# SEMI E86-0200
# PROVISIONAL SPECIFICATION FOR CIM FRAMEWORK FACTORY LABOR COMPONENT

This specification 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 October 21, 1999. Initially available at www.semi.org January 2000; to be published February 2000. Originally published September 1999.

## 1 Purpose

1.1 The Factory Labor Component provides the capability to support the qualification and management of available, qualified human resources for manufacturing operations.

1.2 The technical content of this ballot addresses the portion of the CIM Framework domain specified in SEMI E81, Provisional Specification for CIM Framework Domain Architecture, concerned with management of the availability of qualified factory personnel. This specification provides the interfaces required by Manufacturing Execution Systems to:

- Configure the Person resource model.
  - Each Person's name (or appropriate alias), identification, department, role, etc. There may be cases where identifying information should not be used to associate a person with the required factory labor records.
  - Each Person's initial skills and authorization to perform jobs and access data.
  - Each Person's initial assignment to other factory resources (such as specific machines or factory areas).
  - Each Person's medical qualifications.
- Define skill maintenance tasks and what triggers them.
- Track and report the Person's skills, authorization, and assignments and record these in the Person's histories.
- Support assignment of Persons to factory jobs and monitor and record job progress (from the perspective of the Person's role).
- Monitor skill maintenance triggers (such as expired skill certification) and recommend skill maintenance jobs.
- Execute triggered skill maintenance jobs.
  - Change Person capabilities (before and after training).
  - Execute and monitor skill maintenance jobs (training, certification testing) and report job progress.
  - Record skill maintenance triggers and job results in Person's skill maintenance histories.
- Collect and report Person and skill performance (utilization and effectiveness).
- Maintain relationships between Persons and system security for identity authentication and authorization.

## 2 Scope

2.1 Specifically, this component provides the services to:

- Manage the assignment of qualified and available labor resources for factory operations.
- Manage the training and medical requirements for factory operations.
- Support the maintenance of personnel information (full name, employee number, department, shift assignment, machines qualified to operate, course and medical examination records).
- Support the maintenance of personnel skill qualification histories (courses and medical examinations successfully completed) of factory personnel.

2.2 The Factory Labor Component is divided into two subcomponents:

- Person Management, which tracks, logs, and maintains availability and task assignment for factory operations personnel within the manufacturing facility. Also provides support for the management of personnel information, training, and medical qualifications.
- Skill Management, which provides support for the management of training and medical requirements for factory operations.

2.3 Figure 1 depicts the relationships between these two components and their interaction with other CIM Framework components defined in the Section 7.2 Functional Partitioning of SEMI E81, Provisional

Specification for CIM Framework Domain Architecture.

2.4 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 limitations prior to use.

## 3 Limitations

3.1 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.

3.2 *Provisional Status*

3.2.1 The following items need to be completed before conducting a subsequent ballot to upgrade the Factory Labor Component to full standard status.

- Define skill maintenance tasks and what triggers them.
- Monitor skill maintenance triggers (such as expired skill certification) and recommend skill maintenance jobs.
- Maintain relationships between persons and system security for identity authentication and authorization.

3.2.2 It is anticipated that a future document defining the Resource Abstract Interface will satisfy the first two items.

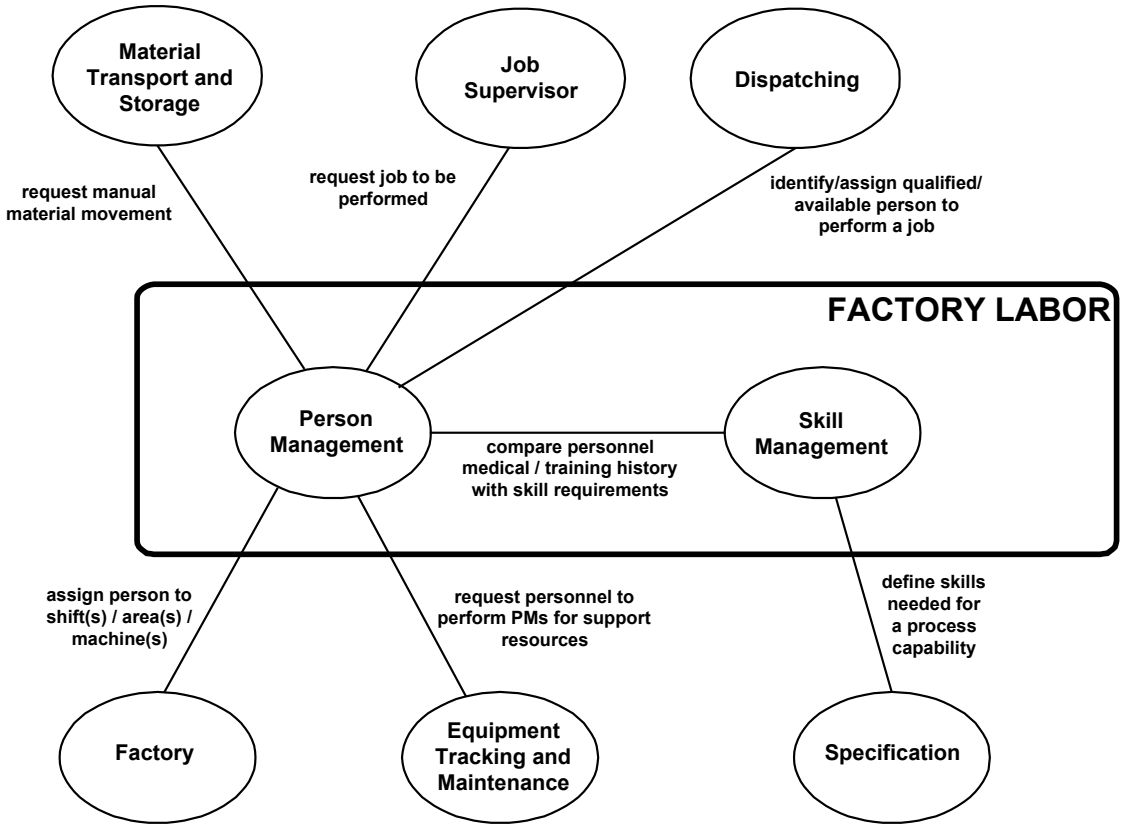


**Figure 1**
**Factory Labor Component Relationships**

## 4 Referenced Standards

4.1 *SEMI Standard*

SEMI E81 — Provisional Specification for CIM Framework Domain Architecture

4.2 *ISO/IEC Standard*[1]

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

4.3 *OMG Standard*[2]

UML Notation Guide, Version 1.1, document number ad/97-08-05

NOTE: As listed or revised, all documents cited shall be the latest publications of adopted standards.

## 5 Terminology

5.1 *person* — personnel performing manufacturing operations within a factory.

5.2 *person responsibility* — the concept of "responsibility" implies that factory personnel are authorized to perform, operate or access a specific factory object (but it does not imply ownership).

5.3 *skill* — skill is an attribute of factory personnel denoting that they are qualified to assist a process resource or job in the performance of a process capability or some other factory operation.

5.4 *skill qualification* — a skill qualification indicates that factory personnel meet a requirement (or all requirements) necessary to perform some factory operation.

5.5 *skill requirement* — a skill requirement is the training or medical examination(s) required to achieve a specific skill.

## 6 Requirements

6.1 The following sections provide the specification of interfaces that comprise the Factory Labor Component. The model notation used is the Unified Modeling Language (UML). UML is documented in the UML Notation Guide. The details of each interface are supplied using standard IT-ODP-IDL. Conformant implementations of the Factory Labor Component shall provide support for all elements of the IDL specifications and the semantics provided in the UML models. Conformant implementations are not constrained to any specific implementation approach, provided that a mapping is provided to the IDL specification.

6.2 *External Interfaces, Global Type Definitions, Global Exceptions and Events*

6.2.1 The following interfaces (external to this component), global type definitions, global exceptions, and events are required to support the Factory Labor component specification.

6.2.2 *Abstract and Base Interfaces*

6.2.2.1 Base interfaces provide a suite of interfaces that are used throughout the CIM Framework Specification. The Abstract Interfaces provide a hierarchy of abstract interfaces that are used as "Inherited Interfaces" in the rest of the framework. These interfaces are expected to be specialized in a specific implementation.

6.2.2.2 This section identifies base and abstract interfaces required by the Factory Labor component specification.

```
interface NamedEntity {
};

interface OwnedEntity : NamedEntity {
};
```

---

*1 ISO Central Secretariat, 1, rue de Varembé, Case postale 56, CH-1211 Genève 20, Switzerland*

*2 Object Management Group, Inc., Framingham Corporate Center, 492 Old Connecticut Path, Framingham, MA 01701, 1.508.820.4300, http://www.omg.org/cgi-bin/doclist.pl*

```
interface Resource : OwnedEntity {
};

interface ComponentManager : Resource {
};

interface Job {
} ;
```

6.2.3 *Other CIM Framework Interfaces*

6.2.3.1 This section identifies interfaces within other CIM Framework components that are needed to satisfy relationships with the Factory Labor component.

```
interface HistoryCollection { // from Factory Services
} ;

interface History { // from Factory Services
} ;

interface MESFactory { // from Factory Management
} ;
```

6.2.4 *Global Type Definitions*

6.2.4.1 The CIM Framework uses IT-ODP-IDL *typedefs* in two ways:

- To define aliases for compound object types (*structs*), which can then be used in IDL specifications and
- 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).

6.2.4.2 This section identifies the global type definitions required by the Factory Labor component specification.

```
typedef sequence<string> stringSequence ;

typedef struct HistoryEvent_body {
        string subject ;
} HistoryEvent ;

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

typedef ulonglong TimeT ;

typedef TimeT TimeStamp ;

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

```
typedef IntervalT TimeWindow ;

typedef struct Shift_Structure {
        string name ;
        // ...
} Shift ;

typedef struct PersonResponsibilitystruct {
        string responsibilityCategory ;
        any responsibility ;
} PersonResponsibility;

typedef sequence<ProcessCapability> ProcessCapabilitySequence ;
typedef stringSequence CategorySequence ;
typedef sequence<HistoryEvent> HistoryEventSequence ;
typedef sequence<Job> JobSequence ;
typedef sequence<Machine> MachineSequence ;
typedef sequence<Person> PersonSequence ;
typedef sequence<QualificationData> QualificationDataSequence ;
typedef sequence<Resource> ResourceSequence ;
typedef sequence<SkillRequirement> RequirementSequence ;
typedef sequence<PersonResponsibility> ResponsibilitySequence ;
typedef sequence<Skill> SkillSequence ;

typedef string PropertyName;
struct Property
    {
    PropertyName property_name;
    any property_value;
    };
typedef sequence<Property> Properties;
```

6.2.5 *Global Exception Declarations*

6.2.5.1 An exception is an infrastructure mechanism used to notify a calling client of an operation that an unusual condition occurred in carrying out the operation. This section identifies the global exception declarations required by the Factory Labor component specification.

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

exception InvalidStateTransitionSignal {
} ;

exception SetValueOutOfRangeSignal {
} ;
```

6.2.6 *Events*

6.2.6.1 An event is an asynchronous message denoting the occurrence of some incident of importance. For example, state change or new object created. This section identifies the events defined by the Factory Labor component specification. Note that "//" or "/* ....*/" infers a comment.

```
/* Person lifecycle event definition. */
const string PersonLifecycleSubject =
        "/PersonManagement/PersonManager/PersonLifecycle" ;
typedef struct PersonLifecycleFilters_Structure {
        Property        name;           // “Name”, aPerson’s name
        Property        lifecycleEvent; // “LifecycleEvent”, LifecycleState
} PersonLifecycleFilters;
typedef struct PersonLifecycleEvent_Structure {
        string                          eventSubject;
        TimeStamp                       eventTimeStamp;
        PersonLifecycleFilters          eventFilterData;
        Properties                      eventNews;
        Person                          aPerson;
                        // Except when “Delete” where value will be nil
} PersonLifecycleEvent;

/* Person capacity changed event definition. */
const string PersonCapacityChangedSubject =
        "/PersonManagement/Person/CapabilityChanged" ;
typedef struct PersonCapacityChangedFilters_Structure {
        Property        name;           // “Name”, aPerson’s name
} PersonCapacityChangedFilters;
typedef struct PersonCapacityChangedEvent_Structure {
        string                          eventSubject;
        TimeStamp                       eventTimeStamp;
        PersonCapacityChangedFilters    eventFilterData;
        Properties                      eventNews;
        Person                          aPerson;
} PersonCapacityChangedEvent;

/* Person state changed event definition. */
const string PersonStateChangedSubject = "/PersonManagement/Person/StateChanged";
enum PersonState {PersonDefined, PersonOffShift, PersonOnShift,
PersonAvailableForWork, PersonNotAvailableForWork, PersonUnassignedToMachines,
PersonAssignedToJobs, PersonAssignedToMachines, PersonAvailableForMoreAssignments,
PersonAssignmentAtCapacityExceeded, PersonIdleWithJob, PersonBusyWithJob };
typedef struct PersonStateChangedFilters_Structure {
        Property        name;           // “Name”, aPerson’s name
        Property        formerState;    // “FormerState”, PersonState
        Property        newState;       // “NewState”, PersonState
} PersonStateChangedFilters;
typedef struct PersonStateChangedEvent_Structure {
        string                          eventSubject;
        TimeStamp                       eventTimeStamp;
        PersonStateChangedFilters       eventFilterData;
        Properties                      eventNews;
        Person                          aPerson;
} PersonStateChangedEvent;

/* Person shift worked changed event definition. */
const string PersonShiftChangedSubject = "/PersonManagement/Person/ShiftChanged";
typedef struct PersonShiftChangedFilters_Structure {
        Property        name;           // “Name”, aPerson’s name
        Property        fromShift;      // “FromShift”, former shift
```

```
        Property        toShift;        // "ToShift", new shift
} PersonShiftChangedFilters;
typedef struct PersonShiftChangedEvent_Structure {
        string                          eventSubject;
        TimeStamp                       eventTimeStamp;
        PersonShiftChangedFilters       eventFilterData;
        Properties                      eventNews;
        Person                          aPerson;
} PersonShiftChangedEvent;
```

6.3 *Person Management Component*

6.3.1 This component defines factory personnel (Person interface) to a CIM system. Persons are viewed as resources in the manufacturing environment, are assigned to a factory area, and own a detailed training and medical examination history (which qualifies them to perform a skill or set of skills). For example, there may be a requirement that an operator periodically completes specific training or medical testing. Qualifications would include verification that required training had occurred within the specified time limit. The associated skill would identify the process capability enabled for this person through that qualification. At any given time, a Person may be assigned to one or more factory jobs associated with other factory resources or operations.

6.3.2 The Component Information Model for Person Management is shown in Figure 2.

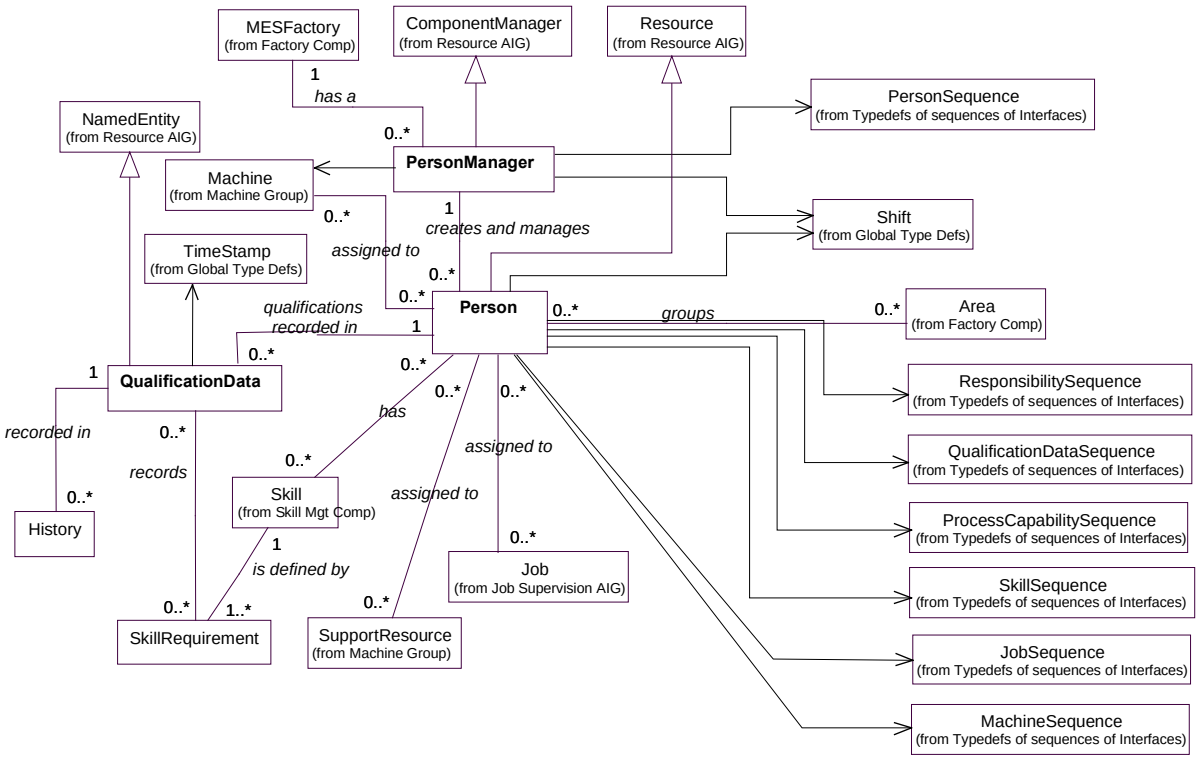


**Figure 2**
**Person Management Component Information Model**

6.3.3 *PersonManager Interface*

Interface: PersonManager

Inherited Interface: ComponentManager

Description: This is the interface between the Person Management Component and the rest of the CIM environment. The PersonManager creates instances of Persons within the factory and manages those Persons (shift, job assignment and status information). There are no limitations to the number of PersonManagers within a factory.

Exceptions:

/* This signal is raised with attempted creation of Person through the use of a Person identifier that already exists. */

```
exception PersonDuplicateSignal {string identificationNumber;};
```

/* This signal is raised when a retrieval operation fails because the Person specified by the argument could not be located. */

```
exception PersonNotFoundSignal {string identificationNumber;};
```

/* This signal is raised when a removal operation fails because the Person specified by the argument is not assigned to the PersonManager. */

```
exception PersonNotAssigned { Person unknownPerson;};
exception PersonRemovalFailedSignal { };
```

Published Events:

```
PersonLifecycleEvent
```

Provided Services:

/* Create an instance of Person with a unique identifier and add to the list managed by PersonManager. */

```
Person createPersonWithIdentifier (in string identificationNumber)
        raises (FrameworkErrorSignal, PersonDuplicateSignal);
```

/* Delete the Person identified by the argument from the list managed by PersonManager. */

```
void removePerson (in Person aPerson)
        raises (FrameworkErrorSignal, PersonRemovalFailedSignal,
                PersonNotAssignedSignal);
```

/* Return a set of Persons managed by the PersonManager. */

```
PersonSequence allPersons ( ) raises (FrameworkErrorSignal);
```

/* Search for an instance of Person using its unique identifier. Raises an exception if the Person is not found. */

```
Person findByIdentifier (in string identificationNumber)
        raises (FrameworkErrorSignal, PersonNotFoundSignal);
```

/* Return a set of all Persons logged on to the specified shift. */

```
PersonSequence allOnShiftPersons (in shift aShift) raises (FrameworkErrorSignal);
```

/* Return a set of all Persons available for work. */

```
PersonSequence allAvailableForWorkPersons ( ) raises (FrameworkErrorSignal);
```

/* Return a set of all Persons not available for work. */

```
PersonSequence allNotAvailableForWorkPersons ( ) raises (FrameworkErrorSignal);
/***** Manage Person / Machine relationship. *****/
```

/* Return a set of Persons qualified to operate a process or material handling machine. The Person is qualified to operate the machine if the machine is noted in the Person's skill list or if a comparison of the person's course and medical exam histories satisfy the requirements defined within the Skill Management Component for the noted machine. This method does not check the availability of the person to actually perform a task. */

```
PersonSequence allPersonsWithSkill (in Skill aSkill) raises (FrameworkErrorSignal);
```

/* Return a set of Persons available ("on duty" on the current shift) to operate the specified machine. This service assumes that a Person is qualified to operate the machine. */

```
PersonSequence allPersonsAvailableForMachine (in Machine aMachine)
        raises (FrameworkErrorSignal);
```

/* Return a set of all Persons currently assigned to the specified machine. */

```
PersonSequence allPersonsAssignedToMachine (in Machine aMachine)
        raises (FrameworkErrorSignal);
```

/* Return a set of all Persons currently assigned to machines. */

```
PersonSequence allAssignedToMachinesPersons ( ) raises (FrameworkErrorSignal);
```

/***** Manage Person/Job relationship. *****/

/* Return a set of Persons qualified to perform a process or material handling job. The Person is qualified for the job if the job qualification is noted in the person's skill list or if a comparison of the person's course and medical exam histories satisfy the requirements defined within the Skill Management Component for the noted job. This method does not check the availability of the person to actually perform a task. */

```
PersonSequence allPersonsQualifiedForJob (in Job aJob)
        raises (FrameworkErrorSignal);
```

/* Return a set of Persons available ("on duty" on the current shift) to perform a process or material movement job. This service assumes that a Person is qualified to perform the job. */

```
PersonSequence allPersonsAvailableForJob (in Job aJob)
        raises (FrameworkErrorSignal);
```

/* Return a set of Persons currently assigned to the specified job. */

```
PersonSequence allPersonsAssignedToJob (in Job aJob) raises (FrameworkErrorSignal);
```

/* Return a set of all Persons currently assigned to jobs. */

```
PersonSequence allAssignedToJobsPersons ( ) raises (FrameworkErrorSignal);
```

Contracted Services: None.

Dynamic Model: Same as inherited interface (Component Manager).

6.3.4 *Person Interface*

Interface: Person

Inherited Interface: Resource

Description: This interface represents factory personnel. As currently defined, Person refers specifically to operating personnel within the manufacturing facility. Instances of this interface hold the following information pertaining to these individuals: employee information (full name, identification number, department, shift assignment); training and medical certification; and factory operations (e.g., operate a machine, perform a process) qualified to perform.

Exceptions:

/* This signal is raised when an addition operation fails because the Qualification specified by the argument already exists. */

```
exception QualificationDuplicateSignal {
        QualificationData qualificationInformation;};
```

/* This signal is raised when a removal operation fails because the Qualification specified by the argument could not be located. */

```
exception QualificationNotAssignedSignal {string identificationNumber;};
```

/* This signal is raised when a retrieval operation fails because the Qualification specified by the argument could not be located. */

```
exception QualificationNotFoundSignal {string identificationNumber;};
```

/* This signal is raised when an addition operation fails because the Responsibility specified by the argument already exists. */

```
exception ResponsibilityDuplicateSignal {any responsibility ;
        string responsibilityCategory;};
```

/* This signal is raised when an operation fails because the Responsibility category specified by the argument could not be located. */

```
exception ResponsibilityCategoryNotFoundSignal {
        string responsibilityCategory;};
exception ResponsibilityCategoryRemovalFailedSignal { };
```

/* This signal is raised when a removal operation fails because the Responsibility specified by the argument could not be located. */

```
exception ResponsibilityRemovalFailedSignal {
        any responsibility;
        string responsibilityCategory;};
```

/* This signal is raised when an addition operation fails because the skill specified by the argument already exists. */

```
exception SkillDuplicateSignal {Skill aSkill;};
```

/* This signal is raised when a removal operation fails because the skill specified by the argument could not be located. */

```
exception SkillRemovalFailedSignal {Skill aSkill;};
exception SkillNotAssignedSignal {Skill aSkill;};
```

Published Events:

/* This event is posted when a Person's capacity to do work has changed. This includes assignment to/deassignment from machines and jobs. It also includes changing the number of jobs and/or machines to which a Person may be assigned. */

```
        PersonCapacityChangedEvent
```

/* This event is posted when a Person's availability has changed. This includes logging on/off a shift, leaving/returning to a work station, or capacity exceeded. */

```
        PersonStateChangedEvent
```

/* This event is posted when a Person has been reassigned to a different shift. */

```
        PersonShiftChangedEvent
```

Provided Services:

/* Set and get the Person's department. */

```
void setDepartment (in string department)
        raises (SetValueOutOfRange, FrameworkErrorSignal);
string getDepartment ( ) raises (FrameworkErrorSignal);
```

/* Set and get the Person's name. */

```
void setFullname (in string fullname)
         raises (SetValueOutOfRange, FrameworkErrorSignal);
string getFullname ( ) raises (FrameworkErrorSignal);
```

/* Set and get the Person's identification number. */

```
void setIdentificationNumber (in string idNumber)
         raises (SetValueOutOfRange, FrameworkErrorSignal);
string getIdentificationNumber ( ) raises (FrameworkErrorSignal);
```

/* Set and get the maximum number of jobs to which a Person can be assigned simultaneously. */

```
void setMaximumAssignedJobs (in long maximumJobs)
         raises (SetValueOutOfRange, FrameworkErrorSignal);
long getMaximumAssignedJobs ( ) raises (FrameworkErrorSignal);
```

/* Set and get the maximum number of machines to which a Person can be assigned simultaneously. */

```
void setMaximumAssignedMachines (in long maximumMachine)
        raises (SetValueOutOfRange, FrameworkErrorSignal);
long getMaximumAssignedMachines ( ) raises (FrameworkErrorSignal);
```

/* Set and get the Person's shift assignment. */

```
void setShift (in shift aShift) raises (SetValueOutOfRange, FrameworkErrorSignal);
shift getShift ( ) raises (FrameworkErrorSignal);
```

/* Add an object within the factory to a Person’s responsibility category. The concept of “responsibility” implies that a Person is authorized to perform, operate or access the factory object. It does not imply ownership. The first argument represents the responsibility (object within the factory) which is associated with a responsibility category, while the second argument is the category. If the specified category does not exist, it is created. The service returns the added factory object. For example, add a machine to the Person’s “maintenance responsibilities” category. Another example: add a change notice to a Person’s “signoff responsibility” category. */

```
any addResponsibility_toCategory (in any responsibility,
        in string responsibilityCategory)
        raises (FrameworkErrorSignal, ResponsibilityDuplicateSignal,
                ResponsibilityCategoryNotFoundSignal );
```

/* Remove an object within the factory from a Person’s responsibility category. If the specified category, or responsibility does not exist, then an exception is raised. */

```
void removeResponsibility_fromCategory (in any responsibility,
        in string responsibilityCategory)
        raises (FrameworkErrorSignal, ResponsibilityRemovalFailedSignal,
                ResponsibilityCategoryRemovalFailedSignal);
```

/* Return the responsibilities (objects within a factory) which are assigned to a Person’s responsibility category. For example, return all of the specifications in the category “specification management.” A dictionary of key/value pairs is returned that represents these responsibility associations. The key is the “responsibility category name” while the value is the set of factory objects associated with that category. */

```
ResponsibilitySequence responsibilities ( ) raises (FrameworkErrorSignal);
```

/* Add the skill (qualification to operate a specific machine) to the Person's list of skills. Skills may be assigned or obtained through the completion of specified course (s) and medical examination (s). This service does not detail specific course and medical examination requirements. */

```
void addSkill (in Skill aSkill)
        raises (FrameworkErrorSignal, SkillDuplicateSignal);
```

/* Remove the skill from the Person's list of skills. */

```
void removeSkill (in Skill aSkill)
        raises (FrameworkErrorSignal, SkillRemovalFailedSignal,
                SkillNotFoundSignal);
```

/* Retrieve the set of skills for which this Person is qualified. */

```
SkillSequence skills ( ) raises (FrameworkErrorSignal);
```

/* Add the specified qualification (training course or medical examination) data to the training or medical history of a Person. */

```
void addQualification (in QualificationData qualificationInformation)
        raises (FrameworkErrorSignal, QualificationDuplicateSignal);
```

/* Remove the specified qualification data from the training or medical history of a Person. */

```
void removeQualification (in QualificationData qualificationInformation)
        raises (FrameworkErrorSignal, QualificationNotAssignedSignal);
```

/* Retrieve all information relative to a Person's particular qualification by performing a search using its unique identifier. */

```
QualificationData findQualificationByIdentifier (in string identifier)
        raises (FrameworkErrorSignal, QualificationNotFoundSignal);
```

/* Retrieve the set of qualifications attained by this Person. */

```
QualificationDataSequence qualifications ( ) raises (FrameworkErrorSignal);
```

/* Return the sequence of process capabilities that this Person is qualified to perform. */

```
ProcessCapabilitySequence processCapabilities ( ) raises (FrameworkErrorSignal);
```

/***** The following services provide basic state control and query. *****/

/* Denote that this person has logged on to the current shift and is AVAILABLE. */

```
void makeOnShift ( )
```

/* Denote that this person has logged off of the current shift and is NOT AVAILABLE. */

```
void makeOffShift ( )
```

/* Denote that this Person is at a workstation and available for work. It is required that the Person has logged on to the current shift. Assignment to machine (s) and/or job (s) may or may not have occurred. */

```
void makeAtWorkStation ( )
       raises (FrameworkErrorSignal, InvalidStateTransitionSignal );
```

/* Denote that this Person is not immediately available at a work station and cannot perform work. An example of this condition is an operator taking a personal break during a shift. */

```
void makeNotAtWorkStation ( )
       raises (FrameworkErrorSignal, InvalidStateTransitionSignal );
```

/* Return a boolean denoting whether or not a Person is logged on to the current shift. */

```
boolean isOnShift ( ) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is available for work. */

```
boolean isAvailableForWork ( ) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is at their work station. */

```
boolean isAtWorkStation ( ) raises (FrameworkErrorSignal);
```

/* Determine if a Person is qualified to perform a process or material handling job for a specific machine (has a specific skill.) The Person is qualified to operate the machine if the machine is noted in the Person's skill list, or if a comparison of the Person's course and medical exam histories satisfy the requirements defined within the Skill Management Component for the noted machine. This service does not check the availability of the Person to actually perform a task. */

```
boolean hasSkill (in Skill aSkill) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is qualified for any machine. */

```
boolean hasSkills ( ) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is available for the specified machine. */

```
boolean isAvailableForMachine (in Machine aMachine) raises (FrameworkErrorSignal);
```

/* Assign Person to a machine. A Person may be assigned to more than one machine at a time.*/

```
void assignToMachine (in Machine aMachine) raises (FrameworkErrorSignal);
```

/* Terminates personnel assignment to a machine. */

```
void deassignFromMachine (in Machine aMachine) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is assigned to the specified machine. */

```
boolean isAssignedToMachine (in Machine aMachine) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is assigned to any machine. */

```
boolean isAssignedToMachines ( ) raises (FrameworkErrorSignal);
```

/* Retrieve the set of machines to which a Person is assigned. */

```
MachineSequence assignedMachines ( ) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is available to be assigned to more machines. */

```
boolean isAvailableForMoreMachineAssignments ( ) raises (FrameworkErrorSignal);
```

/***** The following services support job assignment. *****/

/* Return a boolean denoting whether or not a Person is qualified for the specified job. */

```
boolean isQualifiedForJob (in Job aJob) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is qualified for any job. */

```
boolean isQualifiedForJobs ( ) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is available for the specified job. */

```
boolean isAvailableForJob (in Job aJob) raises (FrameworkErrorSignal);
```

/* Assigns a Person to a job. A Person may be allocated to more than one job at a time. */

```
void assignToJob (in Job aJob) raises (FrameworkErrorSignal);
```

/* Terminates personnel assignment to a job. */

```
void deassignFromJob (in Job aJob) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is assigned to the specified job. */

```
boolean isAssignedToJob (in Job aJob) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is assigned to any jobs. */

```
boolean isAssignedToJobs ( ) raises (FrameworkErrorSignal);
```

/* Retrieve the set of jobs to which a Person is assigned. */

```
JobSequence assignedJobs ( ) raises (FrameworkErrorSignal);
```

/* Return a boolean denoting whether or not a Person is available for more jobs. */

```
boolean isAvailableForMoreJobAssignments ( ) raises (FrameworkErrorSignal);
```

Contracted Services: None.

Dynamic Model:

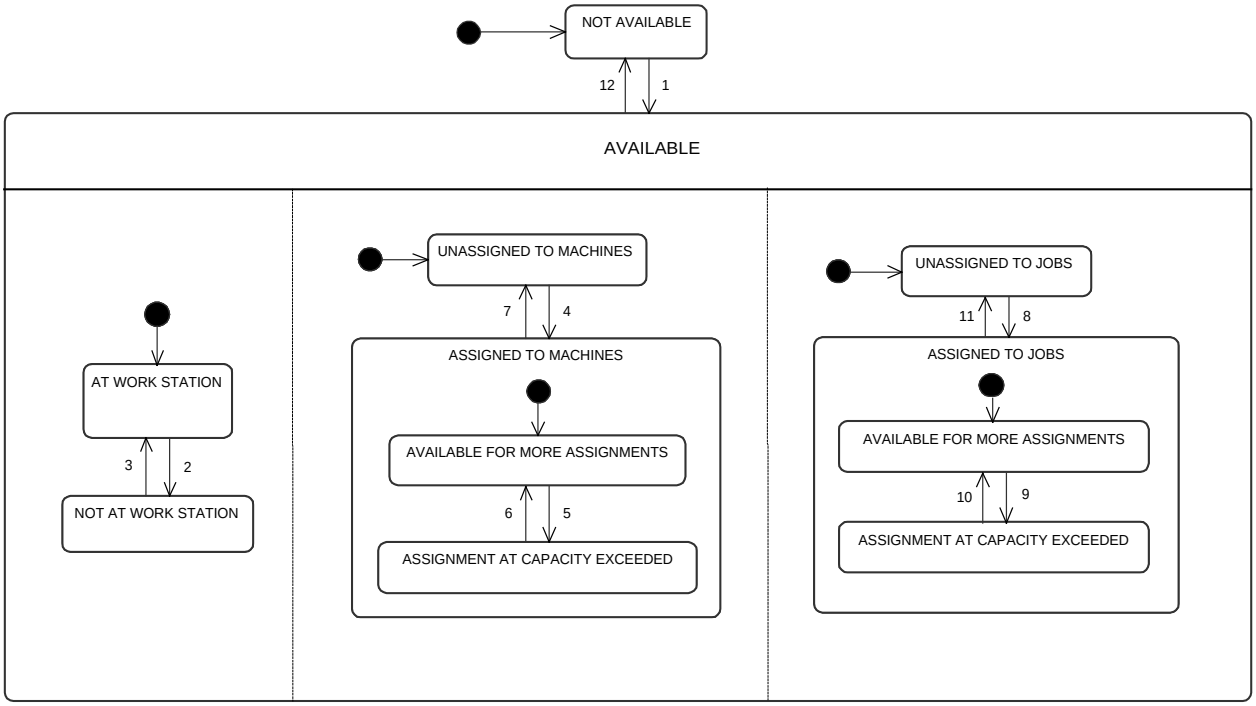


**Figure 3**
**Person Dynamic Model**

6.3.4.1 Object State Tables

**Table 1 Person State Definitions and Query**

| *State* | *Definition* | *Query for State via* |
|---|---|---|
| NOT AVAILABLE | In this state, a Person is not clocked on to shift and cannot perform work. | boolean isOnShift ( ) sent to instance of Person returns FALSE. Person **NOT** included in set returned by PersonManager services. PersonSequence allOnShiftPersons (in Shift aShift) or allAvailableForWorkPersons ( ). |
| AVAILABLE | In this superstate, a Person is clocked on to a shift and is ready to perform work. | boolean isOnShift ( ) sent to instance of Person returns TRUE. Person included in set returned by PersonManager services PersonSequence allOnShiftPersons (in Shift aShift) or PersonSequence allAvailableForWorkPersons ( ). |
| AT WORK STATION | A Person is available to perform some manufacturing task; that is, the Person is physically present at his or her work site. | boolean isAtWorkStation ( ) sent to instance of Person returns TRUE. |
| NOT AT WORK STATION | A Person is not available to perform any manufacturing task; that is, the Person is physically absent from his or her work site. | boolean isAtWorkStation ( ) sent to instance of Person returns FALSE. |
| UNASSIGNED TO MACHINES | In this state, a Person has not been assigned to any machines. A Person may or may not be assigned to a job(s). | boolean isAssignedToMachines ( ) or boolean isAssignedToMachine (in Machine aMachine) [for all Machines] sent to instance of Person returns FALSE. Person **NOT** included in set returned by PersonManager services PersonSequence allPersonsAssignedToMachine (in Machine aMachine) [for all Machines] or PersonSequence allAssignedToMachinesPersons ( ). Empty set returned by Person service MachineSequence assignedMachines ( ); |
| ASSIGNED TO MACHINES | In this parent state, a Person has been assigned to one or more machines. | boolean isAssignedToMachines ( ) or boolean isAssignedToMachine (in Machine aMachine) sent to instance of Person returns TRUE. Person included in set returned by PersonManager services PersonSequence allPersonsAssignedToMachine (in Machine aMachine) or PersonSequence allAssignedToMachinesPersons ( ). Non-empty set returned by Person service MachineSequence assignedMachines ( ); |
| AVAILABLE FOR MORE ASSIGNMENTS | In this substate, a Person is available for assignments to more machines. | boolean isAvailableForMoreMachineAssignments ( ) sent to instance of Person returns TRUE. |
| ASSIGNMENT AT CAPACITY EXCEEDED | In this substate, a Person can no longer be assigned to another machine; that is, they are AT CAPACITY with regard to machine assignment. | boolean isAvailableForMoreMachineAssignments ( ) sent to instance of Person returns FALSE. |

| *State* | *Definition* | *Query for State via* |
|---|---|---|
| UNASSIGNED TO JOBS | In this state a Person has not been assigned to any jobs. A Person may or may not be assigned to job(s). | boolean isAssignedToJobs ( ) or boolean isAssignedToJob (in Job aJob) [for all Jobs] sent to instance of Person returns FALSE. Person **NOT** included in set returned by PersonManager services PersonSequence allPersonsAssignedToJob (in Job aJob) [for all Jobs] or PersonSequence allAssignedToJobsPersons ( ). Empty set returned by Person service JobSequence assignedJobs( ); |
| ASSIGNED TO JOBS | In this parent state, a Person has been assigned to one or more job(s). | boolean isAssignedToJobs ( ) or boolean isAssignedToJob (in Job aJob) sent to instance of Person returns TRUE. Person included in set returned by PersonManager services PersonSequence allPersonsAssignedToJob (in Job aJob) or PersonSequence allAssignedToJobsPersons ( ). Non-empty set returned by Person service JobSequence assignedJobs ( ); |
| AVAILABLE FOR MORE ASSIGNMENTS | In this substate, a Person is available for assignments to more jobs. | boolean isAvailableForMoreJobAssignments ( ) sent to instance of Person returns TRUE. Person included in set returned by PersonManager service PersonSequence allPersonsAvailableForJob (in Job aJob). |
| ASSIGNMENT AT CAPACITY EXCEEDED | In this substate, a Person can no longer be assigned to another job; that is, they are AT CAPACITY with regard to job assignment. | boolean isAvailableForMoreJobAssignments ( ) sent to instance of Person returns FALSE. Person not included in set returned by PersonManager service PersonSequence allPersonsAvailableForJob (in Job aJob). |

**Table 2 Person State Transitions**

| # | *Current State* | *Trigger* | *New State* | *Action* | *Comment* |
|---|---|---|---|---|---|
| | non-existent | createPersonWithIdentifier (in string identificationNumber); sent to an instance of PersonManager. | NOT AVAILABLE | | |
| 1 | NOT AVAILABLE | makeOnShift( ); sent to instance of Person. | AVAILABLE | PersonShiftChanged event posted by instance of Person. PersonStateChanged event posted by instance of Person. | Person logged on to shift. Person is assumed to be initially available for work (at work station) and not assigned to machines or jobs. |
| 2 | AT WORK STATION | makeNotAtWorkStation ( ); sent to instance of Person. | NOT AT WORK STATION | PersonStateChanged event posted by instance of Person. | Typically used for situation where Person is not physically present at work site (e.g., on a break). Person is still assigned to machine(s) and/or job(s). Person may be busy with a job. |

| # | *Current State* | *Trigger* | *New State* | *Action* | *Comment* |
|---|---|---|---|---|---|
| 3 | NOT AT WORK STATION | makeAtWorkStation ( ); sent to instance of Person. | AT WORK STATION | PersonStateChanged event posted by instance of Person. | Person is physically present at work site. |
| 4 | UNASSIGNED TO MACHINES | assignToMachine (...); sent to instance of Person. | ASSIGNED TO MACHINES | PersonCapacityChanged event posted by instance of Person. | Person assigned to a machine. For example, a Scheduling Component might assign an operator to a machine. |
| 5 | AVAILABLE FOR MORE ASSIGNMENTS | Sending assignToMachine (...); to instance of Person increments number of machine assignments to AT CAPACITY level. | ASSIGNMENT AT CAPACITY EXCEEDED | PersonStateChanged event posted by instance of Person. | |
| 6 | ASSIGNMENT AT CAPACITY EXCEEDED | Sending deassignFromMachine (...); to instance of Person decrements number of machine assignments below AT CAPACITY level. | AVAILABLE FOR MORE ASSIGNMENTS | PersonStateChanged event posted by instance of Person. | |
| 7 | ASSIGNED TO MACHINES | Sending deassignFromMachine (...); decrements number of machine assignments to nil. | UNASSIGNED TO MACHINES | PersonCapacityChanged event posted by instance of Person. | Person not assigned to any machines. |
| 8 | UNASSIGNED TO JOBS | assignToJob (...); sent to instance of Person. | ASSIGNED TO JOBS | PersonCapacityChanged event posted by instance of Person. | Person assigned to job. Allocates a Person to a manufacturing task. A Person may be allocated to more than one task at a time. |
| 9 | AVAILABLE FOR MORE ASSIGNMENTS | Sending assignToJob (...); to instance of Person increments number of job assignments to AT CAPACITY level. | ASSIGNMENT AT CAPACITY EXCEEDED | PersonStateChanged event posted by instance of Person. | |
| 10 | ASSIGNMENT AT CAPACITY EXCEEDED | Sending deassignFromJob (...); to instance of Person decrements number of job assignments below AT CAPACITY level. | AVAILABLE FOR MORE ASSIGNMENTS | PersonStateChanged event posted by instance of Person. | |
| 11 | ASSIGNED TO JOBS | Sending deassignFromJob (...) decrements number of job assignments to nil. | UNASSIGNED TO JOBS | PersonCapacityChanged event posted by instance of Person. | Person not assigned to any jobs. |
| 12 | AVAILABLE | makeOffShift ( ); sent to instance of Person. | NOT AVAILABLE | PersonShiftChanged event posted by instance of Person. PersonStateChanged event posted by instance of Person. | Person logged off of shift. |

6.3.5 *QualificationData Interface*

| | |
|---|---|
| Interface: | QualificationData |
| Inherited Interface: | NamedEntity |
| Description: | This interface represents the record of training and medical examinations taken by factory personnel. Successful completion of training and medical examinations qualifies personnel for a factory operation (a skill).<br><br>It is anticipated that a Factory Labor application will specialize and instantiate this interface to maintain the training and medical examination histories of factory personnel. For example, CourseQualificationData and MedicalQualificationData sub-interfaces could be defined. These sub-interfaces would exhibit most of the interesting behavior for this interface. For instance, each sub-interface could be compared with a corresponding sub-interface of the Skill Management Component interface SkillRequirement to determine certification (qualification for a skill).<br><br>The attribute QualificationHistory indicates that either a specific course or a medical examination was taken; information regarding the date and time the Person tested for the qualification (when testing for a course or a medical examination occurred); and the results of that test or examination. |
| Exceptions: | None. |
| Published Events: | None. |

Provided Services:

/* Set and get the qualification descriptor (an abbreviated string which serves as a “short-hand” notation for identifying the course or examination; e.g., a course number.) */

```
void setQualificationDesignator (in string designator)
        raises (FrameworkErrorSignal);
string getQualificationDesignator ( ) raises (FrameworkErrorSignal);
```

/* Set and get the history for a qualification event. Historical information indicates that either a specific course or a medical examination was taken; information regarding the date and time the Person tested for the qualification (when testing for a course or a medical examination occurred); and the results of that test or examination. */

```
void setQualificationHistory (in History qualificationHistory)
        raises (FrameworkErrorSignal);
History getQualificationHistory ( ) raises (FrameworkErrorSignal);
```

/* Return the date and time of qualification (when certification was made.) */

```
TimeStamp certificationDate ( ) raises (FrameworkErrorSignal);
```

Contracted Services: None.

Dynamic Model: None.

6.4 *Skill Management Component*

6.4.1 This component supports the management of the skill or skill set required to qualify personnel for a factory operation. Specifically, the skills required in order that personnel can assist a process resource or job in the performance of a process capability. Support is provided through association of skill requirements (training and medical examinations which must be completed) with a specific skill or skill set. Some personnel management aspect of the factory (e.g., training department) will interface with this component to define the training interface and medical examination roadmaps to achieve certification for factory operations.

6.4.2 The Skill Management Component Information Model is shown in Figure 4.

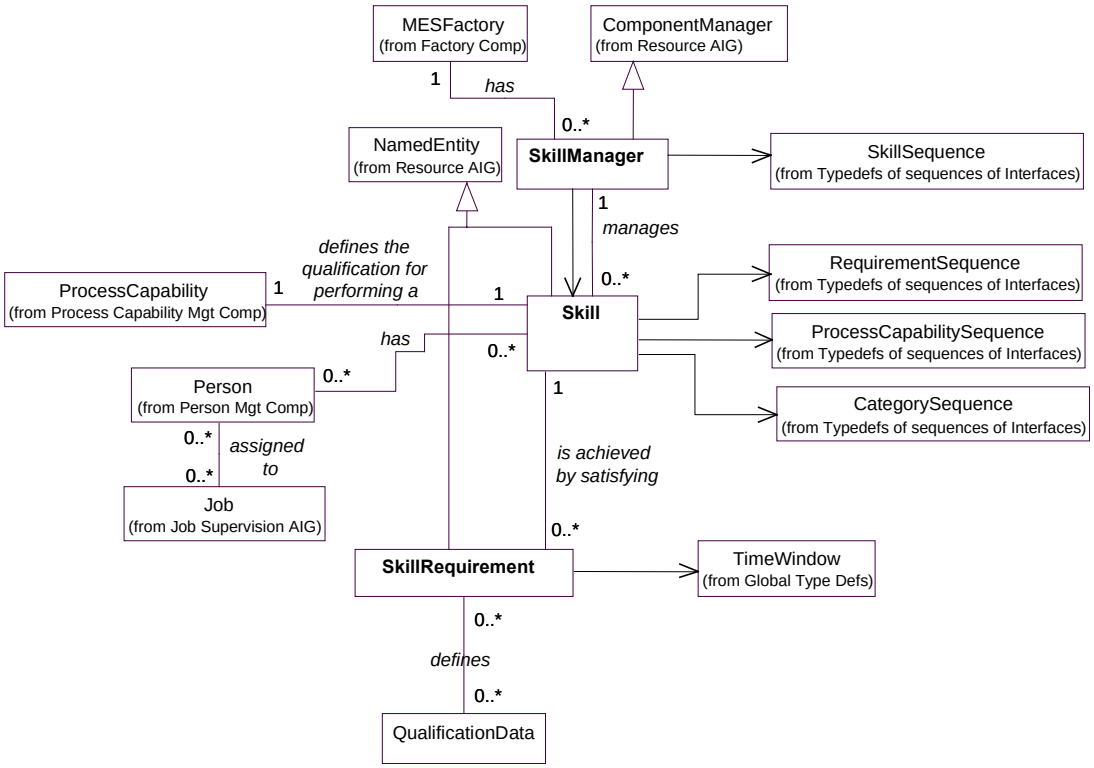


**Figure 4**
**Skill Management Component Information Model**

6.4.3 *SkillManager Interface*

Interface: SkillManager

Inherited Interface: ComponentManager

Description: This is the interface between the Skill Management Component and the rest of the CIM environment. This interface manages the skills or skill sets that personnel must possess before they are permitted to operate factory equipment. There are no limitations to the number of SkillManagers within a factory.

It is anticipated that a Factory Labor application would maintain the appropriate order (if required) of the qualifications related to a specific skill.

Exceptions:

/* This signal is raised with attempted creation of a duplicated Skill name. */

```
exception SkillDuplicateSignal {string skillName;};
```

/* This signal is raised when a removal operation fails because the Skill specified by the argument could not be located. */

```
exception SkillNotAssignedSignal {string skillName;};
exception SkillRemovalFailedSignal {string skillName;};
```

/* This signal is raised when a retrieval operation fails because the Skill specified by the argument could not be found. */

```
exception SkillNotFoundSignal {string skillName;};
```

Published Events: None.

Provided Services:

/* Create an instance of Skill and add to the list managed by this SkillManager. */

```
Skill createSkillNamed (in string skillName)
        raises (FrameworkErrorSignal, SkillDuplicateSignal);
```

/* Delete a specific skill from the list managed by this SkillManager. */

```
void removeSkillNamed (in string skillName)
        raises (FrameworkErrorSignal, SkillNotAssignedSignal,
                SkillRemovalFailedSignal);
```

/* List the skills managed by this SkillManager. */

```
SkillSequence skills ( ) raises (FrameworkErrorSignal);
```

/* Search for an instance of Skill using its name. Raises an exception if the Skill is not found. */

```
Skill findSkillNamed (in string skillName)
        raises (FrameworkErrorSignal, SkillNotFoundSignal );
```

Contracted Services: None.

Dynamic Model: Same as inherited interface.

6.4.4 *Skill Interface*

Interface: Skill

Inherited Interface: NamedEntity

Description: This interface describes the specific Skill or set of Skills required to be held by personnel pursuant to performing factory operations. Specifically, a Skill is required in order that a Person can assist a process resource or job in the performance of a process capability. Each Skill is associated with one or more medical examination and/or training course requirements. A Skill may be associated with one or more process capabilities.

A Skill may represent a singular entity, it may categorize other Skills, or it might be categorized with other Skills. It is anticipated that a Factory Labor training application would maintain the appropriate order (if required) of a set of Skills.

Exceptions:

/* This signal is raised when a definition operation fails because the SkillRequirement specified already exists. */

```
exception SkillRequirementDuplicateSignal {string requirementName;};
```

/* This signal is raised when a removal operation fails because the SkillRequirement specified by the argument could not be located. */

```
exception SkillRequirementRemovalFailedSignal {SkillRequirement aRequirement;};
exception SkillRequirementNotFoundSignal {SkillRequirement aRequirement;};
```

/* This signal is raised when a retrieval operation fails because the SkillRequirement specified by the argument could not be located. */

```
exception SkillRequirementNotFoundSignal {string requirementName;};
```

Published Events: None.

Provided Services:

/* Get and set the Skill categories to which this Skill is associated. */

```
void setSkillCategories (in CategorySequence Categories)
        raises (FrameworkErrorSignal);
CategorySequence getSkillCategories ( ) raises (FrameworkErrorSignal);
```

/* Define a training course or medical examination necessary for obtaining an instance of Skill. */

```
SkillRequirement createSkillRequirementNamed (in string requirementName)
        raises (FrameworkErrorSignal, SkillRequirementDuplicateSignal);
```

/* Delete a training course or medical examination required for an instance of Skill. */

```
void removeSkillRequirement (in SkillRequirement aRequirement)
        raises (FrameworkErrorSignal, SkillRequirementRemovalFailedSignal,
                SkillRequirementNotFoundSignal);
```

/* Search for an instance of SkillRequirement using its name. Exception if the SkillRequirement is not found. */

```
SkillRequirement findSkillRequirementNamed (in string requirementName)
        raises (FrameworkErrorSignal, SkillRequirementNotFoundSignal);
```

/* Return the set of ProcessCapabilities to which this Skill is associated. */

```
ProcessCapabilitySequence processCapabilities ( ) raises (FrameworkErrorSignal);
```

/* List the training course or medical examination requirements that must be completed in order to be qualified for an instance of Skill. */

```
RequirementSequence requirements ( ) raises (FrameworkErrorSignal);
```

Contracted Services: None.

Dynamic Model: None.

6.4.5 *SkillRequirement Interface*

Interface: SkillRequirement

Inherited Interface: NamedEntity

Description: This interface defines training and medical examinations required to achieve a specific skill. The collection of these requirements associated with a specific skill constitutes the “roadmap” for acquiring the skill.

It is anticipated that a Factory Labor training application will specialize and instantiate this interface to maintain the training and medical examination requirements for skills. For example, CourseSkillRequirement and MedicalSkillRequirement sub-interface could be defined.

Exceptions: None.

Published Events: None.

Provided Services:

/* Set and get the complete description of this requirement. The requirement is either a training course which must be successfully completed or a medical examination that must be taken and passed. */

```
void setDescription (in string description) raises (FrameworkErrorSignal);
string getDescription ( ) raises (FrameworkErrorSignal);
```

/* Set and get the requirement descriptor. (An abbreviated string which serves as a “short-hand” notation for identifying a training course or examination; e.g., “OPS-ORIENT-1.”) */

```
void setDesignator (in string designator) raises (FrameworkErrorSignal);
string getDesignator ( ) raises (FrameworkErrorSignal);
```

/* Set and get the required training course grade which must be received or medical examination result which must be achieved for the requirement. */

```
void setRequiredResult (in string result) raises (FrameworkErrorSignal);
string getRequiredResult ( ) raises (FrameworkErrorSignal);
```

/* Return the information pertaining to the period of time in which this requirement is in effect. The information includes a starting time and a duration, from which expiration date can be computed. */

```
TimeWindow effectivity ( ) raises (FrameworkErrorSignal);
```

Contracted Services: None.

Dynamic Model: None.

# APPENDIX 1
# COMPLETE IDL FOR FACTORY LABOR

NOTE: The material in this appendix is an official part of SEMI E86 and was approved by full letter ballot procedures on April 15, 1999.

```
// CIM Framework References

// ********** Forward references **********

interface Job ;
interface Machine ;
interface Person ;
interface ProcessCapability ;
interface QualificationData ;
interface Resource ;
interface SkillRequirement ;
interface Skill ;

// ********** Global Type Definitions **********

typedef struct HistoryEvent_body {
        string subject ;
} HistoryEvent ;

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

typedef ulonglong TimeT ;

typedef TimeT TimeStamp ;

struct IntervalT {
        TimeT lower_bound ;
        TimeT upper_bound ;
} ;

typedef IntervalT TimeWindow ;

typedef struct Shift_Structure {
        string name ;
        // ...
} Shift ;

typedef struct PersonResponsibilitystruct {
        string responsibilityCategory ;
        any responsibility ;
} PersonResponsibility;

typedef string PropertyName;
```

```
struct Property
    {
    PropertyName property_name;
    any property_value;
    };
typedef sequence<Property> Properties;

// ---------- Exceptions  ----------

exception FrameworkErrorSignal {
        unsigned long errorCode ;
        any errorInformation ;
} ;

exception InvalidStateTransitionSignal {
} ;

exception SetValueOutOfRangeSignal {
} ;

// ---------- Sequences  ----------

typedef sequence<ProcessCapability> ProcessCapabilitySequence ;
typedef stringSequence CategorySequence ;
typedef sequence<HistoryEvent> HistoryEventSequence ;
typedef sequence<Job> JobSequence ;
typedef sequence<Machine> MachineSequence ;
typedef sequence<Person> PersonSequence ;
typedef sequence<QualificationData> QualificationDataSequence ;
typedef sequence<Resource> ResourceSequence ;
typedef sequence<SkillRequirement> RequirementSequence ;
typedef sequence<PersonResponsibility> ResponsibilitySequence ;
typedef sequence<Skill> SkillSequence ;
typedef sequence<string> stringSequence ;

// ********** Base Interfaces **********

interface Job {
} ;

// ********** Factory Services **********

interface HistoryCollection {
} ;

interface History {
} ;

// ********** Factory Management **********

interface MESFactory {
} ;
```

```
// ********** Abstract Interfaces **********

interface NamedEntity {
        void setName (in string name) raises (FrameworkErrorSignal);
        string getName ( ) raises (FrameworkErrorSignal);
        boolean isNamed (in string testName) raises (FrameworkErrorSignal);
        HistoryEventSequence getHistoryEvents ( ) raises (FrameworkErrorSignal);
        HistoryCollection getHistoryCollection ( ) raises (FrameworkErrorSignal);
};

interface OwnedEntity : NamedEntity {
        void setOwner (in any owner) raises (FrameworkErrorSignal);
        any getOwner ( ) raises (FrameworkErrorSignal);
};

interface Resource : OwnedEntity {
        void startUp ( ) raises (FrameworkErrorSignal);
        void shutdownNormal ( ) raises (FrameworkErrorSignal);
        void shutdownImmediate ( ) raises (FrameworkErrorSignal);
        string resourceLevel ( ) raises (FrameworkErrorSignal);
        string nameQualifiedTo (in string resourceLevel)
                raises (FrameworkErrorSignal);
        ResourceSequence subresources ( ) raises (FrameworkErrorSignal);
        boolean isAvailable ( );
        boolean isNotAvailable ( );
};

interface ComponentManager : Resource {
        void makeRegistered (in MESFactory aFactory)
                raises (FrameworkErrorSignal,InvalidStateTransitionSignal);
        void makeNotRegistered (in MESFactory aFactory)
                raises (FrameworkErrorSignal,InvalidStateTransitionSignal);
        void makeStartingUp ( )
                raises (FrameworkErrorSignal,InvalidStateTransitionSignal);
        void makeShuttingDown ( )
                raises (FrameworkErrorSignal,InvalidStateTransitionSignal);
        void makeStopped ( )
                raises (FrameworkErrorSignal,InvalidStateTransitionSignal);
        boolean isStopped ( ) raises (FrameworkErrorSignal);
        boolean isStartingUp ( ) raises (FrameworkErrorSignal);
        boolean isShuttingDown ( ) raises (FrameworkErrorSignal);
        boolean isNotRegistered ( ) raises (FrameworkErrorSignal);
        boolean isRegistered ( ) raises (FrameworkErrorSignal);
};

// Person Management Component

module PersonManagement {

// -------------------------  PersonManager  -------------------------

interface PersonManager : ComponentManager {
```

```
exception PersonDuplicateSignal {string identificationNumber;};
exception PersonNotFoundSignal {string identificationNumber;};
exception PersonNotAssignedSignal {Person unknownPerson;};
exception PersonRemovalFailedSignal { };

const string PersonLifecycleSubject =
"/PersonManagement/PersonManager/PersonLifecycle" ;
typedef struct PersonLifecycleFilters_Structure {
        Property        name;           // "Name", aPerson's name
        Property        lifecycleEvent; // "LifecycleEvent", LifecycleState
} PersonLifecycleFilters;
typedef struct PersonLifecycleEvent_Structure {
        string                  eventSubject;
        TimeStamp                       eventTimeStamp;
        PersonLifecycleFilters          eventFilterData;
        Properties                      eventNews;
        Person                  aPerson;
                // Except when "Delete" where value will be nil
} PersonLifecycleEvent;

const string PersonCapacityChangedSubject =
"/PersonManagement/Person/CapabilityChanged" ;
typedef struct PersonCapacityChangedFilters_Structure {
        Property        name;           // "Name", aPerson's name
} PersonCapacityChangedFilters;
typedef struct PersonCapacityChangedEvent_Structure {
        string                  eventSubject;
        TimeStamp                       eventTimeStamp;
        PersonCapacityChangedFilters    eventFilterData;
        Properties                      eventNews;
        Person                  aPerson;
} PersonCapacityChangedEvent;

const string PersonStateChangedSubject =
        "/PersonManagement/Person/StateChanged";
enum PersonState {PersonDefined, PersonOffShift, PersonOnShift,
PersonAvailableForWork, PersonNotAvailableForWork, PersonUnassignedToMachines,
PersonAssignedToJobs, PersonAssignedToMachines,
PersonAvailableForMoreAssignments, PersonAssignmentAtCapacityExceeded,
PersonIdleWithJob, PersonBusyWithJob };
typedef struct PersonStateChangedFilters_Structure {
        Property        name;           // "Name", aPerson's name
        Property        formerState;    // "FormerState", PersonState
        Property        newState;       // "NewState", PersonState
} PersonStateChangedFilters;
typedef struct PersonStateChangedEvent_Structure {
        string                  eventSubject;
        TimeStamp                       eventTimeStamp;
        PersonStateChangedFilters       eventFilterData;
        Properties                      eventNews;
        Person                  aPerson;
} PersonStateChangedEvent;

const string PersonShiftChangedSubject =
```

```
                "/PersonManagement/Person/ShiftChanged";
        typedef struct PersonShiftChangedFilters_Structure {
                Property        name;           // "Name", aPerson's name
                Property        fromShift;      // "FromShift", former shift
                Property        toShift;        // "ToShift", new shift
        } PersonShiftChangedFilters;
        typedef struct PersonShiftChangedEvent_Structure {
                string                  eventSubject;
                TimeStamp                       eventTimeStamp;
                PersonShiftChangedFilters       eventFilterData;
                Properties                      eventNews;
                Person                  aPerson;
        } PersonShiftChangedEvent;

        Person createPersonWithIdentifier (in string identificationNumber)
                raises (FrameworkErrorSignal, PersonDuplicateSignal);
        void removePerson (in Person aPerson)
                raises (FrameworkErrorSignal,PersonRemovalFailedSignal,
                PersonNotAssignedSignal);
        PersonSequence allPersons ( ) raises (FrameworkErrorSignal);
        Person findByIdentifier (in string identificationNumber)
                raises (FrameworkErrorSignal, PersonNotFoundSignal);
        PersonSequence allOnShiftPersons (in Shift aShift)
                raises (FrameworkErrorSignal);
        PersonSequence allAvailableForWorkPersons ( )
                raises (FrameworkErrorSignal);
        PersonSequence allNotAvailableForWorkPersons ( )
                raises (FrameworkErrorSignal);
        PersonSequence allPersonsWithSkill (in Skill aSkill)
                raises (FrameworkErrorSignal);
        PersonSequence allPersonsAvailableForMachine (in Machine aMachine)
                raises (FrameworkErrorSignal);
        PersonSequence allPersonsAssignedToMachine (in Machine aMachine)
                raises (FrameworkErrorSignal);
        PersonSequence allAssignedToMachinesPersons ( )
                raises (FrameworkErrorSignal);
        PersonSequence allPersonsQualifiedForJob (in Job aJob)
                raises (FrameworkErrorSignal);
        PersonSequence allPersonsAvailableForJob (in Job aJob)
                raises (FrameworkErrorSignal);
        PersonSequence allPersonsAssignedToJob (in Job aJob)
                raises (FrameworkErrorSignal);
        PersonSequence allAssignedToJobsPersons ( )
                raises (FrameworkErrorSignal);

}; // end PersonManager

// -------------------------  Person  -------------------------

interface Person : Resource {

        exception QualificationDuplicateSignal {
                QualificationData qualificationInformation;};
```

```
exception QualificationNotAssignedSignal {string identificationNumber;};
exception QualificationNotFoundSignal {string identificationNumber;};
exception ResponsibilityDuplicateSignal {any responsibility ;
        string responsibilityCategory;};
exception ResponsibilityCategoryNotFoundSignal {
        string responsibilityCategory;};
exception ResponsibilityCategoryRemovalFailedSignal { };
exception ResponsibilityRemovalFailedSignal {
        any responsibility;
        string responsibilityCategory;};
exception SkillDuplicateSignal {Skill aSkill;};
exception SkillRemovalFailedSignal {Skill aSkill;};
exception SkillNotAssignedSignal {Skill aSkill;};
exception SkillNotFoundSignal {Skill aSkill;};

void setDepartment (in string department)
        raises (SetValueOutOfRangeSignal, FrameworkErrorSignal);
string getDepartment ( ) raises (FrameworkErrorSignal);
void setFullname (in string fullname)
        raises (SetValueOutOfRangeSignal, FrameworkErrorSignal);
string getFullname ( ) raises (FrameworkErrorSignal);
void setIdentificationNumber (in string idNumber)
        raises (SetValueOutOfRangeSignal, FrameworkErrorSignal);
string getIdentificationNumber ( ) raises (FrameworkErrorSignal);
void setMaximumAssignedJobs (in long maximumJobs)
        raises (SetValueOutOfRangeSignal, FrameworkErrorSignal);
long getMaximumAssignedJobs ( ) raises (FrameworkErrorSignal);
void setMaximumAssignedMachines (in long maximumMachine)
        raises (SetValueOutOfRangeSignal, FrameworkErrorSignal);
long getMaximumAssignedMachines ( ) raises (FrameworkErrorSignal);
void setShift (in Shift aShift)
        raises (SetValueOutOfRangeSignal, FrameworkErrorSignal);
Shift getShift ( ) raises (FrameworkErrorSignal);

any addResponsibility_toCategory (in any responsibility,
in string responsibilityCategory)
        raises (FrameworkErrorSignal, ResponsibilityDuplicateSignal,
                ResponsibilityCategoryNotFoundSignal );

void removeResponsibility_fromCategory (in any responsibility,
in string responsibilityCategory)
        raises (FrameworkErrorSignal, ResponsibilityRemovalFailedSignal,
                ResponsibilityCategoryRemovalFailedSignal);

ResponsibilitySequence responsibilities ( ) raises (FrameworkErrorSignal);
void addSkill (in Skill aSkill)
        raises (FrameworkErrorSignal, SkillDuplicateSignal);
void removeSkill (in Skill aSkill)
        raises (FrameworkErrorSignal, SkillRemovalFailedSignal,
        SkillNotFoundSignal);
SkillSequence skills ( ) raises (FrameworkErrorSignal);
void addQualification (in QualificationData qualificationInformation)
        raises (FrameworkErrorSignal, QualificationDuplicateSignal);
```

```
        void removeQualification (in QualificationData qualificationInformation)
                raises (FrameworkErrorSignal, QualificationNotAssignedSignal);
        QualificationData findQualificationByIdentifier (in string identifier)
                raises (FrameworkErrorSignal, QualificationNotFoundSignal);
        QualificationDataSequence qualifications ( )
                raises (FrameworkErrorSignal);
        ProcessCapabilitySequence processCapabilities ( )
                raises (FrameworkErrorSignal);
        void makeOnShift ( ) raises (FrameworkErrorSignal);
        void makeOffShift ( ) raises (FrameworkErrorSignal);
        void makeAtWorkStation ( )
                raises (FrameworkErrorSignal, InvalidStateTransitionSignal );
        void makeNotAtWorkStation ( )
                raises (FrameworkErrorSignal, InvalidStateTransitionSignal );
        boolean isOnShift ( ) raises (FrameworkErrorSignal);
        boolean isAvailableForWork ( ) raises (FrameworkErrorSignal);
        boolean isAtWorkStation ( ) raises (FrameworkErrorSignal);
        boolean hasSkill (in Skill aSkill) raises (FrameworkErrorSignal);
        boolean hasSkills ( ) raises (FrameworkErrorSignal);
        boolean isAvailableForMachine (in Machine aMachine)
                raises (FrameworkErrorSignal);
        void assignToMachine (in Machine aMachine) raises (FrameworkErrorSignal);
        void deassignFromMachine (in Machine aMachine)
                raises (FrameworkErrorSignal);
        boolean isAssignedToMachine (in Machine aMachine)
                raises (FrameworkErrorSignal);
        boolean isAssignedToMachines ( ) raises (FrameworkErrorSignal);
        MachineSequence assignedMachines ( ) raises (FrameworkErrorSignal);
        boolean isAvailableForMoreMachineAssignments ( )
                raises (FrameworkErrorSignal);
        boolean isQualifiedForJob (in Job aJob) raises (FrameworkErrorSignal);
        boolean isQualifiedForJobs ( ) raises (FrameworkErrorSignal);
        boolean isAvailableForJob (in Job aJob) raises (FrameworkErrorSignal);
        void assignToJob (in Job aJob) raises (FrameworkErrorSignal);
        void deassignFromJob (in Job aJob) raises (FrameworkErrorSignal);
        boolean isAssignedToJob (in Job aJob) raises (FrameworkErrorSignal);
        boolean isAssignedToJobs ( ) raises (FrameworkErrorSignal);
        JobSequence assignedJobs ( ) raises (FrameworkErrorSignal);
        boolean isAvailableForMoreJobAssignments ( )
                raises (FrameworkErrorSignal);

}; // end Person

// -------------------------  QualificationData  -------------------------

interface QualificationData : NamedEntity {

        void setQualificationDesignator (in string designator)
                raises (FrameworkErrorSignal);
        string getQualificationDesignator ( ) raises (FrameworkErrorSignal);
        void setQualificationHistory (in History qualificationHistory)
                raises (FrameworkErrorSignal);
        History getQualificationHistory( ) raises (FrameworkErrorSignal);
```

```
        TimeStamp certificationDate ( ) raises (FrameworkErrorSignal);

}; // end QualificationData

}; // end PersonManagement

// Skill Management Component

module SkillManagement {

// -------------------------  SkillManager  -------------------------

interface SkillManager : ComponentManager {

        exception SkillDuplicateSignal {string skillName;};
        exception SkillNotAssignedSignal {string skillName;};
        exception SkillRemovalFailedSignal {string skillName;};
        exception SkillNotFoundSignal {string skillName;};

        Skill createSkillNamed (in string skillName)
                raises (FrameworkErrorSignal, SkillDuplicateSignal);
        void removeSkillNamed (in string skillName)
                raises (FrameworkErrorSignal, SkillNotAssignedSignal,
                SkillRemovalFailedSignal);
        SkillSequence skills ( ) raises (FrameworkErrorSignal);
        Skill findSkillNamed (in string skillName)
                raises (FrameworkErrorSignal, SkillNotFoundSignal );

}; // end SkillManager

// -------------------------  Skill  -------------------------

interface Skill : NamedEntity {

        exception SkillRequirementDuplicateSignal {string requirementName;};
        exception SkillRequirementRemovalFailedSignal
                {SkillRequirement aRequirement;};
        exception SkillRequirementNotFoundSignal {string requirementName;};

        void setSkillCategories (in CategorySequence Categories)
                raises (FrameworkErrorSignal);
        CategorySequence getSkillCategories ( ) raises (FrameworkErrorSignal);
        SkillRequirement createSkillRequirementNamed (in string requirementName)
                raises (FrameworkErrorSignal, SkillRequirementDuplicateSignal);
        void removeSkillRequirement (in SkillRequirement aRequirement)
                raises (FrameworkErrorSignal, SkillRequirementRemovalFailedSignal,
                SkillRequirementNotFoundSignal);
        SkillRequirement findSkillRequirementNamed (in string requirementName)
                raises (FrameworkErrorSignal, SkillRequirementNotFoundSignal);
        ProcessCapabilitySequence processCapabilities ( )
                raises (FrameworkErrorSignal);
        RequirementSequence requirements ( ) raises (FrameworkErrorSignal);
```

```
}; // end Skill

// -------------------------  SkillRequirement  -------------------------

interface SkillRequirement : NamedEntity {

        void setDescription (in string description) raises (FrameworkErrorSignal);
        string getDescription ( ) raises (FrameworkErrorSignal);
        void setDesignator (in string designator) raises (FrameworkErrorSignal);
        string getDesignator ( ) raises (FrameworkErrorSignal);
        void setRequiredResult (in string result) raises (FrameworkErrorSignal);
        string getRequiredResult ( ) raises (FrameworkErrorSignal);
        TimeWindow effectivity ( ) raises (FrameworkErrorSignal);

}; // end SkillRequirement

}; // end SkillManagement
```

# SEMI E87-0705
# SPECIFICATION FOR CARRIER MANAGEMENT (CMS)

This specification 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 September 1999; previously published March 2004.

## 1 Purpose

1.1 This document provides a standardized behavior for host view communication with production equipment during the coordination, execution, and completion of automated and manual carrier transfers to and from the equipment and, if it exists, its internal buffer space.

## 2 Scope

2.1 This is a standard that covers host and equipment communication for SEMI E15.1 300 mm load ports.

2.2 The scope of this document is to define standards that facilitate the host's knowledge and role in automated and manual carrier transfers, as well as internal buffer equipment carrier transfers. Specifically, this document provides state models and scenarios that define the host interaction with the equipment for the following:

- Carrier transfer between AMHS vehicles and production equipment load ports.
- Carrier transfers to/from production equipment internal buffer space.
- Equipment and load port access mode switching.
- Carrier to load port association.
- CarrierID verification and Carrier slot map verification.

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

## 3 Limitations

3.1 This standard applies to semiconductor equipment with SEMI E15.1 compliant load ports. It may also be applied to other manufacturing equipment that supports automated carrier transfers, and/or contains an internal buffer. This standard is intended to be used for production equipment. It may or may not be applied to other types of equipment. Also, stocker load ports are not addressed by this standard.

## 4 Referenced Standards and Documents

4.1 *SEMI Standards*

SEMI E15 — Specification for Tool Load Port

SEMI E15.1 — Specification for 300 mm Tool Load Port

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

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

SEMI E41 — Exception Management (EM) Standard

SEMI E53 — Event Reporting

SEMI E62 — Provisional Specification for 300 mm Front-Opening Interface Mechanical Standard (FIMS)

SEMI E84 — Specification for Enhanced Carrier Handoff Parallel I/O Interface

SEMI E90 — Specification for Substrate Tracking

SEMI E99 — The Carrier ID Reader/Writer Functional Standard: Specification of Concepts, Behavior, and Services

**NOTICE**: As listed or revised, all documents cited shall be the latest publications of adopted standards.

## 5 Terminology

5.1 *Abbreviations and Acronyms*

5.1.1 *AGT* — Automated Guided Transport

5.1.2 *AGV* — Automated Guided Vehicle

5.1.3 *AMHS* — Automated Material Handling System

5.1.4 *FIMS* — Front-Opening Interface Mechanical Standard

5.1.5 *FOUP* — Front Opening Unified Pod

5.1.6 *GEM* — Generic Equipment Model

5.1.7 *OHT* — Overhead Hoist Transport

5.1.8 *PGV* — Person Guided Vehicle

5.1.9 *PIO* — Parallel Input/Output Interface

5.1.10 *RGT* — Rail Guided Transport

5.1.11 *RGV* — Rail Guided Vehicle

5.2 *Definitions*

5.2.1 *Automated Material Handling System* — an automated system to store and transport materials within the factory.

5.2.2 *automation* — the degree to which activities of machines or production systems are self-acting. In this standard automation provides methods that will reduce the amount of operator intervention required.

5.2.3 *buffer* — a set of one or more locations for holding carriers at/inside the production equipment.

5.2.4 *carrier* — a container, such as a FOUP or open cassette, with one or more positions for holding substrates.

5.2.5 *CarrierID* — a readable and unique identifier for the carrier.

5.2.6 *CarrierID read* — the process of the equipment reading the CarrierID from the carrier.

5.2.7 *carrier ID tag (tag, ID tag)* — a physical device for storing Carrier ID and other information. There are two basic types of tags, read-only tags and read/write tags. [SEMI E99]

5.2.8 *Collection Event* — a collection event is an event (or grouping of related events) on the equipment that is considered to be significant to the host.

5.2.9 *docked position* — the position where the carrier is ready for substrate extraction or insertion.

5.2.10 *FIMS port* — the substrate access port where the FOUP is opened and closed.

5.2.11 *fixed buffer equipment* — production equipment that has only fixed load ports and no internal buffer for carrier storage. Substrates are loaded and unloaded directly from the carrier at the load port for processing.

5.2.12 *host* — the factory computer system or an intermediate system that represents the factory and the user to the equipment.

5.2.13 *internal buffer* — a set of locations within the equipment to store carriers. These locations exclude load ports.

5.2.14 *internal buffer equipment* — equipment that uses an internal buffer.

5.2.15 *load* — the operation of placing a carrier on a load port.

5.2.16 *load port* — the interface location on the equipment where carriers are loaded and unloaded.

5.2.17 *object instantiation* — the act of storing of information related to a physical or logical entity so that it can be recalled on demand based on its public identifier.

5.2.18 *on-line equipment* — equipment that is con-nected to, and able to communicate fully with, the host.

5.2.19 *process equipment* — equipment used to produce product, such as semiconductor devices. This excludes metrology and material handling equipment.

5.2.20 *production equipment* — equipment used to produce product, such as semiconductor devices, including substrate sorting, process, and metrology equipment and excluding material handling equipment.

5.2.21 *properties* — a set of name value pairs assigned to an object or used in a service message to include additional information about the object (i.e., carrier, port, etc.).

5.2.22 *re-initialization* — a process where production equipment is either powered off then on or when some kind of hardware or software reset is initiated to cause the equipment to reset and possibly reload its software. On production equipment that contains some kind of mass storage device this can also be called a “reboot”.

5.2.23 *read position* — any position on a load port or in an internal buffer from which the tag on a carrier can be read.

NOTE 1: This position may vary on any particular equipment depending on the read technology selected by the end user. Some technology/load ports may allow the carrier to be moved during reading. Equipment may have more than one read position.

5.2.24 *single communication connection* — exactly one physical connection using exactly one logical session and a standard set of messages.

5.2.25 *slot map* — the information that relates which slots in a carrier hold substrates, both correctly and incorrectly.

5.2.26 *slot map read* — the process of the equipment reading the slot map for substrate position and placement within the carrier.

5.2.27 *standard message set* — messages conforming to standard message specifications.

5.2.28 *substrate* — material held within a carrier. This can be product, or durables such as reticles.

5.2.29 *substrate port* — the carrier location from which substrates are accessed by the equipment.

5.2.30 *transfer unit* — maximum number of carriers allowed in a specific transfer service:

- AA is the maximum number of carriers allowed for acquisition at the transfer source.
- BB is the maximum number of carriers allowed for deposit at the transfer destination.
- CC is the maximum number of carriers allowed for transfer in one transport vehicle.

The transfer unit is the minimum of AA, BB, and CC.

5.2.31 *undocked* — the status of a carrier on a load port or in an internal buffer that is not at the docked position.

5.2.32 *unload* — the operation of removing a carrier from a load port.

5.2.33 *write position* — any position on a load port or in an internal buffer from which the tag on a carrier can be written to. This position may vary on any particular equipment depending on the write technology selected by the end user. Some technology/load ports may allow the carrier to be moved during writing. The read position and the write position may or may not be the same position.

## 6 Requirements

6.1 Carrier Management Standard (CMS) compliant equipment is required to provide certain capabilities defined by other standards: accessibility to status infor-mation, event reporting, alarm management, and equipment control. These requirements shall be satisfied through compliance to the following sets of standards:

6.2 *Generic Equipment Model Standard (GEM) SEMI E30*

- Event Notification

- Status Data Collection
- Equipment Constants
- Alarm Management
- Equipment Control

6.3 *Object-Based Standards*

- Object Services Standard (SEMI E39)
- Event Reporting (SEMI E53)
- Exception Management (SEMI E41)

## 7 Conventions

7.1 *Objects*

7.1.1 Whenever the equipment is required to know about specific kinds of entities, and required to manage information concerning these entities, it is useful to treat these entities as objects that comply with the basic requirements of SEMI E39 Object Services Standard (OSS). This is especially true whenever there are a large number of objects of a given type or when the entities are transient rather than permanent. In both cases, it is difficult to describe a general way for the host and equipment to specify which particular entity is referenced and to get information related only to a specific one out of many.

7.1.2 By defining these entities as objects that comply with OSS, it is only necessary for the host to specify the type of object and its specific identifier in order to inquire about one or more properties of the specific entity of interest.

7.1.3 *Object Properties*

7.1.3.1 A property (attribute) is information about an individual object that is presented as a name/value pair. The name is a formally reserved text string that represents the property, and the value is the current setting for that property.

7.1.3.2 Properties shall be accessible to the host via the service GetAttr for the Carrier object. Using SEMI E39 Object Services Standard, for example, it is possible to:

- get the list of IDs for the current carriers at the equipment, and
- get the specified properties for one or more individual carriers.

7.1.4 *Rules for Object Properties*

- Attributes with RO access can not be changed using SetAttr service as defined in OSS.
- Attributes with RW access can be changed using SetAttr service as defined in OSS.
- Additional attributes may be specified by the user or the equipment supplier by using an attribute name starting with “UD” (User Defined). Care should be taken to ensure the name of the attribute is unique.

7.1.5 *Object Attribute Table*

7.1.5.1 The object attribute table is used to list all the attributes related to the defined object as shown below the access is defined as Read only (RO) or Read/Write (RW). The REQD column is used to specify whether the attribute is required for implementation. Finally, the Form column is used to specify the format of that particular attribute.

**Table 1 Object Attribute Table**

| *Attribute Name* | *Definition* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| ObjType | Object type | RO | Y | Text = “Carrier” |

7.2 *State Model Methodology*

7.2.1 A state model has three elements: definitions of each state and sub-state, a diagram of the states and the transitions between states, and a state transition table. The diagram of the state model uses the Harel State Chart notation. An overview of this notation is presented in an Appendix of SEMI E30. The definition of this notation is presented in Science of Computer Programming 8, "Statecharts: A Visual Formalism for Complex Systems", by D. Harel, 1987[1].

7.2.2 *State Model Requirements*

7.2.2.1 The state models included in this standard are a requirement for CMS compliance. A state model consists of a state model diagram, state definitions, and a state transition table. All state transitions in this standard, unless otherwise specified, shall correspond to collection events. More explicitly, there must be a unique collection event for each state transition.

7.2.2.2 Equipment must maintain state models for each of the required state models as defined in this document. Equipment shall maintain individual and unique state models for each logical entity instantiated or physical entity in the equipment that has state models associated with it. The event identifier reported during a particular state transition change for each of these state models shall be shared for all associated state models but unique for each transition. For example, if the equipment has two load ports and the load port state model defines 10 transitions, there must be exactly 10 event identifiers for each load port transfer state model but not 10 for each physical load port. The information identifying the physical entity or logical entity undergoing the transition will be contained within the associated event report.

7.2.2.3 A state model represents the host's view of the equipment, and does not necessarily describe the internal equipment operation. All CMS 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 CMS state model for transition to another state. In this case, the equipment makes the required transition without any additional actions in this situation.

7.2.2.4 Some equipment may need to include additional sub-states other than those in this standard. Additional sub-states may be added, but shall not change the CMS defined state transitions. All expected transitions between CMS states shall occur.

7.2.2.5 Transition tables are provided in conjunction with the state diagrams to explicitly describe the nature of each state transition. A transition table contains columns for Transition number, Previous State, Trigger, New State, Actions, and Comments. The "trigger" (column 3) for the transition occurs while in the "previous" state. The "actions" (column 5) includes a combination of:

- Actions taken upon exit of the previous state,
- Actions taken upon entry of the new state, and
- Actions taken which are most closely associated with the transition.

7.2.2.6 When a state model is defined with multiple AND sub-states, the equipment may report all state entry events with only one collection event. When conditional paths are defined in the state model, it is not necessary to report any state transition(s) until a terminal state is reached at which time each transition used to reach that state is reported.

**Table 2 State Transition Table**

| *Num* | *Previous State* | *Trigger* | *New State* | *Actions* | *Comments* |
|---|---|---|---|---|---|
| | | | | | |

1 Elsevier Science, P. O. Box 945, New York, NY 10159-0945, http://www.elsevier.nl/homepage/browse.htt

7.3 *Services*

7.3.1 Services are functions or methods that may be provided by either the equipment or the host. A service message may be either a request message, which always requires a response, or a notification message that does not require a response.

7.3.2 *Service Message Description*

7.3.2.1 A service message description table defines the parameters used in a service, as shown in the following table:

**Table 3 Service Message Description Table**

| *Service Name* | *Type* | *Description* |
|---|---|---|
| | | |

[#1] Type can be either "N" = Notification or "R" = Request & Response.

7.3.2.2 Notification type messages are initiated by the service provider (e.g., the equipment) and the provider does not expect to get a response from the service user. Request messages are initiated by a service user (e.g., the host). Request messages ask for data or an activity from the provider. Request messages expect a specific response message (no presumption on the message content).

7.3.3 *Service Message Parameter Definition*

7.3.3.1 A service parameter dictionary table defines the description, range, and type for parameters used by services, as shown in the following table:

**Table 4 Service Message Parameter Definition Table**

| *Parameter Name* | *Form* | *Description* |
|---|---|---|
| | | |

[#1] A row is provided in the table for each parameter used on a service.

7.3.4 *Service Message Definition*

7.3.4.1 A service message description table defines the parameters used in a service message. It also describes each message and its cause/effect to the equipment. The columns labeled Req/Ind and Rsp/Conf link the parameters to the direction of the message.

| *Service Parameter* | *Req/Ind* | *Resp/Conf* | *Description* |
|---|---|---|---|
| | | | |

7.3.4.2 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". The receiver may then send a "Response", which the original sender terms the "Confirmation".

7.3.4.3 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). |

7.4 *Alarm Requirements Definition*

7.4.1 An alarm requirements definition table defines the specific set of alarms required by CMS. The table is divided up by equipment configuration, and then by alarm. The danger and affected columns are marked with "X" characters to show each alarm and its possible impact to operators, equipment, and material. The table format is shown in the following example:

| *Equipment* | | *Danger* | | *Affected* | | |
|---|---|---|---|---|---|---|
| *Configuration* | *Alarm Text* | *Potential* | *Imminent* | *Operator* | *Equipment* | *Material* |
| Configuration 1 | Alarm 1 | X | | X | X | X |
| | Alarm 2 | | X | | | X |
| Configuration 2 | Alarm 3 | X | | X | X | |
| | Alarm 4 | X | | | X | X |

## 8 Overview

8.1 CMS defines the behavior, data, and services required for equipment supporting automated carrier transfer. This document provides a standard interface for host/equipment communications regarding the transfer of carriers. The standardized carrier transfer host interface includes not only transfers to and from the external load ports, but also transfers to and from the internal buffer positions on internal buffer type equipment.

8.2 *Single Connection Requirement*

8.2.1 The expectation of the production equipment supplier is that this standard be implemented in conjunction with the GEM interface to their production equipment and without the use of a separate communication connection.

## 9 Load Port

9.1 A load port (port) is used by the factory to load and unload carriers to and from production equipment. A load port may be used as an input load port, an output load port, or as an input/output load port, depending upon equipment type, configuration and/or factory practices. This classification may be fixed or it may be programmable by the user. A load port is generally designed to handle one specific carrier type, such as substrate cassettes, leadframe magazines, SMIF pods, or FOUPs.

9.2 *Load Port Numbering*

9.2.1 The load port number shall be assigned incrementally from the bottom left to bottom right, then top left to top right when facing the front of the equipment. The load port numbering requirement is to provide a common reference base to external entities, such as humans.

9.3 *Carrier Slot Numbering*

9.3.1 The slot numbers for a carrier shall be assigned incrementally from the bottom, starting with "1."

9.4 *Load Port Resource Sharing*

9.4.1 A model of a load port must account for any mechanical assemblies that are either active during carrier transfer or are capable of interacting with the transfer. The load port is responsible for such mechanisms when the load port is in the TRANSFER READY state. If these mechanisms are shared with other load ports, then the sharing must be coordinated.

9.5 *Load Port Transfer State Model*

9.5.1 The purpose of the Load Port Transfer State Model is to define the host view of a carrier transfer, which includes the host interactions with the equipment necessary to transfer carriers to and from equipment load ports. Each load port on the equipment shall maintain an independent instance of this state model.

9.5.2 *Load Port Transfer State Model Diagram*

9.5.2.1 Figure 1 is the diagram for the Load Port Transfer State Model.

9.5.3 *Load Port Transfer State Definitions*

9.5.3.1 *LOAD PORT TRANSFER* — The super state for the IN SERVICE and OUT OF SERVICE states.

9.5.3.2 *OUT OF SERVICE* — Transfer to/from this load port is disabled. A transition to IN SERVICE is required to continue using this load port for transfers.

9.5.3.3 *IN SERVICE* — Transfer to/from this load port is enabled. A transition to OUT OF SERVICE disables the load port for transfer use.

9.5.3.4 *TRANSFER READY* — A sub-state of IN SERVICE. The load port is available for carrier transfer. The transfer can either be manual or automated, and can be a load or an unload. This state contains two sub-states, which are used depending on whether or not a carrier is present on the load port (READY TO LOAD and READY TO UNLOAD).

9.5.3.5 *READY TO LOAD* — A sub-state of TRANSFER READY. When transitioning to the TRANSFER READY state, if a carrier is not present on the specified load port, this is the active sub-state. In this state, the load port is available to be loaded with an external carrier, or with a carrier that is currently located inside the equipment (i.e. internal buffer).

9.5.3.6 *READY TO UNLOAD* — A sub-state of TRANSFER READY. When transitioning to the TRANSFER READY state, if a carrier is present on the specified load port, this is the active sub-state. In this state, the load port is available for unloading of a carrier from the loadport to material handling equipment. When the load port is being used by the equipment, the state shall transition to TRANSFER BLOCKED.

9.5.3.7 *TRANSFER BLOCKED* — The carrier transfer state is neither READY TO LOAD nor READY TO UNLOAD. Because of load port related activity being performed, transfer is not available to/from this load port at this time.

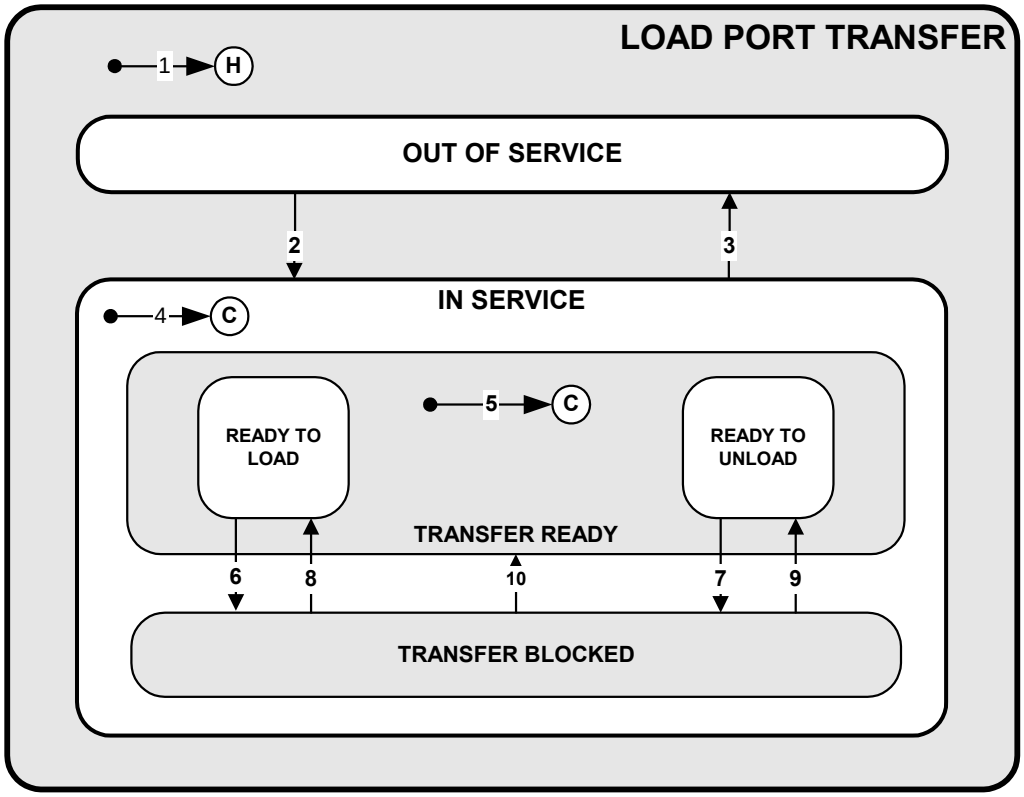


**Figure 2**
**Load Port Transfer State Model Diagram**

9.5.4 *Load Port Transfer State Transition Table*

**Table 5 Load Port Transfer State Transition Definition**

| *Num* | *Previous State* | *Trigger* | *New State* | *Actions* | *Comments* |
|---|---|---|---|---|---|
| 1 | (no state) | System reset. | OUT OF SERVICE<br>or<br>IN SERVICE (History) | | This transition is based on what the current transfer status was prior to system reset.<br>Data required to be available for this event report:<br>PortID<br>PortTransferState |
| 2 | OUT OF SERVICE | The host or an operator has invoked the ChangeServiceStatus service for this load port with a value of IN SERVICE. | IN SERVICE | | Load port is now usable for transfer.<br>Data required to be available for this event report:<br>PortID<br>PortTransferState |
| 3 | IN SERVICE | The host or an operator has invoked the ChangeServiceStatus service for this load port with a value of OUT OF SERVICE. | OUT OF SERVICE | | Load port is now rendered unusable for transfer. Attempted usage of the load port for carrier transfer after the state transition results in an alarm.<br>Data required to be available for this event report:<br>PortID<br>PortTransferState |
| 4 | IN SERVICE | *Service:* The host or an operator has invoked the ChangeServiceStatus service for this load port with a value of IN SERVICE.<br>*System Reset:* This transition can be activated by an equipment re-initialization. | TRANSFER READY<br>or<br>TRANSFER BLOCKED | | This is the default entry into IN SERVICE. The state is TRANSFER BLOCKED if the carrier, or load port, is not available for carrier transfer. Otherwise, the state is TRANSFER READY.<br>Data required to be available for this event report:<br>PortID<br>PortTransferState |
| 5 | TRANSFER READY | *Service:* The host or an operator has invoked the ChangeServiceStatus service for this load port with a value of IN SERVICE.<br>*System Reset:* This transition can be activated by an equipment re-initialization.<br>*Failed Transfer:* If a transfer fails, this transition is activated by transition #10. | READY TO LOAD<br>or<br>READY TO UNLOAD | | When entering the TRANSFER READY state, if a carrier is present, the sub-state is READY TO UNLOAD, else the sub-state is READY TO LOAD.<br>If the state is READY TO LOAD, data required to be available for this event report:<br>PortID<br>If the state is READY TO UNLOAD, data required to be available for this event report:<br>PortID<br>CarrierID<br>PortTransferState |

| *Num* | *Previous State* | *Trigger* | *New State* | *Actions* | *Comments* |
|---|---|---|---|---|---|
| 6 | READY TO LOAD | *Manual*: The equipment recognizes the logical indication of the start of a manual load transfer. This trigger is configurable by the user, examples are included in Table 8.<br>*Automated*: The PIO load transfer is beginning and the PIO "READY" signal is activated (see SEMI E84).<br>*Internal Buffer:* A CarrierOut service has started for this load port. | TRANSFER BLOCKED | | When a CarrierOut service is queued and the equipment load port is currently in the TRANSFER BLOCKED state, the equipment shall keep the load port in the TRANSFER BLOCKED state.<br>Data required to be available for this event report:<br>PortID<br>PortTransferState |
| 7 | READY TO UNLOAD | *Manual*: The equipment recognizes the logical indication of the start of a manual unload transfer. This trigger is configurable by the user, examples are included in Table 8.<br>*Automated*: The PIO unload transfer is beginning and the PIO "READY" signal is activated (see SEMI E84).<br>*Internal Buffer:* A CarrierIn service has started for this load port.<br>*By CarrierReCreate Service:* A CarrierReCreate service command has been issued by host or operator. | TRANSFER BLOCKED | | When a CarrierOut service is queued and the equipment load port is currently in the TRANSFER BLOCKED state, the equipment shall keep the load port in the TRANSFER BLOCKED state.<br>Data required to be available for this event report:<br>PortID<br>PortTransferState |
| 8 | TRANSFER BLOCKED | *Manual:* The carrier unload transfer has completed, and the load port is now empty and ready for load transfer. This is indicated when two conditions are met, the presence signal indicates that no carrier is present and the operator has logically indicated that the transfer is complete.<br>*Automated:* The PIO unload transfer ends with the PIO "COMPT" signal (see SEMI E84).<br>*Internal Buffer:* The carrier has finished its move from the load port into the internal buffer, and no CarrierOut services are queued for this load port. | READY TO LOAD | | A carrier can now be loaded onto the load port from either an external entity, or by the equipment's internal material handling resource.<br>Data required to be available for this event report:<br>PortID<br>PortTransferState |

| *Num* | *Previous State* | *Trigger* | *New State* | *Actions* | *Comments* |
|---|---|---|---|---|---|
| 9 | TRANSFER BLOCKED | *Manual:* Processing for substrates contained within the carrier has completed, or a CancelCarrier/ CancelCarrierAtPort service has been issued, and the carrier has returned to the load/unload position on the load port.<br>*Automated:* Processing for the substrates belonging to the carrier has completed, or a CancelCarrier/CancelCarrier-AtPort service has been received, and the carrier has returned to the load/unload position.<br>*Internal Buffer:* A carrier has completed its move from the internal buffer to the load port. | READY TO UNLOAD | | The carrier on the load port can now be unloaded from the load port to an external entity.<br>Data required to be available for this event report:<br>PortID<br>CarrierID<br>PortTransferState |
| 10 | TRANSFER BLOCKED | The transfer was unsuccessful, and the carrier was not loaded or unloaded. | TRANSFER READY | | The sub-state of TRANSFER READY which is decided by transition #5.<br>Data required to be available for this event report:<br>PortID<br>PortTransferState |

## 10 Carrier Object

10.1 Information about a carrier is encapsulated as an object. This allows the host to exchange information with the equipment about one or more specific carriers using services defined in SEMI E39, Object Services Standard. A carrier has properties (attributes) that are defined in Table 6, Carrier Attribute Definition.

10.2 *Object Instantiation*

10.2.1 The carrier object is a software representation of the carrier in the equipment. Under normal circumstances this object is instantiated by the equipment when the host uses the Bind or Carrier Notification service or when the equipment successfully reads the carrierID from the carrier. A carrier object is instantiated by carrierID read only if there are no currently existing objects with the carrierID just read. A carrier object can also be instantiated by either the ProceedWithCarrier or CancelCarrier Services on an NOT ASSOCIATED port. (This implies a failed carrierID read event.) The ContentMap attribute will be an empty list (a list of zero) when the instantiation is done by CarrierID read. The SlotMap attribute should be a list consisting of all slots enumerated as “UNDEFINED” when the carrier object is instantiated by CarrierID read.

10.2.2 From the host point of view, an object is instantiated if the host is able to query the equipment about that object, it's current state, and other attributes. Once instantiated, the object is considered destroyed (no longer instantiated) if the response to such queries is “unknown object”.

10.2.3 Summary of carrier object instantiation:

1. Bind or Carrier Notification or CarrierReCreate (with PropertiesList) service;
2. CarrierID read with no currently existing carrier objects having the carrierID just read; and
3. ProceedWithCarrier or CancelCarrier Service on an NOT ASSOCIATED port with a carrier.

10.2.4 *Carrier Object Identifier (ObjID)*

10.2.4.1 The purpose of an Object Identifier is to allow references to an object within the system. The object identifier is created when an object is instantiated and should be unchanged or persistent until the end of the object lifecycle. The Object Identifier shall be unique at the equipment during lifecycle of the object. The Carrier ID is the

Carrier Object Identifier. The equipment is responsible for ensuring uniqueness of the Carrier ID prior to instantiation by the bind service.

10.2.5 *Carrier Object Destruction*

10.2.5.1 Normally, the Carrier Object reaches the end of its lifecycle when the carrier is unloaded from the equipment. Abnormally, the Carrier Object reaches the end of its lifecycle when a CancelBind or CancelCarrierNotification service is executed prior to the carrier being loaded, or when an equipment based carrier verification fails following carrier instantiation by the bind service.

10.2.5.2 Summary of carrier object destruction:

1. A carrier is unloaded from the equipment;
2. A CancelBind or CancelCarrierNotification service is received;
3. An equipment based CarrierID verification fails after a carrier object was previously instantiated with a "Bind" service (Equipment initiated CancelBind); and
4. The host or operator has issued a CarrierReCreate service.

10.3 *Carrier Attribute Definitions*

10.3.1 The following table contains the attributes that are of importance to the host and/or the equipment in order to manage the history and the reports about the carrier object.

10.3.2 *REQD Column*

10.3.2.1 All attributes in the following table are required to be associated with the carrier object and are always maintained and updated by the equipment (for example, if the equipment has a waferID reader, the equipment can determine the ContentMap).

10.3.3 *ACCESS Column*

10.3.3.1 Even though a value may be marked as RO (read only), the initial value for the attribute may be provided by the host when attached to either the Bind or ProceedWithCarrier services.

10.3.4 *Carrier Attribute Definition Table*

**Table 6 Carrier Attribute Definition**

| *Attribute Name* | *Definition* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| Capacity | Maximum number of substrates a carrier can hold. | RO | Y | Positive integer. |
| CarrierIDStatus | Current state of the carrier ID verification. | RO | Y | Enumerated:<br>ID NOT READ, [ID] WAITING FOR HOST, ID VERIFICATION OK, ID VERIFICATION FAILED. |
| CarrierAccessingStatus | The current accessing state of the carrier by the equipment. The current substate of the CarrierAccessingStatus state model. | RO | Y | Enumerated:<br>NOT ACCESSED, IN ACCESS, CARRIER COMPLETE, CARRIER STOPPED. |
| ContentMap | Ordered list of lot and substrate identifiers corresponding to slot 1,2,3,…n. | RO | Y | Ordered list of n structures, where n is equal to the value of "Capacity" above, and each structure consists of a LotID and SubstrateID.<br>List of Structure<br>LotID<br>SubstrateID<br>When no Lot ID is provided by the host, the LotID value should be null.<br>When a slot has no substrate or the host does not know substrate identifier, the LotID and SubstrateID value should be null. |

| *Attribute Name* | *Definition* | *Access* | *Reqd* | *Form* |
|---|---|---|---|---|
| LocationID | Identifier of current location. | RO | Y | Text 1 to 80 characters. |
| ObjType | Object type. | RO | Y | Text 1 to 40 characters equal to “Carrier”. |
| ObjID | Object identifier. | RO | Y | Text 1 to 80 characters equal to the CarrierID. |
| SlotMap | Ordered list of slot status as provided by the host and corresponding to slot 1,2,3…n until a successful slot map read, then as read by the equipment. | RO | Y | Ordered list of n, where n is equal to the value of “Capacity” above, each value in the list is enumerated<br>Enumerated<br>Enumerated:<br>UNDEFINED, EMPTY, NOT EMPTY, CORRECTLY OCCUPIED, CROSS SLOTTED, DOUBLESLOTTED. (NOT EMPTY provided for equipment that cannot detect incorrectly slotted substrates). |
| SlotMapStatus | Current state of slot map verification. | RO | Y | Enumerated:<br>SLOT MAP NOT READ, [SLOT] WAITING FOR HOST, SLOT MAP VERIFICATION OK, SLOT MAP VERIFICATION FAILED. |
| SubstrateCount | The number of substrates currently in the carrier. | RO | Y | Non negative integer less than or equal to the Capacity. |
| Usage | The type of material contained in the carrier (i.e., TEST, DUMMY, PRODUCT, FILLER, etc.). | RO | Y | Text as defined by the Equipment. |

[#1] NOT EMPTY is included to indicate presence for equipment that is only able to detect substrate presence but not correct positioning of the substrate slot. For equipment that can detect incorrect positioning such as cross-slotted or double slotted, NOT EMPTY may not be applicable.

10.3.5 *Rules for Carrier Attributes*

- The equipment shall change object attributes, Capacity, ContentMap, SlotMap, Substrate count and Usage, provided by the host. All other attributes, such as LocationID, shall be set and maintained by the equipment.
- The attributes, Capacity, ContentMap, Substrate count and Usage, shall be provided with Bind, CarrierNotification, or ProceedWithCarrier service before or when SlotMap is provided.
- The SlotMap shall be provided with Bind, CarrierNotification, or ProceedWithCarrier to verify CarrierID, when the SlotMap verification is equipment based. And it shall not be provided when the SlotMap verification is host based.
- Carrier properties may be provided before the carrier arrives as part of the Bind service and should be retained until either a CancelBind service is received or the carrier is removed.
- Carrier properties may also be provided by the ProceedWithCarrier service. The carrier properties that are provided by the ProceedWithCarrier service may differ based whether or not the object is instantiated by the service.
- Carrier properties that are required shall be actively updated by the equipment.

10.3.6 *Carrier Location*

10.3.6.1 A carrier location, signified by LocationID, is used for tracking carriers as they move through the equipment. A carrier location is any physical area that is capable of holding a carrier. It is not intended to represent entire mechanisms, which may have a variety of other properties of interest, but only that portion where a Carrier may rest.

10.3.7 *Carrier Location Examples*

10.3.7.1 Carrier Locations include load port locations, substrate port locations, internal buffer locations, as well as grippers, conveyors, and elevators that are used internally for moving the carrier from one fixed location to another.

10.4 *Carrier Location Naming*

10.4.1 Carrier locations shall be assigned a unique name. Information about the carrier location can be ob-tained by querying the CarrierObject for the LocationID or by asking the equipment for the CarrierLocation-Matrix. The text form of the LocationID shall be descriptive of the location. For example, LocationID form for load port load/unload location might be “LPn”, where “n” equal is equal to the load port number (the load port number is determined through the numbering rule in ¶9.1). The LocationID form for the FIMS port location might be “FIMSn”. The LocationID form for a buffer location might be “BUFn”.

10.5 *Load Port Carrier Locations*

10.5.1 For fixed buffer equipment configured to handle FOUPs, a Load Port has two different Carrier Locations. One represents the place where a Carrier is delivered and picked up, while the other represents the place where the Carrier is docked and can be opened.

10.6 *Carriers Between Locations*

10.6.1 When the carrier is traveling from one location to another, the location attribute remains equal to the source location until the carrier movement is complete and the carrier is resting at the new carrier location (the destination location).

10.6.2 *Usage*

10.6.2.1 The Usage parameter indicates the type of substrate the carrier contains. All Usage values are equipment specific values. Some internal buffer equipment manages carriers by establishing logical partitions. This type of equipment shall use the Usage parameter to determine which logical partition where the carrier is held.

10.6.3 *SubstrateCount*

10.6.3.1 The SubstrateCount parameter can be sent to the equipment by the host in either the Bind service or the ProceedWithCarrier service. However the equipment shall update this parameter based on the results of the read slot map operation. Furthermore, the equipment shall update the parameter based on it’s own actions of adding and removing a substrate to and from a carrier. If the equipment does not know the value of SubstrateCount prior to instantiation, the equipment shall instantiate the carrier object with the value of null for SubstrateCount.

10.6.4 *Lot information for ContentMap*

10.6.4.1 Lot is defined in SEMI E90 as a group of one or more substrates of the same type. It is organized external to the equipment. The Lot ID is the identifier of this group. If the equipment is informed of the Lot ID to which a substrate belongs, the equipment must maintain this information.

10.6.5 *Carrier Accessing Status*

10.6.5.1 The CarrierAccessingStatus is used by the host to know whether or not the carrier owned by the equipment can be moved out. If the carrier is within the internal buffer equipment, this status may be used by the host to issue CarrierOut service.

10.7 *Carrier State Model*

10.7.1 The purpose of the Carrier State Model is to define the host’s view of a carrier. The equipment shall maintain a separate and independent state model for each carrier in/at the equipment.

10.7.2 *Carrier State Model Diagram*

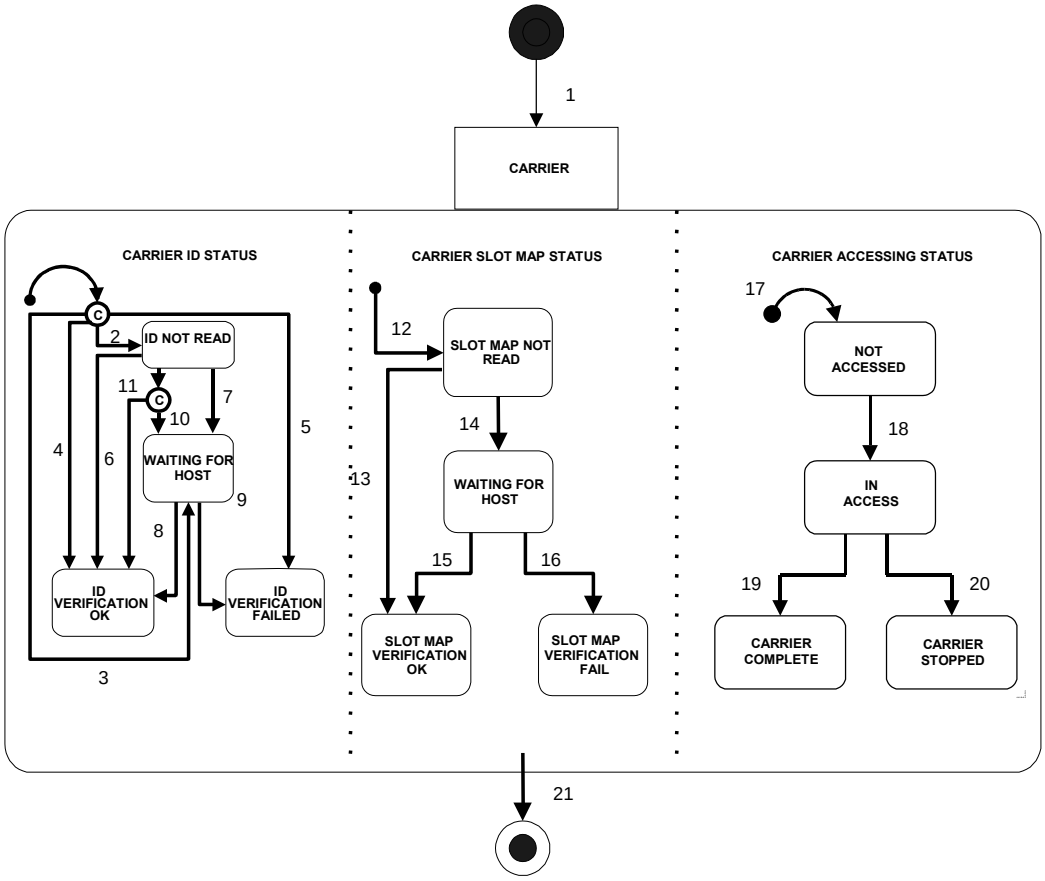


**Figure 3**
**Carrier State Model Diagram**

10.7.3 *Carrier State Definitions*

10.7.3.1 *CARRIER* — The CARRIER state has three ANDed (orthogonal) states: CARRIER ID STATUS, CARRIER SLOT MAP STATUS and CARRIER ACCESSING STATUS.

10.7.3.2 *CARRIER ACCESSING STATUS* — This is a substate of CARRIER and indicates the current accessing status of the carrier. It has four substates, NOT ACCESSED, IN ACCESS, CARRIER COMPLETE, and CARRIER STOPPED. The initial default entry substate is NOT ACCESSED.

10.7.3.2.1 *NOT ACCESSED* — This is a substate of CARRIER ACCESSING STATUS and is active when access by the equipment to the carrier has not been started. The carrier can be moved out.

10.7.3.2.2 *IN ACCESS* — This is a substate of CARRIER ACCESSING STATUS and is active when access by the equipment to the carrier has been started but has not been finished, and the carrier should not be moved out.

10.7.3.2.3 *CARRIER COMPLETE* — This is a substate of CARRIER ACCESSING STATUS and is active when the access by the equipment to the carrier has been finished, and the carrier should be moved out. This is a final state.

10.7.3.2.4 *CARRIER STOPPED* — This is a substate of CARRIER ACCESSING STATUS and is active when the access by the equipment to the carrier has been stopped abnormally, and the carrier should be moved out. This is a final state.

10.7.3.3 *CARRIER ID STATUS* — This is a substate of CARRIER and indicates the current status of the carrier with respect to its identifier. It has four substates, ID NOT READ, WAITING FOR HOST, ID VERIFICATION FAILED, ID VERIFICATION OK. The initial substate is conditional based on information the equipment has

about the carrier. When the carrierID is provided by the Bind or the Carrier Notification service, the carrier object shall be instantiated in the ID NOT READ substate. When the carrierID is provided by the carrier ID reader, the carrier shall be instantiated in the WAITING FOR HOST substate. When the Carrier is instantiated by the ProceedWithCarrier service, the carrier shall be instantiated in the ID VERIFICATION OK substate. Finally when the carrier is instantiated by the CancelCarrier service, the carrier will be instantiated in the ID VERIFICATION FAILED substate.

10.7.3.3.1 *ID NOT READ* — This is a substate of CARRIER ID STATUS. This state is active whenever the CarrierID has not been read by the equipment.

10.7.3.3.2 *ID VERIFICATION FAILED* — This is a substate of CARRIER ID STATUS and is active when the carrierID has failed verification by the host following the CancelCarrier service. This is a final state.

10.7.3.3.3 *ID VERIFICATION OK* — This is a substate of CARRIER ID STATUS and is active as soon as the CarrierID has been accepted. The ID is determined to be accepted by either successful verification by the equipment or the host, or by bypassing ID read because a carrier ID reader is not available and the BypassReadID variable is set to true. This is a final state.

10.7.3.3.4 *WAITING FOR HOST* — This is a substate of CARRIER ID STATUS and is active during the period of time when the carrierID has been read by the equipment successfully or unsuccessfully and has not yet been verified by the host.

10.7.3.4 *CARRIER SLOT MAP STATUS* — This is a substate of CARRIER and indicates the current status of the carrier with respect to its slot map. It has four substates, SLOT MAP NOT READ, WAITING FOR HOST, SLOT MAP VERIFICATION FAILED, SLOT MAP VERIFICATION OK. The initial default entry sub-state is SLOT MAP NOT READ.

10.7.3.4.1 *SLOT MAP NOT READ* — This is a substate of CARRIER SLOT MAP STATUS and is the default entry state. It is active when the Carrier is first loaded at the equipment until the Slot Map has been read successfully by the equipment at the Substrate Port.

10.7.3.4.2 *SLOT MAP VERIFICATION FAIL* — This is a substate of CARRIER SLOT MAP STATUS and is active when the Slot Map has been read by the equipment and has failed verification by the host. This is a final state.

10.7.3.4.3 *SLOT MAP VERIFICATION OK* — This is a substate of CARRIER SLOT MAP STATUS and is active as soon as the slot map has been verified. This is a final state.

10.7.3.4.4 *WAITING FOR HOST* — This is a substate of CARRIER SLOT MAP STATUS and is active when the equipment is waiting for input from the host.

10.7.4 *Carrier State Transition Table*

10.7.4.1 Table 7 indicates the triggers and the expected behavior of the instantiated carrier object.

**Table 7 Carrier State Transition Definition**

| # | *Previous State* | *Trigger* | *New State* | *Actions* | *Comment* |
|---|---|---|---|---|---|
| 1 | (no state) | A carrier is instantiated. | CARRIER | None. | No event is required for this transition |
| 2 | (no state) | *Normal:* A Bind or Carrier Notification service is received. | ID NOT READ | None. | Data required to be available for this event report:<br>CarrierID<br>CarrierIDStatus |

| # | Previous State | Trigger | New State | Actions | Comment |
|---|---|---|---|---|---|
| 3 | (no state) | *Normal:* A carrierID not currently existing at the equipment is successfully read.<br>*Abnormal:* A carrierID is read successfully but an equipment based verification failed. | WAITING FOR HOST | None. | Data required to be available for this event report:<br>CarrierID<br>PortID<br>CarrierIDStatus<br>Normally, this transition will happen after a successful ID read if a bind service has not been issued (host based verification) or abnormally if a bind service is followed by a successful ID read and an unsuccessful equipment based verification. |
| 4 | (no state) | *ID Read fail or UnknownCarrierID Events*: ProceedWithCarrier service is received. | ID VERIFICATION OK | A carrier is instantiated having the carrierID provided by the Proceed WithCarrier service. | Data required to be available for this event report:<br>CarrierID<br>CarrierIDStatus<br>This transition can happen only if a bind service has not been received. |
| 5 | (no state) | *ID Read fail or UnknownCarrierID Events:* A CancelCarrier service is received. | ID VERIFICATION FAIL | A carrier is instantiated having the carrierID provided by the Cancel Carrier service. | Data required to be available for this event report:<br>CarrierID<br>CarrierIDStatus<br>This transition can happen only if a bind service has not been received. |
| 6 | ID NOT READ | Carrier ID is read successfully and the equipment has verified the carrierID successfully. | ID VERIFICATION OK | None. | Data required to be available for this event report:<br>PortID<br>CarrierID<br>CarrierIDStatus |
| 7 | ID NOT READ | Carrier ID is read unsuccessfully. | WAITING FOR HOST | None. | Data required to be available for this event report:<br>PortID<br>CarrierID<br>CarrierIDStatus |
| 8 | WAITING FOR HOST | A ProceedWithCarrier service is received. | ID VERIFICATION OK | None. | Data required to be available for this event report:<br>PortID<br>CarrierID<br>CarrierIDStatus |
| 9 | WAITING FOR HOST | A Cancel Carrier Service is received. | ID VERIFICATION FAIL | None. | Data required to be available for this event report:<br>PortID<br>CarrierID<br>CarrierIDStatus |
| 10 | ID NOT READ | BypassReadID variable is set to FALSE, and a carrier is received when the id reader is not in service or not installed. | WAITING FOR HOST | Wait for ProceedWith-Carrier. | Data required to be available for this event report:<br>PortID<br>CarrierID<br>CarrierIDStatus |

| # | Previous State | Trigger | New State | Actions | Comment |
|---|---|---|---|---|---|
| 11 | ID NOT READ | BypassReadID variable is set to TRUE, and a carrier is received when the id reader is not in service or not installed. | ID VERIFICATION OK | None. | Data required to be available for this event report:<br>PortID<br>CarrierID<br>CarrierIDStatus |
| 12 | (no state) | A carrier is instantiated. | SLOT MAP NOT READ | None. | No event is required for this transition. |
| 13 | SLOT MAP NOT READ | Slot Map is read and verified successfully by the equipment. | SLOT MAP VERIFICATION OK | None. | Data required to be available for this event report:<br>PortID (if valid)<br>CarrierID<br>LocationID<br>CarrierAccessingStatus<br>SlotMapStatus |
| 14 | SLOT MAP NOT READ | *Normal host based verification:* Slot Map is read successfully and the equipment is waiting for host verification.<br>*Equipment based verification fail:* Slot Map is read successfully but equipment based verification has failed.<br>*Slot map read fail:* Slot Map cannot be read.<br>*Abnormal substrate position within the carrier:* The Slot Map read has indicated an abnormal substrate position. | WAITING FOR HOST | Save new slot map in the SlotMap attribute. | Data required to be available for this event report:<br>PortID (if valid)<br>CarrierID<br>LocationID<br>SlotMap (if valid)<br>Reason<br>SlotMapStatus |
| 15 | WAITING FOR HOST | A ProceedWithCarrier service is received. | SLOT MAP VERIFICATION OK | Proceed with the Carrier as instructed. | Data required to be available for this event report:<br>PortID (if valid)<br>CarrierID<br>LocationID<br>SlotMapStatus |
| 16 | WAITING FOR HOST | A CancelCarrier service is received. | SLOT MAP VERIFICATION FAIL | Prepare the Carrier for Unload. | Data required to be available for this event report:<br>PortID (if valid)<br>CarrierID<br>LocationID<br>CarrierAccessingStatus<br>SlotMapStatus |
| 17 | (no state) | A carrier object is instantiated. | NOT ACCESSED | None. | No event is required for this transition |
| 18 | NOT ACCESSED | The equipment starts accessing the carrier. | IN ACCESS | None. | Data required to be available for this event report:<br>CarrierID<br>CarrierAccessingStatus |
| 19 | IN ACCESS | The equipment finishes accessing the carrier normally. | CARRIER COMPLETE | None. | Data required to be available for this event report:<br>CarrierID<br>CarrierAccessingStatus |

| # | Previous State | Trigger | New State | Actions | Comment |
|---|---|---|---|---|---|
| 20 | IN ACCESS | The equipment finishes accessing the carrier abnormally. | CARRIER STOPPED | None. | Data required to be available for this event report:<br>CarrierID<br>CarrierAccessingStatus |
| 21 | CARRIER | *Normal:* The carrier is unloaded from the equipment.<br>*Abnormal by service:* CancelBind or CancelCarrierNotification service is received prior to the carrier load.<br>*Abnormal by equipment:* An equipment based verification fails and the equipment performs a self-initiated CancelBind service. | (no state) | The equipment destroys the instance of this carrier object. | Data required to be available for this event report:<br>CarrierID |

[#1] Only one collection event report is required when entering the Carrier State Model (instantiating a carrier object). This event report shall include the entry state of the all the substates of Carrier State Model, (including CARRIER ID STATUS substate and the CARRIER SLOT MAP STATUS substate).

10.7.5 *Slot Map Read Details*

10.7.5.1 The Slot Map shall be read on all production equipment prior to removal of substrates from the carrier.

10.7.6 *Carrier Read Failure* — A carrier read failure occurs when the carrier ID reader is present, in service, and reports that it is unable to read the ID of a carrier. This represents a transient random failure rather than a steady condition.

10.7.7 *Bypass Read ID* — A carrier ID reader may be unavailable: either out of service, not installed, or otherwise malfunctioning and unable to execute a read operation. This represents a steady condition that often is known in advance. The equipment shall provide a user-configurable variable BypassReadID used to bypass verification of the carrier ID when the carrier ID reader is unavailable or not installed and the loadport is ASSOCIATED. BypassReadID is not used to bypass the carrier ID reader. In this case, the carrier object is instantiated in the ID NOT READ state, and when the carrier is received, the state model transitions to either WAITING FOR HOST or ID VERIFICATION OK, depending upon whether BypassReadID is FALSE (the default value) or TRUE. When TRUE, then the Carrier ID received in the Bind is used automatically. Otherwise, the carrier transitions to WAITING FOR HOST and waits for the host to send a ProceedWithCarrier. The ID used will be the ID included with the ProceedWithCarrier.

## 11 Access Mode

11.1 *Access Mode State Model*

11.1.1 The Access Mode State Model defines the host view of equipment access mode, as well as the host interactions with the equipment necessary to switch the access mode. Each Load Port has its own Access Mode State Model. There are two access mode states: MANUAL and AUTO. These are defined in ¶11.3.3.

11.1.2 The access mode for a load port may be switched at anytime by the host or the operator, except when the Load Port Reservation State Model for that Load Port is in the RESERVED state or during carrier transfer. Carrier transfer boundaries, for determining when access mode may be changed, are designated by Table 8, Carrier Transfer Boundaries.