management services, to change a recipe, or to change the *namespace* (for example, to *store* a new recipe or *delete* an existing one), from an entity other than an attached *DRNS segment*. If the change involves a specified *agent* (for example, *verify* or *upload* requests), then the *DRNS manager* shall delegate the operation to the *DRNS segment* provided by that *agent*. Otherwise, operations are preferably delegated to a *DRNS master segment*.

10.3.4.2 *Internal Change Requests* — A *change request* may be initiated by a specific attached *segment*. For example, a local operator may want to change the *generic* attribute EditedBy (illustrated in Figure 10.3) or to rename a recipe. The *DRNS segment* then requests its *DRNS manager* for approval to make the specific type of change desired. The *DRNS manager*, upon receipt of the request, shall create a *change request record* (Section 10.3.4.4) for the recipe(s), specifying the *DRNS segment* that initiated the request, the type of change requested, and noting that the change was *segment*-initiated.

10.3.4.3 *Allowable Change Requests* — A *change request* may not be allowable. For example, a change to a *write-protected* recipe or a change to introduce a recipe *identifier* already in use may be denied immediately. If the change is allowed, then a *change request record* shall be created in non-volatile memory for each allowed *change request*.

Any operation that creates or changes a recipe body, a recipe attribute, or a recipe identifier, requires explicit *manager* approval. This includes the following operations: create, delete, modify (body), copy (new *identifier*), rename (new *identifier*), protect, verify, link, unlink, certify, de-certify, change read-write *generic attribute*, and change read-write *agent-specific attribute*.

The following requirements shall apply:

— A request to *create* a new recipe, or *copy* an existing recipe, with a specified *identifier* shall be granted only if that *identifier* is not in use for a *write-protected* recipe within the *namespace* as a whole (that is, no attached *DRNS segment* has a *write-protected* recipe using that *identifier*).

— A request from a *DRNS segment* to rename a recipe shall be granted only if no other *segment* has that recipe and the new *identifier* is not otherwise in use.

— A request from a *dedicated segment* to delete a recipe shall be denied only if the recipe is *write-protected*.

10.3.4.4 *Change Request Record Definition* — A **change request record**, as shown in Figure 5.7, consists of the following information:

— The *recipe identifier* (RecipeID),

— a *destination recipe identifier* (DestRecipeID) (used for copy and rename only, and otherwise null),

— the *segment specifier*, (Segment), of the *DRNS segment* that has requested a change or to which the change operation is or will be delegated,

— the specific type of change requested (ChangeType),

— a timestamp of when the request was received (Timestamp),

— an **operation identifier** (OperationID), an integer delegated by the *DRNS manager* that uniquely identifies a given *change request* within the *distributed recipe namespace* at large,

— a boolean (SegmentRequest) used to differentiate requests by attached *segments* from other requests (the *manager* responds differently in the two cases).

All of the above information, with the exception of the *segment specifier*, shall be determined at the time the *change request record* is created. In the case of externally initiated requests, delegation of a *segment specifier* may be postponed until the *change request* is **selected**.

The *operation identifier* is used to identify a specific change request by both the *namespace* and the delegated *DRNS segment*. It is passed to the *DRNS segment* when granting permission to make the requested change (Section 10.3.5) and when sending *segment change approval* to a *DRNS segment* (Section 10.1.2.2) for a previous *change request* made while the recipe was *locked*. It shall also be used for recipe management services defined in Section 12 that use an *operation identifier* (parameter RMOpID).

For *change requests* initiated by an attached *segment*, the boolean SegmentRequest is set to TRUE. Otherwise, it is FALSE.

The *change request record* shall be deleted when the requested change operation has been completed, whether successfully or unsuccessfully.

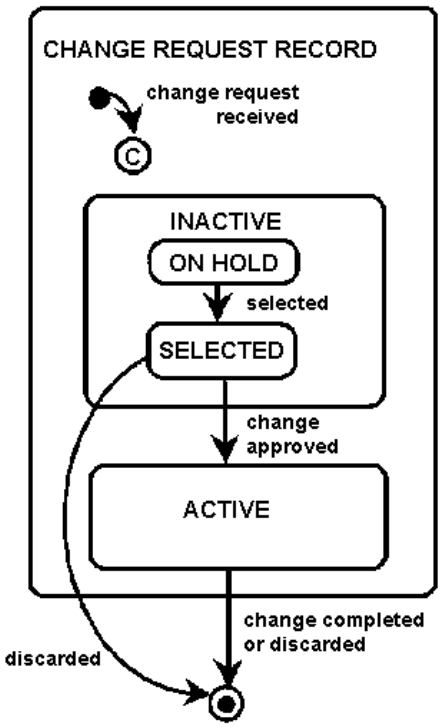

Figure 10.3
Change Request Record State Model

10.3.4.5 *Change Request Lifecycle* — This section describes the states of a *change request*.

A *change request* is received and examined. It is determined either to be allowed or not allowed.

If the requested change is allowed, then a *change request record* is created for the specified recipe by saving the information required. The *change request* is now INACTIVE and ON HOLD (see Figure 10.1). The *operations identifier* is set to a unique unsigned integer. If the change was initiated externally, a *segment* may or may not be designated until the *change request* is selected for approval. The remaining information is filled at this time. The *change request record* is placed in a queue for that recipe in non-volatile memory.

The *change request* remains **on hold** until the *DRNS manager* selects it as the next change and it becomes **selected**. If a *segment* has not already been designated, then one is determined at this time. The *selected* recipe may be discarded after evaluation due to changes in circumstances since it was originally put *on hold*. Otherwise, the *change request* becomes **active**, and the *change request record* is added to any attached *DRNS recorder*. The *manager* is ready to begin its negotiations with the designated *segment* or to approve a *segment*-initiated *change request* already in progress, at this point.

These stages may occur in rapid succession if the request is initiated by a *segment* and the recipe was not *locked*.

Selection occurs in one of two ways: either a *change request* is received when the specified recipe is *unlocked (Section 10.3.4.5)*, or else a currently *active change request* completes, and a new *change request* is *selected* from the queue of *change request records on hold* for that recipe. Rules for selection may vary and are beyond the scope of RMS. For example, rules could be sequence-oriented (first-in, first-out), or they might consider the types of operations specified. For example, requests for *verification* might be given precedence over requests to *link*.

When a *selected change request* becomes *active*, the *manager* adds the *change request record* to any attached *recorder* and begins negotiations with the delegated *segment*.

If the *change request* was *segment*-initiated (Section 10.3.5), the *segment change request* is approved. The *DRNS manager* shall initiate a *segment change approval* to the *DRNS segment* (Section 10.1.2.2).

There are three possible scenarios for the negotiations with the *segment*: an externally initiated request, an internally initiated request from a *segment* when the recipe specified was *unlocked*, and an internally initiated request from a *segment* when the recipe specified was *locked*. These three types of scenarios are illustrated in Table 10.5. NOTE: In both of the first two cases, the *segment* sends a request to make a change with RMDChangeRequest.req. However, in the first case, the initiation of that *change request* was external, and the *segment's* request is its required response to any change request that it receives.

Table 10.5 Active Change Request Negotiations

Manager	*Segment*
Externally Initiated Change Request	
An externally initiated change request becomes active. The manager requests the delegated segment to make the change:	
RMNAction.req ->	
	The segment responds that it will make the change and notify the requestor when done.
	<- RMNAction.rsp
	The segment requests approval to make the change:
	<- RMDChangeRequest.req
The manager approves the request immediately:	
RMDChangeRequest.rsp ->	
	The segment completes the change and notifies its manager:
	<- RMDComplete.nfy
	The segment notifies the requestor of the change (in this case, its manager) that the change is complete.
	<- RMNComplete.nfy
Segment-Initiated Change Request for Unlocked Recipe	
	The segment requests approval to make the change:
	<- RMDChangeRequest.req
The manager approves the request immediately:	
RMDChangeRequest.rsp ->	
	The segment completes the change and notifies its manager:
	<- RMDComplete.nfy
	The segment notifies the requestor of the change (in this case, its manager) that the change is complete.
	<- RMNComplete.nfy
	The segment notifies the initial requestor of the change that the change is complete:
	RMNComplete.nfy ->
Segment-Initiated Change Request for Locked Recipe	
	The segment requests approval to make a change: <- RMDChangeRequest.req
The manager puts the change request on hold:	
RMDChangeRequest.rsp ->	
The change request later becomes active. The manager now approves the segment's earlier request to change generic attributes:	
RMDSApproveAction.req ->	
	The segment responds that it intends to make the change:
	<- RMDSApproveAction.rsp
	The segment completes the change and notifies its manager:
	<- RMDComplete.nfy
	The segment notifies the initial requestor of the change that the change is complete:
	RMNComplete.nfy ->

10.3.4.6 *Change Request Completion* — The *DRNS manager* is informed of the completion of an approved change either immediately or through a later notification using the RMDComplete service.

There are two circumstances when a *DRNS segment* notifies its *manager* that it has completed an approved change, illustrated by Figures 10.1 and 10.2 (b) for the services RMDSegChange and RMDSApproveAction respectively.

Figure 10.2 (a) illustrates the case where the *manager* approves an earlier change request and the *segment* makes the change before responding. In this case, RMDComplete is not sent. However, if the *segment* responds before actually making the change, then it is required to send the notification RMDComplete when done.

When a requested change has been completed by the delegated *segment*, it notifies its *manager* by sending the notification RMDComplete as shown in Figures 10.1 and 10.2 (b). The type of change and results of the change operation dictate how the change shall be updated to the other attached *DRNS segments* of the *namespace*. After all affected *DRNS segments* have been properly updated, then the *change request record* corresponding to the completed change shall be removed from the queue. If there are still *inactive change requests*, the next *request* is selected and then made *active*.

All other *DRNS segments* with the same logical recipe shall be updated as appropriate. A change in *agent-specific datasets* only shall be updated to *master segments*. A change to attributes only shall cause only the attributes of other instances of that recipe to be updated. If the *logical recipe* was changed, then all *segments* having a copy of that *logical recipe* shall be updated.

When all necessary updates have been performed, the *DRNS manager* shall discard the *change request record*. It shall either request its *DRNS recorder* to remove the *change request* for that change or to add the next *change request* to be approved. Then the *DRNS manager* shall select another *inactive change request*. If no further *change requests* exist for that recipe, the *DRNS manager* shall remove the last *change request* from any attached *DRNS recorder*.

10.3.4.7 *Change Management Example* — As an example of change management, if an *operator* at Equipment A wants to *link* Recipe R;3 within the *distributed recipe namespace* D, the resulting changes to the recipe's *generic header* must be passed to all other attached *DRNS segments* within the *namespace* that contain Recipe R;3 before any other change is approved.

In this example, if a request from *segment* A to *link* Recipe R;3 is received by the *manager*, and no *inactive* requests exist, and if Recipe R;3 is not already both *linked* and *write-protected*, and if the request is to be approved, the *DRNS manager* creates a *change request record* for that recipe, which *locks* the recipe. The *DRNS manager* approves the request and denies subsequent requests until the approved change has been made and other *segments* with an *instance* of that recipe have been updated appropriately.

10.3.5 *Segment Change Request* — The **segment change request** operation is a request made by an attached *DRNS segment* to approve a specific type of change. The *DRNS manager* shall respond by either approving the request, denying it, or putting it on hold. When putting the requested change on hold, the *DRNS manager* shall return the *operation identifier* delegated in the corresponding *change request record*.

The *segment change request* operation is invoked with the service RMDSegChange sent by an attached *DRNS segment*.

10.3.6 *Segment Action Complete* — A *DRNS segment* that has requested and received permission to change a recipe shall notify the *DRNS manager* when the action is complete, whether normally or abnormally. Also, when the *manager* approves an earlier request made by a *segment*, and the *segment* makes the change after responding to the approval, the *segment* shall also notify its *manager* of the completion of the change. The notification allows the *DRNS manager* to request the updated recipe and send it to other *segments* as appropriate.

The *segment action complete* operation is invoked with the service RMDComplete, sent by an *attached segment*.

10.3.7 *Segment Notification* — A *dedicated segment* shall notify its *manager* of any change to an *agent-specific dataset*. These changes are pre-approved. They are not preceded by formal requests for change and do not generate change request records. Upon receiving notification of such a change, the *DRNS manager* shall update all *master segments* with the new *agent-specific dataset* values. An example of this case is illustrated in Table 10.6.

The *segment notification* operation is invoked with the service RMDNotify, sent by an attached *segment*.

Table 10.6 Notification of Change to an Agent-Specific Dataset

Manager	*Segment*
	Receives request from local operator to certify recipe.
	Local operator certifies recipe for this agent.
	The segment stores the generic attribute sent by its manager.
	<- RMDComplete.nfy
	The segment notifies its manager that the change has been made.
	<- RMNComplete.nfy
The manager requests the new set of agent-specific attributes:	
RMNRetrieve.req ->	
	The segment sends the generic attributes to any requestor:
	<- RMNRetrieve.rsp
The manager updates all master segments with the new agent-specific attributes for that agent and recipe.	

10.3.8 *Get Change Requests* — The **get change requests** operation returns a list of *active* and *inactive change requests* (in that order) for a *locked* recipe. If the recipe is not *locked*, an empty list is returned. Otherwise, where the *active change request* record is returned first in the list, followed by any *pending change requests*.

The *get change requests* operation is invoked with the service RMDGetChangeRequests.

10.3.9 *Rebuild Distributed Recipe Namespace* — The **rebuild distributed recipe namespace** operation is used to rebuild a previous *namespace* by *reattaching* all of its *attachments* to a different *namespace*.

The *DRNS manager* must first get a list of *DRNS segments* to be used. This list may be provided by either the service user or by a *DRNS recorder* specified by the service user. It then **reattaches** the *DRNS recorder* (where specified) and each of the specified *DRNS segments*. The reattach operation alerts the reattached object that the communication linkages to its *DRNS manager* may have changed and that its *inactive change requests* have been lost.

Next, the *DRNS manager* obtains the list of recipe *identifiers* stored by each *DRNS segment*. Finally, when a *DRNS recorder* has been provided, the *DRNS manager* completes the *rebuild* operation by checking the *DRNS recorder* for any *change request record* in progress at the time the original *namespace* was lost. The *DRNS manager* shall determine if the change was completed (through comparisons of recipe *descriptors*) and, if so, shall update the other *segments* as if it had just received a *segment change completion* notification.

Note that only the currently *active change request record* is retained. Other *inactive change requests* made while a recipe was *locked* have been lost with the original *DRNS manager*.

Note also the potential exists for *agent-specific datasets* stored by *dedicated segments* to be more recent than the corresponding *datasets* stored by *master segments*.

The *rebuild distributed recipe namespace* operation is invoked with the service RMDRebuild.

10.4 *Tables of Operations* — This section provides the operations defined for each object within the *distributed recipe namespace* capability. An additional column for the object authorized to invoke the operation is provided for tables in this section.

10.4.1 *Segment Operations Table* — Table 10.1 lists all the operations defined for *DRNS segment*.

Table 10.1 Distributed Recipe Namespace Segment Operations

Operation	*Invoked by*	*Description*	*Reqd*
Segment change approval	The *manager* of the attached *distributed recipe namespace segment.*	A change to a recipe requested earlier by a segment is now permitted.	Y

10.4.2 *Recorder Operations Table* — Table 10.2 lists all the operations defined *for distributed recipe namespace recorder*.

Table 10.2 Distributed Recipe Namespace Recorder Operations

Operation	*Invoked by*	*Description*	*Reqd*
add segment record	the *recorder's manager*	A *segment specifier* is added to the current list.	Y
delete segment record	the *recorder's manager*	A *segment specifier* is removed from the current list.	Y
add change request record	the *recorder's manager*	A currently active *change request record* is added for a specified recipe.	Y
delete change request record	the *recorder's manager*	An existing *change request record* is removed for a specified recipe.	Y
get change request records	any	The *change request records* for one or more recipes are requested.	Y

10.4.3 *Manager Operations Table* — Table 10.3 lists all the operations defined *for distributed recipe namespace manager*.

Table 10.3 Distributed Recipe Namespace Manager Operations

Operation	*Invoked by*	*Description*	*Reqd*
segment change request	an attached *segment*	A *segment* requests approval for a specified type of change on a specified recipe.	Y
segment change complete	an attached *segment*	A *segment's* notification that an approved change has been completed, either normally or abnormally.	Y
segment notification	an attached *dedicated segment*	A *dedicated segment's* notification that an *agent-specific dataset* has been changed.	Y
get change requests	any	A request for the list of existing change requests for a specified recipe.	Y
rebuild distributed namespace	an *authorized user*	A *manager* is requested to rebuild a *distributed recipe namespace* using a specified *recorder*.	Y

11 Recipe Executor Operations

The operations of the *recipe executor* defined by RMS are: **download and verify, verify, upload, rename, get available storage, delete, select, deselect,** and **change notification.** In addition, the *recipe executor* shall provide object services as described in Section 11.1.

Recipe executor operations may be invoked by an operator or through *recipe executor* services defined in Section 14. In most cases, service scenarios consist of a single message request from the service user and a corresponding response from the *recipe executor*. This case is illustrated in Section 14.1.

Scenarios are shown only for operations that differ from this typical case. For example, some operations may require more time to complete, such as *recipe verification* and the *select* operation that may be performed on more than one recipe and may require interactions with a *recipe namespace* to complete. In this case, the initial response to the message service request may only indicate the intent to perform the operation. The *recipe executor* informs the service user of the completion of each individual operation by sending the notification message RMEComplete.

In addition, in restricted circumstances, the *recipe executor* is required to inform a *namespace* that a recipe body originally downloaded from that *namespace* has been changed.

11.1 *Object Services Operations* — A *recipe executor* shall comply with Object Services Standard specifications for *fundamental requirements* and with the requirements for Owner Objects. Support for Filtering is optional.

A *recipe executor* owns the *execution recipes* that it stores, and it is *owned by* the *agent* that provides *recipe executor*. The owner relationships are used by Object Services to define a *recipe specifier*.

The *recipe executor* shall support operations for Get Attributes and Set Attributes for its own attributes. It shall support the operation of Get Attributes for the execution recipe. (Note that all attributes of the execution recipe are read-only and may not be set through the interface. It shall support the Get Type and Get Attribute Name operations for the object types of *recipe executor* and *execution recipe*.

11.1.1 *Execution Recipe Specifier* — The object specifier for an *execution recipe* includes the object type and *identifier* of the recipe executor, followed by the identifier of the recipe, including its *originating namespace*. For example, to specify an *execution recipe* "NS-MOM>PROCESS/ABC;5>" that is stored by a *recipe executor* named "RE-Etch" owned by "Agent-:Etch01", the object specifier would be

"Agent:Etch01>RcpExec:RE-Etch>NS-MOM>PROCESS/ABC;5>".

11.2 *Description of Operations* — This section describes the operations of the *recipe executor*.

11.2.1 *Recipe Download and Verify* — A recipe is **downloaded** when it is transferred (sent) to the *recipe executor*. *Downloaded* recipes that do not have the Verified attribute set to TRUE shall be immediately *verified*. Otherwise, it is not necessary to *re-verify* a recipe. Any *verification* errors shall be reported to the initiator of the *download* operation.

For further discussion of the *verification* operation, see Section 11.2.2.

Normally, if the *identifier* of the recipe *(namespace name, recipe class, recipe name, and version number)* is already assigned to an *execution recipe* already existing in storage, the *download* is refused. However, the initiator of the download operation may optionally request a **forced overwrite** of any such pre-existing recipe. In this case, if the existing recipe is not currently *selected*, the new recipe replaces the older one. A request to *delete or overwrite* a currently *selected* recipe shall be denied.

The body of a recipe currently being edited shall be protected from inadvertent change or overwriting by a recipe with the same identifier that is downloaded during this time. If the downloaded recipe is accepted (stored), the equipment shall require the operator either to save the edited recipe to a new (unused) identifier or to discard it.

The *download and verify recipe* operation is invoked with the RMEDnldVer service.

11.2.2 *Recipe Verify* — This is a required operation that *verifies* a specified *execution recipe*.

Verification requires reading the recipe's body. For a *source form* recipe, the *verification* operation checks the contents of the recipe for syntactical correctness. If supported, the *recipe executor* may also include one or more checks for semantic correctness as part of its *verification* operation. For recipes in *object form*, *verification* checks the format of the contents to achieve the same effect.

A *verified* recipe merely indicates that it is technically correct and can be executed. It does not indicate that it can be properly executed for the current hardware configuration. Checking to ensure a recipe will execute properly is called **validation** and shall only be performed at the time recipes are *selected* for execution.

A recipe that has syntactical or format errors fails the *verification* process. Information concerning errors shall be reported to the initiator of the *verification* process. This information is required for the user to be able to make appropriate corrections.

Information returned to the *namespace manager* at the conclusion of the *verification* process includes:

- *All* external references*, required for the ExtRef attribute of the* managed *recipe,*
- *All* variable parameter definitions*, required for the Parameters attribute of the* managed *recipe,*
- *The estimated or nominal time of the recipe, in seconds, used for the EstRunTime attribute of the* managed recipe *(optional).*

These items are not returned if the recipe fails *verification*. In this case, the *recipe executor* shall provide sufficient information to the user for identification of the type of error and where within the *body* it occurred, for at least the first error encountered.

The *external references* that are returned shall always include the *name* of the recipe referenced, and any referenced *namespace*, in the proper form for a *recipe identifier*. If the *external reference* specifies a *namespace* other than "Default" (Section 6.4), the *version* is required to be fully specified for *verification* to be successful. Otherwise, *version* shall be left exactly as specified in the *body*. Omission of version allows

"the best version" to be determined by the *namespace* at a later stage.

The *verify* operation is invoked explicitly with the RMEVerify service. It may also be performed as part of the *download and verify* operation as discussed in Section 11.2.1.

11.2.2.1 *Derived Object Form Recipes* — For downloaded recipes that are to be stored, the *recipe executor* may wish to re-write or compress a *source form* recipe into a *derived object form* recipe (Section 3.2.2.1.2) as part of the *verification* process. (The rewritten form may be of any format and is typically **not** a form of "machine-executable code".)

Where downloaded recipes are converted to a *derived object* form prior to storage, a method of re-converting to *source form* is required. Note that the capability to retain user comments is desirable, but not required, where this process is adopted.

The new *derived object form* recipe inherits the attributes of the *source form* recipe with the following exceptions:

- *The* class *or* name *of the* object form *shall be changed in a systematic and documented manner that both makes clear the* object form *derivative and its heritage from the* source form *original. For example, the object form may add an extension to the* name *of the original recipe.*
- *The* BodyFormat *attribute shall be changed to reflect the* object form*.*
- *The attributes* ExecLength*,* ExecChgTime*,* BodyLength*, and* EditTime *shall be updated.*

If the attribute ExecChgCtrl requires the *originating namespace* to be notified of changes, then this request applies to the creation of the *derived object form recipe*, and the ChangedBody attribute is set to TRUE until the recipe has been *uploaded* successfully by the *namespace manager*.

The *source form recipe* from which the *derived object form* was obtained should not be deleted automatically.

11.2.2.2 *Verification ID* — The attribute VerificationID is used for *derived object form* recipes and is provided for the *recipe executor* **identification code**. This is a text string containing a unique and persistent "signature" or identifier for the *recipe executor*. This is used to detect *derived object form* recipes that were derived by earlier software versions of the *recipe executor*, or by versions used by other *agents*, that may have obtained different results.

The *identifier* of the *source form* recipe of the *derived object form* recipe is stored in the attribute SrcRcpID of the *derived object form* recipe.

11.2.3 *Recipe Upload* — **Recipe upload** is the operation that returns an *execution* recipe from the *recipe executor's* storage to the service user. It is required for *recipe executors* that are able to *originate* or modify recipes. *Recipe upload* allows recipes to be managed in a *recipe namespace.*

The *upload recipe* operation is invoked with the RMEUpload service.

11.2.4 *Recipe Rename* — The **recipe rename** operation causes a recipe to be assigned a new *identifier*. If the new *identifier* is already in use by a pre-existing *execution recipe*, or if the recipe to be *renamed* is currently *selected*, the request shall be denied. This is a required operation.

This service is provided to maintain synchronization between *execution recipes* and the *managed recipes* within an *originating namespace*.

The *rename recipe* operation is invoked by the message service RMERename.

11.2.5 *Get Available Storage* — The **get available storage** operation is used to determine the size of the remaining recipe storage capacity, in bytes. The value returned shall exclude any overhead requirements for storage of one *execution recipe*. That is, it shall be assumed that sufficient storage exists for a single recipe with a combined *attribute length* and *body length* less than or equal to the returned value. This is a required operation.

The *get available storage* operation is invoked by the message service RMESpaceInquire.

11.2.6 *Recipe Delete* — The **recipe delete** operation removes one or more recipes from the *execution recipe storage*. Any reusable (rewritable) storage used by a *deleted* recipe shall be made available for other recipes, and the *deleted* recipe shall be unavailable for all subsequent operations. This is useful to release storage for new recipes. This is a required operation.

Recipes stored in non-reusable storage that cannot be physically deleted shall be made unavailable for subsequent *selection*.

If the *recipe executor* does not have *execution recipe storage* (that is, if its sole storage is the *recipe execution area*), then the *delete* operation shall completely remove the specified recipes from the *recipe execution area*.

The *delete recipe* operation is invoked with the RMEDelete service. A request to *delete* a currently *selected* recipe shall be denied.

11.2.7 *Recipe Selection* — Recipe **selection** is the process of locating, *validating*, and preparing a recipe for execution within the *execution area*. Recipe *selection* is a required *operation* of the *recipe executor* that allows specification of initial (default) values of any *variable parameters* defined by the recipe. Initial parameter values specified at this time shall override the corresponding values contained in the ExecLinkParam attribute.

It shall be possible to explicitly specify a recipe for execution. It may also be possible to implicitly *select* a recipe, including initial *variable parameter* values, through other operations, such as within a higher-order "process job". Services for implicit methods and for starting execution itself are beyond the scope of RMS.

If the recipe class is omitted when specifying a recipe for *selection*, then the class "/PROCESS/" shall be assumed.

The request for *recipe selection* specifies one or more recipes to be *selected.* If a recipe cannot be located in the *recipe executor's* recipe storage area, then the *recipe executor* may optionally attempt to locate and *retrieve* it from the *originating namespace* specified.

Only a *linked* recipe (with the Linked attribute set to TRUE) may be specified by the service user. An attempt to *select* an *unlinked* recipe shall be denied. *Subrecipes* that are part of a *linked recipe set* are required only to be *verified* but not *linked*.

The *selection* process completes abnormally whenever an error is detected, including the failure to locate a specified recipe or to successfully store it, invalid parameter settings, an attempt to *select* an *unverified* recipe, and other *validation* errors.

The *select recipe* operation is invoked with the RMESelect service.

11.2.7.1 *Multiple Selection — Executing agents* that support *subclasses* of the PROCESS *class* shall provide at least one of two methods for the *selection* of multiple recipes: (1) they may either allow, or require, that a request for recipe *selection* explicitly specify a recipe from each of certain *subclasses*, and/or (2) they may either allow, or require, specification of required *subclasses* within a *linked* set of multi-part recipes. Such additional permissions and restrictions shall be included in the recipe management documentation.

For *executing agents* that support multi-part recipes (such as described in method 2 above), the *recipe executor* is responsible for *validating*, as a set, the *main* recipe and all its *subrecipes* listed in the LinkList attribute of the *main* recipe as a condition of execution.

If the *namespace* is specified as "Default", the *default namespace* is used. The *name* of this *namespace* is the value in the *recipe executor* attribute DefaultNamespace (see Section 6.4 and 6.7).

All recipes in the *linked recipe set* are considered to be *selected* if the *select* operation is successful.

11.2.7.2 *Validation* — **Validation** consists of type and range-checking of recipe settings and parameters, and of confirmation that the recipe is valid for the current hardware configuration. *Validation* shall occur as part of the process of *selection*. For *recipe executors* that require several different *classes* of recipe, *validation* may depend on the recipes that are *selected* as a set.

For example, a diffusion furnace may have a *class* of recipe that is used for the current gas configuration to map gases to valves. The furnace's normal process recipe that uses names of gases can only be *validated* in combination with the gas configuration table *selected* at the same time.

11.2.7.3 *Delegation* — If *delegated* recipes are specified in LinkList, it is the responsibility of the *recipe executor* to ensure that the designated recipe is accessible to the *recipe executor* of the *component agent* at the time it is *selected*. Otherwise, the *select* request shall be denied.

11.2.7.4 *Variable Parameters* — For recipes with *variable parameters*, new *initial values* for some or all of its *parameters* may be specified as part of the *selection* as a *parameter name* and *parameter initial value* pair. *Parameter values* specified as part of the *selection* take precedence over those *initial values* defined in the ExecLinkParam attribute. However, *parameter restrictions* may not be changed when *selecting* a recipe.

All *parameter values* that are set either prior to, or during[12], the execution of a recipe shall satisfy the conditions of the *parameter restriction.* If a *parameter restriction* is given in the ExecLinkParam attribute, then that *restriction* is used for all occurrences of the *parameter*.

11.2.7.5 *Message Scenarios* — There are two possible message flows between the *recipe executor* and the service user for the *select* service.

If the *recipe executor* is able to locate the recipes in its local storage, to *validate* them as a set, and to ready

12 RMS neither defines, nor prohibits, methods for changing parameter values during the execution process.

them for execution in a timely manner, then it may indicate the results of the *select* operation in the response message, as illustrated in Figure 11.1.

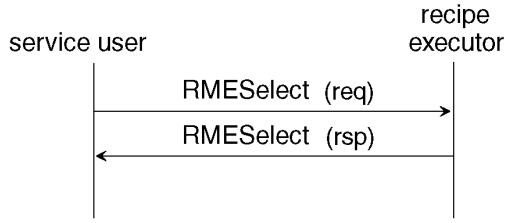

Figure 11.1

RMESelect Message Flow

Alternatively, the *recipe executor* may anticipate that the process of *selection* will require more time. In this case, it may send a response indicating the intent to perform the operation as soon as possible, prior to starting the operation. The notification message RMEComplete is sent when all specified recipes are *selected* or when a *selection* failure occurs. Exactly one notification message is sent, as illustrated in Figure 11.2.

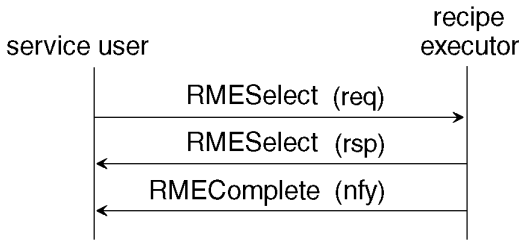

Figure 11.2

RMESelect with RMEComplete Notification

If all recipes are located and in local storage, the recipes are *validated* as a set.

11.2.7.6 *Selected Recipes* — Once a recipe and its associated *subrecipes* have been *selected*, they shall be fully protected from inadvertent (unintentional) change, including change through download, overwrite, deletion, and renaming operations.

The *recipe executor* attribute RecipeSelectID shall be set to the list of currently *selected* recipes, including all *main* recipes specified in the *select* request and all *subrecipes* within a *linked recipe set.*

The attribute RecipeSelectParam shall be set to the list of all *variable parameter definitions* currently in effect following *selection* and prior to the start of actual recipe execution.

11.2.7.6.1 *Select Errors* — If the *recipe executor* is unable to locate a recipe specified for *selection* in its storage, it is responsible for obtaining the recipe from the designated *namespace* included in the *recipe specifier*.

Possible errors resulting from the *select* operation include:

- *Failure to specify recipes from each of the* subclasses *that it requires,*
- *Failure to validate a recipe,*
- *Failure to locate a recipe in a specified* namespace*,*
- *Failure to locate a recipe, and*
- *An attempt by the user to* select *an* unlinked *recipe.*

The *select recipe* operation is invoked with the RMESelect service.

11.2.8 *Recipe Deselection* — A *selected* recipe is **deselected** by making it unavailable for execution for the next (subsequent) processing cycle, deleting its *identifier* from the RecipeSelectID attribute, and deleting its *parameter definitions* from RecipeSelectParam attribute. This is a required operation.

Only a *main* recipe that is currently *selected* may be specified for *deselection*. *Deselecting* a main recipe shall result in the simultaneous *deselection* of all of its *subrecipes.*

If a recipe is *deselected* during execution, this shall have no effect until the current process cycle has completed.

The *recipe executor* may have additional restrictions concerning *deselection*.

The *deselect recipe* operation is invoked with the RMEDeselect service.

11.2.9 *Get Execution Recipe Descriptor* — The **get execution recipe descriptors** operation returns the *descriptors* of a specified *execution* recipe: the *execution attribute descriptor*, the *generic attribute descriptor*, and the *body descriptor*, in that order. This is a required operation.

The *execution recipe descriptor* may be used to compare two *execution* recipes or a *managed* recipe and an *execution* recipe.

The *get execution recipe descriptor* operation is invoked with the RMEGetDescriptor service.

11.2.10 *Change Control* — It is important to the user to be able to control change for execution recipes. In addition, it may be important to store recipes that were *originated* by the *recipe executor* in the managed environment of a *recipe namespace.* The recipe attribute ExecChgCtrl is used to accomplish both purposes.

The ExecChgCtl attribute allows the user to control the modification of an existing *execution recipe* and its subsequent use. ExecChgCtrl is a binary value that specifies permission for behavior related to changes. ExecChgCtrl uses bit settings to indicate permission in order to facilitate the different possible combinations. Values given below are in decimal:

1 = permission to change the body of the *execution recipe*

2 = requirement to notify the *originating namespace* of permitted changes

4 = permission to select (including *re-selection* of an already *selected* recipe) a changed recipe

8 = permission to save the *last values* of *variable parameters* in the ExecParam attribute

11.2.10.1 *Changing Existing Recipes* — Whenever the body is changed, the attributes ExecAttrLength, ExecChgTime, BodyLength, and EditTime shall be updated. The attribute ChangedBody shall be set to TRUE and remain set until the recipe is uploaded successfully to the *originating recipe namespace.* The ChangedBody attribute is *reset* when the upload request is received. It is not included in the uploaded attributes. Note that the attributes ExecAttrLength and ExecChgTime are not kept in the *managed* recipe stored in the *namespace*, and that the attributes AttrLength and AttrChgTime are not maintained by the *recipe executor* and are recalculated for the *managed recipe* when stored.

If the ExecChgCtrl attribute permits both change and subsequent selection or *re-selection* (as for a currently *selected* recipe changed by the execution process itself), then the recipe's Linked attribute is not affected. Otherwise, if change is permitted but subsequent *selection* is not permitted, then the Linked attribute of the *main* recipe shall be *cleared* (set to FALSE) to prevent any subsequent *selection*, and the recipe shall be automatically *de-selected* at the end of any current processing cycle. It shall not be available for *selection* until it has been *uploaded*, *re-linked*, and *downloaded* again, with its changes, by the *originating namespace*.

11.2.10.2 *Creating New Recipes* — A new *execution recipe* may be *created* by various mechanisms. Such mechanisms are beyond the scope of RMS. However, all such recipes shall be assigned an *identifier* that is not already in use by an *execution recipe.* It is not required that an *originating namespace* be assigned immediately in all cases. However, it must be provided before the recipe can be *uploaded* and placed in the management system. New *identifiers* assigned automatically by the *recipe executor* shall use the next available *numeric version*, in conformance with Section 3.2.3.3.

A newly *created* recipe may not be executed. Its ChangedBody attribute shall be set to TRUE, and BodyFormat shall be set TRUE for non-text recipes. The attribute ExecChgCtrl shall be set to a value of 3 (1 plus 2) or higher. Other *non-mandatory* attributes shall be set to their default value. Since the Linked attribute is set to FALSE, the new recipe may not be *executed* as a *main* recipe. New recipes must be *uploaded* to a designated *originating namespace* for management purposes and subsequently *downloaded* after they have been *linked*.

11.2.10.3 *Building Derived Form Recipes* — A new recipe, with a new identifier, is also originated if recipe executor builds a *derived object form recipe* from an existing *source form recipe*. Requirements for the *derived object form recipe* are defined in Section 11.2.2.1.

The *manager* of the *originating namespace* shall be notified when a *derived object form* recipe is built from a *source form recipe* where ExecChgCtrl requires notification of change for the original *source form* recipe.

11.2.10.4 *Saving Last Value* — Saving the most recent settings specified by the user in the attribute ExecLink-Param is a protected activity requiring explicit permission in ExecChgCtrl. A *change notification* requirement in ExecChgCtrl applies to the change of this attribute.

Parameter values in ExecLinkParam are overridden by any parameter settings specified by the user at the time a recipe is *selected*. In this case, the last value of each parameter is stored in the appropriate *parameter definition* in the ExecLinkParam *attribute.* If *change notification* is specified, the *originating namespace* shall be notified of this change. The attributes ExecLength and ExecChgTime shall be updated whenever this change is made. The *body* of the recipe in *execution recipe storage* shall not be changed for this purpose.

11.2.10.5 *Change Notification* — If ExecChgCtrl requires that the *originating namespace* be notified of changes, the notification service RMEChange is sent to the specified *namespace* indicating the type of change that has occurred.

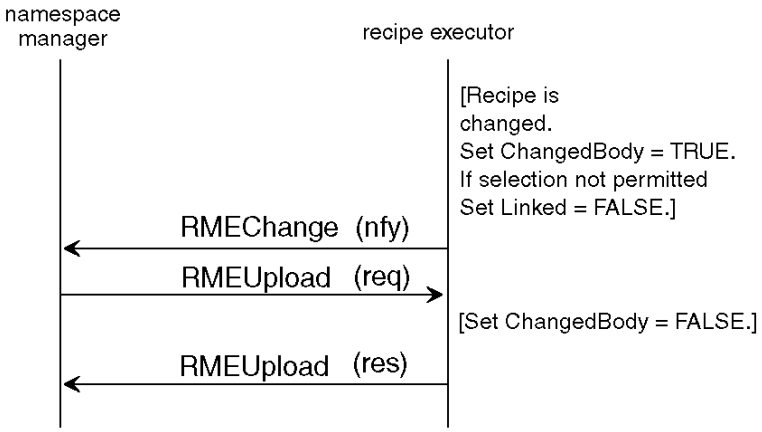

Figure 11.3
RMEChange Message Flow

The *recipe executor* may or may not have direct communication links with the *namespace* itself. However, in this case, the necessary links may be assumed to exist with the *supervisor* or other *owner* of the *recipe executor*, or it would not be possible for recipes to be *downloaded* in the first place.

The *originating namespace*, on receipt of a change notification, is responsible for uploading the changed recipe and for *synchronizing* through the service RMERename any change in *identifier* that might result from changing a *write-protected* recipe.

11.3 *Table of Operations* — Table 11.1 lists all the operations defined for recipes and *namespaces*.

The column labeled "Rqmt" indicates those operations that are required for *fundamental compliance* to the RMS *recipe execution resource.*

Table 11.1 Recipe Executor Operations

Operation	*Description*	*Rqmt*
Recipe download and verify	Transfer *execution recipe* to local storage. If the recipe is *unverified*, then *verify* it at this time.	Y
Recipe verify	*Verify* a stored *execution recipe.*	Y
Recipe unload	Transfer a recipe from local storage. Required if the execution process modifies an executed recipe or if the *recipe executor* is able to *create* recipes.	N
Recipe rename	Change the recipe's *identifier.*	Y
Get available storage	Calculate the amount of storage available for a single *execution recipe*, exclusive of formatting overhead.	Y
Recipe delete	Delete a recipe from local storage.	Y
Recipe select	*Select* recipes for execution.	Y
Recipe deselect	*Deselect* one or more recipes.	Y
Get recipe descriptor	Get the *descriptors* of a specified *execution recipe.*	Y
Change notification	Inform the *originating recipe* that a recipe has been *originated* or changed. Required if the *recipe executor* is able to make a change protected by ExecChgCtrl or to originate a recipe.	N

11.4 *Recipe Executor Event*s — The *recipe executor* shall provide the Recipe Select and Recipe Deselect events.

Recipe executors that are capable of *originating* a new *execution recipe* shall provide the New Execution Recipe event. *Recipe executors* that are capable of *changing* the body of an existing recipe shall provide an Execution Recipe Change Event. These events allow *service users* other than the *originating namespace* to receive notification whenever a new recipe is created or an existing recipe body is modified.

12 Recipe Namespace Services

This section defines *recipe management* and *namespace* services (see Table 12.1), which begin with a prefix of RMN. RMS services used by *distributed recipe namespaces* and by the *recipe executor* are defined in later sections.

Recipe management and *recipe namespace services* request that an operation be performed on a recipe within a *namespace* or on a *namespace*.

Recipe management operations that change recipe attributes are invoked with the RMNAction or RMNVarPar requests.

Certain activities that require extended time to complete may indicate the intent to comply in the response message and then provide confirmation with the RMNComplete message at the completion of each individual operation specified.

Table 12.1 Recipe Namespace Services

Message Service Name	*Type*	*Description*
RMNCreateNS	R	Create a new *namespace*.
RMNDeleteNS	R	Delete a *namespace*.
RMNRenameNS	R	Rename a *namespace* (assign a new *identifier*).
RMNSpaceInquire	R	Determine the available space for recipe storage.
RMNRcpStatInquire	R	Determine a recipe's *read-only* status.
RMNVersionInquire	R	Determine the best version of a recipe.
RMNCreate	R	Initialize new recipe with the specified body.
RMNUpdate	R	Transfer new recipe body.
RMNStore	R	Store a recipe.
RMNRetrieve	R	Retrieve a recipe and give to the requestor.
RMNCopy	R	*Copy* a recipe to a new recipe (originated).
RMNRename	R	*Rename* a recipe.
RMNAction	R	Perform an operation on a recipe (verify, link, unlink, …)
RMNVarPar	N	Modify *agent-specific variable parameter definition*.
RMNGetDescriptor	R	Get recipe *descriptors*.
RMNComplete	N	Notification that a requested action has completed.

12.1 *Recipe Management Message Parameter Dictionary* — Table 12.2 contains the definitions for message parameters, and their components, used by *namespace* services.

For purposes of transferring a recipe with its components, it is regarded as having three types of **sections**: a *section* containing the *generic* attributes, a *section* consisting of the *body*, and a *section* containing the attributes of the *agent-specific dataset*. The *generic section* is always transferred first, and the attributes AttrLength, AttrTimestamp, BodyLength, and EditTime are always sent first, in that order. This allows the length information to be immediately determined by the receiver. *Agent-specific dataset* sections are sent last, since the number of *agent-specific datasets* will vary.

Descriptors are also regarded as consisting of *sections* for the *generic descriptor*, the body *descriptor*, and zero or more *agent-specific descriptors*.

Each start of each *section* is identified with a text string reserved for that *section type*.

Table 12.2 Recipe Management Parameter Dictionary

Parameter	*Definition*	*Form*
AgentSpec_Desc	*Agent-specific dataset* descriptor.	Structure composed of AgentSpec_DescName, AgentSpec_DescLength, AgentSpec_DescTimestamp.
AgentSpec_DescName	Identifies start of *agent-specific dataset descriptor*.	Text = "ASDesc".
AgentSpec_DescLength	Value of AgentSpec_AttrLength.	Unsigned integer.
AgentSpec_DescTimestamp	Value of AgentSpec_AttrChgTime attribute.	Timestamp format: "YYYYMMDDHHMMSSCC".
AgentSpec_Section	A message section for an *agent-specific* dataset.	(List of) structures composed of AgentSpec_SectionName, AgentSpec_SectionData
AgentSpec_SectionData	*Agent-specific* attribute name/value pairs.	(List of) AttrSetting
AgentSpec_SectionName	Identifies start of *agent-specific* dataset.	Text = "ASDS".
AttrData	Value of attribute.	Varies with attribute.
AttrName	Name of attribute.	Text.
AttrSetting	Attribute name/value pair.	Structure composed of AttrName, AttrData.
BodyData	The recipe body contents.	Varies.
BodyDesc	Body descriptor section.	Structure composed of BodyDescName, BodyDescLength, BodyDescTimestamp.
BodyDescLength	Value of BodyLength *attribute*.	Unsigned integer.
BodyDescName	Identifies start of *body descriptor*.	Text = "BodyDesc".
BodyDescTimestamp	Value of EditTime *attribute*.	Timestamp format: "YYYYMMDDHHMMSSCC".
BodySection	The recipe *body* section.	Structure composed of BodySectionName, BodyData.
BodySectionName	Value of BodyName attribute.	Text = "Body".
ErrorCode	Contains the code for the specific error found.	Enumerated: unknown object type in specifier unknown object identifier in specifier unknown attribute name invalid attribute value access denied syntax error recipe not found verification error validation error agent not found
ErrorText	Text in support of the error code to provide additional information.	Text.
GenAttrSection	The *generic* attribute *section*.	Structure composed of GenAttrSectionName and (List of) AttrSettings for *generic* attributes.
GenAttrSectionName	Value of Gen_AttrName *attribute*.	Text ="Generic".
GenDesc	*Generic* attribute descriptor.	Structure composed of GenDescName, GenDescLength, and GenDescTimeStamp.
GenDescLength	Value of AttrLength *attribute*.	Unsigned integer.
GenDescName	Identifies start of *generic section*.	Text ="GenDesc".
GenDescTimestamp	Value of AttrChgTime attribute.	Timestamp format: "YYYYMMDDHHMMSSCC".
LinkID	In response or completion message, has the value of RMOpID in the request. Otherwise, set to zero if, and only if, no further completion messages will be sent.	Unsigned integer.
NSAck	Acknowledge code.	Enumerated: requested operation completed without errors

		requested operation completed with errors requested operation will be performed and notifications of results will be sent
NSStatus	Information concerning the outcome of the requested operations.	Structure composed of NSAck and Status.
RcpClass	Recipe *class*.	Formatted text.
RcpName	Recipe name.	Text.
RcpVersion	Recipe version.	Text.
RMAction	Identifies action performed on recipe.	Enumerated: *delete* *protect* *verify* *link* *unlink* *certify* *de-certify* *download* *upload*
RMAgent	Agent name (identifier).	Text.
RMChangeStatus	Indicates new status of recipe.	Enumerated: *no change* *created (new identifier)* *updated (modified)* *deleted* *copied (new identifier)* *renamed (new identifier)* *verified* *linked* *unlinked* *certified* *de-certified*
RMDestRcpID	Identifier of destination recipe.	Formatted text.
RMDesc	Recipe descriptor (timestamp and length attributes).	Structure composed of GenAttrDesc, BodyDesc, and (list of) AgentSpec_Desc
RMNewNS	New *namespace* name.	Text.
RMSpace	Amount of space available in *namespace*.	Unsigned integer.
RMNSSpec	Target *namespace specifier*.	Formatted text for object specifier.
RMOpID	Operation ID assigned by the service user in a request.	Unsigned integer.
RMParam	Parameter definition.	Structure composed of RMParamName, RMParamSetting, and RMParamRule.
RMParamName	Parameter name.	Text.
RMParamRule	Parameter definition restriction.	Formatted text.
RMParamSetting	The initial value to use for that parameter at execution time.	Varies with parameter definition and recipe language.
RMRcpID	Recipe identifier.	Formatted text.
RMRcpSpec	Target *recipe specifier*.	Formatted text for object specifier.
RMRcpStat	The status of a recipe.	Enumerated: Does not exist Unprotected Protected
RMSectionsCode	Indicates which recipe sections are requested.	Enumerated: All *generic attributes* and the body *Generic* attributes *Generic* and body *Agent-specific dataset* only All *agent-specific datasets*
RMSectionsRequested	Specification of recipe sections requested.	Structure composed of RMSectionsCode and

		RMAgent.
RMSectionsSent	Recipe attributes, recipe body, and attributes of *agent-specific* section(s).	Structure composed of GenAttrSection, BodySection, and (List of) AgentSpec_Section.
Status	Error Information.	Structure composed of ErrorCode and ErrorText.

12.2 *Message Flow* — The flow of messages defined in RMS, between the service user and the service provider, are of two types. The first type, illustrated in Figure 12.1, applies to most namespace operations. A request message is sent from the service user, and the *namespace manager* returns a response message containing the appropriate information.

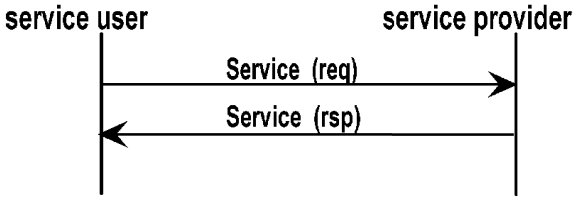

Figure 12.1

Basic Message Flow

The second type of message flow, illustrated in Figure 12.2, is used for messages that specify an operation to be performed that may require more time for completion than is allowed between a request and a response. In this case, the response to the request indicates an intent to complete the operation rather than actual completion. The operation is then performed on each of the specified recipes in turn, until each operation has completed, either successfully or with errors. Any errors that occurred are reported in the completion message.

The second type of message flow may be used for notifying the service user of RMNAction that individual operations on recipes have been completed. The first type of message flow is used for all other *namespace* services.

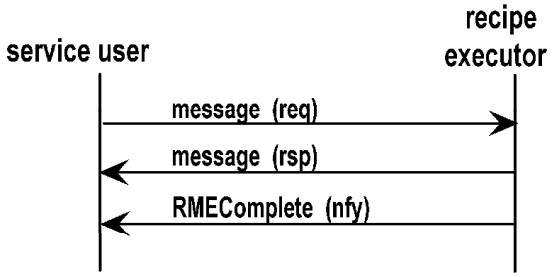

Figure 12.2

Message Flow with Completion Notification

The second type of message flow may be used for notifying the service user of RMNAction that individual operations on recipes have been completed. The first type of message flow is used for all other namespace services.

12.3 *RMNCreateNS* — The service user may request to *create* a new *namespace* and assign it a name (*identifier*). The request shall be denied if the *owner agent* already has a *namespace* with the new name.

The parameters for RMNCreateNS are listed in Table 12.3.

Table 12.3 RMNCreateNS

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMNSSpec	M	-	Target *namespace specifier*.
NSStatus	-	M	Information concerning the result of the requested operation.

12.4 *RMNDeleteNS* — The service user may request to delete a *namespace*. If the *namespace* contains recipes, the request shall be denied.

The parameters for RMNDeleteNS are listed in Table 12.4.

Table 12.4 RMNDeleteNS

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMNSSpec	M	-	Target *namespace specifier*.
NSStatus	-	M	Information concerning the result of the requested operation.

12.5 *RMNRenameNS* — The service user may request to rename a *namespace*. The request shall be denied if the owner agent already has a *namespace* with the new name.

The parameters for RMNRenameNS are listed in Table 12.5.

Table 12.5 RMNRenameNS Service

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMNSSpec	M	-	Target *namespace specifier*.
RMNewNS	M	M(=)	New *namespace* name *(identifier)*.
NSStatus	-	M	Information concerning the result of the requested operation.

12.6 *RMNSpaceInquire* — The service user may request the amount of space available for recipes.

The parameters for RMNSpaceInquire are listed in Table 12.6.

Table 12.6 RMNSpaceInquire

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMNSSpec	M	-	Target *namespace specifier*.
RMSpace	-	M	Amount of space available in target *namespace*.
NSStatus	-	M	Information concerning the result of the requested operation.

12.7 *RMNRcpStatInquire* — The service user may request the current write-protection status of a recipe and the next available numeric version for a recipe of the same class and name.

The parameters for RMNRcpStatInquire are listed in Table 12.7.

Table 12.7 RMNRcpStatInquire

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMNSSpec	M	-	Target *namespace specifier*.
RMRcpStat	-	M	Status of recipe.
RcpVersion	-	C	Next available numeric version. Required, except with the specified recipe does not exist.
NSStatus	-	M	Information concerning the result of the requested operation.

12.8 *RMNVersionInquire* — The service user may request the best version (highest) for a recipe with given qualifiers of *class*, *name*, and (optionally) *agent*.

The parameters for RMNVersionInquire are listed in Table 12.8.

Table 12.8 RMNVersionInquire

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMNSSpec	M	-	Target *namespace specifier*.
RcpClass	M	-	Qualifying recipe *class*.
RcpName	M	-	Qualifying recipe *name*.
RMAgent	C	C(=)	Certified agent. If omitted, agent qualification is not considered. If included, only recipes for which the *agent* is *certified* qualify.
RcpVersion	-	C	The best available *version* of the recipe. Omitted if no qualifying recipe is found.
NSStatus	-	M	Information concerning the result of the requested operation.

12.9 *RMNCreate* — A service user may request to initialize a new recipe in a namespace with a body created elsewhere. The requestor is required to supply the *attributes* of the recipe *body*. If the requested recipe *identifier* is used by a *read-only* recipe, the request shall be denied.

The parameters for RMNCreate are listed in Table 12.9.

12.10 *RMNUpdate* — The service user requests to replace an existing recipe *body* in a namespace with an

updated body. If the requested recipe *identifier* is used by a *read-only* recipe, the request shall be denied.

The parameters for RMNUpdate are listed in Table 12.9.

Table 12.9 RMNCreate and RMNUpdate

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMRcpSpec	M	-	Recipe specifier.
(List of) AttrSetting	M	-	List of recipe attributes required for create/update.
BodySection	M	-	Recipe body.
NSStatus	-	M	Information concerning the result of the requested operation.

12.11 *RMNStore* — The service user may request to store a recipe in a *namespace*. If the recipe *identifier* is already in use by a *read-only* recipe in the *namespace,* or if the *namespace* has insufficient space for storing the recipe, request shall be denied.

The parameters for RMNRetrieve are listed in Table 12.10.

Table 12.10 RMNStore

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMRcpSpec	M	-	The *recipe specifier.*
RMSectionsCode	M	-	Indicates which sections sent.
RMSectionsSent	M	-	Contents of all sections to be sent.
NSStatus	-	M	Information concerning the result of the requested operation.

12.12 *RMNRetrieve* — The service user may request to receive a recipe in its entirety (including all *agent-specific datasets*), the *generic* attributes with or without the *body* but without *agent-specific* datasets, a particular *agent-specific dataset*, or all *agent-specific datasets*.

The parameters for RMNRetrieve are listed in Table 12.11.

Table 12.11 RMNRetrieve

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMRcpSpec	M	-	The *recipe specifier.*
RMSectionsRequested	M	-	Indicates which sections are requested.
RMSectionsSent	-	M	Contents of all sections to be sent.
NSStatus	-	M	Information concerning the result of the requested operation.

12.13 *RMNCopy* — The service user may request to copy a recipe in a *namespace* to another recipe within the same *namespace*. If a *read-only* recipe with the specified Destination ID exists, an Access Denied error shall be returned.

The parameters for RMNCopy are listed in Table 12.12.

12.14 *RMNRename* — The service user may request to rename a recipe in a *namespace* if the recipe is not *write-protected.* If the recipe is *write-protected*, then an Access Denied error shall be returned.

The parameters for RMNRename are listed in Table 12.12.

Table 12.12 RMNCopy, RMNRename

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMRcpSpec	M	-	*Specifier* of the recipe to be copied or renamed (source).
RMDestRcpID	M	-	The identifier for the recipe to be originated (destination).
NSStatus	-	M	Information concerning the result of the requested operation.

12.15 *RMNAction* — RMNAction is used to request the performance of one of the following actions (operations) on one or more recipes: delete, protect, verify, link, unlink, certify, de-certify, download to a specified *agent*, and upload from a specified *agent*. Where more than one recipe is specified, operations shall be performed on each recipe in the requested order.

If performance of the requested action is likely to take more time to complete than is normally allowed between a request message and the corresponding response, then the *namespace manager* should respond with an intent to comply at the earliest opportunity before it begins the requested operation. Immediately following the completion of the operation on each of the specified recipes, the *manager* shall send a notification message RMNComplete to the service requestor, providing the results of that operation together with information concerning any errors causing abnormal completion of the operation. Error information is important for diagnosis and correction of problems. Figure 12.2 in Section 12.2 shows an example of the message flow for the download operation performed on two recipes.

The parameter LinkID is set to a zero value in the response if the operations have been completed and no notifications will be sent. Otherwise, it is set to a non-

zero value, and the value set in the parameter RMOpID shall be retained for use in RMNComplete.

The parameters for RMNAction are listed in Table 12.14.

Table 12.14 RMNAction

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMNSSpec	M	-	Target *namespace specifier*.
RMAction	M	M(=)	Action.
RMOpID	C	C(=)	Operation ID, used where multiple confirmations may occur. May be null or omitted where the service initiator does not request a new service until all confirmations for the outstanding service have been received.
(List of) RMRcpID	M	-	Identifier of the target recipe.
RMAgent	C	C(=)	Omitted, except for certify, de-certify, download, upload.
RMLinkID	-	M	LinkID is set to zero if, and only if, no further completion messages will be required.
NSStatus	-	M	Information concerning the result of the requested operation.

12.16 *RMNVarPar* — The service user may request to set *agent-specific variable parameter definitions*. NOTE: Modification of the *parameter restrictions* in a *variable parameter definition* shall reset the Certified *attribute* for that *agent*.

The parameters for RMNVarPar are listed in Table 12.15.

Table 12.15 RMNVarPar

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMRcpSpec	M	-	*Specifier* of the target recipe.
RMAgent	M	-	Agent name (identifier).
(List of) RMParam	M	-	Variable parameter definition.
NSStatus	-	M	Information concerning the result of the requested operation.

12.17 *RMNGetDescriptor* — The service user may request the descriptors of one or more recipes.

The parameters for RMNGetDescriptor are listed in Table 12.13.

Table 12.13 RMNGetDescriptor

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMNSSpec	M	-	Target *namespace specifier*.
(List of) RMRcpID	M	-	Identifier of the target recipe.
(List of) RMDesc	-	M	Requested *recipe descriptor(s)* in the same order as requested.
NSStatus	-	M	Information concerning the result of the requested operation.

12.18 *RMNComplete* — The service provider notifies the service user that an action requested by the service user has completed.

The parameter RMOpID is retained from the initial request message RMNAction.

The parameter LinkID is set to a non-zero value for all notifications except the notification of completion of the last operation performed. A zero value indicates that no further notifications will be sent.

The parameters for RMNComplete are listed in Table 12.16.

Table 12.16 RMNComplete

Parameter	*Req/ Ind*	*Rsp/ Conf*	*Description*
RMOpID	C(=)	-	Operation ID, used where multiple completion messages may occur, corresponding to the value in the original request. May be null or omitted where the service initiator does not request a new service until all confirmations for the outstanding service have been received.
RMLinkID	M	-	LinkID is set to zero if, and only if, no further completion messages will be required.
RMRcpSpec	-	C	*Specifier* of the changed recipe. Indicates *namespace* and recipe *identifiers*.
RMChange Status	M	-	State of the changed recipe.
(List of) AttrSettings	C	-	Attributes of particular interest to the result. Varies with action.
NSStatus	M	-	Information conerning the result of the requested operation.

13 Distributed Recipe Namespace Services

This section defines the new message services required for implementation of *distributed recipe namespace* services. Table 13.1 lists the messages, the type of each message (request or notification), and provides a brief description.

Table 13.1 Distributed Recipe Namespace Services

Message Service Name	*Type*	*Description*
DRNS Segment Services		
Attach	R	A *manager* requests *Distributed Recipe Namespace Segment* to attach to the *manager*. (OSS service)
Create	R	An *authorized user* requests to create *Distributed Recipe Namespace Segment*. (OSS service)
Delete	R	An *authorized user* requests to delete *Distributed Recipe Namespace Segement*. (OSS service)
Detach	R	A *manager* requests *Distributed Recipe Namespace Segment* to detach from the *manager*. (OSS service)
RMDSApproveAction	R	A *segment's manager* approves an earlier *change request*.
DRNS Recorder Services		
Attach	R	A *manager* requests *Recorder* to attach the *manager*. (OSS service)
AttachSetAttr	R	A *manager* requests *Recorder* to set one or more attributes whenever it is attached to the *manager*. (OSS service)
Create	R	An *authorized user* requests to create *Recorder*. (OSS service)
Delete	R	An *authorized user* requests to delete *Recorder*. (OSS service)
Detach	R	A *manager* requests *Recorder* to detach from the *manager*. (OSS service)
Reattach	R	A *manager* requests *Recorder* to reattach to the *manager*. (OSS service)
RMDRAddSegRecord	R	A *manager* requests a *recorder* to add a *segment specifier* to its Segments attribute.
RMDRDelSegRecord	R	A *manager* requests a *recorder* to delete a *segment specifier* from its Segments attribute.
RMDRAddChgRecord	R	A *manager* requests a *recorder* to add a *change request record* for a specified recipe.
RMDRDelChgRecord	R	A *manager* requests a *recorder* to delete a *change request record* for a specified recipe.
RMDRGetChgRecord	R	A *service user* requests *change request* for a recipe or segment.
DRNS Manager Services		
AttachSupervisedObject	R	An *authorized user* requests to attach *Distributed Recipe Namespace Segment or Recorder*. (OSS service)
DetachSupervisedObject	R	An *authorized user* requests to detach *Distributed Recipe Namespace Segment or Recorder*. (OSS service)
RMDComplete	N	An attached *segment* notifies its *manager* that an approved change has been completed.
RMDNotify	N	An attached *dedicated segment* notifies its *manager* that an *agent-specific dataset* has been changed.
RMDGetChgRequest	R	A request is made for *change request* information for a recipe.
RMDSegChange	R	An attached *segment* requests approval for a specified type of change to a recipe.
RMDRebuild	R	A request is made for a new *manager* to rebuild an existing *distributed namespace*.

13.1 *Distributed Recipe Namespace Message Parameter Dictionary* — This section defines, in alphabetical order, the parameters that are used in *distributed recipe namespace* services and are not already defined in Table 12.2 in Section 12.

Table 13.2 contains the definitions for message parameters.

Table 13.2 Distribute Recipe Namespace Parameter Dictionary

Parameter	*Definition*	*Form*
AttrData	Value of attribute.	Varies with attribute.
AttrName	Name of attribute.	Text.
AttrSetting	Attribute name/value.	Structure composed of AttrName, AttrData.
Attach Token	Attach token provided by an object at the time it is attached or reattached and used by the manager as identification.	Unsigned integer.
ChgRequest	The information provided in a change request.	Structure composed of: RMRcpID, DestRcpID, RMSegSpec, RMChgType, RMOpID, RMTimestamp, RMRequestor
NSAck	Acknowledge code.	Enumerated: no request for this action exists requested operation completed without errors requested operation completed with errors requested operation will be performed and notification of results will be sent
NSStatus	Information concerning the outcome of the requested operation.	Structure composed of NSAck and Status.
ObjID	Object identifier.	Text.
ObjSpec	Object specifier.	Formatted text for object specifier.
ObjType	Object type.	Text.
RMChgType	Type of change requested.	Enumerated: create update delete copy rename verify link unlink certify de-certify change generic attributes(s) change agent-specific attribute(s) store
RMManagerSpec	Object specifier for a *DRNS manager*.	Formatted text.
RMOpID	Operation ID assigned by the *distributed recipe namespace manager*.	Unsigned integer.
RMPermit	Approval for a requested change.	Enumerated: change request approved change request denied change request on hold
RMRcpID	Recipe identifier.	Formatted text.
RMRecorderSpec	Recorder specifier.	Formatted text for object specifier.
RMRequestor	Indicates if the initiator of a change request was an *attached segment* (TRUE) or an external user (FALSE).	Boolean.

RMSegSpec	Segment specifier.	Formatted text for object specifier.
RMTimestamp	Date and time a *change request* was first received.	Timestamp format: "YYYYMDDhhmmsscc".
TargetSpec	Object specifier of target object.	Formatted text for object specifier.

13.2 *Distributed Recipe Namespace Segment Services*

13.2.1 *RMDSApproveAction* — A *distributed recipe namespace manager* may approve or deny a *change request* that the *segment* requested at an earlier time.

Parameters for RMDSApproveAction are listed in Table 13.3.

Table 13.3 RMDSApproveAction Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RMSegSpec	M	-	The segment specifier.
Attach Token	M	-	Segment attach token.
RMPermit	M	-	Approval is granted or denied by the *manager*.
RMOpID	M	-	An *operation identifier* assigned by the *manager* for the change.
RMRcpID	M	-	The *identifier* of the recipe for which a change was requested.
RMChgType	M	-	The type of change requested.
NSStatus	-	M	Information concerning the result of the requested operation.

13.3 *Distributed Recipe Namespace Recorder Services*

13.3.1 *RMDRAddSegRecord* — A *recorder's manager* may request the *recorder* to add a *segment record*.

Parameters for RMDRAddSegRecord are listed in Table 13.4.

Table 13.4 RMDRAddSegRecord Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RMRecorderSpec	M	-	The *recorder specifier*.
RMSegSpec	M	-	The *specifier* of the *segment* to add.
Attach Token	M	-	The recorder's attach token.
NSStatus	-	M	Information concerning the result of the requested operation.

13.3.2 *RMDRDelSegRecord* — A *recorder's manager* may request the *recorder* to delete a *segment record*.

Parameters for RMDRDelSegRecord are listed in Table 13.5.

Table 13.5 RMDRDelSegRecord Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RMRecorderSpec	M	-	The *recorder specifier*.
RMSegSpec	M	-	The *specifier* of the *segment* to add.
Attach Token	M	-	The recorder's attach token.
NSStatus	-	M	Information concerning the result of the requested operation.

13.3.3 *RMDRAddChgRecord* — A *recorder's manager* may request the *recorder* to add a *change request record*.

Parameters for RMDRAddChgRecord are listed in Table 13.6.

Table 13.6 RMDRAddChgRecord Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RMRecorderSpec	M	-	The *recorder specifier*.
ChgRequest	M	-	Change request record.
Attach Token	M	-	The recorder's attach token.
NSStatus	-	M	Information concerning the result of the requested operation.

13.3.4 *RMDRDelChgRecord* — A *recorder's manager* may request the *recorder* to delete a *change request record*.

Parameters for RMDRDelChgRecord are listed in Table 13.7.

13.7 RMDRDelChgRecord Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RMRecorderSpec	M	-	The *recorder specifier*.
RMRcpID	M	-	The *identifier* of the recipe to be changed.
Attach Token	M	-	The recorder's attach token.
NSStatus	-	M	Information concerning the result of the requested operation.

13.3.5 *RMDRGetChgRecord* — A service user may request an attached *recorder* to return the current *change request record(s)* for a recipe or an attached *DRNS segment*.

Parameters for RMDRGetChgRecord are listed in Table 13.8.

Table 13.8 RMDRGetChgRecord Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RMRecorderSpec	M	-	Recorder specifier.
TargetSpec	M	-	Recipe identifier or segment specifier.
Attach Token	M	-	The recorder's attach token.
(list of) ChangeRequest	-	C	List of change request information. Omitted if no change requests exist for specified recipe/segment.
NSStatus	-	M	Information concerning the result of the requested operation.

13.4 *Distributed Recipe Namespace Manager Services*

13.4.1 *RMDComplete* — An attached *segment* notifies its *manager* when it has completed an approved change to a recipe.

The parameter RMOpID is retained from the initial request message RMNAction.

The parameter LinkID is set to a non-zero value for all notifications except the notification of completion of the last operation performed. A zero value indicates that no further notifications will be sent.

Parameters for RMDComplete are listed in Table 13.9.

Table 13.9 RMDComplete

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RMOpID	C(=)	-	Operation ID, used where multiple completion messages may occur, corresponding to the value in the original request. May be null or omitted where the service initiator does not request a new service until all confirmations for the outstanding service have been received.
RMLinkID	M	-	LinkID is set to zero if, and only if, no further completion messages will be required.
RMRcpSpec	M	-	*Specifier* of the changed recipe. Indicates *namespace* and recipe *identifiers*.
RMChangeStatus	M	-	State of the changed recipe.
(List of) AttrSettings	C	-	Attributes of particular interest to the result. Varies with action.
NSStatus	-	C	Information concerning the result of the requested operation.

13.4.2 *RMDNotify* — An attached *dedicated segment* notifies its *manager* that it has changed an *agent-specific dataset* and provides the attribute AgentSpec_Agent and all updated attributes.

Parameters for RMDNotify are listed in Table 13.10.

Table 13.10 RMDNotify

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RMRcpSpec	M	-	*Specifier* of the changed recipe. Indicates *namespace* and recipe *identifiers*.
RMChangeStatus	M	-	State of the changed recipe.
(List of) AttrSettings	C	-	Changed *agent-specific attributes*.
NSStatus	-	C	Information concerning the result of the requested operation.

13.4.3 *RMDSegChange* — An *attached segment* may request its *manager*'s approval to make a change. The recipe specifier RMRcpSpec shall include the requesting *segment's* object type and name to identify the requestor and the specific instance of the logical recipe to be changed first.

Parameters for RMDSegChange are listed in Table 13.11.

Table 13.11 RMDSegChangeService

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RMRcpSpec	M		Recipe specifier.
RMDestRcpID	C		Destination recipe identifier (copy/rename only).
RMChgType	M	M	Type of change requested.
RMPermit		M	Grants or denies permission.
RMOpID		M	A non-zero value returned when the change is completed.

13.4.4 *RMDGetChangeRequests* — A service user may request that all existing *change requests* for a recipe be returned. The first item returned is the *active change request*.

Parameters for RMDGetChangeRequests are listed in Table 13.12.

Table 13.12 RMDGetChangeRequests Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RMNSSpec	M	-	The object specifier of the *distributed recipe namespace*.
RMRcdID	M	-	The *recipe specifier*.
(list of) ChangeRequest	C	-	Omitted if no *change requests* exist for the specified recipe.
NSStatus	-	M	Information concerning the result of the requested operation.

13.4.5 *RMDRebuild* — A service user may request a *distributed namespace manager* to rebuild a *distributed namespace*.

Parameters for RMDRebuild are listed in Table 13.13.

Table 13.13 RMDRebuild Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RMManagerSpec	M	-	The object specifier of the new *distributed namespace manager*.
RMNSSpec	M	-	The specifier of the *distributed namespace*.
RMRecorderSpec	C	-	The *recorder specifier*. Omitted only if list of segments provided.
(list of) RMSegSpec	C	-	List of specifiers of attached *segments*. Omitted if *recorder specifier* provided.
NSStatus	-	M	Information concerning the result of the requested operation.

14 Recipe Executor Services

This section defines *recipe executor services*. *Recipe executor services* request that an operation be performed by the *recipe executor* and begin with the prefix RME.

Certain activities that may require extended time to complete shall provide confirmation with the RMEComplete message at the completion of each individual operation specified.

Table 14.1 Recipe Executor Services

Message Service Name	*Type*	*Description*
RMEDnldVer	R	Receive a recipe, verify it, and put it into storage.
RMEVerify	R	Verify one or more recipes.
RMEUpload	R	Retrieve a recipe from storage.
RMERename	R	Change a recipe's *identifier.*
RMESpaceInquire	R	Determine the available space for recipe storage.
RMEDelete	R	Delete a recipe from storage.
RMESelect	R	*Select* recipes for execution.
RMEDeselect	R	*Deselect* a recipe to prevent its execution.
RMEGetDescriptor	R	Get recipe descriptor.
RMEChange	N	Notification that a permitted change has occurred.
RMEComplete	N	Notification that a requested action is complete.

14.1 *Recipe Executor Message Parameter Dictionary* — For purposes of transferring a recipe, it is regarded as having two types of *sections:* an *attribute section* containing the recipe's attributes, and a **body section** consisting of the *body*. The *attribute section* is always transferred first, and the attributes ExecAttrLength, ExecAttrChgTime, AttrLength, AttrTimestamp, BodyLength, and EditTime are always sent first, in that order. This allows the length information to be immediately determined by the receiver.

Descriptors are also regarded as consisting of *sections* for the *attribute descriptor*, the *generic attribute descriptor*, and the *body descriptor*.

Each start of each *section* is identified with a text string reserved for that *section type*.

Table 14.2 contains the definitions for message parameters.

Table 14.2 Recipe Executor Parameter Dictionary

Parameter	*Definition*	*Form*
AttrName	Name of attribute.	Text.
AttrData	Value of attribute.	Varies with attitude.
AttrSetting	Attribute name/value pair.	Structure composed of AttrName, AttrData.
BodyData	The recipe *body* contents.	Varies.
BodyDesc	*Body descriptor.*	Structure composed of BodyDescName, BodyDescLength, BodyDescTimestamp.
BodyDescLength	Value of BodyLength.	Unsigned integer.
BodyDescName	Identifies the start of the *body descriptor.*	Text ="BodyDesc".
BodyDescTimestamp	Value of EditTime.	Timestamp format: "YYYYMMDDhhmmsscc".
BodySection	The recipe *body section.*	Structure composed of BodySectionName, BodyData.
BodySectionName	Identifies the start of the *body section.*	Text ="Body".
ErrorCode	Contains the code for the specific error found.	Enumerated: unknown object type unknown object identifier unknown attribute name unknown attribute value access denied recipe not found syntax error validation error verification error recipe select error: unlinked recipe recipe approval level too low recipe certification level too low
ErrorText	Text in support of the error code to provide additional information.	Text.
GenDesc	*Generic attribute* descriptor.	Structure composed of GenDescName, GenDescLength, and GenDescTimestamp.
GenDescLength	Value of AttrLength.	Unsigned integer.
GenDescName	Identifies start of *generic descriptor.*	Text ="GenDesc".
GenDescTimestamp	Value of AttrChgTime.	Timestamp format: "YYYYMMDDhhmmsscc".
LinkID	In response or completion message, has the value of REOplD in the request, or is set to zero if, and only if, no further completion messages will be sent.	Unsigned integer.
RcpDerivedID	Identifier of derived object form recipe.	Structure composed of text "DerivedID" and REDerivedID.
RcpEstRunTime	EstRunTime attribute.	Structure composed of text "EstRunTime" and REEstRunTime.
RcpExtRef	ExtRef attribute of *managed* recipe.	Structure composed of text "ExtRef" and (list of) REExtRef.

Parameter	*Definition*	*Form*
RcpParameters	Parameters attribute of *managed* recipe.	Structure composed of text "Parameters" and (list of) REParam.
RcpVerificationID	VerificationID attribute of a *verified* recipe.	Structure composed of text "VerificationID" and REVerID.
REAck	Acknowledge code.	Enumerated: unknown object type unknown object identifier unknown attribute name invalid attribute value access denied recipe not found invalid parameter settings
REAttrDesc	The descriptor of the *execution* recipe attributes.	Structure composed of REAttrDescName, REAttrDescLength, REAttrDescTimestamp.
REAttrDescLength	The value of ExecAttrLength.	Unsigned integer.
REAttrDescName	Identifies the start of the *attribute descriptor*.	Text ="REAttrDesc".
REAttrDescTimestamp	Value of ExecAttrChgTime.	Timestamp format: "YYYYMMDDmmhhsscc".
REAttrSection	Recipe attributes section.	Structure composed of REAttrSectionName and REAttrSectionData.
REAttrSectionData	Recipe attributes name/value pairs.	(List of) AttrSetting.
REAttrSectionName	Identifies recipe attribute section.	Text ="Attributes".
REChangeStatus	Indicates new status of recipe.	Enumerated: stored deleted selected de-selected created changed body updated last value derived object form
REDerivedID	The *identifier* of a *derived form recipe.*	Formatted text.
REDesc	*Descriptor* of the *execution* recipe.	Structure composed of REAttrDesc, GenDesc, and BodyDesc.
REEstRunTime	Estimated run time of recipe, in seconds. Value of EstRunTime attribute.	Unsigned integer.
REExRef	A reference to an external recipe, in the form of a *recipe specifier.*	Formatted text.
REDestRcpID	The new *identifier* of a renamed recipe.	Formatted text.
REOpID	Operation ID assigned by the service user in a request.	Unsigned integer.
REOWCode	Indicates a forced overwrite of any pre-existing recipe of the same *identifier.*	Boolean: If FALSE or omitted, a pre-existing recipe with the same identifier is not overwritten and will cause a download error. If TRUE, any pre-existing recipe with the same *identifier* is effectively overwritten.
REParam	Parameter definition.	Structure composed of REParamName, REParamSetting, and REParamRule.
REParamName	Parameter name.	Text.
REParamSetting	The *initial value* to use for a parameter at execution time.	Varies with parameter definition and recipe language.
RERcpID	Recipe *identifier*.	Formatted text.
RERcpSpec	Recipe specifier[a].	Formatted text.

Parameter	*Definition*	*Form*
RERecipe	*Execution* recipe.	Structure composed of REAttrSection, BodySection.
RESelect	Recipe specifier with option parameter setting.	(List of) Structure composed of RERcpSpec and (list of) RESelectParam.
RESelectParam	*Variable parameter name* and *initial value*.	Structure composed of REParamName and REParamSetting.
RESpace	Amount of available storage, in bytes.	Unsigned integer.
RESpec	The specifier of the *recipe executor*.	Formatted text.
REStatus	Information regarding the success or failure of the request.	Structure composed of REAck and (List of) Status.
REVerData	Information concerning a successfully *verified* recipe.	Structure composed of RcpExtRef, RcpParameters, RcpEstRunTime (optional), RcpVerificationID (optional), and (where applicable) RcpDerivedID.
REVerID	Contents of VerificationID attribute.	Text.
Status	Error information.	Structure composed of Error Code and ErrorText.

a. A recipe specifier may reference a recipe that is in a namespace and not in the recipe executor's storage.

14.2 *Message Flow* — The flow of messages defined in RMS, between the service user and the service provider, are of three types. The first type, illustrated in Figure 14.1, is the most common and applies to all *recipe executor* operations except for cases of RMEDnldVer, RMEVerify, and RMESelect. A request message is sent from the service user, and the *recipe executor* returns a response message containing the appropriate information. All activities that occur as a result of the request are completed before the response message is sent.

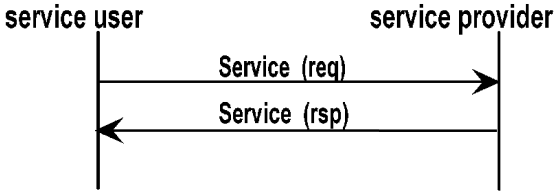

Figure 14.1

Basic Message Flow

The second type of message flow, illustrated in Figure 14.2, is used for messages that specify an operation to be performed that may require more time for completion than is allowed between a request and a response. In this case, the response to the request indicates an intent to complete the operation rather than actual completion. The operation is then performed on each of the specified recipes in turn, until each operation has completed, either successfully or with errors. Any errors that occurred are reported in the completion message.

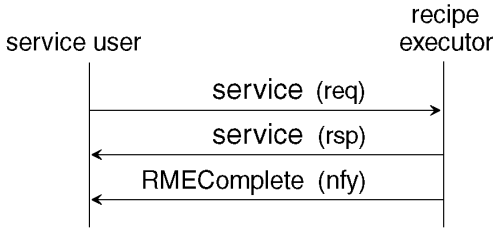

Figure 14.2

Message Flow with Completion Notification

The second type of message flow may be used for notifying the service user of RMEDnldVer, RMEVerify, and RMESelect that individual operations on recipes have been completed.

A third type of message flow consists of a single notification message, shown in Figure 11.3, sent by the *recipe executor* to inform the *originating namespace* of a new or changed *execution* recipe. (See Section 11.2.10.)

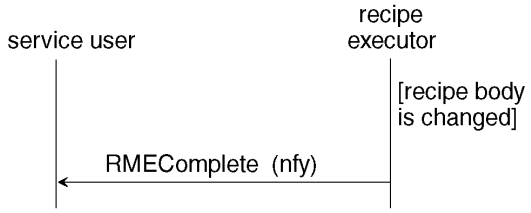

Figure 14.3

Change Notification Message Flow

14.3 *RMEDnldVer* — The service user may request the *recipe executor* to download and *verify* an *execution* recipe. All *unverified* downloaded recipes are *verified* automatically. The service user may optionally specify a *forced overwrite* of any pre-existing *unselected* recipe with the same *identifier.* A currently *selected* recipe shall not be overwritten. If forced *overwrite* is not specified and a recipe with the same *identifier* is already in storage, the request to *download* is denied and the recipe is discarded.

If the *recipe executor* performs a *verification* of the *downloaded* recipe, and if the *verification* is successful, then it shall return the information provided by the conditional parameters.

Parameters for RMEDnldVer are listed in Table 14.3.

Table 14.3 RMEDnldVer Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RERcpSpec	M	-	Destination *specifier* of the *execution* recipe.
REOWCode	M	-	Indicates if a forced overwrite is requested.
RERecipe	M	-	The *execution* recipe.
REVerData	-	C	Information concerning the successfully *verified* recipe. Omitted if recipe has no external references or fails *verification.*
REStatus	-	M	Information concerning the results of the operation.

14.4 *RMEVerify* — The service user may request the *recipe executor* to *verify* an *execution* recipe. If the *verification* is successful, then it shall return the information provided by the conditional parameters.

Parameters for RMEVerify are listed in Table 14.4.

Table 14.4 RMEVerify Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RESpec	M	-	*Specifier* of the target recipe executor.
(List of) RERcpID	M	-	*Identifier* of the recipe.
REOpID	C	C(=)	Operation ID, used where multiple confirmations may occur. May be null or omitted where the service initiator does not request a new service, until all confirmations for the outstanding service have been received.
LinkID	-	M	LinkID is set to zero if, and only if, no further completion messages will be required.
(List of) REVerData	-	C	Information concerning the successfully *verified* recipe. Omitted if recipe has no external references or fails *verification.*
REStatus	-	M	Information concerning the results of the operation.

14.5 *RMEUpload* — RMEUpload retrieves a recipe from local storage and sends it to the service user.

Parameters for RMEUpload are listed in Table 14.5.

Table 14.5 RMEUpload Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RERcpSpec	M	M(=)	Specifier.
RERecipe	-	M	The *execution* recipe.
REStatus	-	M	Information concerning the results of the operation.

14.6 *RMERename* — RMERename changes the *identifier* of an *execution recipe* through changing one or more of its *identification* attributes. If the new *identifier* is already in use by an *execution recipe*, or if the recipe is currently *selected*, permission to rename the recipe shall be denied.

Parameters for RMERename are listed in Table 14.6.

Table 14.6 RMERename Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RERcpSpec	M	-	*Specifier* of the *execution* recipe.
REDestRcpID	M	M(=)	The new recipe *identifier.*
REStatus	-	M	Information concerning the results of the operation.

14.7 *RMESpaceInquire* — The service user may request the amount of space available for recipes. The parameters for RMESpaceInquire are listed in Table 14.7.

Table 14.7 RMESpaceInquire

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RESpec	M	-	Target *recipe executor specifier.*
RESpace	-	M	Amount of space available in target *namespace.*
REStatus	-	M	Information concerning the results of the requested operation.

14.8 *RMEDelete* — RMEDelete removes one or more recipes from local storage and frees any reusable storage area for new recipes. If the recipe is currently *selected*, permission to delete the recipe shall be denied.

Parameters for RMEDelete are listed in Table 14.8.

Table 14.8 RMEDelete Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RESpec	M	-	*Specifier* of the recipe executor.
(List of) RERcpID	M	C	Identifier of the recipe to be deleted. Response returns identifiers of successfully deleted recipe(s) only.
REStatus	-	M	Information concerning the results of the operation.

14.9 *RMESelect* — The service user may specify one or more recipes to be *selected* for *execution*. RMESelect is the only service of the *recipe executor* that allows the service user to specify more than one recipe in a single request.

Parameters for RMESelect are listed in Table 14.9.

Table 14.9 RMESelect Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RESpec	M	-	*Specifier* of the recipe executor.
(List of) RESelect	M	-	Recipe *specifiers* with (optionally) one or more associated parameter names/values per recipe.
REOpID	C	C(=)	Operation ID, used where multiple confirmations may occur. May be null or omitted where the service initiator does not request a new service, until all confirmations for the outstanding service have been received.
LinkID	-	M	LinkID is set to zero if, and only if, no further completion messages will be required.
REStatus	-	M	Indicates the results of the operation.

14.10 *RMEDeselect* — The service user may deselect a recipe to prevent its *re-execution*.

Parameters for RMEDeselect are listed in Table 14.10.

Table 14.10 RMEDeselect Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RESpec	M	-	*Specifier* of the recipe executor.
(List of) RERcpID	M	-	Identifier of the recipe.
REStatus	-	M	Information concerning the results of the operation.

14.11 *RMEGetDescriptor* — The service user may request the descriptors of one or more recipes.

Parameters for RMEGetDescriptor are listed in Table 14.11.

Table 14.11 RMEGetDescriptor

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
RESpec	M	-	*Specifier* of the recipe executor.
(List of) RERcpID	M	-	Identifier of the recipe.
(List of) REDesc	-	M	Requested *recipe descriptor(s)* in the same order as requested.
REStatus	-	M	Information concerning the results of the operation.

14.12 *RMEChange* — The service provider informs an *originating namespace* that a permitted change has occurred.

Parameters for RMEChange are listed in Table 14.12.

Table 14.12 RMEChange Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
REOpID	C	-	Operation ID, used where multiple completion messages may occur, corresponding to the value in the original request. May be null or omitted where the service initiator does not request a new service using REOpID, until the confirmation for the outstanding service has been received.
RERcpID	M	-	*Identifier* of the recipe of interest. Equivalent to *specifier* of corresponding *managed recipe*.
REChangeStatus	M	-	State of the changed recipe.
(List of) AttrSettings	C	-	Attributes of particular interest to the result. Varies with action.
REStatus	M	-	Information concerning the results of the operation.

14.13 *RMEComplete* — The service provider notifies the service user that an action requested by the service user has completed.

Parameters for RMEComplete are listed in Table 14.13.

Table 14.13 RMEComplete Service

Parameter	*Req/Ind*	*Rsp/Conf*	*Description*
REOpID	C	-	Operation ID, used where multiple completion messages may occur, corresponding to the value in the original request. May be null or omitted where the service initiator does not request a new service using REOpID, until the confirmation for the outstanding service has been received.
RERcpID	M	-	*Identifier* of the recipe of interest. Equivalent to *specifier* of corresponding *managed recipe*.
REChangeStatus	M	-	State of the changed recipe.
(List of) AttrSettings	C	-	Attributes of particular interest to the result. Varies with action.
REStatus	M	-	Information concerning the results of the operation.

15 Recipe Management Compliance

This section gives a centralized reference location for compliance for RMS objects. It provides references to other sections of the standard where detailed requirements are located. This section also defines standard terminology and documentation related to RMS compliance that can be used by *agent* suppliers and device manufacturers to describe compliance with this standard.

Each object defined by RMS has both *fundamental requirements* and *additional capabilities*. *Fundamental requirements* shall be met by all instances of an RMS-compliant object as specified in cited sections. Additional capabilities consist of those features that are optional. However, for RMS compliance, where features providing the same or similar capabilities are implemented, they shall be implemented according to RMS specifications.

Except where one or more subsections are specifically excluded, section references include all subsections of the referenced section.

The following table format is used in this section to specify fundamental and additional (optional) capabilities. Fundamental capabilities have a "Y" in the column labeled "Fund." Otherwise, the capability is optional.

Capability	*Reference*	*Fundamental*
Name of capability	Sections in SEMI E42	Y or N

15.1 *Areas of Compliance* — There are seven areas of RMS compliance, as follows:

a. Managed Recipe

b. Recipe Namespace Management

c. Distributed Recipe Namespace Management

e. Distributed Recipe Namespace Recorder

d. Distributed Recipe Namespace Segment

f. Execution Recipe

g. Recipe Executor

15.2 *Managed Recipe* — This section defines RMS compliance for the *managed recipe*.

A *managed recipe* has attributes, a body, and is capable of supporting at least one agent-specific dataset. An RMS-compliant *managed recipe* reflects its compliance (a) through its attributes and (b) through its structure when it is transferred.

In some cases, a capability is represented by a single attribute. In other case, support of an additional capability requires several attributes that are required as a set.

Table 15.1 describes the requirements for compliance to the *managed recipe*.

Table 15.1 Section References for Managed Recipe Capabilities

Capability	*Reference*	*Fundamental*
Recipe identifier (class, name, version)	3.2.3	Y
/PROCESS/ recipe class	3.2.3.2	Y
Mandatory and required generic attributes	3.2.5, 3.4.1, 3.4.2	Y
Mandatory and required agent-specific dataset attributes	3.4.2.2	Y
Recipe State Model	8.3, Table 8.1	Y
/PROCESS/ recipe subclass(es)	3.2.3.2	N
Non-process recipe class(es)	3.2.3.2	N
Multi-part recipe (ExtRef, LinkList)	3.2.3.1, 3.4.2.1	N
Variable parameters (Parameters, LinkParam)	3.2.4.2, 3.4.2.1	N
Approval level (ApprovalLevel)	3.4.2.1, 3.4.2.1	N
Certification Certified	3.4.2.1	N
Source Form recipe body	3.2.2.1.1	N*
Object Form recipe body	3.2.2.1.2	N*

* At least one of the two forms is required.

15.3 *Managed Recipe Compliance Table* — Table 15.2 provides a statement of compliance for the *managed recipe*.

Table 15.2 Recipe Namespace Management Compliance Table

Manual Recipe Compliance Table		
Fundamental RMS Requirements	*Implemented*	*RMS Compliant*
Recipe identifier (class, name, version)	❒ Yes ❒ No	❒ Yes[1] ❒ No
/PROCESS/ recipe class	❒ Yes ❒ No	
Mandatory and required generic attributes	❒ Yes ❒ No	
Mandatory and required agent-specific dataset attributes	❒ Yes ❒ No	
Recipe State Model	❒ Yes ❒ No	
Additional Capabilities	*Implemented*	*RMS Compliant*[2]
/PROCESS/ recipe subclass(es)	❒ Yes ❒ No	❒ Yes ❒ No
Non-process recipe class(es)	❒ Yes ❒ No	❒ Yes ❒ No
Multi-part recipe (ExtRef, LinkList)	❒ Yes ❒ No	❒ Yes ❒ No
Variable parameters (Parameters, LinkParam)	❒ Yes ❒ No	❒ Yes ❒ No
Approval level (ApprovalLevel)	❒ Yes ❒ No	❒ Yes ❒ No
Agent-specific dataset mandatory and required attributes	❒ Yes ❒ No	❒ Yes ❒ No
Certification Certified	❒ Yes ❒ No	❒ Yes ❒ No
Source Form recipe body	❒ Yes ❒ No	❒ Yes ❒ No
Object Form recipe body	❒ Yes ❒ No	❒ Yes ❒ No

[1] Do not mark YES unless all fundamental RMS requirements are implemented.

[2] Additional capabilities may not be marked RMS compliant unless all fundamental RMS requirements are implemented.

15.4 *Recipe Namespace Management* — Recipe Namespace Management requires compliance to specifications for the *managed recipe*, the *recipe namespace*, and the *recipe namespace manager*.

15.4.1 *Recipe Namespace* — This section defines RMS compliance for the *recipe namespace* object. A *recipe namespace* exhibits RMS compliance through its attributes and access to managed recipes.

NOTE: Operations and services are performed by the *recipe manager*.

Table 15.3 describes the requirements for compliance to the *recipe namespace*.

Table 15.3 Section References for Recipe Namespace Capabilities

Capability	*Reference*	*Fundamental*
Fundamental compliance to *managed recipe*	15.2, Table 15.2	Y
Basic requirements	4.3, 4.4, 4.6, 8.2.1	Y

15.4.2 *Recipe Namespace Manager* — Table 15.4 describes RMS compliance for the *recipe namespace manager*.

Table 15.4 Section References for Recipe Namespace Manager Capabilities

Capability	*Reference*	*Fundamental*
Basic requirements	4.3, 4.6 (Table 4.2), 8.2.1	Y
OSS services	9.1, 9.1.1–9.1.2	Y
Create recipe (RMNCreate)	8.2.2, 8.2.2.1, 9.4.1, 12.9	Y
Update recipe (RMNUpdate)	8.2.2.2, 9.4.2, 12.10	Y
Verify recipe (RMNAction)	8.2.3.1, 9.4.7, 12.15	Y
Link recipe (RMNAction)	8.2.3.2, 12.15	Y
Get recipe descriptor (RMNGetDescriptor)	8.2.7.1, 12.17	Y
Get available space (RMNSpaceInquire)	9.3.1, 12.6	Y
Check recipe status (RMNRcpStatInquire)	9.3.2, 12.7	Y
Get best version (RMNVersionInquire)	9.3.3, 12.8	Y
Store recipe (RMNStore)	9.4.3, 12.11	Y
Download *execution* recipe (RMNAction)	6.3.2, 9.4.8, 12.15	Y
Retrieve recipe (RMNRetrieve)	9.4.4, 12.12	Y
Recipe Namespace Change Event	9.6	Y
Action complete notification (RMNComplete)	12.18	Y
Upload *execution* recipe (RMNAction)	9.4.9, 12.15	N
Unlink recipe (RMNAction)	8.2.3.3, 12.15	N
Approve recipe (RMNAction)	8.2.4.1, 12.15	N
Protect recipe (RMNAction)	8.2.5, 12.15	N
Unprotect recipe (RMNAction)	8.2.6, 12.15	N
Certify recipe (RMNAction)	8.2.4.2, 8.2.7, 12.15	N
Decertify recipe (RMNAction)	8.2.8, 12.15	N
Create namespace (RMNCreate)	9.2.1, 12.3	N
Delete namespace (RMNDeleteNS)	9.2.2, 12.4	N
Rename namespace (RMNRename)	9.2.3, 9.4.6, 12.5	N
Copy recipe (RMNCopy)	9.4.5, 12.13	N
Upload recipe (RMNAction)	9.4.9, 12.15	N
Modify recipe variable parameters (RMNVarPar)	8.2.3.4, 12.16	N
Recipe synchronization	9.5	N

15.4.3 *Recipe Namespace Management Compliance Table* — Table 15.5 provides a statement of compliance for recipe namespace management.

Table 15.5 Recipe Namespace Management Compliance Table

RMS Recipe Namespace Management	*Compliance Statement*	
Fundamental RMS Requirements	*Implemented*	*RMS Compliant*
Managed Recipe Object	❒ Yes ❒ No	❒ Yes[3] ❒ No
Recipe Namespace Object	❒ Yes ❒ No	
Recipe Namespace Manager Object Basic Requirements	❒ Yes ❒ No	
OSS Services	❒ Yes ❒ No	
Create Recipe	❒ Yes ❒ No	
Update Recipe	❒ Yes ❒ No	
Verify Recipe	❒ Yes ❒ No	
Link Recipe	❒ Yes ❒ No	
Get Recipe Descriptor	❒ Yes ❒ No	
Get Available Space	❒ Yes ❒ No	
Check Recipe Status	❒ Yes ❒ No	
Get Best Version	❒ Yes ❒ No	
Store Recipe	❒ Yes ❒ No	
Download Recipe	❒ Yes ❒ No	
Retrieve Recipe	❒ Yes ❒ No	
Recipe Namespace Change Event	❒ Yes ❒ No	
Action Complete Notification	❒ Yes ❒ No	
Additional Capabilities	*Implemented*	*RMS Compliant*[4]
Upload Recipe	❒ Yes ❒ No	❒ Yes ❒ No
Unlink Recipe	❒ Yes ❒ No	❒ Yes ❒ No
Approve Recipe	❒ Yes ❒ No	❒ Yes ❒ No
Protect Recipe	❒ Yes ❒ No	❒ Yes ❒ No
Unprotect Recipe	❒ Yes ❒ No	❒ Yes ❒ No
Certify Recipe	❒ Yes ❒ No	❒ Yes ❒ No
Decertify Recipe	❒ Yes ❒ No	❒ Yes ❒ No
Delete Namespace	❒ Yes ❒ No	❒ Yes ❒ No
Rename Namespace	❒ Yes ❒ No	❒ Yes ❒ No
Copy Recipe	❒ Yes ❒ No	❒ Yes ❒ No
Upload Recipe	❒ Yes ❒ No	❒ Yes ❒ No
Modify Recipe Variable Parameters	❒ Yes ❒ No	❒ Yes ❒ No
Recipe Synchronization	❒ Yes ❒ No	❒ Yes ❒ No

[3] Do not mark YES unless all fundamental RMS requirements are implemented.

[4] Additional capabilities may not be marked RMS compliant unless all fundamental RMS requirements are implemented.

15.5 *Compliance for Distributed Recipe Namespace Management* — Compliance for distributed recipe management requires compliance to the following objects:

- *managed recipe*
- *recipe namespace*
- *distributed recipe namespace*
- *recipe namespace manager*
- *distributed recipe namespace manager*

15.5.1 *Distributed Recipe Namespace* — Table 15.6 describes the requirements for compliance for the *distributed recipe namespace*.

Table 15.6 Section References for Distributed Recipe Namespace Capabilities

Capability	*Reference*	*Fundamental*
Recipe namespace fundamental requirements	15.x, Table 15.x	Y
Basic requirements	5.11.3, Table 5.4	Y
OSS services	5.3.1, 10.1, 10.1.1.1–10.1.1.3	Y
Logical recipes	5.3.2	Y

15.5.2 *Distributed Recipe Namespace Manager* — Table 15.7 describes the requirements for compliance to the *distributed recipe namespace manager*.

Table 15.7 Section References for Distributed Recipe Namespace Manager Capabilities

Capability	*Reference*	*Fundamental*
Recipe namespace manager	15.x, Table 15.x	Y
Basic requirements	5.8, 5.8.1, 5.11.4, 10.3	Y
OSS services	10.3.1	
Attachment and detachment	10.3.3 (all)	Y
Attached segment read-only level	5.4.2	Y
Delete *distributed namespace*		Y
Change request management	10.3.4 (all)	Y
Segment change request (RMDSegChange)	10.3.5, 13.4.3	Y
Segment action complete (RMDComplete)	10.3.6?, 13.4.1	Y
Segment action complete (RMDNotify)	10.3.6?, 13.4.2	Y
Get change request (RMDGetChangeRequest)	10.3.8, 13.4.4	Y
Rebuild *distributed recipe namespace* (RMDRebuild)	5.10, 10.3.9, 13.4.5	Y

15.5.3 *Distributed Recipe Namespace Management Compliance Table* — Table 15.8 provides a statement of compliance for *distributed recipe namespace* management.

Table 15.8 Distributed Recipe Namespace Management Compliance Table

RMS Compliance Statement		
Fundamental RMS Requirements	*Implemented*	*RMS Compliant*
Managed Recipe Object	❒ Yes ❒ No	❒ Yes[5] ❒ No
Recipe Namespace Object	❒ Yes ❒ No	
Recipe Namespace Manager Object	❒ Yes ❒ No	
Distributed Recipe Namespace Object	❒ Yes ❒ No	
Distributed Recipe Namespace Manager Basic Requirements	❒ Yes ❒ No	
OSS services	❒ Yes ❒ No	
Attachment and detachment	❒ Yes ❒ No	
Attached segment read-only level	❒ Yes ❒ No	
Delete *distributed namespace*	❒ Yes ❒ No	
Change request management	❒ Yes ❒ No	
Segment change request	❒ Yes ❒ No	
Segment action complete	❒ Yes ❒ No	
Segment action complete	❒ Yes ❒ No	
Get change request	❒ Yes ❒ No	
Rebuild *distributed recipe namespace*	❒ Yes ❒ No	

[5] Do not mark YES unless all fundamental RMS requirements are implemented.

15.6 *Distributed Recipe Namespace Segment* — Table 15.9 describes the requirements for compliance to the distributed recipe namespace segment.

Table 15.9 Section References for Distributed Recipe Namespace Segment Capabilities

Capability	*Reference*	*Fundamental*
Fundamental compliance to *managed recipe*	15.2	
Basic requirements	5.4, 5.11.1 (Table 5.1), 10.1.2	Y
OSS services	5.3.1	Y
Change management	5.4.2, 5.9, 10.1.2.1	Y
Segment change approval (RMDSApproveAction)	10.1.2.2, 13.2.1	Y
Master segment	5.4.1, Table 5.2	N

15.6.1 *Distributed Recipe Namespace Segment Compliance Table* — Table 15.10 provides a statement of compliance for the distributed recipe namespace segment.

Table 15.10 Distributed Recipe Namespace Segment Compliance Table

RMS Compliance Statement		
Fundamental RMS Requirements	*Implemented*	*RMS Compliant*
Managed Recipe	❒ Yes ❒ No	❒ Yes[6] ❒ No
Basic Requirements	❒ Yes ❒ No	
OSS Services	❒ Yes ❒ No	
Change Management	❒ Yes ❒ No	
Segment Change Approval	❒ Yes ❒ No	
Additional Capabilities	*Implemented*	*RMS Compliant*[7]
Master Segment	❒ Yes ❒ No	❒ Yes ❒ No

[6] Do not mark YES unless all fundamental RMS requirements are implemented.

[7] Additional capabilities may not be marked RMS compliant unless all fundamental RMS requirements are implemented.

15.7 *Distributed Recipe Namespace Recorder* — Table 15.11 describes the requirements for compliance to the distributed recipe namespace recorder.

Table 15.11 Section References for Distributed Recipe Namespace Recorder Capabilities

Capability	*Reference*	*Fundamental*
Basic requirements	5.5, 5.11.2, Table 5.3	Y
OSS services	5.3.1, 10.2.1.1–10.2.1.3	Y
Add segment record (RMDRAddSegRecord)	10.2.2, 13.3.1	Y
Delete segment record (RMDRDelSegRecord)	10.2.3, 13.3.2	Y
Add change request record (RMDRAddChgRecord)	10.2.4, 13.3.3	Y
Delete change request record (RMDRDelChgRecord)	10.2.5, 13.3.4	Y
Get change request record (RMDRGetChgRecord)	10.2.6, 13.3.5	Y

15.7.1 *Distributed Recipe Namespace Recorder Compliance Table* — Table 15.12 provides a statement of compliance for the *distributed recipe namespace recorder*.

Table 15.12 Distributed Recipe Namespace Recorder Compliance Table

RMS Compliance Statement		
Fundamental RMS Requirements	*Implemented*	*RMS Compliant*
Basic Requirements	❒ Yes ❒ No	❒ Yes[8] ❒ No
OSS Services	❒ Yes ❒ No	
Add Segment Record	❒ Yes ❒ No	
Delete Segment Record	❒ Yes ❒ No	
Add Change Request Record	❒ Yes ❒ No	
Delete Change Request Record	❒ Yes ❒ No	
Get Change Request Record	❒ Yes ❒ No	

[8] Do not mark YES unless all fundamental RMS requirements are implemented.

15.8 *Execution Recipe Compliance* — This section defines RMS compliance for the execution recipe. An *execution recipe* exhibits RMS compliance (1) through its attributes and (2) through its structure when transferred.

Table 15.13 describes the requirements for compliance to the *execution recipe*.

Table 15.13 Section References for Execution Recipe Capabilities

Capability	*Reference*	*Fundamental*
Mandatory and required attributes	Table 6.1	Y
Recipe identifier (namespace, class, name, version)	6.3.3	Y
Preservation of designated downloaded attributes	6.3.5	Y
Change management (ChangedBody) (required if *recipe executor* can change recipes)	6.3.1, Table 6.1	Y
Recipe modification EditedBy	Table 6.1	N
Variable parameters (ExecLinkParam)	Table 6.1	N
Multi-part recipe (LinkList)	Table 6.1	N
Recipe compression (SrcRcpID)	Table 6.1	N
Verification signature (VerificationID)	Table 6.1, 11.2.2.2	N

15.8.1 *Execution Recipe Compliance Table* — Table 15.14 provides a statement of compliance for the *execution recipe*.

Table 15.14 Execution Recipe Compliance Table

RMS Compliance Statement		
Fundamental RMS Requirements	*Implemented*	*RMS Compliant*
Mandatory and required attributes	❒ Yes ❒ No	❒ Yes[9] ❒ No
Recipe identifier (namespace, class, name, version)	❒ Yes ❒ No	
Preservation of designated downloaded attributes	❒ Yes ❒ No	
Change management	❒ Yes ❒ No	
Additional Capabilities	*Implemented*	*RMS Compliant*[10]
Recipe Modification	❒ Yes ❒ No	❒ Yes ❒ No
Variable Parameters	❒ Yes ❒ No	❒ Yes ❒ No
Multi-Part Recipe	❒ Yes ❒ No	❒ Yes ❒ No
Recipe Compression	❒ Yes ❒ No	❒ Yes ❒ No
Verification Signature	❒ Yes ❒ No	❒ Yes ❒ No

[9] Do not mark YES unless all fundamental RMS requirements are implemented.

[10] Additional capabilities may not be marked RMS compliant unless all fundamental RMS requirements are implemented.

15.9 *Recipe Executor* — Table 15.15 describes requirements for compliance for the *recipe executor*. The *recipe executor* exhibits RMS compliance through its attributes, behavior, and services.

Table 15.15 Section References for Recipe Executor Capabilities

Capability	*Reference*	*Fundamental*
Fundamental compliance to *execution recipe*	15.x	Y
OSS services for *execution recipe* attributes	11.1	Y
OSS services for *recipe executor* attributes	11.1	Y
Unique recipe identifier	6.6	Y
Protection of *recipe execution area*	6.5	Y
Change control	6.6, 11.2.10, 11.2.10.5	Y
Recipe download and vertify (RMEDnLdVer)	11.2.1, 14.3	Y
Recipe verify (RMEVerify)	11.2.2, 14.4	Y
Recipe rename (RMERename)	11.2.4, 14.6	Y
Get available storage (RMESpaceInquire)	11.2.5, 14.7	Y
Recipe delete (RMEDelete)	11.2.6, 14.8	Y
Recipe select (RMESelect)	11.2.7, 11.2.7.1, 11.2.7.6, 11.2.7.6.1, 14.9	Y
Recipe de-select	11.2.8, 14.10	Y
Recipe validation	11.2.7.2	Y
Variable parameters	11.2.7.4	Y
Recipe upload (RMEUpload)	11.2.3, 14.5	Y
Get recipe descriptor	11.2.9, 14.11	Y
Action completion notification	14.12	Y
Recipe delegation	11.2.7.3	N
Recipe creation	6.6.1, 11.2.10.2, 11.4	N
Recipe compression	6.6.2, 11.2.2.1, 11.2.10.3, 11.4	N
Recipe modification	6.6.3, 11.2.10.1, 11.4	N
Save last values	6.6.4, 11.2.10.4	N
SEMI E10 Productive state minimum approval (ProdApprove)	6.7, Table 6.2	N
SEMI E10 Productive state minimum certification (ProdCertify)	6.7, Table 6.2	N
Cycle units (RunCycleUnit)	Table 6.2	N
Variable parameters (RecipeSelectParameters)	Table 6.2	N
Change notification (RMEChange)	14.12	N

15.9.1 *Recipe Executor Compliance Table* — Table 15.16 provides a statement of compliance for the *recipe executor*.

Table 15.16 RMS Compliance Statement

RMS Compliance Statement		
Fundamental RMS Requirements	*Implemented*	*RMS Compliant*
RMS Compliant Execution Recipe	❒ Yes ❒ No	❒ Yes[11] ❒ No
Basic Requirements	❒ Yes ❒ No	
OSS Services for Execution Recipe	❒ Yes ❒ No	
OSS Services for Recipe Executor	❒ Yes ❒ No	
Unique Recipe Identifier	❒ Yes ❒ No	
Protection of Recipe Execution Area	❒ Yes ❒ No	
Change Control	❒ Yes ❒ No	
Recipe Download and Verify	❒ Yes ❒ No	
Fundamental RMS Requirements	*Implemented*	*RMS Compliant*
Recipe Verify	❒ Yes ❒ No	
Recipe Rename	❒ Yes ❒ No	
Get Available Storage	❒ Yes ❒ No	
Recipe Delete	❒ Yes ❒ No	
Recipe Select	❒ Yes ❒ No	
Recipe De-Select	❒ Yes ❒ No	
Recipe Validation	❒ Yes ❒ No	
Variable Parameters	❒ Yes ❒ No	
Recipe Upload	❒ Yes ❒ No	
Get Recipe Descriptors	❒ Yes ❒ No	
Action Complete Notification	❒ Yes ❒ No	
Additional Capabilities	*Implemented*	*RMS Compliant*[12]
Recipe Delegation	❒ Yes ❒ No	❒ Yes ❒ No
Recipe Creation	❒ Yes ❒ No	❒ Yes ❒ No
Recipe Compression	❒ Yes ❒ No	❒ Yes ❒ No
Recipe Modification	❒ Yes ❒ No	❒ Yes ❒ No
Save Last Values	❒ Yes ❒ No	❒ Yes ❒ No
SEMI E10 Productive State Minimum Approval	❒ Yes ❒ No	❒ Yes ❒ No
SEMI E10 Productive State Minimum Certification	❒ Yes ❒ No	❒ Yes ❒ No
Cycle Units	❒ Yes ❒ No	❒ Yes ❒ No
Change Notification	❒ Yes ❒ No	❒ Yes ❒ No

[11] Do not mark YES unless all fundamental RMS requirements are implemented.

[12] Additional capabilities may not be marked RMS compliant unless all fundamental RMS requirements are implemented.

16 Glossary of Terms

The list of terms in this section is intended only as a guide. Formal definitions are provided in the main body of the document.

agent — A type of *resource* system in a factory that includes both hardware and software components, at least some of which are also *resources*.

Examples of *agents* include recipe namespace servers, cells, stations, clusters, attached cluster modules, equipment, and smart equipment subsystems. *Agents* are associated with a physical system or a collection of physical systems, including computer platforms.

agent-specific attribute — A recipe attribute that applies only to a specific agent.

agent-specific parameter — A *variable parameter* that has been "tuned" for a specific *agent* by modifying its original initial value and/or restrictions and placing the result in the *agent-specific attribute* AgentSpecific LinkParam.

agent-specific recipe — A recipe stored in the *default namespace* of a specific *agent*.

agent-specific dataset — The recipe component containing the set of agent-specific attributes that have been set for that recipe for a specific *agent*.

approve — The operation that sets a recipe's ApprovalLevel attribute to a non-zero value.

approved — A recipe is *approved* whenever its ApprovalLevel attribute is non-zero.

approval level — (1) Used to reflect different degrees of recipe development and qualification; (2) The value of the attribute ApprovalLevel.

attribute — Information concerning an *object* that may be used either for internal or external purposes.

attribute length — The length of an individual attribute, calculated as the sum of the lengths of the *attribute name* and the *attribute value*, exclusive of formatting information.

attribute length attribute — An attribute that contains the sum of the lengths of the attributes. A *managed recipe, agent-specific dataset,* and *execution recipe* each have their own *attribute length* attribute.

attribute timestamp attribute — The *timestamp* of the last change to any of a recipe's attributes.

authorized user — A user who is able to identify him or herself to the host or equipment as having the level of authority required for a particular activity, such as *certifying* a recipe for that equipment.

body — See *recipe body*.

centralized namespace — A *recipe namespace* that is not distributed.

certification-level — The value of the attribute Certified.

certified — (1) Factory-level approval that a recipe produces the expected results on a particular instance of equipment; (2) A recipe for which the Certified attribute is defined and given a non-zero value for a specific *agent*.

certify — The operation of changing the *certification-level* of a recipe for an installation of equipment.

class — See *recipe class*.

clear — An attribute is *cleared* (reset) by setting its value to its *default value*.

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

copy — Originate a recipe by the internal operation of duplicating a recipe and assigning a new *identifier*.

create — The operation of entering a recipe *body* into a *namespace* and assigning it a new (unused) *identifier*.

de-certify — The operation of *clearing* the *certification level* (Certified attribute) for a specific *agent*.

dedicated namespace — A *recipe namespace* with exactly one *member*.

default namespace — A *dedicated, centralized namespace* that is used by standalone equipment and/or for hardware-specific *(agent-specific)* recipes.

default value — A pre-defined value. All non-mandatory recipe attributes are assigned a *default value* as part of their definition. An attribute set to its *default value* may be omitted when a recipe is transferred.

delegate — An *agent's supervisor* **delegates** a recipe that is *executed* by its component *agent*. The *delegating agent* is responsible for ensuring that the *component* has access to a current copy of the recipe and that the recipe is *selected*, started, and *deselected* at the proper time.

delete — A recipe is *deleted* by removing its *identifier*, preventing further access, and freeing its storage.

derived object form — An *object form* of a recipe that is based upon a recipe in *source form* and that represents a more efficient format for execution purposes. Typically, but not by definition, this is a "tokenized" form and requires less storage.

descriptor — (1) The *length* and *timestamp* attributes of a recipe's attributes or of one of its components; (2) The set of all of the individual *descriptors* of a recipe.

deselect — The operation that **prevents a selected** recipe from subsequent execution without its first being *selected* again.

distributed recipe namespace — A namespace that is capable of using *namespace segments* provided by multiple *agents*.

download — To send a recipe to the *recipe executor*.

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 *create* or modify the contents of the recipe *body*. *Editors* are not defined by RMS and are generally considered as external to a *namespace*.

eol — The ASCII line-feed character ($0A_{16}$) signifying the end of a line of text.

empty — An attribute with a *binary*, list, or text format with no content.

end of line character — See *eol*.

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

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

exception — An irregular event that indicates some abnormal or unacceptable condition that requires attention but does not constitute a safety hazard. Within RMS, *exceptions* indicate the abnormal completion of an *operation*.

execute — An *agent* **executes** a recipe by reading the recipe contents and implementing its instructions, process parameters, or other information required for its own processing.

executing agent — The *agent* that is capable of *executing* a specific recipe.

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

executable copy — A selected recipe, or "copy" of a selected recipe, which is in the *execution area*.

execution recipe — The subtype of recipe that is stored by the *recipe executor*.

execution recipe descriptor — Consists of the *execution attribute description*, the *generic attribute description*, and the *body descriptor*, in that order.

execution recipe storage — Storage for recipes provided by the *recipe executor* for recipes that are not currently *selected*.

external reference — (1) The reference which a recipe makes to another recipe; (2) The list of *subrecipes* referenced by a *parent* recipe.

forced overwrite — The replacement of an existing *execution recipe* with a *downloaded* recipe with the same *identifier*.

fundamental requirements — The requirements for information and behavior that must be satisfied for compliance to the Recipe Management Standard. *Fundamental requirements* apply to specific areas of application or objects.

generic attribute — A recipe *attribute* that applies to all *agents* capable of *executing* a specific recipe.

get available storage — The operation that calculates the amount of remaining recipe storage capacity, in bytes, exceeding overhead requirements.

header length — The sum of the lengths of the *attributes* the *header* contains at the time the calculation is performed.

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

identification attribute — An attribute that is used to identify a specific object and that is not included in a recipe section when a recipe is transferred.

identification code — A text string containing a unique and persistent "signature" for the *recipe executor* that last performed verification on a recipe.

identifier — One or more attributes of an object that uniquely identify it within a specific context.

last value — The setting for a *variable parameter* specified by the *user* when the recipe was last *selected* for execution.

link — The operation performed on a main recipe that collects and resolves external references, and collects variable parameter definitions, for that recipe and all referenced recipes.

linked recipe set — The set of recipes identified by the *link* operation and specified by the attribute LinkList.

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.

main recipe — (1) The recipe, within a set of one or more recipes to be *linked* together, which is not a *subrecipe*; (2) The starting recipe or "entry point" for a set of recipes.

managed recipe — A recipe that is stored with a *recipe namespace*.

mandatory attribute — A required attribute that always exists and has a *non-default value*.

modify variable parameters — An operation that modifies the *variable parameter definitions* applied to a specific agent.

name — A text-based attribute of an object that may be used as all or part of an identifier. Certain restrictions on its characters exist. (See SEMI E39.)

namespace — A domain within which object identifiers are unique. Also see *recipe namespace*.

namespace specifier — An object specifier applied to a *recipe namespace*.

non-numeric parameter — Any *variable parameter* that is not a *numeric parameter*.

non-shared namespace — A *recipe namespace* that has at most one *member*.

notification service — A *service* that does not expect a response.

numeric parameter — Any *variable parameters* that can take on any *numeric value* for their *format type*, between a *parameter low limit* and a *parameter high limit*. These *limits* shall not exceed absolute minimum and maximum limits set for that *parameter* by the *agent's* supplier.

numeric version — See *version number*.

object form — (1) Any recipe not in text form; (2) The *body* of a recipe which has been compressed from the *source form*. (See also the generic attribute BodyFormat.)

object specifier — A logical pointer to a remote object. See SEMI E39 (OSS).

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

originate — Any operation which produces a new recipe *identifier*, including *copying, renaming, creating, editing* a *read-only* recipe, and *downloading* a new version.

originating namespace — The *recipe namespace* from which an *execution recipe* was originally *downloaded* and/or to which it is to be *uploaded*.

parameter — A control value that affects the *agent's* process. Also see *variable parameter*.

parameter definition — A formal declaration of a *variable parameter* that specifies the *parameter's name*, *initial value*, and *restrictions*.

parameter domain — The set of all possible values for a given *form* that fulfill the conditions of the *parameter restriction* (if any).

parameter restriction — A required part of a *parameter definition* specifying one or more conditions that any *value* assigned to that *parameter* is required to satisfy to be valid.

parent recipe — Any recipe with *external references* to *subrecipes*.

primary class — A recipe *class* which is not a *subclass* of another *class*.

PROCESS class — The required *primary class* for all recipes used for the *agent's* normal process.

protect — The operation of changing the *approval level* of an individual recipe to that of the RecipeRead-OnlyLevel attribute of the *recipe namespace*.

read-only recipe — A recipe that is *protected*. A recipe with an *approval-level* greater than, or equal to, the value of the *namespace* attribute RecipeRead-OnlyLevel cannot be changed by *updating, deleting, renaming, unlinking,* or *re-linking*.

recipe — (1) The pre-planned and reusable portion of the set of instructions, *settings,* and parameters under control of the *agent* that determine the processing environment seen by the manufactured object and that may be subject to change between runs or processing cycles; (2) The aggregation composed of a *generic header*, zero or more *agent-specific headers*, and a recipe body.

recipe attribute — Structured information concerning a recipe within the recipe *header*.

recipe body — The contents of the recipe containing the data used for execution purposes.

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

managed recipe descriptor — Consists of the *generic attribute descriptor* and the *body descriptor*, followed by *descriptors* for any *agent-specific* datasets.

recipe execution area — Storage used for currently *selected* recipes (i.e., for the current process cycle).

recipe executor — The component of an *executing agent* responsible for understanding and *verifying* the syntax and semantics of a recipe, for *validating* it, and for *executing* it.

recipe identifier — Consists of the recipe's *class, name,* and *version*.

recipe name — A user-defined text string used in the recipe *identifier*. The *name* corresponds to PPID in Stream 7 implementations.

recipe namespace — A logical management domain for (1) the storage and management of recipes, and (2) the insurance of the uniqueness of recipe *identifiers* within that domain.

recipe namespace manager (manager) — The component of an *agent* that manages the *recipe namespace* and that represents the interface for the *namespace* to the external world.

recipe namespace segment (segment) — The component of a *namespace* that represents the internal storage and actual manipulation of recipes.

recipe section — A partition of the recipe for purposes of transferring it through an RMS service. All recipes have a section containing the *generic* attributes and a section containing the body. Some recipes also have a section containing the *agent-specific* attributes.

recipe specifier — An object specifier applied to a recipe.

recipe storage area — A storage area for recipes.

rename — The operation of assigning a new *identifier* to a recipe or to a *recipe namespace*.

request service — A *service* that requires a response.

SERVICE class — An optional *primary class* used for recipes whose purpose is to maintain, prepare, calibrate, or test the operation of *equipment*.

reset — See *clear*.

resource — An owned entity that has an active role in factory operations.

select — The act of preparing a recipe, or a linked recipe set, for execution. This includes confirmation that all specified *subrecipes* are available, that the attributes of the *main* recipe and all its referenced *recipes* meet *execution* requirements, that the recipes are *validated* as a set, and that all other steps that may be necessary for the proper *execution* of the recipe have been performed.

service — A **service** (or **message service**) represents a function offered to a user by a provider. A *service* is one of two types: a *request* or a *notification*.

service resource — A set of services within a particular area of specialization, such as a *recipe namespace resource* or a *recipe execution resource*.

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

shared namespace — A *namespace* with more than one *member*.

source form — A recipe *body* which consists of one or more lines of text and which conforms to a formally defined recipe language.

storage area — An area where objects and data are stored. See also *recipe storage area*.

stored recipe attribute — An attribute of a recipe, a *recipe header*, or a *recipe body*, that is stored as a name/value pair within a recipe *header* whenever the recipe is transferred.

subclass — A *class* of recipes within a larger *class*.

subrecipe — Any recipe which is referenced by another recipe within the same *namespace*.

supervisory agent — An *agent* with supervisory responsibilities for one or more subordinate *agents*. Examples include a *host* manufacturing system and a cluster controller.

timestamp — (1) The notation of the date and time of the occurrence of an event; (2) An attribute of a *recipe header* or *recipe body* that contains the date and time the particular section was last changed. This attribute is a text string of the form “yyyymmddhhmmsscc” for the year yyyy, the month mm, the day dd, the hour hh, the minutes mm, and the seconds ss.

unlink — An operation that *clears* all attributes set by a *link* operation.

unprotect — An operation that resets a recipe's ApprovalLevel attribute. (Set it to zero.)

update — The operation of replacing the *body* of an existing recipe.

upload — To transfer a recipe body from the *recipe executor*.

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

validate — The action of checking a recipe to ensure that the recipe and its parameters' type and range are valid for the *agent's* configuration at the time the recipe is *selected* for execution. [A recipe may be syntactically correct and yet may contain statements that cannot be executed under all configurations of the *agent*.]

variable parameter — A formally defined variable *(setting)* defined in the body of a recipe (1) With a specified default value, boundaries or conditions for replacement values, and (where applicable) units, and

which is placed in the Parameters attribute of the recipe when it is *verified*; (2) Permitting the actual value to be supplied externally by a parent recipe, or at the time the recipe is selected for execution by the *supervisor* or the *operator*.

verification area — Optional temporary storage provided for unverified downloaded recipes.

verify — The operation of (1) ensuring that a recipe is syntactically correct, and (2) identifying *variable parameters* and *external references*.

version — A text string that is part of a recipe's *identifier*.

version number — A version consisting only of the digits "0" through "9" and one decimal point character "." that can be translated to a pure number. Only whole numbers (with no decimal point) and numbers with a decimal point that do not begin or end with a "0" may be used.

whitespace — In ASCII, the space (20_{16}), tab (9_{16}), carriage return ($0D_{16}$), and form feed ($0C_{16}$) characters.

write-protect — See *protect*.

RELATED INFORMATION 1

NOTE: This related information is not an official part of SEMI E42 and is not intended to modify or supersede the official standard. Rather, these notes are auxiliary information provided as background or examples of possible application and are included as reference material. The standard should be referred to in all cases. SEMI makes no warranties or representations as to the suitability of the material set forth herein for any particular application. The determination of the suitability of the material is solely the responsibility of the user.

R1-1 RMS Standardized Objects

The *standardized objects* defined by RMS are shown in Table R1-1.

Table R1-1 Standardized Objects

Object Type	*ObjType*	*Section Reference*
Managed Recipe	MRcp	3.4.2.1
Agent-Specific Dataset	MRcpASDS	3.4.2.2
Execution Recipe	ERcp	3.5.4
Recipe Namespace	RNS	4.6
Recipe Namespace Manager	RNS_Mgr	4.6
Recipe Executor	RcpExec	6.8
Agent	Agent	7.3

R1-2 RMS Requirements/Concepts Map

Table R1-2 provides the specific RMS concepts that address each of the requirements from Table 2.1.

Table R1-2 RMS Requirements/Concepts Map

Requirement	*Concept*
Uniquely store, identify, and select recipes in a system.	Namespace
Share recipes among different agents.	
Synchronize the change of shared recipes among agents.	
Share recipes among different installations of the same type of *equipment* by adjusting individual differences.	
Ability to execute recipes with or without a communication link.	Default Namespace
Ability to change recipes managed by another agent when the communication link has failed or before it is established.	
Formal differentiation of, and recognition of, recipes of different types.	Recipe Class
Allow a recipe name to retain its base identity across a series of modifications.	Version Number
Allow shared recipes to be adjusted for individual pieces of *equipment*.	Variable Parameters
Allow a recipe's parameters to be adjusted within specified limits without requiring the recipe body to be changed.	
Support feedback/feed forward control.	
Ensure the recipe is syntactically correct.	Verification
Manage the approval level of specific recipes with respect to process development and production-worthiness.	Approval
Allow management of equipment qualified for specific recipes.	Certification
Change the recipe while it is running.	Protection of execution area from inadvertent change

R1-3 Background

The material in this section provides a context for the Recipe Management Standard and is intended for background information only.

In a complex environment, *equipment* and *host* become relative terms indicating roles of two independent *agents* with respect to each other. In general, both *equipment* and *host* may provide services to one another, and some services may be provided by both. In certain cases, one may provide services to its partner that the other does not. With respect to recipes, *equipment* refers to the recipe's executor, whereas *host* represents a *supervisory agent* who may *delegate* recipes which will be executed by the *equipment*.

For clusters, cells, and local-area stations, where a mid-level controller communicates both to lower-level *components* and to a higher level factory system, that controller may alternately take the role of *host* to its *components* and of *equipment* to the factory system.

Where role is significant, the terms *equipment* and *host* are used in this section to indicate the relationship of the two *agents* to one another.

R1-3.1 *Traceability* — A main management objective in both a Development Fab and a Production Fab is traceability of manufacturing conditions to the final device characteristics. This traceability is required in a Development Fab to support the analysis and characterization of the process under development. Certain customers of a Production Fab require this traceability in order to do business. Even without such a customer requirement, traceability is important for process control and postmortem analysis of unexpected device characteristics.

The recipe is the primary specification of manufacturing conditions to the *equipment*; therefore, the management of recipes is critical to this traceability.

R1-3.2 *Recipe Life Cycle* — The lifetime of a recipe in a Fab typically has several phases, as illustrated in Figure R1-1. While being developed, it may contain errors, or it may not produce the desired results. The recipe may be edited several times during its initial development. At some point, the recipe enters the test phase, where it will be executed and its results evaluated. A check for syntax correctness occurs at or before the time the equipment prepares it for execution. Still further changes may be made to fine-tune the process to achieve a particular result. During this process, it is still desirable to be able to protect it from inadvertent change or deletion. Finally, the recipe is ready for production and goes through a series of signoffs which constitutes a formal authorization procedure. In many factories, only recipes which have been approved as production-worthy may be used in production runs.

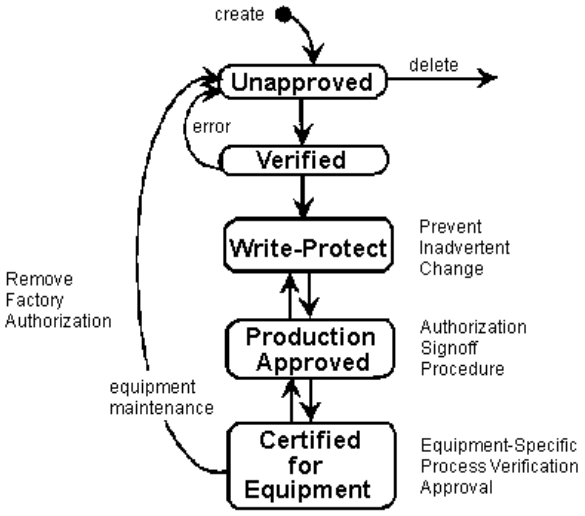

Figure R1-1
Typical Recipe Life Cycle

Also, because results may vary between different installations of *equipment*, an additional level of authorization may be required that certifies individual *equipment* for a given approved recipe.

A factory change control system must accommodate different levels of protection for recipes in different phases of their life cycle. To accomplish this, *equipment* must recognize and support the different requirements for each phase.

R1-3.3 *Recipe Editing* — Traditionally, recipes are developed on the *equipment* that runs them. Besides requiring time on the target machine (a potentially scarce and expensive resource), this requires the person defining the recipe to be inside a clean area. A potential solution is for *equipment* manufacturers to provide versions of their dedicated recipe editors that can be integrated into the factory manufacturing systems. Aside from the difficulty of the *equipment* manufacturer supporting the variety of platforms currently in use internationally, even in a distributed factory environment, a proliferation of different dedicated recipe editors required for the different types of *equipment* is not considered a good solution.

A better solution is to permit editing external to the cleanroom and then to transmit the completed recipe to the *equipment* for verification. This can be achieved by providing a text form of the *equipment* manufacturer's recipe language which can be created and modified with any standard text editor. This may not be possible with some types of recipes (e.g., pattern recognition

programs for vision systems), but this restriction may be removed when the technology advances.

Editing on the *equipment* may still be desirable, particularly in a Development Fab. When the *equipment* permits a recipe to be changed, the *equipment* must provide reporting mechanisms that inform the *host* of changes and allow the *host's* change control system to operate.

R1-3.4 *Recipe Sharing* — A Fab may have several installation of the same type of equipment (equipment with the same functionality and the same recipe language). Therefore, where possible, the same recipe should be able to be shared between different equipment of this same type.

R1-3.5 *Protection and Process Control* — The primary goal of process control is to ensure that the final result of the manufacturing process does not change over time. This may be accomplished by both feedback and feedforward methods. These methods modify the settings used in the process. This requires two capabilities that seem to be in conflict:

- The definition of the process settings (the recipe) should not change.
- The process settings should change to counter the drift inherent in real processes.

The first capability is particularly important in a Production Fab. Most Fab policies prohibit processes from changing after they have been approved for production. This is easy to build into a change control system on the *host*. However, if the *equipment* allows local editing, then it must also enforce this policy.

Wholesale changes in a recipe are never necessary for the second purpose. Usually only a few parameter values are modified in small increments to effect control of the process. A well-defined process specifies the parameters that vary and their range of variation. For any given recipe, it must be possible for the *host* to specify different values of such parameters for each piece of *equipment* so that the recipe may be "tuned" for that *equipment* to produce uniform results. In addition, the process control *agent*, on the *equipment* or the *host*, may need the ability to calculate the parameter value changes for each run and to inform the *equipment* of the new values. Mechanisms are provided to support such interactions.

R1-3.6 *Recipe Selection* — Typical operation of *equipment* requires an *operator* or the *host* to initially *select* the recipe to be run. Recipe *selection* may be specified implicitly within the definition of a higher order "process job". In either case, the specification of a recipe is accomplished by referring to its recipe *identifier* (i.e., the *identifier* used by the change control system) and, where needed to resolve ambiguity, its location. Once *selected*, a recipe is initiated by an appropriate start command, which may be explicitly given by the *operator* or *host* or implicitly given by the *equipment* when executing a higher order process job. Repeated runs of the same recipe may be accomplished via subsequent start commands without the necessity of formal recipe re-selection.

R1-4 Example of a Factory Implementation of Approval Levels

A factory might implement different levels of approval by regulating the following levels of recipe *approval* on the equipment:

```
   Approval Level                          FACTORY POLICY

   Level 0                                 "Unapproved"
   Level 1                                 "In Test"
   Level 2                                 "Non-Production"
   Level 3                                 "Ready for Release"
   Level 4                                 "Released for Production"
with
   RecipeReadOnlyLevel = 2                 Level 2 = read-only
```

R1-5 Examples of Variable Parameters

The recipe *body* may have statements that allow parameters to be changed within fixed limits. For example, recipe BAKE;2.3 may contain statements such as:

```
   Parameter BakeTime of TIME              default = 200 seconds
```

```
                                                    minimum = 100 seconds
                                                    maximum = 300 seconds;
  or
    TIME BakeTime • (200,100,300,"s")                    /* default,min,max,units */
```

that define the process parameter "BakeTime", its *initial value* and *restrictions*. The BAKE;2.3 recipe may use the BakeTime *parameter* in further internal statements to determine the length of time a wafer is processed, such as:

```
wait for TIME BakeTime;
```

This parameter may then be set by a *parent* recipe as in Example (1) below or from the EqpSpec_PARAM *attribute* (see 5.3.1.4) or with a "Select Recipe" command from the *host* or *operator*.

For example, a recipe "BAKE;2.3" may *define* a *variable* "BakeTime", while a recipe "RAMP;1,0" may *define* a *variable* "RampSetPoint". A *parent* recipe "XYZ" might contain *external references* in the text of the *source form* such as:

```
(1)    RUN RAMP;1                                   // use default for RampSetPoint
       RUN BAKE;2.3 with BakeTime=10                // bake for 10 seconds

or

(2)    set RampSetPoint to 500;                     /* ramp up to 500 degC */
       do RAMP(RampSetPoint);
       set Baketime to 120;                         /* 120 sec (2 min.) bake */
       do BAKE;2.3(BakeTime);

or

(3)    RAMP(RampSetPoint:=500)
       BAKE;2.3(BakeTime:=Time_A)                   /* 1st bake time may vary */
       RAMP(RampSetPoint:=650)
       BAKE;2.3(BakeTime:=Time_B)                   /* 2nd bake time may vary*/
       RAMP(RampSetPoint:=800)
       BAKE;2.3(BakeTime:=Time_B+50)                /* 2nd bake plus 50 sec */
```

where "Time_A" and "Time_B" were both formally defined within XYZ.

In example (1), no *initial* value is specified for RampSetPoint. In this case, when the *main* recipe is selected, the *initial value* specified either in EqpSpec_LinkParam or in Gen_LinkParam (that is the same as the *definition* in RAMP;2.3) is used. However, in this example, the *parameter* BakeTime is assigned a value by the *parent* recipe, and this supersedes any *value* from a recipe *attribute* or a "Select Recipe" command from the *host* or *operator*. In the example given, BakeTime is assigned a value outside of the declared *domain*. This should create an error when the recipe is *verified*.

Examples (2) and (3) illustrate how *subrecipes* RAMP and BAKE may be referenced multiple times by a *parent* recipe, and the *parent* may provide different *values* with each reference (Example 2) or may *define* new *parameters* to use with different *references* (Example 3).

In the above examples, PARAMETERS (set when each individual recipe is *verified*) and Gen_LinkParam (set when XYZ is *linked*) will contain *parameter definitions* as follows:

```
Recipe     PARAMETERS                        n_LinkParam
XYZ;0.8        Time_A                            Time_A
               Time_B                            Time_B
                                                 RampSetPoint
                                                 BakeTime
RAMP;1.0       RampSetPoint                      undefined for unlinked recipe
```