SEMI E39-0703 (Reapproved 1109)
OBJECT SERVICES STANDARD: CONCEPTS, BEHAVIOR, AND SERVICES

This Standard was technically approved by the Information & Control Global Technical Committee. This edition was approved for publication by the global Audits and Reviews Subcommittee on September 4, 2009. Available at www.semiviews.org and www.semi.org in November 2009; originally published in 1995; previously published July 2003.

SUPERSEDED

Table of Contents

1 Purpose 2
2 Scope 2
3 Referenced Standards and Documents 2
4 Terminology 2
5 Conventions 4
5.1 *OMT Object Information Model* 5
5.2 *Object Attribute Representation* 5
5.3 *Service Message Representation* 5
6 Background 6
7 Applicable Documents 6
8 Basic Concepts and Behavior 6
8.2 *Object Attributes* 7
8.3 *Object Definition* 8
9 Object Relationships 8
10 Additional Operations 11
11 Services and Scenarios 14
12 Additional Services 19
13 Applications 24
14 Requirements for Compliance 25
Appendix 1 26

List of Tables

Table 1 Top Object Attribute Definition 8
Table 2 Object Services 14
Table 3 Object Services Parameter Dictionary 15
Table 4 GetAttr Service 17
Table 5 SetAttr Service 18
Table 6 GetType Service f18
Table 7 GetAttrName Service 19
Table 8 Additional Object Services 19
Table 9 Additional Object Services Parameter Dictionary 20
Table 10 Create Service 20
Table 11 Delete Service 20
Table 12 Attach Service 21
Table 13 AttachSetAttr Service 21
Table 14 Detach Service 21
Table 15 Reattach Service 21
Table 16 AttachSupervisedObject Service 22
Table 17 DetachSupervisedObject Service 22
Table 18 ObjectAction Service 23
Table 19 ObjectActionCompletion Notification 23
Table 20 GetServiceNames Service 24
Table 21 GetServiceParameters 24

Documents Document Sharing Hub
Full document details: SEMI E39-0703-1109R

1 Purpose

1.1 The purpose of the Object Services Standard (OSS) is to provide general terminology, conventions, and notation for describing behavior and data in terms of objects and object attributes. In addition, it provides basic services for reading object attributes, setting their values, and for asking for an object's contents. This Standard is intended to be referenced by other Standards which define specific objects to reduce redundancy.

2 Scope

2.1 The scope of this Standard is to provide concepts, behavior, and services common to a variety of public objects.

2.2 Object models are common to multiple Standards. Object Services provide basic object-related definitions, and basic services for getting object attributes and setting attribute values, that can be used by all Standards defining public objects. These services allow basic management of data based on objects.

2.3 The object services defined in this document may be included in the services provided by other Standards. They may also be provided independently of such other Standards.

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

3 Referenced Standards and Documents

3.1 *ISO Standard*[1]

ISO 9595 — Common Management Information Service (CMIS)

3.2 *Other Documents*

James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen, *Object-Oriented Modeling and Design,* Englewood Cliffs, New Jersey: Prentice-Hall, 1991.

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

4 Terminology

4.1 The following basic definitions are provided in a logical order.

4.2 *Definitions*

4.2.1 *Requirements*

4.2.1.1 *fundamental compliance* — conformance to all fundamental requirements for an object or service resource.

4.2.1.2 *fundamental requirements* — the requirements for information and behavior that must be satisfied for compliance with a standard. Fundamental requirements apply to specific areas of application, objects, or services.

4.2.2 *Objects*

4.2.2.1 *object*[2] — an entity with a specific set of data and behaviors. Objects may be physical or conceptual.

4.2.2.2 *standardized object* — an object formally defined in SEMI Standards and in compliance with the fundamental requirements of SEMI E39.

4.2.2.3 *object model* — a static graphic model of objects to show structure—the identity of objects, their attributes and operations, and their relationships with one another.

4.2.2.4 *object type* — a formal classification of a group of similar objects. Synonym: *object class*. Examples: *equipment, wafer, carrier*.

[1] International Organization for Standardization, ISO Central Secretariat, BIBC II, Chemin de Blandonnet 8, CP 401, 1214 Vernier, Geneva, Switzerland; Telephone: +41.22.749.01.11, http://www.iso.org

[2] The term 'object' may refer to either an object class or an instance of a class.

SUPERSEDED

Documents
Document Sharing Hub

Full document details: SEMI E39-0703-1109R

4.2.2.5 *object instance* — an instance of an object type. An object type is like a template, while an object instance is the actual object. Example: an actual and specific optical stepper installed in a particular fab is an instance of the type 'Optical Stepper'.

4.2.2.6 *object attribute (attribute)* — information concerning an object. Examples for object type 'equipment': manufacturer, model, serial number.

4.2.2.6.1 Attributes are classified in various ways, according to their visibility (public/private), optionality, and access:

- *fundamental attribute* — an attribute that is required for fundamental compliance with a standard service.
- *optional attribute* — an attribute that is required only in support of one or more optional standard services.
- *private attribute* — an attribute that is used strictly for internal purposes and is unknown (invisible) through public services.
- *public attribute* — an attribute that is known (visible) and whose current value is provided as a service to other entities upon request.
- *read-only attribute (RO)* — may not be changed through public services.
- *read/write attribute (RW)* — may be changed through public services.

4.2.2.7 *attribute name* — the formal name of the attribute that is used to identify it. The names (and data types) of public attributes are included with the object's definition and are unique for that object.

4.2.2.8 *object identifier* — a set of one or more items of information, concerning a particular instance (instantiation) of an object of a given type, that together uniquely distinguish that instance from all other instances of that object within a defined scope. Note: An object may have more than one identifier. An identifier may be *simple* (consist of only one attribute) or *complex* (consist of more than one attribute). Example: The combination of equipment's manufacturer, model, and serial number serve as an identifier that uniquely identifies a specific installation.

4.2.2.9 *object handle* — a numeric or binary identifier assigned by an application for internal use. Note: A handle may also be available as a *public attribute* but cannot generally be guaranteed to be either persistent or unique outside of the current relationship between service user and service provider. The persistence of the handle is specified by the standard that defines the object.

4.2.2.10 *operation* — a function performed by, or inherent to, an object. Example: for equipment, 'run', 'stop', 'abort'.

4.2.2.11 *aggregation object* — an object that is composed (made up) of other objects. An aggregation may lose some degree of integrity if one of its components is missing.

4.2.2.12 *component object* — an object that is part of an aggregation.

4.2.2.13 *container object* — an object that is intended to hold other types of objects. The contents may or may not be ordered.

4.2.2.14 *contents* — an object that is in a container. Examples: a wafer in a cassette, a book in a library.

4.2.2.15 *owner object* — an object that is an aggregation, container, or supervisor of another object. The owner object is said to *own* the other object.

4.2.2.16 *owned object* — an object that is a component of, contained in, or supervised by, another object. The owned object is said to be *owned by* the other object.

4.2.2.16.1 An object may have multiple owners. For example, it may be a shared resource.

4.2.3 *Services*

4.2.3.1 *scope* — the specification of one or more objects that starts with a specific owner object and proceeds downward through a hierarchical sequence of 'owns' relationships.

4.2.3.2 *service (or message service)* — represents a function offered to a user by a provider. A service consists of a sequence of service primitives, each described by a list of parameters. A service excludes definition of message structure and protocol.

SUPERSEDED

Documents Document Sharing Hub
Full document details: SEMI E39-0703-1109R

4.2.3.2.1 A service may or may not be processed by the provider of the service. The invocation of some service may have to interact with other objects to fulfill its requirement completely; an exception will be raised if the service cannot fulfill all its pre or post conditions. The object that provides the service may reject the service request or restrict the completion of the service to prevent an illegal action due to the equipment condition.

4.2.3.3 *notification service* — initiated by the service provider and sent to the service consumer/subscriber. No response is expected.[3]

4.2.3.3.1 Notifications consist of two service primitives — a message from the sender to the communications facility and an indication to the receiver from the communications facility.

4.2.3.4 *request service* — initiated by the service consumer. Requests ask for data or for an activity (operation) from the provider. Requests expect a specific response message.

4.2.3.4.1 A request consists of a message that requires a response from the receiver. The primitives for a request are the same as those of the notification, while the response defines additional primitives called the *response* and the *confirmation*.

4.2.3.5 *service resource* — a logical group of one or more services within a specific area of functionality.

4.2.4 *Data Type*

4.2.4.1 *form* — type of data: positive integer, unsigned integer, integer, enumerated, Boolean, text, formatted text, structure, list, ordered list.

4.2.4.2 *positive integer* — may take the value of any positive whole number. Messaging protocol may impose a limit on the range of possible values.

4.2.4.3 *unsigned integer* — may take the value of any positive integer or zero. Messaging protocol may impose a limit on the range of possible values.

4.2.4.4 *integer* — may take on the value of any negative or unsigned integer. Messaging protocol may impose a limit on the range of possible values.

4.2.4.5 *floating point* — may take on any single (real) numeric value, positive or negative. Messaging protocol may impose a limit on the range of possible values.

4.2.4.6 *enumerated* — may take on one of a limited set of possible values. These values may be given logical names, but they may be represented by any single-item data type.

4.2.4.7 *Boolean* — may take on one of two possible values, equating to TRUE or FALSE.

4.2.4.8 *text* — A text string. Messaging protocol may impose restrictions, such as length or ASCII representation.

4.2.4.9 *formatted text* — a text string with an imposed format. This could be by position, by use of special characters, or both.

4.2.4.10 *structure* — a complex structure consisting of a specific set of items, of possibly mixed data types, in a specified arrangement.

4.2.4.11 *list* — a set of one or more items that are all of the same form (one of the above forms).

4.2.4.12 *ordered list* — a list for which the order in which items appear is significant.

5 Conventions

- Defined terms are presented in boldface when introduced for the first time.
- Formally reserved text strings, such as attribute names, are underlined.

[3] This does not preclude the implementation of a response message in the message protocol.

SUPERSEDED
Documents
Document Sharing Hub
Full document details: SEMI E39-0703-1109R

5.1 *OMT Object Information Model* — The object models are presented using the Object Modeling Technique (OMT) developed by Rumbaugh, James, *et al.*, in *Object-Oriented Modeling and Design,* Prentice Hall, Englewood Cliffs, NJ, c1991.[4]

5.1.1 Overviews of this notation are provided in the appendix. A brief discussion of terminology is also provided in the Appendix.

5.2 *Object Attribute Representation* — The object information models for standardized objects will be supported by an attribute definition table with the following column headings:

Attribute Name	*Definition*	*Access*	*Rqmt*	*Form*
The formal text name of the attribute	Description of the information contained	RO or RW	Y or N	See § 4.2.4.

5.2.1 The Access column uses RO (Read Only) or RW (Read and Write) to indicate the access that users of the service have to the attribute.

5.2.2 A 'Y' or 'N' in the Requirement (Reqt) column indicates if this attribute must be supported in order to meet fundamental compliance for the service.

5.2.3 The Form column is used to indicate the format of the attribute. (See § 4.2.4 for definitions.)

5.3 *Service Message Representation*

5.3.1 *Service Resource Definition* — A service resource definition table defines the specific set of messages for a given service group, as shown in the following table:

Message Service Name	*Type*	*Description*
Message name	N or R	The intent of the service.

5.3.1.1 Type can be either N = Notification or R = Request.

5.3.1.2 Notification type messages are initiated by the service provider, and the provider does not expect to get a response from the consumer/subscriber.

5.3.1.3 Request messages are initiated by a service consumer or subscriber. Request messages ask for data or an activity from the provider. Request messages expect a specific response message (no presumption on the message content).

5.3.2 *Service Parameter Dictionary* — A service parameter dictionary table defines the parameters for one or more services, as shown in the following table:

Parameter	*Form*	*Description*
Parameter X	Data type	A parameter called X is B in A.

5.3.2.1 A row is provided in the table for each parameter of the service. The first column contains the name of the parameter. This is followed by columns describing the form and contents of the corresponding primitive.

5.3.2.2 The Form column is used to indicate the type of data contained in a parameter. (See § 4.2.4 for definitions.)

5.3.2.3 The Description column in the Service Parameter Dictionary table describes the meaning of the parameter, the values it can assume, and any interrelationships with other parameters.

[4] For a full description of the Object Modeling Technique, see Object-Oriented Modeling and Design, Rumbaugh, Blaha, Premerlani, Eddy and Lorenson, Prentice Hall, 1991.

SUPERSEDED
Documents Document Sharing Hub
Full document details: SEMI E39-0703-1109R

5.3.2.4 To prevent the definition of numerous parameters named 'XxxList,' this Document adopts the convention of referring to the list as '(List of) Xxx.' In this case, the definition of the variable Xxx will be given, not of the list. The term 'list' indicates a collection (or set) of zero or more items of the same data type. Where a list is used in both the request and the response, the list order in the request is retained in the response. A list must contain at least one element unless zero elements are specifically allowed.

5.3.3 *Service Message Definition* — A service message definition table defines the parameters used in a service, as shown in the following table:

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
Parameter X	(see below)	(see below)	A description of the service.

5.3.3.1 The columns labeled Req/Ind and Rsp/Conf link the parameters to the direction of the message. The message sent by the initiator is called the 'Request'. The receiver terms this message the 'Indication' or the request. The receiver may then send a 'Response', which the original sender terms the 'Confirmation'.

5.3.3.2 The following codes appear in the Req/Ind and Rsp/Conf columns and are used in the definition of the parameters (e.g., how each parameter is used in each direction):

- 'M' — Mandatory Parameter: must be given a valid value.
- 'C' — Conditional Parameter: may be defined in some circumstances and undefined in others. Whether a value is given may be completely optional or may depend on the values of other parameters.
- 'U' — User-Defined Parameter
- '-' — The parameter is not used.
- '=' — (for response only) Indicates that the value of this parameter in the response must match that in the primary (if defined).

6 Background

6.1 During the development of proposals for several other service standards, it was discovered that they shared a common need for terminology and services related to objects. Originally each proposal included a service-specific message for getting the values of an object's attributes, and some also provided a service-specific message for setting these values.

6.1.1 The decision was made to eliminate this redundancy through provision of a standard to provide those definitions and services used by the other separate service standards.

7 Applicable Documents

SEMI Standards Factory Automation Software Volume[5]

ISO/TR 8509:1987 — Information Processing Systems, Open Systems Interconnection – Service Conventions[6]

William Stallings, Networking Standards, "*A Guide to OSI, ISDN, LAN, and MAN Standards*," Addison-Wesley Publishing Company

8 Basic Concepts and Behavior

8.1 This section defines the concepts and behavior that are common to public objects.

[5] These Documents may be obtained from SEMI, http://www.semi.org
[6] International Organization for Standardization, ISO Central Secretariat, BIBC II, Chemin de Blandonnet 8, CP 401, 1214 Vernier, Geneva, Switzerland; Telephone: +41.22.749.01.11, http://www.iso.org

SUPERSEDED
Documents
Document Sharing Hub
Full document details: SEMI E39-0703-1109R

8.1.1 Object-oriented analysis is a widely accepted tool, both for applications that are implemented using object-oriented technologies and those using traditional technologies. Object information models provide a powerful method for describing relationships that can be intuitively understood.

8.1.2 For those unfamiliar with object-oriented terminology, a brief discussion of terms is provided in the Appendix.

8.2 *Object Attributes* — An *object attribute* is a data value that is held by all instances of a given object type. For communication purposes, public attributes are assigned a logical *name* that is unique for that object. An object's attributes may be individually referenced either by an enumerated numeric value or by a reserved *attribute name* that conforms to the use of text defined in ¶ 8.1.1.

8.2.1 An attribute that can be requested or referenced through formal public *services* is a *public attribute*. Other attributes may exist in specific implementations but are invisible through public services and are called *private attributes*.

8.2.2 Specific attributes may be designated as *fundamental* for a specific object. These shall be supported by all implementations that use or reference the specified object. Attributes that are not fundamental are *optional*.

8.2.3 *Use of Text* — Values defined as text, when given in ASCII, are subject to certain restrictions. This includes object attribute values, attribute names, and service parameters.

8.2.3.1 Text in ASCII is restricted to the characters between 20_{16} and $7E_{16}$, excluding the question mark '?', the asterisk '*', and the tilde '~'.

8.2.3.2 The question mark and asterisk are reserved for use as 'wild characters' in filters and searches, while the tilde is reserved to allow systems that cannot use spaces to convert spaces to tildes for internal use.

8.2.3.3 Text used in specific contexts may have additional restrictions.

8.2.3.4 Unless otherwise stated, case is not significant for purposes of comparison. However, case is used to improve readability and should be preserved whenever possible.

8.2.4 *Object Type* — An object always knows its type. For this reason, the attribute for object type, ObjType, is required for all public objects. ObjType is a text string containing the formal classification of an object. The text string shall conform to the convention defined in ¶ 8.1.1, with the additional restriction that the greater-than symbol '>' and the colon ':' are also excluded. The text string shall not start or end with a space.

8.2.4.1 Object types (the values for ObjType) for standardized objects are reserved. Types reserved are specific to individual standards.

8.2.5 *Object Identifier* — Every instance of an object shall have one or more attributes that together uniquely distinguish that instance from all other instances of objects of the same object type. An object may have more than one identifier. The identifier(s) are defined as part of the object definition.

8.2.5.1 ObjID is a text-based fundamental attribute of all public objects and provides an attribute of a known form to serve as an identifier for an object of any type. The text string shall conform with the convention defined in ¶ 8.1.1 and with the additional restriction that the greater-than symbol '>' and the colon ':' are also excluded. The text string shall not start or end with a space.

8.2.5.2 From the point of view of objects such as managers, aggregates, and containers that have responsibilities for other objects (see ¶¶ 9.1 and 9.2) the combination of ObjType and ObjID for these other objects shall be unique. For example, if several process chambers have a single vacuum pump, each process chamber may have direct access to at most one ObjType:ObjID> combination of 'Pump:Vacuum.' However, the combination is not required to be unique across the different chambers. The host differentiates between the several vacuum pumps by their belonging to different aggregations.

8.2.5.3 The object definition for each object shall specify the particular identifier used for general access. For objects that normally use an identifier with a numeric value as an identifier (such as 'handle,' described below), the value of ObjID is a numeric text string set to the value of that attribute.

8.2.5.4 An object may also have other identifiers in addition to ObjID.

SUPERSEDED

Documents Document Sharing Hub
Full document details: SEMI E39-0703-1109R

8.2.6 *Object Handle* — An object's *handle* is an attribute with a numeric value that is assigned by the application that created the object. The handle may be used for the object's identifier. It is generally intended for local use and may or may not be defined as a public attribute in the object's formal definition.

8.2.6.1 Where used, the handle may be guaranteed to be unique only within a specific context and within a single association between service-provider and service-user for a specific service resource. The handle may or may not be persistent beyond a certain context within that association or beyond it. Persistence of the handle is specified as part of the definition of the object and is beyond the scope of OSS.

8.3 *Object Definition* — An object definition includes an attribute definition table described in § 5.3.

8.3.1 The object that is the super-type of all public objects is called the *top object*. The attributes and operations of the top object, shown in Figure 1, are fundamental requirements for all public objects. That is, all objects shall respond to the object attributes ObjType and text strings, and all public objects shall recognize and respond to the operations *get attributes* and *set attributes*.

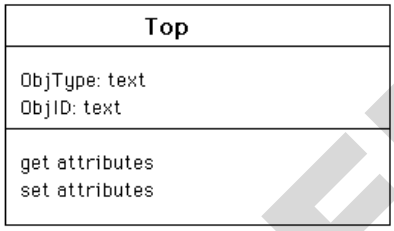

Figure 1
Top Object

8.3.2 Formal definition of an object shall include a table defining the object's public attributes. The Object Attribute Definition table for the top object is given in Table 1 below. It contains the two fundamental attributes required of all public objects.

Table 1 Top Object Attribute Definition

Name	*Definition*	*Access*	*Req*	*Form*
ObjType	The object type.	RO	Y	Text
ObjID	The object's identifier.	RO	Y	Text

8.3.3 Access refers to the ability to read and write the value of the attribute through OSS services. An object's type and identifier may not be changed through OSS services. However, it may be possible for the service user to assign a value for the object's identifier at the time the object (instance) is created. This depends on the object and the services provided.

9 Object Relationships

9.1 This section addresses relationships between objects that affect communications, either by restricting communications or through information concerning relationships.

9.2 *Object Hierarchy* — Certain objects may be specified as *aggregation objects* or as *container objects* when they are defined.

9.2.1 An aggregation object is composed of other objects called the *components* of the aggregation. This is illustrated in Figure 2. Components may be of one or more different types, or they may be of the same type.

SUPERSEDED

Documents
Document Sharing Hub
Full document details: SEMI E39-0703-1109R

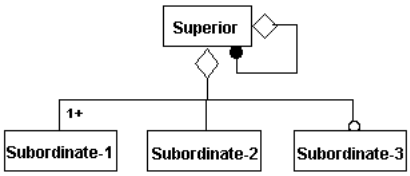

Figure 2
Aggregation with Two Component Types

9.2.2 Container objects *contain* other objects, of the same or different types, but are not made up of them.

9.2.3 Figure 3 shows two object types that are associated with a relationship called 'contains.' The solid circle indicates that Type-1 may contain zero or more objects of Type-2 type. An example of a common container type is a file directory.

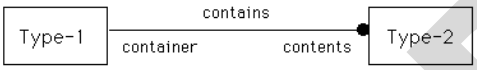

Figure 3
Container and Contents

9.2.4 An aggregation loses some degree of integrity if certain of its components are missing, while a container retains full integrity even if it has no contents. An automobile is an aggregation of many components, some of which are themselves aggregations. An automobile may also have contents (driver, passengers, belongings) which are removable and not considered as components. Other examples of container objects include lists, dictionaries, and libraries. A container that has no contents is said to be *empty*.

9.2.5 The aggregation or container is called the *superior object* and its components or contents are called *subordinate objects*.

9.2.6 In addition to the relationships of 'is composed of' (roles: aggregate/component) and 'contains' (roles: container/contents), the other hierarchical relationship that occurs naturally in factories and in control systems is that of '*supervises*' (roles: supervisor/supervised). This denotes a control relationship where the supervised object accepts part or all of its directions from the supervisor. The supervisor, in turn, has responsibilities that it delegates to the supervised. The supervisor typically will also have a relationship of '*is composed of*' or '*contains*' with its supervised subordinates.

9.3 *Scope and Ownership* — The hierarchical relationships of aggregations, containers, and supervisors may at times be required for pointing to a specific object. These relationships may be shown with a multi-level tree structure, as shown in Figure 4. Subordinate objects at one level may be the superior objects at the next lower level. The superior object at the highest level is called the *root* of the tree.

NOTE 1: Figure 4 is not drawn in OMT notation.

9.3.1 An *owner object* is an object that is an aggregate, container, or supervisor. An *owned object* is an object that is a component of, contained in, or supervised by, an owner object.

9.3.2 *Scope*[7] is the concept and method of pointing to a specific owned object through the use of sequence of hierarchical relationships. Scope allows any particular object (type or instance) within such a hierarchical tree to be fully specified by providing a unique path down this tree using a concatenation of object types and identifiers.

[7] Scoping is used in material ISO 9595 (CMIS).

SUPERSEDED
Documents Document Sharing Hub
Full document details: SEMI E39-0703-1109R

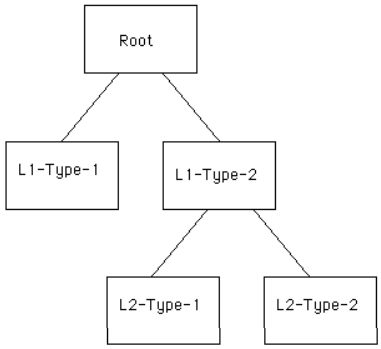

Figure 4
Example of Owner Hierarchy

9.3.3 This path is used to create an *object specifier* that is represented as a sequence of concatenated text strings of the form 'type:id>', which may be repeated as necessary to form the complete path. The character '>' is used to terminate the object type field, and the colon character ':' is used to terminate the identifier field. The object specifier uniquely identifies an object within the entire domain of objects and is able to extend the domain within which a search for an object would otherwise occur.

9.3.4 Formally, an object specifier is a formatted text string of the form:

$$\text{'type}_1\text{:id}_1\text{>...type}_n\text{:id}_n\text{>'}$$

where 'type_i' and 'id_i' represent the object type and object identifier, respectively, of the ith object instance in the sequence, and where the ith object is owned by the (i−1)th object and is the owner of the (i+1)th object.

9.3.5 Figure 5 shows an example of a typical set of hierarchical relationships in the factory. An application of scope might be through Cell AB, Cluster BB, Process Module (PM) CB, to Device DB. In this example, the object specifier for Device DB would be:

'FactoryHost:Hilda>Cell:AB>Cluster:BB>PM:CB> Device:DB>'

9.3.6 It is generally not necessary to start with the root object. It is sufficient to start with an object that is owned by the communications partner. However, it is invalid to omit a level in the hierarchy that is between the starting object and the final object in the path.

9.3.7 Some systems may be able to guarantee the uniqueness of object identifiers for their owned objects. For example, equipment may be able to guarantee that all of its owned objects have unique identifiers regardless of type. In this case, an object's identifier is sufficient to point to a specific object instance. Object type may be omitted without ambiguity, as it may be obtained as an attribute of the specified object. The form of the path then becomes '$\text{id}_1\text{>id}_2\text{>.id}_n\text{>}$'. In the previous example, this might be 'AB>BB>CB>DB>'.

9.3.8 Object type may also be omitted in applications where it may be inferred without ambiguity within a particular context. A service that is dedicated to a specific object type X, for example, that is always owned only by another specific object type Y, may specify usage for object type X. Since object type may then be inferred from the context, object types X and Y need not be included in a specifier for an object of type X.

9.3.9 The terminator of the final identifier is optional. This allows a single object identifier to be used as a simple object specifier. As a result, however, it may be necessary to provide string delimiters to prevent premature termination of string parsing in the event an identifier contains one or more embedded spaces.

9.3.10 Scope can be applied to any set of hierarchical relationships, given a minimum set of requirements for public objects:

- an object always knows its own type (ObjType),
- an object always knows its own identifier(s) (including ObjID),

SUPERSEDED
Documents
Document Sharing Hub
Full document details: SEMI E39-0703-1109R