Version 1.00
February 2011
Static Programmable Device : 410

LONMARK®
Static Programmable Device

Overview

This document describes the Functional Profile of a LonMark Static Programmable Device.

Static Programmable Devices differ from Non-Programmable Devices in that their overall behavior is not hardcoded, but instead is defined by a downloadable program, which may be modified at runtime by the system integrator. The program is executed by a “Program” (SFPTstaticProgrammable) function block on the device, providing a standard and convenient way for system integrators to define the high-level logic, while management of low-level operations such as I/O, communications, memory, etc. is handled by the manufacturer-supplied firmware.

In this context, the program’s internal format is defined by the manufacturer. It may be script code, graphical programming code, a compiled binary, a configuration that connects predefined function blocks, or some other type of program.

Furthermore, Static Programmable Devices differ from Dynamic Programmable Devices in that their Network Interface is static – they do not allow dynamic creation of network variables or function blocks. The SPID (Standard Program ID) of a Static Programmable Device does not change when a new program is loaded, because the XIF is unchanged – the program is simply a new set of instructions to the Program function block, much like sending new configuration to a control loop.

At least one, standard LonMark mechanism, as defined in “Data Transfer” for transferring the program to the device, shall be available.

The demand for programmable devices comes from the need for greater flexibility and adaptability of applications, where domain experts can supply the operational logic they require in each specific case. In order to allow for both standardization and innovation, this functional profile focuses only on the externally visible attributes of Static Programmable Devices, while placing minimal restrictions on the internal, manufacturer-specific features and capabilities.

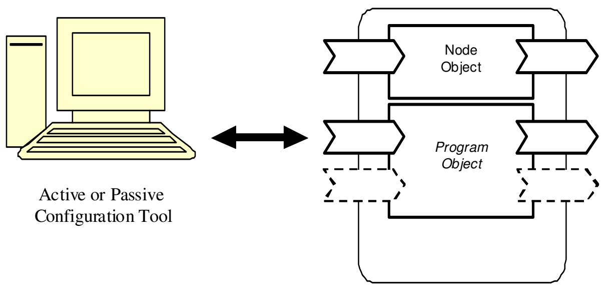


Figure 1 Device Concept

Example Usage

A Static Programmable Device is used for flexible solution programming in an Open Systems installation to support different application cases that cannot be covered by configurable devices. Examples of this would be unusual HVAC equipment configurations, custom control algorithms, and integration with other control systems such as lighting, security, or power management.

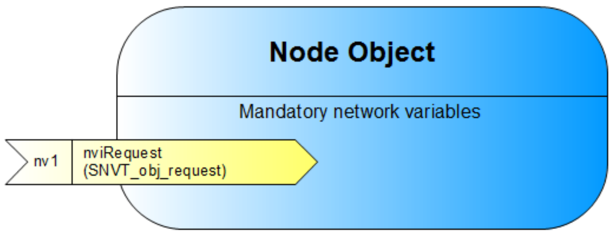
Requirements

The following list summarizes the requirements for the Static Programmable Device.

1. A Static Programmable Device shall contain at least one Program (SFPTstaticProgrammable) function block.
2. A Static Programmable Device shall have a Usage Field of either 3F or BF in its Standard Program Identifier (SPID), where the latter option is 3F logically OR'd with the Changeable Types bit (the most-significant bit of the Usage Field), irrespective of Device Class used.
3. A Static Programmable Device shall have a Device Class in its Standard Program Identifier (SPID) that represents the primary, marketed function of the device:
    a. If the primary, marketed function of the device is that of being programmable, then the Device Class of the SFPTstaticProgrammable profile (04:0A) shall be used;
    b. If the primary, marketed function of the device is that of being static, application-specific, and profiled via a LonMark Functional Profile, then the a Device Class of the primary application-specific LonMark Functional Profile shall be used;
    c. If the primary, marketed function of the device is not represented by a LonMark Functional Profile (*i.e.*, it is desired to use a Device Class that has no association to a LonMark Functional Profile), then the Device Class that is unassociated to a LonMark Functional Profile may be used;
4. A Static Programmable Device shall provide a mechanism for loading a new program into the device, to be used by the Program function block.
5. A Static Programmable Device that is running a program when a power failure occurs shall resume or restart its program automatically when power is restored, without requiring user intervention. See "Power-up State" for details.

6. A Static Programmable Device can be integrated into an open LonMark system, and its operation can be modified at runtime without losing its current configuration, NV type-changing or binding information.
7. The Program function block may support the following commands from a network tool:
    a. Load program
    b. Run program
    c. Halt program
    d. Restart program
    e. Unload program
    f. Step program
    g. (Optional manufacturer-specific commands)
8. The Program function block shall expose the following information about the state of the Program function block:
    a. Loaded program info (name, revision, date)
    b. Program status (Idle, Loading, Running, Halted, Unloading)
    c. Runtime (how long the program has been running, in hours)
    d. Most recent error, with time stamp

Object Details



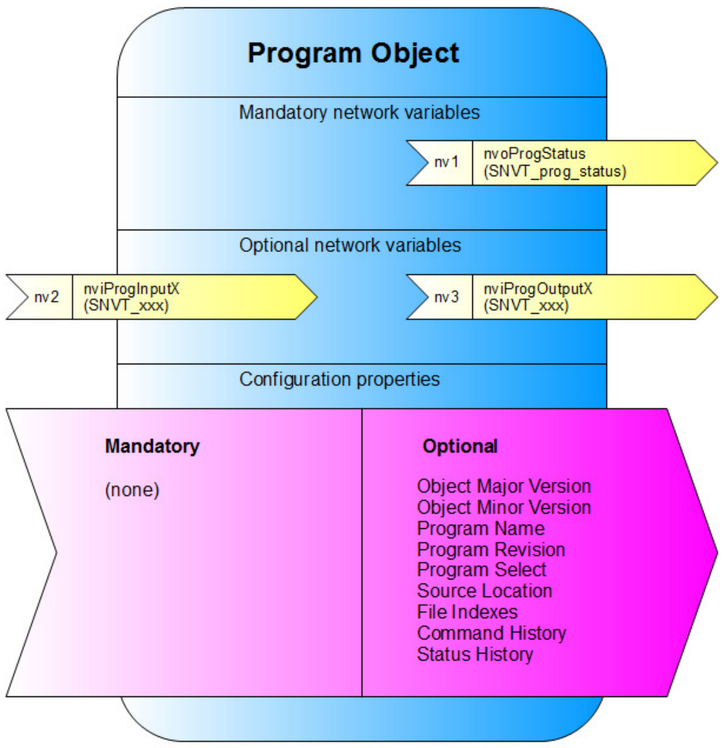


Figure 2 Profile Details

Table 1 Network Variable Members

NV # (M/O)*	Variable Name	SNVT Name	SNVT Index	Description
1 (M)	nvoProgStatus	SNVT_prog_status	202	Indicates the current status info for the Program function block, including state, runtime total, and error code with time stamp.
2 (M)	nviRequest (on Node Object)	SNVT_obj_request	92	Used to send commands to the Program function block.
** (O)	nviProgInput*x*	SNVT_xxx	Any	Network variables available for use as inputs to the running program.
** (O)	nvoProgOutput*x*	SNVT_xxx	Any	Network variables available for use as outputs from the running program.

Table 2 Configuration Property Members

Man. Opt. *	SCPT Name	SCPT Index	Description
Man ***	Network Variable Type (SCPTnvType)	254	Network-variable type for network variables that support changeable types
Man	Program Name (SCPTprogName)	351	Name of currently loaded program
Man	Program Revision (SCPTprogRevision)	352	Revision number and date of currently loaded program
Opt	Major Version (SCPTobjMajVer)	167	Program function block major version number
Opt	Minor Version (SCPTobjMinVer)	168	Program function block minor version number
Opt	Program Select (SCPTprogSelect)	353	Buffer Id where the currently loaded program is stored
Opt	Source Location (SCPTprogSourceLocation)	354	Location from where the current program was downloaded
Opt	File Indexes (SCPTprogFileIndexes)	355	Indexes of first and last LonMark files where programs may be stored
Opt	Command History (SCPTprogCmdHistory)	356	Log of recent commands, with time stamp
Opt	State History (SCPTprogStateHistory)	357	Log of recent state values, with time stamp
Opt	Error History (SCPTprogErrorHistory)	363	Log of recent errors, with time stamp
Opt	Index of Function Block (SCPTnsdsFbIndex)	358	The Function Block index within the Node Self-Documentation String to which this program applies (to which SFPTstaticProgrammable instance)
Opt [a]	NV Usage (SCPTnvUsage)	364	The SCPTnvUsage CPs shall be used to indicate whether the NVs are in use by the loaded program

* M/Man = mandatory, O/Opt = optional

** Manufacturer-Added Inputs/Outputs; programmable and selectable by Installer at time of programming.

*** Manufacturer-Added CPs to denote the present network-variable type of any declared nviProgInput*x* and/or nvoProgOutput*x* NVs.

[a] Manufacturer-Added CPs to denote if any declared nviProgInput*x* and/or nvoProgOutput*x* NVs are in-use at the time of programming or shipment.

Mandatory Network Variables

Program Status

```
network output SNVT_prog_status nvoProgStatus;
```

This input network variable is used to indicate the current status of the Program function block.

This is a structured network-variable type, and contains the following fields:

- Program State {program_state_t state}
- Program Runtime {SNVT_elapsed_tm runtime}
- Last Error {program_status_error_t last_error}
- Time Stamp of Last Error {SNVT_time_stamp time_of_last_error}

Program State is an 8-bit enumerated value, with the following values:

0. No Program
1. Idle (ready to run)
2. Loading (program is being loaded – will become Idle when done)
3. Running (may be halted by user, or halted if error occurs)
4. Halted (program has stopped due to an error or user command)
5. Unloading (program is being unloaded – will become "No Program" when done)

A "null" state is also defined, with a value of -1 (0xFF), indicating the state is "Unknown").

Program Runtime indicates how long the current program has been running.

1. If a program is executing (status = running), the runtime value will count up.
2. If a program is halted manually by a user, the runtime value will remain at its last value when the halt occurred. It will continue counting up again from the halted value if and when the program resumes.
3. If a program halts due to an error, or is unloaded, the runtime value will remain at its last value when the halt or unload occurred.
4. When a program is restarted from the beginning after a halt, or a program is loaded, the runtime value will reset to zero. If the program was loaded, the runtime value will start counting up from zero at the commencement of the program.

See "Power-up State" for further Program Runtime details.

Last Error indicates to the user the most recent error. This may be an error that halted the program, or it may be that the program was able to continue. A value of zero means "No Error", and is the expected value for this field during normal, error-free operation.

Several standard error codes are defined for **Last Error**, but there is also room for vendor-specific codes as well. The first (*i.e.*, most-significant) bit of the error code is unused, in order to avoid sign issues with enumerated types. The second bit divides the range of error codes into standard codes (bit=0) and vendor-specific codes (bit=1). The third bit indicates whether the error should halt the program or not – a value of 1 indicates a halting error. See the list of codes below for details.

Time Stamp of Last Error indicates when the **Last Error** occurred. This may be an actual time stamp, or a runtime time stamp. If a clock is available on the device – that is, either a real-time clock, or a soft-clock based on a CPU timer and an external synchronization mechanism – the actual date and time of the error is reported in this field. If no such clock is available, the time stamp field will show elapsed time, starting at Jan 1 year 0000 at start-up. If the value of the 'year' field is less than 2000, the time stamp shall be considered to be a runtime time stamp.

Note that a manual Halt command will not change the value of the **Last Error**, nor will a subsequent Run (resume) command. However, the program will only resume if the **Last Error** code is a non-halting one.

The following error codes are defined:

------ Standard, non-halting error codes ------

0. PSE_NO_ERROR
1. PSE_PROGRAM_FAULT_NOHALT
2. PSE_INVALID_OPERATION_NOHALT
3. PSE_INVALID_PARAMETER_NOHALT
4. PSE_STACK_OVERFLOW_NOHALT
5. PSE_STACK_UNDERFLOW_NOHALT
6. PSE_INSUFFICIENT_MEMORY_NOHALT
7. PSE_WATCHDOG_NOHALT
8. PSE_ *(reserved for future use)*

:

30. PSE_ *(reserved for future use)*
31. PSE_UNKNOWN_ERROR_NOHALT

------ Standard, halting error codes ------

32. PSE_LOAD_ERROR_HALT
33. PSE_PROGRAM_FAULT_HALT
34. PSE_INVALID_OPERATION_HALT
35. PSE_INVALID_PARAMETER_HALT
36. PSE_STACK_OVERFLOW_HALT

37. PSE_STACK_UNDERFLOW_HALT

38. PSE_INSUFFICIENT_MEMORY_HALT

39. PSE_WATCHDOG_HALT

40. PSE_CORRUPTED_PROGRAM_HALT (e.g. CRC failed)

41. PSE_ *(reserved for future use)*

:

62. PSE_ *(reserved for future use)*

63. PSE_UNKNOWN_ERROR_HALT

------ Manufacturer-defined, non-halting error codes ------

64. PSE_X_ (manufacturer defined)

:

95. PSE_X_ (manufacturer defined)

------ Manufacturer-defined, halting error codes ------

96. PSE_X_ (manufacturer defined)

:

127. PSE_X_ (manufacturer defined)

Valid Range

The valid range for the program state field is any value within the defined limits of the enumeration. Runtime may be any valid SNVT_elapsed_tm value. The error code may be 0-127, and the time stamp has the same range of values as SNVT_time_stamp. As noted above, if no real-time clock or soft-clock is available on the device, an elapsed time may be used, starting at Jan 1, 0000 when the device starts up.

Default Value

The default value is zero for all fields.

Configuration Considerations

None specified.

When Transmitted

The output variable is transmitted when any of the fields change value, or the heartbeat timer expires (if the network variable has a SCPTmaxSendTime CP).

Default Service Type

None specified.

Optional Network Variables

Request (on Node Object)

```
network input SNVT_obj_request nviRequest;
```

This input network variable is declared on the Node Object but is used to command the Program function block. When the object index of a Program function block is specified, the results of the request (command) written to this variable will be indicated by the output variable nvoProgStatus, on the Program function block.

This NV is optional in this profile, in order to accommodate the simplest case for a programmable device, which will always run when the function block is enabled. This allows programmable devices to act like their application-specific counterparts, that is, to run their application whenever they are commissioned and online.

If nviRequest is not present, then the device must remain in a 'good state' while a program is loaded (as opposed to generating errors or faults) and the program must automatically start from the beginning and enter into the running state.

If nviRequest is present, then:

1. If the run-program command or the halt-program command is supported, then both these commands must be supported.
2. If the load-program command or the unload-program command is supported, then both these commands must be supported along with the run-program and halt-program commands.
3. If the load-program command and unload-program commands are supported and SCPTprogSelect CP is present, then the program must be unloaded (with the unload-program command) before another program is selected and must load (with the load-program command) the new program before attempting to run the program. Note that the program state will be "no program" after the unload procedure and must be loaded to go to the idle (ready-to-run state).
4. If the load-program command and unload-program commands are supported, the current program cannot be replaced nor modified nor reloaded by any program-load mechanism without first unloading it (with the unload-program command). Note that the program state will be "no program" after the unload procedure and must be loaded (with the load-program command) to go to the idle (ready-to-run state).
5. If the restart-program command is supported then the run-program and halt-program commands must also be supported.
6. If the step-program command is supported then the run-program and halt-program commands must also be supported.

The following additions have been made to the object_request_t enumeration in order to manage the Program function block:

RQ_LOAD_PROGRAM (19)	Load the program specified in SCPTprogSelect
RQ_RUN_PROGRAM (20)	Begin or resume the currently loaded program. If the program was halted manually, this command will cause it to resume running from the point it was halted.
RQ_HALT_PROGRAM (21)	Halt the currently loaded program. This will preserve the program state and a subsequent run-program command will resume the program from where it was halted.
RQ_RESTART_PROGRAM (22)	Restart the currently loaded program from the beginning. This command shall not be used for starting a newly loaded program that has yet to be run once.
RQ_UNLOAD_PROGRAM (23)	Unload the currently loaded program.
RQ_STEP_PROGRAM (24)	Executes the next logical operation (line, statement, instruction, logic block, etc.) of the currently loaded program. The program state must be “idle” or “halted” to accept this command, otherwise it will be ignored. The program returns to “halted” state after execution of this command.

The RQ_LOAD_PROGRAM command may be used to prepare the device to receive data from an external source, or it may be used to select a one of several programs already stored on the device. Note that the load command may require more information – specifically, what program to load, if the device can store multiple programs. A separate configuration property, Program Select, will be used if this feature is needed. Using a CP will allow the device to preserve the information across a power cycle, so the device can resume running the correct program automatically, and it will also allow the program to load and start the program automatically after a device is replaced, if the manufacturer chooses to implement this feature (*e.g.*, by storing programs as configuration properties).

In order to allow automatic restart of the device after a power cycle, the running state of the program will be copied to non-volatile memory when it changes. The device will use this information to determine whether or not to run the program on start-up - if it was running prior to the power cycle, it shall resume running after power is restored.

It is not necessary for a device to support all (or any) of these commands. However, if an unsupported command is received, the Program function block must indicate that receipt using the “invalid_request” field in the Node Object's nvoStatus variable.

Valid Range

The valid range for the object_id field is the index of any function block on the device. However, for programming commands, the object_id field must specify the index of a Program function block.

The valid range for the object_request field is the valid range of the object_request_t enumeration. Not all requests will be supported by the Program function block, however. Only the commands above will be assured to be supported, although other commands may be supported at the discretion of the manufacturer.

Default Value

Zero ("No Command").

Configuration Considerations

Program Select is used in cases of multiple programs residing on, or downloadable to, a device designed to house multiple programs at the same time.

Program Input / Program Output

The Program function block does not define any input or output variables other than the Program Status NV (described earlier). However, in many cases it will be desirable to implement input NVs and/or output NVs for the Program function block to use. There are two possible ways to handle this:

1) Manufacturers can implement the Standard SFPTstaticProgrammable and create a separate function block (with an FPT Key in the range of 20000 to 25000) to contain the NVs, or
2) Manufacturers can derive a non-standard UFPT from the SFPTstaticProgrammable (UFPTstaticProgrammable, retaining the 410 FPT Key), and add the NVs to that UFPT. If this second method is used, it is important to understand the implications this will have:
    - A manufacturer can only define one scope 3 UFPT derived from the Static Programmable SFPT. Any attempt to create a second Static Programmable UFPT at scope 3 will fail, due to an ID conflict. Therefore, if a scope 3 UFPT is defined, it should anticipate future needs and declare extra NVs and CPs accordingly; as it will not be possible to add more NVs afterward.
    - A similar issue exists for scope 4 – only one Static Programmable UFPT may be defined per Device Class.
    - Declaring the UFPT at scope 6 would allow for many different UFPTs to be defined, but it could become cumbersome to manage and maintain them all.

Although these network variables are not defined in the Static Programmable SFPT, they are shown in the Object Details diagram for illustrative purposes, as it is expected that this will be a common implementation. The specific purposes of these network variables will be determined by the programmer during installation, so they are cannot be described here. Therefore, they are represented in the Object Details diagram as **nviProgInput*x*** and **nvoProgOutput*x***, with the implication that they should be renamed by the programmer during installation, to indicate their purpose:

```
network input  changeable_type SNVT_xxx nviProgInputx;
network output changeable_type SNVT_xxx nvoProgOutputx;
```

Each of these network variables that are implemented in the Program function block may be of any default type, but they must be type-changeable, and must comply with the relevant LonMark Interoperability Guidelines (*e.g.*, 2.7.2.2 in version 3.4 *Application-Layer Guidelines*). Implementing a SCPTmaxNVLength CP for these NVs is not required, but doing so will allow the NVs to be type-changed safely in engineered mode, *i.e.* without the device actually connected.

Network Variables In Use:

Two methods are defined for reporting which NVs are in use by a program. Which method is used per NV is determined by the absence or presence of a SCPTnvUsage CP that applies to the NV. The SCPTnvUsage CP provides additional flexibility for shipping an initial program that is not reflected in the device interface (XIF) file, but is not required. Following is a description of the two methods:

1) If an nviProgInputx or nvoProgOutputx NV does not have a SCPTnvUsage CP that applies to it, the value of the type_category field of the mandatory SCPTnvType CP is used to determine both whether or not an NV is in use, and the type of the NV when it is in use.

    a. The SCPTnvType CP for an unused nviProgInputx and nvoProgOutputx NV shall have a value of NVT_CAT_INITIAL in the type_category field indicating that it is not in use.

    b. If an NV is in use, the SCPTnvType CP for nviProgInputx or nvoProgOutputx NV will have the actual category in the type_category field, and shall not be set to NVT_CAT_INITIAL.

    c. Likewise, if a device is shipped with a pre-loaded program, the initial SCPTnvType CP values in the device and the device interface (XIF) file must reflect the nviProgInputx and nvoProgOutputx NV usage and types for the initial program.

    d. Programming tools must change this field for all NVs used by the Program function block, to reflect the actual category of the NV type when a new program is loaded. The programming tool must ensure that the type_category for unused NVs be set to NVT_CAT_INITIAL.

2) If an nviProgInputx or nvoProgOutputx NV has a SCPTnvUsage CP that applies to it, the value of the in_use field of the SCPTnvUsage CP is used to determine whether or not the NV is in use, and the SCPTnvType CP is only used to determine the NV type.

a. If a device is shipped with a pre-loaded program, the initial SCPTnvType CP value in the device and the device interface (XIF) file must reflect the nviProgInputx and nvoProgOutputx NV types for any NVs in use by the initial program.

b. The usage is determined by the SCPTnvUsage CP.

c. If the SCPTnvUsage CP is not declared as device-specific, the initial SCPTnvUsage CP values in the device and the device interface (XIF) file must reflect the nviProgInputx and nvoProgOutputx NV usage for the initial program. However, if the SCPTnvUsage CP is declared as non-constant device-specific, its value can only be determined by reading the SCPTnvUsage CP value from the device, and the value in the XIF file may be different.

d. The programming tool must ensure that the in_use field be set to reflect whether or not an NV is in use, and must ensure that the type_category field of SCPTnvType be set to reflect the type of any NVs that are in use.

Manufacturers may also add static (non-changeable) NVs to the Program function block, provided they follow the guideline requirements for manufacturer-specific function-block members, and have considered the implications of this, as described earlier.

Note that the Program function block is not limited to reading and writing its own NVs – it may also exchange data with other function blocks on the device. For example, a program may need to read the current scheduled occupancy state, in order to decide whether to turn a digital output point on or off. This might be done by reading the state NV declared in a Schedule function block, and requesting an actuator function block to turn-on the specified output point. This allows the Program function block to be used together with other standard function blocks to provide flexible, industry-standard solutions.

Mandatory Configuration Properties

Network Variable Type (Mandatory)

An SCPTnvType CP for each nviProgInput*x* or nvoProgOutput*x* NV declared on the device, irrespective of whether the NVs are used or are not used in a given program.

Valid Range

The valid range specified by the SCPT definition.

Default Value

type_category shall be NVT_CAT_INITIAL or the actual category type of the default NV's type if a SCPTnvUsage CP is also used for that NV.

Configuration Requirements/Restrictions

This configuration property must not be read-only.

SCPT Reference

SCPTnvType (254)

Program Name (Mandatory)

This configuration property specifies the human-readable name of the currently loaded program. It is a read-only CP, and gets its value from the currently loaded program. It will have a maximum length of 31 bytes so that it will fit within a config network variable if desired. If no program is currently loaded, or the loaded program does not specify a name, the value will be blank (a null string). The program name together with the revision number should specify a unique identifier of the current program.

Valid Range

The valid range specified by the SCPT definition.

Default Value

Blank (a null string).

Configuration Requirements/Restrictions

This configuration property is read-only and specific to the actual device (constant & device-specific).

SCPT Reference

SCPTprogName (351)

Program Revision (Mandatory)

This configuration property specifies the revision number of the currently loaded program. It is a read-only CP, and gets its value from the currently loaded program. If no program is currently loaded, or the loaded program does not specify a revision number, the value will be the default value (all zeroes, and an invalid date). Together with the program name it specifies a unique identifier of the current program. The fields of this CP are:

1. Program Major Version (8-bit)
2. Program Minor Version (8-bit)
3. Program Build Number (16-bit)
4. Program Build Date (SNVT_time_stamp)

Valid Range

The valid range specified by the SCPT definition.

Default Value

Zeroes for all fields.

Configuration Requirements/Restrictions

This configuration property is read-only and specific to the actual device (constant & device-specific).

SCPT Reference

SCPTprogRevision (352)

Optional Configuration Properties

Object Major Version

This configuration property stores the major version of the Program function block. It may be used to identify compatibility of programs with the Program function block. For example, if support for new features and functions are added to the Program function block, the network tool could use this CP to detect that the Program function block is an older version that does not support the new features, and a firmware upgrade is required before this program may be downloaded. It is a read-only CP, and is set by the Program function block.

Valid Range

The valid range specified by the SCPT definition.

Default Value

The default value of the SCPT.

Configuration Requirements/Restrictions

This configuration property is read-only (constant).

SCPT Reference

SCPTobjMajVer (167)

Object Minor Version

This configuration property stores the minor version of the Program function block. It may be used to identify the version of the Program function block, so that a configuration tool can present relevant information to a user. The Minor Version of the object is used to distinguish version changes that do not affect compatibility. For example, if a Program function block has some minor behavioral changes such as range checking added, the tool could decide whether or not to warn the user about this potential issue. Note that any changes that would affect compatibility of programs would require a change to the Major version CP, described above.

Object Minor Version is a read-only CP, and is set by the Program function block.

Valid Range

The valid range specified by the SCPT definition.

Default Value

The default value of the SCPT.

Configuration Requirements/Restrictions

This configuration property is read-only and specific to the actual device (constant & device-specific).

SCPT Reference

SCPTobjMinVer (168)

Program Select (Optional)

This configuration property specifies which program is selected to be loaded, if the device can store multiple programs. For devices that can store only a single program, this CP is not useful. Since the downloading mechanism for programs may differ for each manufacturer, the specific meaning of this value is similarly left up to the manufacturer.

Program Select is a writable CP, and is written by the programming tool when selecting a program to run. Normally this CP will be set while the device state is “No Program”. However, if the state is something else, writing a new value to this CP will initiate a Halt, followed by an Unload. The device will then load the selected program and wait for a Run command.

Valid Range

The valid range specified by the SCPT definition.

Default Value

Zero.

Configuration Requirements/Restrictions

This CP must not be read-only.

SCPT Reference

SCPTprogSelect (353)

Source Location (Optional)

This configuration property specifies the location from where the source file was downloaded to the device. Its purpose is to provide a place for the programming tool to provide human-readable information that will help answer the question of “where can I get the source code for this program?” It is important to be able to identify from where the program came, so that a system integrator can verify it is the correct program, or to retrieve the source code for modification.

The information provided is free-format and up to the manufacturer to use as they see fit. It could include the computer name, the path, and the file name of the source code. It could also indicate the author, or any other useful information that will help to locate the proper source code. This CP may not be necessary if the program source code has some mechanism associating it with a device – for example, if source code is stored as a device-level extension record in an Echelon Corporation LNS® database. In this case, the LNS database name and extension record name could be specified instead. This is a read-only CP, and gets its value from the currently loaded program.

Valid Range

The valid range specified by the SCPT definition.

Default Value

Blank (a null string).

Configuration Requirements/Restrictions

This configuration property is read-only and specific to the actual device (constant & device-specific).

SCPT Reference

SCPTprogSourceLocation (354)

File Indexes (Optional)

This configuration property may be implemented if the program data are stored in one or more files on the device. The CP consists of two fields, the first field indicating the index of the first file where programs are stored, and the second field containing the index of the last file where programs are stored. All files within the specified range are to be used for programs. This allows an external tool to know which files within the device are available for programs, so they can be managed, archived, loaded, etc. For example, if the first field has a value of 4, and the second field has a value of 6, then the device has reserved files 4, 5, and 6 for use by the Program function block.

Note that this CP does not reflect whether there is actually any program data in a file, it simply identifies the file indexes as “to be used by the Program function block”. It is a constant CP, and gets its value from the device interface definition.

Valid Range

The valid range specified by the SCPT definition. The value of the last_file_index must always be greater than or equal to the value of the first_file_index. The first_file_index must be a value of 3 or greater because there will already be a Template file, a Read-only value file, and a Read-write value file, as files 0, 1, and 2.

Default Value

As defined by the device interface definition.

Configuration Requirements/Restrictions

This configuration property is read-only (constant). Its value is set at manufacture time.

See the “Data Transfer” section for more information on using files to store programs.

SCPT Reference

SCPTprogFileIndexes (355)

Command History (Optional)

This configuration property contains a read-only list of the most recent commands to the device, along with a time stamp for each. The purpose of this CP is to provide a short history of changes to the programmable device, so that any operational issues with the device can be tracked and associated with a specific version of the program, and assist system integrators in diagnosing the cause of such issues. It will show whether a program was modified after the system integrator’s work was completed and signed off. It will be declared as a read-only CP array, and will be updated by the firmware during operation. It will act as a circular buffer, so only a finite number of records will be available, as new events overwrite older ones.

The first field of this configuration property will be a SNVT_time_stamp, indicating when the command was received. If no real-time clock or soft-clock is available on the device, an elapsed time may be used, starting at January 01, 0000 when the device starts up.

The second field will be an object_request_t enumerated type. This will indicate what command was issued.

The third and final field will be a text string, containing information relevant to the command. For example, a load command should use this field to record the name/path of the program being loaded.

Valid Range

N/A.

Default Value

Blank (a null string and invalid time stamp).

Configuration Requirements/Restrictions

This configuration property is read-only and specific to the actual device (constant & device-specific). Its value is set by the device during operation.

SCPT Reference

SCPTprogCmdHistory (356)

State History (Optional)

This configuration property contains a read-only list of the most recent states of the device, along with a time stamp for each. The purpose of this CP is to provide a short history of the state of the Program function block, so that any operational issues with the device can be tracked and associated with a specific state of the program, and assist system integrators in diagnosing the cause of such issues. It will show whether a program was running or halted at the time a specific issue occurred. It will be declared as a read-only CP array, and will be updated by the firmware during operation. It will act as a circular buffer, so only a finite number of records will be available, as new events overwrite older ones.

The first field of this configuration property will be a SNVT_time_stamp, indicating when the state change occurred. If no real-time clock or soft-clock is available on the device, an elapsed time may be used, starting at January 01, 0000 when the device starts up.

The second field will be the state of the Program function block, as reported in the “Program State” field of the Program Status network variable.

Valid Range

N/A.

Default Value

Blank (a “null” state and invalid time stamp).

Configuration Requirements/Restrictions

This configuration property is read-only and specific to the actual device (constant & device-specific). Its value is set by the device during operation.

SCPT Reference

SCPTprogStateHistory (357)

Error History (Optional)

This configuration property contains a read-only list of the most recent errors that occurred on the device, along with a time stamp for each. The purpose of this CP is to provide a short history of the errors encountered by the Program function block, so that any operational issues with the device can be tracked and associated with a specific errors detected by the program, and assist system integrators in diagnosing the cause of such issues. It will be declared as a read-only CP array, and will be updated by the firmware during operation. It will act as a circular buffer, so only a finite number of records will be available, as new events overwrite older ones. This CP array is kept separate from the state history, so that repeating errors will not overwrite state or command history.

The first field of this configuration property will be a SNVT_time_stamp, indicating when the error occurred. If no real-time clock or soft-clock is available on the device, an elapsed time may be used, starting at January 01, 0000 when the device starts up.

The second field is the error code, as reported in the “Last Error” field of the Program Status network variable.

Valid Range

N/A.

Default Value

Blank (a “null” state and invalid time stamp).

Configuration Requirements/Restrictions

This configuration property is read-only and specific to the actual device (constant & device-specific). Its value is set by the device during operation.

SCPT Reference

SCPTprogErrorHistory (363)

Index of Function Block (Optional)

This configuration property sets the function-block index within the Node Self-Documentation String to which this program applies.

Valid Range

N/A.

Default Value

1

Configuration Requirements/Restrictions

No restriction

SCPT Reference

SCPTnsdsFbIndex (358)

Data Transfer

The mechanism for transferring a program to the device is left up to the device manufacturer. However, at least one standard LonMark mechanism must be available:

1. LONWORKS® File-Transfer Protocol (LW-FTP),
2. Direct Memory-Read/Write (DM-R/W), or
3. Configuration Properties:
    a. Configuration Network Variables (CPs by NCIs),
    b. CPs by LW-FTP, or
    c. CPs by DM-R/W.

If LW-FTP (1) or DM-R/W (2) is used, the program files shall be registered in the device's file directory using the appropriate file type of `FILE_PROGRAM` (type 4). Development tools generate an enumeration for file types and this enumeration is as follows:

```
typedef enum {
    FILE_NUL,          // -1 = Invalid
    FILE_VALUE,        //  1 = Value File
    FILE_TEMPLATE,     //  2 = Template File
    FILE_DATALOG,      //  3 = Data Log File
    FILE_PROGRAM,      //  4 = Application Program
                       File
} file_type_t;
```

If program files are writeable, then they must also be readable in order to allow backup and restoration of the program data.

Other transfer mechanisms may also be implemented, *e.g.*, explicit messaging via this control-networking protocol, Ethernet FTP, serial communication, and removable-memory devices.

Power-up State

When a programmable device powers up, it should resume execution of the loaded program. It is likely necessary for the Program function block to restart execution from the beginning, although resuming from the current location (before power loss) is also acceptable, at the discretion of the device manufacturer.

If the device resumes execution from its runtime location prior to power loss, its runtime timer shall also resume runtime from its halted state rather than from zero.

Boundary and Error Conditions

None specified.

Additional Considerations

None specified.