

SEMI E54.16-1106 (Withdrawn 0524) SPECIFICATION FOR SENSOR/ACTUATOR NETWORK COMMUNICATIONS FOR LONWORKS

This Standard was technically approved by the Information & Control Global Technical Committee. This edition was approved for publication by the global Audits and Reviews Subcommittee on April 26, 2024. Available at www.semiviews.org and www.semi.org in May 2024; originally published July 2005; previously published December 2011.

NOTICE: The section symbol (§) is used to reference a numbered section, paragraph, or header and all subordinate paragraphs, headers, and embedded material (i.e., EXCEPTIONs, lists) therein. The paragraph symbol (§) is used to reference a particular numbered paragraph and embedded material therein. When duplicated (i.e., §§ and §§) the symbols are used to reference more than one section or paragraph, respectively.

NOTICE: This Standard was balloted and approved for withdrawal in 2024.

1 Purpose

1.1 This Specification defines a communication specification based on the ANSI/EIA/CEA 709.1 Control Networking Protocol (LonWorks) to enable communications between intelligent devices on a sensor/actuator network (SAN) that operate in accordance with SEMI-specified device models (common and device specific) on semiconductor manufacturing equipment.

1.2 This Standard specifies a mapping of the SEMI Common Device Model (CDM) onto LonWorks technology using the LonMark interoperability guidelines established for LonWorks devices. The LonMark International Association will review and approve the enhanced SEMI LonMark functional profiles presented in this network communication Document and incorporate the required SEMI profiles into the LonMark Interoperability Guidelines.

1.3 *Background and Motivation* — The LonWorks communications system provides interconnection of smart control devices such as sensors, actuators, and controllers in a fast-response time, low-cost network for industrial use. LonWorks enables multiple devices to share a single network, thereby significantly reducing the point-to-point wiring between controllers, sensors, and actuators. The LonWorks Network Communications Standard (NCS) is based on the seven-layer ANSI/EIA/CEA 709.1 (LonWorks) protocol, implemented by the Neuron Chip, a physical layer transceiver, and an optional host processor. The ANSI/EIA/CEA 709.1 (LonWorks) control networking protocol was developed by Echelon and may be freely licensed for implementation on any hardware platform. The SEMI NCS for LonWorks is based on the LonMark interoperability guidelines, which provide a framework for interoperable use of the ANSI/EIA/CEA 709.1 (LonWorks) protocol at layers 1 through 6, as well as at the application layer. Where the LonMark interoperability guidelines do not provide the functionality required by the SEMI CDM and the SEMI Specific Device Models (SDM), the guidelines are extended with SEMI-specific enhanced profile requirements.

2 Scope

2.1 This Standard specifies a SAN communications specification standard, based on the ANSI/EIA/CEA 709.1 control networking protocol (LonWorks) specification, that enables communication with SAN devices configured according to the SEMI SAN CDM and appropriate SEMI SAN SDM specifications.

2.2 This Standard specifies the use of LonWorks technology for services that compliant intelligent devices must support in order to exchange information over this semiconductor equipment sensor/actuator network.

2.3 This Standard specifies the utilization of LonWorks technology to present externally visible device structure and behavior, specified in the CDM and appropriate SDMs, on a LonWorks network.

2.4 This Specification is used in conjunction with the SEMI SAN CDM Specification and one or more SEMI SAN SDM Specifications (e.g., for a mass flow controller). Together, the model documents describe the externally visible data structures and behavior of devices using LonWorks technology in a SEMI-compliant SAN system.

2.5 This Specification, together with the SAN CDM, one or more SAN SDM Specifications, the ANSI/EIA/CEA 709.1 (LonWorks) protocol specification, and the LonMark interoperability guidelines, specifies requirements for SEMI SAN implementations based on LonWorks technology.

NOTICE: SEMI Standards and Safety Guidelines do not purport to address all safety issues associated with their use. It is the responsibility of the users of the Documents to establish appropriate safety and health practices, and determine the applicability of regulatory or other limitations prior to use.

3 Limitations

3.1 This Standard specifies a semiconductor equipment SAN based solely on LonWorks technology and is a companion document to the ANSI/EIA/CEA 709.1 (LonWorks) control networking protocol specification; thus, a complete specification of this Standard necessarily includes the LonWorks specifications. There are other semiconductor equipment SAN communications options. The specifications for these options are not included here.

3.2 This Standard specifies enhancements that provide additional capabilities over and above those currently required by the LonMark Interoperability Guidelines. In order to avoid document consistency problems, information in the LonWorks technology specifications that relate to this Standard is not repeated in this Standard. This Standard is limited to describing enhancements or limitations of LonWorks technology and the LonMark interoperability guidelines that are imposed by this Specification.

3.3 A complete specification of the conformance testing procedure shall include the applicable LonMark interoperability conformance testing specification. Conformance testing shall include enhancements to the LonMark Interoperability Guidelines required by this Standard Specification.

4 Referenced Standards and Documents

4.1 *SEMI Standards and Safety Guidelines*

SEMI E54.1 — Standard for Sensor/Actuator Network Common Device Model

SEMI E54.3 — Specification for Sensor/Actuator Network Specific Device Model for Mass Flow Device

SEMI E54.10 — Specification for Sensor/Actuator Network Specific Device Model for an In Situ Particle Monitor Device

SEMI E54.11 — Specification for Sensor/Actuator Specific Device Model for an Endpoint Device

SEMI E54.18 — Specification for Sensor/Actuator Network Specific Device Model for Vacuum Pump Device

4.2 *ANSI® Standards*¹

ANSI/EIA/CEA 709.1 — LonWorks Networking-Protocol Specification

4.3 *LonMark Documents*²

LonMark Application-Layer Interoperability Guidelines

LonMark Layer 1-6 Interoperability Guidelines

NOTICE: Unless otherwise indicated, all documents cited shall be the latest published versions.

5 Terminology

5.1 *Abbreviations and Acronyms*

5.1.1 *APDU* — application protocol device unit

5.1.2 *CDM* — common device model

5.1.3 *CP* — configuration parameter

5.1.4 *DM* — device manager

5.1.5 *DS* — device status

¹ American National Standards Institute, 25 West 43rd Street, New York, NY 10036, USA; Telephone: +1.212.642.4900, Fax: +1.212.398.0023, <http://www.ansi.org>

² LonMark International, 550 Meridian Avenue, San Jose, CA 95126, USA; Telephone: +1.408.938.5266; Fax: +1.408.790.3838; <http://www.lonmark.org>

5.1.6 *NCS* — network communications standard

5.1.7 *NV* — network variable

5.1.8 *NVI* — network variable input

5.1.9 *NVO* — network variable output

5.1.10 *SAC* — sensor actuator controller

5.1.11 *SAN* — sensor/actuator network

5.1.12 *SCPT* — standard configuration parameter type

5.1.13 *SDM* — specific device model

5.2 *Device Component Definitions* — as this Standard defines the presentation or mapping of CDM data structure and behavior over a network, it makes use of many of the terms in the SEMI E54.1 CDM Standard. Table 1 provides a mapping of fundamental terminology of the CDM Standard into this Standard and the LonWorks definitions. The symbol ‘=’ indicates that the definition is used exactly as specified on the CDM Specification.

5.3 In the following sections, additional clarification of some of these terms is provided in the context of the ANSI/EIA/CEA 709.1 (LonWorks) protocol.

Table 1 Mapping of CDM to NCS Terminology

<i>CDM Term</i>	<i>NCS Equivalent</i>	<i>LONWORKS Equivalent</i>
Device	=	Device or node
Device Model	=	Device interface
Object	=, Class	Functional profile, Class
Instance	=	Functional block
Attribute	=	Network variable or configuration property
Behavior	=	=
Service	=	Network variable function or application message
State Diagram	=	=
Byte	=	=
Nibble	=	=
Character String	=	String of ASCII characters or string of international characters

5.3.1 *attribute* — attributes are either input network variables (NVI), output network variables (NVO), or configuration properties (CP). Input and output network variables and configuration properties may be read and/or written by the device itself, and all attributes may be polled over the network. Additionally, input network variables and configuration properties may be updated over the network, and the receipt of such an update may cause an event to be propagated to the device’s application layer. This corresponds to a read and write (RW) attribute of the object owning the network variable. Output network variables may not be updated over the network. This corresponds to a read only (RO) attribute of the object owning the network variable. When the device itself updates one of its output network variables, the value of that variable may be propagated over the network to destination addresses determined at installation time. Finally, configuration properties are attributes typically stored in non-volatile memory and preserved across device resets and power cycles.

5.3.2 *behavior* — generic object behavior is specified by the LonMark Application-Layer Interoperability Guidelines. Additional object-specific behavior is specified by means of functional profiles.

5.3.3 *device* — a device (or node) typically consists of one network transceiver which implements the physical layer of the ANSI/EIA/CEA 709.1 (LonWorks) protocol, one Neuron chip with associated firmware which implements the other layers of the LonTalk protocol, and input/output hardware implementing the physical interface of the device to

external sensor and/or actuator hardware. A LonWorks device may optionally contain a host processor and associated software or firmware which implements the application layer of the ANSI/EIA/CEA 709.1 (LonWorks) protocol.

5.3.4 device model — the device model comprises several elements which fully describe the external interface of the device for an interoperable network. The interface is made of the following pieces: a device manager (DM) object; a sensor/actuator/controller (SAC) object; functional blocks such as sensors, actuators, and controllers; individual network variables; and configuration properties.

5.3.5 instance — real devices may have zero or more instances of each of the defined functional blocks. Object instances are identified by means of an instance number within the device.

5.3.6 object — functional blocks defined as a set of one or more network variable inputs and/or outputs, implemented as standard network variable types (SNVT), and a set of configuration properties, implemented as standard configuration property types (SCPT). Functional blocks form the basis of interoperability at the application layer. The functional blocks describe standard formats for how information is input to, and output from, a device, and shared with other devices on the network.

5.3.7 service — request services are represented by ANSI/EIA/CEA 709.1 input network variables (NVI) being set and delivered to the device application. Notification services are represented by ANSI/EIA/CEA 709.1 output network variables being set by the device application and delivered over the network.

5.3.8 state diagram — in a LonWorks device, state is represented by the collection of values of local and network variables of the application program. Transitions between states are the result of external events (such as the receipt of a network variable update, or other I/O event), or internal events (such as the expiration of a timer).

5.4 LonWorks-Specific Definitions — in addition to the standard data type definitions for bit, nibble, byte, and character, the ANSI/EIA/CEA 709.1 (LonWorks) protocol defines a set of standard data representations for use as attribute values.

5.4.1 binding — network variables on the same or different devices may be associated together by means of a network management service known as binding. Binding is permitted only if all the network variables in the set are of the same data type. The values of network variables that are bound together are propagated over the network by the ANSI/EIA/CEA 709.1 (LonWorks) protocol. Table 2 shows the permitted combinations for updating and polling of network variables.

Table 2 Updating and Polling of Network Variables

<i>Network Variable Class</i>	<i>Update from Network</i>	<i>Update from Device</i>	<i>Poll from Network</i>	<i>Poll from Device</i>
Input	Yes	Yes	Yes	Yes ^{#2}
Output	No	Yes ^{#1}	Yes	No

#1 When the device updates one of its own output network variables, input network variables that are bound to this output network variable receive an update from the network.

#2 When the device polls one of its own input or configuration network variables, output network variables that are bound to this input or configuration network variable receive a poll from the network.

5.4.2 configuration properties — these are attributes of a functional block that are used to configure the application-specific behavior of the functional block, such as sensor gain and offset, linearization table, and sample rate. These attributes are typically updated when the device is installed, configured, or calibrated, and are stored in nonvolatile memory.

5.4.3 device interface — a device interface is a specification of one or more functional blocks, together with semantic definitions relating the behavior of the functional block(s) to the network variable values. The collection of functional blocks in a device corresponds to the SEMI SAN device-specific model for that device. Each type of device interface is identified by a standard program ID (SPID).

5.4.4 network variable — this is a network-visible data attribute of a device, with a well-defined data type. Network variables are either input variables, output variables, or configuration network variables. The value of a network variable may be updated either by the device itself, or over the network by some other device. This corresponds to the SEMI specific service SetAttribute operation. The value of a network variable may be polled over the network by

some other device, or retrieved by the device itself. This corresponds to the SEMI specific service GetAttribute operation.

5.4.5 functional block — 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 is an implementation of a functional profile. A device’s external interface documentation specifies the type identifiers of the functional blocks contained within the device. This documentation may be uploaded from the device, and completely specifies the functional profiles implemented by the device, as well as the network variables and configuration properties contained within each of the functional blocks. Refer to reference ‘LonMark Application-Layer Interoperability Guidelines,’ LonMark International in § 4.3.

5.4.6 functional profile — a functional profile is the definition of the attributes and behaviors of an abstract entity. Each type of functional block is identified by a functional profile number. A specific device type consists of instantiations of one or more of these functional profiles. A functional block is implemented as a collection of network variables and configuration properties.

5.4.7 Neuron — Neuron chip is the 8 bit hardware implementation of the ANSI/EIA/CEA 709.1 (LonWorks) control networking protocol.

5.4.8 standard configuration property types (SCPTs) — these data types, also known as SCPTs, provide a data type definition and a semantic behavior for the configuration properties of functional blocks. A list of all available SCPTs and details of their definitions is provided in the SCPT Master List and Programmer’s Guide.

5.4.9 standard network variable types (SNVTs) — these data types, also known as SNVTs, facilitate interoperability by providing a well-defined interface for communication between devices made by different manufacturers. A device may be installed in a network, and logically connected to other devices via network variables, as long as the data types match. A list of all available SNVTs and details of their definitions is provided in the SNVT Master List and Programmer’s Guide.

6 Communication Protocol High Level Structure

6.1 The ANSI/EIA/CEA 709.1 (LonWorks) protocol is based on a seven-layer architecture. At each layer, there is a description of the services provided within that layer. The high level protocol architecture is shown in Figure 1.

6.1.1 Figure 1 represents a conceptual view of the device architecture. Implementations typically use the Neuron chip and its associated firmware, which provide a conforming implementation of layers 2–6. The LonMark Interoperability Guidelines specify the protocol options to be used, most specifically at the physical layer (network transceivers) and at the application layer (object model).

6.2 Physical Layer — The device shall employ one of the LonMark-approved physical channels as specified in the LonMark Layer 1–6 Interoperability Guidelines. LonWorks-based SEMI SAN-compliant devices shall use, by default, a two-pin screw-terminal open pluggable connector (Weidmüller-Klippon SL2, Phoenix Combicon, or equivalent) for the network connection. The default connector specification may be overridden for specific device types if special requirements apply; any such overrides shall be noted in § 9, Specific Device Type Information, of this Standard. This connection is polarity-insensitive. The requirements of semiconductor equipment may be met by one of the twisted pair channel specifications listed below.

6.2.1 TP/XF-1250 Twisted Pair — This twisted-pair channel operates at a bit rate of 1,250 kbps and supports a bus topology using transformer-coupled transceivers.

6.2.2 TP/FT-10 Twisted Pair — This twisted-pair channel operates at a bit rate of 78 kbps and supports both free topology and bus topology wiring, as well as optional link power.

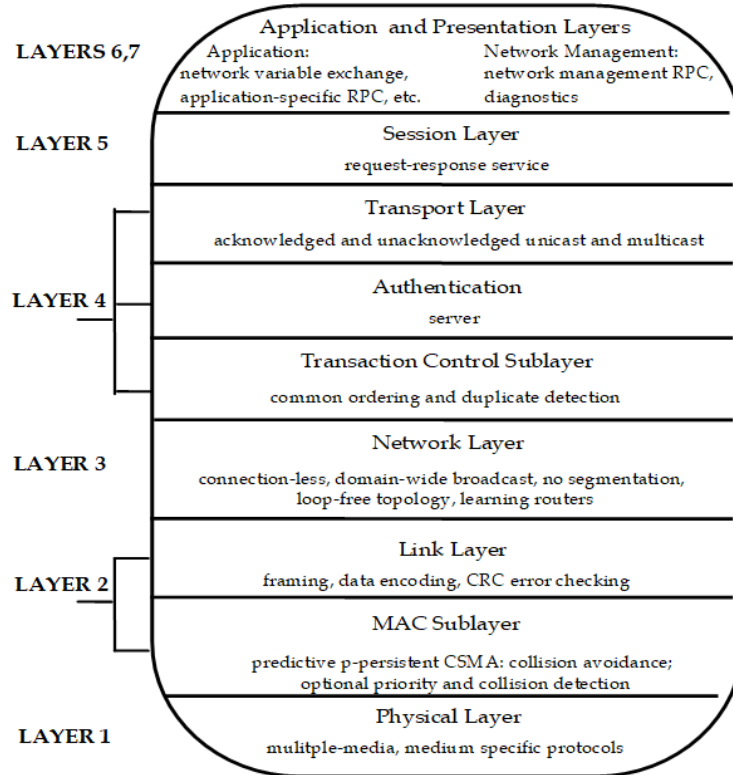


Figure 1
Layered View of the ANSI/EIA/CEA 709.1 (LonWorks) Protocol

6.2.3 The LonMark Layer 1-6 Interoperability Guidelines provide specific details of the characteristics of these transceivers. This Standard also provides specifications of wiring types and interconnection topologies to be used for guaranteed device interoperability. Note that the ANSI/EIA/CEA 709.1 (LonWorks) protocol supports heterogeneous networks. Devices with dissimilar transceivers may be interconnected and communicate via routers or repeaters. Similarly, routers and repeaters may be used to extend a physical channel beyond the device count, wire length, or other physical limitations imposed by the chosen transceiver.

6.2.4 Multiple physical layer protocols and data encoding methods are used in the ANSI/EIA/CEA 709.1 (LonWorks) protocol. Differential Manchester encoding is used on twisted pair physical layers.

6.3 *Link Layer* — The device shall comply with the ANSI/EIA/CEA 709.1 (LonWorks) protocol link layer specification. This layer includes the media access control sub-layer. For a number of reasons, including simplicity and compatibility with the multicast protocol, the ANSI/EIA/CEA 709.1 (LonWorks) protocol supports a simple connectionless service. Its functions are limited to framing, frame encoding, and error detection.

6.3.1 *Media Access Control (MAC) Sub-layer* — In order to deal with a variety of media in the potential absence of collision detection, the MAC sub-layer employs a collision avoidance algorithm called predictive *p*-persistent carrier sense, multiple access (CSMA).

6.4 *Network Layer* — The device shall comply with the ANSI/EIA/CEA 709.1 (LonWorks) protocol network layer specification. This layer handles packet delivery within a single domain, with no provisions for inter-domain communication. The network service is connectionless, unacknowledged, and supports neither segmentation nor re-assembly of messages. The routing algorithms employed by the network layer to learn the topology assume a tree-like network topology; routers with configured tables may operate on topologies with physical loops, as long as the communication paths are logically tree-like. In this configuration, a packet may never appear more than once at the router on the side on which the packet originated. The unicast routing algorithm uses learning for minimal overhead and no additional routing traffic. Use of configured routing tables is supported for both unicast and multicast addresses.

6.5 Transport Layer — The device shall comply with the ANSI/EIA/CEA 709.1 (LonWorks) protocol transport layer specification. The heart of the protocol hierarchy is the Transport and Session layers. A common Transaction Control sub-layer handles transaction ordering and duplicate detection for both layers. The transport layer is connectionless and provides reliable message delivery to both single and multiple destinations. Authentication of the message sender's identity is provided as an optional feature. The authentication server requires only the Transaction Control sub-layer to accomplish its function. The transport and session layer messages may be authenticated using all of the ANSI/EIA/CEA 709.1 addressing modes other than broadcast. The transport layer supports end-to-end acknowledged service and an unacknowledged/repeated service.

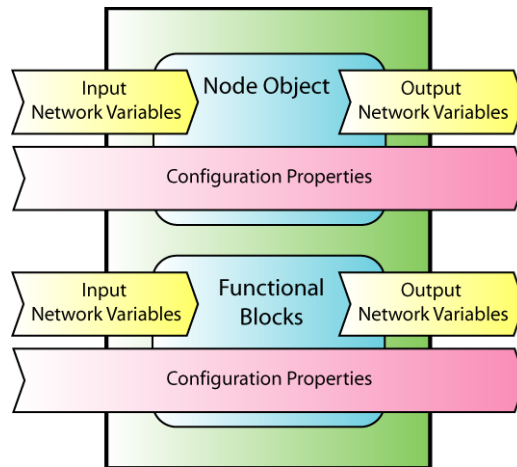
6.6 Session Layer — The device shall comply with the ANSI/EIA/CEA 709.1 (LonWorks) protocol session layer specification. This layer implements a simple Request-Response mechanism for access to remote servers. This mechanism provides a platform upon which application-specific remote procedure calls can be built. The ANSI/EIA/CEA 709.1 (LonWorks) network management protocol, for example, is dependent on the Request-Response mechanism in the Session layer, even though it accesses the protocol via the application layer interface.

6.7 Presentation Layer — The device shall comply with the ANSI/EIA/CEA 709.1 (LonWorks) protocol presentation layer specification. The Presentation layer and the Application layer taken together form the foundation of interoperability for LonWorks devices. The application layer provides all the usual services for sending and receiving messages, but it also contains the concept of network variables. The presentation layer provides information in the application protocol data unit (APDU) header for how the APDU is to be interpreted for network variable updates. This application-independent interpretation of the data allows data to be shared among devices without prior arrangement. With agreement on which network variables are to be used for sensors, actuators, etc., intelligent components from different manufacturers may work together without prior knowledge of each other's characteristics.

6.8 Application Layer — At the application layer, interoperability between LonWorks -based devices is facilitated through the use of functional blocks and SNVTs. Functional blocks build upon network variables and provide a concise application layer interface that incorporates semantic meaning for specific device functions. Functional blocks not only define which SNVTs to use to convey data, but also provide semantic meaning about the information being communicated. To aid in the specification of specific device models with well-defined functional behavior, functional block interfaces and semantics are defined by functional profiles. The Application Layer also includes the LonWorks file transfer protocol, which provides segmentation and reassembly of arbitrary length files of data. This service may be used to get and set object attributes that exceed the network variable size limit of 31 bytes.

6.8.1 Object Models — The ANSI/EIA/CEA 709.1 (LonWorks) protocol provides an object-oriented specification for defining and addressing network variables and configuration properties, which are the representation of object attributes and events. The device shall comply with the object model specifications defined in § 7 of this Standard.

6.8.2 Functional Block Structure — The LonMark Application-Layer Interoperability Guidelines define a structure for functional profiles. Each functional profile may have a set of mandatory network variables, a set of optional network variables, a set of configuration properties (both mandatory and optional), and a manufacturer-defined section, which may be used for non-interoperable extensions to the profile. This is illustrated in Figure 2. This notation is defined in the LonMark Application-Layer Interoperability Guidelines.



#1 Diagram notation, the arrow-like symbol used in Figure 2 is defined in the LonMark Application-Layer Interoperability Guidelines.

Figure 2
LonMark Object Structure

6.8.3 The LonMark Application-Layer Interoperability Guidelines provide for the definition of SNVTs, SCPTs, and Functional Profiles. In the mapping of the SEMI CDM to the LonMark object structure in § 7, extensions to the current SNVT list and LonMark interoperability guidelines are marked with an asterisk (*). Functional profile numbers are specified by the guidelines; a device may consist of one instance of a Node Object type, and one or more instances of other functional profiles.

6.9 *Network Management* — The ANSI/EIA/CEA 709.1 (LonWorks) protocol defines a complete network management and diagnostic protocol for LonWorks devices. This protocol is a layer above the Session layer (request/response service) and provides mechanisms for application downloading, device address assignment, distribution of destination addresses for implicit messaging, router configuration, and device-level diagnostics. The LonMark Application-Layer Interoperability Guidelines define a device management layer for functional blocks.

7 Required and Optional Object Types

7.1 The LonMark guidelines do not require any specific objects to exist in a device in order to be a compliant LonMark device, except that a Node object functional block is required for devices that contain multiple functional blocks (SEMI compliant devices will typically have a Node object functional block). New LonMark standard profiles are defined in this Standard to identify and describe functional blocks that shall exist in devices that are to be interoperable and interchangeable on a LonMark SEMI compliant SAN network.

7.1.1 LonMark International Association publishes functional profiles for various sensor, actuator, and controller objects. A specific device may be implemented using functional blocks based on these profiles. The CDM Specification additionally identifies two functional blocks (namely the DM and SAC objects) that must exist in all SEMI compliant SAN devices. The required functional profiles for a SEMI compliant SAN device utilizing the network communication specification described herein necessarily comprise, at minimum, the union of the LonMark functional profile requirements and the CDM Specification requirements.

7.1.2 A list of required and optional object types is given in Table 3. Additional objects that are specified in a particular SDM are given identifiers in that SDM specification. The LonMark specific presentation information for these identifiers is given in § 9 of this Standard.

Table 3 CDM Required and Optional Object Types

<i>SEMI Object Name</i>	<i>LonMark SEMI Class ID/Instance ID^{#1}</i>	<i>CDM Tag^{#2}</i>	<i>LonMark Functional Profile Name</i>	<i>Required By LonMark^{#1}</i>	<i>Required By CDM^{#2}</i>	<i>Required By NCS</i>
(DM) Device Manager ^{#4}	0/1	DmlO	Node Object	Yes	Yes	Yes
(SAC) Sensor/Actuator/Controller ^{#4}	0/1	SacIO	Node Object	Yes	Yes	Yes
Assembly	180.81/1 through i	Asm	Manufacturer-specific Members	No	No	No
Local Link	Turnaround Connection ^{#3}	Lnk	Turnaround Connection ^{#3}	No	No	No
Sensor-AI	180.11/1 through k	Sai	SFPTsemiSensorAI	No	No	No
Sensor-EI	180.22/1 through l	Sei	SFPTsemiSensorEI	No	No	No
Sensor-BI	180.18/1 through m	Sbi	SFPTsemiSensorBI	No	No	No
Actuator-AO	180.31/1 through n	Aao	SFPTsemiActuatorAO	No	No	No
Actuator-EO	180.34/1 through o	Aeo	SFPTsemiActuatorEO	No	No	No
Actuator-BO	180.33/1 through p	Abo	SFPTsemiActuatorBO	No	No	No
Controller	180.51/1 through q	C	SFPTsemiController	No	No	No
Sensor-BI-TH	180.19/1 through s	Sbith	SFPTsemiSensorBITH	No	No	No

#1 LonMark groups SEMI Class IDs as reference 180 and designates specific object name categories as .XX. Refer to LonMark Application Layer Interoperability Guideline and <http://types.lonmark.org>.

#2 Refer to SEMI E54.1, CDM Specification for further information.

#3 A turnaround connection is not a profile, but is a protocol-specific method for creating a link.

#4 The LonMark Application Layer maps the SEMI DM and SAC objects to the LonMark Node object.

7.1.3 Service Requests Code — Most service requests are implemented as network variable updates addressed to the network variable corresponding to the specified attribute. The exceptions to this mapping are requests that require more than a 31 byte response. In that case, the request is implemented as a network variable update containing a file request to an **nviSemiReq** input to the Node Object functional block of a device. This input is defined by the **SNVT_semi_req** type as follows:

```
typedef enum {
    struct {
        CMD_RESET                = 1,
        CMD_ABORT                 = 2,
        CMD_RECOVER               = 3,
        CMD_GET_ATTRIBUTE         = 4,
        CMD_SET_ATTRIBUTE         = 5,
        CMD_OERATE                = 6,
        CMD_RESTOTE_DEFAULT       = 7,
        CMD_PUBLISH_ATTRIBUTE     = 8,
        CMD_LOCK                  = 9,
        CMD_UNLOCK                = 10,
        CMD_GET_EXCEPTION_QUEUE   = 11,
        CMD_CLEAR_EXCEPTION_QUEUE = 12,
        CMD_EXECUTE               = 13,
        CMD_PERFORM_DIAGNOSTICS   = 14,
```

```

} semi_request_t;
typedef struct {
    unsigned int flblock_index;
    semi_request_t service_code;
    unsigned int parameter;           // Optional parameter
} SNVT_semi_req;

```

7.1.4 A device accepts a SNVT_semi_req request by updating the nvoSemiStat and nvoFileStat outputs of the Node Object function block. The nvoSemiStat output is defined by the SNVT_semi_status type as follows:

```

typedef enum {
    struct {
        STAT_SUCCESS                = 0,
        STAT_UNSUPPORTED_SERVICE    = 1,
        STAT_UNSUPPORTED_ATTRIBUTE  = 2,
        STAT_SET_ATTRIBUTE_ERROR    = 3,
        STAT_GET_ATTRIBUTE_ERROR    = 4,
        STAT_CALIBRATION_ERROR      = 5,
        STAT_SERVICE_REQUEST_ERROR  = 6,
        STAT_DIAGNOSTIC_ERROR       = 7,
    } semi_status_t;
typedef struct {
    semi_status_t status;
    unsigned long selected_file;
    struct address {           // Address of the requesting device
        unsigned short domain_id[6];
        unsigned short domain_length;
        unsigned short subnet;
        unsigned short node;
    }
} SNVT_semi_status;

```

7.1.5 If the nvoSemiStat and nvoFileStat outputs indicate the request was accepted, the requesting device then fetches the requested data by transferring the file with the file index reported by the nvoSemiStat response.

7.1.6 *Object Attributes* — The SEMI GetAttribute and SEMI SetAttribute service requests are implemented as network variable fetch, poll, and update requests addressed to the network variable corresponding to the specified attribute. This is appropriate when responses greater than 31 bytes are not required. A GetAttribute service request may be addressed directly to any network variable as a LonTalk request message, using the LonTalk protocol network management NV fetch mechanism. A GetAttribute service request may also be addressed to an output network variable as a LonTalk NV poll message, using the NV selection mechanism. A SetAttribute service request may be addressed to an input network variable as a LonTalk NV update message, using the NV selection mechanism. The confirmation of a SetAttribute (network variable update) is provided by the acknowledged service of the LonTalk protocol transport layer.

7.1.7 Sequence Numbers — Each network variable in a functional block is identified by means of a self-documentation string stored in the device's memory. This string contains the functional block index of the functional block to which this variable belongs, and the Sequence Number of the network variable within its functional block. For the LonWorks NCS, this sequence number is identical to the numerical sequence number specified by the CDM tag. As an example, using the DM object instance declared as the second object instance in the device it has an Instance Id of 1. The DM attribute Standard Revision Level has the tag DmA2. The self-documentation string for this network variable is, therefore, specified as '@1|2.'

7.1.8 Publish Service — The Publish notification service is implicit when an output network variable is updated. The device propagates the value of the output network variable (equivalent to a read-only attribute) to any input network variable(s) to which it may be bound. The ANSI/EIA/CEA 709.1 (LonWorks) protocol only supports propagation of output network variables. In CDM terminology, this means that only read-only attributes may be published. If a specific device model requires publication of a read/write attribute, an output network variable whose value mirrors the value of the input (read/write) network variable may be introduced to the object definition.

7.2 Sensor/Actuator/Controller Object (*) — The SEMI CDM SAC object coordinates the functionality of Sensor, Actuator, and Controller objects in the device. The SAC object is mapped to the LonMark Node object functional block. Each device must support one (and only one) Node object functional block. The SAC functional block as well as its common required and optional attributes, services and behavior are described in SEMI E54.1, the CDM Standard Specification. Extensions to the current SNVT list and LonMark Interoperability Guidelines are marked with an asterisk (*).

7.2.1 Extensions to the Node Object functional profile are defined in the LonMark interoperability guidelines, which form part of the LonMark Functional Profiles for SEMI SAN-compliant devices based on LonWorks technology. Table 4 summarizes the network variables that implement the attributes of the SAC object. Table 5 summarizes the services implemented by the SAC object.

Table 4 SAC Object Network Variables

<i>SAC Object</i> <i>Profile ID = 00, Instance ID = 01</i>				
<i>Sequence Number</i>	<i>Name</i>	<i>Storage Class</i>	<i>CDM Tag</i>	<i>Standard NV or CP Data Type</i>
1	Last Calibration Date	CP	SacA1	SCPTlastCalibrationDate (*) ^{#2}
2	Next Calibration Date	CP	SacA2	SCPTnextCalibrationDate (*) ^{#2}
3	Expiration Timer	NVO	SacA3	SNVT_time_hour
4	Expiration Warning Enable	CP	SacA4	SCPTexpirationWarningEnable (*) ^{#2}
5	Run Hours	NVO	SacA5	SNVT_time_hour
128	Attribute ID ^{#1} (*) ^{#2}	NVI	SacA65	SNVT_member (*) ^{#2}

#1 This input is used to specify an attribute for the Restore Default and Publish Attribute services.

#2 (*) LonMark interoperability extension. Refer to § 6.8.3 for further definition.

Table 5 SAC Object Services

<i>SAC Object</i> <i>Profile ID = 00, Instance ID = 01</i>				
<i>Service Request Code</i>	<i>Service Name</i>	<i>CDM Tag</i>	<i>Request Parameters</i>	<i>Result Parameters</i>
1	Reset	SacS1	Node Object RQ_RESET	
2	Abort	SacS2	Node Object RQ_ABORT (*) ^{#2}	
3	Recover	SacS3	Node Object RQ_RECOVER (*) ^{#2}	
4	GetAttribute	SacS4	Read NV or CP	NV or CP Value
5	SetAttribute	SacS5	Write NV or CP	
6	Operate	SacS6	Node Object RQ_NORMAL	
7	Restore Default ^{#1}	SacS7	Node Object RQ_RESTORE (*) ^{#2}	NVO Value
8	Publish Attribute ^{#1}	SacS8	Node Object RQ_PROPAGATE (*) ^{#2}	NVO Value

#1 These services may use the new Attribute ID input to the Node Object functional block.

#2 (*) LonMark interoperability extension. Refer to § 6.8.3 for further definition.

7.3 DM Object (*) — The DM object is the device component responsible for managing and consolidating the device operation. The DM object is mapped to the LonMark Node object functional block. Each device must support one (and only one) Node Object functional block. The DM functional block as well as its common required and optional attributes, services and behavior are described in SEMI E54.1, the CDM Standard Specification. Extensions to the current SNVT list and LonMark Interoperability Guidelines are marked with an asterisk (*).

7.3.1 The SEMI CDM DM functional block, which is mapped to the LonMark Node object, combines attributes of device self-documentation with an exception reporting mechanism. A new functional profile is, therefore, defined with the following mandatory configuration parameters (CPs), network variables and behaviors. This functional profile forms part of the LonMark Functional Profiles for SEMI SAN-compliant devices based on LonWorks. Table 6 summarizes the network variables that implement the attributes of the Device Manager functional block.

Table 6 DM Object Network Variables

<i>DM Object</i> <i>Profile ID = 00, Instance ID = 01</i>				
<i>Sequence Number</i>	<i>Name</i>	<i>Storage Class</i>	<i>CDM Tag</i>	<i>Standard NV or CP Data Type</i>
11	Device Type	CP	DmA1	SCPTdeviceType (*) ^{#5}
12	Standard Revision Level	CP	DmA2	SCPTdeviceRevLevel (*) ^{#5}
13	Device Manufacture Identifier	CP	DmA3	SCPTdeviceManufacturer (*) ^{#5}
14	Manufacturer Model Number	CP	DmA4	SCPTdeviceModelNumber (*) ^{#5}
15	S/W or F/W Revision Level	CP	DmA5	SCPTappRevLevel (*) ^{#5}
16	Hardware Revision Level	CP	DmA6	SCPTHardwareRevLevel (*) ^{#5}
17	Serial Number	CP	DmA7	SCPTserialNumber
18	Device Configuration	CP	DmA8	SCPTdeviceConfiguration (*) ^{#5}
19	Device Status	NVO	DmA9	SNVT_dev_status (*) ^{#5}
20	Reporting Mode	CP	DmA10	SCPT_rept_mode (*) ^{#5}
21	Exception Status Report Interval	CP	DmA11	SCPT_exc_sts (*) ^{#5}
22	Exception Status	NVO	DmA12	SNVT_alarm_2

Sequence Number	Name	Storage Class	CDM Tag	Standard NV or CP Data Type
23	Exception Detail Alarm	NVO	DmA13	SNVT_exc_detail (*) ^{#5}
24	Exception Detail Warning	NVO	DmA14	SNVT_exc_detail (*) ^{#5}
25	Visual Indicator	CP	DmA15	SCPTvisualIndicator (*) ^{#5}
26	Alarm Enable	NVI	DmA16	SNVT_obj_request
27	Warning Enable	NVI	DmA17	SNVT_obj_request (*-add RQ_WARNING_NOTIFY_ENABLED and RQ_WARNING_NOTIFY_DISABLED) ^{#5}
28	Exception Detail Type	CP	DmA18	SCPTexceptionDetailType (*) ^{#5}
29	Exception Detail Alarm Queue	NVO	DmA19	SNVT_exc_detail (*) ^{#5}
30	Exception Detail Warning Queue	NVO	DmA20	SNVT_exc_detail (*) ^{#5}
31	Date and Time	NVO	DmA21	SNVT_time_p (*) ^{#5}
32	Date and Time Type	CP	DmA22	SCPTdateTimeType (*) ^{#5}
66	Test ID ^{#2} (*) ^{#5}	NVI	DmA34	SNVT_test_id (*) ^{#5}
67	Password ^{#3} (*) ^{#5}	NVI	DmA35	SNVT_password (*) ^{#5}
68	Exception Queue ^{#4} (*) ^{#5}	NVI	DmA36	SNVT_exc_que (*) ^{#5}
128	Attribute ID ^{#1} (*) ^{#5}	NVI	DmA33	SNVT_member (*) ^{#5}

#1 This input is used to specify an attribute for the Restore Default and Publish Attribute services.

#2 This input is used to specify a test ID for the Perform Diagnostics service.

#3 This input is used to specify a password for the Unlock service.

#4 This input is used to specify a password for the Get and Clear Exception Queue services.

#5 (*) LonMark interoperability extension. Refer to § 6.8.3 for further definition.

7.3.2 *DM Functional Block Services* — Table 7 summarizes the services implemented by the DM functional block.

Table 7 DM Object Services

DM Object Profile ID = 00, Instance ID = 01				
Service Request Code	Service Name	CDM Tag	Request Parameters	Result Parameters
1	Reset	DmS1	Node Object RQ_RESET	
2	Abort	DmS2	Node Object RQ_ABORT (*) ^{#1}	
3	Recover	DmS3	Node Object RQ_RECOVER (*) ^{#1}	
4	GetAttribute	DmS4	Read NV or CP	NV or CP Value
5	SetAttribute	DmS5	Write NV or CP	
13	Execute	DmS6	Node Object RQ_ENABLE	
14	Perform Diagnostics	DmS7	Node Object RQ_SELF_TEST	
8	Publish Attribute	DmS8	Node Object RQ_PROPAGATE (*) ^{#1}	NVO Value
9	Lock	DmS9	Node Object RQ_LOCK (*) ^{#1}	
10	Unlock	DmS10	Node Object RQ_UNLOCK (*) ^{#1}	
11	Get Exception Queue	DmS11	Node Object RQ_GET_EXC_QUEUE (*) ^{#1}	
12	Clear Exception Queue	DmS12	Node Object RQ_CLEAR_EXC_QUEUE (*) ^{#1}	

#1 (*) LonMark interoperability extension. Refer to § 6.8.3 for further definition.

7.3.3 The Publish (DmS8) notification service for the DM exception status is implemented when an RQ_PROPAGATE request is received on the nviRequest input to the Node Object functional block. A specific network variable may be specified on the SNVT_member input. This causes the value of this network variable to be propagated across the network to other network variable(s) to which it may be bound. The implementation of the Device Manager functional block updates the specified output network variable according to the conditions specified by the Reporting Mode and Exception Status Reporting Interval configuration properties of the functional block.

7.3.4 *DM Object Configuration Properties* — The DM object has configuration properties to control exception reporting as shown in Table 8. These parameters are SCPTs.

7.3.4.1 The type SCPT_rept_mode(*) contains two four-bit fields specifying the reporting method for alarms and warning conditions. For example, in Neuron C, the application programming language used on the Neuron Chip, the declaration of SCPT_rept_mode is as follows:

```
typedef enum {
    REP_REQUEST                = 0,
    REP_REQ_LATCHED            = 1,
    REP_EVT_TRIGD_ON           = 2,
    REP_EVT_TRIGD_ONOFF        = 3,
    REP_TIME_TRIGD             = 4,
    REP_EVT_ON_TIME_TRIGD      = 5,
    REP_EVT_ONOFF_TIME_TRIGD   = 6,
} rept_mode_t;

typedef struct {
    rept_mode_t alarm_rept_mode :4;
    rept_mode_t warn_rept_mode  :4;
} SCPT_rept_mode;
```

7.3.4.2 The type SCPT_exc_sts(*) is a 16-bit value representing times from 0.00 to 655.35 seconds, with a resolution of 0.01 seconds. This parameter is optional. The default reporting mode is REP_REQUEST.

7.3.4.3 *DM Functional Block Output Network Variables* — The DM functional block of the CDM defines the attribute variable Device Status. The data type SNVT_dev_status is an enumeration. One field of the union is an enumeration, corresponding to the device status attribute defined by the table ‘Device Status Attribute Values’ of the CDM. The values of this type are defined in Table 8.

Table 8 Device Status Attribute Variable Enumeration Values

<i>Value</i>	<i>Enumeration Tag</i>
0	DS_UNKNOWN
1	DS_INIT_SELFTEST
2	DS_IDLE
3	DS_SELFTEST_EXCPT
4	DS_EXECUTING
5	DS_ABORT_1
6	DS_ABORT_2

7.3.4.4 The application programming language used to declare SNVT_dev_status is as follows:

```
typedef enum {
    DS_UNKNOWN                = 0,
    DS_INIT_SELFTEST         = 1,
    DS_IDLE                   = 2,
    DS_SELFTEST_EXCPT        = 3,
    DS_EXECUTING              = 4,
    DS_ABORT_1                = 5,
    DS_ABORT_2                = 6,
} dev_status_t;
```

7.3.4.4.1 The value DS_ABORT_1 corresponds to the *Abort from Idle or Executing* state, and the value DS_ABORT_2 corresponds to the *Abort from Initialized/Self Testing or Self Test Exception* state of the DM object.

7.3.4.5 The type SNVT_exc_que(*) contains four fields specifying the exception queue and exception elements to retrieve. For example, in Neuron C, the application programming language used on the Neuron chip, the declaration of SNVT_exc_que_mode is as follows:

```
typedef enum {
    REP_ALARM                = 0,
    REP_WARNING               = 1,
} que_t;

typedef struct {
    que_t exception_type :1;
    int element_number    :1;
    SNVT_state exception_list[13];
    int exception_clear    :1;
} SNVT_exc_que;
```

7.3.4.6 The type SNVT_exc_detail(*) is a sequence of three structures containing arrays. The ANSI/EIA/CEA 709.1 protocol limits the size of each of these arrays to 9 bytes, so that the type fits within the network variable size limit of 31 bytes. For example, in Neuron C, the declaration of SNVT_exc_detail is as follows:

```
typedef struct {
    u_char comn_exc_size;
    int    calibration      : 1;
    int    real_time        : 1;
    int    communic         : 1;
    int    RAM              : 1;
    int    EEPROM           : 1;
    int    EPROM            : 1;
    int    microproc        : 1;
    int    diagnostic       : 1;
} /*-----*/
```

```

int    resvdl          : 1;
int    reset           : 1;
int    notify_mfr      : 1;
int    maintenance     : 1;
int    power_inputV     : 1;
int    power_outptV    : 1;
int    power_resvd     : 1;
int    power_overC     : 1;

u_char comn_exc_dtl[7];

/*-----*/

u_char dev_exc_size;
u_char dev_exc_dtl[9];

/*-----*/

u_char mfr_exc_size;
u_char mfr_exc_dtl[9];
} SNVT_exc_detail;

```

7.4 SAC Functional Blocks — These functional blocks are necessarily specific to the specific device models. The LonMark Application-Layer Interoperability Guidelines provide a framework for defining functional profiles. Specific Device Models may employ these functional profiles, and/or may define their own functional profiles and Standard Network Variable Types for device-specific requirements. As long as the functional profile definition guidelines are followed, these profiles may be proposed to the LonMark Interoperability Association as new standard profiles.

7.5 Assembly Object — The Assembly object instances may be used to provide for grouping more than one attribute from one or more functional blocks in a device into a single data structure for communication over the LonWorks network. The presentation of Assembly object instance attributes are indicated in Table 9. The presentations of Assembly object services are indicated in Table 10.

Table 9 Assembly Object Network Variables

<i>Assembly (Asm) Object</i> <i>Profile ID = 180.81, Instance ID = 01 through i</i>				
<i>Sequence Number</i>	<i>Name</i>	<i>Storage Class</i>	<i>CDM Tag</i>	<i>Standard NV or CP Data Type</i>
1	Data	File	AsmA1	LONMARK file transfer NVs and messaging interface.

Table 10 Assembly Object Network Services

<i>Assembly (Asm) Object</i> <i>Profile ID = 180.81, Instance ID = 01 through i</i>				
<i>Service Request Code</i>	<i>Service Name</i>	<i>CDM Tag</i>	<i>Request Parameters</i>	<i>Result Parameters</i>
4	GetAttribute	AsmS4	File transfer read.	File transfer – Context Specific.
5	SetAttribute	AsmS5	File transfer write – Context Specific.	

7.6 Local Link Object — The Local Link object instances may be used to ‘link’ an attribute of one object instance to an attribute of another object instance. Refer to SEMI E54.1, CDM Standard for further explanation and use of this object. The presentation of Local Link object instance attributes are indicated in Table 11. The presentation of Local Link object services are indicated in Table 12.

Table 11 Local Link Object Network Variables

<i>Local Link (Lnk) Object</i>				
<i>Sequence Number</i>	<i>Name</i>	<i>Storage Class</i>	<i>CDM Tag</i>	<i>Standard NV or CP Data Type</i>
1	Source Object Class	NV Config Table/ Address Table ^{#1}	LnkA1	SNVTlnk_context. The structure of this SNVT in context specific.
2	Source Object Instance	NV Config Table/ Address Table ^{#1}	LnkA2	SNVTlnk_context. The structure of this SNVT in context specific.
3	Source Object Attribute	NV Config Table/ Address Table ^{#1}	LnkA3	SNVTlnk_context. The structure of this SNVT in context specific.
4	Destination Object Class	NV Config Table/ Address Table ^{#1}	LnkA4	SNVTlnk_context. The structure of this SNVT in context specific.
5	Destination Object Instance	NV Config Table/ Address Table ^{#1}	LnkA5	SNVTlnk_context. The structure of this SNVT in context specific.
6	Destination Object Attribute	NV Config Table/ Address Table ^{#1}	LnkA6	SNVTlnk_context. The structure of this SNVT in context specific.
7	Commit	NV Config Table/ Address Table ^{#1}	LnkA7	SNVT_switch (TRUE, FALSE).

^{#1} This is not an NV or CP. It is configuration information contained within the device’s NV configuration table and address table that specifies a network variable connection between two or more network variables on the device. It can only be used to connect network variables of the same type. The link is created as soon as the tables are updated, so the Commit operation is not a separate step.

Table 12 Local Link Object Network Services

<i>Local Link (Lnk) Object</i>				
<i>Service Request Code</i>	<i>Service Name</i>	<i>CDM Tag</i>	<i>Request Parameters</i>	<i>Result Parameters</i>
4	GetAttribute	AsmS4	Read NV config table and address table.	Attribute Value
5	SetAttribute	AsmS5	Write NV config table and address table.	

7.7 Sensor-AI Object (Sai) — The presentation of the Sensor Analog Input (Sensor-AI) object instance attributes are as indicated in Table 13. The presentation of the Sensor-AI object services is indicated in Table 14.

Table 13 Sensor-AI Object Instance Network Variables

<i>Sensor-AI Object (Sai)</i> <i>Profile ID = 180.11, Instance ID = 01 through k</i>				
<i>Sequence Number</i>	<i>Name</i>	<i>Storage Class</i>	<i>CDM Tag</i>	<i>Standard NV or CP Data Type</i>
1	Name	CP	SaiA1	Typedef unsigned char SCPTname (*) ^{#5}
2	Status	NVO	SaiA2	nvoStatus output of the Sensor-AI Object
3	Alarm Enable	NVI	SaiA3	nviRequest input of the Sensor-AI Object
4	Warning Enable	NVI	SaiA4	nviRequest input of the Sensor-AI Object
16	Value	NVO	SaiA16	Typedef (*) ^{#5} SNVT_xxx ^{#4}