BRITISH STANDARD

BS EN 14908-5:2009

Open Data Communication in Building Automation, Controls and Building Management Implementation Guideline — Control Network Protocol

Part 5: Implementation

ICS 35.240.99; 91.140.01; 97.120

British Standards

National foreword

This British Standard is the UK implementation of EN 14908-5:2009.

The UK participation in its preparation was entrusted to Technical Committee RHE/16, Performance requirements for control systems.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 June 2009.

ISBN 978 0 580 56866 4

Amendments/corrigenda issued since publication

Date	Comments

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 14908-5

April 2009

ICS 35.240.99; 91.140.01

English Version

Open Data Communication in Building Automation, Controls and Building Management Implementation Guideline - Control Network Protocol - Part 5: Implementation

Réseau ouvert de communication de données pour l'automatisation, la régulation et la gestion technique du bâtiment - Protocole de réseau pour le bâtiment - Partie 5 : Implémentation

Firmenneutrale Datenkommunikation für die Gebäudeautomation und Gebäudemanagement - Gebäude Netzwerk Protokoll - Teil 5: Implementierung

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

Management Centre: Avenue Marnix 17, B-1000 Brussels

Ref. No. EN 14908-5:2009: E

Contents

Page

Foreword

This document (EN 14908-5:2009) has been prepared by Technical Committee CEN/TC 247 “Building automation and controls and building management”, the secretariat of which is held by SNV.

This European Standard shall be given the status of a national standard, either by publication of an identical text or by endorsement, at the latest by October 2009, and conflicting national standards shall be withdrawn at the latest by October 2009.

This standard is part of a series of standards for open data transmission in building automation, control and in building management systems. The content of this standard covers the data communications used for management, automation/control and field functions.

The EN 14908-5 is part of a series of European Standards under the title, *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol (CNP)*, which comprises the following parts:

Part 1: *Protocol Stack*

Part 2: *Twisted Pair Communication*

Part 3: *Power Line Channel Specification*

Part 4: *IP Communication*

Part 5*: Implementation*

Part 6*: Application elements*

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN [and/or CENELEC] shall not be held responsible for identifying any or all such patent rights.

According to the CEN/CENELEC Internal Regulations, the national standards organizations of the following countries are bound to implement this European Standard: Austria, Belgium, Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and the United Kingdom.

Introduction

This document specifies the Layered Implementation Guidelines (LIG) for the Control Network Protocol (CNP) Specification: EN 14908-1:2005. The CNP specification model is based on the ISO Open Systems Interconnection Reference Model. There are also important extensions to the 7-layer OSI Reference Model. Figure 1 shows the scope of this specification in reference to the CNP and companion specifications for handling various data-transport media at the lower ISO protocol layers. A dashed line is used to show that the scope of this document is not treated as redundant, compared with other specifications covering their respective layers but as a complement to those specifications in implementing them in an interoperable fashion.

In this document, the guidelines for implementing a device based on CNP are specified to increase the ability for devices to interoperate regardless of the installer or manufacturer of the devices. Anything outside this boundary is covered in other parts of the standard. Similar specifications exist for CNP data-transport media.

This standard has been prepared to provide mechanisms through which various vendors of building automation, control, and of building-management systems, may exchange information in a standardised way. It defines communication and internal-documentation requirements.

This standard is contributing to the general European policy for energy savings particularly in the field of the “Energy Performance of Building Directive” and the Construction Products Directive (ER No. 6 “Energy Economy and Heat Retention”).

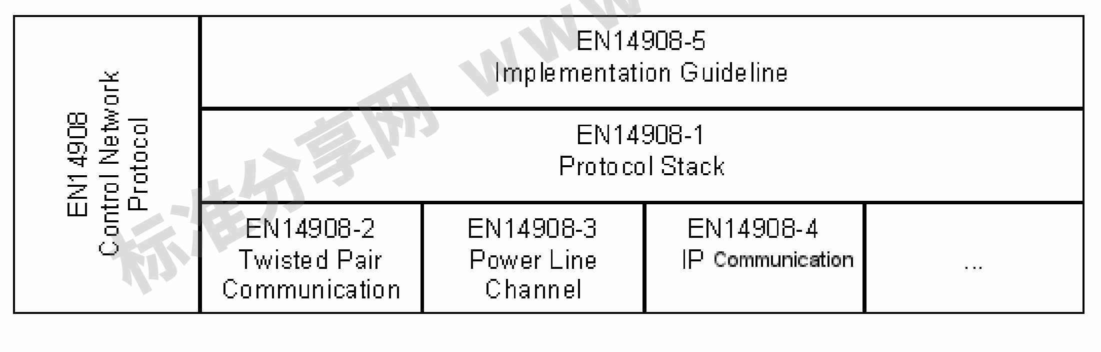

EN14908 Control Network Protocol
EN14908-5 Implementation Guideline
EN14908-1 Protocol Stack
EN14908-2 Twisted Pair Communication
EN14908-3 Power Line Channel
EN14908-4 IP Communication
...

Figure 1 — Scope of this specification

1 Scope

This specification provides mechanisms through which various vendors of networked control systems in commercial building automation, control, and building management may exchange information in a standardised way.

This specification contains all the information necessary to facilitate the exchange of data and control information in an interoperable fashion using EN 14908-1 and its associated data-transport media specifications.

This specification establishes a minimal set of rules for compliance. It does not rule-out extended services to be provided, given that the rules are adhered-to within the system. It is the intention of the standard to permit extended services to coexist and defines the bounds in which those services function, including the format for internal device-documentation of those services. Services outside purvey of this specification so long as they are adherents of the system are permitted but will not necessarily be interoperable with any other devices and shall not be essential for the functioning of the device.

Certain aspects of this standard are defined in other documents. These documents are referenced where relevant. In the case where a referenced standard conflicts with this document, this document will prevail.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

EN 14908-1:2005 *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 1: Protocol Stack*

prEN 14908-6*, Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 6: Application elements*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in EN 14908-1:2005 and the following apply.

3.1
application set
function block or function blocks to which a configuration property applies

EXAMPLE A network variable, a series or compilation of network variables, a functional block, a series or compilation of functional blocks, or the entire device.

3.2
base type
fundamental type that can be used as the basis of a network-variable type or configuration-property type

NOTE The available base types are defined in 5.1.2.2.

3.3
changeable-type network variable
network variable whose type can be changed during installation

NOTE See 4.7.3.3.

3.4
configuration property
CP
data value used to configure the application program in a device

NOTE Configuration properties are used to set parameters such as maximum, minimum, default, and override values. CPs are implemented using configuration network variables or as data items within configuration files. Configuration-property data are kept in a device's non-volatile memory.

3.5
configuration-property member
part of a functional profile

NOTE See 3.22.

3.6
configuration-property member number
part of a functional profile

NOTE See 3.23.

3.7
configuration-property type index
16-bit number that uniquely identifies a configuration-property type within the scope defined by the scope number and program-ID template of the resource file that contains the configuration-property type definition

3.8
device
logical and physical entity of the network containing an application that is designed to communicate with other logical and physical entities

3.9
device channel ID
number that optionally specifies the channel to which a device is attached

3.10
device class
two-byte field identifying the primary function of a device and part of the SPID of the device

3.11
device interface
network-visible interface to a device consisting of the unique node ID, program ID, channel ID, location field, device self-documentation string, device configuration properties, and functional blocks

3.12
device-location field
string or number that optionally specifies the location of a device

3.13
device self-documentation string
DSDS
string that specifies the structure of the contents of the self-documentation strings, the functional blocks, and optionally describes the function of a device

3.14
device subclass
two-byte field specifying the usage in the first byte and the channel type in the second byte and is part of the SPID of a device

NOTE See the usage and channel-type definitions.

3.15
dynamic functional block
functional block that is added to a device by a network tool after the device is installed

3.16
dynamic network variable
network variable that is added to a device by a network tool after the device is installed

3.17
format
<program ID> four-bit value defining the structure of the program ID as being a Standard Program Identifier (SPID) and device self-documentation string (DSDS) in the device

<resource file> string that provides formatting instructions for a network-variable or configuration-property type

3.18
functional block
portion of a device's application that performs a task by receiving configuration and operational data inputs, processing the data, and sending operational data outputs

NOTE A functional block may receive inputs from the network, from hardware attached to the device, and/or from other functional blocks on a device. A functional block may send outputs to the network, to hardware attached to the device, and/or to other functional blocks on the device. A functional block is an implementation of a functional profile. A "standard" functional block is one based on a standard functional profile template (SFPT).

3.19
functional-block index
sequentially assigned number identifying a functional-block implementation on a device

3.20
functional profile FP
template that describes common units of functional behaviour, also known as profiles, or FPs; which can be represented with a machine-readable functional-profile template (FPT)

NOTE Each functional profile consists of a profile description and a specified set of network variables and configuration properties designed to perform a single function on a device. The network variables and configuration properties specified by the functional profile are called the functional-profile members. A functional profile specifies whether the implementation of each functional-profile member is mandatory or optional. A profile is uniquely identified by a program-ID template, scope, and functional-profile number.

3.21
functional-profile key
functional-profile number

NOTE See 3.24.

3.22
functional-profile member
network-variable or configuration-property member of a functional profile

NOTE Each functional-profile member is identified as mandatory or optional by the functional profile. Each member also includes a text description of the member for the functional profile.

3.23
functional-profile member number
two-byte number that uniquely identifies a network-variable or configuration-property member of a functional profile

NOTE This member number is used to associate a network variable or configuration property on a device with the corresponding network-variable or configuration-property member of the functional profile. Member numbers shall be in the range of 1 to 4 095, and need not be continuous. Member numbers shall be unique, with the exception that network-variable and configuration-property members may use the same number. (Therefore, network-variable members' numbers shall be unique, and configuration-property members' numbers shall be unique, but they need not be unique between network-variable members and configuration-property members.) There may be a maximum of 255 mandatory members and 255 optional members of each type (scope 0 NV, inheriting NV, scope 0 CP, and inheriting CP).

3.24
functional-profile number
two-byte number that uniquely identifies a functional profile within the scope defined by the scope number and program-ID template of the resource file that contains the functional-profile definition

NOTE Also called the functional-profile key, or the FPT key.

3.25
functional-profile selector
an ASCII vertical bar ("|") or an ASCII number sign ("#") to denote the association of a network variable or configuration property with a scope-0 profile or a profile of a higher-numbered scope, respectively, where a higher-numbered scope would denote the NV or CP was added to enhance a scope-0 profile or that it applies to a non-standard profile

NOTE If the functional profile selector is a vertical bar, the member number identifies a member of a scope-0 profile. If the functional profile selector is a number sign, the member number identifies a member of the inheriting profile. The number-sign functional profile selector is always used for members of user functional profiles, including profiles that do not use inheritance. The vertical-bar functional profile selector is always used for members of standard functional profiles. Two different functional profile members may have the same member number as long as they use different functional profile selectors.

EXAMPLE The "|1" member of a functional profile is not the same as the "#1" member of the same profile. This prevents conflicts if new members are added to a standard functional profile that has already been used as the basis for inheriting profiles.

3.26
functional-profile template
functional profile in human- and machine-readable form

NOTE See 3.20.

3.27
global index
functional-block index

NOTE See 3.19.

3.28
inheriting profile
functional profile that inherits members from a scope-0 profile

3.29
interoperability
conditions that ensure multiple devices from the same or different manufacturers can be integrated into a single network without requiring custom device or tool development

3.30
CNP device
hardware and software that runs an application and communicates with other devices using the EN 14908-1 protocol

NOTE It may optionally interface with input/output hardware. A CNP device includes at least one processor and a CNP transceiver also called a CNP node, or simply a node.

3.31
CNP network
collection of intelligent devices that communicate with each other using the EN 14908-1 protocol over one or more communications channels

3.32
manufacturer ID
MID
20-bit number that uniquely identifies the device manufacturer of a device and is part of the device's SPID

3.33
network-interface selection
form of network-variable selection that occurs on the network interface

3.34
network variable
NV
data item that a particular device application program expects to get from other devices on a network (an input network variable) or expects to make available to other devices on a network (an output network variable)

NOTE Network variable data are typically stored in a device's volatile memory.

EXAMPLE Examples are a temperature, switch value, and actuator-position setting.

3.35
network-variable declaration
establishment of an instance of a network variable type within the code of an application

3.36
network-variable index
sequentially assigned number identifying a network variable implementation on a device

NOTE For Neuron C applications, the index is assigned by the Neuron C compiler in the order of declaration. The first network variable on a device has an index of 0, the second has an index of 1, etc.

3.37
network-variable member
functional-profile member that is a network variable

NOTE See 3.22.

3.38
network-variable member number
number of a functional-profile member that is a network variable

NOTE See 3.23.

3.39
network-variable programmatic name
name assigned to a network-variable implementation by the device application developer

NOTE The programmatic name is limited to 16 characters, including any optional prefixes. The programmatic name is not significant for interoperability, but conventions are suggested in 4.7.3.4 to make programmatic names easier to use for integrators.

3.40
network-variable selection
process of associating a network-variable selector with a network variable on a device

3.41
network-variable type
specification of the length, units, valid range, and resolution of the data contained within a network variable

NOTE A network variable type may be a simple, one, two, or four-byte scalar type; or a more complex structure or union of up to 31 bytes.

3.42
network-variable type index
16-bit number that uniquely identifies a network-variable type within the scope defined by the scope number and program-ID template of the resource file that contains the network-variable type definition

3.43
unique node ID
unique 48-bit identifier within the read-only data structure of a device as defined by the EN 14908-1 protocol

NOTE It is also called the unique_node_ID.

3.44
node
<common> device

<precise> physical and logical presence on a CNP network with a unique node ID and network address

NOTE The unique node ID relates to the identification of a single instance of an implemented EN 14908-1 protocol stack. A device is also a network presence with an application processor and one or more nodes. A device with multiple unique node IDs would consist of multiple nodes. Some infrastructure devices, such as routers, also consist of more than one unique node ID and thus consist of multiple nodes.

3.45
passive configuration tool
PCT
network tool that can be used on a device to assist in the successful commissioning of the device without disrupting the operation of other network tools

NOTE It may be a plug-in, standalone software, hardware attachment, or other tool. A passive configuration tool has attributes and capabilities as defined in clause 6.3.

3.46
primary functional block
functional block on a device that implements the most important function for the device

3.47
primary functional profile
functional profile that defines the primary functional block on a device

3.48
proprietary data
data and message definitions in the device interface that are known only to the manufacturer and the manufacturer's agents

3.49
self-documentation string
SD string
text string associated with a device, network variable, or configuration property that is stored within a device and within the device interface (XIF) file for a device

NOTE Network tools can read the self-documentation strings from the device itself or from the device interface file.

3.50
self-documentation text
optional text within a device, network variable, or configuration property self-documentation string that provides documentation of the intended use of the device, network variable, or configuration property respectively for use by integrators

3.51
shared-media channel
communications channel where messages can leak between tools and devices belonging to different systems

3.52
standard configuration-property type
SCPT
configuration-property type that has been standardized by prEN 14908-6

NOTE A SCPT is a standardized definition of the units, scaling, encoding, valid range, and meaning of the contents of configuration properties.

3.53
standard network-variable type
SNVT
network-variable type that has been standardized by prEN 14908-6

3.54
standard program ID
SPID
eight-byte number that uniquely identifies the device interface for a device

NOTE Encoded according to rules specified in clause 4.3.

3.55
static functional block
functional block that is statically defined for a device; that is, a functional block that is not a dynamic functional block

3.56
static network variable
network variable that is statically defined for a device; that is, a network variable that is not a dynamic network variable

3.57
subsystem
two or more devices working together to perform a function and bearing fixed, pre-defined relationships to one another

NOTE A subsystem may use one or more EN 14908-1 domains.

3.58
successful commissioning
<noun> process of taking a device and integrating it into a network

<adjective> device can be physically installed in a network and made to perform its application function with the exclusive use of its device interface and a choice of third-party tools

3.59
system
one or more independently managed subsystems working together to perform a function

NOTE A system may use one or more EN 14908-1 domains.

3.60
unconfigured device
device without a valid network configuration

3.61
usage
one-byte value describing the intended usage of the device and is part of the SPID of a device

NOTE The usage field consists of a one-bit changeable-interface flag, a one-bit functional-profile-specific flag, and a 6-bit usage ID.

3.62
usage ID
six-bit value in the least-significant portion of the usage field that identifies the primary intended usage of a device

3.63
user data
non-standardised user functional blocks, user network variables, and user configuration properties used by a device manufacturer to augment the device interface

3.64
wink function
function provided by a device that allows a network integrator to physically identify the device

EXAMPLE A wink function may blink an LED on the device.

4 Device Interfaces

4.1 General

The device interface is the network-visible interface to a device. The elements that comprise an interoperable device interface include the unique node ID, the standard program ID, the device channel ID, the device location field, the device self-documentation string, the device configuration properties, and the functional blocks.

Following is a summary of each of the elements:

— *Unique node ID*. A 48-bit unique identifier for a CNP device.

- *Standard program ID (SPID).* A number that uniquely identifies the device interface for a device.
- *Device channel ID.* A number that optionally specifies the channel to which the device is attached.
- *Device location field.* A string or number that optionally specifies the device location.
- *Device self-documentation string.* A string that specifies the functional blocks on a device.
- *Device configuration properties.* Configuration data used to configure the device. Functional blocks may also have configuration properties.
- *Functional blocks.* Logical components implemented on the device.

With the exception of the unique node ID, a network tool can read all of these elements directly from a device over the network, or from the device interface (XIF) file for the device as described in 4.9 *Device Interface (XIF) File*. The benefit of making this information available directly from the device itself is that a network tool can read all of the information needed to integrate and manage the device over the network, and no accompanying manufacturer documentation is required. The benefit of making this information available from a device information file is that the device may be designed into a network before physical access to the device is available. The latter method is typically used for engineered systems, but the former method is sometimes used when a device interface file is not available.

The device interface elements are described in the remainder of this clause. Device configuration properties are described in 4.7.4.

4.2 Unique Node ID

The unique node ID is a 48-bit number within the read-only data structure of a device as defined by the EN 14908-1 protocol. The unique node ID is a unique number written to a processor during development or manufacturing. Network tools use the unique node ID to send network installation messages to a device, prior to the device being assigned a network address as described in 6.2, Network Addressing.

Guideline 4.2: A device shall implement a unique node ID as defined in 4.2, Unique Node ID.

Manufacturers may wish to provide two copies of the unique node ID in a human- or machine-readable format, attached to the product. One copy should be removable so that an installer may place it on a system drawing, or similar plan. This can even be done using a barcode for ease and accuracy of unique node ID input into a network tool. An example unique node ID barcode label is shown in the following figure.

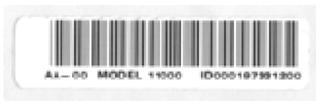


Figure 2 — Example Unique Node ID Barcode Sticker

While this standard does not mandate a bar-coding method, the CODE-39 format (ISO/IEC 16388) should be used for compatibility with many readily available barcode readers.

4.3 Standard Program ID

4.3.1 General

The *standard program ID* (SPID) is an 8-byte number within the read-only data structure of a device as defined by the EN 14908-1 protocol. It uniquely identifies the device interface for a device. It is used by network tools to associate a device with a device interface definition. This speeds up the commissioning process by allowing a network tool to obtain the device interface definition without uploading the entire definition from every device.

Guideline 4.3: A device shall implement a standard program ID as defined in 4.3, Standard Program ID.

The 16 hex digits of the SPID are organized as 6 fields that identify the format (F), manufacturer (M), device class (C), usage (U), channel type (T), and model number (N) of the device. These 6 fields are organized as follows, and are described in the following sections:

```
FM:MM:MM:CC:CC:UU:TT:NN
```

The manufacturer, classification, channel type, and optionally the usage fields contain values defined by prEN 14908-6.

4.3.2 Format Field

The Format field contains a four-bit value defining the structure of the program ID and device self-documentation strings. The format shall be “9” for all devices unless specified to be “8” during testing of the device.

4.3.3 Manufacturer Field

The Manufacturer field contains a 20-bit manufacturer ID (MID). The MID uniquely identifies the device manufacturer.

4.3.4 Device Class Field

The Device Class field is a two-byte value identifying the primary function of the device. This value is drawn from a registry of pre-defined Device Class definitions.

A device may implement multiple functional blocks. One of these functional blocks may be designated as the primary functional block, and the definition of this functional block is called the primary functional profile. If the primary functional profile number is greater than 99 and less than 20 000, the device class may be set to the profile number.

Standard functional profiles are also given device classes equal to their functional profile number. If a developer chooses to use a device class that is assigned to a standard functional profile, then the device containing that device class shall contain a functional block implementation of that profile.

4.3.5 Usage Field

4.3.5.1 General

The Usage field is a one-byte value describing the intended usage of the device. The Usage field consists of a one-bit Changeable-Interface flag, a one-bit Functional Profile-Specific flag, and a 6-bit usage ID. These subfields are described in the following sections.

4.3.5.2 Changeable-Interface Flag

The *Changeable-Interface flag* is the msb of the Usage field. It shall be set if the device uses changeable network variable types or dynamic network variables as described in 4.7.3.3 Changeable-Type Network Variables. The flag shall be clear if the device uses a fully static device interface.

4.3.5.3 Functional Profile-Specific Flag

The *Functional Profile-Specific* flag is the second-msb of the Usage field. It shall be set if the usage ID value is defined by the primary functional profile for the device. The flag shall be clear if the usage ID value is defined by the standard usage ID values or if the Device Class field does not identify the functional profile number of the primary functional profile for the device.

4.3.5.4 Usage ID

The *usage ID* is a 6-bit value in the least-significant portion of the Usage field that identifies the primary intended usage of the device. Based on the setting of the Functional Profile-Specific flag, the usage ID is defined by one of the following:

- If the Functional Profile-Specific flag is clear, the usage ID shall be set to one of the standard usage ID values.
- If the Functional Profile-Specific flag is set, the usage ID shall be set to one of the usage ID values specified by the primary functional profile for the device, as determined by the Device Class field.

4.3.6 Channel Type Field

The Channel Type field is a one-byte value identifying the communications channel type supported by the device's network transceiver. The standard channel-type values are drawn from a registry of pre-defined channel-type definitions.

4.3.7 Model Number Field

The Model Number field is a one-byte value identifying the specific product model of the device. Model numbers are assigned by the product manufacturer and shall be unique within the device class, usage ID, and channel type for a manufacturer ID. The same hardware may be used for multiple model numbers depending on the program that is loaded into the hardware. The model number within the SPID does not have to conform to the manufacturer's marketing or engineering model numbers. It can be used as a decimal reference, hexadecimal reference, or any other method of convenience.

Table 1 — Examples of model number fields

Kind of Model	Numbers
Decimal Model Numbers	0; 1; 2; 3;…9; 10; 11… = sequential by 1 10; 20; 30;…90; 100; 110… = incremental by 10s' place
Hexadecimal Model Numbers	01; 02; 03;…09; 0A; 0B… = sequential by 1 10; 20; 30;…90; A0; B0… = incremental by nibbles' place
ASCII-Character Model Numbers	"A"; "B"; "C"; … = sequential by 1 using ASCII values for the representation of characters (0x41; 0x42; 0x43;…)

4.4 Device Channel ID

The device channel ID is a 2-byte unsigned-long field within the configuration structure of a device as defined by the EN 14908-1 protocol. Network tools may use the device channel ID to track the channel to which a device is attached. A value of zero indicates that the device's channel ID is unassigned. When a device is shipped not configured and installed in a managed network, the device application shall not require specific values in this field since a network tool may change the value as needed.

Guideline 4.4A: A device shall be manufactured with a zero channel ID unless it is shipped configured or is a self-installing device; in which case, the channel ID can be non-zero.

Guideline 4.4B: A device shall not modify its channel ID field unless it is shipped configured or is a self-installing device; in which case, the device application can modify the channel ID.

4.5 Device Location Field

The device location field is a 6-byte field within the configuration structure of a device as defined by the EN 14908-1-protocol. Integrators, network tools, or the device itself may use this field to document the physical location of a device. This field may be read and written over the network using the Read Memory and Write Memory network-management messages defined by the EN 14908-1 protocol. Some devices can determine their physical location by reading external physical inputs such as DIP switches, keyed connectors, or card-cage slot numbers. Such devices may use the device location field to communicate their physical location information to a network tool that can use this information to identify the physical location of the device.

Use of this field is optional, but if used it shall conform to the following guideline. A device may optionally implement a location configuration property (see 4.7.3.5 Dynamic Network Variables and Functional Blocks) that can be used to provide a more complete location description than is possible in the 6-byte location field. The location configuration-property value is a string of up to 31 characters. When used for device location, the location configuration property shall apply to the Node Object functional block of the device if the device has a Node Object functional block, otherwise it shall apply to the entire device. 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 location configuration property to identify the location of each of the remote components. The location configuration property associated with the Node Object identifies the location of the device itself, whereas the other location configuration properties identify the locations of their respective hardware components.

Guideline 4.5A: A device's application program that wishes to communicate its physical location or ID assignment to a network tool can write this information into the location ID field of its configuration structure when the device is reset. If the most-significant bit of the first byte is one, the information is encoded as a 15-bit unsigned integer in the range 0 to 32 767, with the most-significant 7-bits in the lower 7-bits of the first byte (location[0]). If the most-significant bit of the first byte is zero, the information is encoded as a 0- to 6-character ASCII string. If the string is shorter than six characters, it shall be null-terminated (0x00).

Guideline 4.5B: A device that implements a location configuration property to represent the device location shall apply the CP to the Node Object functional block if the device has a Node Object functional block, and shall otherwise apply the CP to the entire device.

4.6 Device Self-Documentation String (DSDS)

The device self-documentation string is a string of up to 1 024 bytes (subject to device memory limits) within the self-identification structure of a device as defined by the EN 14908-1 protocol. This string specifies the self-documentation string structure, the functional blocks, and optionally describes the function of a device.

Guideline 4.6A: A device shall contain a device self-documentation string that specifies the self-documentation string structure and the functional profiles implemented by each functional block on the device as described in 4.6, Device Self-Documentation String.

Guideline 4.6B: A device shall store the device self-documentation string in the application image as described in the EN 14908-1 protocol.

The syntax for the device self-documentation string without arrays is as follows:

```
&3.4@Functionalblock,FunctionalBlock;selfDocText
```

The syntax for the device self-documentation string with arrays is as follows:

```
&3.4@0NodeObjectName,FunctionalBlock[arrayCount];selfDocText
```

The components of the documentation string are the following:

— ampersand ("**&**") prefix to denote an interoperable device.

— "**3.4**" substring identifying the major (**3**) and minor (**4**) version number of the standard implemented by the device, as defined by prEN 14908-6.

— at-sign ("**@**") separator.

— FunctionalBlock List is a list of functional profile numbers with optional array indices and names for each of the functional blocks implemented on the device. These numbers and names are delimited by a comma ("**,**"). The functional profile numbers shall be listed in order of the functional block indices, with the first functional profile number corresponding to functional-block index 0, the second to functional-block index 1, etc. The Node Object functional block, if implemented, shall be first with an index of 0.

  An array of functional blocks can be specified by appending an opening, left square bracket ("**[**") and an array dimension to the functional profile number. These may be followed by a closing, right square bracket ("**]**"), but the closing bracket may be omitted to save a byte of non-volatile memory.

  A device-specific name can be provided for a functional block by appending a string or string reference (as defined in 5.1.4.2, Self-documentation String Reference) immediately following the functional profile number and array dimension (if any). If the name is text, it shall consist of ASCII printable characters, be of no more than 16 characters in length, and shall not contain any left or right square brackets ("**[**" or "**]**"), commas, or periods, and it shall not begin with any number. Functional block names can improve device usability, especially when there are multiple functional blocks of the same type. For example, naming each of multiple sensor functional blocks that report the same type of data may aid the installer in picking the correct functional block on a device.

- An optional semicolon (“**;**”) terminator. The terminator is required if any selfDocText self-documentation text is included.
- selfDocText is optional self-documentation text. A description of the intended device usage for network integrators. The self-documentation text may include references to language strings as described in 5.1.4.2, Self-documentation String Reference. A 0x80 value (represented as a “**\x80**” ASCII string) is reserved for these references. A 0x81 value (represented as a “**\x81**” ASCII string) is reserved for future expansion.

4.7 Functional Blocks

4.7.1 General

A device application is divided into one or more functional blocks. A functional block is a portion of a device’s application that performs a task by receiving configuration and operational data inputs, processing the data, and sending operational data outputs. A functional block may receive inputs from the network, hardware attached to the device, or from other functional blocks on a device. A functional block may send outputs to the network, to hardware attached to the device, or to other functional blocks on the device.

The device application shall implement a functional block for each function on the device to which other devices should communicate, or that requires configuration for particular application behaviour. Each functional block shall be defined by a functional profile as described in Clause 5, Resource Files. Functional profiles are templates for functional blocks, and each functional block is an implementation of a functional profile.

The network inputs and outputs of a functional block, if any, are provided by network variables and configuration properties. A network variable is an operational data input or output for a functional block. A configuration property is a data value used for configuring the behaviour of a network variable, functional block, or the entire device. Configuration properties used to configure an entire device are not part of any functional block they are instead associated with the device itself.

A special type of functional block is called the Node Object functional block. Network tools use the Node Object functional block to test and manage the other functional blocks on a device. The Node Object functional block is also used to report alarms generated by the device. In the case of a device with only a single functional block, other mechanisms may be available for the test and management functions. In such a case, the Node Object functional block may be omitted, provided that both of the following conditions apply:

- application program does not need to continue operating when the functional block is disabled. In this case, setting the device off-line will disable the functional block.
- device does not implement alarms, self-test, range checking, fault detection, file transfer, or other functions belonging to the Node Object functional block.

Guideline 4.7: A device that supports more than one functional block shall include a Node Object functional block (as defined in 4.7) to allow monitoring and control of the functional blocks within the device. A device created with a single functional block shall also include a Node Object functional block if the device implements alarms, self-test, range checking, fault detection, file transfer, or other functions belonging to the Node Object functional block; or if the application program shall continue to operate when the functional block is disabled.

Figure 3 illustrates the relationship between the Node Object functional block, other functional blocks on a device, network variables, and configuration properties. Sections 4.7.3 and 4.7.4 describe network variables and configuration properties.

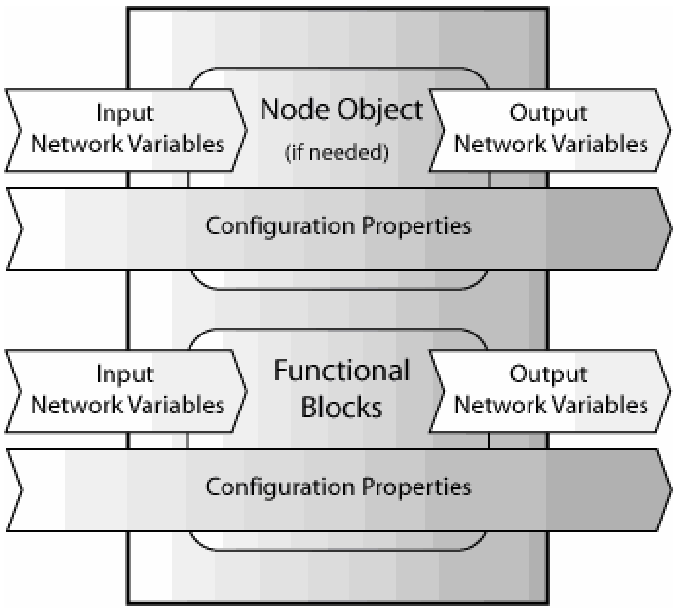


Figure 3 — Functional Block Interfaces

4.7.2 Implementing a Functional Block

The device self-documentation string (DSDS) contains a list of the functional blocks on a device. This list identifies the functional profile number implemented by each functional block, and assigns a unique functional block index number (also called the global index) to each functional block on the device, starting with zero. Each network variable and configuration property on a device includes self-documentation or configuration data that associate the network variable or configuration property with a functional block on the device using the functional block index number. The size of this self-documentation and configuration data is also subject to device memory limits, which is an especially important consideration for devices with limited, non-volatile memory.

To implement a functional block on a device, you create the self-documentation and configuration data as described in 4.6.

The functional block (fblock) is an object-oriented concept to which network variables and configuration properties become members. The relationship between the functional block and its members is shown in the following figure.

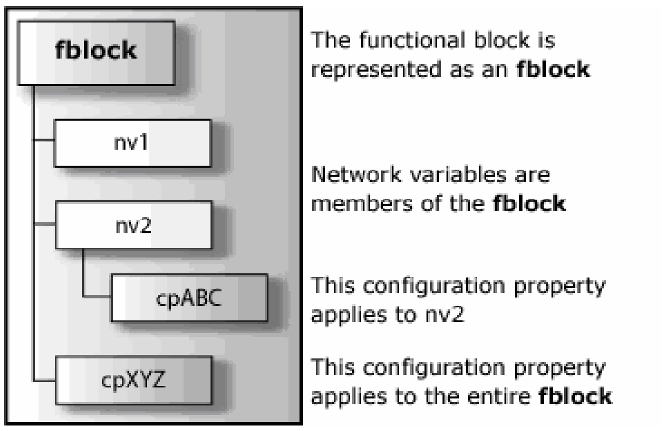

fblock
nv1
nv2
cpABC
cpXYZ
The functional block is represented as an **fblock**
Network variables are members of the **fblock**
This configuration property applies to nv2
This configuration property applies to the entire **fblock**

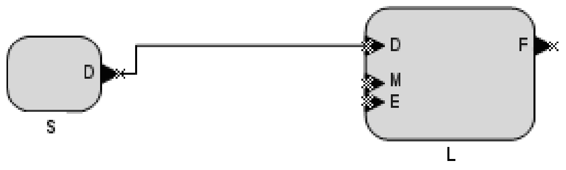
Figure 4 — NV/CP Relation to Functional Block

Sections 4.7.3 and 4.7.4 describe how to implement the network variable and configuration property members of the functional block.

4.7.3 Network Variables

4.7.3.1 General Aspects

The EN 14908-1 protocol employs a data-oriented application layer that supports the sharing of data between devices, rather than simply the sending of commands between devices. In this approach, application data such as temperatures, pressures, states, and text strings can be sent to multiple devices each of which may have a different application for each type of data. Applications exchange data with other devices using network variables. Every network variable has a direction, type, and length. The network variable direction can be either input or output, depending on whether the network variable is used to receive or send data. The network variable type determines the format of the data. The standard resource file set described in Clause 5, *Resource Files,* defines a set of standard types for network variables; these are called *standard network variable types* (SNVTs). Device manufacturers may also create custom network variable types as described in Clause 5. These are called user network variable types (UNVTs). Network variables of identical type and length but opposite directions can be connected to allow the devices to share information. For example, an application on a lighting device could have an input network variable that was of the switch type, while an application on a dimmer-switch device could have an output network variable of the same type. A network tool could be used to connect these two devices, allowing the dimmer switch to control the lighting device, as shown in Figure 5.

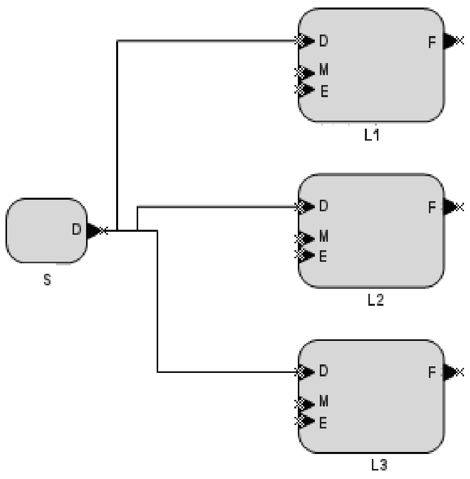


Key

D	Digital	F	Feedback
M	Mode	S	Switch
E	Enable	L	Light

Figure 5 — Network Variable Point-to-Point Connection

The direction indicated by the triangle in the above figure indicates the direction of the network variable. A single network variable may be connected to multiple network variables of the same type but opposite direction. A single network variable output connected to multiple inputs is called a fan-out connection. A single network variable input that receives inputs from multiple network variable outputs is called a fan-in connection. Figure 6 shows the same dimmer switch being used to control three lights using a fan-out connection:



Key

D	Digital	S	Switch
M	Mode	L1	Light 1
E	Enable	L2	Light 2
F	Feedback	L3	Light 3

Figure 6 — Network Variable Fan-out Connection

The application program in a device does not need to know from where input network variable values come nor to where output network variable values go. When the application program has a changed value for an output network variable, it simply passes the new value to the device firmware. Through a process called binding that takes place during network design and installation, the device firmware is configured to know the logical address of the other device or group of devices in the network expecting that network variable's values. It assembles and sends the appropriate packets to these devices. Similarly, when the device firmware receives an updated value for an input network variable required by its application program, it passes the data to the application program. The binding process thus creates logical connections between an output network variable in one device and an input network variable in another device or group of devices. Connections may be thought of as "virtual wires." For example, the dimmer-switch device in the dimmer-switch-light example could be replaced with an occupancy sensor, without making any changes to the lighting device.

4.7.3.2 Implementing a Network Variable

Each network variable that is part of the interoperable interface for a device shall be associated with a functional block. A single network variable can only be associated with a single functional block. To implement a network variable on a device and associate it with a functional block, create the self-documentation data as described in this Clause. The self-documentation strings shall be stored in non-volatile memory of the device to ensure that they are available after a power cycle.

Guideline 4.7.3.2A: A device shall include network variable self-documentation strings that map network variables to the functional blocks declared on the device.

Guideline 4.7.3.2B: The network variable and configuration property self-documentation strings shall be stored in non-volatile memory of a device.

Documentation of network variables is accomplished through the use of network variable self-documentation strings (NVSDs). A network variable self-documentation string is used to define membership of a network variable to a functional block. Network access to the network variable self-documentation strings is defined by the EN 14908-1 protocol.

EXAMPLE

> The third functional block declared on a device (functional block index 2) is based on a closed-loop actuator functional profile (numbered 4 for this example). This functional block has one mandatory input network variable and two output network variables of the same type one of which is mandatory and one of which is optional. The two output variables are of the same type, so it is important to know which network variable within the device corresponds to the first output versus the second output in that closed-loop actuator functional profile.

This mapping of network variables to functional blocks, and to specific network variables within the functional block, is done within the self-documentation string. Different self-documentation string formats are used for regular network variables, configuration network variables, and manufacturer-defined network variables as described below.

The syntax for a self-documentation string for a network variable, or a network variable array belonging to one or more functional blocks, is as follows:

```
@fbIndex[-endFbIndex]||#memberNum[[arraySize]][?][;[ text]]
```

where:

- `fbIndex` is the functional block index,
- *endFbIndex* is optionally the last functional block index in an array,
- '|' is the functional profile selector (to be either "|" or "#"),
- `memberNum` is the member number,
- `arraySize` is the optional array size, and
- `text` is the optional descriptive text.

For example:

```
@1|2;Standard-NV 2 of FB-index 1

@1-5#2[3]?;Changeable-type user-NVs 2-4 of FB-indices 1-5
```

The components of the self-documentation string are the following:

- ASCII at-symbol ("@") prefix.
- fbIndex is the functional block index of the functional block that contains the network variable or network variable array, or the index of the first functional block in a functional block array that contains the first network variable in a network variable array, if endFbIndex is specified. The first functional block on a device is index 0.

— endFbIndex is the functional block index of the last functional block in a functional block array that contains the last network variable in a network variable array. If a network variable array is specified and the endFbIndex value is omitted, the array is assumed to be a member array within the single functional block specified by fbIndex as defined in the profile. The array size is required to convey multiple functional blocks each containing member arrays. In this latter case, the network variable array is mapped as if it were a two-dimensional array as follows: nvName[fblockCount][memberArraySize]

— ASCII vertical bar (“|”) or ASCII number sign (“#”) that is the functional profile selector. If the functional profile selector is a vertical bar, the member number identifies a member of a standard profile (scope 0). If the functional profile selector is a number sign, the member number identifies a member of a user profile (scope 3 to 6).

— memberNum is the network variable member number within the functional profile, or the index of the first member number in a member array if arraySize is specified.

— arraySize specifies the array size for a member array.

— optional ASCII question mark (“`?`”) changeable-type specifier. The changeable-type specifier shall be included if the type of the network variable may change after installation.

— optional semicolon (“`;`”) terminator. The terminator is required if text is included.

— text is optional self-documentation text. A description of the intended network variable usage for network integrators. The self-documentation text may include references to language strings as described in 5.1.4.2, Self-documentation String Reference. A 0x80 value (represented as a “`\x80`” ASCII string) is reserved for these references. A 0x81 value (represented as a “`\x81`” ASCII string) is reserved for future expansion.

4.7.3.3 Changeable-Type Network Variables

Network variables may be of changeable type. Such network variables are used when the device developer cannot know the correct type of the network variable in advance. For example, a changeable-type output would be used for a generic sensor device that can attach to any standard sensor and report any sensed value. The actual type of the network variable can be changed to meet the physical units measured; however, the developer shall still declare an initial type for the network variable.

If a device supports any changeable-type network variables, it shall set the Changeable-Interface flag in the program ID as described in 4.3.5.2, Changeable-Interface Flag. It shall also declare a network-variable type configuration property that applies to the changeable-type network variable. Network tools use this configuration property to notify the device application of changes to the network variable type. The device application will require notification of changes to this configuration property. Notification can be provided using one of the methods described in the following procedure.

Guideline 4.7.3.3: A changeable-type network variable on a device shall implement a network-variable type configuration property that applies to the changeable-type network variable.

A configuration property may inherit its type from a changeable-type network variable. If the configuration property applies to multiple changeable-type network variables, all of the network variables shall share the same configuration properties to define the type and the maximum length.

4.7.3.4 Network Variable Naming Conventions

The programmatic name of a network variable may be prefixed with its storage class, as defined below. For compactness, underscores are typically not used and all characters are typically lowercase, except the first character of a word. The following conventions are used, but not required:

- network variable input: nviXxxxxxxxxxxxx
- network variable output: nvoXxxxxxxxxxxxx
- network variable output (ROM): nroXxxxxxxxxxxxx
- configuration network variable input: nciXxxxxxxxxxxxx

Due to the limitation of 16 characters for names of the network variables and configuration properties, there is a convention for abbreviations. The following list represents some typical abbreviations, but it is not meant to be all-inclusive:

— Actual	Act
— Calendar	Cal
— Clear	Clr
— Continuous	Cont
— Delay	Dly
— Device	Dev
— Discrete	Disc
— Electric	Elec
— Feedback	Fb
— Floating-point	f
— Frequency	Freq
— Hardware	Hw
— Increment	Inc
— Inhibit	Inh
— Input	In
— Level	Lev
— Maximum	Max

—	Micrometer	Micr
—	Minimum	Min
—	Parts-per-million	Ppm
—	Object	Obj
—	Output	Out
—	Position	Pos
—	Range	Rnge
—	Request	Req
—	Rate	Rt
—	Resistance	Res
—	Source	Src
—	Standby	Stby
—	String	Str
—	Table	Tbl
—	Time	T
—	Translation	Trans
—	Volume	Vol
—	Watt-hour	Whr

4.7.3.5 Dynamic Network Variables and Functional Blocks

A dynamic network variable is a network variable that is added to a device by a network tool after the device is installed; a dynamic functional block is a functional block that is added to a device by a network tool after the device is installed. These network variables and functional blocks may be created and deleted at will, rather than being statically declared. A network variable or functional block that is not dynamic is called a static network variable or static functional block. The only static declaration required for a device that implements dynamic network variables in addition to any other network variable declaration requirements defined in this standard is the maximum number of dynamic network variables and aliases supported on the device. The only static declaration required for a device that implements dynamic functional blocks in addition to any other functional block declaration requirements defined in this standard is the maximum number of dynamic functional blocks supported on the device. This information appears in the device interface (XIF) file for the device, and it can be queried from the device.

Support of dynamic network variables and functional blocks is optional; however, if a device can dynamically create and delete network variables or functional blocks after being installed in a network, then the method described in this section shall be used. Dynamic network variables are required to implement dynamic functional blocks, so if a device supports dynamic functional blocks, it shall also support dynamic network variables. A device that does not support dynamic network variables or functional blocks may ignore the commands described in this section.

Guideline 4.7.3.5: If a device implements dynamic network variables or functional blocks, the implementation shall conform to requirements listed in 4.7.3.5, Dynamic Network Variables and Functional Blocks.

A device that supports dynamic network variables shall implement the following:

1) Changeable-Interface flag shall be set in the program ID as described in 4.3.5.2, Changeable-Interface Flag.

2) If the device does not support dynamic functional blocks, it shall have a version 4.1 or later device interface file as specified in prEN 14908-6. If the device does support dynamic functional blocks, it shall have a version 4.401 or later device interface file.

3) device interface file shall specify the static and maximum dynamic portions of the interface.

4) device shall support and respond to the extended network-management commands to manage dynamic network variables including the commands to add and delete network variables and aliases, to query their attributes, and to bind them. These extended commands are based upon the Install command defined by the EN 14908-1 protocol, and are listed in the next section.

5) If the device implements dynamic functional blocks, it may optionally support and respond to the extended network-management commands to determine the number of dynamic functional blocks supported by a device. These extended commands are based upon the Install command defined by the EN 14908-1 protocol.

4.7.3.6 Extended Network-Management Commands

The extended network-management commands are an extension of the EN 14908-1 protocol “Install” command (message code 0x70). They provide methods to query self-identification/self-documentation (SI/SD) data, update SI/SD data, inform the device of a new network variable addition, and remove an existing network variable. Optional methods are also provided to increase the capacity of the domain and address tables, as well as other features. The syntax and usage of the commands are described in the Install and Install Command Data Structures sections of the EN 14908-1 protocol specification.

All devices shall implement support for a Wink request, which is the APP_WINK (0) application command within the Install command. However, devices that support dynamic network variables shall also support the following additional application commands within the Install command:

Table 2 — Additional application commands

Additional application commands	Description
APP_NV_DEFINE (2)	Create a new dynamic network variable declaration.
APP_NV_REMOVE (3)	Remove an existing dynamic network variable declaration.
APP_QUERY_NV_INFO (4)	Query SI/SD data for a network variable.
APP_QUERY_DEVICE_INFO (5)	Query SI/SD data for the device.
APP_UPDATE_NV_INFO (6)	Update SI/SD data for a network variable.

The command formats, data values, and additional commands to support expanded capacities and other extended features are described in the EN 14908-1 protocol specification.

Guideline 4.7.3.6: Devices that support dynamic network variables shall also support the APP_NV_DEFINE (2), APP_NV_REMOVE (3), APP_QUERY_NV_INFO (4), APP_QUERY_DEVICE_INFO (5), and APP_UPDATE_NV_INFO (6) application commands within the Install command as defined by the EN 14908-1 protocol.

4.7.4 Configuration Properties

4.7.4.1 General Aspects

A configuration property (CP) is a data item that, like a network variable, is part of the device interface for a device. Configuration properties characterize the behaviour of a device in the system. Network tools manage this attribute and keep a copy of its value in a database to support maintenance operations. If a device fails and needs to be replaced, the configuration property data stored in the database is downloaded into the replacement device to restore the behaviour of the replaced device in the system.

Configuration properties facilitate interoperable installation and configuration tools by providing a well-defined and standardized interface for configuration data. Each configuration property type is defined in a resource file that specifies the data encoding, scaling, units, default value, range, and behaviour for configuration properties based on the type. Many different standard configuration property types (SCPTs) are defined in the standard resource file set described in Clause 5 Resource Files. SCPTs provide standard type definitions for commonly used configuration properties such as dead-bands, hysteresis thresholds, and message heartbeat rates. Developers may also create their own user configuration property types (UCPTs) that are defined in resource files they create.

A configuration property shall apply to one of the following items:

— single network variable,

— single functional block,

— series of network variables,

— series of functional blocks,

— compilation of network variables,

- compilation of functional blocks,
- entire device.

The item to which the configuration property applies is known as the application set of the configuration property. The application set cannot contain more than one of the items in the list above.

Each configuration property within its specified application set shall be based on a unique configuration property type (SCPT or UCPT). For example, if a functional block has three output network variables, each requiring an independent configuration property X, then separate configuration properties X shall be declared to apply to the network variables within the functional block. The next section describes how configuration properties with compatible application sets may be shared, so a single configuration property may be shared among nv1, nv2, and nv3.

Manufacturer-defined user configuration properties are permitted because the configuration data for a given functional block is often implementation-specific. It is also permitted that modification of these user configuration data be necessary for the successful commissioning of the device. Any user configuration properties necessary to successfully commission the device shall be defined with documentation and machine-readable resource files as described in Clause 5, optionally, a passive configuration tool as defined in 6.3, Passive Configuration Tools, can be provided in addition to the required resource files.

Configuration properties that are members of functional blocks may be implemented as arrays. For example, a configuration property array may be used to create an event schedule or a translation table for the linearization of sensor data. The functional profile contains information for each member configuration property, whether implementation as an array is permitted, required, or disallowed. The functional profile also defines the valid array boundaries for each member configuration property that permits array implementation.

Guideline 4.7.4.1A: If user configuration properties shall be modified for successful commissioning of a device, those configuration properties shall be defined with resource files and user documentation as described in Clause 5. The documentation and resource files, if any, shall be supplied at the time the device is tested.

Guideline 4.7.4.1B: Any and all configuration information required to be modified to successfully commission the device shall be implemented and exposed via configuration properties with user documentation.

4.7.4.2 Configuration Property Distribution Methods

When a configuration property or configuration property array applies to multiple functional blocks or multiple network variables, there are two distribution methods that determine how the configuration property or elements of the array are distributed among the applicable functional blocks or network variables.

The first method is called CP sharing. Using this method, the entire configuration property or CP array applies to all the specified functional blocks or network variables. For example, given a configuration property array, cpMyCps[4], here is the disbursement of the entire array to four functional blocks using CP array sharing:

cpMyCps[0] -> functional block 0

cpMyCps[1] -> functional block 0

cpMyCps[2] -> functional block 0

cpMyCps[3] -> functional block 0

cpMyCps[0] -> functional block 1

cpMyCps[1] -> functional block 1

cpMyCps[2] -> functional block 1

cpMyCps[3] -> functional block 1

cpMyCps[0] -> functional block 2

cpMyCps[1] -> functional block 2

cpMyCps[2] -> functional block 2

cpMyCps[3] -> functional block 2

cpMyCps[0] -> functional block 3

cpMyCps[1] -> functional block 3

cpMyCps[2] -> functional block 3

cpMyCps[3] -> functional block 3

The second method is called CP dividing. CP dividing only applies to CP arrays; a singular CP is atomic and may not be divided. Using this method, each element of a CP array is applied to a corresponding element of the applicable functional blocks or network variables. For example, given a configuration property array, cpMyCps[4], here is the disbursement of the individual elements of the array to four functional blocks using CP array dividing:

cpMyCps[0] -> functional block 0

cpMyCps[1] -> functional block 1

cpMyCps[2] -> functional block 2

cpMyCps[3] -> functional block 3

This avoids having a separate self-documentation string for every element in the array. This technique can also result in substantial memory and code-space savings, though commissioning time is typically increased due to a potentially less efficient memory layout. CP array dividing cannot be used unless the CP array has exactly the same number of elements as the number of functional blocks or network variables to which the CP applies.

CP sharing can be used with both a series of network variables or functional blocks and a compilation of network variables or functional blocks. CP dividing cannot be used with compilations.

The implementation of multiple members of a functional profile may share the same configuration property unless the functional profile documentation specifies that they cannot be shared. For example, a profile may define nvo1, nvo2, and nvo3 outputs, each with a mandatory CP X member that applies to it called cpX1, cpX2, and cpX3. A functional block implementation of this profile may implement a single CP X that is shared among nvo1, nvo2, and nvo3, and still fulfil the requirement to implement all mandatory members of the profile.

4.7.4.3 Configuration Property Implementation Methods

You can implement a configuration property using one of two different methods. The first, called a configuration network variable, uses a network variable to implement the configuration property. This has the advantage of enabling the configuration property to be modified by another CNP device, just like any other network variable. It also has the advantage of having the network variable event mechanism available to provide notification of configuration property updates to the application. The disadvantage of configuration network variables is that they are limited to a maximum of 31 bytes each.

The second method of implementing configuration properties uses configuration files to implement the configuration properties for a device. Rather than being separate, externally exposed elements of data, all configuration properties implemented within configuration files are combined into one or two blocks of data called value files. A value file consists of configuration property records of varying length concatenated together. Each value file shall fit as contiguous bytes into the memory space of the device that is accessible by the application. When there are two value files, one contains writeable configuration properties and the second contains read-only data. To permit a network tool to access individual elements of data in the value file, there is also a template file consisting of an array of text characters that describes the elements in the value files.

The advantages of implementing configuration properties within configuration files are that there are no limits on configuration property size or the number of configuration properties except as constrained by the available memory space on the device and the maximum file size for the Control-Network Protocol File Transfer Protocol (CNP FTP). The disadvantages are that other devices cannot connect to or poll a configuration property implemented within a configuration file typically requiring a network tool to modify a configuration property implemented within a configuration file and no events are automatically generated when a configuration property implemented within a configuration file is updated. The application can force notification of updates by requiring network tools to reset the device, disable the functional block, or take the device offline when a configuration property is updated. Alternatively, the application can also force notification by implementing configuration-file access via the CNP FTP and monitoring the transfer. This option requires additional code space for the CNP FTP-server code.

Table 3 summarizes the advantages and disadvantages of the two implementation methods. For a given configuration property, the developer shall choose one of these methods. However, both methods can be implemented on a device, just not for the same configuration property.

Table 3 — CP Implementation Trade-offs

Configuration NV	CP Within a Configuration File
Limited to 31 bytes in length	May be of any length
Uses one network variable for each configuration property declared	Does not require a network variable
Application notified of updates via NV update event	Requires alternate update notification method
Easily modified by another device (when the device-specific attribute is set)	Typically requires a network tool to update

Guideline 4.7.4.3: A device shall implement a given configuration property as either a configuration network variable or as an element of a configuration file.

4.7.4.4 Configuration-File Access Methods

When configuration properties are implemented within configuration files, you shall provide a method for a network tool to access the configuration file. You may provide one of the following three access methods:

— Direct memory-read/write, (DM-R/W)

— CNP FTP with random and sequential access,

— CNP FTP with sequential access.

The direct memory-read/write access method enables a network tool to read and write configuration files using EN 14908-1 Read Memory and Write Memory network-management messages. This method does not require file-transfer code on the device, but cannot be used with all types of protocol processors. For a protocol processor to support this method, the application-layer processor shall support EN 14908-1 Read Memory and Write Memory network-management messages. For protocol processors where this method does work, it requires that the configuration file fit entirely within the memory space supported by the EN 14908-1 Read Memory and Write Memory network-management messages for the application processor.

The Control-Network Protocol File Transfer Protocol (CNP FTP) with random- and sequential-access method enables a network tool to read and write configuration files using the CNP FTP, with both random and sequential access to any blocks within the configuration file. This method requires file-transfer code on the device, but can be used with any protocol processor.

The Control-Network Protocol File Transfer Protocol (CNP FTP) with sequential access method is identical to the CNP FTP with random- and sequential-access method; with the exception that random access to blocks within the configuration file is not provided. This method is used when a CNP FTP access method is required or preferred, and the protocol processor does not have sufficient application memory space to implement random access. It can be considerably more inefficient than the other two access methods and should only be used if none of the other two access methods can be provided.

Network tools determine which access method to use based on interfaces provided in the Node Object functional block. See the Node Object Functional Profile for details on these interfaces.

Guideline 4.7.4.4: If a device implements any configuration properties within configuration files, then one and only one of the following access methods shall be provided as described in 4.7.4.4, Configuration-File Access Methods: direct memory-read/write, CNP FTP with random and sequential access, or CNP FTP with sequential access.

4.7.4.5 Configuration Property Flags

Each configuration property on a device may specify optional flags that are used to notify a network tool of whether or not a configuration property can be modified, and if so, when it can be modified. These flags are optional. If a configuration property is declared without any flags, a network tool may assume that the configuration property can be modified at any time.

Constant

Specifies a configuration property that can never be changed by a network tool. If the device-specific flag is not set, also specifies a CP that can never be changed by the device application; if the device-specific flag is set, the device application may change the constant CP. Also specifies that the CP is implemented in a read-only value file for CPs implemented within configuration files on devices that contain both a writeable value file and a read-only value file. However, network tools may write such configuration properties when they reside in a value file on a device that does not implement a read-only value file as long as the value is not changed. A network tool may do this as part of an update to another CP adjacent to the constant value. CPs with the Constant flag but without the Device-Specific flag can be assumed to have the same value on all devices using the same standard program ID.

Device-Offline

Specifies that a network tool shall take this device offline, or ensure the device is already offline, before modifying the configuration property. This flag or the FB-Disabled flag is recommended for a configuration property implemented within a configuration file with direct memory-read/write access if the application requires update notification, or if the application cannot tolerate updates from the network at the same time the application is reading the configuration property. This flag should not be used for configuration properties implemented within configuration files that are accessed via CNP FTP, and network tools should ignore this flag for such configuration properties. This is because a device cannot transfer configuration property values via CNP FTP while offline. In fact, an offline application shall be placed into the online state for the duration of any CNP FTP configuration property operations.

Device-Specific

Specifies a configuration property that shall always be read from the device if available instead of relying upon the value in the device interface file or a value stored in a network database. A device-specific CP may be set by the device implementing the CP, or by a passive configuration tool. Network management tools shall never change a device-specific CP value except as a side effect of a new program download, device recommissioning, or device replacement. A device-specific CP may be used for a CP that shall be exclusively managed by the device and/or a passive configuration tool such as a setpoint that is updated by a local operator interface on the device, or a minor version number that varies from device to device. A device-specific passive configuration tool, or a user of a generic passive configuration tool, may determine that a CP is device-specific even though it is not documented as such in the device interface. A network management tool may include an interface to communicate such a determination to the network management tool however, this is not required. If this determination is not communicated to a network-management tool, it is likely that the network-management tool will overwrite a device-specific CP with a stale value when a device is recommissioned or replaced.

FB-Disabled

Specifies that a network tool shall disable the functional block containing the configuration property, take the device offline, or ensure that the functional block is already disabled or the device is already offline, before modifying the configuration property. This flag or the Device-Offline flag is recommended for a configuration property implemented within a configuration file with direct memory-read/write access if the application requires update notification, or if the application cannot tolerate updates from the network at the same time the application is reading the configuration property. A network tool may elect not to disable a functional block before modifying a configuration property with the FB-Disabled flag if that device is already offline and can be updated while offline. This is allowed because an offline device has all its functional blocks implicitly disabled, and because a functional block cannot be directly disabled when the device is already offline.

Manufacturing-Only

Specifies a factory setting that may be read or written when the device is manufactured, but is not normally (or ever) modified in the field. In this way, a standard installation tool may be used during manufacture to calibrate a device, while a field installation tool would observe the flag in the field and prevent modification of the value, or optionally require a password to modify the value.

Reset-Required

Specifies that a network tool shall reset the device after changing the value of the configuration property. If multiple modifications of configuration properties are being made at the same time, then one reset can be completed in lieu of having to reset the device the same number of times as Reset-flagged configuration properties were modified.

4.7.4.6 Implementing a Configuration Property

To implement a configuration property on a device and associate it with a network variable, a functional block, or the device itself, you create the self-documentation data or configuration files as described here.

Guideline 4.7.4.6: A device that includes configuration properties within the interoperable interface shall include configuration property documentation strings that declare the configuration properties and map them to the entire device; one or more network variables; or, one or more functional blocks declared on the device.

Documentation of configuration properties implemented as configuration network variables is accomplished through the use of network variable self-documentation strings (NVSDs), but using a different syntax as detailed in this section. Documentation of configuration properties implemented within configuration files is accomplished through the use of declaration strings within the configuration file. In either case, the documentation defines whether the configuration property applies to the entire device, one or more functional blocks, or one or more network variables. Configuration properties shall be documented if they are to be part of the interoperable interface of a tested device. The syntax for a documentation string for a configuration property (which may be a configuration property array) implemented as a configuration network variable is as follows:

```
&header,select,flag,index,dim,rangeMod[,?];selfDocText
```

The syntax for a documentation string for a configuration property (which may be a configuration property array) implemented within a configuration file is as follows (the only differences are removal of the ampersand prefix and addition of the length value):

```
header,select,flag,index,length,dim,rangeMod[,?];selfDocText
```

The components of the configuration property documentation string are the following:

— ampersand ("&") prefix. It is only included for a configuration network variable.

— header specifies whether the configuration property applies to the entire device ("0"), a functional block or functional blocks ("1"), or a network variable or network variables ("2") on the device.

— select optionally specifies to which functional blocks or network variables the configuration property applies. This field is not specified if the configuration property applies to the entire device. The field values are listed in Table 4. A single configuration property may apply to multiple network variables or functional blocks. Therefore, unlike a network variable, a single configuration property may correspond to multiple members of multiple functional profiles. The association with the member or members in the functional profile or profiles is made by matching the type of a configuration property and the application set objects to which it applies. This means that multiple functional profile configuration property members may be implemented by a single configuration property.

Table 4 — Select-Field Values

CP Applies To	Select-Field Value
Entire device	Null
One functional block	Functional block index. If the CP is an array, all elements of the array apply to the functional block.
Contiguous series of functional blocks, with the CP shared by all functional blocks	First functional block index and last functional block index separated by a hyphen (“–”): *firstFbIndex–lastFbIndex*. If the CP is an array, all elements of the array are shared by all functional blocks.
Contiguous series of functional blocks, with the CP divided among all functional blocks	First functional block index and last functional block index separated by a tilde (“~”): *firstFbIndex~lastFbIndex*. This option may only be used for CP arrays.
Non-contiguous compilation of functional blocks, with the CP shared by all functional blocks	Functional block indices separated by periods (“.”). If the CP is an array, all elements of the array are shared by all functional blocks.
Non-contiguous compilation of functional blocks, with the CP divided among all functional blocks	Functional block indices separated by slashes (“/”). This option may only be used for CP arrays.
One network variable	Network variable index. If the CP is an array, all elements of the array apply to the network variable.
Contiguous series of network variables, with the NV shared by all functional blocks	First network variable index and last network variable index separated by a hyphen (“–”): *firstNvIndex–lastNvIndex*. If the CP is an array, all elements of the array are shared by all network variables.
Contiguous series of network variables, with the NV divided among all functional blocks	First network variable index and last network variable index separated by a tilde (“~“): *firstNvIndex~lastNvIndex*. This option may only be used for CP arrays.
Non-contiguous compilation of network variables, with the NV shared by all functional blocks	Network variable indices separated by periods (“.”). If the CP is an array, all elements of the array are shared by all network variables.
Non-contiguous compilation of network variables, with the NV divided among all functional blocks	Network variable indices separated by slashes (“/”). This option may only be used for CP arrays.

“flag” is a two-digit hexadecimal number encoded as a sequence of two ASCII digits. The first digit specifies the scope of the resource file set that defines the configuration property type. The value may be a “0”, “3”, “4”, “5”, or “6” digit as defined 5.3 Managing Resource Files. The second digit encodes the flags described in 4.7.4.5, Configuration Property Flags. The values for each flag are listed in Table 5. These values may be or’d together. For example, both the Device-offline and the Fbdisabled flag may be specified by or’ing 0x82 with 0x81, yielding a value for the second byte of 0x83. At least one of the values in Table 5 shall be specified. When specified in a C or Neuron C application, the value shall be encoded as “\xhexDigits”, where hexDigits are the two hex-encoded values for the second byte.

Table 5 — Flag Values

Flag	Value
Constant	x84
Device-offline	x82
Device-specific	xA0
FB-disabled	x81
Manufacturing-only	x90
No restrictions	x80
Reset-required	x88

— *index* specifies the configuration property index within the specified resource file set.

— *length* specifies the configuration property size in bytes. If a CP array is specified using the dim field, the length refers to only one element of the array. The length field is only specified for a configuration property implemented within a configuration file.

— *dim* optionally specifies the dimension of a configuration property array. If not specified, the configuration property is not an array. If specified, the dimension shall be at least two.

— *rangeMod* optionally narrows the range specified by the configuration property definition in the functional profile, or specified by the configuration property type. The rangeMod value is a string that can only be used with fixed-point and floating-point types, and consists of a pair of either fixed-point or floating-point numbers delimited by a colon (“:”). The first number is the low limit while the second number is the high limit. If either the high limit or the low limit should be the maximum or minimum specified in the configuration property definition (from the functional profile) or specified in the configuration property type definition (from the resource file set outside of the functional profile), then the field is empty to specify this. In the case of a structure or an array, if one member of the structure or array has a range modification, then all members shall have a range modification specified. In this case, each range modification pair is delimited by a vertical bar (“|”). To specify no range modification for a member of a structure (that is, revert to the default for that member), the field is encoded as a single terminating vertical bar with no other characters. The same encoding is used for structure members that cannot have their ranges modified due to their data type. Empty encoding is only allowed for members of structures. Whenever a member of a structure is not a fixed or floating-point number, its range may not be restricted. Instead, the default ranges shall be used. In the case of an array, the specified range modifications apply to all elements of the array. Both initial values and range modifiers may be used. Development and network tools shall apply them in the following order:

- Values provided with the CP reference within an **fb_properties**, **nv_properties**, or **device_properties** clause, if any.
- Values provided with the declaration of the CP family, if any.
- Values provided with the functional profile template definition in the resource files, if given.
- Values provided with the configuration property or network variable type that is used with the functional profile, if given.
- Initial values are considered to be zero unless defined elsewhere (in any of the places listed in this list above). Range modifiers default to the natural minimum and maximum value for the underlying base data type, unless defined in any of the items listed above.

- optional question mark ("?") changeable-type specifier. The changeable-type specifier shall be included if the type of the configuration property may change after installation. The changeable-type specifier is required for configuration properties with inheritable types if the configuration property is implemented as a configuration network variable. If a configuration property is implemented within a configuration file, the question mark is not needed since the base type of the configuration property is not specified in the configuration file.
- semicolon (";") terminator.
- selfDocText is optional self-documentation text. A description of the intended configuration property usage for network integrators. The self-documentation text may include references to language strings as described in 5.1.4.2, Self-documentation String Reference. A 0x80 value (represented as a "`\x80`" ASCII string) is reserved for these references. A 0x81 value (represented as a "`\x81`" ASCII string) is reserved for future expansion.

Each field in the documentation string is delimited by a comma ("`,`"). Two consecutive commas ("`,,`") indicate that the field is null (empty or unspecified), except when a semicolon is encountered. In the event that a semicolon is encountered, all remaining fields of the string are considered null.

4.8 Device and Functional Block Versioning

Versioning is an important part of upgrading and verifying systems. It can be useful information for a person tasked with such maintenance. Every device shall have a standard program ID (SPID). In many cases, when the device interface changes, the SPID shall be modified to indicate that the device is not the network-interface equivalent of other devices on the network. This convention performs basic device versioning. However, not all changes to a device interface require the SPID of a device to change. Additionally, it is possible to change the program within a device without changing its device interface, which also does not require the SPID to change.

A more flexible method of device versioning can be accomplished with the use of the standard configuration-property types that define minor and major version numbers. These two configuration properties shall have a range of 0–255 and a default value of 0. The configuration property for the major version shall always specify the Constant flag, while the configuration property for the minor version shall always specify both the Constant flag and the Device-Specific flag. The Constant flag without the Device-Specific flag means that all devices with the same program ID will have the same value, while the Constant flag with the Device-Specific flag means that devices with an identical program ID may have different values for this configuration property. See 4.7.4.4, Configuration-File Access Methods, for a complete list of flags and their values.

The presence of these configuration properties within a device defines the major version and minor version of the device. The major version number shall be incremented when the device interface changes, while the minor version number shall be incremented when the device interface remains the same but the device has a different behaviour.

These device-versioning configuration properties are optional configuration properties of the Node Object functional profile. They should not be used outside of a Node Object functional block except for when the device does not implement a Node Object functional block.

For devices with multiple functional blocks declared within them, it is useful to know which functional blocks have changed. To support the versioning of individual functional blocks, the configuration properties for minor and major object version are defined. These two configuration-property types shall have a range of 0–255 and a default value of 0. The configuration property for the object major version shall always specify the Constant flag, while the configuration property for object minor version shall always specify both, the Constant flag and the Device-Specific flag. The Constant flag without the Device-Specific flag means that all devices with the same program ID will have the same value, while the Constant flag with the Device-Specific flag attribute means that devices with an identical program ID may have different values for this configuration property. The presence of these configuration properties within a functional block defines the major version and minor version of the functional block. The major version number shall be incremented when the device interface implemented by the functional block changes, while the minor version number shall be incremented when the network interface remains the same, but the functional block has a different behaviour.

Guideline 4.8: If any of the configuration properties for the versioning is included on a device, they shall be implemented as described in 4.8, Device and Functional Block Versioning

4.9 Device Interface (XIF) File

The device interface (XIF) file is a standalone file that documents the device interface for a type of device. It also documents the default values for all the configuration properties on the device. The XIF file is an important component of a device's definition and shall be submitted with the device's resource files when a device is submitted for testing.

The device interface file provides a summary and documentation of the device interface for a device with a specified SPID. The device interface file is created based on the specifications in prEN 14908-6.

Guideline 4.9A: The device interface for a device shall be documented in a version 4.0 or newer device interface file as described in 4.9, Device Interface (XIF) File. The device interface file shall be submitted with the test application.

Guideline 4.9B: The device interface for a device shall include default values for all configuration properties as described in prEN 14908-6.

5 Resource Files

5.1 Resource File Definitions

5.1.1 General

Resource files are files that define the functional profiles and types referenced by the device interface for one or more CNP devices. These files allow network tools such as installation tools and operator-interface applications to interpret data produced by a device and to correctly format data sent to a device. They also help a system integrator or system operator to understand how to use a device and to control the functional blocks on a device.

Resource files are available that define the standard components used in device interface definitions. Device manufacturers shall create user resource files for any user-defined components used by their device interfaces.

Resource files are used to publish definitions for both standard and manufacturer-defined resources. Standard resources include standard functional profiles, standard network variable types (SNVTs), standard configuration property types (SCPTs), and standard enumeration types. Manufacturer-defined resources include user functional profiles, user network variable types (UNVTs), user configuration property types (UCPTs), and user enumeration types.

Resource files are grouped into resource file sets, where each set defines functional profiles, network variable types, configuration property types, enumeration types, strings, and formats for specified device types. The range of device types that a resource file set applies to is called the “scope” of the resource file set. For example, the scope may specify that the resource file set applies to an individual device type or to all device types. The available scopes are defined in 5.3, Managing Resource Files.

Each resource file set may contain definitions for the following resources:

Table 6 — Definition of resources

Definition	Description
Network Variable Types	Type information for network variables. This information includes the size, units, scaling factors, and type category (float, integer, signed, *etc.*) for each type.
Configuration Property Types	Type information for configuration properties. This information includes the size, units, scaling factors, and type category (float, integer, signed, *etc.*) for each type.
Functional Profiles	Functional profiles define a template for functional blocks. Each functional profile is a collection of network variables and configuration properties designed to perform a single function on a device.
Enumeration Types	An enumeration type is a list of numerical values, each associated with a mnemonic name.
Language Strings	Language-specific strings that are referenced by type definitions, functional profiles, and self-documentation strings.
Formats	Formatting instructions for network variable and configuration property types.

These resources are described in more detail in the following sections.

5.1.2 Type Definitions

5.1.2.1 General Aspects

A type definition may be a network variable, configuration property, or enumeration type definition. Network variable and configuration property types specify the data type, size, units, and scaling factors for the type. The data type may be a base type, an enumeration type, a structure type, a union type, or a reference to another network variable type.

5.1.2.2 Base Types

The following base types are available:

- Signed character (8 bits),
- Unsigned character (8 bits),
- Signed short (8 bits),
- Unsigned short (8 bits),
- Signed long (16 bits),
- Unsigned long (16 bits),
- Signed quad (32 bits),
- Unsigned quad (32 bits),
- IEEE 754 single-precision floating point (32 bits),
- IEEE 754 double-precision floating point (64 bits).

The base types define size in bits but with the exception of floating point do not define fractional values, nor do they define resolution and upper/lower limits. Limits are imposed by the range of values that can be represented using the number of bits of the type.

5.1.2.3 Enumeration Types

An enumeration type is a list of numerical values, each associated with an enumerator name. If a network variable or configuration property type contains an enumeration, the definitions of the enumerated values are maintained separately as an enumeration type.

By convention, enumeration type names use all lower case, with each word in the name separated by an underscore, and ending with “_t” (for example: count_control_t). Enumeration type names are limited to 64 characters, including the “_t” suffix characters.

By convention, all the enumerator names within an enumeration type use a common, unique, prefix. The enumerator names use all upper case, with words separated by underscores (for example: DCM_SPEED_CONST and DCM_PRESS_CONST). Enumerator names are limited to 64 characters, including the unique prefix.

Each of the numerical values for an enumeration type is a signed 8-bit value. The range is – 126 to 127. As convention, a value of – 1 (0xFF) is reserved for an invalid or undefined value, and a value of 0 is typically used for the default value. The – 1 (0xFF) value is always named <Prefix>_NUL.

5.1.2.4 Structure Types

A structure type defines an aggregate data type that consists of one or more fields. For example, a date network variable type may contain 3 fields: one each for the year, month, and day. Each field of a structure type may itself be a base type, a bitfield with a width of 1–8 bits, an enumeration type, another structure type, a union type, or a reference to another network variable type. There is no specified, programmatic naming convention for structures.

When used as the data type for network variables, all fields of a structure are updated when a network variable value update is sent on the network. There is no means to transmit an individual field of a network variable structure.

Structure type names are limited to 48 characters; but, if used as the data type of a network variable, they are limited to 16 characters. Field names are also limited to 48 characters.

5.1.2.5 Union Types

A union type defines an aggregate data type that consists of one or more fields. Unlike a structure type, the start of each of the fields of a union type overlaps. Like a structure type, each field may itself be a base type, a structure type, a union type, or a reference to another network variable type. There is no specified, programmatic naming convention for unions.

To allow for a network tool to determine the active portion of the union, a union is typically defined as a field within a structure, where the first field of the structure is a base-type value that is used as a selector for the union. This enables a format to be defined for the structure that uses one or more ternary operators to select the appropriate format based on the selector value.

Union type names are limited to 48 characters; but if used as the data type of a network variable, then they are limited to 16 characters.

5.1.2.6 Network Variable Type Definitions

A network variable type definition specifies the data type, size, units, and scaling factors for a network variable type. Even though it is possible to declare a network variable directly using a base type, such a declaration cannot be used for tested devices except in the case of a configuration network variable.

Standard network variable type (SNVT) names always begin with a "SNVT_" prefix and are always lowercase (e.g.: SNVT_some_type). By convention, user network variable type names begin with "UNVT_" and are lowercase (e.g.: UNVT_my_type).

Names of network variable types are limited to 16 characters, including the five prefix characters.

The type for a SNVT field may be based on a SNVT, but cannot be based on a user network variable type (UNVT). A UNVT field may be based on a UNVT or SNVT.

5.1.2.7 Configuration Property Type Definitions

A configuration property type definition specifies the data type, size, units, and scaling factors for a configuration property type. Configuration property types are defined in a resource file with a .typ extension (this is the same file used for network variable types).

A network variable type may be used as the data type of a configuration property type, but this is not required. There is no requirement to create a new network variable type for a new configuration property type. Standard configuration property types (SCPTs) and SCPT fields may be based on SNVTs, base types, or enumerations, but they cannot be based on UNVTs. User configuration property types (UCPTs) and UCPT fields may be based on UNVTs, SNVTs, base types, or enumerations.

Standard configuration property type names always begin with a “SCPT” prefix and are always a combination of uppercase and lowercase characters. By convention, user configuration property-type names begin with a “UCPT” prefix and are a combination of uppercase and lowercase characters (e.g.: UCPTmyConfigurationType).

Names of configuration property types are limited to 63 characters, including the four prefix characters.

5.1.3 Functional Profiles

5.1.3.1 Definition of a Template for a Functional Block

A functional profile defines a template for a functional block. Each functional profile consists of a profile description and a specified set of network variables and configuration properties designed to perform a single function on a device. The network variables and configuration properties specified by the functional profile are called the functional profile members. A functional profile specifies whether or not each functional profile member is mandatory or optional. When a functional block implements a functional profile, it shall implement all mandatory functional profile members defined by the functional profile, and it may implement some, all, or none of the optional functional profile members. A functional block may also add implementation-specific members that are not defined in the functional profile, but this is not recommended for tested devices. As described in 4.7.4.2, Configuration Property Distribution Methods, multiple configuration property members may share a single configuration property implementation on a device as long as they are the same type, are part of a compatible application set, and the sharing is not prohibited by the functional profile documentation.

Functional profiles are defined in a resource file with an .fpt extension. Functional profiles are also called functional profile templates.

In addition to the functional profile members, a functional profile also specifies the semantic meaning of the information being communicated. Thus, a functional profile provides additional information on usage, beyond the type information specified by a network variable or configuration property type.

Functional profiles that have been approved and published by EN 14908 are called standard functional profiles. The primary function of a device shall be implemented using one or more standard profiles. Developers can choose the profiles that best fit the functions of the device being developed..

Functional profiles that are not standard are called user functional profiles. A user functional profile cannot be used to implement the primary function of a tested device.

Guideline 5.1.3.1A: The primary function of a device shall be implemented with one or more functional blocks that conform to one or more standard profiles as defined in 5.1.3.1, Definition of a Template for a Functional Block.

Guideline 5.1.3.1B: Each functional block shall, as a minimum, implement all the functional profile’s mandatory network variables and configuration properties. Any optional members implemented by the functional block shall be implemented as specified in the profile; with the exception that the application set of an optional configuration property may be changed.

If an optional configuration property is not implemented in a functional block, then it is recommended that the device follow the specified default value, whenever possible, to ensure consistent behaviour.

5.1.3.2 Functional Profile Names

Each functional profile shall have a name that is unique within its resource file set. The name shall start with "SFPT" for standard profiles and "UFPT" for user profiles. The name may not contain spaces. By convention, there is no underscore following the "SFPT" or "UFPT" prefix; the first letter after the prefix is lower case; and the name uses mixed case. Functional profile names are limited to 64 characters, including the four-character prefix. A functional profile name may include underscores, but cannot use spaces or any other special characters.

5.1.3.3 Functional Profile Numbers

Each functional profile has a unique number, called the functional profile number or the functional profile key, which uniquely identifies the profile. The functional profile number need only be unique within the scope of the functional profile as described in 5.3, Managing Resource Files.

The standard functional profiles are specified in prEN 14908-6 A standard profile number may be used as the device class for a device implementing the standard profile as the primary functional block for functional profile numbers between 100 and 19 999, inclusive and greater then 25 000. As described in the next section, inheriting profiles use the same functional profile number as the scope-0 profile from which they inherit their content. Manufacturers are free to assign any functional profile number to new non-inheriting profiles, as long as they are in the range of 20 000 to 25 000, inclusive and as long as the number is unique for the program ID template and scope of the resource file set containing the functional profile.

EXAMPLE:

A UFPTmyCreation user functional profile is defined as follows:

— Scope: 4

— Program ID template: 80:00:9F:20:00:00:00:00

— Functional profile number: 21 234

In this case, the manufacturer with MID 0:00:9F cannot define another functional profile with a profile number of 21234 in any other resource file sets with a scope of 4 (see 5.3, Managing Resource Files, for a description of this value) and a device class of 20:00. The manufacturer can, however, create another functional profile with the same profile number in a resource file set at scope 3, 5, or 6. The manufacturer can also create another functional profile with the same number in a scope-4 resource file set as long as the device class is not 20:00. Other manufacturers can also create functional profiles that use profile number 21234, subject to the same rules. If the same functional profile number is defined in multiple resource file sets, the definition in the resource file set with a matching program ID template and the numerically highest value applies. For example, if functional profile 21234 is defined in both a scope-3 and scope-4 resource file set, the definition in the scope-4 resource file set applies if the program ID template matches.

5.1.3.4 Inheritance

If a device application implements a functional block based on a standard functional profile and adds additional members not defined in the standard profile, a user functional profile can be created that defines the additional members. To simplify development and maintenance of the user functional profile, the user functional profile may inherit the members defined in the standard functional profile and therefore only require the new members to be defined.

A functional profile that uses inheritance is called an inheriting profile. An inheriting profile includes a flag that specifies that it inherits members from a scope-0 profile. The inheriting profile shall have the same functional profile number as the scope-0 profile. The inheriting profile may have a different name; however, a name similar to that of the scope-0 profile is recommended.

5.1.3.5 Member Names

Each network variable and configuration property member of a functional profile has a unique member name for that profile. Member names may be up to 64 characters. The member name may contain only letters, numerals, and the underscore character. A prefix is not required, but input network variable names may start with “nvi”, output network variable names may start with “nvo”, and configuration property names may start with “cp” (preferred) or “nci”. By convention, the name is mixed case with no underscores, starting with a lower-case character if a standard prefix is used, and otherwise starting with an upper-case character. For example, “nviEnergyHoldOff” follows this convention. The abbreviations listed in 4.7.3.3, Changeable-Type Network Variables, should be used.

5.1.3.6 Member Numbers

Each network variable and configuration property member of a functional profile has a unique member number for that profile. This member number is used to associate a network variable or configuration property on a device with the corresponding network variable or configuration property member of the functional profile. Member numbers may be in the range of 1 to 4095, and need not be contiguous. Member numbers shall be unique, with the exception that network variable and configuration property members may use the same number. There is a maximum of 255 mandatory members and 255 optional members of each type (scope 0 NV, inheriting NV, scope 0 CP, and inheriting CP).

Each member of an inheriting profile may be defined in one of two functional profiles: the inheriting profile itself and the inherited scope-0 profile with the same functional profile number. To correctly associate each network variable and configuration property on a device with either an inheriting profile or a scope-0 profile, the member number is prefixed by a functional profile selector. If the functional profile selector is an ASCII vertical bar (“|”), the member number identifies a member of a scope-0 profile. If the functional profile selector is an ASCII number sign (“#”), the member number identifies a member of the inheriting profile.

The number-sign functional profile selector is always used for members of user functional profiles, including profiles that do not use inheritance. The vertical-bar functional profile selector is always used for members of standard functional profiles. Two different functional profile members may have the same member number as long as they use different functional profile selectors. For example, the “|1” member of a functional profile is not the same as the “#1” member of the same profile. This prevents conflicts if new members are added to a standard functional profile that has already been used as the basis for inheriting profiles.

5.1.4 Language Strings

5.1.4.1 General Language Strings

Language strings are text strings that are referenced by type definitions, functional profiles, and self-documentation strings. Language strings are stored in language files. There is one language file for every language supported by a resource file set. Language strings are referenced by index within the language file so that language string references may be translated by looking up the reference in the appropriate language file. This index is called the language-string index. Language strings may contain all printable ASCII characters except the tilde (“~”). C-like escape codes (“\n”, “\x3D”, etc.) are not supported. A network tool may allow a user to specify a search order for language files, and can therefore control which set of strings are displayed, depending on the chosen and available language files. Each language file uses a unique file extension so that it can use the same base filename as the rest of the resource file set. The standard language file extensions are listed in Appendix B.

5.1.4.2 Self-documentation String Reference

The self-documentation text within a self-documentation string can reference a language string using the reserved 0x80 value (represented as an “\x80” ASCII string). Some network tools may not recognize these string references.

The syntax for a self-documentation string reference is as follows:

\x80[scopeSpecifier:]languageStringIndex

The components of a self-documentation string reference are the following:

— byte containing the value 0x80, represented by the “\x80” string.

— scopeSpecifier may be a “3”, “4”, “5”, or “6” to specify a scope 3, 4, 5, or 6 resource. If not included, the scope is 0.

— colon (“:”) following the scope specifier. The colon is not included if the scope specifier is not included, otherwise it is mandatory.

— languageStringIndex is the index of the language string within the language file. This index ranges from 1 to 16 777 216.

EXAMPLE:

The following string reference specifies a language string index, 522 in this example, within the standard resource file set:

“Dictates the desired state of the actuator, determined by the specific application.”

"@2|1;\x80522"

The following string reference specifies language string index 100 within a user resource file set at scope 3.

"@2#1;\x803:100"

5.1.5 Formats

5.1.5.1 Different Formats for Data

Formats can be created and modified for each network variable type or configuration property type. A format specifies how a value is to be displayed, printed, or entered. Formats allow the physical representation of data to be independent of how users view the data. This is especially important for any type of measurement data since most measurement types have at least two display formats, one for United States (US) units and one for Système Internationale (SI or metric) units. Formats are also important for data that is viewed differently in different locales. For example, times and dates are displayed differently in different regions of the world. Formats may include locale-specific interpretation of times and dates, using locale information from the user’s operating system.

If a format is not available for a network variable or configuration property, most network tools will display the value as raw hexadecimal bytes. Formats allow for customizing how network integrators and network operators see the values. When a network variable or configuration property type is created, a default format should be created. The default format should use the text format specifier (see below Using the Text Format Specifier). In the case of char, short, long, enumeration, float, or quad types, this format should display the raw value. In the case of an array, structure, union, bitfield, or reference type, the format should be set to Missing format for

<TYPENAME>, where <TYPENAME> is replaced by the name of the network variable type or configuration property type.

Each format is named with a type name followed by an optional modifier. For example, if a network variable type named UNVT_my_type is created, it should have a default format also named UNVT_my_type. Multiple formats can be created for a type by appending a modifier to the additional formats. A modifier is a string that is appended to the format name, following a "#" character. Standard modifiers are defined for SI, US, and localized formats, and custom modifiers can also be created.

For example, UNVT_my_type#SI and UNVT_my_type#US can be created if it is desired to have the type be formatted differently when displayed in US or SI units.

5.1.5.2 Using the Text Format Specifier

The text format specifier has the following syntax: text(<text format list>). The text format list is similar to the ANSI C printf() arguments, with some simplifications and extensions. The text format list is a comma-separated list of text formats. Each text format consists of one of the following:

— quoted string called a format string. The format string consists of characters to be included in the formatted output, and may include conversion specifications that specify how a corresponding field data argument is formatted. A conversion specification may apply to the entire value to be formatted, or may apply to fields within the value by adding the field names to the text format list. You can also include localized list separators in format strings. See 5.1.5.3, Using Conversion Specifications, and 5.1.5.6, Using Localized List Separators for more information.

— field name from the value being formatted. The value shall be a structure or union type. Field names are applied to conversion specifications in format specifications that precede the field name in the text format list, applied from left to right. A format can display up to a maximum of 127 fields of a structure or array type. See below: Using Conversion Specifications.

— conditional format to specify one of two different formats, where one format is selected when a value is formatted based on a conditional value. See below: Using a Conditional Format.

— scaling factor to specify a multiplier and adder, and an optional unit string suffix, that are used to scale the value to be formatted. A scaling factor may be applied to the entire value, or to an individual field of a structure or union. See below: Using a Scaling Factor and Unit String.

— localized time or date function. These functions format a time or date according to the user's operating system's locale settings. See below: Using Localized Time and Date Formats.

5.1.5.3 Using Conversion Specifications

You can use a conversion specification within a format string to specify how the value of a field should be formatted. To format a field, append the field name in the text format list after the format string. Include one field name for each conversion specification in the list. The conversion specifications are applied to the field names from left to right. You can specify the following conversion specifications, where the following definitions exist:

[signed] long [int] is a 16-bit quantity;

unsigned long [int] is a 16-bit quantity;

signed char is an 8-bit quantity;

[unsigned] char is an 8-bit quantity;

[signed] [short] [int] is an 8-bit quantity;

unsigned [short] [int] is an 8-bit quantity; and

enum is an 8-bit quantity (int type).

%c	single character. The base type shall (as it is defined above) be char, int, or enum.
%d	signed or unsigned decimal number (based on the signed-ness defined in the type file). The base type shall (as it is defined above) be char, int, long, enum, or a bitfield (“unsigned”).
%f	floating point number. The base type shall (as it is defined above) be char, int, long, float, or enum.
%m	enumeration. The base type shall be an enum, enumerated list. If an numeration does not exist for the value, the format string is processed as if it were %d.
%s	null-terminated string. The base type shall be an array of 8-bit data. String data shall be null terminated (0x00).
%x	An unsigned hexadecimal integer. The size is determined from the type file. The data are always treated as unsigned. The base type shall (as it is defined above) be char, int, long, enum, or a bitfield.

You can use a backslash (‘\’) character as an escape character to include other format characters as text. For example, the following characters can be included in a format string:

\%	% character.
\\\\	\ character.
\\"	" character.
\\\|	| character.

5.1.5.4 Using a Conditional Format

You can use a conditional format to specify one of two different formats, where one format is selected when a value is formatted based on a conditional value. The syntax for a conditional format is similar to the ANSI C “?:” conditional expression. The syntax is as follows:

<condition> ? <format if condition is true> : <format if condition is false>

The condition is limited to expressions with the equal-to ('==') and is-not-equal-to ('!=') comparison operators.

Note: The field that appears in the conditional statement shall appear in a text format list before it appears in the conditional statement. Formats are processed in left-to-right order.

5.1.5.5 Using a Scaling Factor and Unit String

You can use a scaling factor within a format string to specify a multiplier and adder and an optional unit string suffix that are used to scale the value to be formatted. You can scale any simple data type, and you can also scale any field in a structure or union that is a simple data type. The scaling factors are applied as a multiplication and an addition when data is converted for output, and they are applied in the reverse order, as a subtraction and a division when data is input.

You can also specify a scope and language string index that specifies a language string to use as the unit description. This string overrides the unit description string found in the type file.

Alternate formats with scaling factors can be used for converting units to other measurement systems.

The syntax for a scaling factor is as follows:

```
*<Multiplier>+<Adder>[(<Unit String Scope>:<Unit String Index>)]
```

5.1.5.6 Using Localized List Separators

You can include a locale-specific list-separator character in a format string. To do this, specify a localized ("#LO") modifier and include a vertical bar ('|') where you want the list separator in the format string. The vertical bar is translated to the operating system list-separator character for the current operating system default locale.

5.1.5.7 Using Localized Time and Date Formats

You can include time and date localization functions to format a time or date value as specified by the operating system default locale method. The date format specifier requires three parameters, which specify the data fields where it will find the year, month, and day values to be formatted. The time format specifier requires two to four parameters, specifying hour and minute values to be formatted, and optionally, second and millisecond values.

6)

The time and date format specifiers may only be used in localized formats as described in Creating and Modifying a Resource Format.

Following are examples of the time and date localization functions.

5.2 Identifying Appropriate Resources

5.2.1 Standard and User Resources

To promote interoperability between devices, there are defined many standard resources. These standard resources specify standard functional profiles corresponding to specific functions that are common in specific controls industries such as temperature sensors and space comfort controllers, and also specify standard operational and configuration data types required by the controls industry. Standard resources should be used in applications whenever possible. In some cases, a developer may find that there is a resource that they want to use that is not defined in the standard resource file set. In this case, the developer has two options propose a new standard resource or develop a user resource.

If the required resource is specific to a particular implementation, installation, or company, the developer shall create a user resource file set defining any required user functional profiles (UFPs or UFPTs), user network variable types (UNVTs), and user configuration property types (UCPTs) required by the interoperable interface of the device. Section 5.2.3 identifies guidelines for using user resources. To facilitate reuse, a user functional profile should be defined as a general solution, rather than the specific one. Configuration properties should be used to configure a functional profile to meet specific requirements. This approach prepares for future reuse, and also prepares for proposing the user functional profile as a standard.

5.2.2 Using Standard Resources

The standard resources are defined in the standard resource file set. The standard resource file set includes definitions for standard functional profiles (SFPs or SFPTs), standard network variable types (SNVTs), standard configuration property types (SCPTs), standard enumeration types, standard language strings, and standard formats.

5.2.3 Using User Resources

User resources are distinct from a manufacturer's proprietary data because user data are intended to be manipulated by parties other than the manufacturer or the manufacturer's agents. It is allowed that it may be necessary to manipulate user data to successfully commission a tested device, but manipulation of manufacturer data cannot be a requirement to successfully commission a tested device. Manufacturer data may be used for calibration, diagnostic, test, and repair interfaces used solely for manufacturing or field troubleshooting operations that are not required for normal commissioning and operation of the device.

Guideline 5.2.3A: All user functional profiles, user network variable types, user configuration property types, and user enumeration types required for the interoperable interface of a device shall be documented within a resource file set as described in Clause 5, Resource Files. A minimum of one language file shall be included defining any required language strings. A format file shall be included defining a minimum of one format for each user network variable type and user configuration property type. The resource file set, if any, shall be submitted with the test application; and the passive configuration tool, if any, shall be made available at the time of application submittal.

Guideline 5.2.3B: There shall be no requirement to access or modify proprietary data in the course of successfully commissioning a device. The lack of access to proprietary data shall not prevent the successful operation or use of the device's published, interoperable functional blocks.

When creating a user functional profile, manufacturers can either inherit from an existing standard functional profile or define a new user functional profile. A user functional profile that does not inherit from a standard functional profile, as defined in 5.1.3.4, Inheritance, cannot form the primary function or basis of a device, but can be supplemental and complementary to the standard functional profiles on the device.

Developers who choose to add manufacturer-specific network variable and configuration property members, as a part of their interoperable interface, to the functional blocks on their devices shall provide resource files that contain user functional profiles defining the manufacturer-specific members. If the user network variables or configuration properties are added to a standard profile, then the standard profile shall be inheriting or redefined in the user resource files. Inheritance should be used for all new profiles. Additionally, if the network variable or configuration property type is not a standard network variable or configuration property type (SNVT or SCPT), then the user network variable or configuration property type (UNVT or UCPT) shall be defined within the resource file set.

Guideline 5.2.3C: User network variables or configuration properties that are intended to be a part of a functional block's interoperable interface shall be documented in a user functional profile within the device's resource file set. The functional profile shall define the network variable and configuration property members. Network variables or configuration properties that are not associated with a particular functional block, but pertain to the device as a whole, can be assigned to the Node Object functional block as manufacturer-defined network variables or configuration properties.

6 Network Installation

6.1 General

The physical attachment of devices to a communications channel, such as a twisted-pair wire or a power-line circuit, is not enough to commission a control network. The physical attachment only provides a path for the device to send and receive messages. The device also needs information on the system to which it belongs and the other devices with which it should share data. Specifying and loading this additional information is a necessary step for installing a device into a control network. This information is called network configuration data.

Network configuration data is managed through a process called network management. Network management is the act of putting network configuration data into a persistent store (typically a database) with the intent of making the network configuration data in a network of devices consistent with that network configuration data in the persistent store, and maintaining that consistency over time. Just storing the data into a persistent store, for example, is not network management; it is just a backup or snapshot of the data at any one point. Network management tasks include address assignment, binding, and configuration.

The device design plays an essential role in how it will be installed into an interoperable control network. Regardless of whether a single device or a subsystem consisting of a collection of devices needs to be installed into an interoperable network, a network tool shall be able to manage the logical connections between devices. Bringing devices and systems on-line, making connections, polling, and querying devices, are all services that a network tool may perform and to which a device or subsystem shall be able to respond.

This clause outlines the design guidelines that shall be followed so that devices can be installed into an interoperable network. It is also important that the network tools used to install the interoperable network support installation of devices that follow these guidelines.

6.2 Network Addressing

6.2.1 Network Addressing Scheme

Devices use their network addresses to send messages and to determine if messages are destined for them. A device’s network address consists of the following three components defined by the EN 14908-1 protocol:

- domain to which it belongs.
- subnet to which it belongs within the domain.
- node ID within the subnet.

Using EN 14908-1, a device can be a member of up to 65 535 domains. A key function of a network tool is to ensure that in any domain, no two devices are assigned the same subnet and node ID. Different protocol processors may have different limits on the number of domains that they support.

A device can also be addressed by using group addresses, assigned during the binding process. A single message can be addressed to all members of a group. Under EN 14908-1, a device can be a member of up to 65 535 different groups. Different protocol processors may have different limits on the number of groups that they support.

The binding process also allocates network variable selectors. Network variable selectors are 14-bit numbers used to identify network variables. All network variables in a connection shall have the same network variable selector value. In addition, the assigned network variable selector shall allow each device to uniquely associate an incoming network variable update with one of its input network variables. As with network address assignment, in a managed, a network tool is responsible for allocating group addresses, tracking group membership, assigning network variable selectors, and reassigning network variable selectors as needed to produce the desired logical connectivity. That is, for each network variable, the network tool shall ensure that messages are only sent to, received by, and processed by the desired set of devices.

Network addresses may be defined in a number of ways, including the following:

— Programmed into the device when it is manufactured. This is typically used for closed, self-contained systems.

— Self-installed by each device during field installation. This is typically used for small, closed systems.

— Assigned by a network tool during field installation in a managed network. This is used for most systems. A network tool is a component that monitors or modifies network configuration data, application configuration data, and/or application operational data for one or more devices using a network to communicate with the devices. A network tool may be a network management tool, a passive configuration tool, a monitoring tool, a control tool, or any combination of these. A network management tool is a tool that stores network configuration data into its persistent store and actively maintains consistency between the data in its persistent store and the devices on the network being managed.

Each of these methods represents a trade-off in terms of ease of initial installation, flexibility, and cost of tools. The EN 14908-1 protocol and these guidelines have been designed to make all of these installation scenarios compatible. Systems installed with one of the simple scenarios can migrate at a later date to a more sophisticated network-management scenario, without having to change device application code or hardware.

To correctly allocate network resources, such as device addresses and network variable selectors, a network tool shall have the freedom to reassign resources as needed. To support interoperable systems, installation dependencies shall not be built into the devices. When installed in a managed network, a device's functional behaviour shall be independent of its address or the details of its connections to other devices. All messages should be sent using either implicit addressing, with addresses assigned by a network tool, or using explicit addressing where the explicit addresses are determined at installation time using configuration properties.

Guideline 6.2.1: A device application installed in a managed network shall not be dependent upon its network configuration.

6.2.2 Address-Table Entries

Each distinct implicit destination address for an outgoing network variable update, poll, or application message requires an address-table entry. In addition, each group to which the device belongs requires an address-table entry. The maximum number of address-table entries under EN 14908-1 is 65 535, and each requires five bytes of non-volatile memory. Different protocol processors may have different limits on the number of address table entries that they support.

The number of address-table entries may affect the ease of installation of the device, since network variable and message-tag binders may fail if there are an insufficient number of these entries on the device. However, in a memory-limited application, there is a trade-off between application functionality and these table entries. Wherever possible, at least 15 address-table entries should be supported to avoid binder failure.

Guideline 6.2.2: Wherever possible, a device should have sufficient address-table entries to support every bindable network variable and message tag. This will often not be possible for some protocol processors to support. As a minimum, all tested devices shall support a number of address-table entries equal to the number of non-configuration network variables plus the number of bindable message tags, or the protocol processor limit (but no fewer than 15), or the EN 14908-1 protocol limit of 65 535, whichever is fewer.

6.2.3 Network Variable Aliases

Network variable aliases are another tool for the device developer to conserve address-table entries, and also to prevent limitations due to network variable connection constraints. For example, network variable aliases are required on a device when a single network variable output on the device shall be connected to two or more network variable inputs on another device. For example, a single switch connected to a device containing four actuators where the single switch shall simultaneously control all four of the actuator inputs.

Network variable aliases can also conserve address-table and group-address entries on monitoring devices. For example, when an output network variable on a device is connected to one or more other network inputs on another device and that same output variable needs to be bound to the monitoring device there are two alternatives:

— Have the monitoring device be a member of the group in the original connection.

— Allocate an alias to the output network variable, and send the alias as a unicast update to the monitoring device (unicast addressing does not consume address-table entries on the receiver device).

Aliases are often required when installing devices in open networks. The number of aliases to implement on a specific device depends upon the application, the available device resources, and the network topology of the network where it is installed. For these reasons, the guideline regarding network variable aliases only requires that device developers provide a reasonable number by using their application knowledge and understanding, and taking into account the devices' available memory resources. In lieu of more specific guidance by the device's application, the following formula may be used as a rule of thumb for a minimum value:

```
NumAliases = (NVcount==0) ? 0 : min(62, 10+(NVcount/3));
```

Guideline 6.2.3: A device shall support a reasonable number of network variable aliases to avoid binding errors due to network variable connection constraints.

6.2.4 Domain-Table Entries

With regard to the number of domain-table entries, it is often useful to have a device be a member of the zero-length domain so that it may be queried without knowing its unique node ID. This is useful when the network database is lost and shall be recovered from the network itself. While the unique node ID may be acquired by activating the device's service pin, and the domain table read with a second command using the unique node ID, the service pin may not be easily accessible on devices in some applications. For example, the device may be on a roof or behind a wall. If it is inconvenient, or not practical, to activate the service pin on a device which has only a single domain-table entry, and that device's configured domain is unknown, then the device cannot be recovered. In these cases, the Query ID network-management message shall be used to get the unique node ID. While the service-pin message is always sent as a domain-wide broadcast on the zero-length domain, the Query ID network-management message is domain specific. Thus, a network tool shall know one of the domains of the device to use the Query ID network-management message, or it shall already know the unique node ID. Since the zero-length domain is not typically used for normal system operation, the need for the second domain entry arises from the need for devices to be members of their own system domain and the zero-length domain so that the Query ID network-management message may be used on a known domain to assist in database recovery. Once the system domain is known, all devices that are members of that domain may be recovered.

Guideline 6.2.4: A device shall support at least two domains.

6.2.5 Self-Installed Devices

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 and group 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.

Self-installed devices can communicate across an interoperable network in one of the following two ways:

— Using a subsystem gateway (described later).

— After re-installation onto the network using a network tool.

Each self-installed device shall contain a Node Object functional block with a special configuration network variable as specified in the Node Object functional profile to indicate whether the configuration is done by self-installation or by a network-management tool. When the device is installed into a network using a network tool, the network tool changes the value of the configuration network variable to indicate to the device application that the network tool has taken over management of the device. The device application shall not set its own network addresses when the network-management tool has written to this configuration network variable.

Guideline 6.2.5: A self-installed device shall implement a Node Object functional block with a special configuration network variable as described in 6.2.5, Self-Installed Devices. It shall be possible to configure the device with a network tool, and have that device's address be set to any legal EN 14908-1 Control Network Protocol address when installed in a managed network.

6.2.6 Field-Installed Devices

Field-installed devices are installed in a managed network using a network tool. The tool is typically one of the following two types:

— tool is invisible to the user and is embedded in the network. It performs installation and maintenance behind the scenes. This is known as automatic installation. To the end user, the network appears to install itself. In reality, the tool is analyzing the network contents, and is automating installation based on a set of rules.

— user interacts with the network tool to configure the network. In this case, the tool might be embedded in the network for example, integrated into a monitoring and control station or it might be a portable tool that is attached to the network only during installation and maintenance.

6.3 Passive Configuration Tools

A passive configuration tool is a network tool that can be used on a device to assist in the successful commissioning of the device without disrupting the operation of other network tools. It may be a software plug-in, standalone software, hardware attachment, or other tool. A passive configuration tool has the following attributes and capabilities:

— provides one or more means to monitor or alter configuration properties or network variables solely for the purposes of replacing, commissioning, or installing the device.

— may be used for device-specific configuration or monitoring.

— does not interfere with other tools or network management devices.

— does not make changes to any network-configuration information (for example, address-table entries) on any device both installed and not installed on the network.

- leaves a device in the same state as it found it; however, during its operation, it is free to modify the device's state and reset the device in the course of modifying the configuration properties.
- In recognition of the fact that a passive configuration tool may take a device offline or reset a device, there can be system-level disruptions while using a passive configuration tool on a device without first coordinating the activity with the other devices, systems, or system operators that depend upon the normal operation of the device.
- is available to anyone owning the device on equivalent business terms, and such availability shall be demonstrably free of any discriminatory terms and conditions.

Guideline 6.3: If a passive configuration tool is required for successful commissioning of a device, the tool shall conform to the definition of a passive configuration tool in 6.3, Passive Configuration Tools.

6.4 Service Pin

The service pin is a physical or logical button on a device that causes the device to broadcast its unique node ID and program ID. It is used during installation to uniquely identify a device and its application to a network tool. The network tool then uses unique node ID addressing to assign a network address as described in 6.2.1, Network Addressing Scheme.

The method used to activate the service pin varies from application to application. Examples of mechanical methods include activating via an accessible push-button switch, or a magnetic-reed switch located within an enclosure. A service-pin message can also be sent under software control. For example, the device can send the message when the device is powered up or when a predefined series of I/O events occur. Sending the service-pin message exactly at power-up is not recommended because it will cause a spike in network traffic when power is restored after a power failure.

Even if a service pin will not be used as the default identification method for installing the device, some method for activating the service-pin input shall be accessible to a maintenance technician. The service pin is a simple way to ensure that an installer can always identify, and thereby establish communication with, a given device. If necessary, the service pin can be located inside the device such that it is accessible only to service personnel. However, the activation of the physical or logical service pin at an asynchronous and arbitrary moment shall not cause adverse device or network function. For example, activation of the service pin will not cause physical or logical reset of the device, nor will it cause extraneous network traffic.

Guideline 6.4: A device shall provide internal or external access to its service pin. The device shall respond with a service-pin message as defined in the EN 14908-1 protocol when the service pin is activated. For example, the service pin may be activated when a service button is momentarily depressed.

6.5 Gateways to Command-Based Systems

In a command-based system composed of multiple devices, commands are sent between the devices to initiate system actions. This implies that the devices sending and receiving the commands agree on the command semantics and actions. Building a gateway to such a system and simply propagating the command structure across the gateway would not allow the command-based system to interoperate with an EN 14908-1 system because the EN 14908-1 devices were not programmed to use these commands. In fact, to get interactions between the devices on both sides of the gateway, the EN 14908-1 devices would have had to be designed to send and receive the other system's commands. Since EN 14908-1 devices communicate via functional blocks, this method of gateway construction severely limits interoperability.

A better method for constructing a gateway to a command-based system is to think of the entire command-based system as a single EN 14908-1 device with a set of standard functional blocks that accomplish the interoperable functions of the command-based system. Once this abstraction of the command-based system is defined, it then becomes the interface between the gateway device and the EN 14908-1 devices. Within the gateway, translations between the commands and the EN 14908-5 functional blocks are accomplished by the gateway software. In this way, knowledge of the command set is confined to the gateway and the command-based system. Any EN 14908-1 device with functional blocks defined that are compatible with those defined on the gateway can interact with the command-based system without the foresight of the device developer.

This same technique may be used to create gateways to proprietary CNP devices that do not meet the requirements of these guidelines. It can also be used to create gateways between network subsystems that are installed using a network tool, and those that are self-installed. This enables a proprietary device or a self-installed subsystem to be viewed as an interoperable subsystem the proprietary or self-installed network is independently managed and it interfaces to other devices and subsystems through one or more gateway devices.

Guideline 6.5: Under no conditions shall EN 14908-1 gateway pass commands from a command-based system directly into a CNP network. Instead, these commands shall be mapped to EN 14908-5 standard and user functional blocks.

6.6 Shared-Media Considerations

A power-line channel and a radio-frequency channel that contain devices within communicable range of several network tools are two examples of shared-media channels. When two or more network tools share such a medium, messages can leak between one tool and devices belonging to another tool. If the tools and installers do not directly co-ordinate their activities, the tools and devices shall follow conventions to avoid conflicting network changes or installing the wrong devices. The following guidelines apply to devices on a shared medium. The term “shared media” refers not only to communications-medium sharing but also uncoordinated network-management activities as described above. It also refers to open, shared media, like a power-line channel or RF channel; and closed, shared media, like a twisted-pair channel.

If a foreign network tool inadvertently acquires a device and installs it with network-management authentication, the device’s owner is unable to reclaim the device over the network. To prevent this, devices intended for installation on shared media shall provide some means for locally causing the device to go unconfigured. An unconfigured device does not have a network address. For example, invoking a function to cause the device to "go unconfigured" should unconfigure a device and resets its authentication key, thereby allowing the device’s owner to reclaim the device by reinstalling it. A typical implementation requires a pushbutton, often the service-pin button, to be pressed and held for 15 seconds, to cause the device to unconfigure itself.

Guideline 6.6A: A device that is intended for installation on shared media shall provide some means for locally causing the device to go unconfigured.

Since the service-pin message can be received by foreign network tools, a means is required for a network integrator to determine if the correct device was installed upon installing a new device. This can be provided by a wink function as defined in the EN 14908-1 protocol. The wink function allows a network integrator to physically confirm that an intended device has been installed. Device winking, whether due to the installation protocol itself, or post-installation testing, may cause activity in an unintended device if an incorrect device was installed on a foreign network sharing the shared-media channel. Since the existence of the local network may not even be known to people working on the foreign network, the effects of winking shall be benign. For example, an LED may flash on a device, but a motor should not be powered on.

A device that responds to a wink command shall automatically stop winking after a maximum of 30 seconds. A device shall not require special means, like the receipt of a second wink command, to leave the wink state.

Guideline 6.6B: A device that is intended for installation on shared media shall support the wink function as described in 6.6 and shall provide a wink that does not create a potentially dangerous or costly situation if invoked at any arbitrary time in the operational life of the device.

Bibliography

[1] EN 14908-2 *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 2: Twisted Pair Communication*

[2] EN 14908-3 *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 3: Power Line Channel*

[3] EN 14908-4 *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 4: IP Communication*

This page has been intentionally left blank

BSI Group Headquarters 389 Chiswick High Road, London, W4 4AL, UK
Tel +44 (0)20 8996 9001
Fax +44 (0)20 8996 7001
www.bsigroup.com/standards