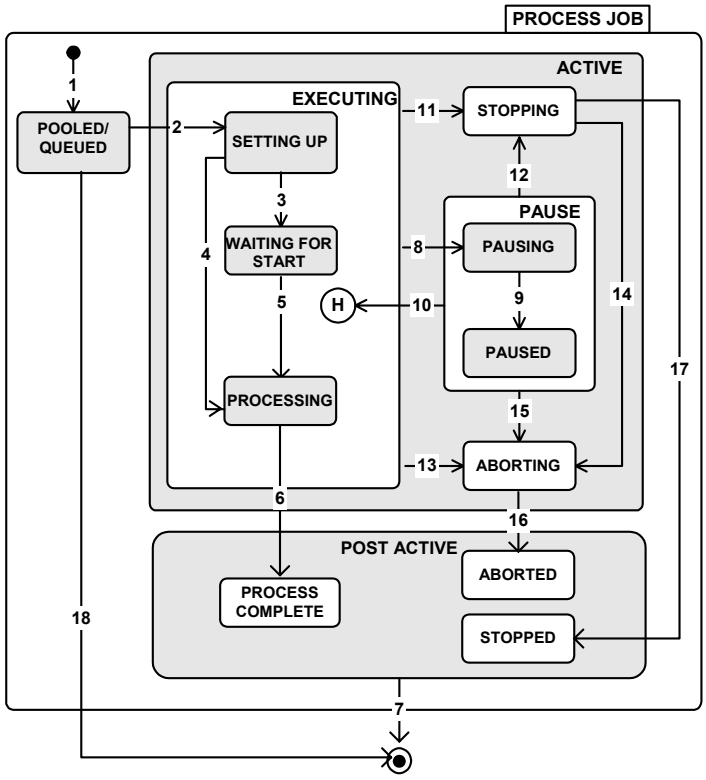

Figure 4
Process Job State Model

8.3.2.6 *PAUSED (PAUSE Substate)* — While the PR Job is in the PAUSED substate all processing resource activity has ceased. The PR Job is awaiting a RESUME (or STOP or ABORT) command.

8.3.2.7 *PAUSING (PAUSE Substate)* — While the PR Job is in the PAUSING substate, the processing resource continues to the first safe, continuable pausing place and then ceases activity. The activity may only cease at points that allow for resumption of the activity such that material integrity is maintained and the processing goals are accomplished.

8.3.2.8 *PROCESSING (EXECUTING Substate)* — While the PR Job is in the PROCESSING substate, the processing resource is doing the actual material processing using the equipment recipe(s) specified by the PR Job.

8.3.2.9 *PROCESS COMPLETE (POST ACTIVE Substate)* — While the PR Job is in the PROCESS COMPLETE substate the processing resource has completed processing all material specified by the PR Job. When all material removed from the processing resource, processing resource performs any required post-conditioning. Post-conditioning includes all operations in the processing resource after material departure, which are required by the recipe.

8.3.2.9.1 In cases where the process job is superseded by another process job on the same material and post-conditioning is not required, the first job terminates successfully while the material is still present. If post-conditioning is required, the second job may not supersede and remains on the queue.

8.3.2.10 *QUEUED/POOLED* — While the PR Job is in the QUEUED/POOLED substate, the process job has been accepted by the processing resource through a PR Job Create/Acknowledge transaction (such as PRJobCreate, PRJobCreateEnh, and PRJobMultiCreate) and is awaiting execution. One or more jobs may be in this state depending upon specific equipment capabilities. That is, if equipment does not support job queuing, then only one PR Job may be in this state at a time. If equipment does support job queuing, then the number of jobs that may be in this state must at least be equal to the number of load ports on the equipment. Advanced equipment job management capabilities require multiple jobs (greater than two) be in this state per load port.

8.3.2.10.1 The order that jobs become active is dependent upon whether the equipment supports job queuing and/or job pooling. Example methods for job activation include FIFO (first-in/first-out) order, material arrival order, and host ordering of jobs (provided by additional services). The equipment may support only one method of selecting jobs for activation, or more than one method allowing only one method to be used at a time or more than one method to be used at a time.

8.3.2.10.2 All process jobs pass through this state. If the processing resource supports queuing/pooling, jobs may remain in this state for prolonged periods. In any case, a process job remains queued/pooled until the material positions (of the processing resource) needed for the process are available or are already occupied by the material to be processed.

8.3.2.11 *STOPPING (ACTIVE Substate)* — While the PR Job is in the STOPPING substate, the processing resource is performing a stop procedure to terminate processing in an orderly manner. It is the responsibility of the processing resource to cease the current activity at the next safe, convenient point, preserving material integrity. For processing equipment this may require sending all related substrates to its output destination. This implies that each material is processed as specified in the recipe or not at all.

8.3.2.12 *WAITING FOR START (EXECUTING Substate)* — The substate WAITING FOR START is used only in manual start process jobs. It is entered once SETUP is complete and a PR Job Start Process has not yet been received by the processing resource. Manual start is defined by the supervisor in PR Job Create.

8.3.2.12.1 The job remains in this state, ready to process the material, until the PR Job Start Process is received or Abort or Stop terminates the job.

8.3.2.13 *POST ACTIVE* — This is the parent state of those states that refer to the final state (completion) of the process jobs. Process jobs that have completed, stopped or aborted should remain in this state until the trigger to transition to extinction is detected (see Table 1). If there is no Control Job associated with the process job, as in the case of the inline stepper/scanner, the process job should remain until all material associated with the process job has left the equipment or a process job specifying the same material replaces it.

8.3.2.14 *STOPPED (POST ACTIVE substate)* — This is the final state for those jobs that have been in the STOPPING state.

8.3.2.15 *ABORTED (POST ACTIVE substate)* — This is the final state for those jobs that have been in the ABORTING state.

8.3.3 *Process Job State Transitions* — The detailed state definitions are defined in Table 1.

Table 1 Process Job State Transition Table

#	*Current State*	*Trigger*	*New State*	*Action(s)*
1	(no state)	The processing resource accepts a Process Job create request.	QUEUED/ POOLED	The job is placed the job queue/pool. Acknowledge the Process Job creation.
2	QUEUED/ POOLED	The processing resource has been allocated to the Process Job.	SETTING UP	The job is removed from the queue/pool. PR Job Setup event is triggered. All required resource preconditioning is performed. When job material arrives all material preparation is performed.
3	SETTING UP	Job material is present AND the processing resource is ready to start the process job AND PRProcessStart attribute is not set.	WAITING FOR START	PR Job Waiting for Start event is triggered.
4	SETTING UP	Material is present and ready for processing. PRProcessStart attribute is set.	PROCESSING	PR Job Processing event is triggered. Material is processed.
5	WAITING FOR START	Job Start directive	PROCESSING	PR Job Processing event is triggered. Material is processed.

#	*Current State*	*Trigger*	*New State*	*Action(s)*
6	PROCESSING	Material processing completed.	PROCESS COMPLETE	PR Job Processing Complete event is triggered. The processing resource performs all required resource post-conditioning. Await material departure.
7	Any POST ACTIVE sub-state	Job material departs from the equipment OR the process job becomes extinct because the process job is replaced by another process job that specifies the same material when no control job is used.	(Extinction)	PR Job Complete event is triggered. The process job is deleted.
8	EXECUTING	The processing resource initiated a process pause action. (it received a PAUSE command or initiated an internal pause)	PAUSING	The processing resource pauses at the first convenient time.
9	PAUSING	The processing resource paused the job.	PAUSED	None.
10	PAUSE	The processing resource resumed the job.	EXECUTING	The processing resource resumes the activity that was paused.
11	EXECUTING	The processing resource initiated a process stop action. (it received a STOP command or initiated an internal stop)	STOPPING	The processing resource stops the current execution activity at the first convenient time.
12	PAUSE	The processing resource initiated a process stop action. (it received a STOP command or initiated an internal stop)	STOPPING	The processing resource stops the current execution activity at the first convenient time.
13	EXECUTING	The processing resource initiated a process abort action. (it received an ABORT command or initiated an internal abort)	ABORTING	The processing resource terminates the current execution activity immediately.
14	STOPPING	The processing resource initiated a process abort action. (it received an ABORT command or initiated an internal abort)	ABORTING	The processing resource terminates the current execution activity immediately.
15	PAUSE	The processing resource initiated a process abort action. (it received an ABORT command or initiated an internal abort)	ABORTING	The processing resource terminates the current execution activity immediately.
16	ABORTING	The abort procedure is completed. For some equipment, substrates are moved out into a safe location as part of the error recovery.	ABORTED	
17	STOPPING	The stop procedure is completed and all material is in a safe condition.	STOPPED	
18	QUEUED/ POOLED	“CANCEL,” “ABORT,” or “STOP” command received.	(no state)	Remove the process job from the queue/pool. PR Job Complete event is triggered. Delete the process job.

9 Object Definitions

9.1 Processing management defines one standard object, the Process Job.

9.2 *Process Job Object Definition* — The process job is a dynamic object created by the processing resource as requested by the supervisor. It tracks progress of the operations required and is deleted by the processing resource automatically upon completion. The process job is uniquely identified by the PRJobID attribute. The object attribute notation used in the table below is described in Conventions, ¶5.2.

9.2.1 The attributes in Table 2 shall be accessible using Object Services Standard (SEMI E39).

Table 2 Process Job Attributes

Attribute Name	*Definition*	*Rqmt*	*Access*	*Form*
ObjID	An identifier for the service user. It is set when the process job is created.	Y	RO	Text
ObjType	The object type.	Y	RO	Text: "PROCESSJOB"
PauseEvent	List of event identifiers that cause the equipment to automatically transition to the PAUSING/PAUSED states when one of the listed events is triggered.	N	RO	List of: EventID
PRJobState	A unique sub-state of the job according to the process job state model in Figure 4.	Y	RO	Enumerated: QUEUED/POOLED SETTING UP WAITING FOR START PROCESSING PROCESS COMPLETE PAUSING PAUSED STOPPING ABORTING STOPPED ABORTED
PRMtlNameList	List of identifiers of the material being processed.	Y	RO	List of: PRMtlName
PRMtlType	Identifies the type of material being processed.	Y	RO	Enumerated
PRProcessStart	Indicates that the processing resource start processing immediately when ready.	N	RO	Boolean: TRUE — Automatic start FALSE — Manual start
PRRecipeMethod	Indication of recipe specification type, whether using is applied and which method is used.	Y	RO	Enumerated: Recipe only Recipe with Variable Tuning
RecID	Identifier of the recipe applied.	Y	RO	Text
RecVariableList	List of variables supporting a recipe method.	N	RO	List of: RecipeVariable

9.2.2 A number of the ProcessJob attributes are composite data types. The constituent data is defined in Table 3.

Table 3 Attribute Data Definitions

Data Itentifier	*Description*	*Form*
PRMtlName	Textual identifier of the material being processed.	Text
RecipeVariable	Variables supporting a recipe method.	Structure composed of: RecipeVarName RecipeVarValue
RecipeVarName	The name of the recipe variable.	Text
RecipeVarValue	Value of the recipe variable.	Depends on variable

10 Messaging Services Detail

10.1 This section defines the messaging services required to implement the processing management concepts. The messages were introduced in ¶8.1. These services are independent of the messaging protocol used. They may be mapped to SECS-II (SEMI E5) or to other comparable protocols.

10.1.1 These messaging services define the messages to be used, the nature of the parameters contained within the messages, and data type of the parameters. Not defined here is the internal structure of the actual messages as transferred, including order of the parameters and how various data structures and data types are represented.

10.1.2 The service message notation used in the tables below is described in Conventions, ¶5.3.

10.2 *Service List* — The following messages are exchanged between host and equipment for the purpose of accomplishing processing management tasks.

Table 4 Service List

Message Name	*Type*	*Description*
PRGetAllJobs	R	Get a list of the jobs and their states for all jobs which have not completed.
PRGetSpace	R	Get the number of jobs which can currently be created on the resource.
PRJobAlert	N	Notification by the processing resource that the process job is setting up, processing, completed process, or that the job is completed.
PRJobCommand	R	Command which affects the process job.
PRJobCreate	R	Supervisor (service-user) request that a process job be performed.
PRJobCreateEnh	R	User request for job to be done. User assigns a unique job identifier.
PRJobDequeue	R	Removes (deletes) process job(s) from the queue.
PRJobEvent	N	Notification by the processing resource that a process-related event has occurred.
PRJobMultiCreate	R	Create several jobs which may be dissimilar. User assigns unique job identifiers.
PRJobSetRecipeVariable	R	User request for setting a new value to one of more recipe variable parameters.
PRJobSetStartMethod	R	Create a set of similar process jobs. User assigns unique job identifiers.
PRSetMtrlOrder	R	Request the service to use a specific methodology for processing order.

10.3 *Parameter Dictionary*

Table 5 Parameter Dictionary, Part 1

Parameter Name	*Definition*	*Form: Possible Values*
CmdParameter	Parameter supporting a command type.	Structure composed of: CmdParmName CmdParmValue
CmdParmName	The name of the parameter.	Text
CmdParmValue	Value of the parameter.	Varies per parameter

Parameter Name	*Definition*	*Form: Possible Values*
ErrorCode	Contains the code for the specific error found.	Enumerated: *PRJobCreate, PRJobCreateEnh, PRJobMultiCreate:* Parameters improperly specified Insufficient parameters specified Unsupported option requested Busy (no queueing or queue full) *PRJobCreateEnh, PRJobMultiCreate:* Object identifier in use *PRJobCommand:* Parameters improperly specified Insufficient parameters specified Unsupported option requested Command not valid for current state *PRJobComplete:* No material altered Material partially processed All material processed Recipe specification related error Failed during processing Failed while not processing Failed due to lack of material Job aborted Job stopped Job cancelled
ErrorText	Text in support of the error code.	Text
PRAck	Indicates whether the activity was successful.	Boolean: TRUE — Successful FALSE — Unsuccessful
PRCmdName	Indicates which process job command to perform.	Text: ABORT STOP CANCEL PAUSE RESUME STARTPROCESS
PREventData	Data related to the specific event.	Varies per parameter reported
PREventID	Identifier of the specific event which occurred.	Enumerated: Unique collection event ID for Waiting for Material and Process Job State Change events.

Table 6 Parameter Dictionary, Part 2

Parameter Name	*Definition*	*Form: Possible Values*
PRJobID	The unique identifier for a process job. It can be accessed as the ObjID attribute of the process job. The host may provide this identifier to the processing resource. In this case, the host must guarantee uniqueness of job identifiers for all the process jobs in the PROCESS JOB STATE in the equipment.	Text
PRJobList	List of process job identifiers and their states.	List of Structure PRJobID PRJobState
PRJobMilestone	Process job milestone.	Enumerated: PR Job Setup PR Job Processing PR Job Processing Complete PR Job Complete PR Job Waiting for Start
PRJobSpace	Used to indicate the number of jobs that can currently be created for the processing resource.	Integer
PRJobState	A unique state of the process job according to the process job state model.	Enumerated: QUEUED/POOLED SETTING UP WAITING FOR START PROCESSING PROCESS COMPLETE PAUSING PAUSED STOPPING ABORTING STOPPED ABORTED
PRMtlName	Textual identifier of the material being processed.	Text: Unique for each material with respect to the processing agent.
PRMtlType	Identifies the type of material being processed.	Enumerated
PRMtrlOrder	Defines the order by which material in the process jobs material list will be processed.	Enumeration: ARRIVAL – process whichever material first arrives. OPTIMIZE – process in an order that maximizes throughput. LIST – follow the order in the list.
PRPauseEvent	Variable containing information which is transferred to the corresponding PRJob attribute. Shall conform to event identifiers as defined in either SEMI E30 or E53.	(list of) text
PRProcessStart	Indicates that the processing resource start processing immediately when ready.	Boolean: TRUE – Automatic Start FALSE – Manual Start
PRRecipe	Specification of the process job recipe.	Structure composed of: PRRecipeMethod RecID (List of) RecipeVariable
PRRecipeMethod	Indication of recipe specification type, whether tuning is applied and which method is used.	Enumerated: Recipe only Recipe with VariableTuning

Parameter Name	*Definition*	*Form: Possible Values*
PRStatus	Reports the acceptance or rejection of a requested operation.	Structure composed of: PRAck (List of) Status
RecID	Identifier of the recipe applied.	Text: Unique with respect to the processing agent.
RecipeVariable	Variables supporting a recipe method.	Structure composed of: RecipeVarName RecipeVarValue
RecipeVarName	The name of the recipe variable.	Text: Depends on recipe
RecipeVarValue	Value of the recipe variable.	Depends on variable
Status	Reports any errors found.	Structure composed of: ErrorCode ErrorText
Timestamp	Event date and time.	Text: yyyymmddhhmmsscc

10.4 *Service Detail* — The tables below define the parameters for each service. In some cases, parameters have additional detail which is defined in the parameter definition section.

10.4.1 *PRJobCreate*

Table 7 PRJobCreate Service Detail

PRJobCreate Service Detail Section

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
PRJobID	-	M	The processing agent assigns the unique identifier which is used in all subsequent process job communications.
PRMtlType	M	-	
(List of) PRMtlName	M	-	All material shall be of the same material type. This is an ordered list and indicates the order in which the process job should process the material, if it is single wafer processing equipment.
PRRecipe	M	-	
PRProcessStart	M	-	Indicates auto or manual start.
PRStatus	-	M	

PRRecipe Parameter Detail Section

PRRecipeMethod	M	-	
RecID	M	-	The process job recipe identifier shall be unique within the domain of the processing agent.
(List of) Recipe Variable	C	-	Parameters required depend on the recipe method selected.

PRStatus Parameter Detail Section

PRAck	-	M	Indication of acceptance to perform the job.
(List of) Status	-	C	Error information, required if PRAck is unsuccessful.

10.4.2 *PRJobCreateEnh*

Table 8 PRJobCreateEnh Service Detail

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
PRJobID	M	M	User supplied Job ID. Must be unique among jobs known by the processing resource or else the resource shall reject the create request.
PRMtlType	M	-	May be NULL when no material is processed.
(List of) PRMtlName	M	-	An ordered list that associates a set materials with process conditions (process programs or recipes).
PRRecipe	M	-	This is a structure.
PRProcessStart	M	-	Indicates auto or manual start.
PRPauseEvent	M	-	If null, then processing will not be automatically paused.
PRStatus	-	M	Indicates success or failure.

10.4.3 *PRJobMultiCreate* — This service creates multiple process jobs. Each job can be created uniquely.

Table 9 PRJobMultiCreate Service Detail

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
(List of) PRJobMultiSpec	M	C	An ordered list of job specifications as follows: (list of) Structure: PRJobID PRMtlType (List of) PRMtlName PRRecipe PRProcessStart PRPauseEvent
(list of) PRJobID	-	M	Shall be returned for process jobs specified in the request service that the equipment created.
PRStatus	-	M	Indicates success or failure.

10.4.4 *PRJobDequeue* — Remove one or more jobs from the queue. PRStatus shall indicate any jobs which could not be removed because they either did not exist or were in the PR JOB ACTIVE state.

Table 10 PRJobDequeue Service Detail

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
PRJobList	M	M	(List of) PRJobID In the request, PRJobList specifies the jobs to be removed. In the reply, PRJobList indicates the jobs successfully removed.
PRStatus	-	M	Indicates success or failure.

10.4.5 *PRJobCommand* — All of the process job commands described in Section 8.2.3 are communicated using the PRJobCommand service. The commands are Abort, Stop, Cancel, Pause, Resume, and Start Process. This standard does not specify any required parameters. Abort is the only command which is required to be supported.

Table 11 PRJobCommand Service Detail

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
PRJobID	M	-	Identifies the process job on which to perform the command.
PRCmdName	M	-	
(List of) CmdParameter	C	-	Dependent on the command selected.

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
PRStatus	-	M	

10.4.6 *PRJobAlert* — Notification of process job milestones achieved by the processing resource are communicated using the PRJobAlert service. Process job milestones, which are described in ¶8.2.1, are events which are important to the control and tracking of the process job. The milestones required to be supported are PR Job Setup, PR Job Processing, PR Job Processing Complete, and PR Job Complete. An additional milestone, PR Job Waiting for Start, is used with the manual start option.

Table 12 PRJobAlert Service Detail

Parameter	*Req/Ind*	*Comment*
Timestamp	M	
PRJobID	M	Identifies the process job on which the milestone has been reached.
PRJobMilestone	M	
PRStatus	M	

10.4.7 *PRJobEvent* — Process job informational event notification, which is described in ¶8.2.2, is communicated using the PRJobEvent service. These are defined for Waiting for Material and Process Job State Change events. Support for informational events is not required.

Table 13 PRJobEvent Service Detail

Parameter	*Req/Ind*	*Comment*
PREventID	M	
Timestamp	M	
PRJobID	M	Identifies the process job which generated the event.
PREventData	C	

10.4.8 *PRJobSetRecipeVariable* — Sends a request to change the settings for a list of recipe variable parameters. Implementation of this service is optional.

Table 14 PRJobSetRecipeVariable Service Detail

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
PRJobID	M	-	
RecVariableList	M	-	(list of) RecipeVariable
PRStatus	-	M	Indicates success or failure. Failure is if a variable can't be set. A List of variables which could not be set is returned in the status.

10.4.9 *PRJobSetStartMethod* — Sends a request to change the start method for job(s). This request will fail if a specified job is not in the QUEUED/POOLED state. Implementation of this service is optional.

Table 15 PRJobSetStartMethod Service Detail

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
PRJobList	M	-	(List of) PRJobID
PRProcessStart	M	-	Indicates auto or manual start.
PRStatus	-	M	Indicates success or failure.

10.4.10 *PRGetAllJobs* — This message shall return a list containing job identifiers and the associated states of those jobs for all jobs which have not completed.

Table 16 PRGetAllJobs Service Detail

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
PRJobList	-	M	

10.4.11 *PRGetSpace* — This message shall return the remaining number of jobs that can be created for the processing resource.

Table 17 PRGetSpace Service Detail

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
PRJobSpace	-	M	

10.4.12 *PRSetMtrlOrder* — Request the Processing Management Service to use a specific strategy for the order in which materials are processed.

Table 18 PRSetMtrlOrder Service Detail

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
PRMtrlOrder	M	-	Sets the value for the strategy the service will use.
PRAck	-	M	Indicates success or failure.

10.5 *Mapping of Semantics to Syntax* — Table 20 provides the correspondence between the message semantics defined in Section 8.2 and the syntax as defined in Section 10.4. The use of '.req', '.rsp', or '.nfy' suffixes shows the direction of message flow. '.req' is a message request from the service user to the service provider. '.rsp' is a response message from the service provider to the service user. '.nfy' is a notification from the service provider to the service user.

Table 19 Correspondence of Message Semantics to Syntax

Parameter	*Comment*
PR Job Create	PRJobCreate.req
PR Job Create Acknowledge	PRJobCreate.rsp
PR Job Setup	PRJobAlert.nfy (PRJobMilestone = PR Job Setup)
PR Job Processing	PRJobAlert.nfy (PRJobMilestone = PR Job Processing)
PR Job Processing Complete	PRJobAlert.nfy (PRJobMilestone = PR Job ProcessingComplete)
PR Job Complete	PRJobAlert.nfy (PRJobMilestone = PR Job Complete)
PR Job Waiting for Material	PRJobEvent.nfy (PREventID = Waiting for Material)
PR Job State Changes	PRJobEvent.nfy (PREventID = Process Job State Change)
PR Job Abort	PRJobCommand.req (PRCmdName = ABORT)
PR Job Stop	PRJobCommand.req (PRCmdName = STOP)
PR Job Cancel	PRJobCommand.req (PRCmdName = CANCEL)
PR Job Pause	PRJobCommand.req (PRCmdName = PAUSE)
PR Job Resume	PRJobCommand.req (PRCmdName = RESUME)
PR Job Waiting for Start	PRJobAlert.nfy (PRJobMilestone = PR Job Waiting for Start)
PR Job Start Process	PRJobCommand.req (PRCmdName = STARTPROCESS)
PR Job Start Acknowledge	PRJobCommand.rsp (PRSatus.PRAck = TRUE)

11 Variable Data

11.1 The purpose of this section is to define the list of variable data requirements for Processing Management compliant equipment. Values of these variables are available to the host via collection event reports.

11.2 *Variable Data Definitions* — The identifier and all other attributes of the ProcessJob object shall be available for inclusion in event reports associated with it. The following attributes are most likely to be used: PRJobID, PRJobState, RecID, RecVariableList and PRMtlNameList.

12 Compliance

12.1 Processing management defines the standard services available to achieve job-based material processing in equipment. The capabilities supported allow flexible management of automated processing encompassing many process types. Only a subset of these capabilities may be needed for a particular implementation.

12.2 *Fundamental Requirements* — All processing agent implementations shall support the fundamental requirements. These have been indicated in the appropriate sections of the document and are listed together below:

- Create and execute a single process job to completion, given:
  - a single material of the appropriate type, uniquely identified.
  - a unique recipe identifier for which the corresponding recipe can be found.
- Detect and report the success or failure of the process job, indicating complete, partial, or non-processing of the material.
- Support Abort of the process job at all times, immediately ceasing activity and terminating the process job.
- Maintain the data of required process job attributes indicated in Table 2.
- Reject requests with incomplete or invalid parameters.
- Reject requests for capabilities not supported.
- Implement the services and messages with the exception of those required for the Optional capabilities.

12.2.1 Satisfying fundamental requirements may not provide sufficient flexibility or performance for some equipment. In such cases, fundamental functionality should be supplemented by optional capabilities as appropriate to the needs of the system.

12.3 *Additional Capabilities* — Optional capabilities defined or enabled in this standard include:

- Processing resource pre-conditioning and post-conditioning.
- Stop, Pause, and Resume of a process job.
- Manual process start.
- Process job queuing and Cancel on a queued job.
- Process tuning.
- Processing of material groups.
- Multiple concurrent process jobs.
- Multiple consecutive process jobs in a single visit.
- Process job with no material.
- Notification of waiting for material.
- Implement PRJobCreateEnh and PRJobMultiCreate.
- Report the process job milestones: Setup, Processing, Processing Complete, and Job Complete.

12.3.1 The services are defined with mechanisms to reject unsupported services and options should they be requested. This improves robustness and enables sophisticated service-users to adjust their requests to the capabilities of the particular processing agent.

12.4 Table 21 provides a checklist for Processing Management (PM) compliance.

Table 20 PM Compliance Statement

Fundamental PM Requirements	*PM Section*	*Implemented*	*PM Compliant*
Single Process Job Execution	8.2	☐ Yes ☐ No	☐ Yes ☐ No
Process Job Failure Indication	8.2.1.7	☐ Yes ☐ No	☐ Yes ☐ No
Abort Command	8.2.3.2	☐ Yes ☐ No	☐ Yes ☐ No
Process Job Object Implementation	8.3, 9	☐ Yes ☐ No	☐ Yes ☐ No
Reject Invalid/Incomplete Parameters	8.2.1.2.1	☐ Yes ☐ No	☐ Yes ☐ No
Reject Unsupported Capabilities	11.2	☐ Yes ☐ No	☐ Yes ☐ No
Services Implementation (not per Additional)	10	☐ Yes ☐ No	☐ Yes ☐ No
Additional PM Capabilities	*PM Section*	*Implemented*	*PM Compliant*
Process Job Milestones	8.2.1 (except 8.2.1.2,3)	☐ Yes ☐ No	☐ Yes ☐ No
Resource Pre/Post-conditioning		☐ Yes ☐ No	☐ Yes ☐ No
Stop, Pause and Resume Commands	8.2.3.3,4,5	☐ Yes ☐ No	☐ Yes ☐ No
Manual Process Start	8.2.3.9	☐ Yes ☐ No	☐ Yes ☐ No
Process Job Queuing	8.3	☐ Yes ☐ No	☐ Yes ☐ No
Process Tuning	7.5, 10.4.9	☐ Yes ☐ No	☐ Yes ☐ No
Processing of Material Groups	7.6	☐ Yes ☐ No	☐ Yes ☐ No
Multiple Concurrent Process Jobs	7.7	☐ Yes ☐ No	☐ Yes ☐ No
Multiple Consecutive Process Jobs	7.8	☐ Yes ☐ No	☐ Yes ☐ No
Process Job with No Material	7.9	☐ Yes ☐ No	☐ Yes ☐ No
Event Notification	8.2.2	☐ Yes ☐ No	☐ Yes ☐ No
Enhanced Job Creation	10.4.2	☐ Yes ☐ No	☐ Yes ☐ No
Multiple Job Creation	10.4.3	☐ Yes ☐ No	☐ Yes ☐ No

SEMI E40.1-0705
SECS-II SUPPORT FOR PROCESSING MANAGEMENT STANDARD

This standard was technically approved by the global Information & Control Committee. This edition was approved for publication by the global Audits and Reviews Subcommittee on April 7, 2005. It was available at www.semi.org in June 2005 and on CD-ROM in July 2005. Originally published in 1995; last published in 2004.

1 Purpose

1.1 This document maps the services and data of SEMI E40 to SECS-II streams and functions and data definitions.

2 Scope

2.1 This is the standard way to implement the Processing Management, which provides remote control of wafer processing, using the SECS-II message protocol.

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

3 Referenced Standards and Documents

3.1 *SEMI Standards*

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

SEMI E40 — Standard for Processing Management

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

4 Terminology

4.1 None.

5 Mapping of Processing Services

Table 1 Processing Management Messages Mapping

Service Message Name	*Stream, Function*	*SECS-II Name*
PRJobCreate request	S16F3,F4	Process Job Create Request/Acknowledge
PrJobCommand request	S16F5,F6	Process Job Command Request/Acknowledge
PRJobAlert notify	If E30 style events: S6F11,F12 If E40 style alerts: S16F7,F8 If E53 style events: S6F11,F12 S6F13,F14	If E30 style events: Event Report Send/Acknowledge If E40 style alerts: Process Job Alert Notify/Confirm If E53 style events: Event Report Send/Acknowledge Annotated Event Report Send/Ack
PRJobEvent notify	If E30 style events: S6F11,F12 If E40 style events: S16F9,F10 If E53 style events: S6F11,F12 S6F13,F14	If E30 style events: Event Report Send/Acknowledge If E40 style events: Process Job Event Notify/Confirm If E53 style events: Event Report Send/Acknowledge Annotated Event Report Send/Ack
PRSetMtrlOrder request	S16F29,30	Process Job Set Material Order
PRJobCreateEnh	S16F11/F12	PRJobCreateEnh
PRJobMultiCreate	S16F15/F16	PRJobMultiCreate

Service Message Name	*Stream, Function*	*SECS-II Name*
PRJobDequeue	S16F17/F18	PRJobDequeue
PRGetAllJobs	S16F19/F20	PRGetAllJobs
PRGetSpace	S16F21/F22	PRGetSpace
PRJobSetRecipeVariable	S16F23/F24	PRJobSetRecipeVariable
PRJobSetStartMethod	S16F25/F26	PRJobSetStartMethod

6 Mapping of Processing Parameter

Table 2 Data Item Mapping

Service Parameter	*SECS-II Data Item*
PrJobID	PRJOBID
PRMtlType	MF
PRMtlName	MID
PRAck	ACKA
PRRecipeMethod	PRRECIPEMETHOD
RecID	RCPSPEC
RecipeVarName	RCPPARNM
RecipeVarValue	RCPPARVAL
PRProcessStart	PRPROCESSSTART
PRCmdName	PRCMDNAME
PRJobMilestone	If E30 style events: CEID If E40 style alerts: PRJOBMILESTONE If E53 style events: CEID
PRJobState	PRSTATE
Timestamp	TIMESTAMP
PREventID	If E30 style events: CEID If E40 style events: PREVENTID If E53 style events: CEID
CmdParmName	CPNAME
CmdParmVal	CPVAL
PRMtrlOrder	PRMTRLORDER
ErrorCode	ERRCODE
ErrorText	ERRTEXT
PREventData	V (SV, ECV, DVVAL)
PRJobSpace	PRJOBSPACE
PRPauseEvent	PRPAUSEEVENT

7 Variable Data Item Mapping

7.1 This section shows the specific SECS-II data classes, and formats needed for SECS-II implementations of SEMI E40 variable data. According to SEMI E40 §11, all ProcessJob object attributes are to be available as variables for Process Job state transition events. These variables will be of SEMI E5 data item DVVAL.

8 Implementation Details

8.1 *Use of Object Services* — Several capabilities of the Processing Management Services are accessed through the Object Services Standard. Whan a Process Job has been created, (PRJOBID is valid), then its attributes can be read and written using the Object Services GetAttr and SetAttr messages.

8.1.1 E39 Object Services shall be used for access to ProcessJob attributes. The GetAttr service may be used for all ProcessJob attributes and the SetAttr service may be used on only those attributes whose Access is set to RW.

8.2 *Multi-Block Messages* — Processing Management Services is protocol independent and therefore, makes no mention of SECS-II multi-block access and grant messages. When these Service use the SECS-II protocol, then S16F3,F5 shall be preceded by an S16F1/S16F2 access request/grant message exchange when the message will be multi-block.

9 SECS-II Attribute Definitions

9.1 *Process Job Object SECS-II Attributes Definitions* — The following are the SECS-II structure definitions for the E40 ProcessJob object.

Table 3 Process Job SECS-II Attribute Definitions

Attribute Name	*Attribute Data Form: SECS-II Structure*
"ObjID"	<PRJOBID> (Conforms to the restrictions of ObjID as specified in SEMI E39.1, §6.)
"ObjType"	"ProcessJob"
"PauseEvent"	L,n n=number of collection events 1. <$CEID_1$> … n. <$CEID_n$> CEID restricted to format U()
"PRJobState"	<PRSTATE> PRJobState PRSTATE enumerated as follows: 51 (U1) Enumerations: 0 – QUEUED/POOLED 1 – SETTING UP 2 – WAITING FOR START 3 – PROCESSING 4 – PROCESS COMPLETE 5 – (Reserved) 6 – PAUSING 7 – PAUSED 8 – STOPPING 9 – ABORTING 10 – STOPPED 11 – ABORTED

Attribute Name	*Attribute Data Form: SECS-II Structure*
"PRMtlNameList"	When MF = 13 (0x0d) (carriers) L,n n=number of carriers 1. L,2 1. <CARRIERID$_1$> 2. L,j j=number of slots 1. <SLOTID$_1$> : j. <SLOTID$_j$> : n. L,2 1. <CARRIERID$_n$> 2. L,k k=number of slots 1. <SLOTID$_1$> : k. <SLOTID$_k$> When MF = 14 (0x0e) (substrate) L,n n=number of material (substrate) 1. <MID$_1$> substrate ID … n. <MID$_n$> MID restricted to format A
"PRMtlType"	<MF> MF restricted to format B **MF Value** **Description** 13 (0x0d) carriers (e.g. FOUP, SMIF pod, cassette) 14 (0x0e) substrate (e.g. wafer, mask, flat panel) other not valid for E40 material
"PRProcessStart"	<PRPROCESSSTART>
"PRRecipeMethod"	<PRRECIPEMETHOD>
"RecID"	<RCPSPEC>
"RecVariableList"	L,n n=number of recipe variables 1. L,2 1. <RCPPARNM$_1$> 2. <RCPPARVAL$_1$> … n. L,2 1. <RCPPARNM$_n$> 2. <RCPPARVAL$_n$>

SEMI E41-95
EXCEPTION MANAGEMENT (EM) STANDARD

1 Purpose

1.1 Interactive exception handling enhances the error recovery ability while maintaining automated control in the factory. This standard addresses the communications needs within the semiconductor manufacturing environment with respect to equipment exception handling.

1.2 This standard specifies capabilities to be provided by the exception agent for effective reporting and interaction with respect to abnormal situations in the equipment. It describes the *concept* of exception management, the *behavior* of the equipment in relation to interactive exception handling, and the *messaging services* which are needed to provide the functionality.

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

1.4 Implementation of automated exception management will help reduce error recovery time and avoid changing from automatic to manual equipment control in many situations. The adoption of the standards described will greatly reduce the effort required to integrate compliant equipment components. Compliance requires a specific set of standard services.

2 Scope

2.1 The current scope of this standard is interactive exception handling within a cluster tool.

2.2 While the functionality provided may be applied to other multi-resource equipment, it may not provide the flexibility required for automated management and command by the factory of all types of equipment. It is anticipated that this standard will be extended to accommodate management of exceptions in other types of multi-resource equipment and by the factory of all types of equipment.

2.3 This standard supports exception condition reporting, including alarms, by an exception agent to a decision authority. The exception agent also has the ability to enable and disable reporting on each exception condition.

2.4 Interactive exception handling is supported through selection by the decision authority of recovery actions in certain situations. The recovery actions are performed by the exception agent with the goal of resolving the abnormal situation and allowing normal equipment operation to continue.

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

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

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

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

3 Referenced Standards

3.1 *SEMI Standards*

3.2 The following SEMI[1] standard is related to the Exception Management standard:

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

4 Definitions

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

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

---

1 These documents can be obtained from Semiconductor Equipment and Materials International, 3081 Zanker Road, San Jose, CA 95134, 408.943.6900.

alarm — an alarm is related to any abnormal situation of the equipment that may endanger people, equipment, or material being processed.

clearing — exception agent to decision authority reporting that an abnormal situation related to an exception condition is no longer apparent or relevant.

decision authority — an entity requiring to be notified of significant exception condition changes and which decides how to proceed to resolve abnormal situations related to recoverable error conditions. The decision authority may be represented by a supervisory controller interacting with an operator who may ultimately choose the recovery action.

error condition — an exception condition which is not an alarm and which may support recovery actions requested by a decision authority.

exception agent — the entity which manages access to and reporting of information on abnormal situations in equipment. It achieves this by defining exception conditions, each related to a significant abnormal situation. It may provide services for a decision authority to direct the recovery from certain situations.

exception condition — a condition managed by an exception agent for reporting on and recovery from an abnormal situation in the equipment.

form — type of data representing information contained in an object attribute or service message parameter. The data types are detailed in Section 4.1.

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

posting — all exception agent to decision authority reporting associated with an exception condition while the related abnormal situation is apparent and relevant.

recovery action — an operation associated with an error condition with the aim of resolving the abnormal situation detected. It may supply information to the exception agent or request the exception agent to perform some activity.

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

service-provider — the software control entity that is the provider of any of the related services.

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

4.1 *Data Type*

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

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

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

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

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

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

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

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

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

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

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

5 Conventions

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

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

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

Attribute Name	*Definition*	*Access*	*Rqmt*	*Form*
The formal text name of the attribute	Description of the information contained	RO or RW	Y or N	(see below)

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

A 'Y' or 'N' in the requirement (Rqmt) column indicates whether or not this attribute must be supported in order to meet fundamental compliance for the service.

The Form column is used to indicates the format of the attribute. (See Section 4.1 for definitions.)

5.3 *Service Message Representation*

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

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

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

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

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

5.3.2 *Service Parameter Dictionary* — A service parameter dictionary table defines the parameters used in a service, as shown in the following table:

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

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

The Form column is used to indicate the type of data contained in a parameter. (See Section 4.1 for definitions.)

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

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

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

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

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

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

"M" — *Mandatory parameter* — must be given a valid value.

"C" — *Conditional parameter* — may be defined in some circumstances and undefined in others. Whether a value is given may be completely optional or may depend on the value of another parameter.

"U" — *User-defined parameter.*

"-" — The parameter is not used.

"=" — (for Response only) Indicates that the value of this parameter in the response must match that in the primary (if defined).

6 Overview

Exception management is concerned with the interactive handling of equipment exception conditions. This standard defines the services provided by which

abnormal situations are reported by an exception agent (service-provider) to a decision authority (service-user). In the case of recoverable situations, services are provided for a decision authority to choose how to proceed to resolve the abnormal situation.

The exception agent is the equipment entity which manages access to and reporting of information on abnormal situations. It achieves this by defining exception conditions, each related to a significant abnormal situation. All services are defined in terms of these exception conditions.

Exception management allows for the decision authority to direct the resolution of an abnormal situation. The decision authority selects a recovery action to be performed to resolve the situation from among the options supplied with the exception condition. The exception agent performs the requested recovery action, which may or may not resolve the situation.

The services are fully defined in terms of the functionality provided by the exception agent (service-provider) and as such do not dictate the architecture of the decision authority (service-user).

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

6.1 *Compliance* — Compliance with this standard includes adherence to all stated requirements in this document where implemented. This includes defined message services and state models.

There are two levels of compliance defined. The first is alarm reporting support. The second extends support to include interactive handling of recoverable exception conditions. Required capabilities are listed in Fundamental Requirements, Section 10.4.

7 Concepts

7.1 *Exception Management Model* — The exception management model describes the mechanism for interactive handling of equipment exception conditions.

An exception condition is a condition monitored in equipment by an exception agent (service-provider) for detecting an abnormal situation. Exception conditions are limited in this standard to those accessible by a remote decision authority (service-user).

An exception condition is persistent, existing whether or not the abnormal situation currently exists. The abnormal situation is indicated by the exception condition state becoming SET. An exception agent is an entity which manages remote decision authority access to any number of exception conditions. Each exception condition is identified by a name which is unique for the exception agent.

A decision authority is a remote entity requiring to be notified of exception condition information and which decides appropriate actions to take to resolve abnormal situations. The decision authority may be represented by a supervisory controller interacting with an operator who may ultimately choose the recovery action.

The exception agent notifies the decision authority on detection of an abnormal situation related to an exception condition (i.e., state is SET) and again when it is no longer apparent. It also provides access to exception condition attribute data and for the execution of recovery actions requested by the decision authority.

Exception management defines two types of exception conditions: alarms and error conditions.

An alarm is related to any abnormal situation on the equipment that may endanger people, equipment, or material being processed. Alarms do not provide for decision authority involvement in the resolution of the abnormal situation.

An error condition is related to any abnormal situation detected which is made accessible to a decision authority. An error condition may supply a list of possible recovery actions from which the decision authority can select to attempt to resolve the abnormal situation, thereby resulting in the error condition state becoming CLEARED.

Being persistent, exception condition attributes may be queried at any time, and reporting of changes, such as state SET/CLEARED, can be disabled by the decision authority.

7.2 *Posting and Clearing* — Significant changes in an exception condition are reported to the decision authority if enabled. The major significant events are the transitioning of the exception condition state to SET or CLEARED, which indicate that the abnormal situation has been detected or is no longer apparent, respectively.

All significant information is sent when reporting that the exception condition state is SET. This includes the unique identifier, type, a message describing the abnormal situation, time and a list of possible recovery actions where appropriate. It is important to keep the decision authority updated on exception condition information while the abnormal situation exists, especially with respect to valid recovery actions. All changes in the list of possible recovery actions are reported to the decision authority while the exception condition state is SET.

The term posting is used to describe all reporting the transition to the SET state and while an exception condition state is SET.

Clearing is the reporting of the occurrence of an exception condition state transition to CLEARED. Note that the exception agent, not the decision authority, transitions the state to CLEARED. Any acknowledgment at the decision authority of exception condition posting (e.g., by the operator) is not relevant to the exception agent.

Exception condition changes while its state is CLEARED are not reported to the decision authority.

7.3 *Enable/Disable Reporting* — The decision authority may enable and disable posting and clearing for a particular exception condition by setting and resetting its enabled attribute, respectively. Reporting on an exception condition is enabled by default.

Note that the exception condition itself is not being enabled or disabled, but the reporting of its state is being enabled or disabled.

Posting of an exception condition shall occur upon being enabled if the exception condition state is SET.

7.4 *Recovery Actions* — Recovery actions provide a mechanism for a decision authority to assist in resolving an abnormal situation detected in the equipment. This is generally needed to resolve failure or conflict where information is required beyond the capabilities of the system. By supplying options related to each error condition, error recovery is directed to the problem area.

One or more recovery actions may be associated with an error condition. A recovery action is any operation with the aim of resolving the abnormal situation detected. It may supply information to the exception agent or request it to perform some activity.

A recovery action may be requested by the decision authority only when the error condition state is SET. Only recovery actions currently valid for the error condition are accepted by the exception agent.

The list of valid recovery actions is supplied in the posting of an error condition. Changes in this list are notified by re-posting. A decision authority may request any of the supplied recovery actions to be performed and is notified of acceptance to perform the recovery and, some time later, its completion. No more than one recovery action may be in progress on a particular error condition and it may be aborted by the decision authority at any time.

A recovery action does not directly change the error condition state to CLEARED. It performs activities or provides information with the object of removing the abnormal situation which in turn changes the state to CLEARED. A recovery action may continue after the state has transitioned to CLEARED, until the activity initiated has completed. A recovery action is rejected if received when the error condition state is CLEARED.

Since the exception agent must maintain system integrity by ensuring that incompatible operations are not performed concurrently, it may reject any recovery action requested.

8 Behavior

This section provides a high-level definition of the communications between the decision authority and the exception agent used in exception management. It does not define the message detail, concentrating on the concepts. The message detail is addressed in the Messaging Services section.

8.1 *Exception Condition Communication*

8.1.1 *Exception Report Messaging* — The message flow for exception reporting is shown in Figure 1. The arrows represent significant information exchange.

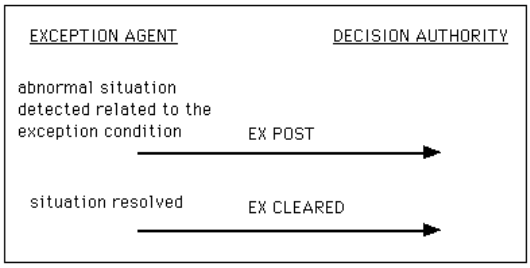

Figure 1
Exception Reporting Message Flow

A detailed description of each message used in exception reporting follows:

EX Post — The exception agent has detected an abnormal situation which is monitored by an exception condition. It changes the exception condition state attribute to SET. The exception agent notifies the decision authority using the EX Post notification, supplying the following information:

- identification of the exception condition,
- type of exception,
- time,
- a message explaining the abnormal situation,
- a list of possible recovery actions (where available with error conditions).

EX Post is re-sent every time the message or recovery action information changes as long as the exception condition state remains SET.

The EX Post notification is not sent if reporting on the exception condition is disabled, that is, if the enabled attribute is false. The EX Post notification is sent when an exception condition becomes enabled if the exception condition state is SET.

Upon receipt of the EX Post, the decision authority has the information needed to take the appropriate action. This may include requesting the exception agent to perform one of the supplied recovery actions.

EX Cleared — The abnormal situation detected by the exception agent is no longer apparent or relevant. The exception agent changes the exception condition state attribute to CLEARED. It notifies the decision authority using the EX Cleared notification, supplying the following information:

- identification of the exception condition,
- type of exception,
- time,
- a message.

The EX Cleared message is not sent if reporting on the exception condition is disabled. That is, the enabled attribute is false.

Upon receipt of the EX Cleared, the decision authority knows that the abnormal situation related to the exception condition is no longer apparent or relevant.

8.1.2 *Recovery Action Messaging* — In this section, the extended messaging of the recovery action is added to the reporting messaging described above. Recovery actions are not available for alarms. The message flow for recovery actions is shown in Figure 2.

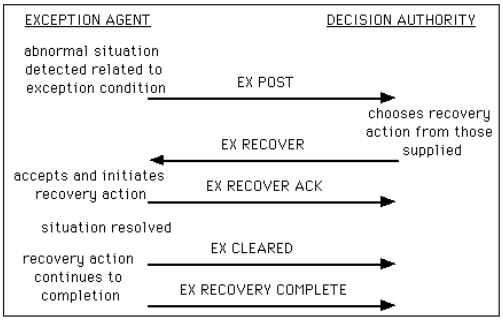

Figure 2
Recovery Action Message Flow

A detailed description of each message used in recovery actions follows:

EX Recover — The decision authority requests that the exception agent perform a recovery action. The particular recovery action selected is identified in the EX Recover request together with the related exception condition identifier. It shall be one of the recovery actions supplied for the exception condition in the EX Post notification.

Upon receipt of the EX Recover request and before acknowledging, the exception agent checks that the specified recovery action is currently valid. The request shall be accepted and initiated immediately or rejected by the exception agent.

The exception agent rejects a recovery action request if there is already a recovery action in progress on that exception condition.

EX Recover Acknowledge — The exception agent responds to the decision authority that the requested recovery action is accepted or rejected, and if rejected, supplies text reasons for failure.

Acceptance of a recovery action indicates that the exception agent has initiated the operation. The operation continues to completion without further intervention by the decision authority. The recovery action may or may not result in the exception condition state changing to CLEARED and the recovery action may continue after the state becomes CLEARED.

The exception agent may reject a requested recovery action for a number of reasons, including:

- unknown recovery for the exception condition,
- recovery currently invalid,
- busy with recovery for this exception condition,
- currently unable to perform the recovery (e.g., other conflicting activity or failure).

EX Recovery Complete — The exception agent declares the recovery action to be complete once it has completed the associated operation. This message is also used when a recovery action ends abnormally. The message indicates whether the operation completed normally and, if not, supplies text reasons for the failure.

Note that normal completion of a recovery action does not indicate that the abnormal situation has been resolved. That is indicated by the EX Cleared notification.

EX Recovery Abort — The decision authority may command the exception agent to abort a recovery action at any time. The goal of the abort command is to end

the recovery action activities as quickly as possible. The abort command terminates the recovery action.

8.2 *Exception Condition State Model* — The behavior required for exception management is fully specified by the exception agent (service-provider) behavior. This is described by the exception agent exception condition state model. All required decision authority (service-user) behavior is inferred by this model.

Message flow diagrams presented in the previous section are useful to show simple situations. The exception condition state model presented in this section provides the information necessary to extrapolate the message flow diagrams for all situations within the scope of this standard.

The exception condition provides for exception agent reporting on an abnormal situation in the equipment and management of a recovery action requested by the decision authority to resolve the situation. Recovery is requested in the context of a particular exception condition so the recovery action behavior forms a part of the exception condition behavior.

Figure 3 shows the state diagram for an exception condition. The Harel state model notation used is described in Conventions, Section 5.1. The corresponding state transition table is shown in Table 1 on the following page.

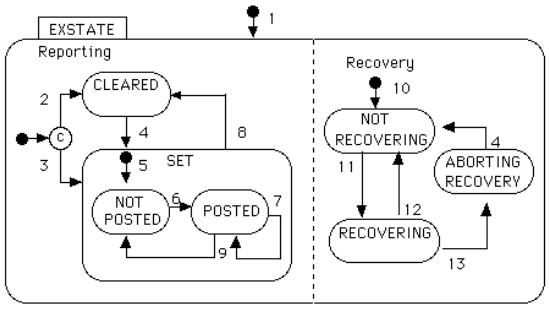

Figure 3
Exception Condition State Model

The detailed state definitions follow:

EXSTATE — When created by the exception agent, an exception condition enters the EXSTATE. It is in this state as long as it is in existence, irrespective of association to the decision authority. The exception condition has two concurrent subsets, which together fully describe its state. These subsets are Reporting and Recovery.

The exception condition initializes to the SET state if the abnormal situation related to the exception condition is apparent and relevant; otherwise, it initializes to the CLEARED state.

Reporting — Reporting is one concurrent subset of EXSTATE. It includes the substates describing whether the abnormal situation related to the exception condition is apparent and relevant, and whether the latest information on the situation has been reported.

Table 1 Exception Condition Transition Table

#	*Current State*	*Trigger*	*New State*	*Action(s)*
1	not EXSTATE	The exception agent creates the exception condition.	EXSTATE	
2	not EXSTATE	Initial creation and the abnormal situation is either not apparent or irrelevant.	CLEARED	
3	not EXSTATE	Initial creation and the abnormal situation is both apparent and relevant.	SET	
4	CLEARED	The abnormal situation is detected.	SET	
5	not SET	Default entry into SET state.	NOT POSTED	
6	NOT POSTED	Reporting enabled.	POSTED	Send “EX Post” message.
7	POSTED	Change in the list of possible recovery actions.	POSTED	Send “EX Post” message.
8	SET	The abnormal situation is either no longer apparent or has become irrelevant.	CLEARED	Send “EX Cleared” message.

9	POSTED	Reporting disabled.	NOT POSTED	
10	not EXSTATE	Default entry into recovery concurrent state at initial exception condition creation.	NOTRECOVERING	
11	NOTRECOVERING	"EXRecover" message received while in the SET state and accepted.	RECOVERING	Initiate requested recovery action and execute to completion.
12	RECOVERING	Recovery action completed.	NOTRECOVERING	Send "EX Recovery Complete" message.
13	RECOVERING	"EXRecoveryAbort" message received.	ABORTING-RECOVERY	Perform the abort procedure to terminate the recovery action in progress.
14	ABORTING-RECOVERY	Abort procedure is complete.	NOTRECOVERING	Send "EX Recovery Complete" message.

CLEARED — The abnormal situation related to the exception condition is either not apparent or not relevant.

SET — The abnormal situation related to the exception condition is apparent and relevant.

The exception agent should generate a collection event each time an exception condition transitions from CLEARED to SET and another from SET to CLEARED.

NOT POSTED — The latest information on the detected abnormal situation has not yet been reported by the exception agent. This may be transient on entering the SET state or may be because reporting is disabled. NOT POSTED is the default state when entering the SET state.

POSTED — The latest information on the detected abnormal situation has been reported by the exception agent to the decision authority.

Recovery — Recovery is one concurrent subset of EXSTATE. It includes the substates describing the behavior in relation to exception condition recovery actions.

NOTRECOVERING — In the NOTRECOVERING substate, there is no recovery action in progress directly related to the exception condition. NOTRECOVERING is the default state when an exception condition is initially created.

Recovery actions may only be initiated when the exception condition is in the SET state.

RECOVERING — A recovery action related to the exception condition is in progress when in the RECOVERING state.

ABORTINGRECOVERY — In the ABORTINGRECOVERY substate, the abort procedure is performed to immediately terminate the recovery action. It is the responsibility of the exception agent to cease physical activity as quickly as possible, having achieved a safe condition.

9 Object Definitions

Exception management defines one standard object, the Exception Condition.

9.1 *Exception Condition Object Definition* — The exception condition is a persistent object created by the exception agent. It provides the decision authority with reporting on, access to, and the possibility to direct resolution of an abnormal situation detected. It also tracks progress of a requested recovery action. The exception condition is uniquely identified by the EXID attribute.

The object attribute notation used in Table 2 is described in Conventions, Section 5.2.

Table 2 Exception Condition Attributes

Attribute Name	*Definition*	*Rqmt*	*Access*	*Form*
ObjType	The object type.	Y	RO	Text: "EXCEPTION"
ObjID	Exception agent unique identifier for the exception condition.	Y	RO	Text: Unique with respect to the exception agent.

EXType	Identifies the type of exception condition.	Y	RO	Enumerated: AlarmError
EXMessage	Text message describing the abnormal situation monitored.	Y	RO	Text
EXEnabled	Indicates that reporting to the decision authority on the exception condition is enabled.	Y	RW	Boolean: TRUE – Enabled FALSE - Disabled
EXRecActList	List of possible recovery actions.	N	RO	List of: Text
EXStateList	All concurrent sub-states of the exception condition according to the state model in Figure 3.	Y	RO	List of: Text
EXRecoveryAction	Recovery action which may be requested to resolve the abnormal situation.	N	RO	Text
EXState	A unique sub-state of the exception condition according to the state model in Figure 3.	Y	RO	Text: EXSTATE/CLEARED EXSTATE/SET/NOTPOSTED EXSTATE/SET/POSTED EXSTATE/NOTRECOVERING EXSTATE/RECOVERING EXSTATE/ABORTINGRECOVERY

10 Messaging Services Detail

This section defines the messaging services required to implement the exception management concepts. The messages were introduced in Section 8.1. These services are independent of the messaging protocol used. They may be mapped to SECS-II (SEMI-E5) or to other comparable protocols.

These messaging services define the messages to be used, the nature of the parameters contained within the messages, and data type of the parameters. Not defined here is the internal structure of the actual messages as transferred, including order of the parameters and how various data structures and data types are represented.

The service message notation used in the tables below is described in Conventions, Section 5.3.

10.1 *Service List* — The messages shown in Table 3 are exchanged between service-provider and service-user for the purpose of accomplishing exception management tasks.

Table 3 Service List

Message Name	*Type*	*Description*
EXPost	N	Notification by the exception agent that the abnormal situation related to the exception condition has been detected or significant information has changed while the situation exists.
EXCleared	N	Notification by the exception agent that the abnormal situation related to the exception condition is no longer apparent or relevant.
EXRecover	R	Decision authority request that a particular recovery action be performed.
EXRecoveryComplete	N	Notification by the exception agent that a recovery action has completed.
EXRecoveryAbort	R	Decision authority request that a recovery action be terminated immediately.

10.2 *Parameter Dictionary*

Table 4 Parameter Dictionary

Parameter Name	*Definition*	*Form: Possible Values*
ErrorCode	Contains the code for the specific error found.	Enumerated: EXRecover: Parameters improperly specified Insufficient parameters specified Recovery action currently invalid Busy with another recovery Currently unable to perform the recovery EXRecoveryAbort: Parameters improperly specified Insufficient parameters specified No active recovery action EXRecoveryComplete: FailedRecovery aborted
ErrorText	Text in support of the error code.	Text
EXAck	Indicates whether the request was successful or the activity completed normally.	Boolean: TRUE – Successful FALSE – Unsuccessful
EXID	Persistent exception condition identifier.	Text: Unique with respect to the exception agent.
EXMessage	Message describing the situation related to an exception condition.	Text
EXRecovery	Identifies a recovery action associated with an exception condition.	Text
EXRecoveryStatus	Reports the acceptance or rejection of a requested recovery action and whether completion was normal.	Structure composed of: EXAck (List of) Status
EXType	Identifies the type of exception condition.	Enumerated: Alarm Error
Status	Reports any errors found.	Structure composed of: ErrorCode ErrorText
Timestamp	Event date and time.	Text yyyymmddhhmmsscc

10.3 *Service Detail* — Tables 5 through 9 define the parameters for each service. Parameters have additional detail which is defined in the parameter dictionary, Table 4.

10.3.1 *EXPost* — Detection of abnormal situations by the exception agent is communicated using the EXPost service, as described in Section 8.1. This notification is also used to communicate any significant changes in exception condition information while its state is SET.

Table 5 EXPost Service Detail

Parameter	*Req/Ind*	*Comment*
Timestamp	M	
EXID	M	Identifies the exception condition which has detected the abnormal situation.
EXType	M	
EXMessage	M	
(List of) EXRecovery	C	List of possible recovery actions. Not available for alarms.

10.3.2 *EXCleared* — Notification that an abnormal situation is no longer apparent or relevant, as is described in Section 8.1, is communicated using the EXCleared service.

Table 6 EXCleared Service Detail

Parameter	*Req/Ind*	*Comment*
Timestamp	M	
EXID	M	Identifies the exception condition which has detected the abnormal situation.
EXType	M	
EXMessage	M	

10.3.3 *EXRecover* — Recovery action requests, described in Section 8.1, are communicated using the EXRecover service. The recovery action services are not available for alarms.

Table 7 EXRecover Service Detail

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
EXID	M	-	Identifies the exception condition on which to perform the recovery action.
EXRecovery	M	-	The particular recovery action being requested.
EXRecoveryStatus	-	M	

10.3.4 *EXRecoveryComplete* — Notification of recovery action completion by the exception agent is communicated using the EXRecoveryComplete service. Recovery action completion, which is described in Section 8.1, is not directly linked to clearing the exception condition state, and successful completion indicates only that the operation performed completed normally.

Table 8 EXRecoveryComplete Service Detail

Parameter	*Req/Ind*	*Comment*
Timestamp	M	
EXID	M	Identifies the exception condition on which the recovery action was performed.
EXRecoveryStatus	M	

10.3.5 *EXRecoveryAbort* — Recovery action abort, described in Section 8.1 is communicated using the EXRecoveryAbort service.

Table 9 EXRecoveryAbort Service Detail

Parameter	*Req/Ind*	*Rsp/Cnf*	*Comment*
EXID	M	-	Identifies the exception condition on which the recovery action is being performed.
EXRecoveryStatus	-	M	

10.4 *Fundamental Requirements* — Exception management defines the standard services available to achieve exception condition-based exception handling and error recovery in equipment.

All exception agent implementations shall support the fundamental requirements. This standard provides for two aspects of fundamental requirements: exception reporting and interactive exception handling. It is possible to support only exception reporting in systems which do not require interactive exception handling. Interactive exception handling requires that exception reporting be supported.

10.4.1 *Exception Reporting* — The fundamental requirements of the exception agent for exception reporting are based on maintaining the decision authority updated on the exception conditions it is interested in. These are listed below.

- Detect and report the occurrence of significant abnormal situations by setting the related exception condition, and posting it if reporting is enabled.

- Detect and report that an abnormal situation is no longer apparent or relevant by clearing the related exception condition and reporting it if enabled.
- Provide for enabling and disabling reporting on each exception condition.
- Maintain the data of exception condition attributes indicated in Table 2.
- Reject requests for capabilities not supported (such as recovery actions).

10.4.2 *Interactive Exception Handling* — The fundamental requirements of the exception agent for interactive exception handling extend the exception reporting to allow the decision authority to request recovery actions to be performed to resolve abnormal situations. These are listed below.

- Support all exception reporting fundamental requirements specified above.
- Supply a list of valid recovery actions for an exception condition when posting.
- Execute a requested recovery action for an exception condition if the requested recovery action is currently valid. The exception agent may support only a single recovery action to be in progress at a time and reject all other recovery action requests for other exception conditions while it is busy.
- Report the completion of the recovery action.
- Support Abort of the recovery action at all times, immediately ceasing recovery activity and terminating the recovery action.
- Reject requests with incomplete or invalid parameters.
- Reject requests for capabilities not supported.

Optional capabilities defined or enabled in this standard include the following:

- Support for multiple concurrent recovery actions. The standard allows only one recovery action in progress for each exception condition.
- Adjust the exception condition message and the valid recovery action list as appropriate during the error recovery, and post the changes if reporting is enabled.

The services are defined with mechanisms to reject unsupported services and options should they be requested. This improves robustness and enables sophisticated service-users to adjust its requests to the capabilities of the particular exception agent.

SEMI E41.1-0996
SECS-II SUPPORT FOR EXCEPTION MANAGEMENT STANDARD

1 Purpose

This document maps the services and data of its prime document, SEMI E41, to SECS-II streams and functions and data definitions.

2 Scope

This is the standard way to implement the Exception Management Standard, which provides remote control communication of exceptions and recovery, using the SECS-II message format.

3 Referenced Documents

3.1 *SEMI Standards*

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

SEMI E41 — Exception Management (EM) Standard

4 Terminology

None.

5 Mapping of Exception Management Messages

Table 1 Exception Management Messages SECS-II Mapping

Service Message Name	*Stream, Function*	*SECS-II Name*
EXPost Notify	S5F9,F10	Exception Post Notify/Confirm
EXCleared Notify	S5F11,F12	Exception Clear Notify/Confirm
EXRecover Request	S5F13,F14	Exception Recover Request/Acknowledge
EXRecoveryComplete Notify	S5F15,F16	Exception Recovery Complete Notify/Confirm
EXRecoveryAbort Request	S5F17,F18	Exception Recovery Abort Request/Acknowledge

6 Exception Parameters Mapping

Table 2 Exceptions Data Item Mapping

Parameter	*SECS-II Data Item*
ErrorCode	ERRCODE
ErrorText	ERRTEXT
EXAck	ACKA
EXEnabled	EXENABLED
EXID	EXID
EXMessage	EXMESSAGE
EXRecovery	EXRECVRA
EXState	EXSTATE
EXType	EXTYPE
Timestamp	TIMESTAMP

7 Implementation Details

7.1 Several capabilities of the Exception Management Services (EMS) are accessed through the Object Services Standard. The Exception Objects are persistent. An Object Services compliant implementation of EMS will allow access to all the attributes specified in the SEMI E5 Object definition table for an Exception Object.

All implementations shall use Object Service's GetAttr and SetAttr to access:

EXENABLED

EXSTATE

NOTE 1: EXSTATE is read-only.

SEMI E42-0704
RECIPE MANAGEMENT STANDARD: CONCEPTS, BEHAVIOR, AND MESSAGE SERVICES

This standard was technically approved by the Global Information & Control Committee and is the direct responsibility of the Japanese Information & Control Committee. Current edition approved by the Japanese Regional Standards Committee on April 30, 2004. Initially available at www.semi.org June 2004; to be published July 2004. Originally published September 1995; previously published February 2000.

CONTENTS

NOTES

This page intentionally left blank.

SEMI E42-0704
RECIPE MANAGEMENT STANDARD: CONCEPTS, BEHAVIOR, AND MESSAGE SERVICES

[E] This standard was editorially modified in September 1999 to conform to its non-provisional status. Changes were made to Section 1.2.

1 Introduction

This standard defines the concepts required for management of recipes, the operations or behavior provided by the Recipe Management Standard (RMS), and the messages through which services are provided through an interface between the provider and the user of these services.

1.1 *Purpose* — The purpose of this standard is twofold:

- *To enable applications software to be developed that can assume the existence of standard concepts, behaviors, and message services that collectively form Recipe Management and that take advantage of them.*
- *To enable software to be developed to offer the Recipe Management capabilities.*

1.2 *Scope* — This is a standard that defines concepts, behavior, and services to support the integration of automated recipe management within a semiconductor factory. These services are applicable to a variety of relationships, including both traditional **host/equipment** and cluster tool controller/attached module communications and control.

The standard provides a set of communications services which allows such systems to transfer and manage recipes to ensure the correct processing of material within semiconductor manufacturing *equipment* and systems. RMS also requires compliance to SEMI E39 (Object Services Standard (OSS): Concepts, Behavior, and Services) for completeness.

This document describes several different hierarchical relationships: supervisory agents and their supervised agents, recipes and their subrecipes, and recipe classes and their subclasses. Such hierarchical relationships provide a natural organizational and classification structure that is reflected in many different kinds of systems, such as telephone switching systems and directory trees. It is the intent of this standard to support logical hierarchical relationships rather than to require that strict hierarchical relationships be implemented in systems architecture.

RMS places no restriction on where the set of defined services is implemented.

1.3 *Referenced Documents*

1.3.1 *Semiconductor Equipment and Materials International (SEMI)*[1]

1.3.1.1 *SEMI Equipment Automation/Hardware Volume*

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

1.3.1.2 *SEMI Equipment Automation/Software 2 Volume*

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

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

SEMI E53 — Event Reporting

1.3.2 *Other References*

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

D. Harel, *Statecharts: A Visual Formalism for Complex Systems,* Science of Computer Programming 8, 1987.

1.4 *Definitions* — Basic definitions for objects, services, and form are provided in SEMI E39, Sections 4.1 through 4.4. This section provides additional definitions.

Definitions in Section 1.4.3 are specific to RMS.

1.4.1 *Services*

service provider — An application (a component of an *agent*) responsible for providing services to the service user.

service user — (service consumer) An application that uses the services provided.

---

1 Semiconductor Equipment and Materials International (SEMI), 805 East Middlefield Road, Mountain View, CA 94043, 650.964.5111, FAX 650.967.5375

1.4.2 *Form*

binary — A string of bit values (zeroes and ones), with a format that is either left unspecified or specified by bit position, with the most significant bit first. The total length of the string is a multiple of eight. Messaging protocol may impose restrictions on length.

1.4.3 *Recipe Management* — This section introduces basic terminology used in RMS. Entries are in alphabetical order. Additional definitions and specifications are provided in later sections.

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

authorized user — A user who can be identified to an *agent* as having the level of authority required for a particular activity, such as *certifying* a recipe for that equipment.

collection event — A detectable occurrence of interest to a service user.

component agent — A subordinate *agent* that provides services to a *supervisory* agent.

download — An operation that transfers a recipe (down) to an execution storage area.

edit — An operation which creates a new recipe body or changes the body of an existing recipe.

editor — A service which allows a *user* to edit a recipe. *Editors* are not specified in RMS.

equipment[2] — An *agent* with associated hardware that provides, at a minimum, recipe **execution** services.

event — A detectable occurrence significant to an object.

execution (recipe execution) — The process of reading the recipe contents and implementing its instructions, process parameters, or other information required for its own processing.

executing agent — An *agent* that provides *recipe execution* capabilities.

execution area — The storage location of the recipe(s) currently *selected* (ready) for *execution*.

host — A *supervisory agent* that represents the factory to its subordinates.

logical recipe — A recipe with a particular set of *attributes* and a particular *body,* considered independently from its physical location. A *logical recipe* may have multiple instances or copies.

name — A text-based *attribute* of an object that may be used as all or part of its *identifier*.

namespace — In general, a domain within which object identifiers are unique. In RMS, the term *namespace* is used as a synonym for *recipe namespace*, unless otherwise stated.

operator — The user who interacts locally with *agent* through the *agent's* interface.

recipe — The pre-planned and reusable portion of the set of instructions, *settings*, and parameters under control of an *agent* that determines the processing environment seen by the manufactured object and that may be subject to change between runs or processing cycles.

recipe class — A formal grouping of recipes with a common language syntax and functionality.

recipe executor — The component of an *executing agent* that executes recipes.

recipe namespace — A logical management domain with the responsibility for the storage and management of recipes, the ensurance of the uniqueness of recipe **identifiers** within that domain, and the provision of services pertaining to recipes stored within that domain.

recipe parameter — A control value that affects the *agent's* process.

select — The act of preparing a recipe for execution.

setting — A static value accessible to the *user*, through one or more methods, that is used by equipment to control its process. *Settings* include, but are not limited to, setpoint values. *Settings* typically may be specified within a recipe.

storage area — An area where objects and data are stored.

subordinate agent — An *agent* that is a component of, or managed by, another *agent*.

supervisory agent — An *agent* with supervisory responsibilities for one or more subordinate *agents*.

timestamp — The notation of the date and time of the occurrence of an event.

upload — An operation that transfers a recipe (up) from an execution storage area.

user — A person interacting with an *agent* directly through the *agent's* human interface or indirectly through the *agent's supervisor*.

2 The term "equipment" is restricted in RMS to "intelligent equipment."

validate — The action of checking recipe contents to ensure that parameter type and range are valid for the equipment configuration prior to execution. [Note that *validation* and *verification* are used in different ways.]

variable parameter — A formally defined variable (*setting*) defined in the body of a recipe permitting the actual value to be supplied externally.

verify — The operation of reading a recipe's contents to ensure that it is syntactically correct and identifying elements that must be made public.

version — Part of a recipe's *identifier* that is used to show its heritage.

1.5 *Conventions* — The following conventions are used in this document:

- *To highlight terms specific to RMS (excluding terms defined in Sections 1.4.1 through 1.4.3 that are common to multiple standards), a defined term appears in* ***boldface*** *wherever it first appears and wherever it is defined. This alerts the reader to those terms with specific meanings. Except for terms that are very common, such as recipe, host, and equipment, defined terms otherwise are in* italics *wherever they appear.*
- *Terms related to objects and object services are in conformance with SEMI E39 (Object Services Standard: Concepts, Behavior, and Services). A brief discussion of objects is provided in the Appendix of that document.*
- *Attribute names are underlined.*
- *Attributes called "names" in RMS are generally intended to be used for the ObjID attribute of a* standardized object.
- *To prevent the definition of numerous message parameters named "XxxList," this document adopts the convention of referring to the list as "(List of) Xxx". In this case, the definition of the parameter Xxx will be given, not of the list. The term "list" indicates a collection (or set) of zero or more items of the same data type.*

For attributes that are lists, this convention is not followed, as the entire attribute, as a list, must be assigned a specific name.

1.5.1 *Text String Restrictions* — Text strings used in attribute names, attribute values, or message parameters, are subject to the restrictions defined by OSS: Text in ASCII is restricted to the characters between 20_{16} and $7D_{16}$, excluding the question mark "?" ($3F_{16}$), the asterisk "*" ($2A_{16}$), and the tilde "~" ($7E_{16}$).

Text strings used as, or within, the object identifier ObjID are additionally restricted to exclude the "greater than" symbol ">" ($2E_{16}$) and the colon character ":" ($3A_{16}$) to conform with OSS requirements.

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

This document also adopts the extension of Harel notation to show the deletion of an object as used by Rumbaugh, et al. (see Section 1.5.2.1).

1.5.3 *Objects — Standardized objects* defined by RMS conform to the requirements of SEMI E39 (Object Services Standard: Concepts, Behavior, and Services). RMS adopts the convention of showing *standardized objects* as drawn with a heavy line in object models. *Non-standardized objects* are used to illustrate concepts and relationships but are not formally defined and cannot be accessed through Object Services. A list of *standardized objects* defined by RMS is included in Related Information 1.

1.5.3.1 *OMT Object Information Model* — The object models are presented using the Object Modeling Technique, developed by Rumbaugh et al., in *Object-Oriented Modeling and Design,* Prentice-Hall, Englewood Cliffs, NJ, 1991. Overviews of this notation are provided in an appendix of SEMI E39 (Object Services Standard: Concepts, Behavior, and Services).

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

Attribute Name	*Definition*	*Access*	*Rqmt*	*Form*
The formal text name of the attribute.	Description of the information contained.	RO or RW	Y or N	(see below)

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

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

The Form column is used to indicate the format of the attribute. (See Section 1.4 for definitions.)

1.5.4 *Service Message Representation*

Service Resource Definition

A **service definition table** defines the specific set of messages for a given *service resource*, as shown in the following table:

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

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

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

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

Service Parameter Dictionary

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

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

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

The Form column is used to indicate the type of data contained in a parameter. (See Section 1.4 for definitions.)

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

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

Service Message Definition

A **service message definition** table defines the parameters used in a service, as shown in the following table:

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

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". When receiver terms this message the "Indication" or the "Request", the receiver may then send a "Response", which the original sender terms the "Confirmation".

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 value of the other parameter.

"U" — **User-defined parameter**.

"-" — The parameter is not used.

"=" — (for Response only) Indicates that the value of this parameter in the response must match that in the primary (if defined).

1.6 *Requirements* — Requirements for recipe management are varied. For example, there is a fundamental need to retain information about each recipe, to provide that information upon request, and to transfer it with the recipe so that it remains available. There is a need to classify recipes, to reuse them, and to share them with multiple installations of *equipment*. The need for sharing recipes introduces a new requirement for safeguarding the relationship between a recipe's **identifier** and its contents, to be able to protect a recipe from unauthorized changes, and to ensure that the *identifier* for a recipe that is used by multiple *equipment* is in fact the same recipe.

The requirements for managing recipes in a factory are given in tabular form in Table 1.1, which presents them according to their functional areas and the specific issues involved. Table R1-1, in Related Information, shows the specific concepts defined in RMS to address each of these requirements.

Table 1.1 Requirements

Functional Areas	*Issue*	*Requirements*
Management of Recipes	Identification	Uniquely store, identify, and select recipes in a system.
		Easy and clear identification of a recipe.
	Multiple Access	Share recipes among *equipment*.
		Enable synchronized change of shared recipes among *equipment*.
	History and Traceability	Capture the history of recipe usage.
		Capture the history of recipe changes.
	Life Cycle Management	Manage approval of recipes for process development and production.
Operations	Protection	Protect recipes from unexpected changes.
		Protect recipes from mistakes in operations.
	Portability	Create, edit, and change recipes outside the *equipment* which execute them.
	Reusability	Change recipes for a specific piece of *equipment*.
		Use recipes developed on one piece of *equipment* for other *equipment*.
		Share recipes among the same kind of *equipment* by adjusting for individual differences.
Execution	Safety	Do not execute recipes that are not syntactically correct.
		Execute the specified (selected) recipes without errors.
		Protect recipes being executed from inadvertent change caused by other activities.
	Flexibility	Change a recipe's **parameters** during or between runs in a systematic way.
System Operation	System Node Operation	Dynamically connect and disconnect *equipment* to/from the communications network.
	Stand-Alone Operation	Execute recipes with no communications link.
		Create and change recipes with no communications link.
		Manage recipes moved between *equipment* or from off-line storage through removable media.
		Enable smooth integration of stand-alone *equipment* into on-line factory systems.

1.7 *Document Structure* — Figure 1.1 depicts the domain of the Recipe Management Standard as consisting of Purpose, Concepts, Behavior, and Message Services. This also reflects an underlying structure of the document.

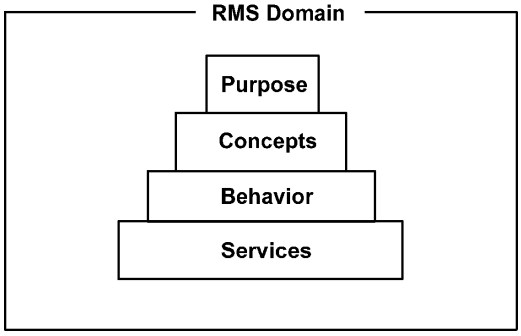

Figure 1.1
Recipe Management Domain

Purpose provides the motivation for Recipe Management capabilities and is addressed in Section 1.6. Concepts provide a detailed introduction to the *standardized objects* of RMS, their attributes, and their relationships with other objects. Behavior describes the operations that are performed by, or on, these objects. Finally, message services define the messages and their parameters independently of the protocol in which they are implemented. Both concepts and behavior represent an "inside view" of an RMS application, while services provide an interface from an external view.

- *Section 1: the formal introduction to RMS.*
- *Section 2: an overview of the major objects of RMS and their relationships, to provide a general context for the technical detail that follows.*
- *Sections 3-6: concepts for recipes, recipe namespace, distributed recipe namespace, and recipe executor.*
- *Section 7: the concept of the* agent *and of* service resources*, to provide a more complete context for RMS implementations.*
- *Sections 8-11: behavior (operations) for recipe management, namespace management, distributed recipe namespace management, and the* recipe executor.
- *Sections 12-14: definitions of message services for the* recipe namespace, distributed recipe namespace, *and* recipe executor ***service resources***.
- *Section 15: RMS compliance.*
- *Section 16: a glossary of terms, provided as a convenient reference.*
- *Related Information: provides background information on the requirements behind RMS and examples of applications.*

1.8 *Applicable Documents*

ISO/TR 8509:1987, Information Processing Systems, Open Systems Interconnection — Service Conventions.

2 Overview of RMS

This section provides an introduction to, and overview of, the major objects of RMS.

2.1 *Recipe Management Models* — To provide a clearer understanding of the major entities or areas of functionality specified by RMS, they are portrayed as objects in RMS models, using OMT notation.

Relationship lines in the object-based models do not represent a direct communication link between objects, but rather the knowledge and association that one object has with respect to another.

The models do not indicate the relative location of two objects.

2.2 *Major Objects in RMS* — Recipe Management is concerned with the major areas of functionality illustrated in Figure 2.1: recipes[3], **recipe namespaces** providing persistent recipe storage, **recipe namespace managers** that provide management of the *namespace* and access to its recipes, and the **recipe executor** that executes recipes.

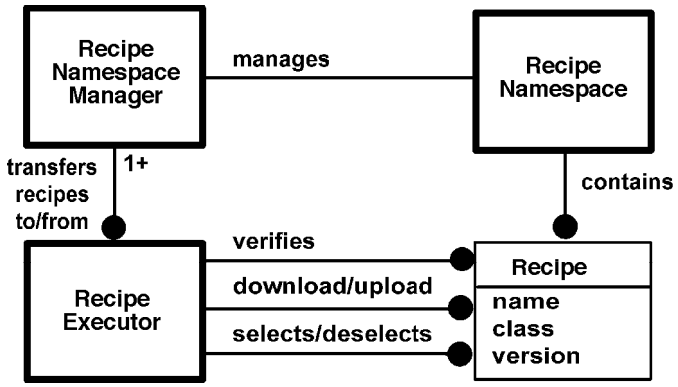

Figure 2.1
Major Objects of RMS

2.2.1 *Recipes* — Recipes provide a flexible, manipulatable, and re-usable form for users to select sequencing and settings to effect a particular result. Equipment uses recipes in a variety of ways to control the processing environment, sometimes using several types of recipes together. Recipes may also be used for maintenance activities, such as calibration or cleaning.

Recipes can be modified and copied from one environment to another. Their inherent flexibility as a form also becomes an endless source of problems unless they can be managed. Misprocessing, unintentionally running the wrong recipe for a given product, is prohibitively expensive for factories.

RMS defines a recipe as an object with attributes — information about the recipe — as well as content. It is the attributes of the recipe that allow true management of recipes, throughout their lifecycles, to occur.

2.2.2 *Recipe Namespace* — A *recipe namespace* in RMS is analogous to a smart file directory. The *namespace* provides long-term recipe storage capability.

Basic operations are part of the *namespace* specification. For example, recipes can be copied from one *namespace* to another, copied within a *namespace*, renamed, created, updated, and deleted. They can be downloaded to, and uploaded from, an application called a *recipe executor*. Several recipes can be *linked* together to form a set.

The *namespace* also can serve as a recipe pool that can be shared by a group of equipment of a common type. The *distributed recipe namespace* is a *namespace* able to utilize and manage recipe storage provided by such an equipment group.

2.2.3 *Recipe Namespace Manager* — The *recipe namespace* itself is passive, a container for recipes. The **recipe namespace manager (manager)** provides the dynamic element that manages the *namespace*. The *manager* represents the interface for the *namespace* to the external world and the internal decision authority within the *namespace*.

3 The recipe object in Figure 2.1 is not itself a standardized object. However, two recipe subtypes defined in Section 3 are standardized objects.

2.2.4 *Recipe Executor* — The **recipe executor (executor)** is the component of an *agent* that understands the contents of a recipe and is able to **verify** their syntactical correctness and to **validate** that it can be executed under the current configuration. It **executes** a recipe by reading its contents and applying them appropriately in order to achieve a desired result.

The *recipe executor* also may provide additional limited storage capability, but it is restricted from modifying existing recipes, except under special conditions, to prevent unexpected and/or unwanted change.

2.3 *Agent* — As used in RMS, an **agent** is a system, or subsystem, in a factory, that has a physical aspect. It consists of one or more applications that provide and/or use *service resources*, such as a *recipe namespace manager* and/or a *recipe executor*, as illustrated in Figure 2.2.

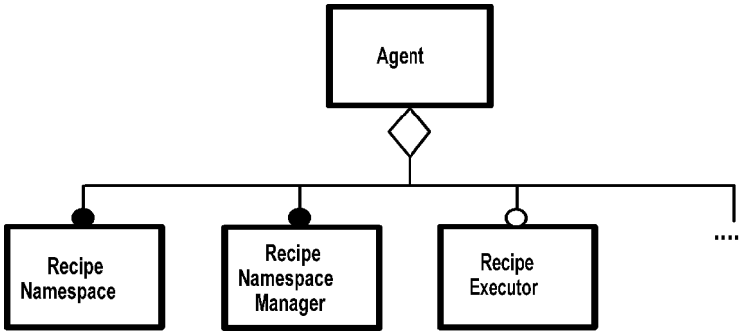

Figure 2.2

An Example of an Agent and Component Applications

Agent is a generalization that covers traditional *equipment*, supervisory or aggregate systems such as stations, cells, and clusters, and intelligent subsystems within *equipment*, such as process modules within a cluster. Other types of *agents* may also be implemented, such as those dedicated to *recipe namespace management*.

An *agent* that provides a *recipe executor* is called an **executing agent**. **Equipment** is an *executing agent* with associated hardware to which the recipe applies and which it uses to do work.

2.4 *Implementations* — RMS can be implemented on different platforms within a factory. A host controller may only provide *recipe namespace management*, while a diskless cluster module may provide only *recipe execution*. Traditional *equipment* capable of operating in stand-alone mode is required to provide a *recipe namespace* that is available on powerup as well as a *recipe executor* component. Additional examples of implementations of RMS are provided in Related Information 1.

RMS defines the services provided through an external interface. Communications between or within applications that are internal to an *agent* are not covered by RMS, so long as the attributes and operations comply with RMS requirements.

3 Recipes

This section describes the concepts concerning recipes.

3.1 *Motivations* — Definitions of recipe structure, attributes, and operations address the following issues:

- *maintenance of recipe attributes for management and traceability,*
- *provision of a standard method for transferring attributes separately, as well as with the recipe body,*
- *reduction or elimination of the need for dedicated recipe editors,*
- *support for applications requiring multiple recipe types, such as formats or language syntaxes,*
- *run-to-run control,*

- *support for sorting by factory approval and* ***certification levels****,*
- *the ability to tune a generic recipe for a specific installation of agent as well as the same agent over time, and*
- *the ability to share recipes across* executing agents *supporting the same recipe syntax or language(s).*

3.2 *Basic Concepts* — There are two types of recipes addressed by RMS, the **managed recipe** that is stored in a *recipe namespace* and the **execution recipe** that is stored in **recipe execution storage**. This section describes the basic concepts concerning both types of recipes.

3.2.1 *Types of Recipes* — There are three types of recipe objects shown in Figure 3.1. Most of Recipe Management is concerned with the management of recipes stored in a ***recipe namespace***. These recipes are called **managed recipes**. Recipes are also stored by the *recipe executor* in the **recipe execution storage**, and these are called **execution recipes**.

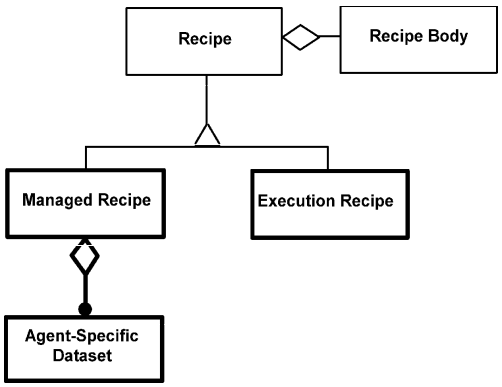

Figure 3.1

Recipe Types

The recipe supertype is an abstract type that shows the common elements and attributes of the *managed recipe* and the *execution recipe* and their differences. The recipe supertype is not itself intended to be implemented. That is, recipe object types formally defined in RMS are either *managed recipes* or *execution recipes*. As indicated by the light lines, the supertype is not a *standardized object*.

3.2.2 *Recipe Structure* — Both the *managed* and the *execution* recipe have a **body**[4] and a set of attributes. The *body* (contents) of the recipe contains the data used by the *recipe executor* for its execution process. The *body* is also not a *standardized object* and may not be accessed through Object Services.

Figure 3.2 illustrates all of the attributes defined by RMS that are common to both the *managed recipe* and the *execution recipe*. Not all of these attributes are required for a minimum implementation of RMS. The rules for setting the values of certain attributes, however, are different for the *managed recipe* and the *execution recipe*.

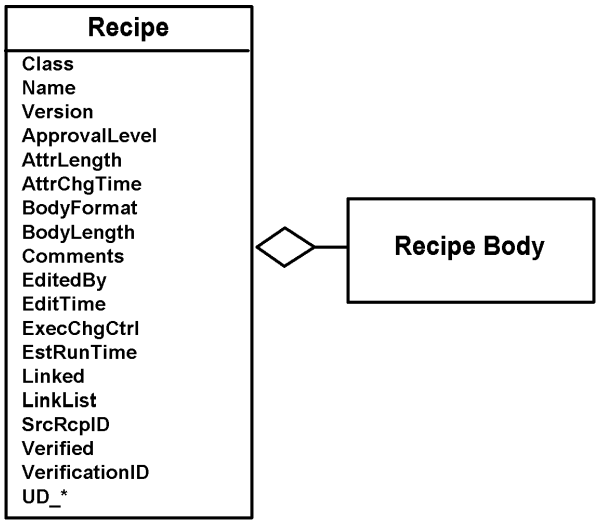

Figure 3.2

Recipe Supertype

The **identification attributes**, those attributes used for purposes of identification, are discussed in Section 3.3. The other attributes are discussed in later sections specific to each recipe type.

The *managed recipe* also may maintain a set of attributes that are specific to a single *executing agent*. The **agent-specific** attributes are technically attributes of the association between the recipe and the *executing agent*. In RMS, however, these attributes are treated as attributes of a component of the recipe called the **agent-specific dataset**, in order to allow them to be managed within the *namespace* along with the other attributes of the recipe. *Agent-specific* attributes allow the recipe to be personalized for particular *executing agents* (i.e., particular *recipe executors*).

---

4 The recipe's body corresponds to a "process program" in SEMI E5 and SEMI E30.

Because the *managed* recipe may be applied to multiple *agents*, it may have multiple *agent-specific datasets*. When a *recipe* is downloaded from a *recipe namespace* to a *recipe executor*, the *managed recipe* is transformed into an *execution recipe* by taking the *agent-specific* attributes of the *agent-specific dataset* for that *agent* (its *recipe executor*) and merging them into the attributes of the *execution recipe*.

Methods of storing a recipe or the components of a recipe are not dictated by RMS. However, the association between a recipe's *body* and its attributes shall be carefully maintained.

3.2.2.1 *Recipe Body* — The recipe **body** contains the reusable instructions, *settings*, *parameters*, and other data that the *recipe executor* reads to *execute* the recipe and control its operation.

A recipe *body* may be in one of two basic forms: **source** (text) or **object**. The purpose of the *source form* is to encourage the use of human-readable and human-editable text. The *object form* supports the types intended only to be read and applied by machines and other automated systems, such as those produced by vision systems or CAD programs, as well as proprietary formats created and used by the *recipe executor*.

3.2.2.1.1 *Source Form* — Recipes that are created and modified by a form of editor should be available in *source form*. The **source form** of a recipe *body* is equivalent to a text file, such as ASCII or JIS-8, that can be created, read, printed, or modified with any "standard" text editor[5]. The *executing agent* is not required to supply such a text editor, but its *recipe executor* should be able both to read and write recipes in *source form*. Editing is not covered by RMS.

The *source form* of a recipe may be copied from one *namespace* to another *namespace* and from a *namespace* to a *recipe executor's recipe execution storage*, and all *recipe executors* with access to a given *namespace* are assumed to use the same recipe language and the same functionality.

Two restrictions are placed on the interpretation of the text transferred between agents:

- *In ASCII, the valid characters are from space (20_{16}) through "~" ($7D_{16}$), plus the tab (09_{16}), linefeed ($0A_{16}$), carriage return ($0D_{16}$), and form feed ($0C_{16}$) characters. The space and tab are considered as* ***horizontal whitespace*** *characters, and the carriage return and form feed characters are considered* ***vertical whitespace*** *characters. The line-feed character is used to define the end of a line of text (eol).*
- *The meaning of a recipe should not depend on embedded control characters, or specific* whitespace *characters, or the existence of* whitespace *at the end of a line.* Horizontal whitespace *characters should be treated alike for purposes of interpretation, as should* vertical whitespace *characters. All other characters, aside from those specified here, are discouraged and, if found, should be removed from the text prior to transfer.*

In general, the recipe language and syntax used within a recipe *body* are beyond the scope of recipe management. However, to reduce arbitrary variability for the *user*, the following "rules" are recommended:

- *Case is not significant for the purpose of comparison or meaning but should be retained as encountered, since it is often used to enhance readability. For example, the tokens alarmlevel, ALARMLEVEL, and AlarmLevel should all have the same meaning.*
- *Comments may be included in the text and are ignored in recipe interpretation. Two types of comments are defined: comments that begin with the character pair "//" continue to the end of the line, while comments that begin with the character pair "/*" are terminated only by the character pair "*/". A comment of the second type is terminated by the first character pair "*/" that follows. Comments may not be nested. Nested comments are not honored and may generate a syntax error.*
- *The minimum line consists of a single line-feed character.*

In addition, recipe languages that support **external references** to other recipes should use the text form defined for the recipe *identifier* (Section 3.2.3.4) and for the recipe **specifier** (Section 3.2.4.1.2).

The supplier of the *recipe executor* shall provide documentation that formally defines each recipe language that the *agent* supports. Two forms are required, one for the *user*, who must be able to write correct recipes, and one that covers both the syntactical and lexical structures of the language using a formal descriptive protocol such as the Backus-Naur Form (BNF). The second form of documentation allows parsers to be built that are capable of pre-checking a recipe for syntactic and lexical correctness prior to downloading it. Pre-checking improves efficiency and

5 The intent is not to specify the text editor, but rather to allow a user to modify text on the basis of clear guidelines and restrictions that may apply. Source form recipes are highly desirable to reduce the proliferation of, and need for, proprietary editors that are dedicated and language-specific.

performance by reducing the demand for recipe checking by the *recipe executor*.

The final responsibility for determining the correctness of a recipe shall belong to the *recipe executor*.

3.2.2.1.2 *Object Form* — Recipes that cannot be translated into a meaningful *source form* include datasets derived from vision systems or mechanical systems through a special hardware-dependent "teach" operation. Such recipes may remain in a proprietary *object form* and may or may not be applicable to other *recipe executors* because of their hardware dependencies.

For greater efficiency, *recipe executors* may rewrite (e.g., **tokenize**) a *source* recipe to generate a proprietary *object form* of the recipe. This is called the **derived object form**, to distinguish it from *object form* recipes that never exist in *source form*. The *agent's* manufacturer may choose to make this form available for transfer as well. From the point of view of the *supervisor*, the *body* of a recipe that is in *object form* is an unstructured binary vector or string. In general, it cannot be edited on a *supervisor*, and the *source form* of the recipe is still required. *Derived object form* recipes that cannot be used by multiple *recipe executors* should not be stored in *recipe namespaces* that are shared.

Where both the *source form* and the *derived object form* exist, the *identifier* for the *object form* of the recipe shall be different from that of the *source form* in some recognizable way. The supplier of the *recipe executor* shall provide a documented method for distinguishing one from the other within the *identifier*. This method must be such that the relationship of the *derived object form* recipe from the original source recipe is obvious to the user. (See Section 3.2.3.4 for examples.) An attribute of the recipe, SrcRcpID, is provided to retain the relationship between the *source form* recipe and the *derived object form* recipe.

3.2.3 *Recipe Identification* — The ability to properly and unambiguously identify a recipe is critical to both recipe management and processing. It is also important for the user that identification be logical. RMS defines three elements used to uniquely identify a recipe within any *recipe namespace*. These three elements are the recipe's **class**, **name**, and version. Each element is expressed as a text string, and they are concatenated together to form the **recipe identifier** (see Section 3.2.3.4).

3.2.3.1 *Recipe Name* — The **recipe name** is a user-defined text string that may be used to encode the technology, layer, manufacturing area, and other characteristics. The *recipe name* alone is not necessarily unique within either a given class or a given *namespace*.

The *recipe name* is subject to the conventions for text as defined in Section 1.5.1.

3.2.3.2 *Recipe Class* — A **recipe class** is a formal grouping of recipes that have a common syntax. A class may contain **subclasses** where the recipes within the *subclasses* operate in different environments or have different syntaxes from the parent *class* and/or from one another. A *subclass* is itself a *class* and may contain further *subclasses*.

The overall model for recipes is a set of *class* hierarchies or "*class* trees" defined by the supplier of the *recipe executor*. The major *classes* of recipes are the **PROCESS** *class*, the **SERVICE** *class*, and possibly other *agent-specific classes*. These are called **primary classes**. A **primary class** is a *class* that is not a *subclass* of another *class*.

The PROCESS *class* is a required *primary class* and is composed of recipes whose primary purpose is to increase the manufactured value of the production material[6].

The SERVICE class is an optional *primary class* used for recipes whose purpose is to maintain, prepare, calibrate, or test the operation of *equipment*. Typical *subclasses* of the SERVICE *class* might be CALIBRATION and CLEANING. SERVICE recipes that use the same recipe language as normal process recipes are not required to be placed into a separate *primary* class. The SERVICE class allows separation and use of a different syntax for special non-process purposes.

Other *primary classes* that do not fit the PROCESS or SERVICE may be provided by the supplier of the *recipe executor*. Such additional *classes*, and their function, shall be documented.

Recipes that are not in the PROCESS *class* or one of its *subclasses* shall not be used for production purposes. However, recipes within the PROCESS class may be used for purposes other than production.

Figures 3.3 and 3.4 show examples of object models of possible *classes* and *subclasses* for a wire-bonder and for a furnace with an automatic boat-loader subsystem.

6 Material handling systems, wafer metrology, wafer measurement, wafer inspection systems, as well as process equipment which directly changes the characteristics of the material, increase the value of the material in some way.

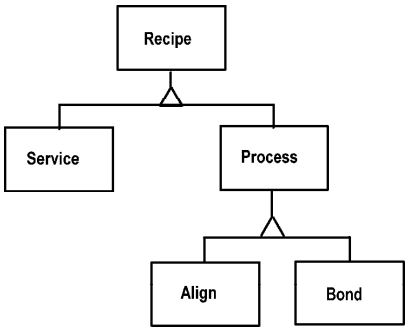

Figure 3.3

Wire-Bonder Recipe Classes

The text form of a recipe *class* shall conform to the convention for text (Section 1.5.1). Within the text form of a recipe *identifier*, a class is delimited on both ends by the forward slash character "/". The complete *class* specification within recipe *identifier* is formed by the concatenation of *classes* and *subclasses*, starting with the *primary class* and ending with the particular *class* of the recipe in question:

"/PRIMARY CLASS/SUBCLASS$_1$/SUBCLASS$_2$/…".

For example, a recipe named DryOx, version 4, within the NORMAL CYCLE *class* in Figure 3.4., has an *identifier* of

"/PROCESS/FURNACE/DIFFUSION/NORMAL CYCLE/DryOx;4".

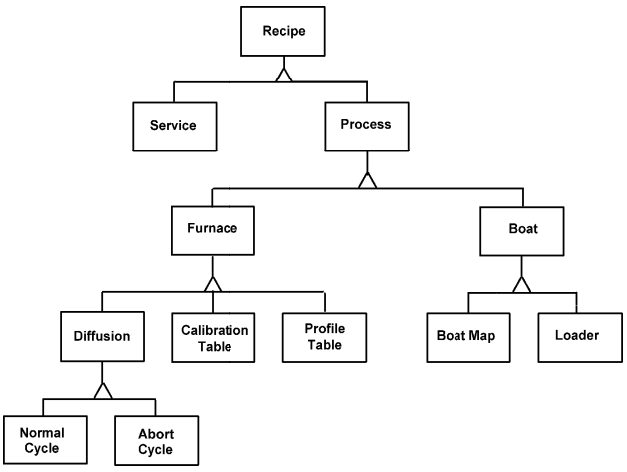

Figure 3.4

Recipe Classes for Furnace

The complete *class* specification, which always starts with the *primary class*, shall always be accepted. If *class names* within a *namespace* are all unique, however, the *service* provider may also accept a *class* designator consisting of a single *class*. The *identifier* in the example above then becomes

"/NORMAL CYCLE/DryOx;4".

3.2.3.3 *Version* — A recipe may evolve over time and exist in several versions. This allows the user to retain a recipe *name* over multiple versions and show the recipe's heritage. Different versions are identified by the **version** portion of the recipe's *identifier*. NOTE: More than one version of a recipe may be in use for production at the same time, not only within the factory, but also within a single equipment.

A **version** is a text string consisting of at least one character. A new *version* is either generated automatically by the *recipe namespace* or is assigned by the *user*. A *user* may assign any combination of text characters and punctuation marks to the *version*, except for:

- *the characters prohibited by the convention for text usage and the object identifier (Section 1.5.1),*
- *whitespace characters.*

It is recommended, but not required, that only upper case be used.

Numeric versions (**version numbers**) consist only of the digits "0" through "9" and one decimal point character "." and can be translated to a pure number. To avoid confusion and multiple *versions* with the same numeric value, *numeric versions* are further restricted as follows:

- *whole numbers (with no decimal point) may not start with a zero "0" followed by another digit, and*
- *decimal numbers (with a decimal point) may not start or end with the decimal point or end with a zero following the decimal point.*

For example, *version numbers* "09", "1.", ".5", and "1.670" are prohibited. The proper forms with the same numeric values are "9", "1", "0.5", and "1.67". The *user* may assign a *version* of "0" but not of "00".

Versions that are assigned automatically have additional restrictions:

- *They shall be* numeric versions*, excluding the decimal point and with a minimum value of "1".*
- Versions *shall be assigned incrementally. When assigning a* version *for a recipe with a given* class *and* name*, if no other recipe exists within the* namespace *with that* class *and* name*, then a* version *of "1" is assigned. Otherwise, the highest* existing version *already in use for that* class *and* name *is determined, and the new* version *is assigned a value equal to that value plus 1. For example, if the highest* version *in use has a numeric value of 5, then the next* version *assigned would be "6". To compare* two versions*, they are converted to upper-case and compared character by character.*

3.2.3.4 *Recipe Identifiers* — The **recipe identifier** is formed from the concatenation of the recipe's *class*, *name*, and *version*, in that order:

"/$CLASS_1$/$CLASS_2$/…/$CLASS_n$/NAME;VERSION"

where $CLASS_1$ is a *primary class* and $CLASS_{i+1}$ is a *subclass* of $CLASS_i$. The recipe *name* follows *class*. *Name* and *version* are always separated by a semicolon ";" ($3B_{16}$).

Where $CLASS_n$ is a unique class name, the form becomes

"/CLASS/NAME;VERSION".

The recipe *identifier* is used for the OSS-required attribute ObjID and shall conform to restrictions imposed on ObjID (see SEMI E39). The total length of ObjID may have additional restrictions imposed by the protocol.

NOTE: There is no necessary relationship between a recipe's *identifier* or *name* and any file name(s) under which the *body* and attributes may be stored internally. The recipe *identifier* is a logical reference to the recipe that is independent of specific platforms and implementations. In particular, file services provided by operating systems may have naming restrictions, such as length, that are incompatible with the requirements of RMS. A recipe may be stored internally in different ways, such as in a single flat file, a set of related files, or a database. Actual storage methods shall be invisible to RMS.

Where suppliers supporting recipes in *source form* also rewrite them, as discussed in Section 3.2.2.1.2, a method based on the recipe *identifier* (ObjID) is required to both distinguish between the two *forms* and to recognize the relationship between the original *source form* recipe and the derived *object form* recipe. The method may use any of the three elements of the *identifier* for this purpose. For example, recipes in *object form* may be placed in a separate *subclass* called "/OBJ/", or a suffix such as ".obj" might be appended to the user-defined *name* of the original recipe.

3.2.3.4.1 *Default Recipe Identifiers* — It is always possible to reference a recipe by specifying its *full identifier* (*class*, *name*, and *version number*). In certain cases, described in Section 3.2.4.1.1, it may also be necessary to specify the *namespace* of a recipe. However, it is not always necessary to specify all components of the *identifier*. Specifically, *class* and/or *version* may be omitted when a recipe is *selected* for execution and within an *external reference*. *Namespace* is not normally specified. Rules define default values