Data collection is then disabled.

```
CC                                        PM2
>>-------------------------------------->>
Set.Req(EREvent,EventID=Event1,
Enabled=FALSE)
CC                                        PM2
<<--------------------------------------<<
Set.Rsp(EREvent,EventID=Event1,Status=OK)
```

15. PM2 to CM TRANSFER JOB SETUP

The CC informs the TM that a wafer transfer is required from the PM2 to CM.

```
CC                                         TM
>>-------------------------------------->>
TRJobCreate.Req(MID="Wafer1",
TRSourceAmID="PM2",
TRDestinationAmID="CM")
```

The transport module assigns an identifier for this particular transfer job. The transfer job is then started by the transport module.

```
CC                                         TM
<<--------------------------------------<<
TRJobCreate.Rsp(JobID=TMJob3,Status= OK)
CC                                         TM
<<--------------------------------------<<
TRJobStarted.Req(JobID=TMJob3, TimeStamp)
```

16. CC TO PM2 SEND WAFER

The CC informs the PM2 that a wafer is to be transferred to the TM.

```
CC                                        PM2
>>-------------------------------------->>
TRJobCreate.Req(MID="Wafer1",Port=
"Port1", Dir=Send))
CC                                        PM2
<<--------------------------------------<<
TRJobCreate.Rsp(JobID=IOJob6,Status=OK)
CC                                        PM2
<<--------------------------------------<<
TRJobStarted.Req(JobID=IOJob6, TimeStamp)
```

When the process module is ready, it directly communicates with the transport module to indicate its readiness for wafer handoff.

```
TM                                        PM2
<<--------------------------------------<<
HOReady.Req(PortID="Port1",AmID="PM2",
HODir=GET,MID="Wafer1")
```

17. CC TO CM RECEIVE WAFER

The CC informs the CM that a wafer is to be transfered from the TM.

```
CC                                         CM
>>-------------------------------------->>
TRJobCreate.Req(MID="Wafer1",Port=
"Port1", Dir=Receive)
CC                                        PM2
<<--------------------------------------<<
TRJobCreate.Rsp(JobID=IOJob7,Status=OK)
CC                                        PM2
<<--------------------------------------<<
TRJobStarted.Req(JobID=IOJob7, TimeStamp)
```

When the cassette module is ready, it directly communicates with the transport module to indicate its readiness for wafer handoff.

```
TM                                         CM
<<--------------------------------------<<
HOReady.Req(PortID="Port1",AmID="CM",
HODir=PUT,MID="Wafer1")
```

18. PM2 TO TM WAFER HANDOFF

The PM2 transfers the wafer to the TM through the handoff sequence, and the TM will keep the cluster controller informed of its transfer progress. The transport module informs the process module that it is proceeding with the “getting” of the wafer.

```
PM2                                        TM
<<--------------------------------------<<
HOReady.Req(Port=Port1,AmID="PM2",HODir=G
ET,MID="Wafer1")
```

The process module informs the TM to use the EXTEND scenario as the put operation.

```
PM2                                        TM
>>-------------------------------------->>
HOExtend.Req(Port=Port1,AmID="PM2")
PM2                                        TM
<<--------------------------------------<<
HOExtend.Rsp(Port=Port1,AmID="PM2",
Status=OK)
```

When the putting of the wafer is now successful, the PM commands the transport module to Retract.

```
PM2                                    TM
>>-------------------------------------->>
HORetract.Req(Port=Port1,AmID="PM2")
PM2                                    TM
<<--------------------------------------<<
HORetract.Rsp(Port=Port1,AmID="PM2",
Status=OK)
```

Upon successful completion of the handoff, a verify sequence is used to complete the transaction.

```
PM2                                    TM
>>-------------------------------------->>
HOVerify.Req(Port=Port1,AmID="PM2")
PM2                                    TM
<<--------------------------------------<<
HOVerify.Rsp(Port=Port1,AmID="PM2",
Status=OK)
CC                                    PM2
<<--------------------------------------<<
TRJobComplete.Req(JobID=IOJob6,
TimeStamp)
```

The cluster controller is informed that the processing job in PM2 is complete.

```
CC                                    PM2
<<--------------------------------------<<
PRJobComplete.Req(JobID=PM2Job1,
TimeStamp,Status=OK)
```

19. TM TO CM WAFER HANDOFF

The TM now has possession of the wafer and is ready to proceed with the putting of the wafer into the cassette module. First, the TM informs the cluster controller that it is proceeding with the transfer of the wafer to CM through the handoff sequence.

```
CM                                     TM
<<--------------------------------------<<
HOReady.Req(Port=Port1,AmID="CM",
HODir=PUT,MID="Wafer1")
```

The cassette module informs the transport module to use the PLACE sequence to put the wafer.

```
CM                                     TM
>>-------------------------------------->>
HOPlace.Req(Port=Port1,AmID="CM")
CM                                     TM
<<--------------------------------------<<
HOPlace.Rsp(Port=Port1,AmID="CM",
Status=OK)
```

When the putting of the wafer has been successful, the cassette module informs the transport module with a verify sequence.

```
CM                                     TM
>>-------------------------------------->>
HOVerify.Req(Port=Port1,AmID="CM")
CM                                     TM
<<--------------------------------------<<
HOVerify.Rsp(Port=Port1,AmID="CM",Status=
OK)
```

The cassette module then informs the cluster controller of its current status.

```
CC                                     CM
<<--------------------------------------<<
TRJobComplete.Req(JobID=IOJob7,
TimeStamp,Status=OK)
```

The TM then informs the cluster controller that the putting of material is finished and that the requested transfer job is complete.

```
CC                                     TM
<<--------------------------------------<<
TRJobComplete.Req(JobID=TMJob3,
TimeStamp,Status=OK)
```

20. CM TO FACTORY CASSETTE HANDOFF

The cassette is passed from the cluster tool to the factory environment.

```
CC                                     CM
>>-------------------------------------->>
TRJobCreate.Req(MID="Carrier1",
Port=Port2,Dir=Send)
CC                                     CM
<<--------------------------------------<<
TRJobCreate.Rsp(JobID=IOJob8,Status=OK)
CC                                     CM
<<--------------------------------------<<
TRJobStarted.Req(JobID=IOJob8, TimeStamp)
```

At this point, it is assumed that the cassette module performs actions such as loadlock venting, door opening, and initiating communication with the user (human or host) to perform the handoff to the factory. The cluster controller is informed of progress. When the transfer to the factory is complete, the cluster controller is informed.

```
CC                                        CM
<<--------------------------------------<<
TRJobComplete.Req(JobID=IOJob8,
TimeStamp,Status=OK)
```

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 E38.1-95
COMMUNICATIONS ENVIRONMENT HSMS/SECS-II FOR CLUSTER TOOL MODULE COMMUNICATIONS

1 Purpose

The purpose of this standard is to specify the communication environment for communication with and among modules in a cluster tool.

2 Scope

The scope of this standard is communication within a cluster tool as defined in the standard Cluster Tool Module Communication Standard (CTMC, SEMI E38). This standard specifies the transmission of SECS-II messages by HSMS (SEMI E37), HSMS-GS (SEMI E37.2), and Ethernet. Other methods of message communication are possible but beyond the scope of this standard.

This standard is intended to be one of possibly several supplements to the CTMC SEMI E38.

3 Referenced Documents

3.1 *SEMI Standards*[1]

The following SEMI standards are related to the Communications Environment standard:

SEMI E5 — SEMI Equipment Communication Standard 2 - Message Content (SECS -II).

SEMI E37 — High-Speed SECS Message Services (HSMS) Generic Services

SEMI E37.2 — High-Speed SECS Message Services General Session (HSMS-GS)

3.2 *IEEE Standards*[2]

The following IEEE standard is related to the Communications Environment standard:

IEEE 802.3 — Carrier Sense Multiple Access with Collision Detection

4 Definitions

The following definitions are arranged in alphabetical order. Some are defined using terms defined elsewhere within this section. No definitions beyond this section and the referenced standards should be necessary for a basic understanding of these terms.

10 BASE 2 — a common physical medium for the transmission of Ethernet signals.

AUI (Attachment Unit Interface) — a specification of the connector, pins, and signals used to interface a computer to a MAU. Defined formally in IEEE 802.3.

Cluster Tool Environment — a collection of interconnected modules forming a cluster tool.

Communications Stack — a specification of successive layers of interfaces through which communication services are provided.

Ethernet — a method of transmitting signals. Ethernet is defined formally by IEEE 802.3

MAU (Medium Attachment Unit) — an adaptor from the AUI interface to a physical wire or medium such as 10 BASE 2. Defined formally in IEEE 802.3

Platform — a physical computer in a cluster tool that contains the Ethernet Interface or station as defined in IEEE 802.3. In a cluster tool, a platform may have one or more logical modules residing on it.

5 Required Sessions

In order to support the Services specified in the CTMC standard, there is an associated set of Protocol standards that specify a set of SECS-II messages to support these Services. This standard specifies the transmission of these SECS-II messages using HSMS sessions.

In order to communicate with a given logical module, a separate HSMS connection (TCP/IP connection) is required per logical module.

For correct operation of a cluster tool using these facilities, a set of standard sessions, given in Table 1, is required. Other sessions are possible, but are not required. In order to be compliant a module must not require further sessions in order to operate.

Table 1 Required Sessions

Client	*Server*	*Services*
Cluster Controller	Transport Module	Object Services, Material Movement, Exception Management, Event Reporting
Cluster Controller	Cassette Module	Object Services, Material Movement, Exception Management, Event Reporting

[1] Semiconductor Equipment & Materials International, 805 East Middlefield Road, Mountain View, CA 94043

[2] Institute of Electrical and Electronics Engineers, 345 East 47th Street, New York, NY 10017

Client	*Server*	*Services*
Cluster Controller	Process Module	Object Services, Processing Management, Recipe Management, Material Movement, Exception Management, Event Reporting
Cassette Module	Transport Module	Object Services, Material Movement
Process Module	Transport Module	Object Services, Material Movement

6 Communication Stack Specification

There are many layers of software required in order to communicate using a facility such as HSMS. In order to interoperate, modules shall have identical or equivalent protocols at each layer on both sides of a physical connection.

HSMS does not specify a particular physical layer. In order to ensure interoperability between modules, of possibly different origin, a complete interoperable communication stack is required.

6.1 *SECS-II Messages* — The Cluster Tool Module Communications (CTMC) standard specifies a set of messaging services from applicable standards for use by a cluster tool module. Each service standard has a supplemental protocol standard which defines the SECS-II message support. This standard specifies a method for transporting these SECS-II messages. In order to be compliant with this standard, these SECS-II messages must be used. The detail of the SECS-II messages is found in SEMI E5. SECS-II message definitions refer to messages communicating between a host and equipment. With reference to implementing CTMC, the host is the service-user or cluster-controlling entity, and the equipment is the service-provider or cluster module entity.

6.2 *HSMS Messaging* — For the purposes of communication of SECS-II messages within the cluster tool communication environment, SEMI High-Speed SECS Message Service (HSMS) shall be used (SEMI E37).

CTMC defines a number of services. For inter-module, and service user-to-module communication, many services will be required simultaneously. As a result, the capabilities of the HSMS General Session (SEMI E37.2) are required and shall be used for communication of all services specified by the CTMC.

6.2.1 *IP Addresses, Port Numbers, and Session IDs* — HSMS defines the setting of IP Addresses, Port Numbers, and Session IDs. In order to uniquely specify a service, all three are required as configuration parameters. The actual numbers shall be within the allowed range and unique within their respective domains. Table 2 shows these parameters, what they define, and their respective domains.

Table 2 Configuration Parameters

Parameter	*Defines*	*Domain of Uniqueness*
IP Address	Platform	Network
Port Number	TCP/IP Connection	Platform
Session ID	CTMC Service	TCP/IP Connection

6.2.2 *Session ID's* — Each association required by CTMC (such as Processing, Material Movement, Exception, etc.) shall have a session within a connection. These Session ID's shall be documented by the service provider, and, if configurable, the method of configuration shall be documented.

The Session ID for a given service shall be defined and may be dictated by the service provider (at the server end of a connection).

In order to use a service, the service user (which is the client end of the connection) shall select the service by Session ID. Thus, the Session ID shall be configurable for the service user and the method of configuring shall be documented for each service user. Session IDs shall be as shown in Table 3.

Object services shall use Session ID 1; all other services shall use Session ID's from the range labeled "Available" in Table 3.

Table 3 Allowed Session ID's

Session ID	*Service*
0	Reserved
1	Object Services
2-63	Reserved (future)
64-65534	Available
65535 (OxFFFF)	Reserved (HSMS)

6.2.3 *Linktest* — The linktest procedure is an optional facility of HSMS. For the purposes of cluster tool communications, the linktest procedure as described in the HSMS standard shall be supported. On detection of a linktest timeout, a communication failure will be considered to have occurred, and higher communication layers shall be notified of this failure.

6.2.4 *Connection Mode* — HSMS provides for Active, Passive, and Alternating Connection Modes. For the purposes of establishing connections in a cluster tool, either Active or Passive shall be used. For each service provided, there is a service provider (server) and a service user (client). With respect to each service, the

client shall connect in the Active mode and the server in the Passive mode. Table 1 specifies the client and server (and hence connection mode) of the required sessions.

In addition to establishing the connection, the client shall initiate the sessions with the Select message, as defined in HSMS. As a result, all sessions at the client end of a connection shall be service users and all sessions at the server end of the connection shall be service providers.

6.3 *Use of TCP/IP* — HSMS defines the use of TCP/IP for communication of messages. Other simultaneous uses of TCP/IP are possible but beyond the scope of this standard.

Within TCP, Port Numbers are used to uniquely identify the connection on a given platform. A port number may be any value between 0 and 65535, but TCP reserves port numbers starting at 0. Refer to HSMS and the references given in HSMS for the definition of the reserved port numbers.

Multiple logical modules may exist on a given platform. Each service for a logical module on a given platform must use a unique session within a connection. It is possible to configure a module so that each logical module uses a separate connection, but it is also possible for several logical modules to share a connection. Each service requires a unique connection for each external entity, but an external entity may, depending on the configuration, access the services of several logical modules on a platform, identifying them by unique session, except for Object Services. Object Services is always identified by Session ID 1.

Connection to a specific module shall be established using a non-reserved port number which is unique among modules on that platform. A given service may have several simultaneous connections.

6.3.1 *Setting Port Number* — The port numbers used for establishing HSMS connections shall be configurable for both the service user and service provider. This is because the port numbers may be used by some other software entity on the same platform. Port numbers shall be chosen that are not in the reserved range and do not conflict with any other software entity on the platform. The method of configuration of the port numbers shall be documented.

6.4 *Physical Layer* — HSMS allows for many different physical layers. In order for modules to interoperate, a single physical layer is required. This standard specifies Ethernet — as defined in the standard IEEE 802.3 — to be used for physical layer transmission of HSMS messages.

Other physical layers could be used, but the use of other such physical layers is not supported by this standard. It is anticipated that other related standards will be developed to support other physical layers. In such a case, only those modules compliant with the same standard would be interoperable.

6.5 *Media and Connectors* — With Ethernet, there is a choice of media used (twisted pair, coax, etc.). It is beyond the scope of this standard to specify the media used. The media is thus left to the configuration of a specific cluster tool and may depend on the available media used in the factory.

Most Ethernet interfaces provide a media-independent AUI interface. A transceiver is then used to access the physical media. In order to be compliant with this standard, a module shall provide an AUI interface. Many interfaces also provide a media-specific interface. This media specific interface may be used in the event that the cluster tool media is the same as the media specific interface.

7 Compliance

Compliance with this standard includes adherence to all stated requirements where implemented. In order to be compliant to this standard, an implementation shall:

1. use SECS-II messages as specified by CTMC document,
2. use SEMI E37 (HSMS) and SEMI E37.2 (HSMS-GS),
3. allow and document IP Address configuration,
4. allow and document Session ID configuration,
5. allow and document Port ID configuration,
6. use Ethernet as defined in IEEE 802.3,
7. provide an AUI connector.

Some Ethernet interfaces provide additional connectors (such as 10 BASE 2 or thinnet). These may be used in cases where this is the medium chosen, but for compliance, an AUI connector shall be provided.

APPENDIX 1
APPLICATION NOTES

NOTE: This appendix was approved as a part of SEMI E38.1 by full letter ballot procedure.

A1-1 Example Configuration

The following example shows configuration of a cluster tool consisting of an existing process module with a new cluster controller and other modules.

The cluster consists of a Cluster Controller, two Process Modules (PM1, PM2), and a combined Transport/Cassette Module (TM/CM).

A1-1.1 *IP Addresses* — IP addresses are chosen so that they are unique to each platform on the network and consistent with the conventions used at the facility. For this example, the convention is to use 192.25.63.X for IP Addresses, where X is between 0 and 255. PM1 had an existing IP address of 192.25.63.122, which will not be changed. IP addresses starting at 205 are unused and thus available. Thus, the following IP addresses are chosen:

```
Cluster Controller      192.25.63.205
TM and CM               192.25.63.206
PM1                   192.25.63.122
PM2                   192.25.63.207
```

As part of the configuration, the IP Address of each module is identified as shown above, on all of the platforms. This configuration depends on the TCP/IP software used on the respective platforms.

A1-1.2 *Port Numbers* — TCP reserves port numbers below 1024 in RFC1340. There are no other TCP/IP programs on any of the platforms in the cluster using port number in the range 2000-10000, so the following port numbers are chosen:

```
2000 first TM
5000 first CM
8000 first PM
8001 second PM
```

The cluster controller is always a client and initiates the connections, so no port number assignment is required.

A1-1.3 *Session ID Assignment* — Before defining the Session ID's, recall that the following connections are required:

```
Client          Server
CC              TM
CC              CM
CC              PM1
CC              PM2
CM              TM
PM1             TM
PM2             TM
```

The Session ID's are arbitrary. PM1 already has ID's which are as follows:

```
PM1 IDs
Clients
Object Services           1     CC, TM
Processing Management   100     CC
Recipe Management       101     CC
Material Movement       102     CC, TM
Exception Management    103     CC
Event Reporting         104     CC
```

The other modules have not yet been set, so we choose to begin with ID 64.

The session IDs are established as follows:

```
PM2  Ids
Clients
Object Services           1     CC, TM
Processing Management    64     CC
Recipe Management        65     CC
Material Movement        66     CC, TM
Exception Management     67     CC
Event Reporting          68     CC

TM  IDs
Clients
Object Services           1     CC
Material Movement        64     CC
Exception Management     65     CC
Event Reporting          66     CC

CM  IDs
Clients
Object Services           1     CC, TM
Material Movement        64     CC, TM
Exception Management     65     CC
Event Reporting          66     CC
```

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 E39-0703
OBJECT SERVICES STANDARD: CONCEPTS, BEHAVIOR, AND SERVICES

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 American Information & Control Committee on March 12, 2003. Initially available at www.semi.org May 2003; to be published July 2003. Originally published in 1995; previously published November 2002.

CONTENTS

LIST OF TABLES

SEMI E39-0703
OBJECT SERVICES STANDARD: CONCEPTS, BEHAVIOR, AND SERVICES

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 American Information and Control Committee on March 12, 2003. Initially available at www.semi.org May 2003; to be published July 2003. Originally published in 1995; previously published November 2002.

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: 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 Referenced Standards

3.1 *ISO Standard*[1]

ISO 9595 — Common Management Information Service (CMIS)

3.2 *Other Standards*

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* — a object formally defined in SEMI standards and in compliance with the fundamental requirements of SEMI E39 (Object Services Standard: Concepts, Behavior, and Services).

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, 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 The term “object” may refer to either an object class or an instance of a class.

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.

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 fulfil its requirement completely; an exception will be raised if the service can not 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

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

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.

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 Section 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 Section 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.

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.

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 Section 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.

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 Book of Standards, Equipment Automation/ Software Volumes 1 and 2[5]

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

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.

8.1.1 Object-oriented analysis is a widely accepted tool, both for applications that are implemented using

5 This set of documents may be obtained from SEMI, 3081 Zanker Road, San Jose, CA 95134.

6 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

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 Section 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 Section 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 Section 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 Sections 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.

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 Section 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*.

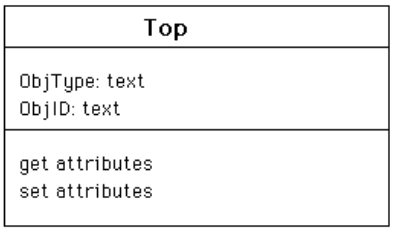

Figure 1
Top Object

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.

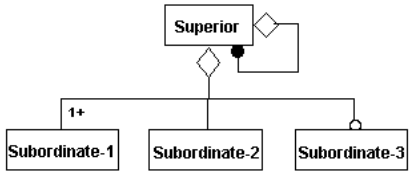

Figure 2
Aggregation with Two Component Types

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.

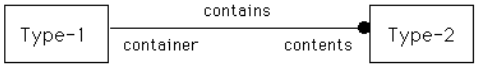

Figure 3
Container and Contents

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.

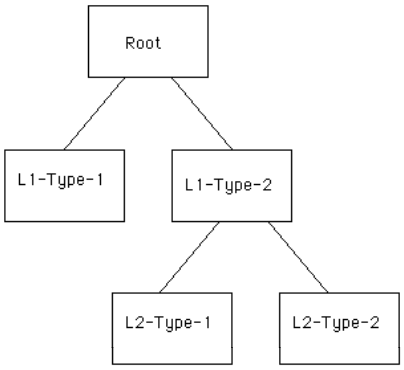

Figure 4
Example of Owner Hierarchy

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:

"$type_1$:id_1>...$type_n$:id_n>"

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 " id_1>id_2>.id_n>" . 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),
- an owner object knows the types of objects that are its components and/or contents, or that it otherwise

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

supervises, and is able to determine their identifiers (see GetType, Section 11.6),

- an owner object is able to determine the types and identifiers of its owned objects.

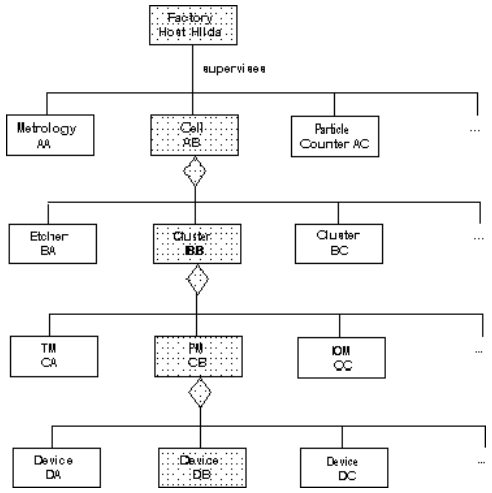

Figure 5
Hierarchical Relationships in the Factory

9.4 *Multiple Inheritance Hierarchy* — Multiple hierarchical inheritance may be required as an optional capability for pointing to a specific object that provides the same services from the inherited objects. Usually, inherited services are implemented by the domain object such that specifying "overrides" of attributes or services from inherited objects is not required. Sometimes the capability to override specified attributes and services from inherited object is necessary to resolve duplicated attribute names or services or to allow duplicated names in user extended objects.

9.4.1 If Andy an object TypeA inherits from object TypeB and TypeC and a service requestor is focusing services or attributes of the object TypeB rather than those with the same name in the object TypeC, the following text string form could specify the focused part of the object Andy:

`TypeA@TypeB:Andy`

9.4.2 If any names of service and/or attribute that TypeB provides are the same as those provided by TypeA or TypeC, then, the object Andy will have to be specified with the complete explanation from the supplier as how the object handles the exception and the differences from the original attribute or services. So, the above expression is the same as "TypeA:Andy" without notes to describe difference.

10 Additional Operations

10.1 This section defines operations that are common to many types of public objects but are not required for all types of public objects. These operations are not required for OSS compliance. However, they may be required for support of specific object types defined by other standards that use OSS services.

10.2 *Create and Delete* — Creation and deletion are optional services for object lifecycle management. Object definitions may specify conditions under which the services are supported. Objects may be created and/or deleted in other ways. For example, the creation and deletion of certain types of persistent objects may be outside of the scope of OSS services, while transient objects may be created and/or deleted automatically by their owners as the result of other activities.

10.2.1 *Create* — The create operation creates an instance of an object type.

10.2.1.1 Initial settings for one or more attributes may be specified by the service user. Initial settings may be required, prohibited, or optional, depending upon the specific object type. The ability to set attribute values through the create operation shall be clearly identified as part of the object definition for those objects supporting the create operation. For example, the ability to set the value of the object identifier ObjID may be required, optional, or prohibited, depending upon the type of object.

10.2.1.2 The object definition may also specify attributes of the new object instantiation that are returned to the service user by the service provider.

10.2.1.3 A request to create an object is invalid if it does not provide all required attribute settings, if it attempts to set prohibited attributes, or if it provides values invalid for a given attribute.

10.2.1.4 The definitions of certain objects may specify that the delete service only be provided to, or under the authority of, the service user that requested the original create operation. In this case, a private attribute (that is, an attribute not accessible through services defined in Section 11) named Delete Token shall return a unique integer as an attribute of the new object. This attribute is then used in a subsequent request to delete the object. The service user may delegate this token at its own discretion.

10.2.1.5 The create operation is invoked by the service Create sent to the owner that is to instantiate the specified object. In order for the Create service to be accepted if no owner for the object to be created is defined in the target entity or if it is physically outside of the domain of the entity; the object specifier shall define the owner as " " (an empty string). If the entity

is just the equipment rather than a component of the equipment; the owner may be defined as"Equipment" even if no such object is defined.

10.2.2 *Delete* — The delete operation is the inverse of the create operation. Individual object types may restrict deletion based on well-defined set of criteria. For example, the definition of a type of container may specify that a container may only be deleted when it is empty.

10.2.2.1 If an object to be deleted requires the authority of the service user that invoked the original create operation, then the Delete Token attribute of the target object shall be provided by the user of the delete service.

10.2.2.2 A request to delete an object is invalid if it does not provide all required attribute settings or if it provides values invalid for a given attribute.

10.2.2.3 The delete operation is invoked by the service Delete sent to the object to be deleted.

10.3 *Attachment* — When an object receives a remote request for an operation, it may be unable to identify the requestor. At the same time, it may be necessary for an object to determine if certain messages were sent by its supervisor. This conflict is resolved by formalizing the relationship between a supervised object and its supervisor.

10.3.1 Attachment is a dynamic behavioral binding between two objects in a hierarchical relationship where it is important that the attached (supervised) object be able to differentiate certain service requests made by its supervisor from other requests. An attached object has exactly one supervisor, as illustrated in Figure 6.

10.3.2 An attached object may be detached from its current supervisor, or it may be reattached to a different supervisor. The latter operation is used when the previous supervisor has become damaged and is not intended as a general method for moving the object's attachment, as the former supervisor is not notified of the change.

10.3.3 *Attach* — When sent to an unattached object, the attach operation creates a logical connection between the object and a supervisor. The object that is being attached creates a unique, private, non-zero numeric value, called an attach token. The attach token is treated as an attribute named AttachToken. It is a private attribute of the attached object, not visible through basic OSS services (see Section 11).

10.3.3.1 When an object is first created, it is unattached, and the value of its attach token is set to a default value of zero. An object is attached to another object as a formal relationship. The operation of attach generates the attach token, which is returned to the service user. The supervisor uses this token in subsequent critical service requests to identify itself to the attached object.

10.3.3.2 The attach operation also may accept settings for one or more attributes of the attached object, as defined for the attached object type. This allows the supervisor to provide values for attributes that are otherwise regarded as read-only values and cannot be set through the SetAttr service defined in Section 11. An example of an attribute value that it is often desirable to access in this way is the object specifier of the supervisor.

10.3.3.3 Requests to attach an object that attempt to set attributes shall be denied unless the attributes are among those accessible to the supervisor and all specified settings represent valid values of those attributes. The object type and identifier shall not be changed by the supervisor.

10.3.3.4 An attached object is owned by its supervisor.

10.3.3.5 The attach operation is invoked by the message service Attach by an intended supervisor. Requests to attach an already attached object shall be denied.

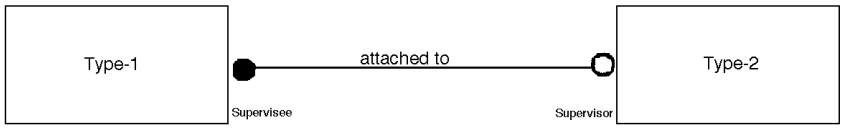

Figure 6
The Attachment Relationship

10.3.4 *Attach Set Attributes* — A supervisor may need the ability to set certain read-only attributes of one of its attached objects subsequent to the attach operation. The set attached attributes operation provides this capability to a supervisor of attached objects. The supervisor identifies itself by using the attach token for that object.

10.3.4.1 The definition of the attached object type may restrict the attributes that may be set by this operation. The attach set attributes operation shall be denied if the attributes or settings are invalid for this operation.

10.3.4.2 The attach set attributes operation is invoked by the message service AttachSetAttr sent by the object's supervisor and is otherwise invalid.

10.3.5 *Detach* — The detach operation breaks the logical connection between an attached object and its supervisor. The attach token is reset to zero, and the object becomes unattached.

10.3.5.1 The detach operation is invoked by the message service Detach sent by the object's supervisor and is otherwise invalid.

10.3.6 *Reattach* — The reattach operation is similar to the attach operation, except that it is sent only to an attached object to change the logical connection from the attached object to a new supervisor. This operation is used to replace a damaged supervisor. Any existing connections to the old supervisor shall be closed. Note the reattach operation is vulnerable to misuse. Applications supporting the reattach operation may have additional requirements, such as notification sent to the previous supervisor, to provide a trace of misuse.

10.3.6.1 The reattach operation provides a unique attach token that has not been previously used by the attached object. This effectively disables its previous supervisor.

10.3.6.2 An attempt to reattach an object shall be denied unless the attributes and settings are valid for the operation.

10.3.6.3 The reattach operation is invoked with the message service Reattach sent to an attached object by the object's new supervisor and is otherwise invalid.

10.3.7 *Attach Supervised Object* — The attach supervised object operation is invoked by a service user to request a supervisor to attach to itself a valid target object. If the supervised object does not already exist or is not a valid target type for the supervisor, then the attached supervised object operation shall fail. The results of the operation are returned to the service user.

10.3.7.1 The attach supervised object operation is invoked with the service AttachSupervisedObject.

10.3.8 *Detach Supervised Object* — The detach supervised object operation is invoked by a service user to request a supervisor to detach from itself an attached object. The supervisor sends a detach request to the specified attached object. Regardless of any response from the specified supervised object, the supervisor shall consider it thereafter as unattached. This is required to allow a damaged attached object to be removed. This operation is successful except when the target object is not already attached to the supervisor.

10.3.8.1 The detach supervised object operation is invoked with the service DetachSupervisedObject.

11 Services and Scenarios

11.1 The services defined by OSS are contained in Table 2.

Table 2 Object Services

Message Service Name	*Type*	*Description*
GetAttr	R	Get values of specified attributes based on attribute filters.
SetAttr	R	Set values of specified attributes based on attribute filters.
GetAttrName	R	Get names of attributes of objects owned by the service provider.
GetType	R	Get types for objects owned by the service provider.

11.2 *Scope* — Scope is supported through the object specifier parameter. The object specifier provides a method for pointing to objects owned by the service provider that are otherwise not accessible to the service user. That is, the service provider may not provide direct services for, or to, that object. This can occur when an object for which the service provider offers services is an owner object, such as an aggregate or file directory, as described in Section 9. In some cases, the owned object may itself be remote from the service provider. For example, the owned object may be provided by a separate system.

11.2.1 The service provider may be an owner object with managerial responsibilities, and therefore it may prohibit direct access to one of its component or contained objects. In this case, requests concerning an owned object from a service consumer shall be directed to the owner object instead.

11.2.2 Object services use the *object specifier* for the owner of the target object(s). The construction of the object specifier is defined in Section 9.

11.2.3 Object type may be omitted from the object specifier where it may be otherwise determined without ambiguity. However, inclusion of type is always valid.

11.3 *Filtering* — An *attribute filter* is an optional set of one or more *attribute qualifications*. An *attribute qualification* is a boolean expression that makes a statement about the presence or values of attributes in a target object. It identifies an attribute, a *qualifying value* of that attribute, and a *qualifying relationship* that the value has to the target attribute. The attribute filter is a boolean that consists of the expression formed by an AND of the set of qualifications.

11.3.1 If the attribute is of text form, the qualifying value may be used as a *mask* with the embedded *wild characters* "? " (question mark) and "*" (asterisk). The character "? " may be used within the mask to represent "any single character" and may be repeated. The string "? ? ? ? ? " represents any text string with a length of five characters.

11.3.2 The character "*" may be used in the mask to represent a variable-length string, including a null string. The string "*x" represents a string of any length that ends in "x"; the string "x*" represents any string that begins with "x".

11.3.3 When the character "*" is used by itself as the string "*", however, it represents any string of any non-zero length. It may also be repeated within a string, as in "*x*y*", which represents any string with embedded characters "x" and "y", such as "abxaby", "x_y", and "xy".

11.3.4 The comparison for text characters shall be case insensitive.

11.3.5 The qualifying value specifies a qualifying relationship, "R", between the value specified in the attribute qualification and the matching attribute of object instances. The qualifying value for the attribute is compared with the attribute of an instance of the target object to test the relationship.

11.3.6 If the qualifying value has the relationship "R" to the attribute of the target object, then the instance *qualifies* for the relationship and is included in the set of objects for which requested attributes are returned. For example, if the attribute name is "Length," qualifying value is the number 5, and the qualifying relationship is "is less than," then the Length attributes of all instances of the target object are tested. All instances with a "Length" attribute greater than 5 or equal to 5 qualify.

11.4 *Object Services Parameter Dictionary* — Table 3 defines all of the parameters, including the elements of complex parameters, used in object services.

Table 3 Object Services Parameter Dictionary

Parameter	*Definition*	*Form*
AttrData	The value of an attribute.	Varies with attribute. The form of an attribute's value is specified as part of the definition of an object type.
AttrFilter	Attribute filter.	Structure composed of AttrName, AttrData, AttrReln.
AttrName	The attribute's name.	Text. Varies with object type.
AttrReln	Qualifying relationship between the qualifying value and the matching attribute of object instances.	Enumerated: Equal To Not Equal To Less Than Less Than or Equal To Greater Than Greater Than or Equal To Present (specified attribute is present) Absent (specified attribute is absent) Contained (The qualifying value is in (i.e., equal to a member of) the set of the attribute's values.) Not Contained (If omitted, the relationship "Equal to" is assumed.)
AttrSetting	The name and value of an attribute.	Structure composed of AttrName and AttrData.

Parameter	*Definition*	*Form*
ErrorCode	Contains the code for the specific error found.	Enumerated: Unknown Object in Object Specifier Unknown Target Object Type Unknown Object Instance Unknown Attribute Name Read-Only Attribute — Access Denied
ErrorText	Text in support of the error code to provide additional information.	Text.
ObjAck	Acknowledgement code.	Unsigned integer. Possible values: Successful completion of request Error
ObjectAction	The name of an action to be performed on, or by, an object.	Text. Enumerated per individual object.
ObjectAction Acknowledgement	Status of action request.	Enumerated: Action was performed successfully. Action does not exist. Cannot perform now – try later. Action will be performed and notification sent later. Action is prohibited. Target object unknown.
ObjectActionParameter Request	Structure composed of a name and a value for a given object parameter used in the request.	Structure of ServiceParameterName and ServiceParameterValue.
ObjectActionParameter Result	Structure composed of a name and a value for given object parameter used in the result.	Structure of ServiceParameterName and ServiceParameterValue.
ObjectActionReportStatus	Status of receipt of an ObjectActionComplete-Notification.	Binary.
ObjectActionStatus	Result of the action request.	Structure of ObjectActionAcknowledgement and Status.
ObjectLinkID	Set to non-zero if and only if additional completion reports will be sent.	Unsigned integer.
ObjectServiceList	Specified the services supported by a given object type.	Structure composed of ObjType, ServiceName.
ObjID	Object identifier (ObjID).	Varies with object type.
ObjSetting	Attribute values for an object instance.	Structure composed of ObjID and (List of) AttrSetting.
ObjSpec	The object specifier, used to specify the owner of the target object.	Formatted text.
ObjStatus	Information concerning the success or failure of the operation.	Structure consisting of ObjAck and Status.
ObjType	Object type.	Object types are text strings reserved by individual standards. Object Services do not reserve object types.
OperationID	Identifies a specific request.	Unsigned Integer.
ServiceName	The name of a service or action to be performed on, or by, an object.	Text.
ServiceParameterDef	A structure of a service name and a list of the parameter	A structure of ServiceName and associated list of ServiceParameterNames.

Parameter	*Definition*	*Form*
	names that are supported by an object type.	
ServiceParameterName	Parameter name.	Text.
ServiceParameterValue	Parameter value.	Any. Varies depending on the parameter.
Status	Error information.	(List of) Structure composed of ErrorCode and ErrorText.

11.5 *Get and Set Attributes* — The GetAttr and SetAttr services are provided for reading and setting values of an object' s attributes. Theyaddress attribute values of instances of objects. Figures 7 and 8 show the message flow for the GetAttr service.

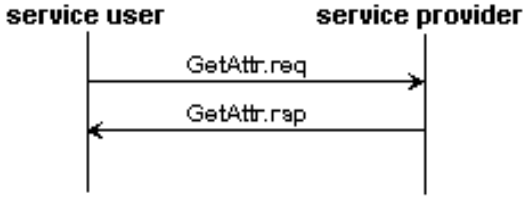

Figure 7
GetAttr Message Flow

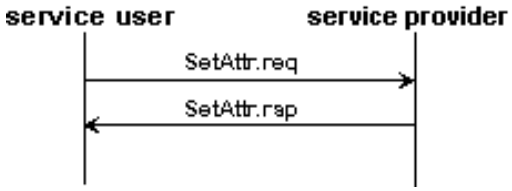

Figure 8
SetAttr Message Flow

11.5.1 GetAttr and SetAttr allow access to attributes of objects that are owned by another object and are capable of addressing multiple levels of ownership (object hierarchy).

11.5.1.1 For example, GetAttr may be used as a “get directory” request to find the identifiers of all instances of one or more owned objects.

11.5.2 GetAttr and SetAttr may be used for objects that use a complex identifier, such as recipes with separate attributes for name and version.

11.5.3 GetAttr and SetAttr allow searching for those instances of an object that satisfy certain conditions through the specification of one or more qualifiers or filters. Only attributes of those instances of the desired object type that satisfy the conditions defined in the filters are returned by the service provider.

11.5.3.1 The filter is familiar to users of file services as a convenient way to obtain a partial directory of files, such as files with a date later than a specified date. This is particularly useful when the list of contents may be quite long.

11.5.4 The attribute value (all or a part of the list of ObjSetting) returned by the GetAttr service may be invalid if the acknowledge code of ObjStatus indicates a failure. In that case; the value(s) requested by the invocation of GetAttr service may be unknown, invalid or unavailable by any reason (e.g. corresponding hardware status, lack of privilege, mutual exclusion or other requirement not fulfilled by the application). Supplier shall document all conditions or timing requirements of any attributes if they are not always available.

11.5.5 The attribute value (all or a part of the list of ObjSetting) replied by the SetAttr service may be still unchanged if the acknowledge code of ObjStatus indicates a failure. In that case; the attribute value may be unchangeable, inhibited or blocked internally by any reason (e.g. corresponding hardware status, lack of privilege, mutual exclusion or other requirement not fulfilled by the application. The object providing the SetAttr service shall identify whether or not it is valid to change or set an attribute value. Supplier shall document all conditions and timing requirements of each attribute if they are not always allowed to be set.

11.5.6 *Object Specifier* — This parameter is used for specifying an owner object and may be omitted from the message. When included, it is a formatted text string that conforms to the description in Section 9. An object specifier is rejected by service providers that do not use owner or owned objects.

11.5.7 *Filter* — This is a optional list of qualifications to be applied to specifed attribute and may be omitted from the message. When included, the complete filter is formed by a logical AND of each attribute qualification in the list.

11.5.8 *Requested Attribute* — The final parameter in the GetAttr service gives the names of one or more attributes of interest. These are the attributes whose values shall be returned in the response, in the order requested.

11.5.9 *Settings* — In the SetAttr service, a list of one or more attribute name/value pairs is specified. This is the set of desired attribute values. SetAttr rejects any attempt to set a read-only attribute and returns

information in Status, indicating the error and the attribute.

11.5.10 *ObjStatus* — This is a required parameter in the response message. It is a structure that consists of an acknowledge code and a list of status parameters. The acknowledge code is used to indicate errors that apply to the response as a whole. A status parameter is a structure consisting of a code indicating a specific error and accompanying text providing additional information. Text is often used to inform a person of the results of an operation.

11.5.11 Tables 4 and 5 define the parameters for the GetAttr and SetAttr services, respectively.

Table 4 GetAttr Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	C	-	The object specifier, used to specify the owner of the target object.
ObjType	M	-	The type of the target object.
(List of) ObjID	C	-	The identifier of the target object. If present, identifiers are treated as a pre-filter of the form "ObjID = <identifier>", where the individual identifiers are joined with logical OR's.
(List of) AttrFilter	C	-	An attribute filter to be applied to the target object. If both AttrFilter and ObjID are omitted, attributes are requested for all instances of the target object type.
(List of) AttrName	C	-	The name of a desired attribute. If the name is "ObjType," and if ObjType (above) is omitted, then the object type of all subordinate objects is requested. If omitted, then all public attributes of the target object are requested.
(List of) ObjSetting	-	M	Attributes that match the specified characteristics, per qualifying object instance. Setting value may not be valid if corresponding ObjStatus shows a failure.
ObjStatus	-	M	Information concerning the success or failure of the request.

Table 5 SetAttr Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	C	-	The object specifier, used to specify the owner of the target object.
ObjType	M	-	The type of the target object.
(List of) ObjID	C	-	The identifier of the target object. If present, is treated as a primary filter of the form "ObjID = <identifier>".
(List of) AttrSetting	M	-	An attribute's name and desired value.
(List of) ObjSetting	-	M	The attributes' resulting values. Requested setting value(s) may have not changed if corresponding ObjStatus shows a failure.
ObjStatus	-	M	Information concerning the success or failure of the request.

11.6 *GetType* — GetType is a service that is directed at object types rather than object instances. It is used to ask an owner object for the object types that it owns.

11.6.1 Figure 9 shows the message flow for the GetType service.

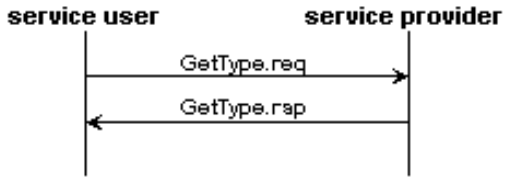

Figure 9
GetType Message Flow

11.6.2 Comments for the ObjSpec and ObjStatus parameters of GetAttr and SetAttr also apply to the parameters for the GetType service.

11.6.3 Table 6 defines the parameters for the GetType service.

Table 6 GetType Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	C	-	The object specifier, used to specify the owner of the target object.
(List of)ObjType	-	M	The types of objects owned by the service provider.
ObjStatus	-	M	Information concerning the success or failure of the request.

11.7 *GetAttrName* — GetAttrName is a service that is directed at object types rather than object instances. It is used to ask for the names of the attributes of one or more objects owned by the service provider.

11.7.1 Figure 10 shows the message flow for the GetAttrName service.

Figure 10
GetAttName Message Flow

11.7.2 Wild characters may be used in the object types requested.

11.7.3 Comments for the ObjSpec and ObjStatus parameters of GetAttr and SetAttr also apply to the parameters for the GetAttrName service.

11.7.4 Table 7 defines the parameters for the GetAttrName service.

Table 7 GetAttrName Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	C	-	The object specifier, used to specify the owner of the target object.
(List of)ObjType	M	-	The type of the target object owned by the service provider. ObjType may use wild characters.
(List of)AttrName	-	M	The names of the attributes of each of the target object(s).
ObjStatus	-	M	Information concerning the success or failure of the request.

12 Additional Services

12.1 Additional services described in this section may have some restrictions when invoked. In some cases, services that are invoked may report a failure. For example: the owner of an object shall not delete the object during its lifecycle or while it is valid and required by another agent. Existence of an object may depend on preconditions, provided parameters, state of another object, corresponding operation, context of application, etc. Supplier shall document any restrictions or information regarding the validity of the additional services listed below.

12.1.1 This section defines the services required for additional operations defined in Section 10. These services are listed in Table 8.

Table 8 Additional Object Services

Message Service Name	*Type*	*Description*
Create	R	A request is made to create an object.
Delete	R	A request is made to delete an object.
Attach	R	A supervisor requests an object to attach itself to the supervisor.
AttachSetAttr	R	A supervisor requests to set one or more attributes of an attached object.
AttachSupervisedObject	R	A request is made to a supervisor to attach another object to the supervisor.
Detach	R	A supervisor requests an object to detach itself from the supervisor.
DetachSupervisedObject	R	A request is made to a supervisor to detach a specified attached object.
Reattach	R	A supervisor requests an object to reattach to itself to the new supervisor.
ObjectAction	R	A request for an object to perform an action.
ObjectActionCompletion	N	Notification that a delayed action has been completed, sucessfully or unsuccessfully.
GetServiceNames	R	A request to an object owner for a list of services supported by its owned objects.
GetServiceParameters	R	A request to an object for the service parameters supported for specified services.

12.2 *Object Services Parameter Dictionary* — Parameter definitions are provided in Section 11.1.

12.2.1 Table 9 defines the parameters used in services defined in Section 12, in addition to those parameters defined in Table 3.

Table 9 Additional Object Services Parameter Dictionary

Parameter	*Definition*	*Form*
AttachToken	Attach token provided by an object at the time it is attached or reattached.	Unsigned integer
TargetSpec	Object specifier of target object to attach or detach.	Formatted text for object specifier.

12.3 *Create* — The service user may request the service provider to create a new object and assign the value of one or more of its attributes. The request may be denied if an attempt is made to set attributes that are not allowed for that object type.

12.3.1 Parameters for Create are listed in Table 10.

Table 10 Create Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	M	M	In the request, the object specifier of agent providing the object create service. In the response, the object specifier of the new instance. NOTE: The values may be different in the request and the response. The parameter in the request may be " " (an empty string) if there is no object or the owner object is outside of the domain of the receiving entity. If the entity is the equipment itself rather than a component of equipment, the owner may be defined by "Equipment" even if no such object is defined.
ObjType	M	-	The object type to be created.
(list of) AttrSetting	C	C	Initial attribute settings. Conditions for inclusion of an attribute are object-specific in both request and response.
ObjStatus	-	M	Information concerning the result of the requested operation.

12.4 *Delete* — The service user may request to delete an object.

12.4.1 Parameters for Delete are listed in Table 11.

Table 11 Delete Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	M	-	The object specifier of the object to be deleted.
(list of) AttrSetting	C	C	Conditions for inclusion of an attribute are object-specific both request and response.
ObjStatus	-	M	Information concerning the result of the requested operation.

12.5 *Attach* — A service user may request an object to attach itself to the service user.

12.5.1 Parameters for Attach are listed in Table 12.

Table 12 Attach Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	M	-	Specifier of server object to be attached.
AttachToken	-	M	Attach token. A value of zero shall be used if and only if the request to attach is denied.
(list of) AttrSetting	C	C	Settings for specific attributes. Object-dependent. Attributes in the list sent in the request and the response may be different.
ObjStatus	-	M	Information concerning the result of the requested operation.

12.6 *AttachSetAttr* — A supervisor may request an attached object to set one or more attributes at any time while the object is attached.

12.6.1 Parameters for AttachSetAttr are listed in Table 13.

Table 13 AttachSetAttr Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	M	-	Specifier of server object to be attached.
AttachToken	M	-	Attach token.
(list of) AttrSetting	C	C	Settings for specific attributes. Object-dependent. Attributes in the list sent in the request and the response may be different.
ObjStatus	-	M	Information concerning the result of the requested operation.

12.7 *Detach* — The supervisor of an attached object may request the object to detach itself.

12.7.1 Parameters for Detach are listed in Table 14.

Table 14 Detach Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	M	-	The specifier of the attached object.
AttachToken	M	-	Attach token.
(list of) AttrSetting	C	C	Attribute settings. Object-dependent. Attributes in the list sent in the request and the response may be different.
ObjStatus	-	M	Information concerning the result of the requested operation.

12.8 *Reattach* — An object may request an object attached to another supervisor to reattach to itself to the service user as its new supervisor.

12.8.1 Parameters for Reattach are listed in Table 15.

Table 15 Reattach Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	M	-	Specifier of object to reattach.
AttachToken	-	M	Attach token. A value of zero shall be used if and only if the request to reattach is denied.
(list of) AttrSetting	C	C	Specific attribute settings. Object-dependent. Attributes in the list sent in the request and the response may be different.
ObjStatus	-	M	Information concerning the result of the requested operation.

12.9 *AttachSupervisedObject* — A service user may request a supervisor to attach a specified object.

12.9.1 Parameters for AttachSupervisedObject are listed in Table 16.

Table 16 AttachSupervisedObject Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	M	-	The object specifier for the supervisor.
TargetSpec	M	-	The object specifier of the object to attach.
(list of) AttrSetting	C	C	Attribute settings. Object-dependent. May be omitted. Attributes in the list sent in the request and the response may be different.
ObjStatus	-	M	Information concerning the result of the requested operation.

12.10 *DetachSupervisedObject* — A service user may request a supervisor to detach a specified attached object.

12.10.1 Parameters for DetachSupervisedObject are listed in Table 17.

Table 17 DetachSupervisedObject Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	M	-	The object specifier of the supervisor.
TargetSpec	M	-	The object specifier of the object to be detached.
(list of) AttrSetting	C	C	Attribute settings. Object-dependent. May be omitted. Attributes in the list sent in the request and the response may be different.
ObjStatus	reserved	reserved	Information concerning the result of the requested operation.

12.11 *ObjectAction* — A service user may request a specific action to be performed by, or on, a particular object. The object specifier represents either the target object as its owner, depending upon the object and action.

12.11.1 Completion, either normal (successful) or abnormal, may be indicated in either of two ways. If the action can be completed in a very short time, then it should be completed before a response to the request is sent. If the action takes more time than would be normal for the response, or if the action must wait for one or more conditions to be fullfilled, then a response to the request indicating the action will be performed later and the notification sent of its completion. See Section 12.12 below.

12.11.2 Figure 11 illustrates the first case, where the action is completed before the response is sent. The value of ObjectActionAcknowledgement should be set to any value other than “Action will be performed and notification sent later”.

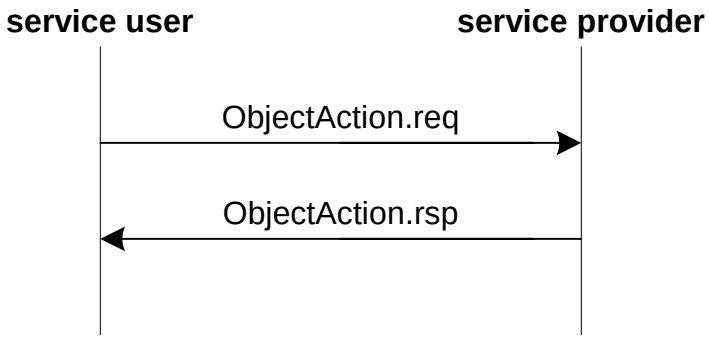

Figure 11
ObjectAction Request and Response

12.11.3 It is recommended that OperationID be always set to a unique non-zero value. However, if a value of zero is used, then the host should not send additional request until informed of the results of the outstanding request. Otherwise it may be ambiguous to which request an ObjectActionCompletion notification pertains. The definition of the target object type must include specification of the specific actions that it supports, together with the arguments that are required and the arguments that are optional for each action.

Table 18 ObjectAction Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
OperationID	M	M(=)	Operation identifier. Used where the service user may send multiple requests to link subsequent events to the original requests. May be zero otherwise.
ObjSpec	C	-	The object specifier of the target object or owner of the target object.
ObjectAction	M	-	Specific action requested.
(list of) ObjectActionParameter Request	C	-	A list of name/value pair arguments providing additional information about the request.
(list of) ObjectActionParameter Result	-	C	A list of name/value pair arguments providing additional information about the reply.
ObjectActionStatus	-	M	Information concerning the result of the requested action.
ObjectLinkID	-	M	Set to non-zero if and only if additional completion reports will be sent.

12.12 *ObjectActionCompletion Notification* — This notification is sent to the requestor of an earlier action when the completion of that action occurred after the response to the request was sent.

12.12.1 Figure 12 illustrates the scenario where the notification message is sent. The value of ObjectActionAcknowledgement should be set to “Action will be performed and notification sent later”.

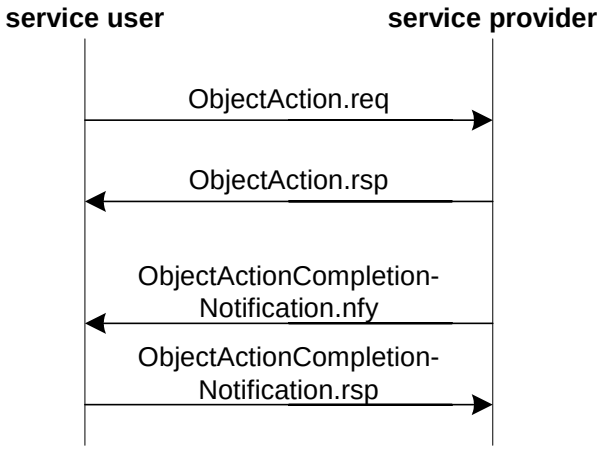

Figure 12
ObjectAction Request, Response, and Notification

Table 19 ObjectActionCompletion Notification

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
OperationID	M	-	Operation identifier. Set to the same value as in the original request.
OperationActionStatus	M	-	Information concerning the result of the service completion.
(list of) ObjectActionParameter Result	C	-	A list of name/value pair arguments providing additional information about the reply.
ObjectActionReportStatus	-	M	Indicates whether the report has been received with no errors.

12.13 *GetServiceNames* — A service user may ask an object owner for a list of the services supported by its owned objects.

12.13.1 The definition of the target object type must include specification of the specific actions that it supports, together with the arguments that are required and the arguments that are optional for each action.

Table 20 GetServiceNames Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	C	-	The object specifier of the owner object.
(list of) ObjType	M	M	A list of object type values
ObjStatus	-	M	Information concerning the result of the requested action.
(list of) ObjServiceList	-	M	List of structures of the form ObjType, (list of) service names.

12.14 *GetServiceParameters* — This message may be sent to an object to determine the service parameters that are used by that object for a set of different services that it supports.

Table 21 GetServiceParameters

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
ObjSpec	C	-	The object specifier of the owner object.
ObjType	M	-	The type of object referenced.
(list of) ServiceName	M	-	A list of service names.
ObjStatus	-	M	Information concerning the result of the requested action.
(list of) ServiceParameterDef	-	C	List of structures that contain a service name with a list of parameter names.

13 Applications

13.1 This section provides examples of applications of object services.

13.2 *Get All of an Object's Attributes* — In the most straight-forward case, GetAttr is used to get all of the current attribute values of a single target object known to the provider of the GetAttr service. The object specifier is omitted, and the object type is set to the type of the target object. To get all of the object's attributes, both the attribute filter and the names of requested attributes are also omitted.

13.3 *Determine All Objects of a Specific Type and with Specific Characteristics* — To determine subordinate objects with specific characteristics, such as of type "Module" with an identifier *ObjID* starting with "ABC", set object type to "Module". In the attribute filter, set the attribute name to "ObjID", set the attribute value to "ABC*", and set the qualifying relationship to "is equal to" (or omit it).

13.4 *Determine Specific Attributes of a Specific Object Instance* — To determine the current value of certain attributes of a specific object, set the object type appropriately. Either set the list of ObjID to a list containing the one identifier of the target instance, or alternatively, in the attribute filter, set the attribute name to "ObjID", set the attribute value to the identifier, and set the qualifying relationship to "is equal to" (or omit it).

13.5 *Determine Types of Subordinate Objects* — To determine all object types that are owned by another object, the GetType service is used with the single parameter of the object specifier of the owner. The service provider returns an error of "Unknown object type" if it has no types of owned objects.

13.6 *Determine Names of Attributes of Subordinate Objects* — To determine the names of the attributes of specific objects owned by the service provider, the GetAttrName service is used with two parameters: the object specifier of the owner and a list of the types of the target objects. The service provided returns an error of "Unknown object type" if it has no types of owned objects.

14 Requirements for Compliance

14.1 Object Services are common to all service resources that define public objects with operations for getting (reading) and setting (writing) attribute values and for getting object types and attribute names for objects. Object services provide common definitions for public objects and object services that may be incorporated into these service resources, thereby avoiding unnecessary duplication.

14.2 *Fundamental Requirements* — All objects compliant to any part of OSS shall be subtypes of the top object. That is, they shall inherit (provide) the *ObjType* and *ObjID* attributes as described in Section 8.2.

14.2.1 They shall provide documentation of their public attributes in the form of an Object Attribute Definition table as described in Section 5.2.

14.2.2 They shall also provide the services GetAttr and SetAttr as defined in Section 11.5. Support for an owner's object specifier and filter are not required for fundamental compliance with OSS. If the service user provides parameters for scope or filter in its request, they may be ignored by the service provider. However, they shall not cause errors in the response due to their presence.

14.3 *Additional Capabilities*

14.3.1 *Filters* — Support for attribute filters in GetAttr and SetAttr requests is an optional capability.

14.3.2 *Owner Objects* — All owner objects (aggregates, containers, and supervisors) shall support both scope and filters for the GetAttr and SetAttr services, as defined in Sections 9.3, 11.2, and 11.3.

14.3.2.1 In addition, they shall provide the service GetType as defined in Section 11.6, with support for wild characters in the specification of object types.

14.3.3 *Multiple Inheritance Hierarchy* — Inheriting objects shall identify inherited objects with an inheritance expression in the object specifier.

APPENDIX 1

NOTICE: The material in this appendix is an official part of SEMI E39 and was approved by full letter ballot procedures on March 16 and April 21, 2000 by the Japanese Regional Standards Committee.

A1-1 Overview of Object Terminology

A1-1.1 This section provides an introduction to the basic terminology for object models.

A1-1.2 A *model* is an abstraction of a problem, a real-world phenomena, things, etc. for the purpose of understanding it. A model typically is a simplification that omits nonessential details. *Examples: architectural scale models, behavioral state models*.

A1-1.3 An *object* is an entity (concept, abstraction, or "real world" thing) with a particular behavior and with associated properties (information, or attributes). An *object type* (class) refers to a group of objects that have common (1) properties (but not specific values), (2) behavior, (3) relationships with other objects, and (4) semantics (public interfaces). The term "object" may be used either to refer to a type of object or to a particular instance of an object type.[8] The notation used for diagrams of objects used in this document is described in Section A1-2.

A1-1.4 An *object model* is a static graphic model of objects to show structure — the identity of objects, their attributes and operations, and their relationships with one another.

A1-1.5 An *instance* of an object is an instantiation of an object type. For example, a specific installation of equipment is an instance of the object type "Equipment."

A1-1.6 Objects have items of associated information called *attributes*. For example, for an object "Equipment," attributes of interest include the manufacturer, model, serial number, and a logical user-assigned name (nickname). Attributes used to uniquely identify a particular instance of an object type is called an identifier. An object may have more than one *identifier*. Also, a set of more than one attribute may be used as an identifier. In the example of "Equipment," name could be used as an identifier, and the combination of manufacturer and serial number also could be an identifier.

A1-1.7 Objects have *operations* that may be applied to or by an object type. Operations are functions or transformations that are either performed by, or on, an object. Operations of interest to Object Services are "get (read) attributes," "set (write) attributes," and "get type." A *service* provides a *service user* with an interface to the functionality of the operation.

A1-1.8 A high-level model may show a particular type of object with certain operations, while a more detailed model shows that a second type of object actually performs one or more of the operations on the first object. As an example, a model might show a process program object with the operation "delete," to show that the "delete" operation is inherently associated with the process program. A more detailed model might show that a "process program manager" actually performs the operation of deleting a process program.

A1-1.9 A *method* is an implementation of an operation (e.g., the software code that performs the operation).

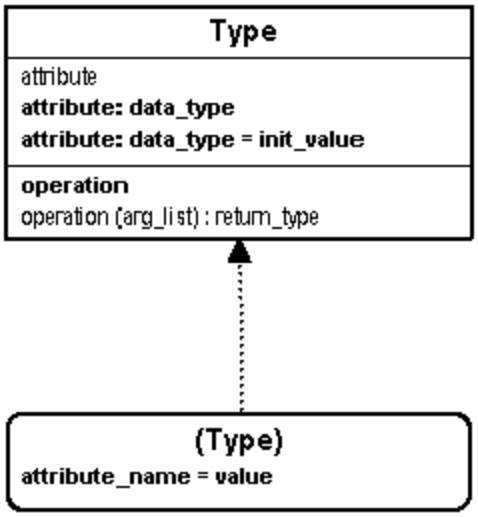

Figure A1-1
Object Type and Instance

8 The term "object" is used by some authors to only refer to instances and by other authors as a synonym for object type.

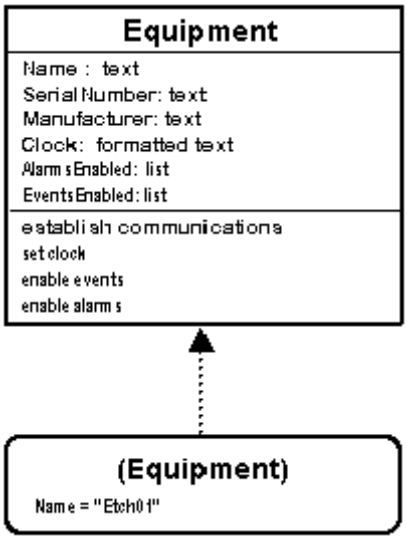

Figure A1-2
Example of Object Type and Instance

A1-2 Object Modeling Technique (OMT) Notation

A1-2.1 The material contained in this section is provided as a reference for auxiliary information.

A1-2.1.1 Object Modeling Technique (OMT) is a graphical notation for models of objects that is useful for analyzing a wide variety of problems, in all phases of software design and development, and in preparing documentation. OMT was developed by Rumbaugh, Blaha, et al., in *Object Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, ©1991. OMT has been adopted by different standards to describe requirements in terms of objects and relationships between objects. The purpose of this document is to provide a description of OMT notation as a reference for such SEMI standards and other SEMI documents.

A1-2.2 *Basic Notation* — Figure A1-1 illustrates notation for object type, showing object type, object attributes, object operations, object instance, and the relationship between an object type and instance. Figure A1-2 provides an example.

A1-2.2.1 Object *types* in OMT are shown as rectangles. The rectangle may be further subdivided into two or three parts. The object type always appears at the top. Object *attributes* are shown in a second part of the rectangle. *Operations* are shown in a third part of the rectangle. High-level models often omit the operations section and sometimes omit the attributes section.

A1-2.2.2 The *name* of an attribute may be followed by additional details, such as data type and initial value.

A1-2.2.3 *Instances* of objects are shown as rounded rectangles. The *instantiation* relationship is shown by a dotted arrow from an instance of an object type to the type.

A1-2-2.4 The example for an object type "Equipment" is shown in Figure A1-2; here "Etch01" is a specific real-world installation of type "Equipment."

A1-2.3 *Associations* — An association describes a set of potential bi-directional relationships between instances of objects. An association uses a formal structure and semantics.

A1-2.3.1 An association shows a specific multiplicity at both ends of the relationship. For example, a single factory manufacturing system may be associated with many equipment (one-to-many). Possible multiplicities of association types are shown in Figure A1-3.

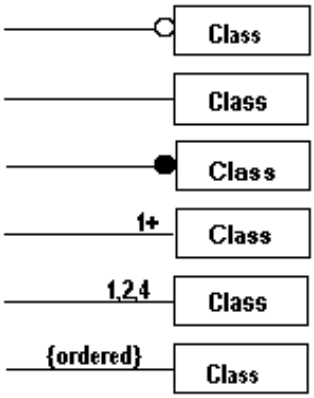

Figure A1-3
Multiplicities of Associations

A1-2.3.2 Figure A1-4 shows an example of a one-to-one relationship. The name of the association (using a verb) appears above the line connecting a pair of objects and describes the relationship that the object to the left (or above) has to the object to the right (or below).

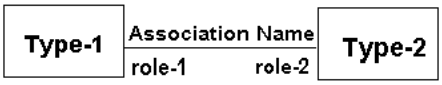

Figure A1-4
Association of Objects

A1-2.3.3 The name of the role of each object within the relationship may be placed beneath the line and near that object. Role names are nouns. In the example above, the roles might be "supervisor" and "subordinate." Role names are optional.

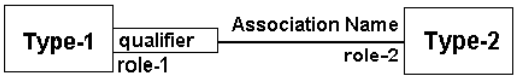

Figure A1-5
Qualified Association

A1-2.3.4 A *qualified association*, shown in Figure A1-5, uses a *qualifier* to reference an object. *Qualifiers* may be shown to specify the identifier used by one object to associate with instances of the other object. If Type-1 object is "Factory Host" and Type-2 object is "Equipment," a user-assigned "name" attribute may be used as a qualifier for the equipment.

A1-2.3.5 Figure A1-6 illustrates a *link object*, where link attributes are placed in an object box that is attached to the association with a loop. A *link object* may be used to show attributes that are dependent on the association between instances of one object type to instances of another type.

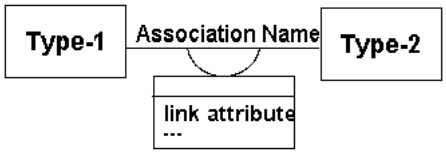

Figure A1-6
Link Object

A1-2.3.6 Figure A1-7 gives an example of a link object for the association "Authorized On" between an "Authorized User" and "Equipment."

Figure A1-7
Example of a Link Object

A1-3 Generalization and Inheritance

A1-3.1 Figure A1-8 shows the concept of *generalization*. Generalization categorizes a set of object types and allows the abstraction of their common features into a supertype, with refinements of the *supertype* shown as *subtypes*. The subtype may be thought of as a specialization of the supertype. For example, "vehicle" is a supertype with many subtypes, including "aircraft," "automobile, "cart," and "robot."

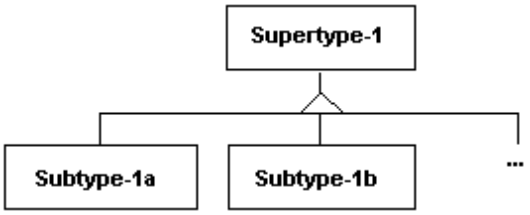

Figure A1-8
Non-Overlapping Subtypes

A1-3.1.1 An instance of a subtype is also an instance of its supertype. Attributes and operations of the supertype are inherited by all the subtypes. That is, the subtype has all of the same attributes and operations of the supertype, and in addition it adds attributes and operations of its own.

A1-3.1.2 The OMT notation for the generalization (supertype/subtype) relationship is shown by a solid triangle. White triangles indicate that the subtypes are *non-overlapping*; an instance of one subtype may not be an instance of another subtype.

A1-3.1.3 The method for a subtype of an object might be different from that of the supertype or from a different subtype, perhaps due to more specialized knowledge that allowed a more efficient implementation for the subtype. As an example, a Geometrical Figure object type might have the operation "draw." The methods for subtypes of "circle" and "rectangle" would differ, however.

A1-3.1.4 Figure A1-10 gives an example of an "agent" as a generalization of various types of active entities that might be found in a factory, and "equipment" as a generalization of specific kinds of equipment.

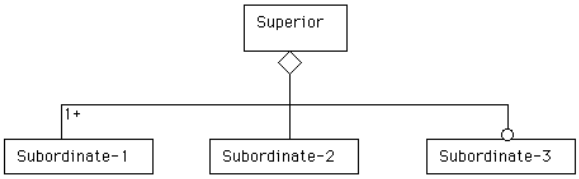

Figure A1-9
Aggregation with Two Component Types

A1-3.2 *Object Composition and Containment* — Figures A1-9 and A1-10 illustrate the diamond notation used to show aggregation. An aggregation object is also referred to as an assembly in object-oriented literature.

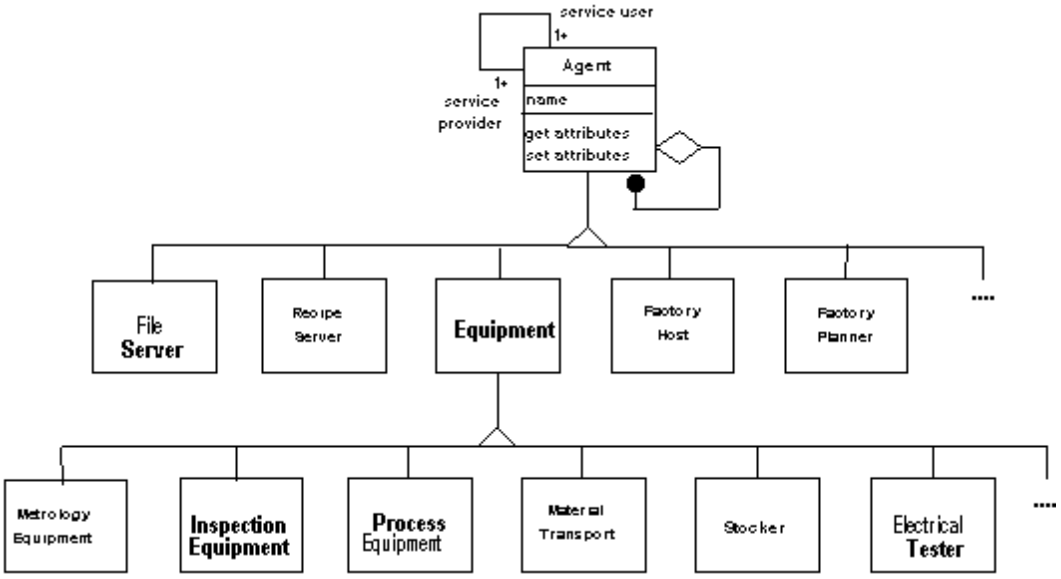

Figure A1-10
An Example of Generalization

A1-3.2.1 An aggregation object is composed of other objects called the *components* of the aggregation. Components may be of the same type as the aggregation (illustrated in Figure A1-10), or they may be of one or more different types.

A1-3.2.2 In Figure A1-9, the "Superior" object type is composed of one or more objects of type "Subordinate-1," of exactly one component of object type "Subordinate-2," and of zero or one objects of type "Subordinate-3."

A1-3.2.3 In addition, an aggregate also may be optionally composed of other objects of the same type, as Figure A1-10 illustrates. The "agent" in this figure is a very general supertype of a factory system. Additional subtypes of agents, not shown in the figure, could include cells, clusters, and cluster modules. Cells, with equipment as components, and clusters, with modules as components, are examples of types of agents that are aggregates composed of other subtypes of agent.

SEMI E39.1-0703
SECS-II PROTOCOL FOR OBJECT SERVICES STANDARD (OSS)

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 American Regional Standards Committee on March 12, 2003. Initially available at www.semi.org May 2003; to be published July 2003. Originally published in 1995; previously published November 2002.

1 Purpose

1.1 This document maps the services and data of the parent document to SECS-II streams and functions and data definitions.

2 Scope

2.1 This document applies to all implementations of Object Services that use the SECS-II message protocol (SEMI E5).

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 Referenced Documents

SEMI E5 — SEMI Equipment Communications Standard 2 Message Content (SECS-II)

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

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

4 Mapping of Object Services

4.1 Table 1 shows the specific SECS-II streams and functions that shall be used for SECS-II implementations for the object services defined in OSS.

Table 1 Services Mapping Table

Message Name	*Stream, Function*	*SECS-II Name*
GetAttr.req	S14,F1	GetAttr Request
GetAttr.rsp	S14,F2	GetAttr Data
SetAttr.req	S14,F3	SetAttr Request
SetAttr.rsp	S14,F4	SetAttr Data
GetType.req	S14,F5	GetType Request
GetType.rsp	S14,F6	GetType Data
GetAttrName.req	S14,F7	GetAttrName Request
GetAttrName.rsp	S14,F8	GetAttrName Data
Create.req	S14,F9	Create Object Request
Create.rsp	S14,F10	Create Object Acknowledge
Delete.req	S14,F11	Delete Object Request
Delete.rsp	S14,F12	Delete Object Acknowledge
Attach.req	S14,F13	Object Attach Request
Attach.rsp	S14,F14	Object Attach Acknowledge
AttachSetAttr.req	S14,F15	Attached Object Action Request
AttachSetAttr.rsp	S14,F16	Attached Object Action Acknowledge
AttachSupervisedObject.req	S14,F17	Supervised Object Action Request
AttachSupervisedObject.rsp	S14,F18	Supervised Object Action Acknowledge
Detach.req	S14,F15	Attached Object Action Request
Detach.rsp	S14,F16	Attached Object Action Acknowledge
DetachSupervisedObject.req	S14,F17	Supervised Object Action Request
DetachSupervisedObject.rsp	S14,F18	Supervised Object Action Acknowledge
Reattach.req	S14,F13	Object Attach Request

Message Name	*Stream, Function*	*SECS-II Name*
Reattach.rsp	S14,F14	Object Attach Acknowledge
GetServiceNames.req	S14,F25	Get Service Name Request
GetServiceNames.rsp	S14,F26	Get Service Name Data
GetServiceParameters.req	S14,F27	Get Service Parameter Name Request
GetServiceParameters.rsp	S14,F28	Get Service Parameter Name Data
ObjectAction.req	S14,F19	Generic Service Request
ObjectAction.rsp	S14,F20	Generic Service Acknowledge
ObjectActionCompletion.nfy	S14,F21	Generic Service Completion Information
ObjectActionCompletion.rsp	S14,F22	Generic Service Completion Acknowledge

5 Service Parameter Mapping

5.1 Table 2 shows the mapping between message parameters defined by OSS and data items defined by SECS-II. For parameters specified in the definitions of an OSS service, either the parameters themselves, or individual elements of complex parameters, map to a specific data item.

Table 2 Service Parameters Item Mapping Table

Parameter Name	*SECS-II Data Item*
AttachToken	OBJTOKEN
AttrData	ATTRVAL
AttrName	ATTRID
AttrReln	ATTRRELN
ErrCode	ERRCODE
ErrText	ERRTEXT
ObjAck	OBJACK
ObjectAction	SVCNAME
ObjectActionAcknowledgement	SVCACK
ObjectActionParameterRequest	L,2 1. <SPNAME> 2. <SPVAL>
ObjectActionParameterResult	L,2 1. <SPNAME> 2. <SPVAL>
ObjectActionReportStatus	DATAACK
ObjectActionStatus	L,2 1. <SVCACK> 2. Status
ObjID	OBJID
ObjSpec	OBJSPEC
ObjType	OBJTYPE
OperationID	OPID
ServiceParameterName	SPNAME
ServiceParameterValue	SPVAL

Parameter Name	*SECS-II Data Item*
Status	L,n 1. L,2 1.<ERRCODE$_1$> 2.<ERRTEXT$_1$> : n. L,2 1. <ERRCODE$_n$> 2. <ERRTEXT$_n$>
TargetSpec	TARGETSPEC

5.2 *SECS-II Data Items Without Corresponding SEMI E94 Parameters*

5.2.1 Table 3 contains the SECS-II data items that do not correspond to SEMI E39's service parameters.

Table 3 Additional Data Item Requirements Table

Function	*SECS-II Data Item*
Used to satisfy SECS-II conventions for linking a multi-block inquiry with a subsequent multi-block message. Neither required nor specified by OSS.	DATAID
Used to inform receiver of total message length size for SECS-II multi-block conventions. Neither required nor specified by SEMI E39.	DATALENGTH
Used to satisfy SECS-II multi-block requirements. Neither required nor specified by OSS.	GRANT

6 Data Item Format Restrictions

ATTRID Format: 20

The ASCII version of ATTRID is restricted to the characters from 20_{16} through $7E_{16}$, excluding the following characters: the "greater than" symbol ">", the colon ":", the question mark "? ", the asterisk "*", and the tilde "~ ". The space character (20_{16}) may not be used as the first or last character. Maximum length is 40 characters.

OBJTYPE Format: 20

The ASCII version of OBJTYPE is restricted to the characters from 20_{16} through $7E_{16}$, excluding the following characters: the "greater than" symbol ">", the colon ":", the question mark "? ", the asterisk "*", and the tilde "~ ". The space character (20_{16}) may not be used as the first or last character. Maximum length is 40 characters.

OBJID Format: 20

The ASCII version of OBJID is restriced to the characters from 20_{16} through $7E_{16}$, excluding the following characters: the "greater than" symbol ">", the colon ":", the question mark "? ", the asterisk "*", and the tilde "~ ". The space character (20_{16}) may not be used as the first or last characer. The equipment shall allow the OBJID to be any length from 1 to 80 characters, inclusive. A zero length OBJID implies that no OBJID is specified.

SEMI E40-0705
STANDARD FOR PROCESSING MANAGEMENT

This standard was technically approved by the global Information & Control Committee. This edition was approved for publication by the global Audits and Reviews Subcommittee on April 7, 2005. It was available at www.semi.org in June 2005 and on CD-ROM in July 2005. Originally published in 1995; last published June 2003.

1 Purpose

1.1 Automated management and command of material processing in equipment is a crucial aspect enabling factory automation. This standard addresses the communications needs within the semiconductor manufacturing environment with respect to the processing of material in equipment.

1.2 This standard specifies the application of the appropriate processing to specified material received at the processing agent. It describes the concepts of material processing, the behavior of the equipment in relation to processing, and the messaging services which are needed to accomplish the task.

1.3 The communications services defined here enable standards-based interoperability of independent systems. They allow application software to be developed that can assume the existence of these services and allow software products to be developed which offer them.

1.4 Implementation of automated processing management will help eliminate misprocessing of material. The adoption of the standards described will greatly reduce the effort required to integrate compliant equipment components and reduce time to set up for processing. Compliance requires a minimal but specific set of standard services.

2 Scope

2.1 The scope of this standard is automated material processing based on discrete processing jobs. It provides the functionality required for process management for modules within a cluster tool. It may be applied to sub-systems of other multi-resource equipment, as well as to host control of many types of equipment.

2.2 This standard supports individual management of jobs for identical processing of material within a group and concurrent processing of independent groups. Where material contains other material (such as carriers containing wafers), processing may be specified in terms of either material type.

2.3 A simple tuning mechanism is provided for limited feedforward and feedback control between process steps. A method is defined for taking advantage of recipe variable parameters. This is not expected to satisfy all closed loop control requirements. Other mechanisms are anticipated with greater flexibility for late tuning and handling complex data.

2.4 This standard does not provide services for receiving material for processing, or disposing of it after processing is complete. Automation of material transfer is assumed to be provided through other services, such as those defined in applicable SEMI standards.

2.5 This standard presents a solution from the concepts and behavior down to the messaging services. It does not define the messaging protocol.

2.6 A messaging service includes the identification that a message shall be exchanged and a definition of the data which is contained in that message. It does not include information on the structure of the message, how the data is represented within the message, or how the message is exchanged. This additional information is contained within the message protocol.

2.7 The defined services may be applied to multiple protocols. Information on the mapping of processing management services to special protocols (e.g., SECS-II) are added as adjunct standards.

2.8 The services assume a communications environment in which a reliable connection has been established between the user of the services and the provider of the services. Establishing, maintaining, releasing a connection, and handling communication failures are beyond the scope of this standard.

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 Referenced Standards and Documents

3.1 *SEMI Standards*

SEMI E5 — SEMI Equipment Communications Standard 2 Message Content (SECS-II)

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

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

SEMI E53 — Event Reporting

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

4 Terminology

4.1 The following definitions are arranged in alphabetical order. Some definitions use terms defined elsewhere within this section. No references beyond this section should be necessary for a basic understanding of these terms.

4.2 *Definitions*

4.2.1 *agent* — an intelligent system within a factory that provides one or more service resources and uses the services of other agents. A generalization of host, equipment, cell, cluster, cluster module, station controller, and work station. Agents are associated with a physical system or a collection of physical systems, including computer platforms.

4.2.2 *form* — type of data representing information contained in an object attribute or service message parameter.

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

4.2.4 *post-conditioning* — activities performed by the processing resource after departure of the material being processed but related to the processing of that material (e.g., cleanup).

4.2.5 *pre-conditioning* — activities performed by the processing resource before arrival of the material being processed but related to the processing of that material.

4.2.6 *processing agent* — an intelligent system within a factory which is independently capable of providing manufacturing value added to material.

4.2.7 *processing resource* — an entity within a processing agent which provides the manufacturing value added to material.

4.2.8 *process job* — a material processing job for a processing resource specifying and tracking the processing to be applied to the material.

4.2.9 *recipe* — the pre-planned and reusable portion of the set of instructions, settings, and parameters under control of a processing resource that determines the processing environment seen by the material. Recipes may be subject to change between runs or processing cycles.

4.2.10 *recipe executor* — a component of a module that stores and executes recipes.

4.2.11 *recipe namespace* — a logical management domain with the responsibility for the storage and management of recipes. It ensures the uniqueness of recipe identifiers and provides services pertaining to recipes stored within that domain.

4.2.12 *service* — the set of messages and definition of the behavior of a service-provider that enables remote access to a particular functionality.

4.2.13 *service-provider* — the software control entity that is the provider of a particular functionality which may be accessible remotely.

4.2.14 *service-user* — the software control entity that is the user of any of the related services.

4.2.15 *supervisor* — an entity or entities having supervisory control responsibilities for one or more processing resource. It is the service-user of the processing management services.

4.2.16 *tuning* — specification of information which supplements the pre-defined recipe used to achieve the particular process goals.

4.3 *Data Type*

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

4.3.2 *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.

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

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

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

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

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

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

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

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

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

5 Conventions

5.1 *Harel State Model* — This document uses the Harel State Chart notation to describe the dynamic behavior of the objects defined. An overview of this notation is presented in an appendix of SEMI E30. The formal definition of this notation is presented in *Science of Computer Programming 8*, "Statecharts: A Visual Formalism for Complex Systems," by D. Harel, 1987.

5.1.1 The Harel notation does not include the concept of "creation" and "deletion" of state models to represent transient entities. The "job" described in this document is such an entity, where each new job created uses a copy of the same state model. In this document, an oval is used to denote the creation of an entity and also the deletion of that entity.

5.1.2 Transition tables are provided in conjunction with the state diagrams to describe explicitly the nature of each state transition. A transition contains columns for Transition #, Current State, Trigger, New State, Action(s). The "trigger" (column 3) for the transition occurs while in the "current" state. The "actions" (column 5) include a combination of (1) actions taken upon exit of the current state, (2) actions taken upon entry of the new state, and (3) actions taken which are most closely associated with the transition. No differentiation is made.

5.1.3 The state models included in this standard are a requirement for Processing Management compliance. A state model consists of a state model diagram, state definitions, and a state transition table. When using SEMI E30, E53 or similar style collection events, all state transitions in this standard, unless otherwise specified, shall correspond to collection events.

5.1.4 A state model represents the host's view of the equipment, and does not necessarily describe the internal equipment operation. When using collection events, all Processing Management state model transitions shall be mapped sequentially into the appropriate internal equipment collection events that satisfy the requirements of those

transitions. In certain implementations, the equipment may enter a state and have already satisfied all of the conditions required by the Processing Management state models for transition to another state. In the case, the equipment makes the required transition without any additional actions in this situation.

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*	*Requirement*	*Form*
The formal text name of the attribute.	Description of the information contained.	RO or RW	Y or N	(see below)

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

5.2.2 A 'Y' or 'N' in the requirement (Rqmt) 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 ¶4.1 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 used in a service, 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 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 take on, and any inter-relationships with other parameters.

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/Cnf*	*Description*
Parameter X	(see below)	(see below)	A description of the service.

5.3.3.1 The columns labeled Req/Ind and Rsp/Cnf 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/Cnf 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 value of the other parameter.
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 Overview

6.1 Processing management is concerned with the processing of material by a processing resource. Its principle function is to ensure that material delivered to the processing resource is processed with the correct recipe. It defines the services needed by a supervisor (service-user) to initiate and track processing of a particular material. It also defines commands which affect the processing operation.

6.1.1 The processing resource is the entity which adds manufacturing value to the material. It may take several forms, including the processing element of a cluster tool process module or the entity managing processing for a complete stand-alone equipment. The processing agent is considered to be the provider of the processing services.

6.1.2 Process management allows for pre-conditioning before material arrival and post-conditioning after material departure. A simple tuning mechanism provides support for limited feedforward and feedback control. The tuning, applied at process initiation, sets recipe variable parameters.

6.1.3 The services are fully defined in terms of the functionality provided by the processing agent (service-provider) and, as such, do not dictate the architecture of the supervisor (service-user).

6.1.4 This standard describes the concepts and processing model on which the communications are based, followed by the detailed behavioral model used. It then describes the standard object attributes and message services in detail.

6.2 *Compliance* — Compliance to this standard includes adherence to all stated requirements in this document where implemented. Standard services are to be used where related functionality is required. This includes defined message services and state models.

6.2.1 Some capabilities are not required to be supported for compliance, such as queuing, multiple concurrent jobs, material groups, manual start, pause/resume, and tuning. Required capabilities are indicated throughout the document and are also listed in the Fundamental Requirements section.

6.2.2 A processing agent shall provide the funda-mental requirements, plus the set of optional services, appropriate to achieve effective processing management for the particular hardware architecture and automated processing requirements.

7 Concepts

7.1 *Material Processing Model* — Processing management ensures that the appropriate processing is applied to a particular material by a processing resource through the definition of a process job. The process job provides a

widely applicable supervisory control capability for automated processing of material in equipment, irrespective of the particular process being used.

7.1.1 This standard assumes that, given the material and the recipe specification, the processing resource is capable of independently achieving the required processing objectives.

7.1.2 Processing management does not provide services for material movement, but the service-provider does need to coordinate its activities with regard to the receiving and sending of material, thereby maintaining system integrity.

7.2 *Process Job* — The process job is a dynamic ob-ject specified by the process supervisor (service-user) to effect material processing by the processing resource. The high-level job contains all the information required by the processing resource to achieve processing of the material, once it arrives, without further intervention by the supervisor.

7.2.1 The process job encompasses up to four sequential phases:

- processing resource pre-conditioning before material arrival,
- material and processing resource preparation for processing,
- material processing, and
- processing resource post-conditioning after material departure.

7.2.2 The material processing phase is the only phase in which the material is altered and is the only required phase.

7.2.3 This standard specifies services for the creation, control (pausing, aborting, etc.) and tracking of the process job. It does not define the low-level control of processing since this is application-dependent. The processing resource performing a process job is responsible for doing whatever is appropriate to achieve its processing objectives, as specified by the recipe and tuning parameters.

7.2.4 The material specified in a process job may be the actual single material elements to be processed or a container, such as in the case of a cassette of wafers.

7.2.5 The process job lifecycle may extend beyond the active processing of the material. It may exist from before material arrival, through setup and processing, and until after material departure. This allows for material processing-related pre-conditioning of the processing resource before the material is received and for processing resource post-conditioning (e.g., cleanup) after material is sent. Pre-conditioning and post-conditioning support is not a fundamental requirement.

7.2.6 The processing resource may provide process job queuing in order to offer flexibility in systems where work is pre-scheduled or the order of material arrival is unknown. Queuing is the acceptance of multiple process jobs in advance of performing the processing activities. Queuing is generally needed to support more complex systems requiring concurrent and consecutive jobs (see below). The jobs are listed in the queue in the order created. Execution order may be significant, such as consecutive jobs on the same material. Queuing is not a fundamental requirement.

7.3 *Relation to Material Movement* — Processing Management does not provide services for receiving material into the processing agent domain for processing or sending the material away after processing is complete.

7.3.1 Processing depends on the presence of the material, and material departure depends on process completion. There is also an interdependency requiring synchronization with material movement if processing resource pre-conditioning and post-conditioning are applied. The equipment is responsible for maintaining integrity between material transfer and processing.

7.3.2 Material movement management is outside the scope of this standard but may be achieved using applicable SEMI standards.

7.4 *Processing Description* — The description of the processing to be applied by the Process Job is crucial. The description may be supplied in the form of a Process Recipe (see SEMI E42) or a Process Program (see SEMI E30). This specification will define mes-sages referencing only Process Recipes. Where there are special considerations for using Process Programs instead of a Process Recipe, they will be noted.

7.4.1 The process job includes a processing descrip-tion (a recipe or process program) identifier that shall be unique within the domain of the processing agent. The type and content of the processing description must be appropriate for the processing resource and the type of material.

7.4.2 Creation and management of recipes and process programs is outside the scope of this standard.

7.5 *Process Tuning* — Feedforward and feedback control between process steps is becoming increasingly important for process tuning in stabilizing processes, such as those which lack in-situ metrology, and demand increasing yields (or performance). The process tuning requirements differ considerably with application, and there is little consensus on any particular method. It is not the intention of this standard to provide compre-hensive support for process tuning, but rather to provide a simple mechanism which may be extended. Support for recipe tuning is not a fundamental requirement.

7.5.1 Processing management provides a mechanism for specification of the type of recipe method to be applied. Two methods are defined in the standard: RecipeID only, and RecipeID and Variables. User-defined methods may also be used, requiring all com-municating entities to have a common understanding of the particular definition and application requirements.

7.5.2 The RecipeID only method accepts the identifier of the recipe to be applied but no additional tuning parameters. The application recipe body is not precluded from defining any application tuning, but there is no standardized support.

7.5.3 The RecipeID and Variables method provides a simple process tuning mechanism at process job creation to support limited run to run feedforward and feedback control. It defines the VariableTuning method, which supplies a list of variable names and values in the process job create. This sets variables defined in the recipe. Each variable name shall be one of the exposed variable definitions supported in recipe management, and its value shall fall within the range specified in the variable definition.

7.5.4 Recipe parameter names (RecipeVarName) shall be specified using the nomenclature defined for ‘Object Specification’ (ObjSpec) in SEMI E39 (Object Services Standard) within the scope of SEMI E40 (Note: see OBJSPEC in SEMI E5). This use of the ObjSpec nomenclature is required to unambiguously identify parameters within some complex recipes (e.g., cluster tool process recipes). If the specification of a recipe parameter is unambiguous, then the form of the ObjSpec may be simplified to just the parameter name. Using the ObjSpec nomenclature for SEMI E40 recipe parameter names requires an object model to describe equipment recipe structure.

7.5.5 Figure 1 shows an example of a typical set of cluster tool hierarchical recipe relationships. A processing parameter might be specified through Sequence Step AB, Process Recipe BB, Process Step CB, to Process Parameter DB. In this example, the object specifier for Process Parameter DB would be:

“Sequence:Erma>SequenceStep:AB>ProcessRecipe:
BB>ProcessStep:CB>ProcessParameter:DB>”

or

“Erma>AB>BB>CB>DB>”

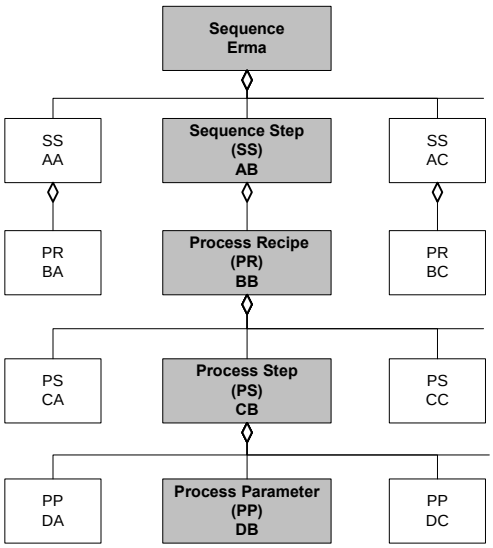

Figure 1
Cluster Tool Sequence Recipe Hierarchical Relationships

7.6 *Processing Material Groups* — Many equipment architectures require concurrent processing of groups of material. A single process job can control a group of material, with certain restrictions. All material in the group needs to be of the same type and processed identically. The processing of each material should be dependent on the arrival of the group. That is, the full material group shall be received by the equipment before processing begins and may not depart until all processing is complete. This ensures that the process job remains a simple logical control mechanism while maintaining robust control, good coordination with material movement, and effective material data tracking.

7.6.1 Two common examples of allowable material group processing are:

a) an equipment receiving a cassette of wafers to be processed identically. All the wafers in the group are received together, in the cassette. The processing of the wafers may or may not be simultaneous but can only proceed once the cassette has arrived. All wafer processing shall complete before the cassette may be removed. Note that this may alternatively be specified as a simple process job with the cassette as the material; and

b) a cluster tool batch process module processing wafers together. Wafers are received singly. Processing of the wafer group in the batch chamber begins when all the wafers specified in the process job have arrived. Wafers are sent after processing is complete.

7.7 *Concurrent Process Jobs* — Concurrent process jobs is the situation where multiple jobs are active (not queued) at the same time. A single process job is not always appropriate for concurrent processing of multiple material. In such cases, multiple concurrent jobs shall be supported by the processing resource. Support of concurrent process jobs is not a fundamental requirement.

7.7.1 An example of concurrent wafer processing which may not be achieved with a single process job is a carousel type cluster tool process module. The processing of a single wafer is not dependent on the arrival of the group. In this case, concurrent process jobs would be created, one for each wafer, even if they were to be processed identically. This allows effective control and tracking of the process jobs.

7.7.2 Processing management considers concurrent process jobs to be logically independent of one other. They are distinguished by unique job identifiers. Concurrent jobs may not apply to the same material because the processing resource may not have more than one active process job associated with a particular material.

7.7.3 There may be interdependencies between concurrent jobs due to resource availability and equipment hardware architecture.

7.8 *Consecutive Process Jobs* — Consecutive process jobs is a situation where multiple process jobs are applied to material while it is in the processing resource. The process jobs requested on the same material are maintained in the order received, and a subsequent job becomes active once the previous one has completed material processing.

7.8.1 A process job normally specifies all the processing to be applied to the material during a single visit to the processing resource. For example, a process job for a cluster tool would specify all the processing in the multiple sequential steps in the various process modules of the tool. Certain situations may require application of subsequent process jobs to material without it leaving the processing resource.

7.8.2 Processing management requires that a subsequent process job on the same material does not interrupt the previous process job processing. The previous process job terminates immediately once it completes active material processing, even though the material has not left the processing resource, and it is superceded by the subsequent job. The material becomes associated with the superceding process job. This allows sequential processing and maintains a single active association between material and process job.

7.9 *Process Job without Material* — This standard is primarily intended for the management of material processing. However, it permits application of a process job to a processing resource which contains no processing material. This may be used to achieve processing resource conditioning which is not related to a specific material.

7.9.1 The process job has the normal control characteristics, except that it has no dependency on material arrival and terminates at the end of active processing. Support of process jobs without material is not a fundamental requirement.

8 Behavior

8.1 This section provides a high-level definition of the communications between the supervisor and the processing resource needed to achieve material processing. It does not define the message detail, concentrating instead on the concepts. The message detail is addressed in §10, Messaging Services.

8.2 *Process Job Communications*

8.2.1 *Process Job Control Messaging* — The control message flow for normal operation is presented in Figure 1. The arrows represent significant information exchange.

8.2.1.1 A detailed description of each message used in normal operation follows.

8.2.1.2 *PR Job Create* — The supervisor requests that the processing resource perform the specified process job. This request may be acted upon immediately, or queued for later execution if the processing resource is busy or the order of material arrival unknown. If the processing resource does not support queuing or the queue is full, the request may be rejected. The request shall supply a process specification in which the supervisor supplies such information as:

- identification of the material to be processed,
- the recipe defining the processing, and
- whether processing will be started manually (optional, normal operation is automatic).

8.2.1.2.1 Upon receipt of the PR Job Create request and before acknowledging, the processing resource checks the process specifications to ensure that they are valid (i.e., that the specified parameters (recipe, material, manual start) are sufficient) and have legal values for the capabilities of the processing resource. Depending on its ability to queue jobs, it may also check such dynamic information as availability of the processing resource to receive the material or presence of correct material, etc., to determine whether to accept or reject.

8.2.1.2.2 The processing resource sets the process start attribute if automatic start is requested (normal operation).

8.2.1.3 *PR Job Create Acknowledge* — The processing resource informs the supervisor that the requested job is accepted or rejected, and if rejected, supplies error codes and textual reasons for the failure.

8.2.1.4 *PR Job Setup* — The processing resource reports that the process job is active and setting up for process. It may have been on the queue or have just been created. If the material is not already present, the processing resource performs any pre-conditioning required then awaits material arrival. Upon arrival of all the material, it prepares for processing and automatically initiates processing if the process start attribute is set.

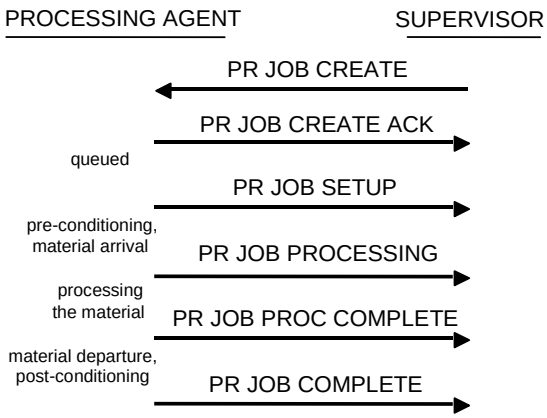

Figure 2
Process Job Message Flow

8.2.1.5 *PR Job Processing* — The processing resource reports that material processing has commenced.

8.2.1.6 *PR Job Processing Complete* — The processing resource reports that material processing is completed and that the material is available for removal.

8.2.1.7 *PR Job Complete* — The processing resource declares the process job to be complete once it has completed processing the material, the material has departed, and any required processing resource post-conditioning has completed. This message is also used when a process job ends abnormally. The message provides information on the success or failure of the processing and, if failed, supplies error codes and textual reasons for the failure.

8.2.2 *Process Job Informational Events* — There are process job related events which may be of significance to the service-user. These are designated as "collection events" and shall be available for event reporting. This section describes those collection events and shows how they fit into the chronology of a process job. These collection events, as specified by the PRJobEvent service, shall be implemented per definition of one of the following standards, SEMI E30, SEMI E40, SEMI E53 or similar style events, as required by the service-user. The equipment may also optionally implement the collection events per the remaining standards. However, the service-user shall only utilize one of the standards for collection event implementations. The selection mechanism by the service-user is equipment specific. For example, it may be part of an equipment power-on process or via an equipment specific equipment constant.

8.2.2.1 If SEMI E30, SEMI E53 or similar style collection events are used for PRJobEvent, the equipment may also implement equivalent events for the process job milestones as defined by the PRJobAlert service.

8.2.2.2 Collection event messages provide valuable information but are not strictly required to perform material processing. Therefore, it is expected that some message protocol implementations will provide a method by which the service-user may disable those events which are not needed in a particular implementation. For SEMI E40 style events, the activation for disabling the events is user specific. For SEMI E30, SEMI E53 or similar style events, the activation is as defined in those standards.

8.2.2.3 *PR Job Waiting for Material* — The processing resource reports that all required pre-conditioning has completed and that it cannot proceed until the process job material arrives. It is considered to be awaiting material arrival if it is not aware of activities in progress with the aim of receiving all or part of the material. This event may be generated only during process job setup. This event requires, at a minimum, PRJobID and Timestamp as its data.

8.2.2.4 *PR Job State Change* — The processing resource reports that it has changed state. All state transitions in the state model Figure 4 shall trigger this collection event. These events require, at a minimum, variables PRJobID, PRJobState, and TimeStamp (SEMI E30).

8.2.3 *Process Job Extended Messaging* — In this section, the extended messaging of Abort/Stop/Cancel, Pause/Resume and manual Start Process is added to the normal messaging described above. The only extended functionality required to be supported in processing management is Abort.

8.2.3.1 A detailed description of each message used in extended operation follows.

8.2.3.2 *PR Job Abort* — The supervisor may command the processing resource to abort a process job at any time. The goal of the abort command is to end the process job activities as quickly as possible. This includes halting all processing of material in progress, which may result in an unknown material condition. Abort is intended for use when serious problems are detected and further damage needs to be prevented. The abort command terminates the process job. In many cases, error recovery may be required before normal operation may continue and subsequent jobs can be executed. For processing equipment a part of the error recovery procedure may require the removal of substrates belonging to the aborted process job that still reside in the equipment. This is determined by the processing agent, which may use applicable service standards to handle the exception.

8.2.3.2.1 The abort command takes precedence over the stop, cancel, and pause commands. If the specified process job is queued, the abort command acts identically to a cancel command.

8.2.3.3 *PR Job Stop* — The supervisor may command the processing resource to stop a process job at any time. The stop command terminates the job in an orderly manner. The object of the stop command is to cease the current activity at the next safe, convenient point, preserving material integrity. In the situation of processing equipment, this convenient point may require that all related substrates are sent to their output destination. This implies that each material is either processed as specified in the recipe or not at all. As stop terminates the job, a new process job is needed to continue processing the material in the processing resource. If restart of the same job is needed, pause and resume should be used instead of stop. A new job is required if additional processing is needed after a job is stopped.

8.2.3.3.1 If the specified process job is queued, the stop command acts identically to a cancel command.

8.2.3.4 *PR Job Cancel* — The supervisor may cancel a process job which has not yet become active (e.g., a job which is queued). Cancel is used when the supervisor would like to remove a process job — to reschedule, for example — but does not want to affect process job activities already in-progress. A cancelled job is removed from the queue and ceases to exist. No physical action is associated with canceling a process job. If the specified process job is active, the processing resource shall reject this command.

8.2.3.5 *PR Job Pause* — The supervisor may issue a command to pause a process job at any time. A pause command shall cause the processing resource to continue to the first safe, continual, pausing place and then cease activity. The activity may cease only at points that allow for resumption of the activity (see the resume command) such that material integrity is maintained and the processing goals are accomplished. Note that a paused process job may be aborted or stopped as an alternative to the resume command. In this case, the stop command may cause the equivalent of a resume in order to ensure material integrity (either fully processed or not processed at all) upon termination of the process job.

8.2.3.6 *PR Job Resume* — The resume command is used to continue a previously paused process job activity.

8.2.3.7 *PR Job Create* — The PR Job Create request as previously described is for normal automatic process job operation. If the process start attribute is not set, the processing resource waits for a manual start from the supervisor before processing the material. The control message flow for manual start is presented in Figure 3.

8.2.3.8 *PR Job Waiting for Start* — The processing resource is ready to process once the material has arrived and has been prepared for processing. The processing resource reports that it is ready to process and is waiting for start, if the process job PRProcessStart attribute is not set. The PRProcessStart attribute is not set if the job is defined to start manually and the start command has not yet been received. The WAITING FOR START state shall be a safe condition which maintains material integrity. If the process job is stopped or aborted while in this state, the material will not have been altered.

8.2.3.9 *PR Job Start Process* — The supervisor which has defined a process job to start manually issues a PR Job Start Process to allow processing of the material to proceed when the processing resource is ready. The start may be issued at any time after process job creation. On receiving the start command, the processing resource sets the process start attribute and starts processing if it is already in the WAITING FOR START state.

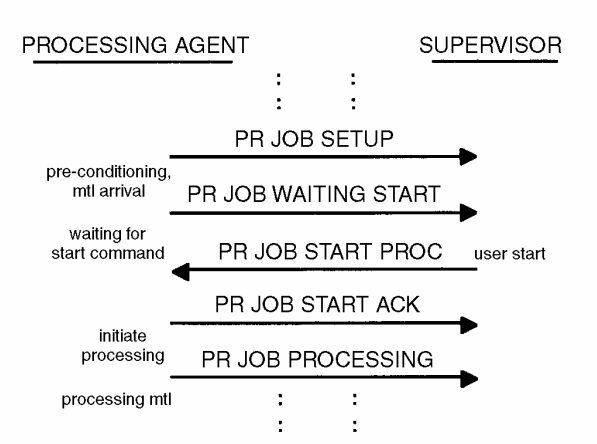

Figure 3
Manual Start Message Flow

8.2.3.10 *PR Job Start Acknowledge* — The processing resource responds to the supervisor that the requested start process is accepted or rejected and, if rejected, supplies errorcodes and textual reasons for failure.

8.3 *Process Job State Model* — The process job is a transient entity. It is created on request of the supervisor, executes, and then is deleted by the processing resource. The job usually spans the time period from shortly before material is physically delivered to the processing resource, through the processing, and until shortly after material is taken away.

8.3.1 *Process Job State Model Diagram* — Figure 4 shows the Process Job State Model diagram.

8.3.2 *Process Job State Descriptions* — The detailed state definitions follow.

8.3.2.1 *ABORTING (ACTIVE Substate)* — While the PR Job is in the ABORTING substate, the processing resource is performing an abort or an optional error recovery procedure. The abort procedure will cause immediate termination of the processing. It is the responsibility of the processing resource to cease physical activity as quickly as possible, having achieved a safe condition.

NOTE 1: For processing equipment the termination may have to be followed by an error recovery procedure with which remaining substrates can be brought to the output destination.

8.3.2.2 *ACTIVE* — ACTIVE is the parent state of all substates where the context of an active process job execution exists.

8.3.2.3 *EXECUTING (ACTIVE Substate)* — EXECUTING is the parent state of those substates that refer to the preparation and execution of a process job.

8.3.2.4 *SETTING UP (EXECUTING Substate)* — While the PR Job is in the SETTING UP substate, the processing resource performs pre-conditioning, awaits material arrival, and prepares for material processing. Pre-conditioning includes all operations in the processing resource, which are required by the recipe in advance of material arrival.

8.3.2.4.1 In cases where the material is already present, it is simply prepared for processing. If pre-conditioning (without material present) is required to achieve the processing goals specified by the recipe, the job fails and terminates.

8.3.2.5 *PAUSE (ACTIVE Substate)* — While the PR Job is in the PAUSE substate the processing resource is suspending or has suspended activity.