SEMI E97-0200A (Reapproved 0307)
PROVISIONAL SPECIFICATION FOR CIM FRAMEWORK GLOBAL DECLARATIONS AND ABSTRACT INTERFACES

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 November 21, 2006. It was available at www.semi.org in February 2007 and on CD-ROM in March 2007. Originally published February 2000.

NOTICE: This document was published twice during the February 2000 (0200) publishing cycle.

1 Purpose

1.1 This document defines the global declarations used by all other components of the CIM Framework and also specifies the common architecture patterns that serve to functionally integrate CIM Framework components. The material architecture defines functionality common to product management, durables management and consumables management components. The factory resource architecture defines relationships and common functionality of a variety of factory resources. The job architecture defines a factory-wide model for controlling factory jobs that drive a variety of manufacturing tasks. These specifications are separated into a distinct group to enable them to be specified once and then logically included or inherited wherever they are subsequently needed.

2 Scope

2.1 This specification provides the common interfaces required by Manufacturing Execution Systems to:

- Provide type definitions for common data structures to ensure consistent representation. These items include data types for common concepts such as coordinates, priorities, timestamps, and sequences of basic data types.
- Provide definitions for common exceptions used consistently throughout the CIM Framework.
- Provide the material architecture interfaces common to identifying, grouping, moving, locating and tracking material in the factory.
- Provide the factory resource architecture interfaces common to defining, organizing, tracking usage of and maintaining factory resources including equipment, sensors, durables, and people.
- Provide the job architecture interfaces common to creating, executing and managing work in the factory. The job architecture is specialized for material processing jobs, material transport jobs, resource maintenance jobs and factory jobs that drive product material through their process flows.

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

3 Limitations

3.1 *Provisional Status*

3.1.1 This specification is designated as provisional due to known areas that need to be completed. The following items summarize the deficiencies of the provisional specification to be addressed before a subsequent ballot to upgrade it to full standard status.

3.1.2 The specification uses the IDL typedef “any” in several places. While this usage provides flexibility, it can have the effect of reducing interoperability due to differences in interpretation of the value provided by separate implementations that interact through a standard interface. The “any” typedefs should be replaced with explicit data types prior to upgrade from Provisional to full Standard status.

3.1.3 The definition of interfaces for retrieval of history associated with CIM Framework objects may need to be added to abstract interfaces in this document after the complete specification for the history facility within the CIM Framework Factory Services Component.

3.1.4 The specification of CIM Framework states reported through published state change events is currently based on a type definition for an enumeration of state values. There may be alternate representations for states that are better able to capture the semantics of nested and parallel states. The state representation used for the CIM Framework will be reviewed and possibly changed before upgrade to full standard status.

INACTIVE

Documents Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

3.1.5 The Resource model defined in SEMI E81 includes several extensions that are not yet included in this specification. These extensions include composition of resources from other resources, capabilities associated with resources, and associations with tracking and maintenance functions for resources. These extensions will be addressed before upgrade to full standard status.

3.1.6 The interfaces specified for MESFactory, Area, and ComponentManager are included in SEMI E81 responsibilities for the Factory Component within Factory Operations. These interfaces will need to be moved to that component when it is considered in a future ballot.

4 Referenced Standards and Documents

4.1 *SEMI Standards*

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

SEMI E10 — Standard for Definition and Measurement of Equipment Reliability, Availability, and Maintainability (RAM)

SEMI E81 — Provisional Specification for CIM Framework Domain Architecture

4.2 *Other Standards*

UML Notation Guide, Version 1.1, document number ad/97-08-05, Object Management Group[1]

ISO/IEC International Standard 14750 (also ITU-T Recommendation X.920) — Information Technology – Open Distributed Processing – Interface Definition Language [2]

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

5 Terminology

5.1 *Definitions*

5.1.1 *abstract interface* — an interface specified only for inheritance rather than for implementation in order to standardize common features shared by all specializations of the interface.

6 Requirements

6.1 *Global Type Definitions*

6.1.1 This section describes data type definitions and exceptions that are global in nature. By "global" it is meant that they are designed to be used by any component within the CIM Framework. This specification does not address how they are implemented within a CIM Framework conformant application; that level of detail is within the realm of the development effort.

6.1.2 Global type definitions are specified as IDL declarations which may be referenced by any CIM Framework interface. The CIM Framework uses the keyword *typedef* to define aliases for basic object types, but with additional implied semantics (e.g., the *units* typedef defines a *string*, whose contents conform to definitions found in SEMI E5). The keyword *struct* begins the type declaration for record structures composed of a collection of named and typed values. The third form of type definition is an enumerated type beginning with the reserved word *enum*. Enumerated types are used to declare a list of tokens that can be used as values of that type. Enumerated types are mainly used to denote the states of an object for communication in CIM Framework events. Finally, collections of values are declared with the keyword *sequence*. This kind of type definition may or may not imply significance to the ordering of the members in the sequence. Where no ordering constraint is mentioned, the elements of the *sequence* are not assumed to be in any meaningful order.

NOTE 1: In the following definitions, "//" or "/* .... */" delimits a comment.

NOTE TO THE READER: The comments in the following sections (described in NOTE 1, above) appear to immediately precede, rather than follow, the items they discuss. — SEMI Staff

6.1.3 All CIM Framework specifications will be declared within the context of the CIMFW module which spans all of the components of the CIM Framework. Within the CIMFW module, the global type definitions will be declared within a second-level module called Global.

---

1 UML Notation Guide v1.1 is available to the general public at http://www.omg.org/cgi-bin/doclist.pl, +1-508-820 4300, Object Management Group, Inc. , Framingham Corporate Center, 492 Old Connecticut Path, Framingham, MA 01701.

2 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: http://www.iso.ch

INACTIVE

Documents
Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

Module: Global

```
typedef string Identifier;

typedef unsigned long Flags;

struct NamedValue {
        Identifier name;
        any argument;
        long len;
        Flags arg_modes;
};

typedef NamedValue NameValue;

typedef sequence <NamedValue> NameValueSequence;

typedef string PropertyName;

struct Property {
        PropertyName property_name;
        any property_value;
};

typedef sequence <Property> Properties;
```

/* This type definition represents units for factory parameters, measurements. etc., and conforms to the SEMI E5 standard for representation of units. In that standard, the string contains a code representing a value of the units. For example, “ns” would mean nano-seconds; “A” for ampere; and “wfr” for wafer. */

```
typedef string Unit;

typedef string Units;
```

/* This type definition represents a sequence of *string* values. */

```
typedef sequence <string> StringSequence;
```

/* This type definition represents a sequence of *any* values. */

```
typedef sequence <any> AnySequence;
```

/* This type definition represents a sequence of *long* values. */

```
typedef sequence <long> LongSequence;
```

/* This enumerated type identifies event priorities and is used in each event definition. */

```
enum PriorityOfEvent {
        Low,
        Medium,
        High,
        Alarm };
```

/* This enumerated type identifies the lifecycle states that an object may go through. It is used in event notifications of state changes. */

```
enum LifecycleState {
        Undefined,
        Created,
        Deleted,
        Moved,
        Copied };
```

/* This enumerated type identifies the states of objects that can be reserved (Lot, Durable and Machine). It is used in event notifications of state changes. */

```
enum ReservationState {
        UndefinedReservationState,
        Reserved,
        UnReserved };
```

INACTIVE

Documents Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

/* This enumerated type represents the SEMI E10 states for Machines and Support Resources. It is used in event notifications of state changes. */

```
enum E10State {
        E10Productive,
        E10Standby,
        E10Engineering,
        E10ScheduledDowntime,
        E10UnscheduledDowntime,
        E10NonscheduledTime };
```

/* *TimeT* is a *ulonglong* value (64 bits) that represents the number of 100 nanosecond increments that have passed since a base time (October 15, 1582 at 00:00, the Universal Time Representation which refers to time in Greenwich Mean Time). The specification for *TimeT* is: */

```
struct ulonglong {
        unsigned long low;
        unsigned long high;
};
```

```
typedef ulonglong TimeT;
```

/* *TimeStamp* is mapped to the data type of *TimeT*. */

```
typedef TimeT TimeStamp;
```

```
typedef sequence <TimeStamp> TimeStampSequence;
```

/* The notion of a specific interval of time denoting a start time and an end time is represented as a struct called *IntervalT*. */

```
struct IntervalT {
        TimeT lower_bound ;
        TimeT upper_bound ;
};
```

/* *TimeWindow* is mapped to the data type *IntervalT*. */

```
typedef IntervalT TimeWindow;
```

/* Duration is mapped to the datatype TimeT. */

```
typedef TimeT Duration;
```

/* This structure is for the representation of a single schedule instance. It should be noted that “EndTime” should never proceed “StartTime.” */

```
struct ResourceSchedule {
        TimeStamp plannedStartTime;
        TimeStamp plannedEndTime;
        TimeStamp actualStartTime;
        TimeStamp actualEndTime;
};
```

/* The definition of a sequence of ResourceSchedules. This sequence is ordered in increasing time order and that order must be maintained in any manipulation of the sequence. */

```
typedef sequence <ResourceSchedule> ResourceScheduleSequence;
```

6.2 *Global Exception Declarations*

6.2.1 This section describes the standard CIM Framework exceptions that may be thrown by operations in any component.

Module: Global

/* This signal is raised when a lookup or find fails. */

```
exception NotFoundSignal { string errorMessage; };
```

INACTIVE

Documents
Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

/* This signal is raised when an add fails because an object already exists with the given identifier. Interfaces may also define and raise a <ObjectType>DuplicateSignal. */

```
exception DuplicateIdentifierSignal {
        string errorMessage;
        string duplicateIdentifier; };
```

/*This signal is raised when an invalid state transition request is made of an object. */

```
exception InvalidStateTransitionSignal {
        string errorMessage; };
```

/*This signal is raised when a “set” attribute contains a value out of range. */

```
exception SetValueOutOfRangeSignal {
        string errorMessage; };
```

/* This signal is raised when an incorrect TimePeriod is used. */

```
exception TimePeriodInvalidSignal {
        string errorMessage; };
```

/* This signal is raised when a Property name is not valid. */

```
exception InvalidPropertyNameSignal {};
```

/* This signal is raised when a a Property with this name is not defined. */

```
exception PropertyNotFoundSignal {};
```

/* This signal is raised when a Property is not supported. */

```
exception UnsupportedPropertySignal {};
```

/* This signal is raised when a Property is read-only and cannot be set. */

```
exception ReadOnlyPropertySignal {};
```

/* This signal is raised when no other defined signal matches the error condition. */

```
exception FrameworkErrorSignal {
        string errorMessage;
        unsigned long errorCode;
        any errorInformation; };
```

/* Definition of fields for FrameworkErrorSignal:

errorMessage — is a text field representing a description of the circumstances of the exception for use by developers in debugging the exception.

errorCode — is a numeric field representing the code for the given exception.

errorInformation — is any further debugging information related to the circumstances of the exception.

The errorCode has certain reserved values that are defined and standardized in the CIM Framework.

- 0000–0999 — reserved for the CIM Framework.
- 1000–1999 — reserved for extensions to the CIM Framework.
- 2000–2999 — reserved for specializations of CIM Framework interfaces.
- 3000–maximum — reserved for implementers. /*

/* This errorCode should be used for any operation where the supplier has chosen to not provide implementation, but needs to communicate to the user that nothing has happened as a result of this operation invocation. */

```
const unsigned long NOT_IMPLEMENTED = 0;
```

/* This errorCode should be used for any operation where the supplier assumes a specialization will implement this operation. If this exception is received, the user will realize that an interface has not been properly specialized. */

```
const unsigned long IMPLEMENTED_BY_SUBCLASS = 1;
```

INACTIVE

Documents Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

/* This errorCode should be used for any operation where an unknown exception has been caught by the implementation and, rather than crashing, the implementation can map the "unknown" exception into this known exception. This probably does not aid in program debugging but does prevent program crashing. */

```
const unsigned long UNKNOWN_EXCEPTION = 2;
```

/* This errorCode should be used for any invocation where some unknown error has occurred that left the server object in an ambiguous state. */

```
const unsigned long COMPLETION_UNKNOWN = 3;
```

6.3 *Abstract Interface Type Definitions*

6.3.1 *Referenced Declarations*

6.3.1.1 The following declarations are not part of this specification, but are required for reference by Abstract Interface elements. These referenced declarations are defined in separate documents but are noted here as dependencies that appear in IDL compilations.

Module: EquipmentTracking

Interface: Machine

```
module EquipmentTracking {
interface Machine {}; // Stub
typedef sequence <Machine> MachineSequence;
exception MachineDuplicateSignal { };
exception MachineNotAssignedSignal { };
exception MachineRemovalFailedSignal { };
}; // module EquipmentTracking
```

Module: Labor

Interface: Person

```
module Labor {
interface Person {}; // Stub
typedef sequence <Person> PersonSequence;
exception PersonDuplicateSignal { };
exception PersonNotAssignedSignal { };
exception PersonRemovalFailedSignal { };
}; // module Labor
```

6.3.2 *Abstract Interface Declarations*

Module: AbstractIF

/* The following IDL interfaces will be fully defined in the sections below. They are declared here as forward references to support the sequence typedefs. */

```
interface Resource;
interface Material;
interface MaterialGroup;
interface JobSupervisor;
interface Job;
interface JobRequestor;
```

/* Type definitions for sequences of interfaces instances. */

INACTIVE

Documents
Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

```
typedef sequence <Resource> ResourceSequence;
typedef sequence <Material> MaterialSequence;
typedef sequence <MaterialGroup> MaterialGroupSequence;
typedef sequence <Job> JobSequence;
typedef sequence <JobSupervisor> JobSupervisorSequence;
```

6.4 *Resource Abstract Interface Group*

6.4.1 The Resource Abstract Interface Group provides a set of abstractions that are globally useful. Figure 1 is the Information Model for the Resource Abstract Interface Group.

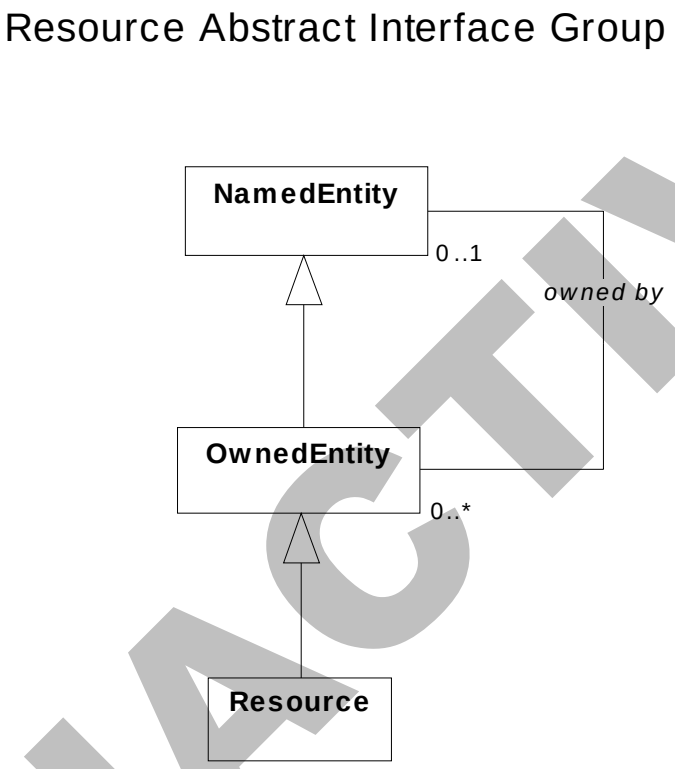

Figure 1
Resource Abstract Interface Group Information Model

6.4.1.1 All CIM Framework interfaces will inherit from one of the interfaces shown in Figure 1. NamedEntity provides the most basic naming functions. An OwnedEntity is a NamedEntity with functions supporting the concept of ownership. A Resource is an OwnedEntity which also takes an active role in product manufacturing.

6.4.2 *Named Entity Interface*

Module:	AbstractIF
Interface:	NamedEntity
Inherited Interface:	Implementation-dependent.
Description:	The abstract interface NamedEntity provides the concept of a named item. This allows for comparison and conversion of names via a standard object.
Exceptions:	None.
Published Events:	None.
Provided Services:	

INACTIVE

Documents Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

```
interface NamedEntity {
```

/* Set and get the name. The NamedEntity interface does not specify scoping of names or enforce uniqueness of names. This could allow distinct instances of a NamedEntity to use the same string as a name. */

```
void setName (in string name)
        raises (Global::FrameworkErrorSignal);

string getName ( )
        raises (Global::FrameworkErrorSignal);
```

/* Tests the equality of the name with the name provided as an argument. */

```
boolean isNamed (in string testName)
        raises (Global::FrameworkErrorSignal);

}; // NamedEntity
```

Contracted Services: None.

Dynamic Model: None.

6.4.3 *Owned Entity Interface*

6.4.3.1 The concept of ownership in the CIM Framework relates to the hierarchical structure that may be defined where one object "owns" another object. This should not be confused with the business concept of ownership relating to an item's value as an asset.

Module: AbstractIF

Interface: OwnedEntity

Inherited Interface: NamedEntity

Description: The abstract interface OwnedEntity provides for the concept of an "owned" entity. There may be only one "owner" for each instance of an OwnedEntity. The OwnedEntity is able to communicate with the owner to request services, or forward information of interest. To build a "parts of" hierarchy, a series of ownerships can be established.

Exceptions: None.

Published Events: None.

Provided Services:

```
interface OwnedEntity : NamedEntity {
```

/* Set and get owner. */

```
void setOwner (in NamedEntity owner)
        raises (Global::FrameworkErrorSignal);

NamedEntity getOwner ( )
        raises (Global::FrameworkErrorSignal);

}; // OwnedEntity
```

Contracted Services: None.

Dynamic Model: None.

6.4.4 *Resource Interface*

Module: AbstractIF

Interface: Resource

Inherited Interface: OwnedEntity

Description: Resource is an abstract inherited interface for any entity in the factory that takes an active role in advancing a product along its manufacturing life cycle (adds value). This includes the factory itself, personnel, production, planning and scheduling resources, and all of the machines used for processing, transporting, and storing materials. Resource provides a common set of services for monitoring and control. Resource uses the NamedEntity and OwnedEntity characteristics together to allow for the building of resource hierarchies.

INACTIVE

Documents
Document Sharing Hub

Full document details: SEMI E97-0200A-0307R

There must be a clear division between the state of the Resource and the condition of the physical entity which the Resource represents. For instance, a Machine is a resource, but the fact that it is "Out of Service" may not mean the physical equipment is shutdown on the shop floor. In fact, the equipment may be operating in manual mode. The Resource state represents the availability of the Resource object to accept work for the factory system.

Exceptions: None.

Published Events: None.

Provided Services:

```
interface Resource : OwnedEntity {
```

/* Perform the startup activities for this Resource. Should be implemented by Resource specializations. */

```
void startUp ( )
        raises (Global::FrameworkErrorSignal);
```

/* Perform normal shutdown activities for this Resource. Normal is defined as allowing the Resource to complete any current activities and "gracefully" shutdown. */

```
void shutdownNormal ( )
        raises (Global::FrameworkErrorSignal);
```

/* Perform immediate shutdown activities for this Resource. Immediate is defined as aborting or terminating any current activities and stopping activity as soon as possible. This should be implemented by Resource specializations. */

```
void shutdownImmediate ( )
        raises (Global::FrameworkErrorSignal);
```

/* Respond with the receiver's level in the Resource hierarchy. Resource specifies that each different type of Resource provide a "resourceLevel" identifier.

```
string resourceLevel ( )
        raises (Global::FrameworkErrorSignal);
```

/* The following service provides name scoping for Resources. Resource name scoping makes use of the notion of "Resource level" and the ownership hierarchy. For example, unique identification of MachineResources within a Machine is possible, but to identify them outside the Machine additional information about their ownership will be required.

Thus: nameQualifiedTo ("Machine") sent to the ProcessResource named "Chamber" answers

"TestMachine>Chamber".

If the Machine was owned by a Factory named "TestFactory", then:

nameQualifiedTo ("Factory") sent to the ProcessResource answers

"TestFactory>TestMachine>Chamber"

where the ProcessResource has now been uniquely identified for the given Factory.

There is no limit to the number of levels that may be addressed this way. Based on the implementations of nameQualifiedTo (string); a name need not always be concatenated, if a particular Resource level is not applicable to identification. */

```
string nameQualifiedTo (in string resourceLevel)
        raises (Global::FrameworkErrorSignal);
```

/* Returns the set of subordinate Resources for a given Resource. */

```
ResourceSequence subResources ( )
        raises (Global::FrameworkErrorSignal);
```

/* Answer if the resource is in service. In service means the resource is functional and ready to accept and perform its normal tasks. Derivatives of Resource are expected to expand this state (e.g., add sub-interfaces) that explicitly deal with such additional issues as capacity, "normal" work versus maintenance, etc. */

INACTIVE

Documents Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

```
boolean isInService ( );
```

/* Answer if the resource is out of service. Out of service means the resource is unable to accept or begin new tasks. Previously begun tasks may continue in some cases. */

```
boolean isOutOfService ( );
};  // Resource
```

Contracted Services: None.

Dynamic Model:

Implementations of Resource may extend the state model by providing additional sub-states that are wholly contained within a state defined here. Extending the state model by the addition of state transitions is also an option for subtypes of Resource.

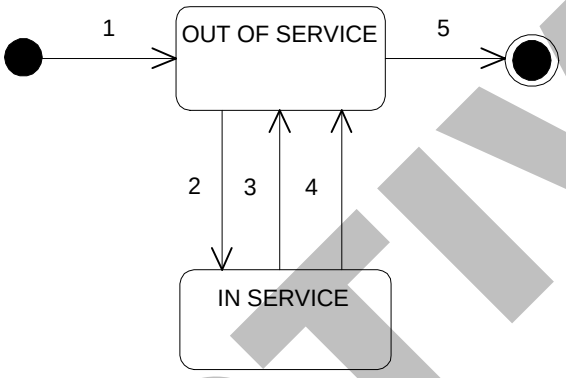

Figure 2
Resource Dynamic Model

Table 1 Resource State Definitions and Query Table

State	*Definition*	*Query for State via*
IN SERVICE	Resource is capable of interacting with other resources with its full service interface.	boolean isInService ( ); sent to the instance of Resource returns TRUE.
OUT OF SERVICE	Resource is not able to provide services	boolean isOutOfService ( ); sent to the instance of Resource returns TRUE.

Table 2 Resource State Transitions

#	*Current State*	*Trigger*	*New State*	*Action*	*Comment*
1	Non Existent	Object creation.	OUT OF SERVICE	None.	Default entry.
2	OUT OF SERVICE	startUp ( ) or object initialization.	IN SERVICE	Initiate Resource and subresources.	startUp() service initiates the trigger.
3	IN SERVICE	shutdownNormal ( ).	OUT OF SERVICE	Complete current execution of resources normally.	shutdownNormal() service initiates the trigger.
4	IN SERVICE	shutdownImmediate ( ). Take resource out of service.	OUT OF SERVICE	Stop execution of resource immediately.	shutdownImmediate() service initiates the trigger.
5	OUT OF SERVICE	Resource removed.	Non Existent	None.	

INACTIVE

Documents
Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

6.5 *Material Abstract Interface Group*

6.5.1 The CIM Framework uses a common architecture for identifying, grouping and locating materials. Any material has an identification, history, location, and associations to any material containers that contain it. Material can also be in multiple material groups that physically (same location or carrier) or logically (same lot, same process batch, same product family, etc.) associate material. Specializations of material include products, durables and consumables. Specializations of material groups include product groups, lots, and process groups.

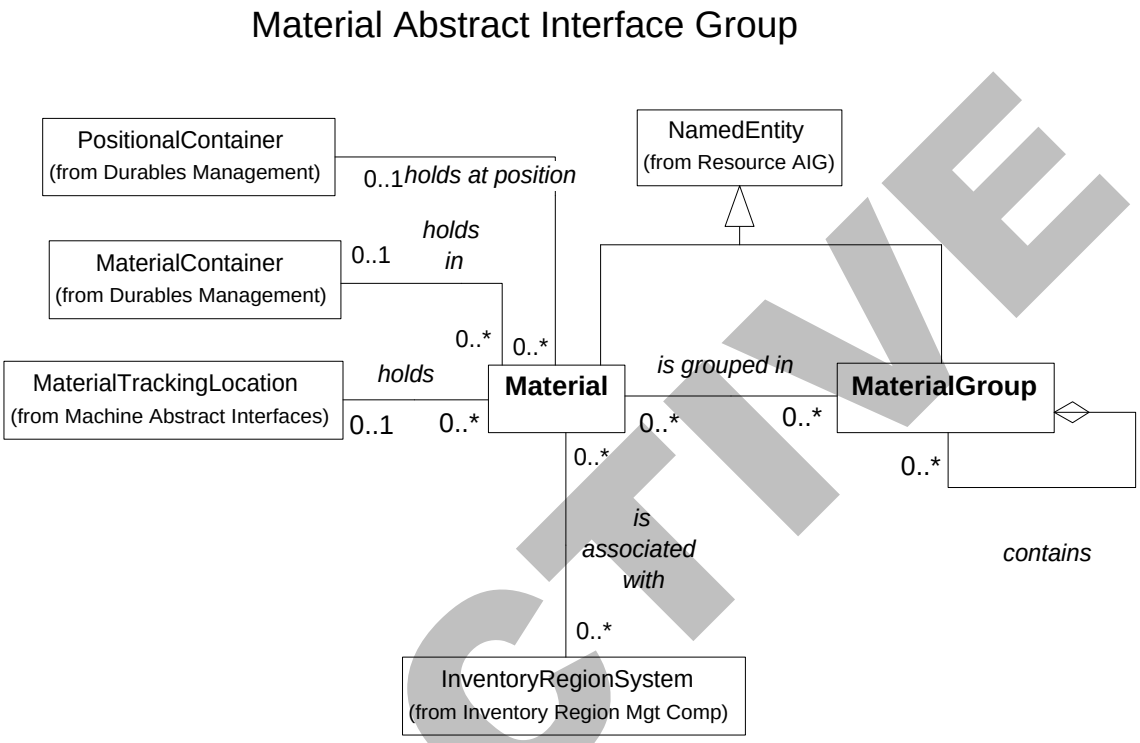

Figure 3
Material Abstract Interface Group Information Model

6.5.2 *Material Interface*

Module: AbstractIF

Interface: Material

Inherited Interface: NamedEntity

Description: Material is an abstract interface for physical items or substances that are required as inputs to the manufacturing process. This includes the product itself, consumables and durables used in the manufacturing process. It does not include the resources used to transform material into product.

Exceptions: None.

Published Events: None.

Provided Services:

```
interface Material : NamedEntity {
```

/* Set and get the identifier for this material. The identifier is unique within the extent of all material for an MES Factory. */

```
string getIdentifier ( )
        raises (Global::FrameworkErrorSignal);
```

INACTIVE

Documents Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

```
void setIdentifier (in string identifier)
        raises (Global::FrameworkErrorSignal,
        Global::DuplicateIdentifierSignal);
```

/* Returns the material groups of which the receiver is a member. */

```
MaterialGroupSequence materialGroups ( )
        raises (Global::FrameworkErrorSignal);
```

/* Answers whether the receiver is a member of the material group indicated by the argument. */

```
boolean isMemberOf (in MaterialGroup aMaterialGroup)
        raises (Global::FrameworkErrorSignal);
```

```
}; // Material
```

Dynamic Model: None.

6.5.3 *Material Group Interface*

Module: AbstractIF

Interface: MaterialGroup

Inherited Interface: NamedEntity

Description: MaterialGroup is the abstract interface for any aggregation of Material.

```
interface MaterialGroup : NamedEntity {
```

Exceptions:

/* Signals an attempt to add Material to the MaterialGroup that is already in the group. */

```
exception DuplicateMaterialSignal {Material aMaterial;};
```

/* Signals an attempt to add a MaterialGroup to a MaterialGroup that is already in the group. */

```
exception DuplicateMaterialGroupSignal {Material aMaterialGroup;};
```

/* Signals an attempt to remove Material that wasn't found in this MaterialGroup. */

```
exception MaterialRemovalFailedSignal {Material aMaterial;};
```

/* Signals an attempt to remove a MaterialGroup that wasn't found in this MaterialGroup. */

```
exception MaterialGroupRemovalFailedSignal {MaterialGroup aMaterialGroup;};
```

Published Events: None.

Provided Services:

/* Set and get the unique identifier for this group. */

```
string getIdentifier ( )
        raises (Global::FrameworkErrorSignal);
```

```
void setIdentifier (in string identifier)
        raises (Global::FrameworkErrorSignal,
        Global::DuplicateIdentifierSignal);
```

/* Adds the argument MaterialSequence to the collection of Matrial held by the receiver. */

```
void addMaterials (in MaterialSequence aMaterialSequence)
        raises (Global::FrameworkErrorSignal,
        DuplicateMaterialSignal);
```

/* Adds the argument Material to the collection of Material held by the receiver. */

```
void addMaterial (in Material aMaterial)
        raises (Global::FrameworkErrorSignal,
        DuplicateMaterialSignal);
```

/* Removes the Material indicated from the MaterialGroup. Throws the exception if not found. */

INACTIVE

Documents
Document Sharing Hub

Full document details: SEMI E97-0200A-0307R

```
void removeMaterial (in Material aMaterial)
        raises (Global::FrameworkErrorSignal,
        MaterialRemovalFailedSignal,
        Global::NotFoundSignal);
```

/* Remove and return all Material from the MaterialGroup. */

```
MaterialSequence removeAllMaterials ( )
        raises (Global::FrameworkErrorSignal);
```

/* Add the argument MaterialGroup to this MaterialGroup. */

```
void addMaterialGroup (in MaterialGroup aMaterialGroup)
        raises (Global::FrameworkErrorSignal,
        DuplicateMaterialGroupSignal);
```

/* Removes the argument MaterialGroup from this MaterialGroup. */

```
void removeMaterialGroup (in MaterialGroup aMaterialGroup)
        raises (Global::FrameworkErrorSignal,
        MaterialGroupRemovalFailedSignal,
        Global::NotFoundSignal);
```

/* Remove and return all MaterialGroups from this MaterialGroup. */

```
MaterialGroupSequence removeAllMaterialGroups ( )
        raises (Global::FrameworkErrorSignal);
```

/* Returns the collection of material in this MaterialGroup. */

```
MaterialSequence allMaterials ( )
        raises (Global::FrameworkErrorSignal);
```

/* Returns all the MaterialGroups in this MaterialGroup. */

```
MaterialGroupSequence allMaterialGroups ( )
        raises (Global::FrameworkErrorSignal);
```

/* Returns the count of the items in the MaterialGroup. */

```
long count ( )
        raises (Global::FrameworkErrorSignal);

}; // MaterialGroup
```

Dynamic Model: None.

6.6 *Job Supervision Abstract Interface Group*

6.6.1 The Job Supervision Abstract Interface Group provides the abstractions common to creating, executing and managing a "job," where a job can be defined as some system level operation which may be requested from the JobSupervisor. The job often spans a significant amount of time and multiple resources within the system. It is intended for specialization to provide specific job supervisors and jobs to provide system solutions.

INACTIVE

Documents Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

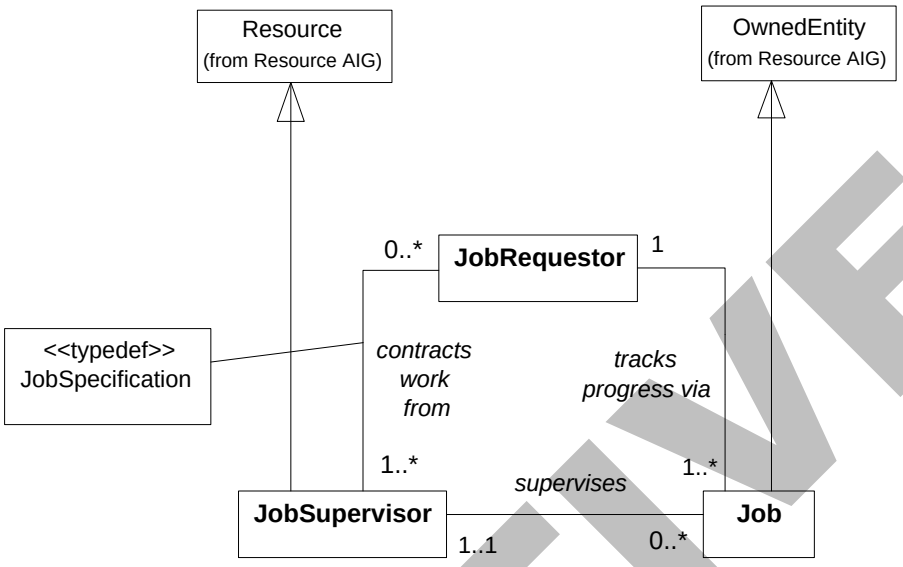

Figure 4
Job Supervision Abstract Interface Group Information Model

6.6.1.1 The basic Job Supervision Abstract Interface Group does not interact with other components, except to the extent that other components instantiate its interfaces. Figure 4 shows how the interfaces of Job Supervision relate to one another.

6.6.1.2 A Job Specification is a sequence of properties containing the parameters required to sufficiently define the work to be done. This sequence is passed by the JobRequestor to the JobSupervisor in the Job request message. See the JobSupervisor interface for more details.

6.6.1.3 JobSupervision levels are hierarchical. One level may accept a Job and delegate portions of that Job to lower levels. Jobs, however, are not purely hierarchical. A Job accepted by one JobSupervisor may be broken down, along with other Jobs of that component and reconstituted as needed to optimize the activities of the factory.

6.6.1.4 For example, a ProductRequest may ask for 15 wafers of a particular product. The ProductRequestManager may delegate a LotJob to Factory Operations with a Lot containing those 15 wafers and 10 more from a different ProductRequest. In the factory, this Lot may be split up and processed in smaller groups at various stages or, as scrap reduces the wafer count, combined with another small lot to create a more optimal process group. The Job Supervision implementation is allowed great latitude to optimize performance. Its only requirement is to fulfill the specification of the Job.

6.6.2 *JobSupervisor Interface*

Module:	AbstractIF
Interface:	JobSupervisor
Inherited Interface:	Resource
Description:	The JobSupervisor manages all the Jobs being performed by the component which implements it. It receives the requests for work, facilitates the creation of a Job for the task and returns (a reference to) that Job. A JobSupervisor will have a well defined domain which it can call on to perform work. These may be CIM Framework Resources if it delegates the work, or internal resources if it performs the work itself. Only activity requests which can be accomplished within the domain of a JobSupervisor should be issued to/accepted by that JobSupervisor.

INACTIVE

Documents
Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

Jobs, as subtypes of NamedEntity, are named by the JobSupervisor in such a way that their name attribute may be used to query for the Job. Jobs from different JobSupervisors may have the same name.

The definition of the work to be performed is the JobSpecification, a sequence of name/value pairs (see "Properties" definition). Specializations of Job Supervision may require certain properties in the JobSpecification. Some commonly useful properties are defined in the following table. When possible, specializations should reuse these definitions. Specializations should also document the allowable and mandatory properties that are supported. See Job definition for more information.

```
interface JobSupervisor : Resource {
```

Type Definitions:

/* Type for returning results of Job execution. */

```
typedef Global::NameValueSequence Results;
```

JobSpecification Properties:

Name	*Value Type*	*Description*
"JobType"	string	The kind of Job to run. This is useful when a JobSupervisor can initiate more than one type of Job. String values reserved for this property are: ProductRequestJobType, LotJobType, AreaJobType, ProcessMachineJobType, TransportJobType, PMJobType, ControlStrategyJobType, AlgorithmJobType.
"Priority"	long	Integer value, ranges from 1–99, where 1 is the highest priority and 99 is the lowest.
"Deadline"	TimeStamp	The Job is expected to be completed no later than the specified value of the Deadline.

Exceptions:

/* Requested Job was rejected. */

```
exception JobRejectedSignal {
        string errorMessage; };
```

/* Requested Job was not found */

```
exception JobNotFoundSignal {
        string errorMessage;
        string missingJobName; };
```

Published Events:

/* Provide lifecycle event for tracking */

```
const string JobLifecycleSubject =
        "/JobSupervision/JobSupervisor/JobLifecycle";
```

/* The use of "name" here (and in all other events) indicates the string value for the name or identifier of the Job to which the event refers. Since filtering does not support object reference comparisons, the filtering must be on the "name" of the object. */

```
struct JobLifecycleFilters {
        Global::Property name;
        Global::Property lifecycle;
};
```

INACTIVE

Documents Document Sharing Hub
Full document details: SEMI E97-0200A-0307R

JobLifecycleFilters Properties:

Name	*Value Type*	*Description*
"Name"	string	The name of the Job.
"Lifecycle"	Global::LifecycleState	New lifecycle state.

```
struct JobLifecycleEvent {
        string eventSubject;
        Global::TimeStamp eventTimeStamp;
        JobLifecycleFilters eventFilterData;
        Global::Properties eventNews;
        Job aJob; // on Delete, aJob is nil
};
```

Provided Services:

/* Request that work be done according to the referred specification. A Job which represents the work is returned for future reference. The post-condition for this operation is the specified Job successfully created. */

```
Job requestJob (
        in Global::Properties aJobSpecification,
        in JobRequestor aJobRequestor)
        raises (Global::FrameworkErrorSignal,
        JobRejectedSignal);
```

/* Request that work be done according to the Job specification. This operation blocks until the Job completes. Results generated by the Job are returned. The post-condition for this operation is a successful execution of the specified Job. This interface offers a lightweight alternative to ***requestJob*** that does not require that requestors support the JobRequestor interfaces (e.g., informJobStateChange operation). A Job may or may not be created by runJob, but if created, the Job may be accessed with the Job query operations of JobSupervisor. */

```
Results runJob (
        in Global::Properties aJobSpecification)
        raises (Global::FrameworkErrorSignal,
        JobRejectedSignal);
```

/* Ask whether the Job specified by the JobSpecification would be accepted for current or future (queued) processing if a requestJob or runJob message were issued now. */

```
boolean canPerform (in Properties aJobSpecification)
        raises (Global::FrameworkErrorSignal);
```

/* Command to begin the pausing of all Jobs of this JobSupervisor which can be paused (e.g., Jobs that have not reached the Finished state). */

```
void pauseAllJobs ()
        raises (Global::FrameworkErrorSignal);
```

/* Command to "resume" all Jobs of this JobSupervisor which are currently Paused. */

```
void resumeAllJobs ()
        raises (Global::FrameworkErrorSignal);
```

/* This command aborts all the Jobs under the control of the JobSupervisor immediately, without regard to the impact of abruptly halting the Job. This service should be used with great caution. It may result in irrevocable change to factory or material state. */

```
void abortAllJobs ()
        raises (Global::FrameworkErrorSignal);
```

/* This command stops all Jobs under control of the JobSupervisor, as quickly as possible. Stopping Jobs should not result in damage to the factory or material being processed. */

```
void stopAllJobs ()
        raises (Global::FrameworkErrorSignal);
```

INACTIVE

Documents
Document Sharing Hub
Full document details: SEMI E97-0200A-0307R