whether the device supports references to *gas standard number*, *gas standard symbol*, or both. It is strongly recommended that, if a device supports only one method of referencing gas types, it should be *gas standard number*.

8.4.2.2 A device may support any of these services, in whole or in part, as specified by the manufacturer. A manufacturer may specify a subset of the service parameter values which is supported. The following table lists the services sup-ported by the Sensor-AI-MF object instance:

Table 9 Sensor-AI-MF Object Services

Service	*Service Identifier*	*Type*	*Description*
Perform Zero Offset	S1	R	Used to instruct the object to perform an automatic zeroing operation.
Query-Supported Gas Types	S2	R	Used to query the device to determine whether a specific gas calibration is supported.
Selected Programmed Gas Type	S3	R	Used to select the gas type and associated data to be used as the current programmed gas calibration.
Insert Gas Type	S4	R	Used to add a gas type to the list of available gas types.
Delete Gas Type	S5	R	Used to remove a gas type from the list of available gas types.
Get Gas Calibration Data Value	S6	R	Used to request the value of a specific gas calibration data value.
Set Gas Calibration Data Value	S7	R	Used to set the value of a specific gas calibration data value.
Autorange	S8	R	Used to enter the AUTORANGING state.
Reserved	S9–S64	—	Reserved for future expansion.
Manufacturer-Specified	> S64	—	Manufacturer-Specific services

8.4.2.3 *Perform Zero Offset (Optional)* — This service is used to instruct the Sensor-AI-MF object instance to perform a one-time automatic zeroing operation on the device or to reset the *offset* attribute value to zero. This service causes the automatic zeroing of the Sensor-AI-MF object instance *value* attribute by modifying the value of the *offset* attribute in order to yield a value of zero for the *value* attribute. Table 10 describes the parameters specified for this service.

Table 10 Perform Zero Service Parameter Definitions

Parameter	*Request/Indication*	*Response/ Confirmation*	*Data Type*	*Description*
Command	M	—	Byte	Enumerated Byte: 0 = Set offset attribute to zero 1 = Calculate offset 2 = Cancel Zeroing 3–63 = Reserved 64–255 = Manufacturer-Specified

8.4.2.4 *Query-Supported Gas Types Service (Optional)* — This service is used to query the device to determine whether a specific gas type, range, and units is supported. Supported, in this context, implies that the device contains suitable gas calibration correction data or methods to correct the flow measurement for the specified gas type, range, and units. The query can be made by referencing either *gas standard number* or *gas standard symbol* (see Section 5 for a definition of "gas standard number" and "gas standard symbol"). The entire list of supported gas standard numbers or gas standard symbols (with ranges and units) can also be requested. The following table describes the parameters specified for this service:

Table 11 Query-Supported Gas Types Service Parameter Definitions

Parameter	*Request/ Indication*	*Response/ Confirmation*	*Data Type*	*Description*
Query Type	M	—	Byte	0 = specific gas type 1 = all currently supported gas calibrations 2 = currently programmed gas calibration
Gas Standard Number	C*	—	Unsigned Integer	0 = use standard gas symbol field n = gas standard number
Gas Standard Symbol	C*	—	Text String	null character = not specified text string = gas standard symbol
Full Scale Range	C*	—	Real	0 = not specified n = full scale range
Units	C*	—	UINT	Indication of the units associated with the full scale range.
Valid Flag	—	M	Byte	0 = Not Valid 1 = Valid
Size of List	—	M	Unsigned Integer	Number of gas calibrations in the list.
List of Gas Calibrations	—	M	Array of Structures	The list of gas calibrations.

* Parameter is Mandatory for Query Type = 0.

8.4.2.4.1 *Query Type* — This parameter is used to specify the type of service response. It is an enumerated byte that can take on the following values:

0 = specific gas type
1 = all currently supported gas calibrations
2 = currently programmed gas calibration
3–63 = Reserved
64–255 = Manufacturer-Specified

8.4.2.4.2 *Gas Standard Number* — This parameter is used to specify a gas standard number, for which device support is being queried. A value of "0" indicates that the following parameter "gas standard symbol" is used instead to reference the gas type.

8.4.2.4.3 *Gas Standard Symbol* — This parameter is used to specify a gas standard symbol, for which device support is being queried. If the gas standard number parameter has a value that is not zero, then this parameter will have the value of null character.

8.4.2.4.4 *Full Scale Range* — This parameter is used to specify the full scale range, for which device support is being queried. See Section 5 for a definition of "full scale range." A value of "0" queries the device to return the entire list of full scale ranges for the specified gas type.

8.4.2.4.5 *Units* — This parameter is used to specify the units associated with the full scale range specified in the service request. Its values are defined in SEMI E54.1. The supported list includes:

SCCM
SLM
Percent
Volts
Millivolts
Counts

8.4.2.4.6 *Valid Flag* — The first parameter of a Query-Supported Gas Type service response is a byte that indicates whether the requested gas type, range, and units are supported. If an entire list was requested, and at least one gas calibration exists, this byte will have the value of "Valid."

8.4.2.4.7 *Size of List* — This parameter is an unsigned integer that specifies the number of gas calibrations in the list that follows.

8.4.2.4.8 *List of Gas Calibrations* — This attribute is an array of structures that identifies the gas calibrations supported in the device. The format of the structure is *gas standard number*, *gas standard symbol* (zero if not supported), *full scale range*, and *units*.

8.4.2.5 *Select Programmed Gas Type Service (Optional)* — This service is used to select the gas type, range, and units of the programmed gas calibration to be used by the device (see Section 5 for a definition of "programmed gas calibration"). The following table describes the parameters specified for this service:

Table 12 Select Programmed Gas Type Service Parameter Definitions

Parameter	*Request/ Indication*	*Response/ Confirmation*	*Data Type*	*Description*
Gas Standard Number	M	—	UINT	0 = use gas standard symbol field n = gas standard number
Gas Standard Symbol	M	—	Text String	null character = not specified text string = gas standard symbol
Range Select Mode	M	—	Byte	0 = select highest full scale range 1 = use full scale range and units specified 2 = use full scale range and units greater than or equal to specified
Full Scale Range	C*	—	REAL	Full scale range of which the select references.
Units	C*	—	UINT	Indication of the units associated with the full scale range.

* Parameter is Mandatory for Range Select Mode = 1 and 2.

8.4.2.5.1 *Gas Standard Number* — This parameter is used to specify a gas standard number, for which device support is being queried. A value of "0" indicates that the following parameter "gas standard symbol" is used instead to reference the gas type.

8.4.2.5.2 *Gas Standard Symbol* — This parameter is used to specify a gas standard symbol, for which device support is being queried. If the gas standard number parameter has a value that is not zero, then this parameter will have the value of null character.

8.4.2.5.3 *Range Select Mode* — This parameter is used to specify the mode by which the programmed gas is selected. It is an enumerated byte that can take on the following values:

0 = Select highest full scale range
1 = Use full scale range and units specified
2 = Use full scale range and units greater than or equal to specified
3–63 = Reserved
64–255 = Manufacturer-Specified

8.4.2.5.4 *Full Scale Range* — This parameter is used to specify the full scale range to be selected as the current programmed gas calibration full scale range.

8.4.2.5.5 *Units* — This parameter specifies the units associated with the full scale range to be set as the current programmed gas calibration being used by the device. If the Full Scale Range parameter has the value "0," this parameter is not included in the request. Its values are defined in Section 8.4.2.2.

8.4.2.6 *Insert Gas Type Service (Optional)* — This service is used to add a gas calibration to the list of supported gas calibrations. This service will typically be invoked while calibrating the device, since it has not specified what, if any, gas calibration data or methods will be set as the new gas calibration. The Set Gas Calibration Data Value service will generally be required to set the gas calibration data or methods after invoking this service. The following table describes the parameters specified for this service:

Table 13 Insert Gas Type Service Parameter Definitions

Parameter	*Request/ Indication*	*Response/ Confirmation*	*Data Type*	*Description*
Gas Standard Number	M	—	UINT	0 = use gas standard symbol field n = gas standard number
Gas Standard Symbol	M	—	Text String	null character = not specified text string = gas standard symbol
Full Scale Range	M	—	REAL	n = full scale range
Units	M	—	UINT	Indication of the units associated with the full scale range.

8.4.2.6.1 *Gas Standard Number* — This parameter is used to specify a gas standard number, for which a gas calibration is to be added. A value of "0" indicates that the following parameter "gas standard symbol" is used instead to reference the gas type.

8.4.2.6.2 *Gas Standard Symbol* — This parameter is used to specify a gas standard symbol for which a gas calibration is to be added.

8.4.2.6.3 *Full Scale Range* — This parameter is used to specify the full scale range for which a gas calibration is to be added. See Section 5 for a definition of full scale range.

8.4.2.6.4 *Units* — This parameter is used to specify the units associated with the full scale range specified in the service request. Its values are defined in Section 8.4.2.4.

8.4.2.7 *Delete Gas Type Service (Optional)* — This service is used to remove a gas calibration from the list of supported gas calibrations. The gas calibration data or method used by the device, if the current programmed gas type is deleted, is manufacturer-specified. The following table describes the parameters specified for this service:

Table 14 Delete Gas Type Service Parameter Definitions

Parameter	*Request/ Indication*	*Response/ Confirmation*	*Data Type*	*Description*
Gas Standard Number	M	—	UINT	0 = use gas standard symbol field n = gas standard number
Gas Standard Symbol	M	—	Text String	null character = not specified text string = gas standard symbol
Full Scale Range	M	—	REAL	n = full scale range
Units	M	—	UINT	Indication of the units associated with the full scale range.

8.4.2.7.1 *Gas Standard Number* — This parameter is used to specify a gas standard number for which a gas calibration is to be deleted. A value of "0" indicates that the following parameter "gas standard symbol" is used instead to reference the gas type.

8.4.2.7.2 *Gas Standard Symbol* — This parameter is used to specify a gas standard symbol for which a gas calibration is to be deleted.

8.4.2.7.3 *Full Scale Range* — This parameter is used to specify the full scale range for which a gas calibration is to be deleted. See Section 5 for a definition of "full scale range."

8.4.2.7.4 *Units* — This parameter is used to specify the units associated with the full scale range specified in the service request. Its values are defined in Section 8.4.2.4.

8.4.2.8 *Get Gas Calibration Data Value Service (Optional)* — This service is used to retrieve the values associ-ated with a gas calibration. The mechanism for referencing a data value involves specifying the gas type and full scale range (a default is provided to specify the programmed gas calibration currently in use by the device), followed by an index value which is used to specify the particular data value that is being requested. The following table describes the parameters specified for this service:

Table 15 Get Calibration Data Value Service Parameter Definitions

Parameter	*Request/ Indication*	*Response/ Confirmation*	*Data Type*	*Description*
Gas Standard Number	M	—	INT	-1 = current programmed gas calibration 0 = use gas standard symbol field n = gas standard number
Gas Standard Symbol	M	—	Text String	null character = not specified text string = gas standard symbol
Full Scale Range	M	—	REAL	0 = not specified n = full scale range
Units	M	—	UINT	Indication of the units associated with the full scale range.
Data Index	M	—	Byte	The identifier of the particular gas calibration datum within the list of data values.
Size of List	—	M	Byte	This parameter specifies the number of values returned in the list.
Zero	—	C	REAL	zero offset
Span	—	C	REAL	span multiplier
Calibration Date	—	C	Date	The date of calibration for a particular gas type and full scale range.
Calibration Gas Standard Number	—	C	UINT	The gas standard number representing the gas used to calibrate the device.
Calibration Temperature	—	C	REAL	The standard temperature of calibration conditions.
Calibration Pressure	—	C	REAL	The standard pressure of the calibration conditions.
Manufacturer-Specified Parameters	—	C	Manufacturer-Specific	

8.4.2.8.1 *Gas Standard Number* — This parameter is used to specify a gas standard number for which a gas calibration data value is being requested. Two special values are allowed: A value of "0" indicates that the following parameter "gas standard symbol" is used instead to reference the gas type. A value of "-1" indicates that the current programmed gas calibration is requested.

8.4.2.8.2 *Gas Standard Symbol* — This parameter is used to specify a gas standard symbol for which a gas calibration data value is being requested. If the *gas standard number* parameter has a value that is not "0," then this parameter is set to the null character.

8.4.2.8.3 *Full Scale Range* — This parameter is used to specify the full scale range for which a gas calibration data value is being requested. If the *gas standard number* parameter has a value that is "-1," then this parameter is set to zero.

8.4.2.8.4 *Units* — This parameter specifies the units associated with the *full scale range parameter*. Its values are defined in Section 8.4.2.4. If the *gas standard number* parameter has a value that is "-1," then this parameter is set to 0.

8.4.2.8.5 *Data Index* — This parameter is used to specify the index of the value being requested. It is an enumerated byte that can take on the values listed in the following table:

Table 16 Gas Calibration Data Value Matrix

Data Value Name	*Data Index*	*Access Network*	*Required*	*Form*
All	0	—	No	
Zero	1	RW	No	REAL
Span	2	RW	No	REAL
Calibration Date	3	R	No	Date
Calibration Gas Standard Number	4	R	No	UINT
Calibration Temperature	5	R	No	REAL
Calibration Pressure	6	R	No	REAL
Reserved	7–63	—	—	
Manufacturer-Specified	64–255	—	—	

Table 17 Gas Calibration Data Initial and Default Values

Data Value Name	*Data Index*	*Initial Value*	*Default Value*	*Comment*
Zero	1	LVV	0	
Span	2	LVV	1	
Calibration Date	3	LVV	0, 0, 0	
Calibration Gas Standard Number	4	LVV	Manufacturer-Specific	
Calibration Temperature	5	LVV	0.0	Defined by SEMI E12 as 0.0°C. Other industries may specify a different value.
Calibration Pressure	6	LVV	101.32	Defined by SEMI E12 as 101.32 kPa. Other industries may specify a different value.
Reserved	7–63	—	—	
Manufacturer-Specified	64–255	—	—	

8.4.2.8.5.1 *All* — This value for the data index parameter requests the list of all data values associated with a gas calibration. The returned parameter values are ordered by index as defined in Table 15. The size of the list returned is manufacturer-specified, since it includes the Manufacturer-Specific values.

8.4.2.8.5.2 *Size of List* — This parameter is used to specify the number of parameters returned in the response list.

8.4.2.8.5.3 *Zero* — This value is used in conjunction with the span value to correct the flow measurement. It is expressed in terms of the units parameter.

8.4.2.8.5.4 *Span* — This value is used in conjunction with the zero value to correct the flow measurement. It is a dimensionless value.

8.4.2.8.5.5 *Calibration Date* — This value identifies the date the device was last calibrated for the specified gas type and full scale range.

8.4.2.8.5.6 *Calibration Gas Standard Number* — This value identifies the gas type used when the device was calibrated for the referenced gas type and full scale range. It may be the same gas as gas type, a surrogate gas, nitrogen, or some other gas.

8.4.2.8.5.7 *Calibration Temperature* — This value identifies the Standard Temperature with respect to calibration conditions. The units for this value are degrees Centigrade.

8.4.2.8.5.8 *Calibration Pressure* — This value identifies the Standard Pressure with respect to calibration conditions. The units for this value are KiloPascal.

8.4.2.9 *Set Gas Calibration Data Value Service (Optional)* — This service is used to set the values associated with a gas calibration. The mechanism for referencing a data value involves specifying the gas type and full scale range (a default is provided to specify the programmed gas calibration currently in use by the device), followed by an index value which is used to specify the particular data value associated with the referenced gas calibration that is

being set. The specific value to be set is the last parameter passed in the service request. The following table describes the parameters specified for this service:

Table 18 Set Calibration Data Value Service Parameter Definitions

Parameter	*Request/ Indication*	*Response/ Confirmation*	*Data Type*	*Description*
Gas Standard Number	M	—	INT	-1 = current programmed gas calibration 0 = use gas standard symbol field n = gas standard number
Gas Standard Symbol	M	—	Text String	null character = not specified text string = gas standard symbol
Full Scale Range	M	—	REAL	0 = not specified n = full scale range
Units	M	—	UINT	Indication of the units associated with the full scale range.
Data Index	M	—	Byte	The identifier of the particular gas calibration datum within the list of data values.
Size of List	M	—	Byte	This parameter specifies the number of values in the list that follows.
Zero	C	—	REAL	zero offset
Span	C	—	REAL	span multiplier
Calibration Date	C	—	Date	The date of calibration for a particular gas type and full scale range.
Calibration Gas Standard Number	C	—	UINT	The gas standard number representing the gas used to calibrate the device.
Calibration Temperature	C	—	REAL	The standard temperature of calibration conditions.
Calibration Pressure	C	—	REAL	The standard pressure of the calibration conditions.
Manufacturer-Specified Parameters	C	—	Manufacturer-Specific	

For a description of these parameters, see the preceding section.

8.4.2.10 *Autorange Service (Optional)* — This service is used to instruct the Sensor-AI-MF object instance to enter the AUTORANGING state from the NOT AUTORANGING state or to enter the NOT AUTORANGING state from the AUTORANGING state. The following table describes the parameters specified for this service:

Table 19 Autorange Service Parameter Definitions

Parameter	*Request/ Indication*	*Response/ Confirmation*	*Data Type*	*Description*
Command	M	—	Byte	Enumerated Byte: 0 = enter NOT AUTORANGING state 1 = enter AUTORANGING state 2–63 = Reserved 64–255 = Manufacturer-Specified

8.4.3 *Sensor-AI-MF Object Behavior* — The behavior exhibited by the Sensor-AI-MF object instance is inherited from the Sensor-AI object defined in SEMI E54.1. Additional specific behavior associated with the Sensor-AI-MF object is defined below.

8.4.3.1 *Sensor-AI-MF OPERATING Application Process* — A reading is retrieved from a physical flow sensor. This reading may be corrected with a manufacturer-specified algorithm. This corrected reading becomes the input to the offset and gain formula to generate the *value* attribute as referenced in SEMI E54.1.

8.4.3.1.1 For Gas Correction, the value retrieved is corrected with a manufacturer-specified algorithm using the correction values, parameters, coefficients, or methods for the programmed gas calibration.

8.4.3.2 *Sensor-AI-MF OPERATING Application Process–Flow Totalizer* — The value of the *flow totalizer* attribute is incremented at a rate of once every cubic centimeter of gas flow. Whenever the *flow totalizer* attribute reaches its maximum allowed value, it no longer is incremented and remains at the maximum value.

8.4.3.3 *Sensor-AI-MF OPERATING Application Process–Flow Hours* — The value of the *flow hours* attribute is incremented at a rate of once every hour. Whenever the *flow hours* attribute reaches its maximum allowed value, it no longer is incremented and remains at the maximum value.

8.4.3.4 *Sensor-AI-MF ZEROING Application Process* — The Zeroing application process is described as follows: Certain manufacturer-specified service requests may be sent to other objects. The value of *offset* is set such that *value* is nulled to zero. The Sensor-AI-MF object instance determines, using a manufacturer-specified method, when a Zeroing application process is completed.

8.4.3.4.1 If a Perform Zero Offset: Command = 2 (cancel) service request is received while in the ZEROING state, the original value of the *offset* attribute is restored, and the process is considered Failed. If a Perform Zero Offset: Command = 0 (set offset attribute to zero) service request is received while in the ZEROING state, the *offset* attribute is set to zero, and the process is considered Failed. If a Perform Zero Offset: Command = 1 (calculate offset) service request is received while in the ZEROING state, the process is simply restarted.

8.4.3.4.2 Upon completion, the appropriate Pass or Fail service response is reported to the requesting object instance, and an internal Operate service request to this object is generated.

8.4.3.4.3 The Sensor-AI-MF object instance supports the additional behavior sub-states (defined in Figure 3) within the OPERATING state. Table 20 defines the additional sub-states, and Table 21 defines the additional state transitions specified for this object.

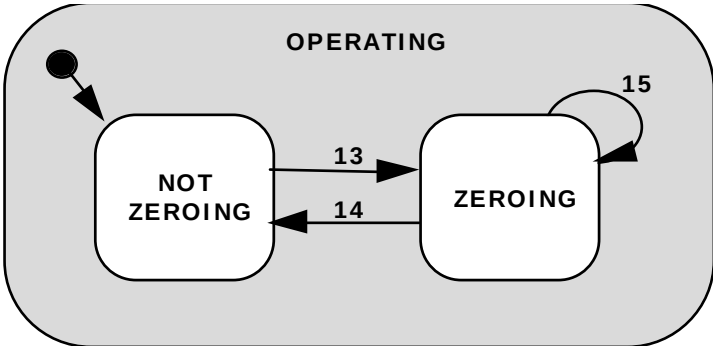

Figure 3
Sensor-AI-MF Object Behavior Additional Sub-States

Table 20 Sensor-AI-MF Object Behavior State Descriptions

State	*Description*
NOT ZEROING	The Sensor-AI-MF object instance is running the Sensor-AI-MF OPERATING application process.
ZEROING	The Sensor-AI-MF object instance is running the Sensor-AI-MF OPERATING application process. Additionally, the Sensor-AI-MF object instance is running the Sensor-AI-MF ZEROING application process.

Table 21 Sensor AI-MF Object Behavior State Transition Matrix

#	*Current State*	*Trigger*	*New State*	*Action*	*Comments*
4	NORMAL OPERATING	Get Attribute, Set Attribute, Restore Defaults request, Perform Zero Offset request (command ≠ 1)	NORMAL OPERATING	Get Attribute, Set Attribute, Restore Defaults appropriate response	Valid for all sub-states of NORMAL OPERATING.
13	NOT ZEROING	Perform Zero Offset request (command = 1)	ZEROING	Run the ZEROING application process. Set *zeroing status* to Zeroing.	
14	ZEROING	Operate request or Perform Zero Offset request (command ≠ 1)	NOT ZEROING	Resume the OPERATING application process. Set *zeroing status* to Not Zeroing. Report appropriate service response.	The mechanism for reporting is network-specific.
15	ZEROING	Perform Zero Offset request (command = 1)	ZEROING	Restart the ZEROING application process.	

8.4.3.5 *Sensor-AI-MF AUTORANGING Application Process* — The Sensor-AI-MF object instance supports the additional behavior sub-states (defined in Figure 4) within the OPERATING state. Table 22 defines the additional sub-states, and Table 23 defines the additional state transitions specified for this object.

8.4.3.5.1 The object instance is automatically selecting gas calibrations based on a manufacturer's specified method. This method may include a determination based on the value of the *value* attribute and/or on the value of the *setpoint value* attribute.

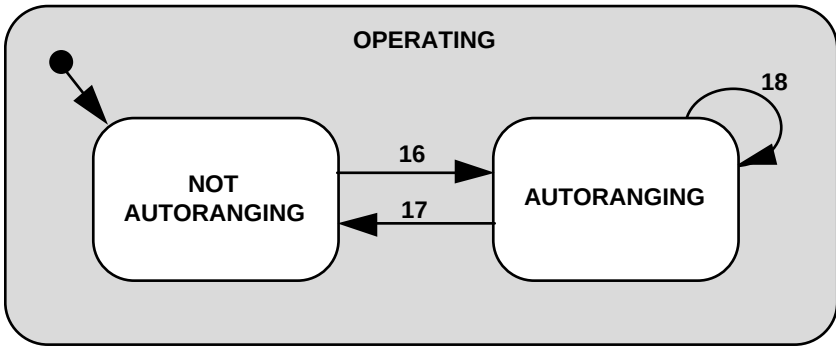

Figure 4
Sensor-AI-MF Object Behavior Additional Sub-States

Table 22 Sensor-AI-MF Object Behavior State Descriptions

State	*Description*
NOT AUTORANGING	The Sensor-AI-MF object instance is running the Sensor-AI-MF OPERATING application process.
AUTORANGING	The Sensor-AI-MF object instance is running the Sensor-AI-MF OPERATING application process. Additionally, the Sensor-AI-MF object instance is running the Sensor-AI-MF AUTORANGING application process (defined above).

Table 23 Sensor-AI-MF Object Behavior State Transition Matrix

#	*Current State*	*Trigger*	*New State*	*Action*	*Comments*
4	NORMAL OPERATING	Get Attribute, Set Attribute, Restore Defaults request, Autorange request (command = 0)	NORMAL OPERATING	Get Attribute, Set Attribute, Restore Defaults appropriate response	Valid for all sub-states of NORMAL OPERATING.
16	NOT AUTORANGING	Autorange request Command = 1	AUTORANGING	Run AUTORANGING application process. Set *Autoranging status* to Autoranging.	
17	AUTORANGING	Autorange request Command = 0	NOT AUTORANGING	Halt the Autoranging application process. Set *Autoranging status* to Not Autoranging.	
18	AUTORANGING	Autorange request Command = 1	AUTORANGING		No change.

8.5 *Sensor-AI-AT Object* — The Sensor-AI-AT object is an Ambient Temperature object instance which inherits attributes, services, and behavior from the Sensor-AI object as defined in SEMI E54.1. The Sensor-AI-AT object instance is the device component responsible for retrieving a reading from a physical temperature sensor and making the value available to the network.

8.5.1 *Sensor-AI-AT Object Attributes* — The inherited *value* attribute represents the ambient temperature measurement. There are no additional attributes defined for this object in this document.

8.5.2 *Sensor-AI-AT Object Services* — There are no additional services defined for this object in this document.

8.5.3 *Sensor-AI-AT Object Behavior* — The behavior exhibited by the Sensor-AI-AT object instance is defined in SEM E54.1. Additional specific behavior associated with the Sensor-AI-AT object is defined below.

8.5.3.1 *Sensor-AI-AT OPERATING Application Process* — A reading is retrieved from a physical temperature sensor. This reading may be corrected with a manufacturer-specified algorithm. This corrected reading becomes the input to the offset and gain formula to generate the *value* attribute as referenced in SEMI E54.1.

8.6 *Assembly-MFM Object* — The Assembly-MFM object inherits attributes, services, and behavior from the Assembly object. The Assembly object instance is the device component which provides a mechanism of grouping more than one attribute from one or more object instances into a single data structure for access over the network.

8.6.1 *Assembly-MFM Object Attributes*

Table 24 Assembly-MFM Object Attributes

Attribute Name	*Attribute Identifier*	*Access Network*	*Required*	*Form*
Data	A1	R	Yes	Structure as defined below.

8.6.1.1 *Data* — An attribute with the following list of attributes within its structure:

Table 25 Assembly-MFM Data List

Data Index	*Source Object ID*	*Source Attribute ID*	*Description*
1	DM1	A12	Device Manager Exception Status
2	MFD3	A4	Sensor-AI-MF Value

8.7 *Sensor-AI-Aux Object* — The Sensor-AI-Aux object is an Auxiliary Input object instance which inherits attributes, services, and behavior from the Sensor-AI object as defined in SEMI E54.1. The Sensor-AI-Aux object instance is the device component responsible for retrieving a reading from a physical analog input and making the value available to the network.

8.7.1 *Sensor-AI-Aux Object Attributes* — The attributes provided by the Sensor-AI-Aux object instance are defined in SEMI E54.1. The *value* attribute represents the analog input measurement.

8.7.2 *Sensor-AI-Aux Object Services* — The services provided by the Sensor-AI object instance are defined in SEMI E54.1.

8.7.3 *Sensor-AI-Aux Object Behavior* — The behavior exhibited by the Sensor-AI-Aux object instance is defined in SEMI E54.1. Additional specific behavior associated with the Sensor-AI-Aux object is defined below.

8.7.3.1 *Sensor-AI-Aux OPERATING Application Process* — A reading is retrieved from a physical analog input. This reading may be corrected with a manufacturer-specified algorithm. This corrected reading becomes the input to the offset and gain formula to generate the *value* attribute as referenced in SEMI E54.1.

8.8 *Actuator-AO-MF Object* — The Actuator-AO-MF object instance inherits from the Actuator-AO object attributes, services, and behavior as defined in SEMI E54.1. The Actuator-AO-MF object instance is the component responsible for driving the physical mass flow control valve of the device. Override attributes are available to produce valve drives which either fully close or fully open the valve, irrespective of the value of the *setting* attribute.

8.8.1 *Actuator-AO-MF Object Attributes*

Table 26 Actuator-AO-MF Object Attributes

Attribute Name	*Attribute Identifier*	*Access Network*	*Required*	*Form*
Valve Type	A1	R	No	Enumerated Byte
Override	A2	RW	No	Enumerated Byte
Reserved	A3–A64	—	—	Reserved for future expansion.
Manufacturer-Specified	> A64	—	—	Manufacturer-Specific attributes.

8.8.1.1 *Valve Type (Optional)* — An attribute which specifies the type of valve present in the device. This attribute is an enumerated byte that can take on one of the following values:

0 = Solenoid

1 = Voice Coil

2 = Piezo Electric

3 = Thermal

4–63 = Reserved

64–255 = Manufacturer-Specified

8.8.1.2 *Override (Optional)* — An attribute which specifies an override to the controlled valve position derived from the Controller object instance. This attribute is an enumerated byte that can take on one of the following values:

0 = Normal — Normal control mode

1 = Flow Off — Valve Closed

2 = Purge — Valve Open

3 = Power Off — Valve to unpowered state

4–63 = Reserved

64–255 = Manufacturer-Specified

8.8.1.3 *Initial and Default Values*

Table 27 Actuator-AO-MF Object Attribute Initial and Default Values

Attribute	*Initial Value*	*Default Value*	*Comment*
Valve Type	Default Value	Manufacturer-Specific	
Override	LVV	Normal	

8.8.2 *Actuator-AO-MF Object Services* — There are no additional services defined for this object in this document.

8.8.3 *Actuator-AO-MF Object Behavior* — The behavior exhibited by the Actuator-AO-MF object instance is defined in SEMI E54.1. The specific behavior associated with the Actuator-AO-MF object is defined below.

8.8.3.1 *Actuator-AO-MF OPERATING Application Process–Override* — The *override* attribute determines whether the position of the physical flow control valve will be overridden and, if so, to what position it will be driven. Otherwise, the physical flow control valve is driven by the converted signal derived from the Controller object instance.

8.9 *Controller Object* — The Controller object definition is provided in SEMI E54.1. This object instance is the device component which provides the closed-loop control of mass flow. There are no additional attributes, services, or behavior defined for this object in this document.

8.9.1 *Controller Object Attributes* — The following attribute table is provided to enhance the C Class Attribute table and provides a specific attribute "Form" definition.

Table 28 C Class Attributes

Attribute Name	*Attribute Identifier*	*Access Network*	*Required*	*Form*
Alarm Settling Time	CA21	RW	N	Real
Warning Settling Time	CA24	RW	N	Real

8.10 *Local Link Object* — The Local Link object is defined in SEMI E54.1. This object instance is the device component which provides a mechanism of linking two attributes within the device. There are no additional attributes, services, or behavior defined for this object in this document.

8.11 *SISO Object* — The SISO object provides a Single Input, Single Output function. At this level, only the input and output attributes are defined. An instance of this object makes no sense, because there is no transfer function defined, but rather, the attributes, services, and behavior of this object are inherited by the next level down (e.g., SISO-Setpoint Object).

8.11.1 *SISO Object Attributes*

Table 29 SISO Function Object Attributes

Attribute Name	*Attribute Identifier*	*Access Network*	*Required*	*Form*
Input	A1	RW	Yes	Data Type
Output	A2	R	Yes	Data Type
Data Type	A3	R	No	USINT
Reserved	A4–A32	—	—	For future revisions to this object.
Reserved	A33–A64			For next level object.
Manufacturer-Specified	> A64	—	—	For manufacturer specification.

8.11.1.1 *Input* — An attribute which specifies the input value, or independent variable, for the transfer function. The data type is specified by the *data type* attribute.

8.11.1.2 *Output* — An attribute whose value is the output, or dependent variable, of the transfer function. The data type is specified by the *data type* attribute.

8.11.1.3 *Data Type* — An attribute which specifies the data type of the *input* attribute and the *output* attribute. The format and values of this attribute are defined in SEMI E54.1.

8.11.1.4 *Initial and Default Values*

Table 30 Table 30 SISO Object Attribute Initial and Default Values

Attribute	*Initial Value*	*Default Value*	*Comment*
Input	LVV	0	
Output	LVV	manufacturer-specified	
Data Type	LVV	manufacturer-specified	

8.11.2 *SISO Object Behavior* — The only behavior defined at this level is that of a continuously operating transfer function: The Output attribute value is calculated as a function of the Input attribute value. The specific transfer function is defined by lower level objects which inherit from this object.

Output = F(Input)

where: F is the function specified

8.12 *SISO-Setpoint Object* — The SISO-Setpoint Object inherits from the SISO object. The definitions added by the SISO-Setpoint object include the behavior associated with the transfer function.

8.12.1 *SISO-Setpoint Object Attributes*

Table 31 SISO-Setpoint Object Attributes

Attribute Name	*Attribute Identifier*	*Access Network*	*Required*	*Form*
Ramp Type	A33	RW	No	USINT
Ramp Rate	A34	RW	No	Data Type
Ratio	A35	RW	No	REAL
Reserved	A36–A64	—	—	Reserved for future expansion.

8.12.1.1 *Ramp Type (Optional)* — An attribute which specifies a mechanism by which the *output* attribute is ramped to the current value of the *input* attribute.

0 = Disable

1 = Time in seconds

2 = Amount per second

3–63 = Reserved

64–255 = Manufacturer-specified

8.12.1.2 *Ramp Rate (Optional)* — An attribute specified to define the ramp rate at which the SISO-Setpoint object instance tracks towards the current *input* value. The ramp rate specifies how quickly the *output* is ramped from the previous *output* value to the current *input* value. This attribute is expressed in terms of seconds or amount of change per second.

8.12.1.3 *Ratio (Optional)* — An attribute which specifies the ratio multiplier to be applied to the *input* attribute value prior to the application of the ramp transfer function.

8.12.1.4 *Initial and Default Values*

Table 32 SISO-Setpoint Object Attribute Initial and Default Values

Attribute	*Initial Value*	*Default Value*	*Comment*
Ramp Type	LVV	Disable	
Ramp Rate	LVV	0	
Ratio	LVV	1.0	

8.12.2 *SISO-Setpoint Object Behavior* — The general behavior exhibited by the SISO-Setpoint object instance is defined in SEMI E54.1. The specific behavior associated with this object is defined below.

8.12.2.1 *SISO-Setpoint OPERATING Application Process–Ramp* — The *ramp type* and *ramp rate* attributes determine the conditions under which the value of the *output* attribute is calculated and modified by this application process. The *ramp rate* attribute is defined to express the rate at which the *output* attribute is ramped to the value of the *input* attribute based on the value of the *ramp type* attribute. Table 33 describes the behavior of the SISO-Setpoint object instance based on the value of the *ramp type* attribute.

Table 33 SISO-Setpoint Object Behavior - Ramp

Ramp Type Value	*SISO-Setpoint Object Instance Behavior*
Disabled	Achieve a new output value in one step.
Time in Seconds	Achieve a new output value in the time interval specified by the *ramp rate* attribute.
Amount per Second	Achieve a new output value in the amount per second increments specified by the *ramp rate* attribute.

8.12.2.2 *SISO-Setpoint OPERATING Application Process–Ratio* — Prior to the application of the Ramp calculation, a multiplier is applied to the value of the *input* attribute. The value of the *ratio* attribute is multiplied to the value of the *input* attribute, and the result becomes the value which is used in the ramp calculation.

The combined formula for the SISO-Setpoint object is as follows:

Output = F(x)

where: F is the function as defined by ramp type above

and: x = (ratio)(input)

8.13 *Assembly-MFC Object* — The Assembly-MFC object inherits attributes, services, and behavior from the Assembly object. The Assembly object instance is the device component which provides a mechanism of grouping more than one attribute from one or more object instances into a single data structure for communication over the network.

8.13.1 *Assembly-MFC Object Attributes*

Table 34 Assembly-MFC Object Attributes

Attribute Name	*Attribute Identifier*	*Access Network*	*Required*	*Form*
Data	A1	R	Yes	Structure as defined below.

8.13.1.1 *Data* — An attribute with the following list of attributes within its structure:

Table 35 Assembly-MFC Data List

Data Index	*Source Object ID*	*Source Attribute ID*	*Description*
1	DM1	A12	Device Manager Exception Status
2	MFD8	A1	Controller Status
3	MFD3	A4	Sensor-AI-MF Value
4	MFD8	A4	Controller Setpoint
5	MFD8	A6	Controller Control Variable

SEMI E54.4-0704
STANDARD FOR SENSOR/ACTUATOR NETWORK COMMUNICATIONS FOR DEVICENET

This standard was technically approved by the Global Information & Control Committee and is the direct responsibility of the North American Information & Control Committee. Current edition approved by the North AmericanRegional Standards Committee on April 22, 2004. Initially available at www.semi.org June 2004; to be published July 2004. Originally published September 1997.

1 Purpose

1.1 This standard defines a communication specification based on the DeviceNet protocol to enable communications between intelligent devices on a sensor/actuator network (SAN) that operate according to SEMI- specified device models (common and device specific) in a semiconductor manufacturing tool.

1.2 *Background and Motivation* — DeviceNet provides for networking between simple industrial devices (e.g., sensors and actuators) and higher level devices such as controllers. DeviceNet provides:

- A solution to low-level device networking.
- Access to intelligence present in low-level devices.
- Master/Slave and Peer-to-Peer capabilities.

1.2.1 DeviceNet is based on the Controller Area Network (CAN) technology. CAN defines a Media Access Control (MAC) methodology and physical signaling characteristics. DeviceNet wraps a communication model and protocol as well as a complete Physical Layer definition around CAN to provide a complete network definition.

1.3 This document enables communications between intelligent devices on a SEMI-compliant SAN by providing a presentation mapping of common and specific device network visible structure and behavior to a DeviceNet network.

2 Scope

2.1 This document specifies the protocol and services that compliant intelligent devices must support to interchange information over this semiconductor equipment sensor/actuator network.

2.2 This document specifies the utilization of the DeviceNet protocol to present externally visible device structure and behavior, specified in the Common Device Model (CDM) and appropriate Specific Device Models (SDM' s), on a DeviceNet network.

2.3 This document is used in conjunction with a SEMI standard SAN Common Device Model specification and one or more SEMI standard-specific device model specifications (e.g., for a mass flow controller). Together, they describe the externally visible data structure and behavior of devices using the DeviceNet networking capability in a SEMI-compliant SAN system.

2.4 This standard, together with a sensor/actuator network interoperability guideline, the sensor/actuator network common device model, one or more sensor/actuator network specific device model documents, and the DeviceNet specifications, form a complete interoperability standard. The general sensor/actuator network document architecture is shown in the Sensor/Actuator Network Common Device Model document in Figure 1.

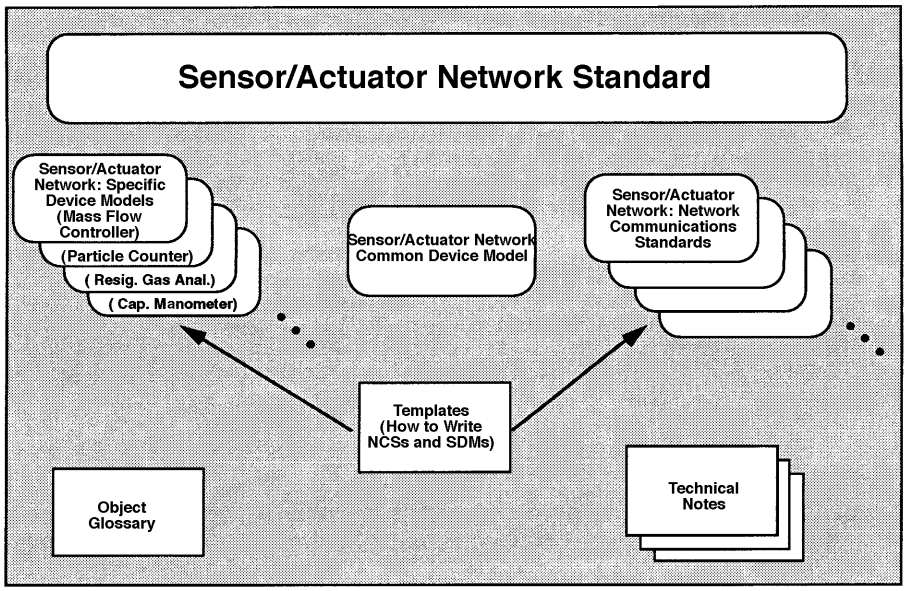

Figure 1
Sensor/Actuator Network Related Documents

2.5 *Document Structure* — The DeviceNet network communication standard complies with the SEMI SAN NCS template document structure; this structure is shown in Figure 2. The standard document is composed of two main parts. The first part (Sections 1 through 8) specifies the SAN enabling protocol as well as the presentation (i.e., mapping) of CDM object structure and behavior onto the network (referred to as the "CDM mapping"). The second part (Section 9) specifies the presentation (i.e., mapping) of SDM object structure and behavior onto the network for each SEMI-specified SDM (referred to as the "SDM mapping").

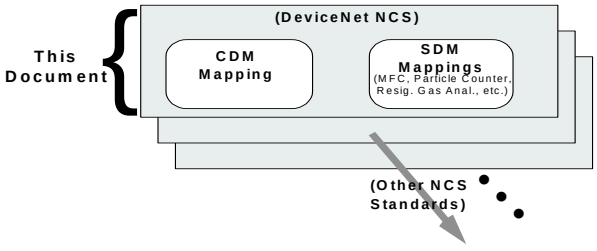

Figure 2
DeviceNet NCS Document Structure

2.6 *Adding SDM Mappings* — SDM mappings added to part two of this document are considered document additions and are balloted as such. An SDM mapping may only be balloted for addition to this document if the corresponding SEMI SDM has been standardized or is in the process of being balloted for standardization.

NOTICE: This standard does not purport to address safety issues, if any, associated with its use. It is the responsibility of the users of this standard to establish appropriate safety and health practices and determine the applicability of regulatory or other limitations prior to use.

3 Limitations

3.1 This document specifies a semiconductor equipment SAN based solely on DeviceNet and is a companion document to the DeviceNet specification; thus, a complete specification of this standard necessarily includes the DeviceNet 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 DeviceNet. In order to avoid document consistency problems, information in the DeviceNet specification that relates to this standard is not repeated in this document. This document is limited to describing enhancements or limitations to the DeviceNet specification that are imposed by this standard.

3.3 A complete specification of the conformance testing procedure shall include the DeviceNet protocol conformance testing specification. Conformance testing shall also include enhancements and limitations to the DeviceNet specification required by this standard.

4 Referenced Standards

4.1 *SEMI Standards*

SEMI E30 — Generic Model for Communications and Control of SEMI Equipment (GEM)

SEMI E39 — Object Services Standard: Concepts, Behavior, and Services

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

4.2 *ISO Standards*[1]

ISO 7498 — Basic Reference Model for Open Systems Interconnection

ISO 11898 — Road Vehicles — Interchange of Digital Information — Controller Area Network (CAN) for High-Speed Communications

4.3 *Other Documents*

DeviceNet Specification — ODVA[2]

Controller Area Network Specification — Version 2, R. Bosch GmbH, + Postfach 50 D-7000, Stuttgart 1, Germany, 1991

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

5 Terminology

Terminology that is common to all of the documents in this SAN standard may also be defined in the Sensor Actuator Network Standard. Terminology may be reproduced here which is defined in other SEMI documents.

5.1 *Abbreviations & Acronyms*

5.1.1 *CAN* — Controller area network

5.1.2 *CDM* — Common device model

5.1.3 *DM* — Device manager (object)

5.1.4 *DN* — DeviceNet

5.1.5 *NCS* — Network communication standard

5.1.6 *OSI* — Open systems interconnect

5.1.7 *OSS* — Object services standard

5.1.8 *SAC* — Sensor, actuator, controller (object)

5.1.9 *SAN* — Sensor/actuator network

5.1.10 *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 CDM document. Table 1 provides a mapping of fundamental terminology of the CDM document into this document and the DeviceNet specification. Note that Column 2 contains an equal sign "=" if the definition is used exactly as specified in the CDM specification.

Table 1 Mapping of CDM to NCS Terminology

CDM Term	*NCS Equivalent*	*DeviceNet Equivalent*
Device	=	=
Device Model	=	=
Object	=, Class	=, Class
Instance	=	=
Attribute	=	=
Behavior	=	=
Service	=	=
State Diagram	=	=
Byte	=	=
Nibble	=	=
Character String	=	=

5.3 *DeviceNet Specific Definitions*

5.3.1 *class* — a set of objects that all represent the same kind of system component. A class is a generalization of an object. All objects in a class are identical in form and behavior, but may contain different attribute values.

5.3.2 *controller area network (CAN)* — a protocol developed by the Bosch corporation for automotive in-vehicle networking. The CAN specification specifies OSI reference model layers 1 and 2, specifically the physical signaling and media access/data link protocols.

5.3.3 *device profile* — a DeviceNet specification for a device that contains an object model for the device type, the I/O data format for the device type, and the configuration data and the public interface(s) to that data.

5.3.4 *explicit message connections* — connections over a DeviceNet network that provide generic, multi-purpose communication paths between two devices. These connections often are referred to as just messaging connections. Explicit messages provide the typical request/response - oriented network communications.

1 International Organization for Standardization, ISO Central Secretariat, 1, rue de Varembé, Case postale 56, CH-1211 Geneva 20, Switzerland. Telephone: 41.22.749.01.11; Fax: 41.22.733.34.30, Website: www.iso.ch

2 Open DeviceNet Vendor Association, www.odva.org

5.3.5 *input/output connections* — connections over a DeviceNet network that provide dedicated, special-purpose communication paths between a producing application and one or more consuming applications. Application-specific I/O data moves through these ports.

6 Communication Protocol High Level Structure

6.1 The DeviceNet protocol is loosely based on a three-layer architecture. These layers constitute a collapsed form of the OSI seven-layer architecture, mapping into the physical, data link, and application layers of the Reference Model; however, DeviceNet provides additional functionality (such as connection support) commonly attributed to other OSI layers. The high level protocol architecture is shown in Figure 3.

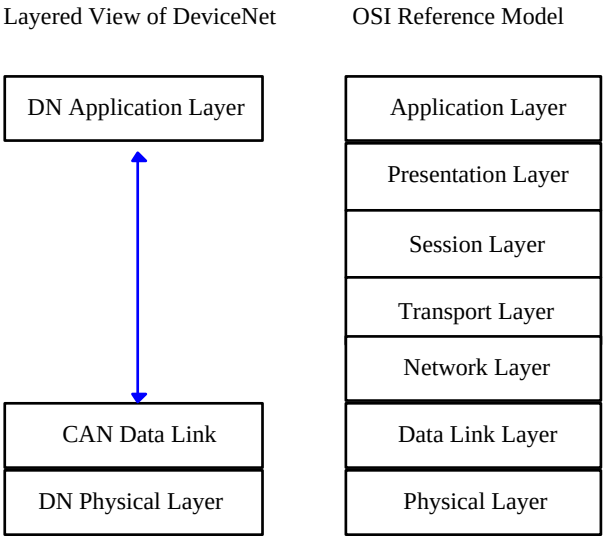

Figure 3
Layered View of DeviceNet

6.1.1 Note that Figure 3 represents a conceptual view of the device architecture. Conforming implementations must implement the services defined in this specification at each layer and must appear (from the network) to have implemented this architecture; however, an internal modular partitioning is not required. Implementations may sacrifice modularity in order to achieve high performance.

6.2 The DeviceNet physical layer is fully specified in Volume 1 of the DN Specification. Features of this layer include Trunkline - dropline configuration, simultaneous support for both network-powered and self-powered devices, and selectable data rates including at least 125k, 250k, and 500k baud. At the data link layer, the CAN specification defines a carrier sense multiple access mechanism for media access control that avoids collisions and sends frames reliably. The application layer is specified in Volumes 1 and 2 of the DN Specification and provides for the definition of DeviceNet applications as a collection of addressable objects. Two basic categories of objects exist: Communication Objects and Application Objects. Communication Objects manage and provide for the runtime exchange of messages across DeviceNet. Application Objects implement product-specific features and/or provide a logical interface to product-specific information of devices on a DeviceNet network.

6.3 In the remainder of this section, the protocol structure is described in more detail in terms of the OSI seven layer reference model, the object model environment, and network management specifications.

6.4 *Physical Layer* — The device shall comply with the DeviceNet physical layer specification as well as the " Interface Guidelines for DeviceNet Devices on Semiconductor Manufacturing Tools" which is published as part of the DeviceNet specification. This includes physical signaling (levels and baud rates - detailed in the CAN specification), transceivers, node isolation, media topology, cable specifications, network connectors and taps, and power considerations (load limits, system tolerances, and power supply options).

6.4.1 *Physical Layer Enhancements* — This standard supports the following physical layer enhancement to the DeviceNet specifications:

- The transmission media, including both trunk cable and drop cable, shall be DeviceNet Thin Cable. Specifications on this cable and its utilization in DeviceNet systems (e.g., maximum trunk and drop lengths) are included in the DeviceNet specification.

6.5 *Data Link Layer* — The device shall comply with the DeviceNet Data Link Layer Specifications (i.e., Controller Area Network Specification: Version 2). This includes the media access control mechanism and the logical link control mechanism. Addressing is currently limited to 11 bits.

6.6 *Network Layer* — There is no distinct network layer.

6.7 *Transport Layer* — There is no distinct transport layer. Some of the functionality of this layer is implemented in the Application Layer. Specific functions include: segmentation/reassembly for full message delivery and the establishment of node-to-node connections.

6.8 *Session Layer* — There is no distinct session layer.

6.9 *Presentation Layer* — There is no distinct presentation layer. Data types and data presentation in DeviceNet messages are specified as part of the DeviceNet object definitions and object attribute and service communication protocol.

6.10 *Application Layer* — The device shall comply with the DeviceNet application layer specification for defining and addressing objects, including their attributes and services, and enabling specified network behavior. The device shall comply with the object model specifications provided in the DeviceNet specification as well as the "Interface Guidelines for DeviceNet Devices on Semiconductor Manufacturing Tools" component of the DeviceNet specification. In addition, the device shall comply with the object specifications defined in Section 7 of this document.

6.10.1 *Object Models* — The DeviceNet protocol provides an object-oriented specification for creating, defining, and addressing objects explicitly, including their attributes and services (i.e., explicit messaging), and creating, defining, and communicating object attribute assemblies in an application-dependent format (i.e., input/output messaging). The device shall comply with the object model specifications provided in the DeviceNet documentation. In addition, the device shall comply with the object specifications defined in Section 7 of this document.

6.11 *Network Management* — The device shall comply with the DeviceNet network management specifications (e.g., physical layer bit rate, duplicate MAC ID detection, master-slave, and peer-to-peer network management). No (additional) network management functions are specified in this document.

7 Required Object Types

7.1 The DeviceNet specification identifies and describes objects (i.e., classes) that must exist in all DeviceNet-compliant devices. The Common Device Model specification additionally identifies two objects (namely the Device Manager (DM) and Sensor Actuator Controller (SAC) objects) that must exist in all SEMI-compliant SAN devices. The required object types for a SEMI-compliant SAN device, using the network communication specification described herein, necessarily comprise the union of the above to requirements.

7.2 A list of required and optional object types is given in Table 2. Note that the Sensor, Actuator, and Controller object types are not required and are indicated as optional in the CDM specification. These objects are aggregated together to form a SEMI- and DN- compliant device, as shown in Figure 4.

7.3 The mapping of the DM and SAC objects are combined into a single object in DeviceNet called the S-Device Supervisor (DS) object.

Table 2 Required Object Types

Object	*DN Class #**	*CDM Tag ***	*Required by DN **	*Required by CDM ***	*Required by NCS*
Identity	01	N.A.***	Yes	No	Yes
MR	02	N.A.	Yes	No	Yes
DN	03	N.A.	Yes	No	Yes
CNX	05	N.A.	Yes	No	Yes
DM(DS)	48	DmI0	No	Yes	Yes
SAC(DS)	48	SACI0	No	Yes	Yes
Sensor	****	SenIn	No	No	No
Actuator	****	ActIn	No	No	No
Controller	****	CntIn	No	No	No
(Other)	****	N.A.	No	No	No

* See DeviceNet specification for further information; values are hexadecimal.

** See CDM specification for further information.

*** Not applicable.

**** Application-dependent.

7.4 An embodiment of a specific device type represented as an aggregation of the object types listed in Table 2, that is compliant with both the CDM specification and the DN specification, is a candidate for a SEMI SDM as well as a DN Device Profile. Conversely, all SEMI SDM's and DN Device profiles specified for operation over a SEMI-compliant DN network must be an aggregation of the object types listed in Table 2 and be compliant with both the CDM specification and the DN specification.

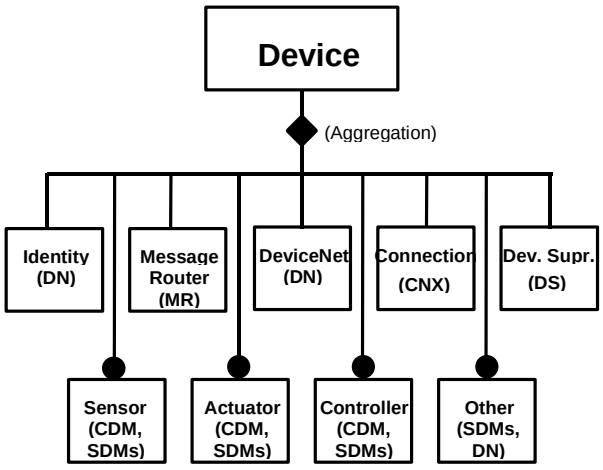

Figure 4
Aggregation of a Compliant Device

7.5 In the following sections, the presentation to the network of object addressing, object attributes, and object services for each of the object types listed in Table 2 and Figure 4 is described in detail.

7.6 *Identity Object* — This object provides identification of, and general information about, the device. The Identity Object must be present in all DeviceNet products. As specified by DeviceNet, each DeviceNet product shall support one (and only one) Identity object per physical connection to the DeviceNet communication link. Presentation of object identity, attributes, and services to the network for this object is described in the DeviceNet specification. Compliance with the DeviceNet specification shall constitute compliance with this NCS for the Identity Object.

7.7 *Message Router (MR) Object* — The MR Object provides a messaging connection point through which a service of any object class or instance residing in the physical device may be addressed. The MR object must be present in all DeviceNet products. As specified by DeviceNet, each DeviceNet product shall support one (and only one) Message Router object per physical connection to the DeviceNet communication link. Presentation of object identity, attributes, and services to the network for this object is described in the DeviceNet specification. Compliance with the DeviceNet specification shall constitute compliance with this NCS for the MR Object.

7.8 *DeviceNet (DN) Object* — The DN Object provides the configuration and status of a DeviceNet port. As specified by DeviceNet, each DeviceNet product shall support one (and only one) DN object per physical connection to the DeviceNet communication link. Presentation of object identity, attributes, and services to the network for this object is described in the DeviceNet specification. Compliance with the DeviceNet specification shall constitute compliance with this NCS for the DN Object.

7.9 *Connection (CNX) Object* — The CNX Object provides configuration and management of DeviceNet connections. A CNX object exists at each end of a DeviceNet connection (point-to-point or multicast). The CNX object handles the negotiation for connection establishment and manages a set of timers in order to handle cyclic traffic, connection timeout and fault containment and recovery. Several connection behaviors are supported including: explicit messaging, polled, cyclic, change-of-state and multicast messaging.

7.10 *S-Device Supervisor (DS) Object* — The DS object is the device component responsible for managing and consolidating the device operation as well as coordinating the interaction of the device with the sensory/actuation/control environment. Each device must support one (and only one) DS object. The DS

object combines both the DM object and the SAC object into a single object for DeviceNet. The DM and SAC, as well as their common required and optional attributes, services, and behavior, are described in the CDM standard. The presentation of object attributes and services to the DN network shall be as indicated in Table 3.

Table 3 Network Presentation of DM Object Attributes and Services

S-Device Supervisor Object - - Object ID == 48			
Attributes			
ID	**Name**		**CDM Tag**
3	Device Type		DmA1
4	Standard Revision Level		DmA2
5	Device Manufacturer Identifier		DmA3
6	Manufacturer Model Number		DmA4
7	Software or Firmware Revision Level		DmA5
8	Hardware Revision Level		DmA6
9	Serial Number (optional)		DmA7
10	Device Configuration (optional)		DmA8
11	Device Status		DmA9
-- *	Reporting Mode		DmA10
-- *	Exception Status Report Interval (optional)		DmA11
12	Exception Status		DmA12
13	Exception Detail Alarm (optional)		DmA13
14	Exception Detail Warning (optional)		DmA14
Services			
ID (hex)	**Name (SEMI)**	**Name (DN)**	**CDM Tag**
05	Reset	Reset	DmS1
4B	Abort	Abort	DmS2
4C	Recover	Recover	DmS3
0E	Get_Attribute	Get_Attribute_Single	DmS4
10	Set_Attribute	Set_Attribute_Single	DmS5
06	Execute	Start	DmS6
4E	Perform_Diagnostics	Perform_Diagnostics	DmS7

* There is no one-to-one mapping of the Reporting Mode attribute in DeviceNet. Instead, DeviceNet specifies several attributes, services and behaviors for its Connection Object to effect a superset of the behaviors associated with the DM attributes Reporting Mode and Exception Status Report Interval. See the DeviceNet specification for details.

7.10.1 Note that the format of DM object attributes is detailed in the CDM document; the presentation of DM object attributes to the DN network is detailed in Table 3 and the DN specification; the format of DM object services is detailed in the CDM document and the DN specification; and the presentation of the DM object services is detailed in Table 3 and the DN specification.

7.11 *Sensor Object, Actuator Object, Controller Object, and Other Object Types* — These object types are used collectively to model the type-specific structure and behavior of the device. The requirement and number of each of these object types in a device model is device type-specific. Further, the attributes, services, and behavior associated with each of these object classes and instances in a device is also device type-specific, but must be compliant with both SEMI and DN specifications. The specification of these object types for a specific device type can be found in the appropriate SDM. The method of presentation of object structure and behavior to the DN network for objects defined for, and associated with, a specific device type can be found in Section 9 of this document.

8 Protocol Compliance

8.1 A method of testing protocol compliance is required to verify implementation conformance to the standard. The test plan includes tests for duplicate address resolution, mandatory objects, etc.

8.2 The compliance test suite for this protocol necessarily includes the DeviceNet protocol compliance specification and test suite as well as the “Conformance Test Procedures for DeviceNet Devices on Semiconductor Manufacturing Tools” which is published as part of the DeviceNet specification. Additional compliance specification required for compliance to this NCS is provided as a set of Protocol Specification Sheets.

8.3 Any enhancements to DeviceNet presented in this document must be accepted by the Open DeviceNet Vendors Association (ODVA) and incorporated into the DeviceNet Specification before they can be implemented as a DeviceNet product.

8.4 *Protocol Specification Sheets* — Compliance to this NCS necessarily requires adherence to a set of protocol specification sheets. These sheets are included as Appendix 1 of this document and are included here for reference only. For compliance with this NCS, it is necessary to reference the latest revision of the Protocol Specification Sheets from ODVA. Note that these sheets provide statements of compliance for general device data, physical conformance data, communications data, required object implementation (see also Section 7 or this document), and optional object implementation. Note also that these specification sheets must conform with DeviceNet specifications for “Statement of Compliance” forms. Finally note that additional specification sheets shall be completed as necessary to specify compliance with objects defined in SDM’s and SDM mappings (see Section 9).

9 Specific Device Model Mappings

9.1 This section provides for the mapping of network-visible specific device structure and behavior, specified in a SEMI standard SDM specification, to the DN network. Each subsection is devoted to a single SDM specification (e.g., 9.1: Mass Flow Controller). Additional SDM mappings are added as sub-sections to this NCS specfication according to SEMI guidelines and the guidelines of the SEMI SAN Interoperability standard.

APPENDIX 1
PROTOCOL SPECIFICATION SHEETS

NOTICE: This appendix is not an official part of SEMI E54.4 and is not intended to modify or supercede the official standard. Determination of the suitability of the material is solely the responsibility of the user. This appendix contains a set of protocol specification sheets. Compliance to this NCS necessarily requires adherence to this set of protocol specification sheets (see Section 8.1).

A1-1 The protocol specification set of sheets contains the following components:

- A General Device Data/Physical ConformanceData/ DeviceNet Communication Data statement of compliance. (Form F_1, 1 page.)
- DeviceNet required object implementation statements of compliance (for Identity, Message Router, and DeviceNet objects, see Section 7). (Forms F_2 through F_5, 4 pages.)
- NCS - DeviceNet required object implementation statements of compliance (for Device Manager and Sensor/Actuator/Controller objects, see Section 7). (Form F_6, 2 pages.)
- Open object- and vendor- specific implementation templates (for utilization by SDMs, vendors, etc.). (Forms F_6 through F_7, 2 pages.)

A1-1.1 Note that these protocol specification sheets are aligned with DeviceNet specifications for “Statement of Compliance” documentation, available with the DeviceNet specification from ODVA. Detailed instructions on completing these forms are in this “Statement of Compliance” documentation. The set of protocol specification sheets begins on the following page.

DeviceNet **Statement of Compliance**

Complete this form using the definitions previously outlined.

					Fill in the blank or X the appropriate box
General Device Data	Conforms to DeviceNet Specification	Volume I - Release			_____
		Volume II - Release			_____
	Vendor Name				_____
	Device Profile Name				_____
	Product Catalog Number				_____
	Product Revision				_____
DeviceNet Physical Conformance Data	Network Power Consumption (Max)				_____A @11V dc (worst case)
	Connector Style	Open-Hardwired	❑	Sealed-Mini	❑
		Open-Pluggable	❑	Sealed-Micro	❑
	Isolated Physical Layer	Yes ❑			
		No ❑			
	LEDs Supported	Module	❑	Combo Mod/Net	❑
		Network	❑	I/O	❑
		DIP Switch	❑	Software-Settable	❑
		Other_____			
	Default MAC ID	_____			
	Communication Rate Setting	DIP Switch	❑	Software-Settable	❑
		Other_____			
	Communication Rates Supported	125k bit/s	❑	500k bit/s	❑
		250k bit/s	❑		
DeviceNet Communication Data	❑ Predefined Master/Slave Connection Set	Group 2 Client	❑	Group 2 Only Client	❑
		Group 2 Server	❑	Group 2 Only Server	❑
	❑ Dynamic Connections Supported (UCMM)	Group 1	❑	Group 3	❑
		Group 2	❑		
	Fragmented Explicit Messaging Implemented	Yes ❑			
		No ❑			
		If yes, Transmission Time Out_____ms			
		Typical Target Address		Class_____	
				Instance_____	
				Attribute_____	

X Get to indicate that attribute value is returned by the use of Get_Attribute_Single service.

X Set to indicate that attribute value is written to by the use of Set_Attribute_Single service.

F_ 1

DeviceNet			Statement of Compliance			
DeviceNet Required Object Implementation	**Identity Object 0x01**					
	Object Class		**ID**	**Description**	**Get**	**Set** **Value Limits**
	Attributes	Open	1	Revision	❑	❑ _____
			2	Max instance	❑	❑ _____
	❑ None Supported		6	Max ID of class attributes	❑	❑ _____
			7	Max ID of instance attributes	❑	❑ _____
			DeviceNet Services		**Parameter Options**	
	Services		❑	Get_Attribute_All	_____	
			❑	Reset	_____	
	❑ None Supported		❑	Get_Attribute_Single	_____	
			❑	Find_Next_Object_Instance	_____	
	Object Instance		**ID**	**Description**	**Get**	**Set** **Value Limits**
	Attributes	Open	1	Vendor	❑	❑ _____
			2	Product type	❑	❑ _____
	❑ None Supported		3	Product code	❑	❑ _____
			4	Revision	❑	❑ _____
			5	Status	❑	❑ _____
			6	Serial number	❑	❑ _____
			7	Product name	❑	❑ _____ _____
			8	State	❑	❑ _____
			DeviceNet Services		**Parameter Options**	
	Services		❑ Reset		_____	
			❑ Get_Attribute_All		_____	
	❑ None Supported					
	Vendor-Specific Additions		If yes, fill out the Vendor-Specific Additions form on page F_7.		Yes ❑ No ❑	

X Get to indicate that attribute value is returned by the use of Get_Attribute_Single service.

X Set to indicate that attribute value is written to by the use of Set_Attribute_Single service.

DeviceNet			Statement of Compliance			
DeviceNet			**Message Router Object 0x02**			
Required Object Implementation	**Object Class**		**ID**	**Description**	**Get**	**Set** **Value Limits**
	Attributes	Open	1	Revision	❑	❑ _____
			4	Optional attribute list	❑	❑ _____
	❑ None Supported		5	Optional service list	❑	❑ _____
			6	Max ID of class attributes	❑	❑ _____
			7	Max ID of instance attributes	❑	❑ _____
			DeviceNet Services		**Parameter Options**	
	Services		❑	Get_Attribute_all	_____	
			❑	Get_Attribute_Single	_____	
	❑ None Supported					
	Object Instance		**ID**	**Description**	**Get**	**Set** **Value Limits**
	Attributes	Open	1	Object list	❑	❑ _____
			2	Maximum connections supported	❑	❑ _____
	❑ None Supported		3	Number of active connections	❑	❑ _____
			4	Active connections list	❑	❑ _____
			DeviceNet Services		**Parameter Options**	
	Services		❑	Get_Attribute_All	_____	
			❑	Get_Attribute_Single	_____	
	❑ None Supported					
	Vendor-Specific Additions		If yes, fill out the Vendor-Specific Additions form on page F_7.		Yes ❑	No ❑

X Get to indicate that attribute value is returned by the use of Get_Attribute_Single service.

X Set to indicate that attribute value is written to by the use of Set_Attribute_Single service.

DeviceNet			Statement of Compliance		
DeviceNet Required Object Implementation	**DeviceNet Object 0x03**				
	Object Class		**ID**	**Description**	**Get Set Value Limits**
	Attributes	Open	1	Revision	❑ ❑ _____
	❑ None Supported				
			DeviceNet Services		**Parameter Options**
	Services		❑	Get_Attribute_Single	_____
	❑ None Supported				
	Object Instance		**ID**	**Description**	**Get Set Value Limits**
	Attributes	Open	1	MAC ID	❑ ❑ _____
			2	Baud rate	❑ ❑ _____
	❑ None Supported		3	BOI	❑ ❑ _____
			4	Bus-off counter	❑ ❑ _____
			5	Allocation information	❑ ❑ _____
			6	MAC ID switch changed	❑ ❑ _____
			7	Baud rate switch changed	❑ ❑ _____
			8	MAC ID switch value	❑ ❑ _____
			9	Baud rate switch value	❑ ❑ _____
			DeviceNet Services		**Parameter Options**
	Services		❑	Get_Attribute_Single	_____
			❑	Set_Attribute_Single	_____
	❑ None Supported		❑	Allocate M/S connection set	_____
			❑	Release M/S connection set	_____
	Vendor-Specific Additions		If yes, fill out Vendor-Specific		Yes ❑
			Additions form on page F_7.		No ❑

X Get to indicate that attribute value is returned by the use of Get_Attribute_Single service.

X Set to indicate that attribute value is written to by the use of Set_Attribute_Single service.

DeviceNet	Statement of Compliance					
DeviceNet Required Object Implementation	**Connection Object 0x05**					
	Object Class		**ID**	**Description**	**Get Set Value Limits**	
	Attributes ❑ None Supported	Open	1	Revision	❑ ❑	_____
			DeviceNet Services		**Parameter Options**	
	Services ❑None Supported		❑	Reset		_____
			❑	Create		_____
			❑	Delete		_____
			❑	Get_Attribute_Single		_____
			❑	Find_Next_Object_Instance		_____
	Total Active Connections Possible		_____			
	Object Instance		**Section**		**Information Max**	
	The Object Instance section must be completed for each combination of Instance type, Production trigger, Transport type, and Transport class supported		Instance type		Explicit Message ❑	_____
					Polled I/O ❑	_____
					Bit Strobed I/O ❑	_____
					Dynamic I/O ❑	_____
			Production trigger		Cyclic ❑	_____
					Change of State ❑	_____
					Application Trig. ❑	_____
			Transport type		Server ❑	_____
					Client ❑	_____
			Transport class		0 ❑	_____
					2 ❑	_____
					3 ❑	_____
			ID	**Description**	**Get Set Value Limits**	
	Attributes	Open	1	State	❑ ❑	_____
			2	Instance type	❑ ❑	_____
			3	Transport class trigger	❑ ❑	_____
			4	Produced connection ID	❑ ❑	_____
			5	Consumed connection ID	❑ ❑	_____
			6	Initial comm. characteristics	❑ ❑	_____
			7	Produced connection size	❑ ❑	_____
			8	Consumed connection size	❑ ❑	_____
			9	Expected packet rate	❑ ❑	_____
			12	Watchdog time-out action	❑ ❑	_____
			13	Produced connection path length	❑ ❑	_____
			14	Produced connection path	❑ ❑	_____
			15	Consumed connection path length	❑ ❑	_____
			16	Consumed connection path	❑ ❑	_____
			DeviceNet Services		**Parameter Options**	
	Services		❑	Reset		_____
			❑	Delete		_____
			❑	Apply_Attributes		_____
			❑	Get_Attribute_Single		_____
			❑	Set_Attribute_Single		_____
	Vendor-Specific Additions		If yes, fill out the Vendor-Specific Additions form on page F_7.		Yes ❑ No ❑	

X Get to indicate that attribute value is returned by the use of Get_Attribute_Single service.

X Set to indicate that attribute value is written to by the use of Set_Attribute_Single service.

DeviceNet Open Object Implementation

OBJECT NAME Device Manager **OBJECT ID** 64

Object Class		**ID**	**Description**	**Get**	**Set** Value Limits
Attributes	Open	_____	_____	❑	❑ _____
		DeviceNet Services		**Parameter Options**	
Services		_____		_____	
Object Instance		**ID**	**Description**	**Get**	**Set** Value Limits
Attributes	Open	31	Device Type	❑	❑ _____
		32	Standard Revision Level	❑	❑ _____
		33	Device Manufacturer ID	❑	❑ _____
		34	Manufacturer Model Number	❑	❑ _____
		35	Soft/Firmware Rev. Level	❑	❑ _____
		36	Hardware Revision Level	❑	❑ _____
		37	Serial Number	❑	❑ _____
		38	Device Configuration	❑	❑ _____
		39	Device Status	❑	❑ _____
		3A	Reporting Mode	❑	❑ _____
		3B	Exception Status Timer	❑	❑ _____
		3C	Exception Status	❑	❑ _____
		3D	Exception Detail Alarm	❑	❑ _____
		3E	Exception Detail Warning	❑	❑ _____
		DeviceNet Services		**Parameter Options**	
Services		❑	Reset	_____	
		❑	Abort	_____	
		❑	Recover	_____	
		❑	Get_Attribute_Single	_____	
		❑	Set_Attribute_Single	_____	
		❑	Execute	_____	
		❑	Perform Diagnostics	_____	
Meaning of Zero Length I/O Data Production_____					
Vendor-Specific Additions		If yes, fill out the Vendor-Specific Additions form on page F_7.		Yes ❑ No ❑	

X Get to indicate that attribute value is returned by the use of Get_Attribute_Single service.

X Set to indicate that attribute value is written by the use of Set_Attribute_Single service.

DeviceNet	OBJECT NAME Sensor Actuator Controller		OBJECT ID 66	
Open Object Implementation	**Object Class**	**ID Description**	**Get Set Value Limits**	
	Attributes Open	_____ _____	❑ ❑ _____	
		_____ _____	❑ ❑ _____	
		_____ _____	❑ ❑ _____	
		DeviceNet Services	**Parameter Options**	
	Services	_____	_____	
		_____	_____	
		_____	_____	
	Object Instance	**ID Description**	**Get Set Value Limits**	
	Attributes Open	_____	❑ ❑ _____	
		_____	❑ ❑ _____	
		_____	❑ ❑ _____	
		DeviceNet Services	**Parameter Options**	
	Services	❑ Reset	_____	
		❑ Abort	_____	
		❑ Recover	_____	
		❑ Get_Attribute_Single	_____	
		❑ Set_Attribute_Single	_____	
		_____	_____	
		_____	_____	
		_____	_____	
	Meaning of Zero Length I/O Data Production_____			
	Vendor-Specific Additions	If yes, fill out the Vendor-Specific Additions form on page F_7.	Yes ❑	
			No ❑	

X Get to indicate that attribute value is returned by the use of Get_Attribute_Single service.

X Set to indicate that attribute value is written to by the use of Set_Attribute_Single service.

DeviceNet	**OBJECT NAME**_____				**OBJECT ID**_____	
Open Object Implementation	**Object Class**		**ID Description**		**Get Set Value Limits**	
	Attributes	Open	_____	_____	❑ ❑	_____
			_____	_____	❑ ❑	_____
			_____	_____	❑ ❑	_____
			DeviceNet Services		**Parameter Options**	
	Service		_____		_____	
			_____		_____	
			_____		_____	
	Object Instance		**ID Description**		**Get Set Value Limits**	
	Attributes	Open	1	_____	❑ ❑	_____
			2	_____	❑ ❑	_____
			3	_____	❑ ❑	_____
			DeviceNet Services		**Parameter Options**	
	Services		_____		_____	
			_____		_____	
			_____		_____	
	Meaning of Zero Length I/O Data Production_____					
	Vendor-SpecificAdditions		If yes, fill out the Vendor-Specific		Yes ❑	
			Additions form on page F_7.		No ❑	

X Get to indicate that attribute value is returned by the use of Get_Attribute_Single service.

X Set indicate that attribute value is written to by the use of Set_Attribute_Single service.

F_ 8

	❑ Extension to Open Object		
	❑ Vendor-Specific Object		
Vendor Specific Object Implementation	**OBJECT NAME**_____		**OBJECT ID**_____
	Object Class	**ID Description**	**Get Set Value Limits**
	Attributes	_____ _____	❑ ❑ _____
		_____ _____	❑ ❑ _____
		_____ _____	❑ ❑ _____
		Code (Hex) Service Description	**Parameter Type/Options**
	Services	_____ _____	_____
		_____ _____	_____
		_____ _____	_____
	Object Instance	**ID Description**	**Get Set Type/Value Limits**
	Attributes	_____ _____	❑ ❑ _____
		_____ _____	❑ ❑ _____
		_____ _____	❑ ❑ _____
		Code (Hex) Service Description	**Parameter Type/Options**
	Services	_____ _____	_____
		_____ _____	_____
		_____ _____	_____
	Meaning of Zero Length I/O Data Production_____		

X Get to indicate that attribute value is returned by the use of Get_Attribute_Single service.

X Set to indicate that attribute value is written to by the use of Set_Attribute_Single service.

F_ 9

NOTICE: SEMI makes no warranties or representations as to the suitability of the standards set forth herein for any particular application. The determination of the suitability of the standard is solely the responsibility of the user. Users are cautioned to refer to manufacturer's instructions, product labels, product data sheets, and other relevant literature respecting any materials mentioned herein. These standards are subject to change without notice.

The user's attention is called to the possibility that compliance with this standard may require use of copyrighted material or of an invention covered by patent rights. By publication of this standard, SEMI takes no position respecting the validity of any patent rights or copyrights asserted in connection with any item mentioned in this standard. Users of this standard are expressly advised that determination of any such patent rights or copyrights, and the risk of infringement of such rights, are entirely their own responsibility.

SEMI E54.5-0997 (Withdrawn 0704)
STANDARD FOR SENSOR/ACTUATOR NETWORK COMMUNICATIONS FOR THE SMART DISTRIBUTED SYSTEM (SDS)

NOTE: This document was previously designated SEMI E60. Because this document is part of a suite of documents, its designation has been reassigned for ease of reference. Please note that the technical content of this document is unchanged from the 0697 version.

NOTICE: This document was balloted and approved for withdrawal in 2004.

1 Purpose

This standard defines a communication protocol based on the Smart Distributed System (SDS) to enable communications between intelligent devices on a sensor/actuator network (SAN) to be used in semiconductor manufacturing equipment.

1.1 *Background and Motivation* — SDS provides interconnection of smart control devices such as sensors, actuators, and controllers in a fast-response time, low-cost network for industrial use. SDS enables multiple devices to share a single bus, thereby significantly reducing the point-to-point wiring between controllers, sensors, and actuators. The SDS system is based on CAN, the controller area network used in the automotive industry and defined by Bosch.

2 Scope

This document specifies a SAN communications standard based on the Smart Distributed System (SDS) specification that is in compliance with the SEMI SAN Common Device Model specification.

2.1 This document specifies the protocol and services that compliant intelligent devices must support to interchange information over this semiconductor equipment sensor/actuator network.

2.2 This document is used in conjunction with a SEMI standard SAN Common Device Model specification and one or more SEMI standard specific device model specifications (e.g., for a mass flow controller). Together, the model documents describe the data structure and behavior that are characteristic of the various devices on the network. This SAN communications standard identifies the protocol for the interaction with such a device over the network to make the data structures and behavior available to other devices.

2.3 This standard, together with a sensor/actuator network interoperability guideline, the sensor/actuator network common device model, one or more sensor/actuator network specific device model documents, and the SDS specifications, form a complete interoperability standard. The general sensor/actuator network document architecture is shown in the Sensor/Actuator Network Common Device Model document in Figure 1.

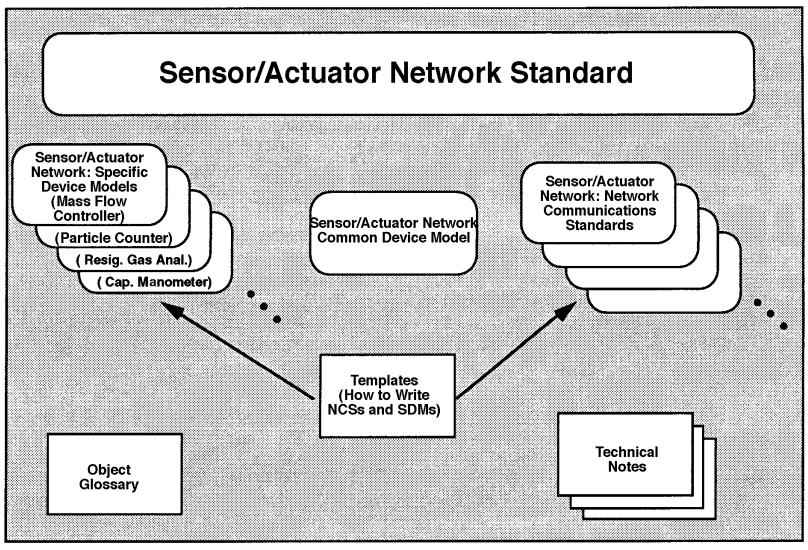

Figure 1
Sensor/Actuator Network Related Documents

The SDS Network communications standard document structure is shown in Figure 2. This standard specifies the mapping of the SAN common device model onto the SDS-specific network. In addition, the document will include mappings to specific device models (e.g., the mass flow controller). The latter mappings will be included in this document as the specific device models are defined.

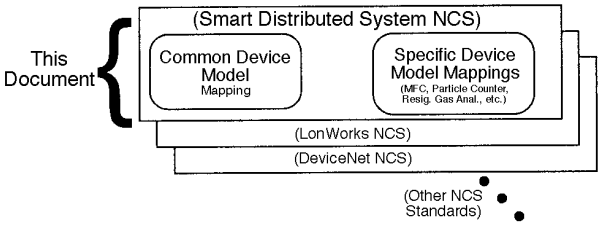

Figure 2
SDS Network Communications Standard Document Structure

3 Limitations

3.1 This document specifies a semiconductor equipment SAN based solely on SDS and is a companion document to the SDS specification; thus, a complete specification of this standard necessarily includes the SDS 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 SDS. In order to avoid document consistency problems, information in the SDS specification that relates to this standard is not repeated in this document. This document is limited to describing enhancements or limitations to the SDS specification that are imposed by this standard.

3.3 A complete specification of the conformance testing procedure shall include the SDS protocol conformance testing specification. Conformance testing shall also include enhancements and limitations to the SDS specification required by this standard.

4 Referenced Standards

4.1 *SEMI Standards*

SEMI E30 — Generic Model for Communications and Control of SEMI Equipment (GEM)

SEMI E39 — Object Services Standard: Concepts, Behavior, and Services

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

4.2 *ISO Standards*[1]

ISO 7498 — Basic Reference Model for Open Systems Interconnection

ISO 11898 — Road Vehicles — Interchange of Digital Information — Controller Area Network (CAN) for High-Speed Communications

4.3 *Other Documents*

Doc 84-08420-1 (GS 052 103) — SMART DISTRIBUTED SYSTEM Application Layer Protocol Specification, Honeywell MICRO SWITCH, February 15, 1995

Doc 84-08421-A (GS 052 104) — SDS Physical Layer Specification, Honeywell MICRO SWITCH, December 15, 1994

Doc 85-08453-0 (GS 052 107) — SMART DISTRIBUTED SYSTEM Component Model Specification, Honeywell MICRO SWITCH, January 29, 1995

Doc GS 052 108 Issue 1 — SMART DISTRIBUTED SYSTEM Conformance Test Procedure Specification, Honeywell MICRO SWITCH, January 2, 1995

Controller Area Network Specification: Version 2, R. Bosch GmbH, + Postfach 50 D-7000, Stuttgart 1, Germany, 1991

5 Terminology

Terminology that is common to all of the documents in this SAN series may also be defined in the *Sensor Actuator Network Interoperability Guideline.* Terminology may be reproduced here which is defined in other SEMI documents.

5.1 *Device Component Definitions* — This standard is based on the concepts developed in the Sensor Actuator Network Common Device Model (SEMI E54) document and makes use of the following terms defined in it:

a) device

b) device model

c) object

d) instance

e) attribute

f) behavior

g) service

[1] International Organization for Standardization, 1 rue de Varembe, Case postale 56, CH-1211 Geneva 20, Switzerland

h) state diagram

i) byte

j) nibble

k) character string

5.2 *SDS-Specific Definitions* — In addition to the above terms, the following terms are defined for SDS networks.

5.2.1 *actions* — operations that a client may request an Embedded Object to perform. The action typically modifies the state of the device. Actions are more powerful than writing to an attribute, in that multiple input arguments may be provided as part of an action request. Also, results of the action are typically returned. Actions, like attributes, have identifiers that are specific to the type of the object. However, these identifiers are independent from attribute or event identifiers. The "request" services specified in SEMI E54 are implemented as SDS actions.

5.2.2 *controller area network (CAN)* — a protocol developed by the Bosch corporation for automotive in-vehicle networking. The CAN specification specifies OSI reference model layers 1 and 2, specifically the physical signaling and media access/data link protocols.

5.2.3 *embedded objects* — each SDS Logical Device contains at least one, and at most 32, Embedded Objects. An SDS Embedded Object is an abstraction representing an addressable entity "embedded" within a Logical Device having specific application-related interface characteristics. These characteristics include a defined set of attributes, actions, and events that are specific to the Embedded Object. Combinations of these Embedded Objects provide a mechanism for describing an arbitrary sensor/actuator network device. The behaviors required by SEMI E54 are specified as part of the definition of attributes, actions, and events.

5.2.4 *events* — Are used by objects to asynchronously report the occurrence of events within an Embedded Object. Event definitions specify the event reports that a specific Embedded Object type may emit. Events also have type-specific identifiers. Event definitions include: a text string defining the semantics of the event, an event identifier numerical value and textual name, and a syntax definition. The "notification" services specified in SEMI E54 are implemented as SDS events.

5.2.5 *logical device* — It is possible to have one or more independent Logical Devices within the Physical Component. This provides the illusion of multiple devices on the network that actually are implemented on the same physical hardware. Logical Devices are distinguished within the Physical Component using unique SDS bus addresses. The Logical Device defines a separately addressable entity within a Physical Component that has an independent interface definition.

5.2.6 *physical component* — An SDS Physical Component is an abstraction representing a single physical package of hardware and software that is connected to the network. This is equivalent to the CDM SEMI E54 definition for "device." The Physical Component contains one or more Logical Devices.

Table 1 provides a mapping of SEMI E54 definitions to SDS-specific definitions. Column 2 contains an equal sign "=" if the definition is used exactly as specified in SEMI E54. Otherwise, the appropriate SDS-specific term(s) are identified.

Table 1 Table 1 Mapping of Terms Between SEMI E54 and SDS

SEMI E54 Term	*SDS Term*
Device	=
Device Model	Physical Component
Object	Embedded Object
Instance	=
Attribute	=
Behavior	Set of Attribute/Action/Event definitions
Service	Actions for Request Services and Events for Notification Services
State Diagram	=
Byte	=
Nibble	=
Character String	=

6 Communication Protocol High Level Structure

The SDS protocol is based on a three-layer architecture. These layers constitute a collapsed form of the OSI seven-layer architecture, mapping into the physical, datalink, and application layers of the Open Systems Interconnection Reference Model. The high level protocol architecture is shown in Figure 3.

Note that Figure 3 represents a conceptual view of the device architecture. Conforming implementations must implement the services defined in this specification at each layer and must appear (from the network) to have implemented this architecture; however, an internal modular partitioning is not required. Implementations may sacrifice modularity in order to achieve high performance.

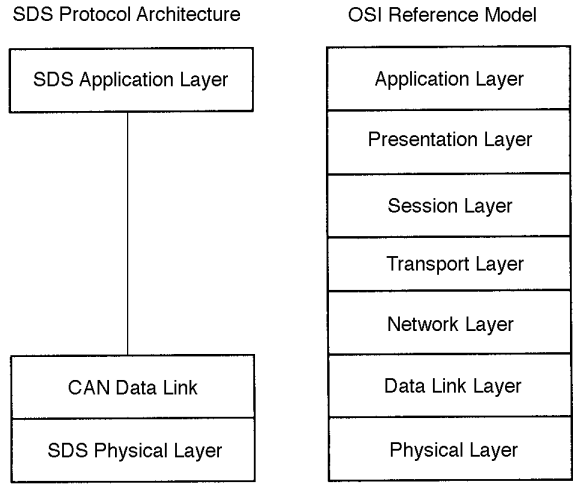

Figure 3
Protocol Architecture

SDS uses a three-layer protocol stack consisting of the physical, datalink, and application layers. The SDS physical layer uses two twisted pair to deliver power and data to smart devices at speeds ranging from 125 Kbps to 1 Mbps and distances up to 500 meters. At the data link layer, the CAN specification defines a carrier sense multiple access mechanism for media access control that avoids collisions and sends frames reliably. The application layer supports a concise set of services to set and get attributes of objects, to invoke operations on objects, and to report notifications from objects. These services are optimized for high performance in discrete control applications (e.g., 3 byte messages convey digital I/O state changes). In addition, a set of pre-defined and extensible component objects flexibly specifies all network-visible application level device capabilities.

6.1 *Physical Layer* — The device shall comply with the SDS physical layer specifications. These include physical signaling (levels and baud rates - detailed in the CAN specification), transceivers, node isolation, media topology, cable specifications, network connectors and taps, and power considerations (load limits, system tolerances, and power supply options).

6.2 *Data Link Layer* — The device shall comply with the SDS Data Link Layer Specifications (i.e., Controller Area Network Specification: Version 2). These include the media access control mechanism and the logical link control mechanism. Frame formats, interframe spacing, and error signaling shall comply with the CAN specifications. CAN frame identifiers shall be 11 bits in length.

6.3 *Network Layer* — There is no distinct network layer. A future extension of the SDS SEMI SAN protocol will support inter network messaging.

6.4 *Transport Layer* — There is no distinct transport layer. Specific functionality of this layer is implemented in the Application Layer. Functions include: segmentation and reassembly for large message delivery.

6.5 *Session Layer* — There is no distinct session layer.

6.6 *Presentation Layer* — There is no distinct presentation layer. Data types are specified as part of the SDS object definitions.

6.7 *Application Layer* — The device shall comply with the SDS application layer specification. This includes application object to application object communication mechanisms.

6.7.1 *Object Models* — The SDS protocol provides an object-oriented specification for defining and addressing objects, including their attributes, actions, and events. The device shall comply with the object model specifications provided in the SDS Component Model Specification. In addition, the device shall comply with the object specifications defined in Section 7 of this document.

6.8 *Network Management* — The device shall comply with the SDS system and network management specifications.

7 Required Object Types

The Common Device Model specification identifies the objects that must be supported in SEMI SAN-compliant devices. These objects are specified in Table 1 of SEMI E54 and include the following:

a) Device Manager (DM)

b) Sensor/Actuator/Controller (SAC)

c) Sensor (S_i)

d) Actuator (A_i)

e) Controller (C_i).

As specified in SEMI E54, a conforming device must support the DM, the SAC, and one or more of the remaining object types.

This section specifies the implementation of these required and optional objects in the SDS object model. SDS object types are defined for the DM and SAC objects. Sensor, actuator, or controller objects may be implemented using existing standard SDS object types, or new types may be specified, depending on the specific device model needs.

7.1 *Embedded Object Type Hierarchy* — SDS classifies Embedded Objects into specific object types (classes), and these types are organized into a class inheritance hierarchy. (For more information on classification and inheritance, see SEMI E39 -- Object Services Standard). Each SDS object type specifies the details of an object's attributes, actions, and events. The Embedded Object (an instance of a type) is a network-addressable entity within a logical device. The classification identifies common characteristics and enables reuse of object definitions. The top two levels of that hierarchy are shown in Figure 4 below.

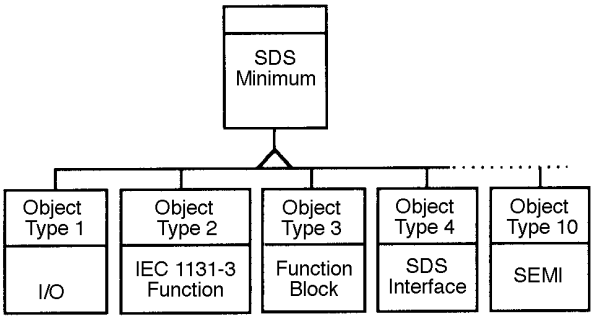

Figure 4
Top Levels of the SDS Embedded Object Type Hierarchy

The SDS object types are specified in detail in the SDS Component Modeling Specification. There are four primary Embedded Object types in the SDS type hierarchy. These include: I/O, IEC 1131-3 Function Object, Function Block, and SDS Interface. In addition, a new SEMI object type has been defined for use in developing classes necessary for SEMI SAN-compliant networks. Each of these object types is derived from a common type called SDS Minimum, which defines the minimum characteristics that are common to all SDS Embedded Objects.

All SDS object types inherit from the SDS Minimum object type, which provides the fundamental characteristics of an SDS device. The other top level SDS object types shown in Figure 4 are summarized below.

I/O Object (Type 1) — Defines the general characteristics of objects used for process I/O type functions. Subtypes of this type specify input and output objects with either analog or digital process variables. These may be used in SAN devices as the sensor or actuator objects.

IEC 1131-3 Function Object (Type 2) — Defines standard IEC 1131-3 Function Objects such as counters, timers, type conversion, bit-wise operations, character strings, etc.

Function Block Object (Type 3) — Collections of function objects that have a specific set of inputs and outputs (e.g., a predefined control algorithm). These may be used in the SAN devices as the controller objects.

SDS Interface Object (Type 4) — Models, network interfaces, gateways, etc.

SEMI Object (object Type 10) — Implement the additional object types necessary for SEMI SAN-compliant networks.

7.1.1 *SEMI Objects* — To implement SEMI-specific object types, the SDS object type hierarchy is augmented with type number 10 - SEMI object. This type, and the types derived from it, specify the new functionality necessary to implement this SEMI standard. Object type 10 is defined to include the common required attributes, services, and behaviors as described in the SAN Common Device Model standard SEMI E54. This definition is in terms of SDS Attributes, Actions, and Events.

In addition, there are derived types: types 10.1 implementing the SAC object and 10.2 implementing the Device Manager (DM) object (Figure 5). Object instances of these types are required in the Common Device Model document SEMI E54.

Each object type inheriting from the SEMI Object type 10 (e.g., 10.1 SAC object) automatically includes the features of the SEMI Object. Thus, both the SAC and the DM object have the reset, abort, and recover actions that are defined for the SEMI Object type 10. (See Section 7.2.) The Device Manager type augments the SEMI Object to provide the required characteristics over and above those specified in type 10 and SDS minimum.

As specific device models are defined, the SAC object type will be refined to be specific to that device. These object types are indicated in Figure 5 as object types SDM-1, SDM-2, etc. (examples include: Mass Flow Controller, Particle Counter, and Capacitance Manometer SAC objects). These object types have (at least) the set of attributes, actions, and events inherited from SDS Minimum, the SEMI Object type, and the SAC object type. Additionally, each SDM-n may have other, unique attributes, actions, and events which implement the desired behavior of a specific device SAC object.

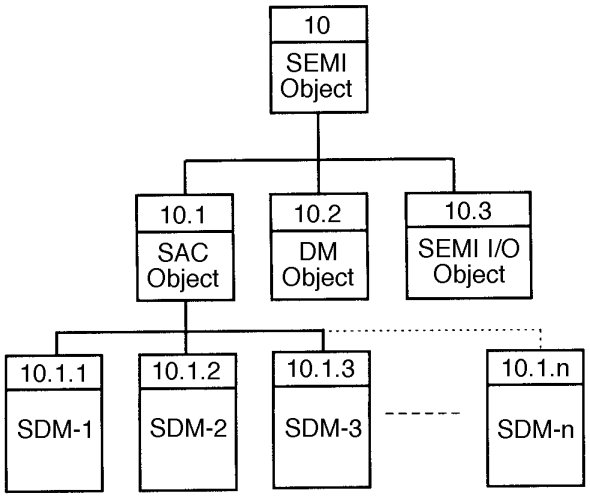

Figure 5
Second and Third Levels of the SEMI Object Type Hierarchy

The optional sensor and actuator objects described in the SEMI E54 specification may be instances of existing SDS object types (e.g., of type I/O object in Figure 4). In cases where a desired object type is not available under SDS object type 1, new I/O object types may be defined as sub-types of object type 10.3 SEMI I/O object.

7.1.2 *Example SEMI Device* — For clarification purposes, an example SEMI device is shown in Figure 6 (using Rumbaugh notation). It consists of three (or more) objects, including a DM, SAC, and binary input object. This is a complete and fully functional SEMI SAN-compliant digital input device.

In Figure 6, the Device is assigned an SDS address (e.g., 125), and each object is assigned an object instance number. The device address may be changed during installation. Each SDS message contains a device address and an object instance number.

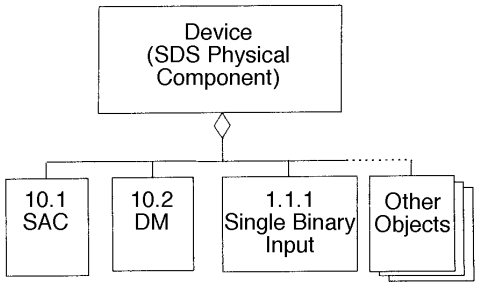

Figure 6
Example Device with Multiple Objects

7.2 *SEMI Object Type 10 Definition* — Table 2 formally specifies the SEMI Object (type 10). The three sections in this table specify SDS identifiers to code the required attributes, actions, or events. The first column is the SDS identifier number, the unique identifier used in SDS messaging. The second column is the SEMI E54-specified name associated with the ID#. The third column identifies the associated tag name (identifier) used in SEMI E54.

Table 2 Table 2 SEMI Object Common Level Model(Object Type 10)

Type 10 SEMI Object Common Structure and Behavior		
Attributes		
ID	Name	SEMI E54 tag
	None defined.	
80–144	SEMI Reserved Attribute IDs	Reserve
Actions		
ID	Name	SEMI E54 tag
80	Reset	SacS1/DmS1
81	Abort	SacS2/DmS1
82	Recover	SacS3/DmS3
83–95	SEMI Reserved Action IDs	Reserve
Events		
ID	Name	SEMI E54 tag
	None defined.	
80–95	SEMI Reserved Event IDs	Reserve

There are no attributes that are currently identified in SEMI E54 that are common to all SEMI objects. As a result, there are no attributes defined in the table. However, 64 attribute IDs have been reserved for use in other sub-types derived from type 10 in this document and in future revisions of this document.

SEMI E54 defines three service requests (reset, abort, and recover) that are common to the Device Manager and the SAC objects. These service requests are implemented as specific SDS Actions. Each service is mapped to a designated SDS action in Table 2. The actions are identified as ID# 80, 81, and 82 respectively. Additional action identifiers are reserved for use in sub-types of this type and for future use.

Not explicitly shown as Actions are the SEMI E54 Get and Set Attribute services. The equivalent behavior is provided by the SDS Read and Write Attribute services that are implicit for every attribute.

The execution of a service request may cause a change in the object's state variable. When a device receives an SDS action message indicating a service request, the object transitions to the appropriate state, after which an SDS response message is transmitted to the service

requester indicating the response to the request. For example, the SAC Object Instance Behavior State Transition Matrix in Table 4 of the SEMI E54 document defines the valid SAC state transitions.

Service Notifications (as defined in SEMI E54) are implemented with specific SDS event messages which are specified in the third part of the table. The SEMI object (type 10) has no events defined at this time. Additional event identifiers are reserved for use in sub-types of this type and for future use.

7.3 *Implementation of Sensor/Actuator/Controller Object Type (SAC)* — A single instance of a SAC object type is required in each SAN device. The actual object type used will typically be derived from the SAC object type rather than this exact type. The derived type will have added services and attributes that are device-specific (according to a Specific Device Model specification). The generic SAC type is, nevertheless, defined as a semantic type definition (sensor actuator controller object). The features of the SAC object include service requests (for Reset, Abort, Recover, etc.) that are mapped to SDS Action functions. Since these have been defined in the super type (object type 10), they are not re-defined here.

7.3.1 *SDS Object Model for Sensor/Actuator/Controller Object Type (SAC)* — The SDS object model for the SAC object type (Table 3) needs only to include the appropriate mapping for SAC-specific attributes and behavior (i.e., augmenting the SEMI Object Common Level - Table 8-1). The SAC Object Model contains no attributes, actions, or events beyond those specified in its super type (type 10).

An implementor of a specific device will normally derive a new SAC object type and add new attributes, actions, or events as necessary (i.e., 10.1.1 MFC SAC Object). Object type 10.1 is, in essence, a semantic grouping only—for a group of device-specific SAC object types implementing specific behavior.

The default object identifier that should be used for the SAC object in an SDS device is 1. The actual object identifier used may be reassigned by the implementor. The identifier may be determined on line via reading the object type attributes from all objects on the device.

Table 3 Table 3 SAC Object Model (Object Type 10.1)

Type 10.1 Sensor/Actuator/Controller Object Model		
Attributes		
ID	Name	SEMI E54 tag
	None defined.	
Actions		
ID	Name	SEMI E54 tag
	None defined.	
Events		
ID	Name	SEMI E54 tag
	None defined.	

7.4 *Implementation of Device Manager Object Type (DM)* — A single instance of this type is required on each device.

7.4.1 *SDS Object Model for Device Manager Object Type (DM)* — The SDS object model for the DM object type includes the appropriate mapping for DM-specific attributes and behavior to SDS-specific identifiers (Table 4).

Table 4 Table 4 DM Object Model (Object Type 10.2)

Type 10.2 Device Manager (DM) Object Model		
Attributes		
ID	Name	SEMI E54 tag
81	Device Type	DmA1
82	Standard Revision Level	DmA2
83	Device Manufacturer Identifier	DmA3
84	Manufacturer Model Number	DmA4
85	Software or Firmware Revision Level	DmA5
86	Hardware Revision Level	DmA6
87	Serial Number (optional)	DmA7
88	Device Configuration (optional)	DmA8
89	Device Status	DmA9
90	Reporting Mode	DmA10
91	Exception Status Timer (optional)	DmA11
92	Exception Status	DmA12
93	Exception Detail Alarm (optional)	DmA13
94	Exception Detail Warning (optional)	DmA14
Actions		
ID	Name	SEMI E54 tag
83	Execute	DmS6
84	PerformDiagnostics	DmS7
Events		
ID	Name	SEMI E54 tag
81	Publish Attribute	DmS8

The DM Object Model maps exactly to the specification for the DM in SEMI E54. Refer to that document for detailed descriptions for the attributes, actions, and events defined in Table 4. The SEMI E54 tag column indicates the identifier used in SEMI E54.

The default object identifier that should be used for the DM object in an SDS device is 2. The actual object identifier used may be reassigned by the implementer.

7.5 *Implementation of Sensor Object Type (S_i)* — Zero or more instances of this object type are permitted on each device.

Sensor object types are already defined within the SDS object hierarchy (e.g., Object Type 1 - I/O Device). Implementation of a device which is compliant with this type results in compliance with this standard.

If the existing sensor object types defined in the SDS specifications do not meet the requirements of the application, new SEMI-specific types may be defined deriving from type 10.3. If new types are required that are generic (i.e., not specific to SEMI), they should be derived from type 1 - I/O Device.

7.6 *Implementation of Actuator Object Type (A_i)* — Zero or more instances of this object type are permitted on each device.

Actuator object types are already defined within the SDS object hierarchy (e.g., Object Type 1 - I/O Device). Implementation of a device which is compliant with this type results in compliance with this standard.

If the existing actuator object types defined in the SDS specifications do not meet the requirements of the application, new SEMI-specific object types may be defined deriving from type 10.3. If new types are required that are generic (i.e., not specific to SEMI), they should be derived from type 1 - I/O Device.

7.7 *Implementation of Controller Object Type (C_i)* — Zero or more instances of this object type are permitted on each device.

Specific controller object types are defined within the SDS object hierarchy (e.g., Object type 3 - Function Block Object). Implementation of a device which is compliant with these types results in compliance with this standard.

If the controller types defined in the SDS specifications do not meet the requirements of the application, new SEMI-specific object types may be defined deriving from type 10. If new types are required that are generic (i.e., not specific to SEMI), they should be derived from type 3 - Function Block Object.

8 Protocol Compliance

A method of testing protocol compliance is required to verify conformance to this standard. The device must satisfy the SDS protocol conformance requirements as documented in the SDS specifications. The SDS partners group provides a conformance verification test procedure service to its members. When this SEMI Sensor/Actuator Network standard is incorporated into the SDS Partners specification set, this service may be used to verify compliance with the SEMI guidelines.

8.1 *Compliance Statement* — Addendum A includes a compliance statement form that should be completed by device implementors for compliance verification.

9 Specific Device Model Mappings

9.1 *Device Model for Mass Flow Controller* — This model will be specified after the MFC-specific device model (SDM) standard is complete.

9.2 *Device Model for Capacitance Manometer* — This model will be specified after the capacitance manometer SDM standard is complete.

9.3 *Device Model for Particle Counter* — This model will be specified after the particle counter SDM standdard is complete.

9.4 *Device Model for Residual Gas Analyzer* — This model will be specified after the residual gas analyzer SDM is defined.

APPENDIX 1
STATEMENT OF COMPLIANCE

NOTE: This appendix was approved as an official part of SEMI E54.5 by full letter ballot procedure.

This form is used to specify conformance options with respect to the SDS and common device model SEMI E54 specifications.

		Fill in the blank or the appropriate box. ❑ ❑			
General Device Data	Conforms to Smart Distributed System Specification:	Release _____			
	Vendor Name	_____			
	Vendor Partner ID#	_____			
	Catalog Listing	_____			
	Object Type	_____			
	Software Version	_____			
Smart Distributed System Physical Conformance Data	Network Power Consumption			_____mA @ 18 VDC	
	Connector Style	Open-Hardwired	❑	Sealed-Mini	❑
		Open-Pluggable	❑	Sealed-Micro	❑
		9 Pin	❑		
	Isolated Physical Layer	Yes		No	❑
		Opto	❑		
		Transformer	❑		
	Default Device Address	_____			
	Communication Data Rates Supported	125K bits/s	❑	500K bits/s	❑
		250k bits/s	❑	1M bits/s	❑
Smart Distributed System Logical Device & Object Data	Device Type	Master	❑		
		Slave	❑		
		Peer-to-Peer	❑		
	Number of Logical Devices	Default = 1			
	Object Class(es) on Logical Device #0	Minimum = 1			
		Object #0 _____			
		Object #1 _____			
		Object #2 _____			
		Object #3 _____			
		...			
		Object #31 _____			

If there are additional logical devices, they should be documented as above.

NOTICE: These standards do not purport to address safety issues, if any, associated with their use. It is the responsibility of the user of these standards to establish appropriate safety and health practices and determine the applicability of regulatory limitations prior to use. SEMI makes no warranties or representations as to the suitability of the standards set forth herein for any particular application. The determination of the suitability of the standard is solely the responsibility of the user. Users are cautioned to refer to manufacturer's instructions, product labels, product data sheets, and other relevant literature respecting any materials mentioned herein. These standards are subject to change without notice.

The user's attention is called to the possibility that compliance with this standard may require use of copyrighted material or of an invention covered by patent rights. By publication of this standard, SEMI takes no position respecting the validity of any patent rights or copyrights asserted in connection with any item mentioned in this standard. Users of this standard are expressly advised that determination of any such patent rights or copyrights, and the risk of infringement of such rights, are entirely their own responsibility.

SEMI E54.6-0997 (Withdrawn 0704)
STANDARD FOR SENSOR/ACTUATOR NETWORK COMMUNICATIONS FOR LONWORKS

NOTE: This document was previously designated SEMI E61. Because this document is part of a suite of documents, its designation has been reassigned for ease of reference. Please note that the technical content of this document is unchanged from the 0697 version.

NOTICE: This document was balloted and approved for withdrawal in 2004.

1 Purpose

This standard defines a communication specification based on the LonWorks technology specification to enable communications between intelligent devices on a sensor/actuator network (SAN) that operate according to SEMI-specified device models (common and device specific) in a semiconductor manufacturing tool.

This document specifies a mapping of the SEMI common device model (CDM) onto LonWorks technology using the *LonMark Interoperability Guidelines* established for LonWorks devices. The LonMark Interoperability Association may incorporate into the Interoperability Guidelines any enhancements presented in this document.

1.1 *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 LonTalk Protocol, implemented by the Neuron Chip, a physical layer transceiver, and an optional host processor. The LonTalk 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 LonTalk Protocol at layers 1-6, as well as at the application layer. Where the LonMark interoperability guidelines do not provide the functionality required by the SEMI CDM, the guidelines are extended with SEMI-specific requirements.

2 Scope

This document specifies a SAN communications standard, based on the LonWorks specification, that enables communication with SAN devices configured according to the SEMI SAN Common Device Model and appropriate Specific Device Model (SDM) specifications.

2.1 This document 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.

2.2 This document 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.

2.3 This document is used in conjunction with a SEMI standard SAN Common Device Model specification and one or more SEMI standard Specific Device Model 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.

2.4 This standard, together with a sensor/actuator network interoperability guideline, the sensor/actuator network common device model, one or more sensor/actuator network specific device model documents, the LonTalk protocol specification, and the LonMark interoperability guidelines, specifies requirements for SEMI SAN implementations based on LonWorks technology. The general sensor/actuator network document architecture is shown in the Sensor/Actuator Network Common Device Model document in Figure 1.

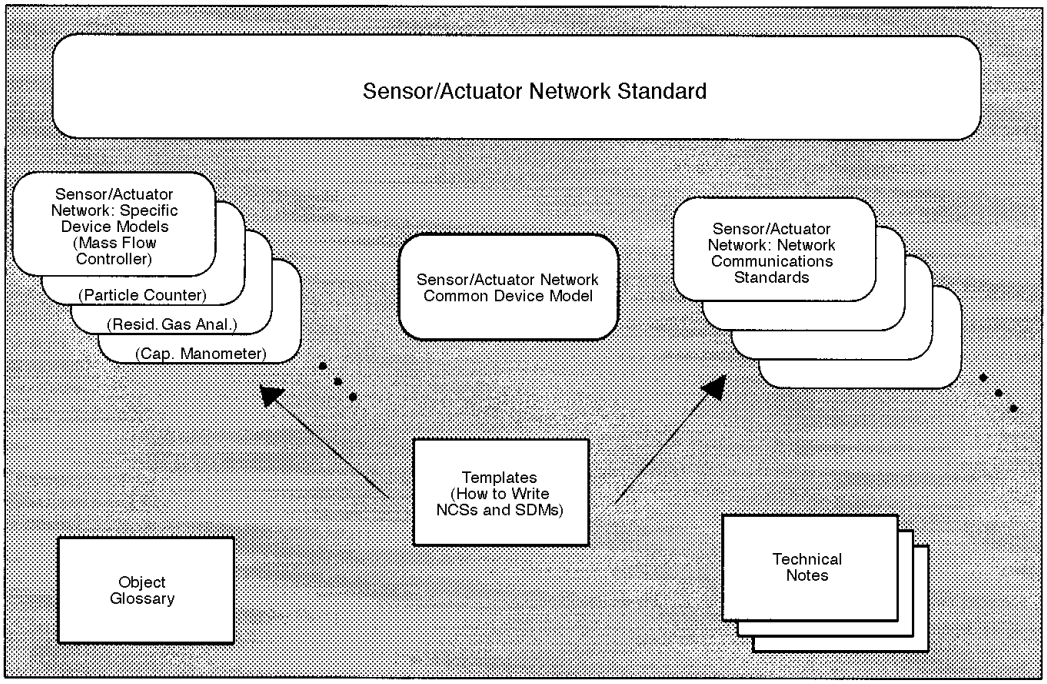

Figure 1
Sensor/Actuator Network Related Documents

2.5 *Document Structure* — The LonWorks network communications standard complies with the SEMI SAN NCS template document structure; this structure is shown in Figure 2. The standard document is composed of two main parts. The first part (Sections 1 through 8) specifies the SAN-enabling protocol as well as the presentation (i.e., mapping) of CDM object structure and behavior onto the network (referred to as the "CDM mapping"). The second part (Section 9) specifies the presentation (i.e., mapping) of SDM object structure and behavior onto the network for each SEMI-specified SDM (referred to as the "SDM mapping"). Device-type-specific items, such as connector deviations (see Section 6.1), may also be noted in Section 9.

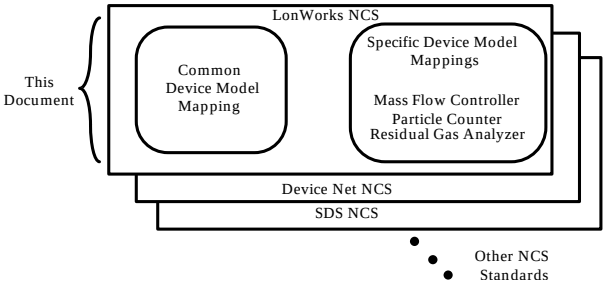

Figure 2
LonWorks Network Communications Standard Document Structure

3 Limitations

3.1 This document specifies a semiconductor equipment SAN based solely on LonWorks and is a companion document to the LonWorks 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 standard that relates to this standard is not repeated in this document. This document is limited to describing enhancements or limitations of LonWorks technology and the LonMark interoperability guidelines that are imposed by this standard.

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 guidelines required by this standard.

4 Referenced Documents

4.1 *SEMI Documents*

SEMI E30 — Generic Model for Communications and Control of SEMI Equipment (GEM)

SEMI E39 — Object Services Standard: Concepts, Behavior, and Services

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

4.2 *ISO Standard*[1]

ISO 7498 — Basic Reference Model for Open Systems Interconnection

4.3 *Other Documents*

LonTalk Protocol Specification, Echelon Corporation, 4015 Miranda Avenue, Palo Alto, CA 94304 USA

LonMark Layers 1–6 Interoperability Guidelines, Echelon Corporation

LonMark Application Layer Interoperability Guidelines, Echelon Corporation

Neuron Chip Data Book, Echelon Corporation

Neuron C Programmer's Guide, Echelon Corporation

Standard Network Variable Type Master List and Programmer's Guide, Echelon Corporation

Standard Configuration Parameter Type Master List and Programmer's Guide, Echelon Corporation

5 Terminology

Terminology that is common to all of the documents in this SAN series may also be defined in the *Sensor Actuator Network Standard*. Terminology may be reproduced here which is defined in other SEMI documents.

5.1 *Acronyms*

5.1.1 *CDM* — Common device model

5.1.2 *DM* — DeviceManager

5.1.3 *DS* — Device status

5.1.4 *NCS* — Network communications standard

5.1.5 *NV* — Network variable

5.1.6 *OSI* — Open systems interconnect

5.1.7 *OSS* — Object services standard

5.1.8 *SAC* — Sensor, actuator, controller object

5.1.9 *SAN* — Sensor/actuator network

5.1.10 *SCPT* — Standard configuration parameter type

5.1.11 *SDM* — Specific device model

5.1.12 *SNVT* — Standard network variable type

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 CDM document. Table 1 provides a mapping of fundamental terminology of the CDM document into this document and the LonWorks definitions. The symbol "=" indicates that the definition is used exactly as specified on the CDM specification.

In the following sections, additional clarification of some of these terms is provided in the context of the LonWorks protocol.

Table 1 Mapping of CDM to NCS Terminology

CDM Term	*NCS Equivalent*	*LonWorks Equivalent*
Device	=	Device or node
Device Model	=	Functional profile
Object	=	LonMark object type
Instance	=	LonMark object instance
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

5.2.1 *attribute* — Attributes are either input network variables, output network variables, or configuration properties. Input and output network variables 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 causes an event to be propagated to the device' s application layer. This corresponds to a RW (Read and Write) attribute of the object owning the network variable. Output network variables may not be updated over the network. This corresponds to a RO (Read Only) 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

1 International Organization for Standardization, 1 rue de Varembe, Case postale 56, CH-1211, Geneva 20, Switzerland

address(es) determined at installation time. Finally, configuration properties are attributes typically stored in non-volatile memory and preserved across device resets and power cycles.

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

5.2.3 *device* — A device (or node) consists of one network transceiver which implements the physical layer of the LonTalk 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 LonTalk Protocol.

5.2.4 *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; LonMark objects such as sensors, actuators, and controllers; individual network variables; and configuration properties.

5.2.5 *instance* — Real devices may have zero or more instances of each of the defined LonMark objects and functional profiles. Object instances are identified by means of an instance number within the device.

5.2.6 *object* — LonMark objects are defined as a set of one or more network variable inputs and/or outputs, implemented as Standard Network Variable Types, and a set of configuration properties, implemented as Standard Configuration Property Types. LonMark objects form the basis of interoperability at the application layer. The LonMark objects describe standard formats for how information is input to, and output from, a device, and shared with other devices on the network.

5.2.7 *service* — Request services are represented by LonTalk messages delivered to the device application. Notification services are represented by LonTalk messages originated by the device application.

5.2.8 *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).

5.3 *LonWorks-Specific Definitions* — In addition to the standard data type definitions for bit, nibble, byte, and character, the LonTalk Protocol defines a set of standard data representations for use as attribute values.

5.3.1 *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 LonTalk protocol. Table 2 shows the permitted combinations for updating and polling of network variables.

Table 2 Updating and Polling of Network Variables

Network Variable Class	*Update from Network*	*Update from Device*	*Poll from Network*	*Poll from Device*
Input	Yes	Yes	Yes	Yes[2]
Output	No	Yes[1]	Yes	No
Config'n	Yes	No	Yes	Yes[2]

NOTES:

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.

5.3.2 *configuration properties* — These are attributes of a LonMark object that are used to configure the application-specific behavior of the object, 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 non-volatile memory.

5.3.3 *functional profile* — A functional profile is a set of one or more LonMark objects, together with semantic definitions relating the behavior of the object(s) to the network variable values. The collection of functional profiles and LonMark objects in a device corresponds to the device-specific model for that device. Each type of functional profile is identified by a type number which is allocated when the profile is standardized.

5.3.4 *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 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 a 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 a GetAttribute operation.

5.3.5 *object instance* — A device's external interface documentation specifies the type identifiers of the LonMark object instances contained within the device. Each instance is allocated an index number based on the order of the declaration of the instance in the device's external interface documentation. This documentation may be uploaded from the device, and completely specifies the functional profiles and LonMark objects contained within the device, as well as the network variables and configuration properties contained within each of the functional profiles.

5.3.6 *object type* — An object type is the definition of the attributes and behaviors of an abstract entity. Each type of LonMark object is identified by a type number which is allocated when the object type is standardized. A specific device type consists of instantiations of one or more of these object types. The term *LonMark object* is loosely used to refer to either a LonMark object type or to a specific instance of a LonMark object type. The attributes of a LonMark object are implemented as a collection of network variables of SNVT types and configuration properties of SCPT types.

5.3.7 *standard configuration parameter types* — These data types, also known as SCPTs, provide a data type definition and a semantic behavior for the configuration properties of LonMark objects. A list of all available SCPTs and details of their definitions is provided in the *SCPT Master List and Programmer's Guide.*

5.3.8 *standard network variable types* — 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

The LonTalk 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 3.

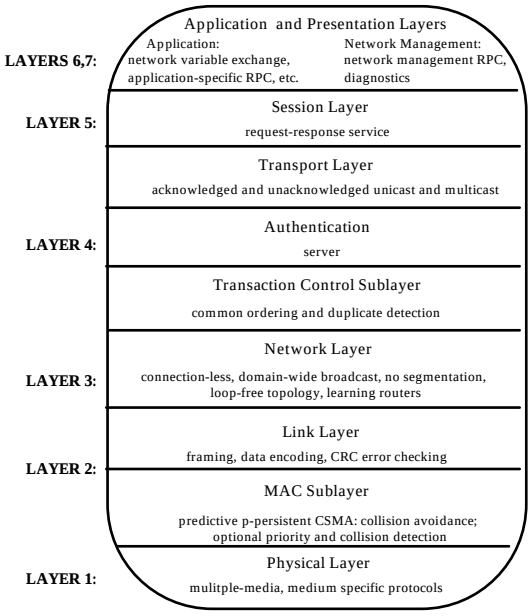

Figure 3
Layered View of the LonTalk Protocol

Note that Figure 3 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 through 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).

6.1 *Physical Layer* — The device shall employ one of the LonMark-approved physical channels as specified in the *LonMark Layers 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 Section 9 of this document. This connection is polarity-insensitive. The requirements of semiconductor equipment may be met by one of the twisted pair channel specifications listed below.

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

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

The *LonMark Layers 1-6 Interoperability Guidelines* provide specific details of the characteristics of these

transceivers. This document also provides specifications of wiring types and interconnection topologies to be used for guaranteed device interoperability. Note that the LonTalk 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.

Multiple physical layer protocols and data encoding methods are used in the LonTalk Protocol. Differential Manchester encoding is used on twisted pair physical layers.

6.2 *Link Layer* — The device shall comply with the LonTalk protocol link layer specification. This layer includes the media access control sublayer. For a number of reasons, including simplicity and compatibility with the multicast protocol, the LonTalk protocol supports a simple connectionless service. Its functions are limited to framing, frame encoding, and error detection, with no error recovery by retransmission.

6.2.1 *Media Access Control Sublayer* — In order to deal with a variety of media in the potential absence of collision detection, the MAC (Media Access Control) sublayer employs a collision avoidance algorithm called Predictive *p*-persistent CSMA (Carrier Sense, Multiple Access).

6.3 *Network Layer* — The device shall comply with the LonTalk protocol network layer specification. This layer handles packet delivery within a single domain, with no provisions for inter-domain communication. The network service is connection-less, 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.

6.4 *Transport Layer* — The device shall comply with the LonTalk protocol transport layer specification. The heart of the protocol hierarchy is the Transport and Session layers. A common Transaction Control sublayer 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 sublayer to accomplish its function. The transport and session layer messages may be authenticated using all of the LonTalk addressing modes other than broadcast. The transport layer supports end-to-end acknowledged service and an unacknowledged/ repeated service.

6.5 *Session Layer* — The device shall comply with the LonTalk 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 LonTalk 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.

6.6 *Presentation Layer* — The device shall comply with the LonTalk protocol presentation layer specification. The Presentation layer and the Application layer taken together form the foundation of interoperability for LonTalk 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.

6.7 *Application Layer* — At the application layer, interoperability between LonWorks-based devices is facilitated through the use of LonMark objects and Standard Network Variable Types (SNVTs). LonMark objects build upon network variables and provide a concise application layer interface that incorporates semantic meaning for specific device functions. LonMark objects 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, collections of objects with defined relations can be aggregated and referenced as functional profiles. The Application Layer also includes the LonTalk file transfer protocol, which provides segmentation and reassembly of arbitrary length files of data. This service may be used to get and set object