type of functional block—called the *Node Object functional block*—that is used by network tools to test and manage all the functional blocks on a device. Figure 1 illustrates a typical device with a Node Object functional block and three other functional blocks. In this example, the Node Object functional block can be used to independently manage functional blocks 1, 2, and 3.

The Node Object functional block may also be used to set the time for the device, report alarms generated by the device, document the location of the device, and manage schedules within the device. Manufacturers may add other device-level functions to the Node Object functional block by implementing manufacturer-specific functional block members (network variables and configuration properties).

The Node Object functional block includes a mandatory **nviRequest** input network variable and a mandatory **nvoStatus** output network variable. Other devices and applications may request a Node Object function by sending a request to the **nviRequest** network variable. Upon receiving an update to the **nviRequest** network variable, the request is processed and the **nvoStatus** network variable is updated with either the results of the request, an in-process indication, or an error indication. If included in the device, the optional **nvoAlarm** and **nvoAlarm2** network variables may be updated also. In addition, the **nvoAlarm** and **nvoAlarm2** network variables report alarm conditions as they occur. The definition of the **nviRequest** network variable includes an object ID field to allow the Node Object to report status and alarm conditions for all functional blocks on a device.

Only one Node Object functional block is allowed on a device, and it must be functional-block index zero if it is implemented on a device.

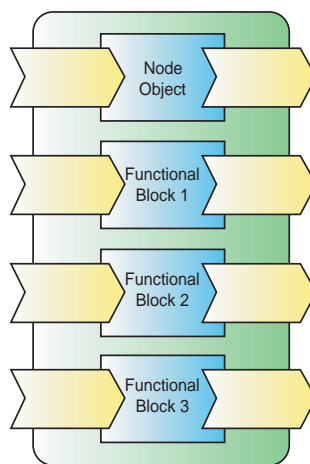


Figure 1 Device Concept

Functional-Block Details

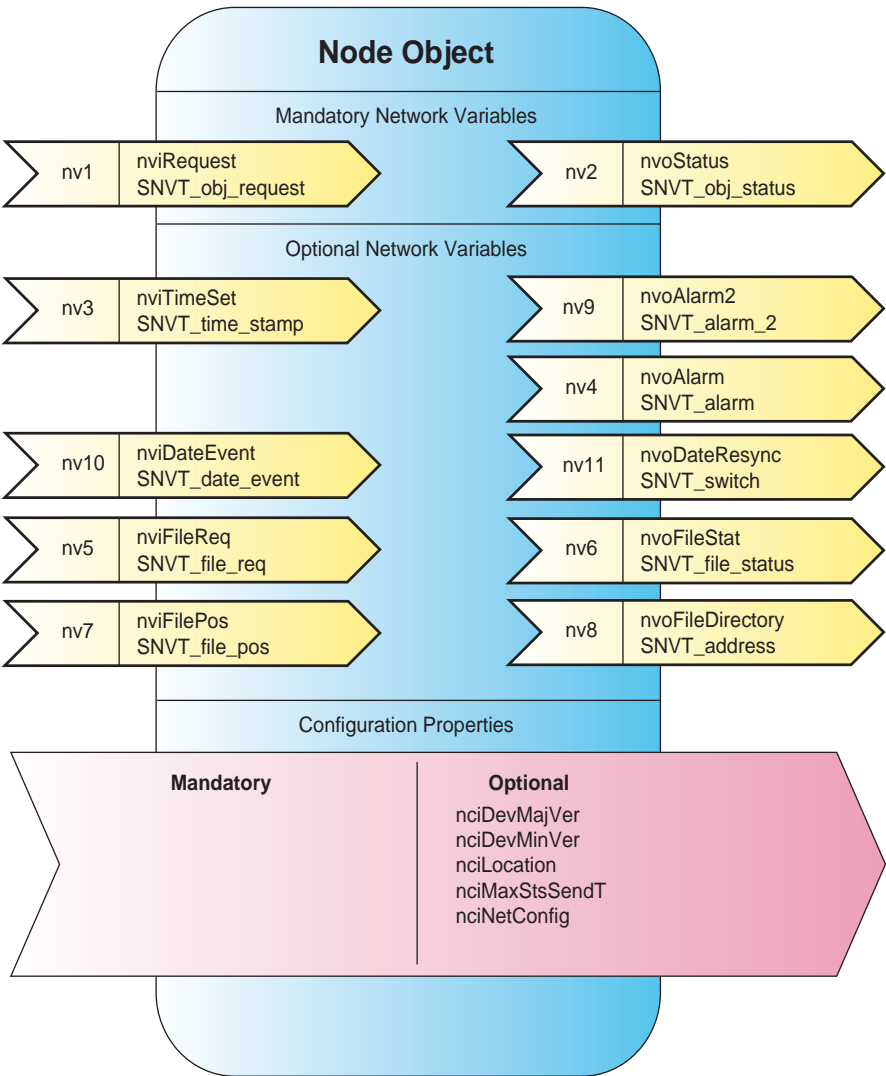


Figure 2 Functional-Block Details

Table 1 Network Variable Details

NV # (M/O)*	Variable Name	SNVT Name	SNVT Index	Description
1 (M)	nviRequest	SNVT_obj_request	92	Requests a particular mode for a particular functional block in the device
2 (M)	nvoStatus	SNVT_obj_status	93	Reports the status of the requested functional block in the device
3 (O)	nviTimeSet	SNVT_time_stamp	84	Synchronize the device's internal real time clock with an external time source
4 (O)	nvoAlarm <i>(use nvoAlarm2 for new device designs)</i>	SNVT_alarm	88	Transmits alarm data for each functional block on a device whenever an alarm occurs or is cleared, and upon request. It is superseded by nvoAlarm2 for new designs
5 (O)	nviFileReq	SNVT_file_req	73	Requests an operation on a particular file
6 (O)	nvoFileStat	SNVT_file_status	74	Reports the status of the last requested file operation
7 (O)	nviFilePos	SNVT_file_pos	90	Controls the position of the read/write pointer in a file
8 (O)	nvoFileDirectory	SNVT_address	114	Address for the file directory containing descriptors for configuration files
9 (O)	nvoAlarm2	SNVT_alarm_2	164	Transmits alarm data for each functional block on a device whenever an alarm occurs or is cleared, and upon request. Replaces nvoAlarm
10 (O)	nviDateEvent	SNVT_date_event	176	Reports the status of a schedule. Optional input for devices with Scheduler functional blocks
11 (O)	nvoDateResync	SNVT_switch	95	Requests an update for all defined exceptions via the nviDateEvent input. Optional output for devices with Scheduler functional blocks

* M = mandatory, O = optional

Table 2 Configuration Property Details

Man. Opt. *	SCPT Name NV Name Type or SNVT	SCPT Index	Associated NVs **	Description
Opt	SCPTnwrkCnfg nciNetConfig SNVT_config_src (69)	25	Entire Functional Block	Controls the maximum period of time before the object status is transmitted. Zero means disabled
Opt	SCPTmaxSndT nciMaxStsSendT SNVT_elapsed_tm (87)	22	nv2 (O)	Controls the maximum period of time before the object status is transmitted. Zero means disabled
Opt	SCPTlocation nciLocation SNVT_str_asc (36)	17	Entire Functional Block	Provides descriptive physical location information related to the entire device.
Opt	SCPTdevMajVer nciDevMajVer unsigned short	165	Entire Functional Block	The major version number for the device
Opt	SCPTdevMinVer nciDevMinVer unsigned short	166	Entire Functional Block	The minor version number for the device

* “Man” = mandatory, “Opt” = optional.

It should be Mandatory for CPs that are Mandatory for an NV that is also Mandatory. This is also valuable for CPs that apply to the Entire Functional Block.

** List of NVs to which this configuration property applies.

An “(M)” means that the CP is Mandatory if the NV (to which it applies) is implemented. An “(O)” means that the CP is Optional if the NV (to which it applies) is implemented.

Mandatory Network Variables

Object Request

```
network input SNVT_obj_request nviRequest;
```

This input network variable provides the mechanism to request an operation or a mode for a functional block within a device. For a listing of all possible request codes, and for the meaning of the function codes for **SNVT_obj_request**, see the *SNVT Master List*.

A request consists of an object ID (the **object_id** field) and an object request (the **object_request** field). The object ID is the functional block index for a functional block on the device. If a device has a Node Object functional block, its functional block index must be zero. The remaining functional blocks are numbered sequentially, starting with one.

The object request specifies a request function for the functional block identified by the object ID. The **object_request_t** definition in the *SNVT Master List* defines the available request functions; the following requests are the only mandatory request functions:

- ❑ **RQ_NORMAL**
- ❑ **RQ_UPDATE_STATUS**
- ❑ **RQ_REPORT_MASK**

If an **nviRequest** update specifies an unsupported request function, the **nvoStatus** output network variable must be updated with the **invalid_request** field set to one. Support for the object-disable, self-test, override, and alarm-reporting request functions is not required.

The request functions are defined as follows:

RQ_NORMAL (mandatory support). If the specified functional block was in the disabled or overridden state, this request cancels that state, and returns the functional block to normal operation. If the functional block was already in the normal state, a request to enter the normal state is not an error. After device reset, the state of functional blocks on the device is application-specific.

An **RQ_NORMAL** request that specifies the Node Object functional block index is a request for all functional blocks in the device to leave the disabled and overridden states.

RQ_UPDATE_STATUS (mandatory support). Requests the status of the specified functional block to be sent to the **nvoStatus** output network variable. The state of the functional block is unchanged.

An **RQ_UPDATE_STATUS** request that specifies the Node Object functional block is a request for the status of the device and all functional blocks on the device. The status bits of the Node Object (with the exception of **invalid_request** and **invalid_id**) are defined to be the inclusive-OR of the status bits of all the other functional blocks in the device; with the possible addition of error conditions and other conditions attributed to the device as a whole, rather than to any individual functional block. For example, if **comm_failure** is supported for the Node Object, then it should be set when reporting the Node Object functional block status whenever any of the functional blocks in the device reports communications failure, as well as when there is a communications failure at the device level.

RQ_REPORT_MASK (mandatory support). Requests a *status mask* reporting the status bits that are supported by the specified functional block to be sent to the **nvoStatus** output network variable. A one bit in the status mask means that the device may set the corresponding bit in the object status when the condition defined for that bit occurs. A zero bit in the status mask means that the bit is never set by the device. For example, if object disable (**RQ_DISABLED**) is not supported for a functional block, the **disabled** bit in the status mask must be zero for that functional block. If self-test (**RQ_SELF_TEST**) is not supported for a functional block, the **fail_self_test** and **self_test_in_progress** bits in the status mask must be zero for that functional block. If alarm reporting (**RQ_UPDATE_ALARM** or asynchronous notification) is not supported, the **in_alarm** bit in the status mask must be zero for that functional block.

An **RQ_REPORT_MASK** request that specifies the Node Object functional block requests a status mask that is the inclusive-OR of supported status bits for the device and all functional blocks on the device.

RQ_CLEAR_STATUS. Requests all status and report bits for the specified functional block and in the **nvoStatus** output network variable to be cleared. The state of the functional block is unchanged.

An **RQ_CLEAR_STATUS** request that specifies the Node Object functional block clears the status and report bits for all functional blocks on the device in addition to the **nvoStatus** output network variable.

RQ_DISABLED. Requests the specified functional block to change to the disabled state. In the disabled state, output network variables belonging to the functional block are not propagated to the network. However, it must be possible to poll the output network variables of a functional block in this state. In the disabled state, the functional block must not respond to any updates received on its input network variables, but it must support reading and writing of any configuration properties belonging to the functional block. If the functional block was already in the disabled state, a request to disable the functional block is not an error. A functional block may be in both the overridden and disabled states at the same time. In this case, output network variables are set to their override values, but the values are only propagated if they are polled.

An **RQ_DISABLED** request that specifies the Node Object functional block is a request to disable all functional blocks in the device that support the disable function, including the Node Object functional block. If any of the functional blocks in the device support the disable function, then those functional blocks are disabled and **invalid_request** is not reported. However, if none of the functional blocks in the device support the disable function, then **invalid_request** is reported.

If the Node Object functional block is disabled, the LONWORKS[®] File Transfer Protocol, direct memory read/write, and any other request to the Node Object functional block, are not disabled. In addition, status and alarm reporting via the **nvoStatus**, **nvoAlarm**, and **nvoAlarm2** outputs is not disabled when the Node Object functional block is disabled.

RQ_ENABLE. Requests the specified functional block to change to the enabled state without modifying any overridden behavior. In the enabled state, output network variables belonging to the functional block are propagated to the network as defined by the functional block. If a functional block is both overridden and enabled, the functional block outputs are set to their overridden values, but are propagated normally.

An **RQ_ENABLE** request that specifies the Node Object functional block enables only the Node Object functional block and does not enable any other functional blocks on the device.

RQ_SELF_TEST. Requests the specified functional block to execute a self-test and report the results. If the self-test can be completed in the same critical section as the receipt of the request, the **fail_self_test** bit in the **nvoStatus** output network variable is set appropriately, and the **self_test_in_progress** bit remains zero. Otherwise, the **self_test_in_progress** bit is set in the critical section in which the self-test request is received. When the test is complete, the **fail_self_test** bit is set appropriately, and the **self_test_in_progress** bit is cleared.

An **RQ_SELF_TEST** request that specifies the Node Object functional block is a request to initiate a device-level diagnostic self-test. The manufacturer defines the testing to be performed. It may include a self-test of each functional block in the device, but this is not required.

RQ_OVERRIDE. Requests the specified functional block to change to the override state. Default values for an override state are usually configuration properties of individual network variables or functional blocks. They are often (but not exclusively) specified by **SCPTovrBehave**, **SCPTdefOutput**, and **SCPTovrValue** configuration properties. After device reset, the state of functional blocks on the device is application-specific. A functional block may be in both the override and disabled states at the same time. In this case, output network variables are set to their override values, but the values are only propagated if they are polled.

An **RQ_OVERRIDE** request that specifies the Node Object functional block is a request to override all functional blocks in the device that implement the override state. An **RQ_OVERRIDE** request that specifies the Node Object functional block does not override the LONWORKS File Transfer Protocol, direct memory read/write, or any other request to the Node Object functional block. The **nvoStatus** output is not overridden when the Node Object functional block is overridden.

RQ_RMV_OVERRIDE. Requests an override to be removed for the specified functional block without modifying the enabled state.

An **RQ_RMV_OVERRIDE** request that specifies the Node Object functional block is a request to remove the override for all functional blocks in the device that implement the override state.

RQ_UPDATE_ALARM. Requests the alarm status for the specified functional block to be sent to the **nvoAlarm** and **nvoAlarm2** outputs, if present. The state of the functional block, and of any alarms, is unchanged. See the **nvoAlarm** and **nvoAlarm2** output network variable description for details on how alarms are reported.

An **RQ_UPDATE_ALARM** request that specifies the Node Object functional block requests the alarm status for the highest priority device-level alarm for the **nvoAlarm** output, and requests all active alarms for the **nvoAlarm2** output. An active alarm is an alarm that has an **alarm_type** value that is not equal to **AL_NO_CONDITION**. If the **nvoAlarm** output is implemented, and the **nvoAlarm2** output is not implemented, and the device does not have any device-level alarms, then an **RQ_UPDATE_ALARM** request that specifies the Node Object functional block is invalid. If the **nvoAlarm2** output is implemented, and multiple alarms are active when an **RQ_UPDATE_ALARM** request that specifies the Node Object functional block is received, then the **nvoAlarm2** output is updated once with a Header alarm (**alarm_type** set to **AL_HEADER**), then updated once per active device-level alarm, then updated once per active functional block alarm, and then updated once with a Footer alarm (**alarm_type** set to **AL_FOOTER**). The Header and Footer alarms are not sent if only a single alarm is active, or if no alarms are active. If no alarms are active, a single alarm update is sent with **alarm_type** set to **AL_NO_CONDITION**. Device-level alarms may be total/service-interval alarms, for example. See the *SNVT Master List* for more details of **SNVT_alarm** and **SNVT_alarm_2**.

RQ_CLEAR_ALARM. Requests the alarm state of the functional block to be cleared, and the **nvoAlarm** and **nvoAlarm2** outputs to be updated to indicate no alarms for the functional block. The state of the functional block is unchanged. If any alarm conditions are still present for the functional block, the alarms are reported again as they are detected. If any alarms are active immediately following an **RQ_CLEAR_ALARM** request, the active alarms are reported as described for the **RQ_UPDATE_ALARM** request.

RQ_CLEAR_RESET. Requests the **reset_complete** flag in the **nvoStatus** output network variable of the corresponding functional block to be cleared. Further requests have no effect, until the next Reset sequence has again been executed.

An **RQ_CLEAR_RESET** request that specifies the Node Object functional block clears the **reset_complete** flags for all functional blocks on the device.

RQ_RESET. Requests the specified functional block to execute a Reset sequence and report its completion in the **nvoStatus** output network variable. The **reset_complete** flag in the **nvoStatus** output network variable is set when the Reset sequence is complete. The flag must be cleared by an **RQ_CLEAR_RESET** request.

An **RQ_RESET** request that specifies the Node Object functional block is a request to execute a Reset sequence for the device and for all functional blocks on the device, and to report the completion in the **nvoStatus** output network variable.

Valid Range

The valid range is any value within the defined limits of **SNVT_obj_request**.

Default Value

The default value is undefined.

Configuration Considerations

None specified.

Object Status

```
network output SNVT_obj_status nvoStatus;
```

This output network variable reports the status for any functional block on a device. It is also used to report the status of the entire device and all functional blocks on the device.

A status update consists of an object ID (the **object_id** field) and multiple status fields. The object ID is the functional block index as described under **nviRequest**. If the object ID is zero, the status of the device itself and all functional blocks on the device is reported.

The status fields are one-bit bitfields. The only required status fields are the **report_mask**, **invalid_id**, and **invalid_request** fields; all other status fields are optional. If an alarm condition is active for a reported functional block, the **in_alarm** field is set to one if it is supported, and additional information on the alarm may be reported by the optional **nvoAlarm** and **nvoAlarm2** outputs. Following is a description of the required status fields. See the *SNVT Master List* for a description of the optional fields.

invalid_request Set to one if an unsupported request code (**RQ_xxx**) is received on the **nviRequest** input network variable.

invalid_id	Set to one if a request is received for a functional block index that is not defined in the device. No further checking of the request code is required when set to one.
report_mask	Set to one if an RQ_REPORT_MASK request is received by the nviRequest input network variable, and the nvoStatus output network variable is set to contain the status mask. The <i>status mask</i> is an nvoStatus value that describes the status bits that are supported beyond the three mandatory status bits. The status mask consists of all fields in the nvoStatus output network variable, with the exception of the report_mask , invalid_id , and invalid_request fields. A one bit in the mask means that the functional block may set the corresponding bit in the nvoStatus output network variable when the condition defined for that bit occurs. A zero bit means that the functional block may never set the bit.

Valid Range

The valid range is any value within the defined limits of **SNVT_obj_status**, with the exception that the **report_mask**, **invalid_id**, and **invalid_request** fields must be set to one.

Default Value

The default value must be the actual status of the device for all supported fields. All other fields must be set to zero. The application must update the status such that a polling of the status, following the request, returns a reasonable value. In a Neuron[®] C program, this means setting the status in the same critical section as the request event is processed. Setting the status may mean setting the actual result; or, for an **RQ_SELF_TEST** request, setting the **self_test_in_progress** field.

Configuration Considerations

The optional **nciMaxStsSendT** configuration property specifies a heartbeat for sending this network variable. If the CP is not implemented, or is implemented and is set to zero or the invalid value, a heartbeat is not provided.

When Transmitted

The output variable is transmitted when either of the following conditions occurs:

- ❑ A request is received by the **nviRequest** input network variable.
- ❑ The heartbeat interval specified by the optional **nciMaxStstSendT** CP expires.

When the heartbeat timer expires, the status of each functional block (including the Node Object functional block) is returned sequentially in round-robin fashion—one object status per expiration of the timer.

Default Service Type

The default service type is unspecified, but acknowledged service is recommended. It is also a recommendation that the network variable be polled.

Optional Network Variables

Time Setting

```
network input SNVT_time_stamp nviTimeSet;
```

This input network variable synchronizes the device's internal real-time clock with an external time source.

Valid Range

The valid range for all fields is any value within the defined limits of **SNVT_time_stamp**.

Default Value

The default value is the **SNVT_time_stamp** invalid value.

Configuration Considerations

None specified.

Alarm Output

```
network output sync SNVT_alarm_2 nvoAlarm2;
```

or, for legacy devices and legacy support:

```
network output SNVT_alarm nvoAlarm;
```

These output network variables transmit alarm data for the device, and each functional block on the device, to a monitoring device. A message containing all the data relating to the alarm condition is sent whenever an alarm condition occurs, or is cleared, and upon the functional block receiving an **RQ_UPDATE_ALARM** request via the **nviRequest** input network variable.

An nvoAlarm2 output must be used for all new device designs that require reporting of alarm reporting. If desired, an nvoAlarm output can be used in

addition to an **nvoAlarm2** output to provide an interface to legacy devices and applications.

An **nvoAlarm2** alarm value consists of an alarm type, priority, timestamp, sequence number, and description. The alarm type specifies the alarm condition, and is set to **AL_NO_CONDITION** if there is no active alarm. The timestamp supports a resolution of up to 1 millisecond and reports the time the alarm condition occurred if available, the time the alarm was first reported if the time of the condition is not available but time of report is, or contains an invalid value if time is not available. The sequence number starts at zero and is incremented by one for each **nvoAlarm2** update; it wraps to zero when it reaches 255. The sequence number can be used by a receiving device or application to determine if an alarm update has been missed.

The description consists of a text description of the alarm that can be easily interpreted by a user via a summary log, historical log, Web page, email message, or SMS message. Manufacturers may embed manufacturer-specific text error codes in the description. The description may include references to LONMARK resource files as described in the *SNVT Master List*.

The **nvoAlarm2** output can report multiple alarms per functional block. If multiple alarms occur or an **RQ_UPDATE_ALARM** request is received for a functional block with multiple active alarms, the **nvoAlarm2** output is updated once with a Header alarm (**alarm_type** set to **AL_HEADER**), then updated once per active alarm, and then updated once with a Footer alarm (**alarm_type** set to **AL_FOOTER**). The Header and Footer alarms are not sent if only a single alarm is active, or if no alarms are active.

The **nvoAlarm** output can only report a single alarm per functional block. If multiple alarms occur, or an **RQ_UPDATE_ALARM** request is received for a functional block with multiple active alarms, only the highest-priority alarm is reported.

Valid Range

The valid range of **SNVT_alarm_2** and/or **SNVT_alarm**.

Default Value

The default value is unspecified.

Configuration Considerations

A heartbeat may be specified for sending this network variable using an optional **nciMaxStsSendT** configuration property, or a manufacturer-specific **SCPTmaxSendTime** or **SCPTmaxSndT** CP. If specified, the heartbeat CP must apply to the **nvoAlarm** and/or **nvoAlarm2** outputs. If a heartbeat CP is defined and has a value of “0 0:0:0:0” or other invalid value, a heartbeat is not used.

When Transmitted

The output variable is transmitted when any of the following conditions occurs:

- ❑ An alarm condition occurs or changes. A change may be due to a change in an external condition or input, or due to a configuration change.
- ❑ An **RQ_UPDATE_ALARM** request is received by the **nviRequest** network variable. A single alarm value with an **alarm_type** field value of **AL_NO_CONDITION** is reported if there are no active alarms.
- ❑ The heartbeat interval, specified by an optional heartbeat CP, expires.

The **nvoAlarm2** output can report multiple alarms per functional block, so a single **RQ_UPDATE_ALARM** request may result in multiple updates to the **nvoAlarm2** output. The outputs may be throttled with a manufacturer-specific **SCPTminSendTime** or **SCPTminSndT** CP that applies to the **nvoAlarm2** output. If a throttle is not specified but a heartbeat is, the multiple outputs are sent at the heartbeat interval. If neither a throttle nor a heartbeat is defined, the multiple outputs are sent at a manufacturer-defined rate.

Default Service Type

If the **nvoAlarm2** output is to be updated at a rate of more than once per event-loop time interval, it must be declared as a synchronous output.

The default service type is unspecified, but acknowledged service is recommended.

File Request

```
network input SNVT_file_req nviFileReq;
```

This input network variable specifies a requested operation code and file index. When a request is received, the device performs the requested operation and returns the status of that operation in the **nvoFileStat** output network. The request operation codes are defined in the *SNVT Master List*, and described in the Echelon *LONWORKS File Transfer Protocol Engineering Bulletin* (005-0025-01). Files are identified with a unique 16-bit number called the *file index*. Up to 65 535 files can be identified on any device.

This input network variable must be implemented in the Node Object functional block if the device supports LONWORKS FTP. It must not be implemented if the device supports the direct memory read/write access method for data files.

Valid Range

The valid range of **SNVT_file_req**.

Default Value

The typical default value is **FR_NUL**.

Configuration Considerations

None specified.

File Status

```
network output SNVT_file_status nvoFileStat;
```

This output network variable transmits the status of the last file-request received via the **nviFileReq** input network variable. The returned status codes are defined in the *SNVT Master List*, and described in the Echelon *LONWORKS File Transfer Protocol Engineering Bulletin* (005-0025-01).

This output network variable must be implemented in the Node Object functional block if the device supports LONWORKS FTP. It must not be implemented if the device supports the direct memory read/write access method for data files.

Valid Range

The valid range of **SNVT_file_status**.

Default Value

The typical default value is **FS_NUL**.

Configuration Considerations

The Node Object implements the file-request and file-position network variables as inputs, and the file-status network variable as an output. The device can therefore act as the Sender or the Receiver in a file transfer, but it cannot act as the Initiator of a file transfer using these network variables.

When Transmitted

The output variable is transmitted when either of the following conditions occurs:

- ❑ During file transfer
- ❑ When polled

Default Service Type

The default service type is unspecified, but acknowledged service is recommended.

File Position

```
network input SNVT_file_pos nviFilePos;
```

This input network variable controls the position of the read/write pointer in a file used for random access, and is also used to specify the length of the next file transfer. The **rw_ptr** field is a 32-bit value compatible with the Neuron C **s32_type** signed 32-bit type. For more details, see the Echelon *LONWORKS File Transfer Protocol Engineering Bulletin* (005-0025-01).

This input network variable must be implemented in the Node Object functional block if the device supports the LONWORKS FTP with random and sequential access method. It must not be implemented if the device supports the LONWORKS FTP with sequential access or the direct memory read/write access methods for data files.

Valid Range

The valid range of **SNVT_file_pos**.

Default Value

None specified.

Configuration Considerations

None specified.

File Directory Address

```
network output SNVT_address nvoFileDirectory;
```

This output network variable reports the starting address of the configuration-file directory on a Neuron hosted device. It is used when configuration properties are implemented within configuration files accessed by ANSI/EIA/CEA-709.1 Read Memory and Write Memory network-management messages. If an **nvoFileDirectory** output network variable is implemented on a device, all files on the device must be accessible using network management read/write messages. For more details, see *Configuration Properties* within the *LONMARK Application-Layer Interoperability Guidelines*.

This output network variable must be implemented in the Node Object functional block if the device supports the LONWORKS FTP with random and sequential access method. It must not be implemented if the device supports the LONWORKS FTP with sequential access or the direct memory read/write access methods for data files.

Valid Range

The valid range for the file directory address is any value within the user-data memory space of a Neuron Chip or Smart Transceiver.

Default Value

The typical default value is **FS_NUL**.

Configuration Considerations

The Node Object implements the file-request and file-position network variables as inputs, and the file-status network variable as an output. The device can therefore act as the Sender or the Receiver in a file transfer, but it cannot act as the Initiator of a file transfer using these network variables.

When Transmitted

The output variable is transmitted when either of the following conditions occurs:

- ❑ During file transfer
- ❑ When polled

Default Service Type

The default service type is unspecified. Network tools may wish to poll this network variable for values.

Date Event

```
network input SNVT_date_event nviDateEvent;
```

This input network variable reports the status of a schedule. It is an optional Node Object input for devices with Scheduler functional blocks. It provides the status of each defined exception to the schedule.

A device implementing a Scheduler may have an **nviDateEvent** input and an **nvoDateResync** output on the Node Object functional block. If provided, the input shall be used to identify the status of all externally controlled schedules, and the output shall be used to request a resynchronization of all externally

controlled schedules; for example, following a power cycle on a scheduler device. Up to 256 schedules may be externally controlled. An externally controlled schedule input shall be ignored or deleted on the second midnight after its last update.

Valid Range

The valid range of **SNVT_date_event**.

Default Value

None specified.

Configuration Considerations

None specified.

Date Resynchronization Request

```
network output bind_info(ackd) SNVT_switch  
nvoDateResync;
```

This network variable requests an update for all defined exceptions to schedules via the **nviDateEvent** input. It is an optional output network variable for devices with Scheduler functional blocks.

A device implementing a Scheduler may have an **nviDateEvent** input and an **nvoDateResync** output on the Node Object functional block. If provided, the input shall be used to identify the status of all externally controlled schedules, and the output shall be used to request a resynchronization of all externally controlled schedules; for example, following a power cycle on a scheduler device. Up to 256 schedules may be externally controlled. An externally controlled schedule input shall be ignored or deleted on the second midnight after its last update.

Valid Range

The valid range of **SNVT_switch** as a discrete output using the “2-state interpretation.” For details, consult the *SNVT Master List*.

Default Value

The default value is “0.0 0” (value=0; state=0).

Configuration Considerations

None specified.

When Transmitted

This output variable may be transmitted at any time to request a resynchronization of all externally controlled schedules. This may be required after a power cycle or reset.

Default Service Type

The default service type is acknowledged.

Configuration Properties

Network Configuration Source (Optional)

```
SCPTnwrkCnfg cp_family nciNetConfig;
```

This configuration property sets the source for network configuration for a device. The source may be the device itself, using a process called *self-installation*, or an external network tool. All devices that support self-installation must provide this configuration property to allow a network tool to take control of the device's network configuration.

A self-installed device updates its own network-addressing information based on local inputs—with no interaction with other devices on the network during the installation process. In a typical self-installed system, the only information set at installation time is a domain number, group number, and possibly a network variable selector number. The rest of the installation information—including the majority of the binding information—is set at the time of manufacture. The user interface at each device is usually very simple; for example, push-button switches, DIP switches, rotary switches, or a backplane slot ID.

Valid Range

<i>Value</i>	<i>Identifier</i>	<i>Notes</i>
0	CFG_LOCAL	Device will use self-installation functions to set its own network address.
1	CFG_EXTERNAL	Device's network address will be set by an outside source. The device's application will not

		interfere with addresses assigned by external network tools. The device must be compatible with any valid ANSI/EIA/CEA-709.1 protocol address.
-1 (0xFF)	CFG_NUL	Value not available.

Default Value

For a self-installed device, the default value is **CFG_LOCAL**.

Configuration Requirements/Restrictions

This CP cannot be constant, nor can it be restricted to modification by the manufacturer only.

Maximum Send Time (Optional)

```
SCPTmaxSndT cp_family nciMaxStsSendT;
```

Also known as a *send heartbeat*, this configuration property sets the maximum period of time that can expire before the functional block automatically updates the **nvoStatus** output network variable. This configuration property may apply to additional output network variables such as the optional **nvoAlarm2** output, in which case the heartbeat applies to all applicable network variables.

Valid Range

Minimum is “0 0:0:0:0”

Maximum is “0 17:59:59:999” (0 days, 17 hours, 59 minutes, 59 seconds, 999 milliseconds).

The **day** field must remain zero for normal operation. A value of 65 535 in the **day** field is an invalid value, and means that the **nciMaxStsSendT** value should be ignored. This may imply that a hard-coded default value be used, but the actual device behavior is manufacturer-specific. The **nciMaxStsSendT** value should be interpreted as “0 0:0:0:0” if there are any other non-valid values in any of the fields. Any variant to this must be stated in the manufacturer’s documentation for the device.

Default Value

The default value is “0 0:0:0:0” (no automatic update).

Configuration Requirements/Restrictions

This CP has no modification restrictions. It can be modified at any time.

Location Label (Optional)

```
SCPTlocation cp_family nciLocation;
```

This configuration property identifies the subsystem containing the device. It provides a more detailed description of the device location than can be provided by the Neuron Chip's 6-byte location string. This allows network-recovery tools to recover subsystem information from a device. The subsystem may be a simple location name, or may be a hierarchical subsystem name. If a hierarchical subsystem name is specified, the subsystem hierarchy components must be separated by periods (`.`). For example, a device may have an **nciLocation** value of "Bldg 1.Floor 2.Rm 29", representing the Bldg 1/Floor 2/Rm 29 subsystem. Periods must not otherwise be used in the **nciLocation** value. Other characters that cannot be used in the **nciLocation** value are the backslash (`\`), colon (`:`), forward slash (`/`), or double-quotation (`"`) characters (however, stylistic quotation characters can be used, such as `„`, `“`, `”`, `«`, and `»`). For very large networks, subsystem numbers may be used instead of subsystem names, for example: "1.2.29". This allows deeply nested hierarchies to fit within the 31-character limit for **SCPTlocation**.

If a device has multiple locations, such as a device with multiple remote sensors, each of the functional blocks on the device may also implement a **SCPTlocation** configuration property to identify the location of each of the remote components. The **SCPTlocation** configuration property associated with the Node Object identifies the location of the device itself, whereas the other **SCPTlocation** configuration properties identify the locations of their respective hardware components.

If a device does not have a Node Object functional block, a **SCPTlocation** configuration property that applies to the device may be implemented to document the device location.

Valid Range

Any NUL-terminated ASCII string up to 31 bytes of total length (including NUL). The string must be truncated if the length does not allow the thirty-first character to be the NUL (0).

Default Value

The default value is an ASCII string containing 31 NULs (0).

Configuration Requirements/Restrictions

This CP has no modification restrictions. It can be modified at any time.

Device Major Version (Optional)

```
SCPTdevMajVer cp_family nciDevMajVer;
```

This configuration property provides the major version number of a device. If it is implemented on a device with a Node Object functional block, it must be a member of the Node Object functional block. If it is implemented on a device without a Node Object functional block, it must be implemented as a device configuration property.

The major version number must be incremented when the network interface for the device changes.

Valid Range

Any integer number from 0 to 255.

Default Value

The default value is zero.

Configuration Requirements/Restrictions

An **nciDevMajVer** configuration property must always specify the Constant flag. The Constant flag means that all devices with the same Standard Program ID (SPID) will have the same value, while the Device-Specific flag means that devices with an identical SPID may have different values for that configuration property (see **nciDevMinVer** for an example). This means that the major version is the same for all devices with the same program ID, but the minor version numbers may be different.

Device Minor Version (Optional)

```
SCPTdevMinVer cp_family nciDevMinVer;
```

This configuration property provides the minor version number of a device. If it is implemented on a device with a Node Object functional block, it must be a member of the Node Object functional block. If it is implemented on a device without a Node Object functional block, it must be implemented as a device configuration property.

The minor version number must be incremented when the network interface remains the same, but the device has a different behavior.

Valid Range

Any integer number from 0 to 255.

Default Value

The default value is zero.

Configuration Requirements/Restrictions

An **nciDevMinVer** configuration property must always specify the Device-Specific flag. The Constant flag means that all devices with the same Standard Program ID (SPID) will have the same value (see **nciDevMajVer** for an example), while the Device-Specific flag means that devices with an identical SPID may have different values for this configuration property. This means that the major version is the same for all devices with the same program ID, but the minor version numbers may be different.

Data Transfer

Devices may implement data files that can be transferred to other devices and applications using the LONWORKS File Transfer Protocol (FTP) with random and sequential access method, the LONWORKS FTP with sequential access method, or the direct memory read/write access method. All data files on a device must use the same access method. The following rules must be followed to enable clients to determine which access method is supported:

- ❑ If LONWORKS FTP with random and sequential access method is used, the **nviFileReq**, **nviFilePos**, and **nvoFileStat** network variables must be implemented in the Node Object functional block, and the **nvoFileDirectory** network variable must not be implemented in the Node Object functional block.
- ❑ If LONWORKS FTP with sequential access method is used, the **nviFileReq** and **nvoFileStat** network variables must be implemented in the Node Object functional block, and the **nvoFileDirectory** and **nviFilePos** network variables must not be implemented in the Node Object functional block.
- ❑ If direct memory read/write access method is used, the **nvoFileDirectory** network variable must be implemented in the Node Object functional block, and the **nviFileReq**, **nviFilePos**, and **nvoFileStat** must not be implemented in the Node Object functional block.

Power-up State

During device power-up, the Node Object functional block implementation should initialize the other functional blocks on the device by resetting/clearing the status of those functional blocks. The Node Object functional block should be enabled and unlocked before leaving the Reset tasks, but may optionally be locked (**nvoStatus.locked_out** set) during the Reset tasks. Locking other functional blocks during the Reset tasks is recommended.

Manufacturers must define whether or not alarm states are persistent across resets. If alarm states are not persistent, alarms may be cleared when a device is reset. Manufacturers should state the alarm-persistence behavior of a device in its user documentation.

Boundary and Error Conditions

Commands that are not understood by the device must cause the **nvoStatus.invalid_request** status bit to be set.

Additional Considerations

None specified.

Echelon, LON, Neuron, LONWORKS, LonTalk, LONMARK, and the LONMARK logo are trademarks of Echelon Corporation registered in the United States and other countries.